



1000 Examples programming in

Python

by Gábor Szabó

1000 Python Examples

Gábor Szabó

This book is for sale at <http://leanpub.com/python-examples>

This version was published on 2020-05-28



* * * * *

This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

* * * * *

© 2020 Gábor Szabó

Table of Contents

First steps

What is Python?

What is needed to write a program?

The source (code) of Python

Python 2 vs. Python 3

Installation

Installation on Linux

Installation on Apple Mac OSX

Installation on MS Windows

Editors, IDEs

Documentation

Program types

Python on the command line

First script - hello world

Examples

Comments

Variables

Exercise: Hello world

What is programming?

What are the programming languages

A written human language

A programming language

Words and punctuation matter!

Literals, Value Types in Python

Floating point limitation

Value Types in Numpy

Rectangular (numerical operations)

Multiply string

[Add numbers](#)

[Add strings](#)

[Exercise: Calculations](#)

[Solution: Calculations](#)

[Second steps](#)

[Modules](#)

[A main function](#)

[The main function - called](#)

[Indentation](#)

[Conditional main](#)

[Input - Output I/O](#)

[print in Python 2](#)

[print in Python 3](#)

[print in Python 2 as if it was Python 3](#)

[Exception: SyntaxError: Missing parentheses in call](#)

[Prompting for user input in Python 2](#)

[Prompting for user input in Python 3](#)

[Python2 input or raw_input?](#)

[Prompting both Python 2 and Python 3](#)

[Add numbers entered by the user \(oops\)](#)

[Add numbers entered by the user \(fixed\)](#)

[How can I check if a string can be converted to a number?](#)

[Converting string to int](#)

[Converting float to int](#)

[Conditionals: if](#)

[Conditionals: if - else](#)

[Conditionals: if - else \(other example\)](#)

[Conditionals: else if](#)

[Conditionals: elif](#)

[Ternary operator](#)

[Case or Switch in Python](#)

[Exercise: Rectangular](#)

[Exercise: Calculator](#)

[Exercise: Standard Input](#)

[Solution: Area of rectangular](#)

[Solution: Calculator](#)

[Command line arguments](#)

[Command line arguments - len](#)

[Command line arguments - exit](#)

[Exercise: Rectangular \(argv\)](#)

[Exercise: Calculator \(argv\)](#)

[Solution: Area of rectangular \(argv\)](#)

[Solution: Calculator eval](#)

[Solution: Calculator \(argv\)](#)

[Compilation vs. Interpretation](#)

[Is Python compiled or interpreted?](#)

[Flake8 checking](#)

[Numbers](#)

[Numbers](#)

[Operators for Numbers](#)

[Integer division and the future](#)

[Pseudo Random Number](#)

[Fixed random numbers](#)

[Rolling dice - randrange](#)

[Random choice](#)

[built-in method](#)

[Exception: TypeError: 'module' object is not callable](#)

[Fixing the previous code](#)

[Exception: AttributeError: module 'random' has no attribute](#)

[Exercise: Number guessing.game - level 0](#)

[Exercise: Fruit salad](#)

[Solution: Number guessing.game - level 0](#)

Solution: Fruit salad

Boolean

if statement again

True and False

Boolean

True and False values in Python

Comparision operators

Do NOT Compare different types

Boolean operators

Boolean truth tables

Short circuit

Short circuit fixed

Incorrect use of conditions

Exercise: compare numbers

Exercise: compare strings

Solution: compare numbers

Solution: compare strings

Strings

Single quoted and double quoted strings

Long lines

Triple quoted strings (multiline)

String length (len)

String repetition and concatenation

A character in a string

String slice (instead of substr)

Change a string

How to change a string

String copy

String functions and methods (len, upper, lower)

index in string

[index in string with range](#)
[rindex in string with range](#)
[find in string](#)
[Find all in the string](#)
[in string](#)
[index if in string](#)
[Encodings: ASCII, Windows-1255, Unicode](#)
[raw strings](#)
[ord](#)
[ord in a file](#)
[chr - number to character](#)
[Exercise: one string in another string](#)
[Exercise: to ASCII CLI](#)
[Exercise: from ASCII CLI](#)
[Solution: one string in another string](#)
[Solution: compare strings](#)
[Solution: to ASCII CLI](#)
[Solution: from ASCII CLI](#)

Loops

[Loops: for-in and while](#)
[for-in loop on strings](#)
[for-in loop on list](#)
[for-in loop on range](#)
[Iterable, iterator](#)
[for in loop with early end using break](#)
[for in loop skipping parts using continue](#)
[for in loop with break and continue](#)
[while loop](#)
[Infinite while loop](#)
[While with complex expression](#)
[While with break](#)

[While True](#)
[Duplicate input call](#)
[Eliminate duplicate input call](#)
[do while loop](#)
[while with many continue calls](#)
[Break out from multi-level loops](#)
[Exit vs return vs break and continue](#)
[Exercise: Print all the locations in a string](#)
[Exercise: Number guessing game](#)
[Exercise: MasterMind](#)
[Exercise: Count unique characters](#)
[Solution: Print all the locations in a string](#)
[Solution 1 for Number Guessing](#)
[Solution for Number Guessing \(debug\)](#)
[Solution for Number Guessing \(move\)](#)
[Solution for Number Guessing \(multi-game\)](#)
[Solution: MasterMind](#)
[Solution: Count unique characters](#)
[MasterMind to debug](#)

[PyCharm](#)
[PyCharm Intro](#)
[PyCharm Project](#)
[PyCharm Files](#)
[PyCharm - run code](#)
[PyCharm Python console at the bottom left](#)
[Refactoring example \(with and without pycharm\)](#)

[Formatted printing](#)
[format - sprintf](#)
[Examples using format - indexing](#)
[Examples using format with names](#)

[Format columns](#)

[Examples using format - alignment](#)

[Format - string](#)

[Format characters and types](#)

[Format floating point number](#)

[f-strings \(formatted string literals\)](#)

[printf using old %-syntax](#)

[Format braces, bracket, and parentheses](#)

[Examples using format with attributes of objects](#)

[raw f-strings](#)

[Lists](#)

[Anything can be a lists](#)

[Any layout](#)

[Lists](#)

[List slice with steps](#)

[Change a List](#)

[Change with steps](#)

[List assignment and list copy](#)

[join](#)

[join list of numbers](#)

[split](#)

[for loop on lists](#)

[in list](#)

[Where is the element in the list](#)

[Index improved](#)

[\[\] .insert](#)

[\[\] .append](#)

[\[\] .remove](#)

[Remove element by index \[\] .pop](#)

[Remove first element of list](#)

[Remove several elements of list by index](#)

[Use list as a queue](#)

[Queue using deque from collections](#)

[Fixed size queue](#)

[List as a stack](#)

[stack with deque](#)

[Exercises: Queue](#)

[Exercise: Stack](#)

[Solution: Queue with list](#)

[Solution: Queue with deque](#)

[Solution: Reverse Polish calculator \(stack\) with lists](#)

[Solution: Reverse Polish calculator \(stack\) with deque](#)

[Debugging Queue](#)

[sort](#)

[sort numbers](#)

[sort mixed](#)

[key sort](#)

[Sort tuples](#)

[sort with sorted](#)

[sort vs. sorted](#)

[key sort with sorted](#)

[Sorting characters of a string](#)

[range](#)

[Looping over index](#)

[Enumerate lists](#)

[List operators](#)

[List of lists](#)

[List assignment](#)

[List documentation](#)

[tuple](#)

[Exercise: color selector menu](#)

[Exercise: count digits](#)

[Exercise: Create list](#)

[Exercise: Count words](#)

[Exercise: Check if number is prime](#)

[Exercise: DNA sequencing](#)

[Solution: menu](#)

[Solution: count digits](#)

[Solution: Create list](#)

[Solution: Count words](#)

[Solution: Check if number is prime](#)

[Solution: DNA sequencing](#)

[Solution: DNA sequencing with filter](#)

[Solution: DNA sequencing with filter and lambda](#)

[\[\] .extend](#)

[append vs. extend](#)

[split and extend](#)

[Files](#)

[Open and read file](#)

[Filename on the command line](#)

[Filehandle with and without](#)

[Filehandle with return](#)

[Read file remove newlines](#)

[Read all the lines into a list](#)

[Read all the characters into a string \(slurp\)](#)

[Not existing file](#)

[Open file exception handling](#)

[Open many files - exception handling](#)

[Writing to file](#)

[Append to file](#)

[Binary mode](#)

[Does file exist? Is it a file?](#)

[Exercise: count numbers](#)

[Exercise: strip newlines](#)

[Exercise: color selector](#)
[Exercise: ROT13](#)
[Exercise: Combine lists](#)
[Solution: count numbers](#)
[Solution: strip newlines](#)
[Solution: color selector](#)
[Solution: Combine lists](#)
[Read text file](#)
[Open and read file](#)
[Direct access of a line in a file](#)
[Example](#)

[Dictionary \(hash\)](#)

[What is a dictionary](#)
[When to use dictionaries](#)
[Dictionary](#)
[keys](#)
[Loop over keys](#)
[Loop using items](#)
[values](#)
[Not existing key](#)
[Get key](#)
[Does the key exist?](#)
[Does the value exist?](#)
[Delete key](#)
[List of dictionaries](#)
[Shared dictionary](#)
[immutable collection: tuple as dictionary key](#)
[immutable numbers: numbers as dictionary key](#)
[Sort dictionary by value](#)
[Sort dictionary keys by value](#)
[Insertion Order is kept](#)

[Change order of keys in dictionary - OrderedDict](#)
[Set order of keys in dictionary - OrderedDict](#)
[Exercise: count characters](#)
[Exercise: count words](#)
[Exercise: count words from a file](#)
[Exercise: Apache log](#)
[Exercise: Combine lists again](#)
[Exercise: counting DNA bases](#)
[Exercise: Count Amino Acids](#)
[Exercise: List of dictionaries](#)
[Exercise: Dictinoary of dictionaries](#)
[Solution: count characters](#)
[Solution: count characters with default dict](#)
[Solution: count words](#)
[Solution: count words in file](#)
[Solution: Apache log](#)
[Solution: Combine lists again](#)
[Solution: counting DNA bases](#)
[Solution: Count Amino Acids](#)
[Loop over dictionary keys](#)
[Do not change dictionary in loop](#)
[Default Dict](#)

Sets

[sets](#)
[set operations](#)
[set intersection](#)
[set subset](#)
[set symmetric difference](#)
[set union](#)
[set relative complement](#)
[set examples](#)

defining an empty set

Adding an element to a set (add)

Merging one set into another set (update)

Functions (subroutines)

Defining simple function

Defining a function

Parameters can be named

Mixing positional and named parameters

Default values

Several defaults, using names

Arbitrary number of arguments *

Fixed parameters before the others

Arbitrary key-value pairs in parameters **

Extra key-value pairs in parameters

Every parameter option

Duplicate declaration of functions (multiple signatures)

Recursive factorial

Recursive Fibonacci

Non-recursive Fibonacci

Unbound recursion

Variable assignment and change - Immutable

Variable assignment and change - Mutable

Parameter passing of functions

Passing references

Function documentation

Sum ARGV

Copy-paste code

Copy-paste code fixed

Copy-paste code further improvement

Palindrome

Exercise: statistics

[Exercise: recursive](#)

[Exercise: Tower of Hanoi](#)

[Exercise: Merge and Bubble sort](#)

[Solution: statistics](#)

[Solution: recursive](#)

[Solution: Tower of Hanoi](#)

[Solution: Merge and Bubble sort](#)

[Modules](#)

[Before modules](#)

[Create modules](#)

[path to load modules from - The module search path](#)

[sys.path - the module search path](#)

[Flat project directory structure](#)

[Absolute path](#)

[Relative path](#)

[Python modules are compiled](#)

[How “import” and “from” work?](#)

[Runtime loading of modules](#)

[Conditional loading of modules](#)

[Duplicate importing of functions](#)

[Script or library](#)

[Script or library - import](#)

[Script or library - from import](#)

[assert to verify values](#)

[mycalc as a self testing module](#)

[doctest](#)

[Scope of import](#)

[Export import](#)

[Export import with all](#)

[import module](#)

[Execute at import time](#)

[Import multiple times](#)

[Exercise: Number guessing](#)

[Exercises: Scripts and modules](#)

[Exercise: Module my_sum](#)

[Exercise: Convert your script to module](#)

[Exercise: Add doctests to your own code](#)

[Solution: Module my_sum](#)

[Regular Expressions](#)

[What are Regular Expressions \(aka. Regexes\)?](#)

[What are Regular Expressions good for?](#)

[Examples](#)

[Where can I use it ?](#)

[grep](#)

[Regexes first match](#)

[Match numbers](#)

[Capture](#)

[Capture more](#)

[Capture even more](#)

[findall](#)

[findall with capture](#)

[findall with capture more than one](#)

[Any Character](#)

[Match dot](#)

[Character classes](#)

[Common character classes](#)

[Negated character class](#)

[Optional character](#)

[Regex 0 or more quantifier](#)

[Quantifiers](#)

[Quantifiers limit](#)

[Quantifiers on character classes](#)

[Greedy quantifiers](#)
[Minimal quantifiers](#)
[Anchors](#)
[Anchors on both end](#)
[Match ISBN numbers](#)
[Matching a section](#)
[Matching a section - minimal](#)
[Matching a section negated character class](#)
[DOTALL S \(single line\)](#)
[MULTILINE M](#)
[Two regex with logical or](#)
[Alternatives](#)
[Grouping and Alternatives](#)
[Internal variables](#)
[More internal variables](#)
[Regex DNA](#)
[Regex IGNORECASE](#)
[Regex VERBOSE X](#)
[Substitution](#)
[findall capture](#)
[Fixing dates](#)
[Duplicate numbers](#)
[Remove spaces](#)
[Replace string in assembly code](#)
[Full example of previous](#)
[Split with regex](#)
[Exercises: Regexes part 1](#)
[Exercise: Regexes part 2](#)
[Exercise: Sort SNMP numbers](#)
[Exercise: parse hours log file and give report](#)
[Exercise: Parse ini file](#)
[Exercise: Replace Python](#)

[Exercise: Extract phone numbers](#)

[Solution: Sort SNMP numbers](#)

[Solution: parse hours log file and give report](#)

[Solution: Processing INI file manually](#)

[Solution: Processing config file](#)

[Solution: Extract phone numbers](#)

[Regular Expressions Cheat sheet](#)

[Fix bad JSON](#)

[Fix very bad JSON](#)

[Raw string or escape](#)

[Remove spaces regex](#)

[Regex Unicode](#)

[Anchors Other example](#)

[Python standard modules](#)

[Some Standard modules](#)

[sys](#)

[Writing to standard error \(stderr\)](#)

[Current directory \(getcwd, pwd, chdir\)](#)

[OS dir \(mkdir, makedirs, remove, rmdir\)](#)

[python which OS are we running on \(os, platform\)](#)

[Get process ID](#)

[OS path](#)

[Traverse directory tree - list directories recursively](#)

[os.path.join](#)

[Directory listing](#)

[expanduser - handle tilde ~](#)

[Listing specific files using .glob](#)

[External command with system](#)

[subprocess](#)

[subprocess in the background](#)

[Accessing the system environment variables from Python](#)

[Set env and run command](#)
[shutil](#)
[time](#)
[sleep in Python](#)
[timer](#)
[Current date and time datetime now](#)
[Converting string to datetime](#)
[datetime arithmetic is](#)
[Rounding datetime object to nearest second](#)
[Signals and Python](#)
[Sending Signal](#)
[Catching Signal](#)
[Catching Ctrl-C on Unix](#)
[Catching Ctrl-C on Unix confirm](#)
[Alarm signal and timeouts](#)
[deep copy list](#)
[deep copy dictionary](#)
[Exercise: Catching Ctrl-C on Unix 2nd time](#)
[Exercise: Signals](#)
[Ctrl-z](#)

JSON

[JSON - JavaScript Object Notation](#)
[dumps](#)
[loads](#)
[dump](#)
[load](#)
[Round trip](#)
[Pretty print JSON](#)
[Sort keys in JSON](#)
[Set order of keys in JSON - OrderedDict](#)
[Exercise: Counter in JSON](#)

[Exercise: Phone book](#)

[Exercise: Processes](#)

[Solution: Counter in JSON](#)

[Solution: Phone book](#)

[Command line arguments with argparse](#)

[Modules to handle the command line](#)

[argparse](#)

[Basic usage of argparse](#)

[Positional argument](#)

[Many positional argument](#)

[Convert to integers](#)

[Convert to integer](#)

[Named arguments](#)

[Boolean Flags](#)

[Short names](#)

[Exercise: Command line parameters](#)

[Exercise: argparse positional and named](#)

[Exception handling](#)

[Hierarchy of calls](#)

[Handling errors as return values](#)

[Handling errors as exceptions](#)

[A simple exception](#)

[Working on a list](#)

[Catch ZeroDivisionError exception](#)

[Module to open files and calculate something](#)

[File for exception handling example](#)

[Open files - exception](#)

[Handle divide by zero exception](#)

[Handle files - exception](#)

[Catch all the exceptions and show their type](#)

List exception types

Exceptions

How to raise an exception

Stack trace

Exercises: Exception int conversion

Exercises: Raise Exception

Solution: Exception int conversion (specific)

Solution: Exception int conversion (all other)

Solution: Raise Exception

Classes - OOP - Object Oriented Programming

Why Object Oriented Programming?

Generic Object Oriented Programming terms

OOP in Python

OOP in Python (numbers, strings, lists)

OOP in Python (argparse)

Create a class

Import module containing class

Import class from module

Initialize a class - constructor, attributes

Attributes are not special

Create Point class

Initialize a class - constructor, attributes

Methods

Stringify class

Inheritance

Inheritance - another level

Modes of method inheritance

Modes of method inheritance - implicit

Modes of method inheritance - override

Modes of method inheritance - extend

Modes of method inheritance - delegate - provide

[Composition - Line](#)
[Some comments](#)
[Class in function](#)
[Serialization of instances with pickle](#)
[Quick Class definition and usage](#)
[Exercise: Add move rad to based on radians](#)
[Exercise: Improve previous examples](#)
[Exercise: Polygon](#)
[Exercise: Number](#)
[Exercise: Library](#)
[Exercise: Bookexchange](#)
[Exercise: Represent turtle graphics](#)
[Solution - Polygon](#)

[PyPi - Python Package Index](#)

[What is PyPi?](#)
[Easy Install](#)
[pip](#)
[Upgrade pip](#)
[PYTHONPATH](#)
[Virtualenv](#)
[Virtualenv for Python 3](#)

[SQLite Database Access](#)

[SQLite](#)
[Connecting to SQLite database](#)
[Create TABLE in SQLite](#)
[INSERT data into SQLite database](#)
[SELECT data from SQLite database](#)
[A counter](#)

[MySQL](#)

[Install MySQL support](#)
[Create database user \(manually\).](#)
[Create database \(manually\).](#)
[Create table \(manually\).](#)
[Connect to MySQL](#)
[Connect to MySQL and Handle exception](#)
[Select data](#)
[Select more data](#)
[Select all data fetchall](#)
[Select some data fetchmany](#)
[Select some data WHERE clause](#)
[Select into dictionaries](#)
[Insert data](#)
[Update data](#)
[Delete data](#)
[Exercise MySQL](#)
[Exercise: MySQL Connection](#)
[Solution: MySQL Connection](#)

[PostgreSQL](#)
[PostgreSQL install](#)
[Python and Postgresql](#)
[PostgreSQL connect](#)
[INSERT](#)
[INSERT \(from command line\)](#)
[SELECT](#)
[DELETE](#)

[SQLAlchemy](#)
[SQLAlchemy hierarchy](#)
[SQLAlchemy engine](#)
[SQLAlchemy autocommit](#)

[SQLAlchemy engine CREATE TABLE](#)
[SQLAlchemy engine INSERT](#)
[SQLAlchemy engine SELECT](#)
[SQLAlchemy engine SELECT all](#)
[SQLAlchemy engine SELECT fetchall](#)
[SQLAlchemy engine SELECT aggregate](#)
[SQLAlchemy engine SELECT IN](#)
[SQLAlchemy engine SELECT IN with placeholders](#)
[SQLAlchemy engine connection](#)
[SQLAlchemy engine transaction](#)
[SQLAlchemy engine using context managers](#)
[Exercise: Create table](#)
[SQLAlchemy Meta](#)
[SQLAlchemy types](#)
[SQLAlchemy ORM - Object Relational Mapping](#)
[SQLAlchemy ORM create](#)
[SQLAlchemy ORM schema](#)
[SQLAlchemy ORM reflection](#)
[SQLAlchemy ORM INSERT after automap](#)
[SQLAlchemy ORM INSERT](#)
[SQLAlchemy ORM SELECT](#)
[SQLAlchemy ORM SELECT cross tables](#)
[SQLAlchemy ORM SELECT and INSERT](#)
[SQLAlchemy ORM UPDATE](#)
[SQLAlchemy ORM logging](#)
[Solution: Create table](#)
[Exercise: Inspector](#)
[SQLAlchemy CREATE and DROP](#)
[SQLAlchemy Notes](#)
[SQLAlchemy Meta SQLite CREATE](#)
[SQLAlchemy Meta Reflection](#)
[SQLAlchemy Meta INSERT](#)

SQLAlchemy Meta SELECT

NoSQL

Types of NoSQL databases

MongoDB

MongoDB CRUD

Install MongoDB support

Python MongoDB insert

MongoDB CLI

Python MongoDB find

Python MongoDB find refine

Python MongoDB update

Python MongoDB remove (delete)

Redis

Redis CLI

Redis list keys

Redis set get

Redis incr

Redis incrby

Redis setex

Web client

urllib the web client

urllib2 the web client

httpbin.org

requests get

Download image using requests

Download image as a stream using requests

Download zip file

Extract zip file

Interactive Requests

[requests get JSON](#)
[requests get JSON UserAgent](#)
[requests get JSON UserAgent](#)
[requests get header](#)
[requests change header](#)
[requests post](#)
[Tweet](#)
[API config file](#)
[bit.ly](#)
[Exercise: Combine web server and client](#)

Python Web server

[Hello world web](#)
[Dump web environment info](#)
[Web echo](#)
[Web form](#)
[Resources](#)

Python Flask

[Python Flask intro](#)
[Python Flask installation](#)
[Flask: Hello World](#)
[Flask hello world + test](#)
[Flask generated page - time](#)
[Flask: Echo GET](#)
[Flask: Echo POST](#)
[Flask: templates](#)
[Flask: templates](#)
[Flask: templates with parameters](#)
[Flask: runner](#)
[Exercise: Flask calculator](#)
[Static files](#)

[Flask Logging](#)
[Flask: Counter](#)
[Color selector without session](#)
[Session management](#)
[Flask custom 404 page](#)
[Flask Error page](#)
[Flask URL routing](#)
[Flask Path params](#)
[Flask Path params \(int\)](#)
[Flask Path params add \(int\)](#)
[Flask Path params add \(path\)](#)
[Jinja loop, conditional, include](#)
[Exercise: Flask persistent](#)
[Exercise: Flask persistent](#)
[Flask Exercises](#)
[Flask login](#)
[Flask JSON API](#)
[Flask and AJAX](#)
[Flask and AJAX](#)
[passlib](#)
[Flask Testing](#)
[Flask Deploy app](#)
[Flask Simple Authentication + test](#)
[Flask REST API](#)
[Flask REST API - Echo](#)
[Flask REST API - parameters in path](#)
[Flask REST API - parameter parsing](#)
[Flask REST API - parameter parsing - required](#)

[Networking](#)
[Secure shell](#)
[ssh](#)

[ssh from Windows](#)
[Parallel ssh](#)
[telnet](#)
[prompt for password](#)
[Python nmap](#)
[ftp](#)

[Interactive shell](#)

[The Python interactive shell](#)
[REPL - Read Evaluate Print Loop](#)
[Using Modules](#)
[Getting help](#)
[Exercise: Interactive shell](#)

[Testing Demo](#)

[How do you test your code?](#)
[What is testing?](#)
[What is testing really?](#)
[Testing demo - AUT - Application Under Test](#)
[Testing demo - use the module](#)
[Testing demo: doctests](#)
[Testing demo: Unittest success](#)
[Testing demo: Unittest failure](#)
[Testing demo: pytest using classes](#)
[Testing demo: pytest without classes](#)
[Testing demo: pytest run doctests](#)
[Testing demo: pytest run unittest](#)
[Exercise: Testing demo](#)
[Solution: Testing demo](#)

[Types in Python](#)

[mypy](#)

[Types of variables](#)

[Types of function parameters](#)

[Types used properly](#)

[TODO: mypy](#)

[Testing Intro](#)

[The software testing equation](#)

[The software testing equasion \(fixed\)](#)

[The pieces of your software?](#)

[Manual testing](#)

[What to tests?](#)

[Continuous Integration](#)

[Functional programming](#)

[Functional programming](#)

[Iterators \(Iterables\)](#)

[range](#)

[range with list](#)

[range vs. list size](#)

[for loop with transformation](#)

[map](#)

[map delaying function call](#)

[map on many values](#)

[map with list](#)

[double with lambda](#)

[What is lambda in Python?](#)

[lambda returning tuple](#)

[map returning tuples](#)

[lambda with two parameters](#)

[map for more than one iterable](#)

[map on uneven lists](#)

[replace None \(for Python 2\)](#)

[map on uneven lists - fixed \(for Python 2\)](#)

[map mixed iterators](#)

[map fetch value from dict](#)

[Exercise: string to length](#)

[Exercise: row to length](#)

[Exercise: compare rows](#)

[Solution: string to length](#)

[Solution: row to length](#)

[Solution: compare rows](#)

[filter](#)

[filter with lambda](#)

[filter - map example](#)

[filter - map in one expression](#)

[Get indexes of values](#)

[reduce](#)

[reduce with default](#)

[zip](#)

[Creating dictionary from two lists using zip](#)

[all, any](#)

[Compare elements of list with scalar](#)

[List comprehension - double](#)

[List comprehension - simple expression](#)

[List generator](#)

[List comprehension](#)

[Dict comprehension](#)

[Lookup table with lambda](#)

[Read lines without newlines](#)

[Read key-value pairs](#)

[Create index-to-value mapping in a dictionary based on a list of values](#)

[Exercise: min, max, factorial](#)

[Exercise: Prime numbers](#)

[Exercise: Many validator functions](#)

[Exercise: Calculator using lookup table](#)

[Exercise: parse file](#)

[Solution: min, max, factorial](#)

[Solution: Prime numbers](#)

[Solution: Many validator functions](#)

[Solution: Calculator using lookup table](#)

[map with condition](#)

[map with lambda](#)

[map with lambda with condition](#)

[List comprehension - complex](#)

[Iterators - with and without Itertools](#)

[Advantages of iterators and generators](#)

[The Fibonacci research institute](#)

[Fibonacci plain](#)

[Fibonacci copy-paste](#)

[Iterators Glossary](#)

[What are iterators and iterables?](#)

[A file-handle is an iterator](#)

[range is iterable but it is not an iterator](#)

[Iterator: a counter](#)

[Using iterator](#)

[Iterator without temporary variable](#)

[The type of the iterator](#)

[Using iterator with next](#)

[Mixing for and next](#)

[Iterable which is not an iterator](#)

[Iterator returning multiple values](#)

[Range-like iterator](#)

[Unbound or infinite iterator](#)

[Unbound iterator Fibonacci](#)

Operations on Unbound iterator

itertools

itertools - count

itertools - cycle

Exercise: iterators - reimplement the range function

Exercise: iterators - cycle

Exercise: iterators - alter

Exercise: iterators - limit Fibonacci

Exercise: iterators - Fibonacci less memory

Exercise: read char

Exercise: read section

Exercise: collect packets

Exercise: compare files

Solution: iterators - limit Fibonacci

Solution: iterators - Fibonacci less memory

Solution: read section

Solution: compare files

Solution: collect packets

Generators and Generator Expressions

Generators Glossary

Iterators vs Generators

List comprehension and Generator Expression

List comprehension vs Generator Expression - less memory

List comprehension vs Generator Expression - lazy evaluation

Generator: function with yield - call next

Generators - call next

Generator with yield

Generators - fixed counter

Generators - counter

Generators - counter with parameter

[Generators - my_range](#)
[Fibonacci - generator](#)
[Infinite series](#)
[Integers](#)
[Integers + 3](#)
[Integers + Integers](#)
[Filtered Fibonacci](#)
[The series.py](#)
[generator - unbound count \(with yield\)](#).
[iterator - cycle](#)
[Exercise: Alternator](#)
[Exercise: Prime number generator](#)
[Exercise: generator](#)
[Exercise: Tower of Hanoi](#)
[Exercise: Binary file reader](#)
[Exercise: File reader with records](#)

Logging

[Simple logging](#)
[Simple logging - set level](#)
[Simple logging to a file](#)
[Simple logging format](#)
[Simple logging change date format](#)
[getLogger](#)
[Time-based logrotation](#)
[Size-based logrotation](#)

Closures

[Counter local - not working](#)
[Counter with global](#)
[Create incrementors](#)
[Create internal function](#)

[Create function by a function](#)
[Create function with parameters](#)
[Counter closure](#)
[Make incrementor with def \(closure\)](#).
[Make incrementor with lambda](#)
[Exercise: closure bank](#)
[Solution: closure bank](#)
[Solution: counter with parameter](#)

Decorators

[Function assignment](#)
[Function inside other function](#)
[Decorator](#)
[Use cases for decorators in Python](#)
[A recursive Fibonacci](#)
[trace fibo](#)
[tron decorator](#)
[Decorate with direct call](#)
[Decorate with parameter](#)
[Decorator accepting parameter](#)
[Decorate function with any signature](#)
[Decorate function with any signature - implementation](#)
[Exercise: Logger decorator](#)
[Exercise: memoize decorator](#)
[Solution: Logger decorator](#)
[Solution: Logger decorator \(testing\)](#).
[Solution memoize decorator](#)

Context managers (with statement)

[Why use context managers?](#)
[Context Manager examples](#)
[cd in a function](#)

[open in function](#)
[open in for loop](#)
[open in function using with](#)
[Plain context manager](#)
[Param context manager](#)
[Context manager that returns a value](#)
[Use my tempdir - return](#)
[Use my tempdir - exception](#)
[cwd context manager](#)
[tempdir context manager](#)
[Context manager with class](#)
[Context managers with class](#)
[Context manager: with for file](#)
[With - context managers](#)
[Exercise: Context manager](#)
[Exercise: Tempdir on Windows](#)
[Solution: Context manager](#)

Advanced lists

[Change list while looping: endless list](#)
[Change list while looping](#)
[Copy list before iteration](#)
[for with flag](#)
[for else](#)
[enumerate](#)
[do while](#)
[list slice is copy](#)

Advanced Exception handling

[Exceptions else](#)
[Exceptions finally](#)
[Exit and finally](#)

[Catching exceptions](#)
[Home made exception](#)
[Home made exception with attributes](#)
[Home made exception hierarchy](#)
[Home made exception hierarchy - 1](#)
[Home made exception hierarchy - 2](#)
[Home made exception hierarchy - 3](#)
[Exercise: spacefight with exceptions](#)
[Exercises: Raise My Exception](#)
[Solution: spacefight with exceptions](#)
[Solution: Raise My Exception](#)
[Exception finally return](#)

[Warnings](#)

[Warnings](#)

[CSV](#)

[Reading CSV the naive way](#)
[CSV with quotes and newlines](#)
[Reading a CSV file](#)
[CSV dialects](#)
[CSV to dictionary](#)
[Exercise: CSV](#)
[Solution: CSV](#)

[Excel](#)

[Spreadsheets](#)
[Python Excel](#)
[Create an Excel file from scratch](#)
[Worksheets in Excel](#)
[Add expressions to Excel](#)
[Format field](#)

[Number series and chart](#)

[Read Excel file](#)

[Update Excel file](#)

[Exercise: Excel](#)

[XML](#)

[XML Data](#)

[Expat - Callbacks](#)

[XML DOM - Document Object Model](#)

[XML SAX - Simple API for XML](#)

[SAX collect](#)

[XML elementtree](#)

[SciPy - for Scientific Computing in Python](#)

[Data Science tools in Python](#)

[Data Analysis resources](#)

[Python and Biology](#)

[Biopython](#)

[Biopython background](#)

[Bio python sequences](#)

[Download data](#)

[Read FASTA, GenBank files](#)

[Search nucleotids](#)

[Download nucleotids](#)

[Exercise: Nucleotid](#)

[Biology background](#)

[Chemistry](#)

[Chemistry links](#)

[Bond length](#)

[Covalent radius](#)

[Python energy landscape explorer](#)

Other chemistry links

numpy

[What is NumPy](#)

[Numpy - vector](#)

[NumPy 2D arrays](#)

[Numpy - set type](#)

[NumPy arrays: ones and zeros](#)

[Numpy: eye](#)

[NumPy array random](#)

[NumPy Random integers](#)

[NumPy array type change by division \(int to float\)](#)

[Numpy: Array methods: transpose](#)

[Numpy: reference, not copy](#)

[Numpy: copy array](#)

[Numpy: Elementwise Operations on Arrays](#)

[Numpy: multiply, matmul, dot for vectors](#)

[Numpy: multiply, matmul, dot for vector and matrix](#)

[Numpy: multiply, matmul, dot for matrices](#)

[Numpy: casting - converting from strings to integer.](#)

[Numpy: indexing 1d array](#)

[Numpy: slice is a reference](#)

[Numpy: slice - copy](#)

[Numpy: abs value on a Numpy array](#)

[Numpy: Logical not on a Numpy array](#)

[Numpy: Vectorize a function](#)

[Numpy: Vectorize len](#)

[Numpy: Vectorize lambda](#)

[Numpy: Filtering array](#)

[Numpy: Filter matrix values](#)

[Numpy: Filter matrix rows](#)

[Numpy: Stat](#)

[Numpy: Serialization](#)

[Numpy: Load from Matlab file](#)

[Numpy: Save as Matlab file](#)

[Numpy: Horizontal stack vectors \(hstack\)](#)

[Numpy: Append or vertically stack vectors and matrices \(vstack\)](#)

[Numpy uint8](#)

[Numpy int8](#)

[Pandas](#)

[Pandas](#)

[Planets](#)

[Pandas Planets - Dataframes](#)

[Pandas Stocks](#)

[Pandas Stocks](#)

[Merge Dataframes](#)

[Analyze Alerts](#)

[Analyze IFMetrics](#)

[Create Excel file for experiment with random data](#)

[Calculate Genome metrics](#)

[Calculate Genome metrics - add columns](#)

[Calculate Genome metrics - vectorized](#)

[Calculate Genome metrics - vectorized numpy](#)

[Genes using Jupyter](#)

[Combine columns](#)

[Pandas more](#)

[Pandas Series](#)

[Pandas Series with names](#)

[Matplotlib](#)

[About Matplotlib](#)

[Matplotlib Line](#)

[Matplotlib Line with dates](#)
[Matplotlib Simple Pie](#)
[Matplotlib Simple Pie with params](#)
[Matplotlib Pie](#)
[Matplotlib Pie 2](#)
[Plot, scatter, histogram](#)

[Seaborn](#)

[Searborn use examples](#)
[Seaborn tip](#)
[Seaborn Anscombes Quartet](#)

[Jupyter notebooks](#)

[Jupyter on Windows](#)
[Jupyter on Linux and OSX](#)
[Jupyter add](#)
[Planets](#)
[Jupyter notebook Planets](#)
[Jupyter StackOverflow](#)
[Jupyter StackOverflow - selected columns](#)
[Jupyter processing chunks](#)
[Jupyter StackOverflow - selected rows](#)
[Jupyter StackOverflow - biggest countries \(in terms of number of responses\)](#)
[Jupyter StackOverflow - historgram](#)
[Jupyter StackOverflow - filter by country](#)
[Jupyter StackOverflow - OpenSourcer](#)
[Jupyter StackOverflow - cross tabulation](#)
[Jupyter StackOverflow - salaries](#)
[Jupyter StackOverflow - replace values](#)
[Jupyter StackOverflow - selected rows](#)
[Jupyter notebook Intellisense \(TAB completition\)](#)

[Jupyter examples](#)

[IPy Widgets](#)

[Testing](#)

[Traditional Organizations](#)

[Quality Assurance](#)

[Web age Organizations](#)

[TDD vs Testing as an Afterthought](#)

[Why test?](#)

[Testing Modes](#)

[Testing Applications](#)

[Testing What to test?](#)

[Testing in Python](#)

[Testing Environment](#)

[Testing Setup - Fixture](#)

[Testing Resources](#)

[Testing with unittest](#)

[Use a module](#)

[Test a module](#)

[The tested module](#)

[Testing - skeleton](#)

[Testing](#)

[Test examples](#)

[Testing with PyTest](#)

[Pytest features](#)

[Pytest setup](#)

[Testing with Pytest](#)

[Testing functions](#)

[Testing class and methods](#)

[Pytest - execute](#)

[Pytest - execute](#)

[Pytest simple module to be tested](#)

[Pytest simple tests - success](#)

[Pytest simple tests - success output](#)

[Pytest simple tests - failure](#)

[Pytest simple tests - failure output](#)

[Exercise: test math functions](#)

[Exercise: test this app](#)

[Exercise: test the csv module](#)

[Solution: Pytest test math functions](#)

[Solution: Pytest test this app](#)

[Solution: test the csv module](#)

[PyTest bank deposit](#)

[PyTest expected exceptions \(bank deposit\)](#)

[PyTest expected exceptions \(bank deposit\) - no exception happens](#)

[PyTest expected exceptions \(bank deposit\) - different exception is raised](#)

[PyTest expected exceptions](#)

[PyTest expected exceptions output](#)

[PyTest expected exceptions \(text changed\)](#)

[PyTest expected exceptions \(text changed\) output](#)

[PyTest expected exceptions \(other exception\)](#)

[PyTest expected exceptions \(other exception\) output](#)

[PyTest expected exceptions \(no exception\)](#)

[PyTest expected exceptions \(no exception\) output](#)

[PyTest: Multiple Failures](#)

[PyTest: Multiple Failures output](#)

[PyTest Selective running of test functions](#)

[PyTest: stop on first failure](#)

[Pytest: expect a test to fail \(xfail or TODO tests\)](#)

[Pytest: expect a test to fail \(xfail or TODO tests\)](#)

[PyTest: show xfailed tests with -rx](#)
[Pytest: skipping tests](#)
[Pytest: show skipped tests woth -rs](#)
[Pytest: show extra test summary info with -r](#)
[Pytest: skipping tests output in verbose mode](#)
[Pytest verbose mode](#)
[Pytest quiet mode](#)
[PyTest print STDOUT and STDERR using -s](#)
[PyTest failure reports](#)
[PyTest compare numbers](#)
[PyTest compare numbers relatively](#)
[PyTest compare strings](#)
[PyTest compare long strings](#)
[PyTest is one string in another strings](#)
[PyTest test any expression](#)
[PyTest element in list](#)
[PyTest compare lists](#)
[PyTest compare short lists](#)
[PyTest compare short lists - verbose output](#)
[PyTest compare dictionaries](#)
[PyTest compare dictionaries output](#)
[PyTest Fixtures](#)
[PyTest Fixture setup and teardown](#)
[PyTest Fixture setup and teardown output](#)
[PyTest: Class setup and teardown](#)
[PyTest: Class setup and teardown output](#)
[Pytest Dependency injection](#)
[Pytest fixture - tmpdir](#)
[Pytest capture STDOUT and STDERR with capsys](#)
[Pytest Fixture - home made fixtures](#)
[More fixtures](#)
[Pytest: Mocking - why?](#)

[Pytest: Mocking - what?](#)
[Pytest: One dimensional spacefight](#)
[Pytest: Mocking input and output](#)
[Pytest: Mocking random](#)
[Pytest: Flask echo](#)
[Pytest: testing Flask echo](#)
[PyTest: Run tests in parallel with xdist](#)
[PyTest: Order of tests](#)
[PyTest: Randomize Order of tests](#)
[PyTest: Force default order](#)
[PyTest: no random order](#)
[Anagram on the command line](#)
[PyTest testing CLI](#)
[PyTest test discovery](#)
[PyTest test discovery - ignore some tests](#)
[PyTest select tests by name](#)
[PyTest select tests by marker](#)
[PyTest: Test Coverage](#)
[Exercise: module](#)
[Exercise: Open Source](#)
[Pytest resources](#)
[Pytest and tempdir](#)
[PyTest compare short lists - output](#)
[PyTest with parameter](#)
[PyTest with parameters](#)
[Pytest reporting in JUnit XML format](#)
[No test selected](#)

Advanced functions

[Variable scopes](#)
[Name resolution order \(LEGB\)](#)
[Scoping: global seen from function](#)

Assignment creates local scope

Local scope gone wrong

Changing global variable from a function

Global variables mutable in functions

Scoping issues

sub in sub

Scoping sub in sub (enclosing scope)

Function objects

Functions are created at run time

Mutable default

Use None as default parameter

Inner function created every time the outer function runs

Static variable

Static variable in generated function

Inspect

Variable number of function arguments

Python function arguments - a reminder

Functions with unknown number of arguments

Variable length argument list with * and **

Passing arguments as they were received (but incorrectly).

Unpacking args before passing them on

Exercise: implement the my_sum function

Solution: implement the my_sum function

Exercise: implement the reduce function

Solution: implement the reduce function

Exercise: sort pairs

Solution: sort pairs

Python Packages

Why Create package

Create package

[Internal usage](#)

[use module in package - relative path](#)

[use package \(does not work\)](#)

[package importing \(and exporting\) module](#)

[use package \(module\) with import](#)

[use package with import](#)

[Creating an installable Python package](#)

[Create tar.gz file](#)

[Install Package](#)

[Dependencies](#)

[Add README file](#)

[Add README file \(`setup.py`\).](#)

[Include executables](#)

[Add tests](#)

[Add tests calc](#)

[Add tests all](#)

[`setup.py`](#)

[Run tests and create package](#)

[Packaging applications \(creating executable binaries\)](#)

[Using PyInstaller](#)

[Other PyInstaller examples](#)

[Other](#)

[Py2app for Mac](#)

[Exercise: package](#)

[Exercise: create executable](#)

[Ctypes](#)

[`ctypes` - hello](#)

[concat](#)

[links](#)

[Advanced OOP](#)

Class count instances

Class Attributes

Class Attributes in Instances

Attributes with method access

Instance Attribute

Methods are class attributes

Monkey patching

Classes: instance method

Class methods and class attributes

Classes: constructor

Class methods - alternative constructor

Abstract Base Class

Abstract Base Class with abc

ABC working example

ABC - cannot instantiate the base-class

ABC - must implement methods

Use Python @property to fix bad interface (the bad interface).

Use Python @property to fix bad interface (first attempt)

Use Python @property to fix bad API

Use Python @property decorator to fix bad API

Use Python @property for value validation

class and static methods

Destructor: del

Destructor delayed

Destructor delayed for both

Opearator overloading

Operator overloading methods

Exercise: rectangular

Exercise: SNMP numbers

[Exercise: Implement a Gene inheritance model combining DNA](#)

[Exercise: imaginary numbers - complex numbers](#)

[Solution: Rectangular](#)

[Solution: Implement a Gene inheritance model combining DNA](#)

[Instance counter](#)

[2to3](#)

[Converting from Python 2 to Python 3](#)

[division](#)

[print in Python 2](#)

[print in Python 3](#)

[input and raw_input](#)

[Code that works on both 2 and 3](#)

[Compare different types](#)

[Octal numbers](#)

[2to3 Resources](#)

[Design Patterns](#)

[What are Design Patterns?](#)

[Don't replace built-in objects](#)

[Facade - simple interface to complex system](#)

[Monkey Patching](#)

[Creation DPs "Just One"](#)

[Singleton](#)

[Monostate \(Borg\)](#)

[Dispatch table](#)

[Parallel](#)

[Types of Problems](#)

[Types of solutions](#)

[How many parallels to use?](#)

[Dividing jobs](#)

[Performance Monitoring](#)

[Threads](#)

[Python Threading docs](#)

[Threaded counters](#)

[Simple threaded counters](#)

[Simple threaded counters \(parameterized\)](#)

[Pass parameters to threads - Counter with attributes](#)

[Create a central counter](#)

[Lock - acquire - release](#)

[Counter - plain](#)

[GIL - Global Interpreter Lock](#)

[Thread load](#)

[Exercise: thread files](#)

[Exercise: thread URL requests.](#)

[Exercise: thread queue](#)

[Solution: thread queue](#)

[Solution: thread URL requests.](#)

[Forking](#)

[Fork](#)

[Forking](#)

[Fork skeleton](#)

[Fork with load](#)

[Fork load results](#)

[Marshalling / Serialization](#)

[Fork with random](#)

[Exercise: fork return data](#)

[Solution: fork return data](#)

Asynchronous programming with AsyncIO

Sync chores

Async chores

Explanation

Coroutines

More about asyncio

Async files

Asynchronous programming with Twisted

About Twisted

Echo

Echo with log

Simple web client

Web client

Multiprocess

Multiprocess CPU count

Multiprocess Process

Multiprocess N files: Pool

Multiprocess load

Multiprocess: Pool

Multiprocess load async

Multiprocess and logging

Exercise: Process N files in parallel

Exercise: Process N Excel files in parallel

Exercise: Fetch URLs in parallel

Exercise: Fetch URLs from one site.

Solution: Fetch URLs in parallel

Multitasking

What is Multitasking?

Multitasking example

[Multitasking example with wait](#)
[Multitaksing - second loop waits for first one](#)
[Multitasking counter](#)
[Multitasking counter with thread locking](#)

[Improving Performance - Optimizing code](#)

[Problems](#)
[Optimization strategy](#)
[Locate the source of the problem](#)
[Optimizing tactics](#)
[DSU: Decorate Sort Undecorate](#)
[Profile code](#)
[Slow example](#)
[profile slow code](#)
[cProfile slow code](#)
[Benchmarking](#)
[Benchmarking subs](#)
[Levenshtein distance](#)
[Generate words](#)
[Levenshtein - pylev](#)
[Levenshtein - editdistance](#)
[Editdistance benchmark](#)
[A Tool to Generate text files](#)
[Count characters](#)
[Memory leak](#)
[Garbage collection](#)
[Weak reference](#)
[Exercise: benchmark list-comprehension, map, for](#)
[Exercise: Benchmark Levenshtein](#)
[Exercise: sort files](#)
[Exercise: compare split words:](#)
[Exercise: count words](#)

GUI with Python/Tk

[Sample Tk app](#)

[GUI Toolkits](#)

[Installation](#)

[Python Tk Documentation](#)

[Python Tk Button](#)

[Python Tk Button with action](#)

[Python Tk Label](#)

[Python Tk Label - font size and color](#)

[Python Tk Keybinding](#)

[Python Tk Entry \(one-line text entry\)](#)

[Python Tk Entry for passwords and other secrets \(hidden text\)](#)

[Python Tk Checkbox](#)

[Python Tk Radiobutton](#)

[Python Tk Listbox](#)

[Python Tk Listbox Multiple](#)

[Python Tk Menubar](#)

[Python Tk Text](#)

[Python Tk Dialogs](#)

[Python Tk Filedialog](#)

[Python Tk messagebox](#)

[Python Tk Combobox](#)

[Python Tk OptionMenu](#)

[Python Tk Scale](#)

[Python Tk Progressbar](#)

[Python Tk Frame](#)

[Not so Simple Tk app with class](#)

[Tk: Hello World](#)

[Tk: Quit button](#)

[Tk: File selector](#)

[Tk: Checkbox](#)

[Tk: Runner](#)

[Tk: Runner with threads](#)

[Getting started with Tk](#)

[Exercise: Tk - Calculator one line](#)

[Exercise: Tk Shopping list](#)

[Exercise: Tk TODO list](#)

[Exercise: Tk Notepad](#)

[Exercise: Tk Copy files](#)

[Exercise: Tk](#)

[Solution: Tk - Calculator one line](#)

[Solution: Tk](#)

[Solution: Tk Notepad](#)

[Simple file dialog](#)

[Python Pitfalls](#)

[Reuse of existing module name](#)

[Use the same name more than once](#)

[Compare string and number](#)

[Compare different types](#)

[Sort mixed data](#)

[Linters](#)

[Static Code Analyzis - Linters](#)

[PEP8](#)

[F811 - redefinition of unused](#)

[Warn when Redefining functions](#)

[Python .NET](#)

[IronPython](#)

[Use .NET libraries from Python](#)

[Python and .NET console](#)

[Python and .NET examples](#)

[Exercise Python and .NET](#)

[Python and Java](#)

[Jython](#)

[Calling Java from Python](#)

[Jython - Python running on the JVM](#)

[Jython Installation](#)

[Jython Installation](#)

[Jython load Java class](#)

[Jython load Java class in code](#)

[Jython test Java class](#)

[PIL - Pillow](#)

[Install Pillow](#)

[Create First Image](#)

[Write Text on Image](#)

[Select font for Text on Image](#)

[Font directories](#)

[Get size of an Image](#)

[Get size of text](#)

[Resize an existing Image](#)

[Crop an existing Image](#)

[Combine two images](#)

[Rotated text](#)

[Rotated text in top-right corner](#)

[Embed image \(put one image on another one\).](#)

[Draw a triangle](#)

[Draw a triangle and write text in it](#)

[Draw a triangle and write rotated text in it](#)

[Draw a rectangular](#)

[Draw a rectangle](#)

[Draw circle](#)

[Draw heart](#)

[Rectangle with rounded corners](#)

[TODO](#)

[FAQ](#)

[How not to name example scripts?](#)

[Platform independent code](#)

[How to profile a python code to find causes of slowness?](#)

[pdb = Python Debugger](#)

[Avoid Redefining functions](#)

[Appendix](#)

[print function](#)

[Dividers \(no break or continue\)](#)

[Lambdas](#)

[Abstract Class](#)

[Remove file](#)

[Modules: more](#)

[import hooks](#)

[Python resources](#)

[Progress bar](#)

[from future](#)

[Variable scope](#)

[scope](#)

[type](#)

[Look deeper in a list](#)

[Exercise: iterators - count](#)

[Simple function \(before generators\)](#)

[Other slides](#)

Other slides

Atom for Python

IDLE - Integrated Development Environment

sh-bang - executable on Linux/Apple

Strings as Comments

pydoc

How can I check if a string can be converted to a number?

Spyder Intro

Interactive Debugging

Parameter passing

Command line arguments and main

Infinite loop

break

continue

While with many conditions

while loop with many conditions

Format with conversion (stringification with str or repr)

Name of the current function in Python

Name of the caller function in Python

Stack trace in Python using inspect

Module Fibonacci

PyTest - assertion

PyTest - failure

PyTest - list

SAX with coroutine

Getting the class name of an object

Inheritance - super

Inheritance - super - other class

iterator - pairwise

iterator - grouped

itertools - groupby

Circular references

[Context managers: with \(file\) experiments](#)
[itertools - izip](#)
[mixing iterators](#)
[mixing iterators](#)
[itertools - pairwise](#)
[itertools - grouped](#)
[range vs xrange in Python](#)
[profile \(with hotshot\) slow code](#)
[Abstract Base Class without abc](#)
[Abstract Base Class with abc Python 2 ?](#)
[Abstract Base Class with metaclass](#)
[Create class with metaclass](#)
[Python Descriptors](#)
[alter iterator](#)
[Create a counter queue](#)
[A Queue of tasks](#)
[Filtered Fibonacci with ifilter](#)
[Python from .NET](#)

First steps

What is Python?

- A snake.
- A British comedy group called [Monty Python](#).
- A programming language. The definition of the language: words, punctuation (operators) and grammar (syntax).
- The compiler/interpreter of the Python programming language. (aka. CPython).

When people say they Python in relation to programming they either mean the Python programming language or they mean the tool that can translate some text (code) written in the Python programming language to the language a computer can actually understand. On MS Windows this is the **python.exe** you need to install. On Linux/Mac it is usually called **python** or **python3**. The generic name of the tool that translates a programming language for the computer is either called a compiler or an interpreter. We'll talk about this later on.

What is needed to write a program?

- An **editor** where we can write in a language.
- A **compiler or interpreter** that can translate our text to the language of the computer.

In order to write and run a program you basically need two things. A text editor in which you can write the program and a compiler or interpreter that can translate this program to the computer.

The source (code) of Python

- [Python](#)

Python 2 vs. Python 3

- Python 2.x - old, legacy code at companies, answers on the Internet. Retires on January 1, 2020.
- Python 3.x - the one that you should use. (not fully backward compatible) Available since December 3, 2008.

Python has two major lines the version 2.x and the version 3.x. In a nutshell you **should** always use Python 3 if possible.

Unfortunately you can still encounter many companies and many projects in companies that are stuck on Python 2.
In such cases you probably will have to write in Python 2.

In addition when you search for solutions on the Internet in many cases you'll encounter solution that were written for Python 2. Luckily in most of the cases it is almost trivial to convert these small examples to work on Python 3.
You just need to be able to recognize that the code was originally written for Python 2 and you need to be able to make the adjustments.

For this reason, while the majority of these pages cover Python 3, we are going to point out the places where it might be useful to know how Python 2 works.

You are free to skip these parts and come back to them when the need arises.

Installation

- MS Windows
- Linux
- Apple/Mac OSX

We are going to cover how to install Python all 3 major operating systems.

Installation on Linux

- On Linux you usually have Python 2 installed in **/usr/bin/python**
- Python 3 in **/usr/bin/python3**.
- If they are not installed, you can install them with the appropriate **yum** or **apt-get** command of your distribution.
- An alternative is to install [Anaconda with Python 3.x](#)

```
1 $ which python3
2
3 $ sudo apt-get install python3
4 $ sudo yum install python3
```

Installation on Apple Mac OSX

- On Mac OSX you can have Python 2 installed in **/usr/bin/python** and Python 3 installed as **/usr/bin/python3**.
- [Homebrew](#)
- An alternative is to install [Anaconda with Python 3.x](#)

```
1 $ which python3
2
3 $ brew install python3
```

Installation on MS Windows

- [Anaconda with Python 3.x](#)
 - Anaconda shell
 - Anaconda Jupyter notebook
- *
- An alternative is to [install from here](#).

Editors, IDEs

Basically you can use any text editor to write Python code. The minimum I recommend is to have proper syntax highlighting. IDEs will also provide intellisense, that is, in most of the cases they will be able to understand what kind of objects do you have in your code and will be able to show you the available methods and their parameters. Even better, they provide powerful debuggers.

PyCharm seems to be the most popular IDE. It has a free version called community edition.

Linux

- [Emacs](#)
- [vi, vim, gvim](#)
- [spf13-vim](#)
- [Kate](#)
- [Gedit](#)
- [jEdit](#)

Windows

- [Notepad++](#)
- [Textpad](#)
- [Ultra Edit](#)

Mac

- [CotEditor](#)
- [TextWrangler](#)
- [TextMate](#)
- Type “text editor” in your Apple Store (filter to free)

All platforms

- [Sublime Text](#) (commercial)
- [Ligth Table](#)

IDEs

- [PyCharm community edition](#)
- [Visual Code of Microsoft](#)
- [Spyder](#), a scientific environment (included in Anaconda)
- [Jupyter](#) with [IPython](#) behind the scene.
- [IDLE](#) (comes with Python)
- [Komodo of ActiveState](#)
- [Aptana](#)
- [Pyscripter](#)
- [PyDev \(for Eclipse\)](#).
- [Wing IDE](#)
- [Atom](#)

Documentation

- [Google](#)
- [Bing](#)
- [DuckDuckGo](#)
- [official documentation of Python](#)
- [Stack Overflow](#)
- [Code Maven](#)
- ...

Program types

- Desktop application (MS Word, MS Excel, calculator, Firefox, Chrome, ...)
- Mobile applications - whatever runs on your phone.
- Embedded applications - software in your car or in your shoelace.

- Web applications - they run on the web server and send you HTML that your browser can show.
- **Command Line Applications**
- Scripts and programs are the same for our purposes
- ...

Python on the command line

More or less the only thing I do on the command line with python is to check the version number:

```
1 python -V  
2 python --version
```

You can run some Python code without creating a file, but I don't rememeber ever needing this. If you insists

```
1 python -c "print 42"
```

```
1 python3 -c "print(42)"
```

Type the following to get the details:

```
1 man python
```

cmdline

First script - hello world

```
1 print("Hello World")
```

- Create a file called **hello.py** with the above content.

- Open your terminal or the Anaconda Prompt on MS Windows in the directory (folder)
- Change to the directory where you saved the file.
- Run it by typing **python hello.py** or **python3 hello.py**
- The extension is .py - mostly for the editor (but also for modules).
- Parentheses after print() are required in Python 3, but use them even if you are stuck on Python 2.

Examples

- [The examples are on GitHub](#)
- You can download them and unzip them.

Comments

marks single line comments.

There are no real multi-line comments in Python, but we will see a way to have them anyway.

```
1 print("hello")
2
3 # Comments for other developers
4
5 print("world") # more comments
6
7 # print("This is not printed")
```

Variables

```
1 greeting = "Hello World!"  
2 print(greeting)
```

Exercise: Hello world

Try your environment:

- Make sure you have access to the right version of Python.
- Install Python if needed.
- Check if you have a good editor with syntax highlighting.
- Write a simple script that prints **Hello world.**
- Add some comments to your code.
- Create a variable, assign some text to it and then print out the content of the variable.

What is programming?

- Use some language to tell the computer what to do.
- Like a cooking recepie it has step-by-step instructions.
- Taking a complex problem and dividing it into small steps a computer can do.

What are the programming languages

- A computer CPU is created from transistors, 1 and 0 values. (aka. bits)
- Its language consists of numbers. (e.g 37 means move the content of ax register to bx register)
- English? too complex, too much ambiguity.
- Programming languages are in-betweeen.

A written human language

- Words
- Punctuation: - . , ! ?
- Grammar
- ...

A programming language

- Built-in words: print, len, type, def, ...
- Literal values: numbers, strings
- Operators: + - * = , ; ...
- Grammar (syntax)
- User-created words: variables, functions, classes, ...

Words and punctuation matter!

- What did you chose? (Correctly: choose, but people will usually understand.)
- Lets do the homework. (Correctly: Let's, but most people will understand.)
- Let's eat, grandpa!
- Let's eat grandpa!
- see more
- Programming languages have a lot less words, but they are very strict on the grammar (syntax).
- A mising comma can break your code.
- A missing space will change the meaning of your code.
- An incorrect word can ruin your day.

Literals, Value Types in Python

```
1 print( type(23) )           # int
2 print( type(3.14) )          # float
3 print( type("hello") )       # str
4
5 print( type("23") )          # str
6 print( type("3.24") )        # str
7
8 print( type(None) )          # NoneType
9 print( type(True) )           # bool
10 print( type(False) )         # bool
11
12 print( type([]) )            # list
13 print( type({}) )            # dict
14
15 print( type(hello) )         # NameError: name 'hello' is
not defined
16 print("Still running")
```

```
1 Traceback (most recent call last):
2   File "python/examples/basics/types.py", line 15, in
<module>
3     print( type(hello) )    # str
4 NameError: name 'hello' is not defined
```

- Strings must be enclosed in quotes.
- Numbers must be NOT enclosed in quotes.

Floating point limitation

```
1 print(0.1 + 0.2)    # 0.3000000000000004
```

- [floating point](#)

Value Types in Numpy

Numpy but also other programming languages might have them.

- int8
- int32
- float32
- float64
- ...

Rectangular (numerical operations)

```
1 width = 23
2 height = 17
3 area = width * height
4 print(area)      # 391
```

Multiply string

```
1 width = "23"
2 height = "17"
3 area = width * height
4 print(area)
```

```
1 Traceback (most recent call last):
2   File "python/examples/basics/rectangular_strings.py",
line 3, in <module>
3     area = width * height
4 TypeError: can't multiply sequence by non-int of type
'str'
```

Add numbers

```
1 a = 19
2 b = 23
3 c = a + b
4 print(c)      # 42
```

Add strings

```
1 a = "19"  
2 b = "23"  
3 c = a + b  
4 print(c)      # 1923
```

Exercise: Calculations

- Extend the rectangular_basic.py from above to print both the area and the circumference of the rectangle.
- Write a script that has a variable holding the radius of a circle and prints out the area of the circle and the circumference of the circle.
- Write a script that has two numbers a and b and prints out the results of a+b, a-b, a*b, a/b

Solution: Calculations

```
1 width = 23  
2 height = 17  
3 area = width * height  
4 print("The area is ", area)      # 391  
5 circumference = 2 * (width + height)  
6 print("The circumference is ", circumference)    # 80
```

```
1 r = 7  
2 pi = 3.14  
3 print("The area is ", r * r * pi)                  # 153.86  
4 print("The circumference is ", 2 * r * pi)        # 43.96
```

```
1 import math  
2  
3 r = 7  
4 print("The area is ", r * r * math.pi)            #  
153.9380400258998
```

```
5 print("The circumference is ", 2 * r * math.pi)  #
43.982297150257104
```

```
1 a = 3
2 b = 2
3
4 print(a+b)    # 5
5 print(a-b)    # 1
6 print(a*b)    # 6
7 print(a/b)    # 1.5
```

Second steps

Modules

```
1 import sys
2
3 print( sys.executable )                                     #
/home/gabor/venv3/bin/python
4 print( sys.platform )                                      # linux
5 print( sys.argv[0] )                                       #
examples/basics/modules.py
6 print( sys.version_info.major )                            # 3
7
8 print( sys.getsizeof( 1 ) )                                # 28
9 print( sys.getsizeof( 42 ) )                               # 28
10 print( sys.getsizeof( 1.0 ) )                             # 24
11
12 print( sys.getsizeof( "" ) )                              # 49
13 print( sys.getsizeof( "a" ) )                            # 50
14 print( sys.getsizeof( "ab" ) )                           # 51
15 print( sys.getsizeof( "abcdefghijkl" ) )                 # 59
```

A main function

```
1 def main():
2     print("Hello")
3     print("World")
```

This won't run as the main function is declared, but it is never called (invoked).

The main function - called

You could write your code in the main body of your Python file, but using functions and passing arguments to it will make your code easier to maintain and understand. Therefore I recommend that you always write every script with a function called “main”.

- Function definition starts with the **def** keyword, followed by the name of the new function (“main” in this case), followed by the list of **parameters in parentheses** (nothing in this case).
- The content or body of the function is then **indented** to the right.
- The function definition ends when the indentation stops.

```
1 def main():
2     print("Hello")
3     print("World")
4
5 print("before")
6 main()
7 print("after")
```

```
1 before
2 Hello
3 World
4 after
```

- Use a main function to avoid globals and better structure your code.
- Python uses **indentation** for blocks instead of curly braces, and it uses the colon : to start a block.

Indentation

- Standard recommendations: 4 spaces on every level.

Conditional main

```
1 def main():
2     print("Hello World")
3
4 if __name__ == "__main__":
5     main()
```

- We'll cover this later but in case you'd like, you can include this conditional execution of the main function.

Input - Output I/O

Input

- Keyboard (Standard Input, Command line, GUI)
- Mouse (Touch pad)
- Touch screen
- Files, Filesystem
- Network (e.g. in Web applications)

Output

- Screen
- File
- Network

print in Python 2

print is one of the keywords that changed between Python 2 and Python 3. In Python 2 it does not need parentheses, in Python 3 it is a function and it needs to have parentheses.

```
1 print "hello"  
2 print "world"  
3 print "Foo", "Bar"
```

```
1 hello  
2 world  
3 Foo Bar
```

```
1 print "hello",  
2 print "world"  
3 print "Foo", "Bar",
```

```
1 hello world  
2 Foo Bar
```

No newline, but a space is added at the end of the output and between values.

```
1 import sys  
2 sys.stdout.write("hello")  
3 sys.stdout.write("world")
```

```
1 helloworld
```

write takes exactly one parameter

print in Python 3

```
1 print("hello")
2 print("world")
3 print("Foo", "Bar")
```

```
1 hello
2 world
3 Foo Bar
```

```
1 print("hello", end=" ")
2 print("world")
3 print("Foo", "Bar")
```

```
1 hello world
2 Foo Bar
```

end will set the character added at the end of each print statement.

```
1 print("hello", end="")
2 print("world")
3
4 print("Foo", "Bar", sep="")
5 print("END")
```

```
1 helloworld
2 FooBar
3 END
```

sep will set the character separating values.

print in Python 2 as if it was Python 3

```
1 from __future__ import print_function
2 print("hello", end="")
3 print("world")
```

```
1 helloworld
```

Exception: SyntaxError: Missing parentheses in call

What if we run some code with `print "hello"` using Python 3?

```
1 File "examples/basics/print.py", line 1
2     print "hello"
3
4 SyntaxError: Missing parentheses in call to 'print'.
Did you mean print("hello")?
```

Prompting for user input in Python 2

```
1 from __future__ import print_function
2
3 def main():
4     print("We have a question!")
5     name = raw_input('Your name: ')
6     print('Hello ' + name + ', how are you?')
7
8 main()
```

```
1 /usr/bin/python2 prompt2.py
2
3 We have a question!
4 Your name: Foo Bar
5 Hello Foo Bar, how are you?
```

What happens if you run this with Python 3 ?

```
1 /usr/bin/python3 prompt2.py
```

```
1 We have a question!
2 Traceback (most recent call last):
```

```
3  File "prompt2.py", line 7, in <module>
4      main()
5  File "prompt2.py", line 4, in main
6      name = raw_input('Your name: ')
7 NameError: name 'raw_input' is not defined
```

Prompting for user input in Python 3

In Python 3 the **raw_input()** function was replaced by the **input()** function.

```
1 def main():
2     print("We have a question!")
3     name = input('Your name: ')
4     print('Hello ' + name + ', how are you?')
5
6 main()
```

What happens if you run this using Python 2 ?

```
1 /usr/bin/python2 prompt3.py
```

```
1 We have a question!
2 Your name: Foo Bar
3 Your name: Traceback (most recent call last):
4   File "prompt3.py", line 5, in <module>
5       main()
6   File "prompt3.py", line 2, in main
7       name = input('Your name: ')
8   File "<string>", line 1
9       Foo Bar
10          ^
11 SyntaxError: unexpected EOF while parsing
```

```
1 We have a question!
2 Your name: Foo
3 Your name: Traceback (most recent call last):
4   File "prompt3.py", line 5, in <module>
5     main()
6   File "prompt3.py", line 2, in main
7     name = input('Your name: ')
8   File "<string>", line 1, in <module>
9 NameError: name 'Foo' is not defined
```

Python2 input or raw_input?

In Python 2 always use `raw_input()` and never `input()`.

Prompting both Python 2 and Python 3

```
1 from __future__ import print_function
2 import sys
3
4 def main():
5     if sys.version_info.major < 3:
6         name = raw_input('Your name: ')
7     else:
8         name = input('Your name: ')
9     print('Hello ' + name + ', how are you?')
10
11 main()
```

Add numbers entered by the user (oops)

```
1 def main():
2     a = input('First number: ')
3     b = input('Second number: ')
4     print(a + b)
5
6 main()
```

```
1 First number: 2
2 Second number: 3
3 23
```

When reading from the command line using `input()`, the resulting value is a string.

Even if you only typed in digits. Therefore the addition operator `+` concatenates the strings.

Add numbers entered by the user (fixed)

```
1 def main():
2     a = input('First number: ')
3     b = input('Second number: ')
4     print(int(a) + int(b))
5
6 main()
```

```
1 First number: 2
2 Second number: 3
3 5
```

In order to convert the string to numbers use the `int()` or the `float()` functions.

Whichever is appropriate in your situation.

How can I check if a string can be converted to a number?

- [stdtypes](#)
-

```
1 val = input("Type in a number: ")
2 print(val)
3 print(val.isdecimal())
4 print(val.isnumeric())
```

```
5
6 if val.isdecimal():
7     num = int(val)
8     print(num)
```

```
1 Type in a number: 42
2 True
3 True
4 42
```

- We'll talk about this later. For now assume that the user enters something that can be converted to a number.
- Use Regular Expressions (regexes) to verify that the input string looks like a number.
- Wrap the code in try-except block to catch any exception raised during the conversion.

Converting string to int

```
1 a = "23"
2 print(a)          # 23
3 print( type(a) ) # <class 'str'>
4
5
6 b = int(a)
7 print(b)          # 23
8 print( type(b) ) # <class 'int'>
```

```
1 a = "42 for life"
2 print(a)          # 42 for life
3 print( type(a) ) # <class 'str'>
4
5 b = int(a)
6 print(b)
7 print( type(b) )
8
9 # Traceback (most recent call last):
10 #   File "converting_string_to_int.py", line 5, in
```

```
<module>
11 #     b = int(a)
12 # ValueError: invalid literal for int() with base 10:
'42 for life'
```

Converting float to int

```
1 a = 2.1
2 print( type(a) )    # <class 'float'>
3 print(a)            # 2.1
4
5 b = int(2.1)
6 print( type(b) )    # <class 'int'>
7 print(b)            # 2
```

```
1 a = "2.1"
2 print(a)          # 2.1
3 print( type(a) )  # <class 'str'>
4
5 b = int(a)
6 print(b)
7 print( type(b) )
8
9 # Traceback (most recent call last):
10 #   File "converting_floating_string_to_int.py", line
5, in <module>
11 #     b = int(a)
12 # ValueError: invalid literal for int() with base 10:
'2.1'
```

```
1 a = "2.1"
2 b = float(a)
3 c = int(b)
4 print(c)          # 2
5 print( type(a) )  # <class 'str'>
6 print( type(b) )  # <class 'float'>
7 print( type(c) )  # <class 'int'>
8
9 d = int( float(a) )
10 print(d)         # 2
11 print( type(d) ) # <class 'int'>
```

```
12
13 print( int( float(2.1) ) )    # 2
14 print( int( float("2") ) )    # 2
15 print( int( float(2) ) )     # 2
```

Conditionals: if

```
1 def main():
2     expected_answer = "42"
3     inp = input('What is the answer? ')
4
5     if inp == expected_answer:
6         print("Welcome to the cabal!")
7
8 main()
```

Conditionals: if - else

```
1 def main():
2     expected_answer = "42"
3     inp = input('What is the answer? ')
4
5     if inp == expected_answer:
6         print("Welcome to the cabal!")
7     else:
8         print("Read the Hitchhiker's guide to the
galaxy!")
9
10 main()
```

Conditionals: if - else (other example)

```
1 def main():
2     a = input('First number: ')
3     b = input('Second number: ')
4
5     if int(b) == 0:
6         print("Cannot divide by 0")
7     else:
8         print("Dividing", a, "by", b)
```

```
9         print(int(a) / int(b))
10
11
12 main()
```

Conditionals: else if

```
1 def main():
2     a = input('First number: ')
3     b = input('Second number: ')
4
5     if a == b:
6         print('They are equal')
7     else:
8         if int(a) < int(b):
9             print(a + ' is smaller than ' + b)
10            else:
11                print(a + ' is bigger than ' + b)
12
13 main()
```

Conditionals: elif

```
1 def main():
2     a = input('First number: ')
3     b = input('Second number: ')
4
5     if a == b:
6         print('They are equal')
7     elif int(a) < int(b):
8         print(a + ' is smaller than ' + b)
9     else:
10        print(a + ' is bigger than ' + b)
11
12
13 main()
```

Ternary operator

```
1 x = 3
2 answer = 'positive' if x > 0 else 'negative'
3 print(answer)    # positive
4
5 x = -3
6 answer = 'positive' if x > 0 else 'negative'
7 print(answer)    # negative
```

```
1 x = 3
2 if x > 0:
3     answer = 'positive'
4 else:
5     answer = 'negative'
6 print(answer)    # positive
7
8 x = -3
9 if x > 0:
10    answer = 'positive'
11 else:
12    answer = 'negative'
13 print(answer)    # negative
```

Case or Switch in Python

- There is no case or switch statement in Python.

Exercise: Rectangular

- Write a script that will ask for the sides of a rectangular and print out the area.
- Provide error messages if either of the sides is negative.

```
1 python rect.py
2 Side: 3
3 Side: 4
4 The area is 12
```

Exercise: Calculator

Create a script that accepts 2 numbers and an operator (+, -, *, /), and prints the result of the operation.

```
1 python calc.py
2 Operand: 19
3 Operand: 23
4 Operator: +
5 Results: 42
```

Exercise: Standard Input

- In the previous exercises we expected the userinput to come in on the “Standard Input” aka. STDIN.
- If you would like to practice this more, come up with other ideas, try to solve them and tell me about the task. (in person or via e-mail.)
- (e.g. you could start building an interactive role-playing game.)

Solution: Area of rectangular

```
1 def main():
2     #length = 10
3     #width = 3
4
5     length = int(input('Length: '))
6     width = int(input('Width: '))
7
8     if length <= 0:
9         print("length is not positive")
10        return
11
12    if width <= 0:
13        print("width is not positive")
14        return
15
```

```
16     area = length * width
17     print("The area is ", area)
18
19 main()
```

Same in Python 2

```
1 from __future__ import print_function
2
3 def main():
4     #length = 10
5     #width = 3
6
7     length = int(raw_input('Length: '))
8     width = int(raw_input('Width: '))
9
10    if length <= 0:
11        print("length is not positive")
12        return
13
14    if width <= 0:
15        print("width is not positive")
16        return
17
18    area = length * width
19    print("The area is ", area)
20
21 main()
```

Solution: Calculator

```
1 def main():
2     a = float(input("Number: "))
3     b = float(input("Number: "))
4     op = input("Operator (+-*/): ")
5
6     if op == '+':
7         res = a+b
8     elif op == '-':
9         res = a-b
10    elif op == '*':
```

```
11         res = a*b
12     elif op == '/':
13         res = a/b
14     else:
15         print("Invalid operator: '{}'".format(op))
16     return
17
18     print(res)
19     return
20
21
22 main()
```

Same in Python 2

```
1 from __future__ import print_function
2
3 a = float(raw_input("Number: "))
4 b = float(raw_input("Number: "))
5 op = raw_input("Operator (+-* /): ")
6
7 if op == '+':
8     res = a+b
9 elif op == '-':
10    res = a-b
11 elif op == '*':
12    res = a*b
13 elif op == '/':
14    res = a/b
15 else:
16     print("Invalid operator: '{}'".format(op))
17     exit()
18
19
20 print(res)
```

Command line arguments

```
1 import sys
2
3 def main():
```

```
4     print(sys.argv)
5     print(sys.argv[0])
6     print(sys.argv[1])
7     print(sys.argv[2])
8
9 main()
```

```
1 $ python examples/basic/cli.py one two
```

```
1 ['examples/basics/cli.py', 'one', 'two']
2 examples/basics/cli.py
3 one
4 two
```

```
1 $ python examples/basic/cli.py
```

```
1 ['examples/basics/cli.py']
2 examples/basics/cli.py
3 Traceback (most recent call last):
4   File "examples/basics/cli.py", line 6, in <module>
5     print(sys.argv[1])
6 IndexError: list index out of range
```

Command line arguments - len

```
1 import sys
2
3 def main():
4     print(sys.argv)
5     print(len(sys.argv))
6
7 main()
```

Command line arguments - exit

```
1 import sys
2
```

```
3 def main():
4     if len(sys.argv) != 2:
5         exit("Usage: " + sys.argv[0] + " VALUE")
6     print("Hello " + sys.argv[1])
7
8 main()
```

```
1 echo %errorlevel%
2 echo $?
```

Exercise: Rectangular (argv)

- Change the above script that it will accept the arguments on the command line like this: python rect.py 2 4

Exercise: Calculator (argv)

- Create a script that accepts 2 numbers and an operator (+, -, *, /), on the command line and prints the result of the operation.
- python calc.py 2 + 3
- python calc.py 6 / 2
- python calc.py 6 * 2

Solution: Area of rectangular (argv)

```
1 import sys
2
3 def main():
4     if len(sys.argv) != 3:
5         exit("Needs 2 arguments: width length")
6
7     width = int( sys.argv[1] )
8     length = int( sys.argv[2] )
9
10    if length <= 0:
```

```
11         exit("length is not positive")
12
13     if width <= 0:
14         exit("width is not positive")
15
16     area = length * width
17     print("The area is ", area)
18
19 main()
```

Solution: Calculator eval

```
1 def main():
2     a = input("Number: ")
3     b = input("Number: ")
4     op = input("Operator (+-* /): ")
5
6     command = a + op + b
7     print(command)
8     res = eval(command)
9     print(res)
10
11 main()
```

```
1 $ python examples/basics/calculator_eval.py
2
3 Number: 2
4 Number: 3
5 Operator (+-* /): +
6 2+3
7 5
```

Solution: Calculator (argv)

```
1 import sys
2
3
4 def main():
5     if len(sys.argv) < 4:
6         exit("Usage: " + sys.argv[0] + " OPERAND
```

```

OPERATOR OPERAND")
7
8     a = float(sys.argv[1])
9     b = float(sys.argv[3])
10    op = sys.argv[2]
11
12    if op == '+':
13        res = a + b
14    elif op == '-':
15        res = a - b
16    elif op == '*':
17        res = a * b
18    elif op == '/':
19        res = a / b
20    else:
21        print("Invalid operator: '{}'".format(op))
22        exit()
23
24    print(res)
25
26 main()

```

The multiplication probably won't work because the Unix/Linux shell replaces the * by the list of files in your current directory and thus the python script will see a list of files instead of the *. This is not your fault as a programmer. It is a user error. The correct way to run the script is `python calc.py 2 '*' 3`.

Compilation vs. Interpretation

Compiled

- Languages: C, C++
- Development cycle: Edit, Compile (link), Run.
- Strong syntax checking during compilation and linking.
- Result: Stand-alone executable code.

- Need to compile to each platform separately. (Windows, Linux, Mac, 32bit vs 64bit).

Interpreted

- Shell, BASIC
- Development cycle: Edit, Run.
- Syntax check only during run-time.
- Result: we distribute the source code.
- Needs the right version of the interpreted on every target machine.

Both?

- Java (running on JVM - Java Virtual Machine)
- C# (running on CLR - Common Language Runtime)

Is Python compiled or interpreted?

There are syntax errors that will prevent your Python code from running

```
1 x = 2
2 print(x)
3
4 if x > 3
```

```
1 File "examples/other/syntax_error.py", line 4
2     if x > 3
3
4 SyntaxError: invalid syntax
```

There are other syntax-like errors that will be only caught during execution

```
1 x = 2
2 print(x)
3 print(y)
4 y = 13
5 print(42)
```

```
1 2
2 Traceback (most recent call last):
3   File "compile.py", line 5, in <module>
4     print y
5 NameError: name 'y' is not defined
```

- Python code is first compiled to bytecode and then interpreted.
- CPython is both the compiler and the interpreter.
- Jython and IronPython are mostly just compiler to JVM and CLR respectively.

Flake8 checking

```
1 conda install flake8
2 pip install flake8
3
4 flake8 --ignore= compile.py
```

```
1 compile.py:3:7: F821 undefined name 'y'
2 compile.py:6:1: W391 blank line at end of file
```

Numbers

Numbers

```
1 a = 42    # decimal
2 h = 0xA   # 10 - hex                      - staring with 0x
3 o = 0o11  # 9   - octal                   - starting with 0o
4          # 011 works in Python 2.x but Python 3.x
5          # requires the o that works in
6          # (recent versions of) Python 2.x
7 b = 0b11  # 3   - binary numbers - starting with 0b
8
9 r = 2.3
10
11 print(a)  # 42
12 print(h)  # 10
13 print(o)  # 9
14 print(b)  # 3
15 print(r) # 2.3
```

In Python numbers are stored as decimals, but in the source code you can also use hexadecimal, octal, or binary notations.

This is especially useful if the domain you are programming in is using those kinds of numbers.

For example hardware engineers often talk in hexadecimal values.

In that case you won't need to constantly translate between the form used in the current domain and decimal numbers.

Operators for Numbers

```
1 a = 2
2 b = 3
3 c = 2.3
4
5 d = a + b
6 print(d)          # 5
7 print(a + b)    # 5
8 print(a + c)    # 4.3
9 print(b / a)     # 1.5  # see the __future__
10 print(b // a)   # 1      # floor division
11 print(a * c)    # 4.6
12
13 print(a ** b)   # 8      (power)
14
15 print(17 % 3)   # 2      (modulus)
16
17 a += 7          # is the same as a = a + 7
18 print(a)         # 9
19
20 # a++           # SyntaxError: invalid syntax
21 # a--           # SyntaxError: invalid syntax
22
23 a += 1
24 print(a)         # 10
25 a -= 1
26 print(a)         # 9
```

There is no autoincrement (++) and autodecrement (--) in Python, because they can be expressed by `+= 1` and `-= 1` respectively.

Integer division and the future

```
1 from __future__ import print_function
2
3 print(3/2)
```

```
1 $ python divide.py
2 1
3
```

```
4 $ python3 divide.py  
5 1.5
```

```
1 from __future__ import print_function  
2 from __future__ import division  
3  
4 print(3/2)      # 1.5
```

If you need to use Python 2, remember that by default division is integer based so $3/2$ would return 1.

Importing the ‘division’ directive from **future** changes this to the behavior that we usually expect $3/2$ being 1.5.

This is also the behavior we have in Python 3.

In case you already use Python 3 and would like to get the “old” behavior, that is to get the integer part of the division, you can always call the “int” function: `int(b/a)`.

Pseudo Random Number

```
1 import random  
2  
3 a = random.random()  
4 print(a) # 0.5648261676148922 a value between 0.0 <=   
< 1.0  
5 print(random.random())  
6 print(random.random())
```

- [random](#)
- [Pseudo random generator](#)

Fixed random numbers

```
1 import random
2
3 random.seed(37)
4
5 print(random.random()) # 0.6820045605879779
6 print(random.random()) # 0.09160260807956389
7 print(random.random()) # 0.6178163488614024
```

Rolling dice - randrange

```
1 import random
2
3 print( 1 + int( 6 * random.random() ) )
4
5 print(random.randrange(1, 7))
6
7 # One of the following: 1, 2, 3, 4, 5, 6
```

Random choice

```
1 import random
2
3 letter = "abcdefghijklmno"
4 print(random.choice(letters))      # pick one of the
letters
5
6 fruits = ["Apple", "Banana", "Peach", "Orange",
"Durian", "Papaya"]
7 print(random.choice(fruits))
8      # pick one of the fruits
```

built-in method

- A common mistake. Not calling the method.

```
1 import random
2
3 rnd = random.random
```

```
4 print(rnd)      # <built-in method random of Random  
object at 0x124b508>  
5  
6  
7 y = rnd()  
8 print(y)        # 0.7740737563564781
```

When you see a string like the above “built-in method ...” you can be almost certainly sure that you have forgotten the parentheses at the end of a method call.

Exception: TypeError: ‘module’ object is not callable

- A common mistake. Calling the class and not the method.

```
1 import random  
2  
3 print("hello")  
4 x = random()  
5 print(x)
```

```
1 Traceback (most recent call last):  
2   File "examples/numbers/rnd.py", line 3, in <module>  
3     x = random()  
4 TypeError: 'module' object is not callable
```

Fixing the previous code

```
1 import random  
2
```

```
3 x = random.random()  
4 print(x)
```

```
1 from random import random  
2  
3 x = random()  
4 print(x)
```

Exception: AttributeError: module 'random' has no attribute

- A common mistake. Using the wrong filename.

This works fine:

```
1 print("Hello World")
```

This gives an error

```
1 import random  
2 print(random.random())
```

```
1 Traceback (most recent call last):  
2   File "rnd.py", line 2, in <module>  
3     print(random.random())  
4 AttributeError: module 'random' has no attribute  
'random'
```

Make sure the names of your files are not the same as the names of any of the python packages.

Exercise: Number guessing game - level 0

Level 0

- Using the random module the computer “thinks” about a whole number between 1 and 20.
- The user has to guess the number. After the user types in the guess the computer tells if this was bigger or smaller than the number it generated, or if was the same.
- The game ends after just one guess.

Level 1-

- Other levels in the next chapter.

Exercise: Fruit salad

- Write a script that will pick 3 fruits from a list of fruits like the one we had in one of the earlier slides. Print the 3 names.
- Could you make sure the 3 fruits are different?

```
1 fruits = ["Apple", "Banana", "Peach", "Orange",
"Durian", "Papaya"]
```

Solution: Number guessing game - level 0

```
1 import random
2
3 hidden = random.randrange(1, 21)
4 print("The hidden values is", hidden)
5
6 user_input = input("Please enter your guess: ")
7 print(user_input)
8
9 guess = int(user_input)
```

```
10 if guess == hidden:  
11     print("Hit!")  
12 elif guess < hidden:  
13     print("Your guess is too low")  
14 else:  
15     print("Your guess is too high")
```

Solution: Fruit salad

```
1 import random  
2  
3 fruits = ["Apple", "Banana", "Peach", "Orange",  
"Durian", "Papaya"]  
4 salad = random.sample(fruits, 3)  
5 print(salad)
```

Boolean

if statement again

```
1 x = 2
2
3 if x == 2:
4     print("it is 2")
5 else:
6     print("it is NOT 2")
7
8
9 if x == 3:
10    print("it is 3")
11 else:
12    print("it is NOT 3")
13
14 # it is 2
15 # it is NOT 3
```

True and False

- True and False are real boolean values.
-

```
1 x = 2
2
3 v = x == 2
4 print(v)
5 if v:
6     print(v, "is true - who would thought? ")
7
8 v = x == 3
9 print(v)
10 if v:
11     print(v, "is true - who would thought? ")
12 else:
13     print(v, "is false - who would thought? ")
```

```
14  
15 # True  
16 # True is true - who would thought?  
17 # False  
18 # False is false - who would thought?
```

Boolean

```
1 x = 23  
2  
3 if x:  
4     print("23 is true")  
5  
6 y = 0  
7 if y:  
8     print("0 is true")  
9 else:  
10    print("0 is false")  
11  
12 # 23 is true  
13 # 0 is false
```

True and False values in Python

- None
- 0
- "" (empty string)
- False
- []
- {}
- ()

Everything else is true.

```
1 values = [None, 0, "", False, [], (), {}, "0", True]  
2  
3 for v in values:  
4     if v:
```

```
5         print("True value: ", v)
6     else:
7         print("False value: ", v)
8
9 # False value: None
10 # False value: 0
11 # False value:
12 # False value: False
13 # False value: []
14 # False value: ()
15 # False value: {}
16 # True value: 0
17 # True value: True
```

`None` is like `undef` or `Null` or `Nill` in other languages.

Comparision operators

```
1 ==          equal
2 !=          not equal
3
4 <           less than
5 <=          less than or equal
6 >           greater than
7 >=          greater than or equal
```

```
1 a = "42"
2 b = 42
3
4 print(a == b)      # False
5 print(a != b)      # True
6 print(b == 42.0)    # True
7
8 print(None == None) # True
9 print(None == False) # False
```

Do NOT Compare different types

```
1 x = 12
2 y = 3
3 print(x > y)    # True
4
5 x = "12"
6 y = "3"
7 print(x > y)    # False
8
9 x = "12"
10 y = 3
11 print(x > y)   # True
12
13 x = 12
14 y = "3"
15 print(x > y)   # False
```

In Python 2 please be careful and only compare the same types.
Otherwise the result will look strange.

```
1 True
2 False
3 True
4 False
```

In Python 3, comparing different types raises exception:

```
1 True
2 False
3 Traceback (most recent call last):
4   File "examples/other/compare.py", line 6, in <module>
5     print(x > y)      # True
6 TypeError: '>' not supported between instances of 'str'
and 'int'
```

Boolean operators

- and
- or

- **not**
-

```
1 if COND:  
2     do something  
3 else:  
4     do something other  
5  
6 if not COND:  
7     do something other  
8  
9 if COND1 and COND2:  
10    do something  
11  
12 if COND1 or COND2:  
13    do something  
14  
15 if COND1 and not COND2:  
16    do something
```

Boolean truth tables

1 COND1 and COND2	Result
2 True True	True
3 True False	False
4 False True	False
5 False False	False

1 COND1 or COND2	Result
2 True True	True
3 True False	True
4 False True	True
5 False False	False

1 not COND	Result
2 True	False
3 False	True

Short circuit

```
1 def check_money():
2     return money > 1000000
3
4 def check_salary():
5     salary += 1
6     return salary >= 1000
7
8 while True:
9     if check_money() or check_salary():
10         print("I can live well")
```

Short circuit fixed

```
1 def check_money():
2     return money > 1000000
3
4 def check_salary():
5     salary += 1
6     return salary >= 1000
7
8 while True:
9     has_good_money = check_money()
10    has_good_salary = check_salary()
11
12    if has_good_money or has_good_salary:
13        print("I can live well")
```

Incorrect use of conditions

In your normal speech you could probably say something like “If status_code is 401 or 302, do something.”.

Meaning status_code can be either 401 or 302.

If you tried to translate this into code directly you would write something like this:

```
1 if status_code == 401 or 302:  
2     pass
```

However this is incorrect. This condition will be always true as this is actually same as if you wrote:

if (status_code == 401) or (302) so it will compare status_code to 401, and it will separately check if 302 is True, but any number different from 0 is considered to be True so the above expression will always be True.

What you probably meant is this:

```
1 if status_code == 401 or status_code == 302:  
2     pass
```

Alternative way:

An alternative way to achieve the same results would be though probbaly at this point we have not learned the “in” operator, nor lists (comma separated values in square brackets):

```
1 if status_code in [401, 302]  
2     pass
```

Exercise: compare numbers

- Ask the user to enter two numbers and tell us which one is bigger.

Exercise: compare strings

- Ask the user to enter two strings
- Then ask the user to select if she wants to compare them based on ASCII or based on their length
- Then tell us which one is bigger.

```
1 Input a string: (user types string and ENTER)
2 Input another string: (user types string and ENTER)
3 How to compare:
4 1) ASCII
5 2) Length
6 (user types 1 or 2 and ENTER)
```

Solution: compare numbers

```
1 a_in = input("Please type in a string: ")
2 b_in = input("Please type in another string: ")
3 print("How to compare:")
4 print("1) ASCII")
5 print("2) Length")
6 how = input()
7
8 if how == '1':
9     first = a_in > b_in
10    second = a_in < b_in
11 elif how == '2':
12     first = len(a_in) > len(b_in)
13     second = len(a_in) < len(b_in)
14
15 if first:
16     print("First number is bigger")
17 elif second:
18     print("First number is smaller")
19 else:
20     print("They are equal")
```

Solution: compare strings

```
1 a_in = input("Please type in a string: ")
2 b_in = input("Please type in another string: ")
3 print("How to compare:")
4 print("1) ASCII")
5 print("2) Length")
6 how = input()
7
8 if how == '1':
9     first = a_in > b_in
10    second = a_in < b_in
11 elif how == '2':
12     first = len(a_in) > len(b_in)
13     second = len(a_in) < len(b_in)
14
15 if first:
16     print("First number is bigger")
17 elif second:
18     print("First number is smaller")
19 else:
20     print("They are equal")
```

Strings

Single quoted and double quoted strings

In Python, just as in most of the programming languages you must put any free text inside a pair of quote characters. Otherwise Python will try to find meaning in the text.¹

These pieces of texts are called “strings”.

In Python you can put string between two single quotes: ‘’ or between two double quotes: “”. Which one does not matter.

```
1 soup = "Spiced carrot & lentil soup"
2 salad = 'Ceasar salad'
3
4 print(soup)
5 print(salad)
```

```
1 Spiced carrot & lentil soup
2 Ceasar salad
```

Long lines

```
1 text = "abc" "def"
2 print(text)
3
4 other = "abcdef"
5 print(other)
6
7
```

```
8 long_string = "one" "two" "three"
9 print(long_string)
10
11 short_rows = "one" \
12     "two" \
13     "three"
14 print(short_rows)
15
16 long_string = "first row second row third row"
17 print(long_string)
18
19 shorter = "first row \
20 second row \
21 third row"
22 print(shorter)
```

```
1 abcdef
2 abcdef
3 onetwothree
4 onetwothree
5 first row second row third row
6 first row second row third row
```

Triple quoted strings (multiline)

If you would like to create a string the spreads on multiple lines, there is a possibility to put the text between 3 quotes on both sides. Either 3 *single-quotes* or 3 double-quotes.

```
1 text = """first row
2 second row
3 third row"""
4
5 print(text)
```

Can spread multiple lines.

```
1 first row  
2 second row  
3 third row
```

String length (`len`)

The `len` function returns the length of the string in number of characters.

```
1 line = "Hello World"  
2 hw = len(line)  
3 print(hw)  # 11  
4  
5 text = """Hello  
6 World"""  
7 print(len(text))  # 12
```

String repetition and concatenation

You might be used to the fact the you can only multiple numbers, but in python you can also “multiply” a string by a number. It is called repetition. In this example we have a string “Jar “ that we repeat twice.repetition

We can also add two strings to concatenate them together.repetition

I don’t think the repetition operator is used very often, but in one case it could come very handy.

When you are writing some text report and you’d like to add a

long line of dashes that would be exactly the same length as your title.

```
1 name = 2 * 'Jar '
2 print(name)          # Jar Jar
3
4 full_name = name + 'Binks'
5 print(full_name)    # Jar Jar Binks
6
7
8 title = "We have some title"
9 print(title)
10 print('-' * len(title))
11
12 # We have some title
13 # -----
```

A character in a string

```
1 text = "Hello World"
2
3 a = text[0]
4 print(a)          # H
5
6 b = text[6]
7 print(b)          # W
```

String slice (instead of substr)

```
1 text = "Hello World"
2
3 b = text[1:4]
4 print(b)          # ell
5
6 print(text[2:])   # llo World
7 print(text[:2])   # He
8
9 start = 1
```

```
10 end = 4  
11 print(text[start:end]) # ell
```

Change a string

In Python strings are “immutable”, meaning you cannot change them. You can replace a whole string in a variable, but you cannot change it.

In the following example we wanted to replace the 3rd character (index 2), and put “Y” in place. This raised an exception

```
1 text = "abcd"  
2 print(text)      # abcd  
3  
4 text[2] = 'Y'  
5  
6 print("done")  
7 print(text)
```

```
1 abcd  
2 Traceback (most recent call last):  
3   File "string_change.py", line 4, in <module>  
4     text[2] = 'Y'  
5 TypeError: 'str' object does not support item assignment
```

Replace part of a string

- Strings in Python are **immutable** - they never change.

How to change a string

```
1 text = "abcd"
2 print(text)      # abcd
3
4 text = text[:2] + 'Y' + text[3:]
5 print(text)      # abYd
```

String copy

```
1 text = "abcd"
2 print(text)      # abcd
3
4 text = text + "ef"
5 print(text)      # abcdef
6
7 other = text
8 print(other)     # abcdef
9 text = "xyz"
10 print(text)     # xyz
11 print(other)    # abcdef
```

When assigning a variable pointing a string, the new variable is pointing to the same string..

If we then assign some other string to either of the variables, then they will point to two different strings.

String functions and methods (`len`, `upper`, `lower`)

```
1 a = "xYz"
2 print(len(a))      # 3
3
4 b = a.upper()
5 print(b)           # XYZ
```

```
6 print(a)          # xyz      - immutable!
7 print(a.lower())  # xyz
```

- Type `dir("")` in the REPL to get the list of string methods.
- List of [built-in functions](#).
- List of [string methods](#).

index in string

```
1 text = "The black cat climbed the green tree."
2 print(text.index("bl"))      # 4
3 print(text.index("The"))     # 0
4 print(text.index("dog"))
```

```
1 4
2 0
3 Traceback (most recent call last):
4   File "examples/strings/index.py", line 6, in <module>
5     print a.index("dog")      # -1
6 ValueError: substring not found
```

index in string with range

```
1 text = "The black cat climbed the green tree."
2 print(text.index("c"))      # 7
3 print(text.index("c", 8))    # 10
4
5 print(text.index("gr", 8))    # 26
6 print(text.index("gr", 8, 16))
```

```
1 7
2 10
3 26
4 Traceback (most recent call last):
5   File "examples/strings/index2.py", line 8, in
<module>
6     print a.index("gr", 8, 16)
7 ValueError: substring not found
```

rindex in string with range

```
1 text = "The black cat climbed the green tree."  
2 print(text.rindex("c"))          # 14  
3 print(text.rindex("c", 8))       # 14  
4 print(text.rindex("c", 8, 13))   # 10  
5  
6 print(text.rindex("gr", 8))      # 26  
7 print(text.rindex("gr", 8, 16))
```

```
1 14  
2 14  
3 10  
4 26  
5 Traceback (most recent call last):  
6   File "examples/strings/rindex.py", line 10, in  
<module>  
7     print(a.rindex("gr", 8, 16))  
8 ValueError: substring not found
```

find in string

Alternatively use find and rfind that will return -1 instead of raising an exception.

```
1 text = "The black cat climbed the green tree."  
2 print(text.find("bl"))          # 4  
3 print(text.find("The"))         # 0  
4 print(text.find("dog"))         # -1  
5  
6 print(text.find("c"))           # 7  
7 print(text.find("c", 8))        # 10  
8  
9 print(text.find("gr", 8))       # 26  
10 print(text.find("gr", 8, 16))    # -1  
11  
12 print(text.rfind("c", 8))      # 14
```

Find all in the string

Later, when we learned loops.

in string

Check if a substring is **in** the string?

```
1 txt = "hello world"
2 if "wo" in txt:
3     print('found wo')
4
5 if "x" in txt:
6     print("found x")
7 else:
8     print("NOT found x")
```

```
1 found wo
2 NOT found x
```

index if in string

```
1 sub = "cat"
2 txt = "The black cat climbed the green tree"
3
4 if sub in txt:
5     loc = txt.index(sub)
6     print(sub + " is at " + str(loc))
7
8 sub = "dog"
9 if sub in txt:
10    loc = txt.index(sub)
11    print(sub + " is at " + str(loc))
12
13 # cat is at 10
```

Encodings: ASCII, Windows-1255, Unicode

- [ASCII](#)
- [Hebrew Character](#)
- [Windows-1255](#)
- [Unicode \(UTF-8\)](#)

raw strings

```
1 # file_a = "c:\Users\Foobar\readme.txt"
2 # print(file_a)
3
4 # Python2: eadme.txtFoobar
5 # Python3:
6 #     File "examples/strings/raw.py", line 6
7 #         file_a = "c:\Users\Foobar\readme.txt"
8 #                     ^
9 # SyntaxError: (unicode error) 'unicodeescape' codec
10#     can't decode bytes in position 2-3: truncated
\XXXXXXXXXX escape
11
12
13 file_b = "c:\\Users\\\\Foobar\\\\readme.txt"
14 print(file_b)  # c:\\Users\\Foobar\\readme.txt
15
16 file_c = r"c:\\Users\\Foobar\\readme.txt"
17 print(file_c)  # c:\\Users\\Foobar\\readme.txt
18
19 text = r"text \n \d \s \ and more"
20 print(text)      # text \n \d \s \ and more
```

Escape sequences are kept intact and not escaped. Used in regexes.

ord

- [ord](#)

```
1 print( ord('a') )    # 97
2 print( ord('=') )    # 61
```

```
3 print( ord('\r') )    # 13
4 print( ord('\n') )    # 10
5 print( ord(' ') )    # 32
6
7 print( ord('á') )    # 225
8 print( ord('ó') )    # 243
9 print( ord('1488 #  ( ('`
```

ord in a file

```
1 import sys
2
3 filename = sys.argv[1]
4
5 with open(filename) as fh:
6     content = fh.read()
7
8 for c in content:
9     print(ord(c))
```

chr - number to character

- chr
-

```
1 for i in range(32, 126):
2     print( i, chr(i) )
```

```
1 32
2 33 !
3 34 "
4 35 #
5 36 $
6 37 %
7 38 &
8 39 '
9 40 (
10 41 )
11 42 *
12 43 +
```

13 44 ,
14 45 -
15 46 .
16 47 /
17 48 0
18 49 1
19 50 2
20 51 3
21 52 4
22 53 5
23 54 6
24 55 7
25 56 8
26 57 9
27 58 :
28 59 ;
29 60 <
30 61 =
31 62 >
32 63 ?
33 64 @
34 65 A
35 66 B
36 67 C
37 68 D
38 69 E
39 70 F
40 71 G
41 72 H
42 73 I
43 74 J
44 75 K
45 76 L
46 77 M
47 78 N
48 79 O
49 80 P
50 81 Q
51 82 R
52 83 S
53 84 T
54 85 U
55 86 V
56 87 W
57 88 X

```
58 89 Y
59 90 Z
60 91 [
61 92 \
62 93 ]
63 94 ^
64 95 -
65 96 \
66 97 a
67 98 b
68 99 c
69 100 d
70 101 e
71 102 f
72 103 g
73 104 h
74 105 i
75 106 j
76 107 k
77 108 l
78 109 m
79 110 n
80 111 o
81 112 p
82 113 q
83 114 r
84 115 s
85 116 t
86 117 u
87 118 v
88 119 w
89 120 x
90 121 y
91 122 z
92 123 {
93 124 |
94 125 }
```

Exercise: one string in another string

Write script that accepts two strings and tells if one of them can be found in the other and where?

Exercise: to ASCII CLI

Write script that gets a character on the command line and prints out the ascii code of it.

Maybe even:

Write script that gets a string on the command line and prints out the ascii code of each character.

Exercise: from ASCII CLI

Write script that accepts a number on the command line and prints the character represented by that number.

Solution: one string in another string

```
1 import sys
2
3 if len(sys.argv) != 3:
4     exit(f"Usage: {sys.argv[0]} short-STRING long-
STRING")
5
6 string = sys.argv[1]
7 text    = sys.argv[2]
8
9 if string in text:
10     loc = text.index(string)
11     print(string, "can be found in ", text, "at", loc)
12 else:
13     print(string, "can NOT be found in ", text)
```

Solution: compare strings

```
1 mode = input("Mode of comparision: [length|ascii|")
2 if mode != "length" and mode != "ascii":
3     print("Not good")
```

```
4     exit()
5
6 str1 = input("String 1:")
7 str1 = input("String 2:")
8
9 if mode == "length":
10    print(len(str1) > len(str2))
11 elif mode == "ascii":
12    print(str1 > str2)
```

Solution: to ASCII CLI

```
1 import sys
2
3 if len(sys.argv) != 2:
4     exit(f"Usage: {sys.argv[0]} CHARACTER")
5
6 print( ord( sys.argv[1]) )
```

```
1 import sys
2
3 if len(sys.argv) != 2:
4     exit(f"Usage: {sys.argv[0]} STRING")
5
6 for cr in sys.argv[1]:
7     print( ord( cr ) )
```

Solution: from ASCII CLI

```
1 import sys
2
3 if len(sys.argv) != 2:
4     exit(f"Usage: {sys.argv[0]} NUMBER")
5
6 print( chr( int(sys.argv[1]) ) )
```

Loops

Loops: for-in and while

- **for in** - to iterate over a well defined list of values.
(characters, range of numbers, shopping list, etc.)
- **while** - repeat an action till some condition is met. (or stopped being met)

for-in loop on strings

```
1 txt = 'hello world'  
2 for c in txt:  
3     print(c)
```

```
1 h  
2 e  
3 l  
4 l  
5 o  
6  
7 w  
8 o  
9 r  
10 l  
11 d
```

for-in loop on list

```
1 for fruit in ["Apple", "Banana", "Peach", "Orange",  
"Durian", "Papaya"]:  
2     print(fruit)
```

```
1 Apple
2 Banana
3 Peach
4 Orange
5 Durian
6 Papaya
```

for-in loop on range

```
1 for i in range(3, 7):
2     print(i)
```

```
1 3
2 4
3 5
4 6
```

Iterable, iterator

- [iterable](#)

for in loop with early end using break

```
1 txt = 'hello world'
2 for c in txt:
3     if c == ' ':
4         break
5     print(c)
```

```
1 h
2 e
3 l
4 l
5 o
```

for in loop skipping parts using continue

```
1 txt = 'hello world'
2 for c in txt:
3     if c == ' ':
4         continue
5     print(c)
```

```
1 h
2 e
3 l
4 l
5 o
6 w
7 o
8 r
9 l
10 d
```

for in loop with break and continue

```
1 txt = 'hello world'
2 for cr in txt:
3     if cr == ' ':
4         continue
5     if cr == 'r':
6         break
7     print(cr)
8 print('DONE')
```

```
1 h
2 e
3 l
4 l
5 o
6 w
7 o
8 DONE
```

while loop

```
1 import random
2
3 total = 0
4 while total <= 100:
5     print(total)
6     total += random.randrange(20)
7
8 print("done")
```

```
1 0
2 10
3 22
4 29
5 45
6 54
7 66
8 71
9 77
10 82
11 93
12 done
```

Infinite while loop

```
1 import random
2
3 total = 0
4 while total >= 0:
5     print(total)
6     total += random.randrange(20)
7
8 print("done")
```

```
1 ...
2 1304774
3 1304779
4 1304797
5 ^C1304803
6 Traceback (most recent call last):
7   File "while_infinite.py", line 5, in <module>
```

```
8     print(total)
9 KeyboardInterrupt
```

- Don't do this!
- Make sure there is a proper end-condition. (exit-condition)
- Use Ctrl-C to stop it

While with complex expression

```
1 import random
2
3 total = 0
4 while (total < 10000000) and (total % 17 != 1) and
5 (total ** 2 % 23 != 7):
6     print(total)
7     total += random.randrange(20)
8 print("done")
```

While with break

```
1 import random
2
3 total = 0
4 while total < 10000000:
5     print(total)
6     total += random.randrange(20)
7
8     if total % 17 == 1:
9         break
10
11    if total ** 2 % 23 == 7:
12        break
13
14 print("done")
```

While True

```
1 import random
2
3 total = 0
4 while True:
5     print(total)
6     total += random.randrange(20)
7
8     if total >= 10000000:
9         break
10
11    if total % 17 == 1:
12        break
13
14    if total ** 2 % 23 == 7:
15        break
16
17 print("done")
```

Duplicate input call

```
1 id_str = input("Type in your ID: ")
2
3 while len(id_str) != 9:
4     id_str = input("Type in your ID")
5
6 print("Your ID is " + id_str)
```

Eliminate duplicate input call

```
1 while True:
2     id_str = input("Type in your ID: ")
3     if len(id_str) == 9:
4         break
5
6 print("Your ID is " + id_str)
```

do while loop

There is no `do ... while` in Python but we can write code like this to have similar effect.

```
1 while True:  
2     answer = input("What is the meaning of life? ")  
3     if answer == '42':  
4         print("Yeeah, that's it!")  
5         break  
6  
7 print("done")
```

while with many continue calls

```
1 while True:  
2     line = get_next_line()  
3  
4     if last_line:  
5         break  
6  
7     if line_is_empty:  
8         continue  
9  
10    if line_has_an_hash_at_the_beginning: # #  
11        continue  
12  
13    if line_has_two_slashes_at_the_beginning: # //  
14        continue  
15  
16    do_the_real_stuff
```

Break out from multi-level loops

Not supported in Python. “If you feel the urge to do that, your code is probably too complex. create functions!”

Exit vs return vs break and continue

- `exit` will stop your program no matter where you call it.

- **return** will return from a function (it will stop the specific function only)
- **break** will stop the current “while” or “for” loop
- **continue** will stop the current iteration of the current “while” or “for” loop

Exercise: Print all the locations in a string

Given a string like “The black cat climbed the green tree.”, print out the location of every “c” character.

Exercise: Number guessing game

Level 0

- Using the random module the computer “thinks” about a whole number between 1 and 20.
- The user has to guess the number. After the user types in the guess the computer tells if this was bigger or smaller than the number it generated, or if was the same.
- The game ends after just one guess.

Level 1

- The user can guess several times. The game ends when the user guessed the right number.

Level 2

- If the user hits ‘x’, we leave the game without guessing the number.

Level 3

- If the user presses ‘s’, show the hidden value (cheat)

Level 4

- Soon we’ll have a level in which the hidden value changes after each guess. In order to make that mode easier to track and debug, first we would like to have a “debug mode”.
- If the user presses ‘d’ the game gets into “debug mode”: the system starts to show the current number to guess every time, just before asking the user for new input.
- Pressing ‘d’ again turns off debug mode. (It is a toggle each press on “d” changes the value to the other possible value.)

Level 5

- The ‘m’ button is another toggle. It is called ‘move mode’. When it is ‘on’, the hidden number changes a little bit after every step (+/-2). Pressing ‘m’ again will turn this feature off.

Level 6

- Let the user play several games.
- Pressing ‘n’ will skip this game and start a new one.
Generates a new number to guess.

Exercise: MasterMind

Implement the MasterMind game.

The computer “thinks” a number with 4 different digits.
You guess which digits. For every digit that matched both

in value, and in location the computer gives you a *. For every digit that matches in value, but not in space the computer gives you a +. Try to guess the given number in as few guesses as possible.

```
1 Computer: 2153
2 You:      2467  *
3 You:      2715  *++
```

Exercise: Count unique characters

Given a string on the command line, count how many different characters it has.

```
1 python count_unique.py abcdaaa
2 4
```

Solution: Print all the locations in a string

```
1 text = "The black cat climbed the green tree."
2 start = 0
3 while True:
4     loc = text.find("c", start)
5     if loc == -1:
6         break
7     print(loc)
8     start = loc + 1
```

Solution 1 for Number Guessing

```
1 import random
2
3 hidden = random.randrange(1, 201)
4 while True:
5     user_input = input("Please enter your guess[x]: ")
6     print(user_input)
7
```

```
8 if user_input == 'x':
9     print("Sad to see you leaving early")
10    exit()
11
12 guess = int(user_input)
13 if guess == hidden:
14     print("Hit!")
15     break
16
17 if guess < hidden:
18     print("Your guess is too low")
19 else:
20     print("Your guess is too high")
```

Solution for Number Guessing (debug)

```
1 import random
2
3 hidden = random.randrange(1, 201)
4 debug = False
5 while True:
6     if debug:
7         print("Debug: ", hidden)
8
9     user_input = input("Please enter your guess
[x|s|d]: ")
10    print(user_input)
11
12    if user_input == 'x':
13        print("Sad to see you leaving early")
14        exit()
15
16    if user_input == 's':
17        print("The hidden value is ", hidden)
18        continue
19
20    if user_input == 'd':
21        debug = not debug
22        continue
23
24    guess = int(user_input)
25    if guess == hidden:
```

```
26         print("Hit!")
27         break
28
29     if guess < hidden:
30         print("Your guess is too low")
31     else:
32         print("Your guess is too high")
```

Solution for Number Guessing (move)

```
1 import random
2
3 hidden = random.randrange(1, 201)
4 debug = False
5 move = False
6 while True:
7     if debug:
8         print("Debug: ", hidden)
9
10    if move:
11        mv = random.randrange(-2, 3)
12        hidden = hidden + mv
13
14    user_input = input("Please enter your guess
15 [x|s|d|m]: ")
16    print(user_input)
17
18    if user_input == 'x':
19        print("Sad to see you leaving early")
20        exit()
21
22    if user_input == 's':
23        print("The hidden value is ", hidden)
24        continue
25
26    if user_input == 'd':
27        debug = not debug
28        continue
29
30    if user_input == 'm':
31        move = not move
32        continue
```

```

32
33     guess = int(user_input)
34     if guess == hidden:
35         print("Hit!")
36         break
37
38     if guess < hidden:
39         print("Your guess is too low")
40     else:
41         print("Your guess is too high")

```

Solution for Number Guessing (multi-game)

```

1 import random
2
3 debug = False
4 move = False
5 while True:
6     print("\nWelcome to another Number Guessing game")
7     hidden = random.randrange(1, 201)
8     while True:
9         if debug:
10             print("Debug: ", hidden)
11
12         if move:
13             mv = random.randrange(-2, 3)
14             hidden = hidden + mv
15
16         user_input = input("Please enter your guess [x|s|d|m|n]: ")
17         print(user_input)
18
19         if user_input == 'x':
20             print("Sad to see you leaving early")
21             exit()
22
23         if user_input == 's':
24             print("The hidden value is ", hidden)
25             continue
26
27         if user_input == 'd':
28             debug = not debug

```

```

29         continue
30
31     if user_input == 'm':
32         move = not move
33     continue
34
35     if user_input == 'n':
36         print("Giving up, eh?")
37         break
38
39     guess = int(user_input)
40     if guess == hidden:
41         print("Hit!")
42         break
43
44     if guess < hidden:
45         print("Your guess is too low")
46     else:
47         print("Your guess is too high")

```

Solution: MasterMind

```

1 import random
2
3 width = 4
4 USED = '_'
5
6 hidden = random.sample(range(10), width)
7 # print(hidden)
8
9 while True:
10    # print(hidden)
11
12    inp = input("your guess ({} digits)".format(width))
13    if inp == 'x':
14        print("Bye")
15        exit()
16    if len(inp) != width:
17        print("We need exactly {} characters".format(width))
18        continue

```

```
19
20     guess = list(map(int, inp))
21     # print(guess)
22
23     if hidden == guess:
24         print("Match!")
25         break
26
27     my_hidden = hidden[:]
28     my_guess = guess[:]
29
30     result = ''
31     for i in range(width):
32         if my_hidden[i] == my_guess[i]:
33             result += '★'
34             my_hidden[i] = USED
35             my_guess[i] = USED
36     for i in range(width):
37         if my_guess[i] == USED:
38             continue
39         if my_guess[i] in my_hidden:
40             loc = my_hidden.index(my_guess[i])
41             my_hidden[loc] = USED
42             guess[i] = USED
43             result += '+'
44
45     print(''.join(result))
```

Solution: Count unique characters

```
1 import sys
2
3 s = sys.argv[1]
4
5 unique = ''
6 for c in s:
7     if c not in unique:
8         unique += c
9
10 print(len(unique))
```

```
1 import sys
2
3 s = sys.argv[1]
4
5 print(len(set(s)))
```

MasterMind to debug

Debug the following version of the MasterMind game.

```
1 import random
2
3
4 def number_generator():
5     y = [0, 0, 0, 0]
6
7     for i in range(0, 4):
8         y[i] = random.randrange(0, 10)
9         # print(y)
10        if i:
11            number += str(y[i])
12        else:
13            number = str(y[i])
14        # print(number)
15    return number
16
17
18 def user_input():
19     x = input("Type in 4 digits number:")
20     if len(x) == 4:
21         return x
22     else:
23         print("wrong input")
24         user_input()
25
26
27 def string_compare(x, y):
28     r = 0
29     q = 0
30     for i in range(0, 4):
31         if x[i] == y[i]:
32             r += 1
```

```
33         continue
34     for j in range(0, 4):
35         if x[i] == y[j]:
36             if i == j:
37                 continue
38             else:
39                 q += 1
40             break
41     return r, q
42
43
44 def print_result(r):
45     print("")
46     for i in range(0, r[0]):
47         print("*", end="")
48     for i in range(0, r[1]):
49         print("+", end="")
50     print("\n")
51
52
53 def main():
54     comp = number_generator()
55     result = 0
56     while True:
57         user = user_input()
58         result = string_compare(comp, user)
59         print_result(result)
60         # print(result)
61         if result[0] == 4:
62             print("Correct!")
63             return
64
65
66 main()
```

PyCharm

PyCharm Intro

- IDE
- Introspection
- Running, Debugging

PyCharm Project

- At the opening create a new project (directory + Python version)
- File/New Project

PyCharm Files

- New file on Mac: Click on the project on the left hand side / Right-Click / New / File; Windows, Linux: Alt-Insert
- PyCharm Python console - see next slide
- Change Python on Mac: PyCharm / Preferences / Project: (name) / Project Interpreter
- Later File/New also starts to work.

PyCharm - run code

- Run/Run
- Set command line parameters
- Set environment variables

- Run/Debug (but set breakpoints before)

PyCharm Python console at the bottom left

```
1 2 + 3
2 x = 2
3 print(x)
4 def f(x, y):
5     return x+y
6
7 f(4, 5)
```

Refactoring example (with and without pycharm)

- Change variable name (in scope only)
- Extract method

Formatted printing

format - sprintf

```
1 age = 42.12
2 name = 'Foo Bar'
3
4 str_concatenate = "The user " + name + " was born " +
str(age) + " years ago."
5 print(str_concatenate)
6
7 str_percentage = "The user %s was born %s years ago."
% (name, age)
8 print(str_percentage)
9
10 str_format = "The user {} was born {} years
ago.".format(name, age)
11 print(str_format)
12
13 str_f_string = f"The user {name} was born {age} years
ago."
14 print(str_f_string)
```

```
1 The user Foo Bar was born 42.12 years ago.
2 The user Foo Bar was born 42.12 years ago.
3 The user Foo Bar was born 42.12 years ago.
4 The user Foo Bar was born 42.12 years ago.
```

- When using % to print more than one values, put the values in parentheses forming a tuple.
- In version 2.6 and below you need to write etc, as a placeholder of the format method.
- f-string are from Python 3.6

Examples using format - indexing

```
1 txt = "Foo Bar"
2 num = 42.12
3
4 print("The user {} was born {} years ago.".format(txt,
num))
5 print("The user {0} was born {1} years
ago.".format(txt, num))
6 print("The user {1} was born {0} years
ago.".format(num, txt))
7
8
9 print("{} is {} and {} years old.".format(txt, num))
```

```
1 The user Foo Bar was born 42.12 years ago.
2 The user Foo Bar was born 42.12 years ago.
3 The user Foo Bar was born 42.12 years ago.
4 Foo Bar is Foo Bar and 42.12 years old.
```

Examples using format with names

```
1 txt = "Foo Bar"
2 num = 42.12
3
4 print("The user {name} was born {age} years
ago.".format(name = txt, age = num))
```

```
1 The user Foo Bar was born 42.12 years ago.
```

Format columns

In this example we use a list of lists that we have not learned yet, but don't worry about that for now.

Focus on the output of the two print statements.

```
1 data = [
2     ["Foo Bar", 42],
```

```
3     ["Bjorg", 12345],
4     ["Roza", 7],
5     ["Long Name Joe", 3],
6     ["Joe", 12345677889],
7 ]
8
9 for entry in data:
10    print("{} {}".format(entry[0], entry[1]))
11
12 print('-' * 16)
13
14 for entry in data:
15    print("{:<8}|{:>7}".format(entry[0], entry[1]))
```

```
1 Foo Bar 42
2 Bjorg 12345
3 Roza 7
4 Long Name Joe 3
5 Joe 12345677889
6 -----
7 Foo Bar | 42
8 Bjorg | 12345
9 Roza | 7
10 Long Name Joe | 3
11 Joe | 12345677889
```

Examples using format - alignment

```
1 txt = "Some text"
2
3 print("'{ }'".format(txt))      # as is:      'Some text'
4 print("'{:12}'".format(txt))   # left:       'Some text
'
5 print("'{:<12}'".format(txt))  # left:       'Some text
'
6 print("'{:>12}'".format(txt))  # right:      '      Some
text'
7 print("'{:^12}'".format(txt))   # center:     ' Some text
'
```

Format - string

```
1 name = "Foo Bar"  
2  
3 print("{:s}".format(name))  
4 print("{}".format(name))
```

```
1 Foo Bar  
2 Foo Bar
```

Format characters and types

```
1 x = 42  
2  
3 print("{:b}".format(x)) # binary:      101010  
4 print("{:c}".format(x)) # character:   *  
5 print("{:d}".format(x)) # decimal:     42  
(default)  
6 print("{:o}".format(x)) # octal:       52  
7 print("{:x}".format(x)) # hexa:        2a  
8 print("{:X}".format(x)) # hexa:        2A  
9 print("{:n}".format(x)) # number:      42  
10  
11  
12 print("{}".format(x)) # defaults to decimal
```

Format floating point number

```
1 x = 412.345678901  
2  
3 print("{:e}".format(x))    # exponent:  
4.123457e+02  
4 print("{:E}".format(x))    # Exponent:  
4.123457E+02  
5 print("{:f}".format(x))    # fixed point:  412.345679  
(default precision is 6)  
6 print("{:.2f}".format(x))  # fixed point:  412.35 (set  
precision to 2)  
7 print("{:F}".format(x))    # same as f.      412.345679  
8 print("{:g}".format(x))    # generic:       412.346
```

```
(default precision is 6)
 9 print("{:G}".format(x))      # generic:          412.346
10 print("{:n}".format(x))      # number:           4412.346
11
12
13 print("{}".format(x))       # defaults to g
412.345678901
```

f-strings (formatted string literals)

Since Python 3.6

```
1 name = "Foo Bar"
2 age = 42.12
3 pi = 3.141592653589793
4 r = 2
5
6 print(f"The user {name} was born {age} years ago.")
7 print(f"The user {name:10} was born {age} years ago.")
8 print(f"The user {name:>10} was born {age} years
ago.")
9 print(f"The user {name:>10} was born {age:>10} years
ago.")
10
11 print(f"PI is '{pi:.3}'.")    # number of digits
(defaults n = number)
12 print(f"PI is '{pi:.3f}'.")   # number of digits after
decimal point
13
14 print(f"Area is {pi * r ** 2}")
15 print(f"Area is {pi * r ** 2:.3f}")
```

```
1 The user Foo Bar was born 42.12 years ago.
2 The user Foo Bar      was born 42.12 years ago.
3 The user      Foo Bar was born 42.12 years ago.
4 The user      Foo Bar was born          42.12 years ago.
5 PI is '3.14'.
6 PI is '3.142'.
7 Area is 12.566370614359172
8 Area is 12.566
```

printf using old %-syntax

This slides is here only as a historical page. It is recommended to use the **format** method!

```
1 v = 65
2 print("<%s>" % v)      # <65>
3 print("<%10s>" % v)    # <       65>
4 print("<%-10s>" % v)   # <65       >
5 print("<%c>" % v)      # <A>
6 print("<%d>" % v)      # <65>
7 print("<%0.5d>" % v)   # <00065>
```

Format braces, bracket, and parentheses

These are just some extreme special cases. Most people won't need to know about them.

To print { include {{.

To print } include }}.

```
1 print("{ {{ }} }".format(42))    # { 42 }
2
3 print("{ {  } }".format(42))     # { 42 }
4
5 print("[ { } ] ({ })".format(42, 42))  # [42] (42)
6
7 print("%{ }".format(42))        # %42
```

Anything that is not in curly braces will be formatted as they are.

Examples using format with attributes of objects

This is also a rather strange example, I don't think I'd use it in real code.

```
1 import sys
2
3 print("{0.executable}".format(sys))
4 print("{system.argv[0]}".format(system = sys))
```

```
1 /home/gabor/venv3/bin/python
2 formatted_attributes.py
```

raw f-strings

```
1 name="foo"
2 print(r"a\nb {name}")
3 print(rf"a\nb {name}")
4 print(fr"a\nb {name}") # this is better (for vim)
```

```
1 a\nb {name}
2 a\nb foo
3 a\nb foo
```

Lists

Anything can be a lists

- Comma separated values
- In square brackets
- Can be any value, and a mix of values: Integer, Float, Boolean, None, String, List, Dictionary, ...
- But usually they are of the same type:
- Distances of astronomical objects
- Chemical Formulas
- Filenames
- Names of devices
- Objects describing attributes of a network device.
- Actions to do on your data.

```
1 stuff = [42, 3.14, True, None, "Foo Bar", ['another',  
'list'], {'a': 'Dictionary', '\\\nlanguage' : 'Python'}]  
2 print(stuff)
```

```
1 [42, 3.14, True, None, 'Foo Bar', ['another', 'list'],  
{'a': 'Dictionary', 'language\\\n': 'Python'}]
```

Any layout

- Layout is flexible
- Trailing comma is optional. It does not disturb us. Nor Python.

```
1 more_stuff = [
2     42,
3     3.14,
4     True,
5     None,
6     "Foo Bar",
7     ['another', 'list'],
8     {
9         'a': 'Dictionary',
10        'language' : 'Python',
11    },
12 ]
13 print(more_stuff)
```

```
1 [42, 3.14, True, None, 'Foo Bar', ['another', 'list'],
2 {'a': 'Dictionary', 'language\
2 ': 'Python'}]
```

Lists

- Access single element: **[index]**
- Access a sublist: **[start:end]**
- Creates a copy of that sublist

```
1 planets = ['Mercury', 'Venus', 'Earth', 'Mars',
'Jupiter', 'Saturn']
2
3 print(planets)    # ['Mercury', 'Venus', 'Earth',
'Mars', 'Jupiter', 'Saturn']
4 print(len(planets))    # 6
5
6 print(planets[0])      # Mercury
7 print(type(planets[0]))    # <class 'str'>
8 print(planets[3])      # Mars
9
10 print(planets[0:1])     # ['Mercury']
11 print(type(planets[0:1]))    # <class 'list'>
12 print(planets[0:2])     # ['Mercury', 'Venus']
13 print(planets[1:3])     # ['Venus', 'Earth']
```

```
15 print(planets[2:])          # ['Earth', 'Mars',
'Jupiter', 'Saturn']
16 print(planets[:3])          # ['Mercury', 'Venus',
'Earth']
17
18 print(planets[:])           # ['Mercury', 'Venus',
'Earth', 'Mars', 'Jupiter', 'Saturn\
19 ']
```

List slice with steps

- List slice with step: **[start:end:step]**

```
1 letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h',
'i', 'j']
2
3 print(letters[:::])          # ['a', 'b', 'c', 'd', 'e',
'f', 'g', 'h', 'i', 'j']
4
5 print(letters[::-1])          # ['a', 'b', 'c', 'd', 'e',
'f', 'g', 'h', 'i', 'j']
6
7 print(letters[::2])           # ['a', 'c', 'e', 'g', 'i']
8
9 print(letters[1::2])           # ['b', 'd', 'f', 'h', 'j']
10
11 print(letters[2:8:2])          # ['c', 'e', 'g']
12
13 print(letters[1:20:3])          # ['b', 'e', 'h']
```

Change a List

```
1 x = ['abc', 'def', 'ghi', 'jkl']
2 x[0] = 'qqrq'
3 print(x)      # ['qqrq', 'def', 'ghi', 'jkl']
4
5 x[1:3] = ['xyz', 'dod']
6 print(x)      # ['qqrq', 'xyz', 'dod', 'jkl']
7
8
9 x[1:3] = ['bla']
```

```
10 print(x)      #  ['qqrq', 'bla', 'jkl']
11
12 x[1:2] = ['elp', 'free']
13 print(x)      # ['qqrq', 'elp', 'free', 'jkl']
14
15
16 #x[1] = ['elp', 'free']
17 #print(x)    # ['qqrq', ['elp', 'free'], 'jkl']
```

- Unlike strings, lists are mutable. You can change the content of a list by assigning values to its elements.
- You can use the slice notation to change several elements at once.
- You can even have different number of elements in the slice and in the replacement. This will also change the length of the array.

Change with steps

```
1 numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
2 print(numbers)  # [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
12]
3
4 numbers[1::2] = [0, 0, 0, 0, 0, 0]
5 print(numbers)  # [1, 0, 3, 0, 5, 0, 7, 0, 9, 0, 11, 0]
```

List assignment and list copy

```
1 x = ['apple', 'bob', 'cat', 'drone']
2 y = x
3 x[0] = 'qqrq'
4 print(x)      # ['qqrq', 'bob', 'cat', 'drone']
5 print(y)      # ['qqrq', 'bob', 'cat', 'drone']
```

- There is one list in the memory and two pointers to it.

- If you really want to make a copy the pythonic way is to use the slice syntax.
 - It creates a shallow copy.
-

```
1 x = ['apple', 'bob', 'cat', 'drone']
2 y = x[:]
3
4 x[0] = 'qqrq'
5
6 print(x)      # ['qqrq', 'bob', 'cat', 'drone']
7 print(y)      # ['apple', 'bob', 'cat', 'drone']
```

Deep copy

```
1 from copy import deepcopy
2
3 x = ['apple', 'bob', 'cat', 'drone']
4 y = deepcopy(x)
5
6 x[0] = 'qqrq'
7
8 print(x)      # ['qqrq', 'bob', 'cat', 'drone']
9 print(y)      # ['apple', 'bob', 'cat', 'drone']
```

join

```
1 fields = ['one', 'two and three', 'four', 'five']
2
3 together = ':'.join(fields)
4 print(together) # one:two and three:four:five
5
6 mixed = ' -=> '.join(fields)
7 print(mixed) # one -=> two and three -=> four -=>
5ive
8
9 another = ''.join(fields)
10 print(another) # onetwo and threefourfive
```

join list of numbers

```
1 a = ["x", "2", "y"]
2 b = ["x", 2, "y"]
3 print(":".join(a))      # x:2:y
4 # print ":".join(b)      # TypeError: sequence item 1:
expected string, int found
5
6 # convert elements to string using map
7 print(":.".join( map(str, b) ))          # x:2:y
8
9
10 # convert elements to string using list comprehension
11 print(":.".join( str(x) for x in b ))   # x:2:y
```

split

- Special case: To split a string to its characters: Use the **list()** function.
- Split using more than one splitter: use **re.split**

```
1 words = "ab:cd:ef".split(':')
2 print(words)    # ['ab', 'cd', 'ef']
3
4 # special case: split by spaces
5 names = "foo    bar baz".split()
6 print(names)    # ['foo', 'bar', 'baz']
7
8 # special case: split to characters
9 chars = list("abcd")
10 print(chars)   # ['a', 'b', 'c', 'd']
```

for loop on lists

```
1 things = ['apple', 'banana', 'peach', 42]
2 for var in things:
3     print(var)
```

```
1 apple
2 banana
3 peach
4 42
```

in list

Check if the value is in the list?

```
1 words = ['apple', 'banana', 'peach', '42']
2 if 'apple' in words:
3     print('found apple')
4
5 if 'a' in words:
6     print('found a')
7 else:
8     print('NOT found a')
9
10 if 42 in words:
11     print('found 42')
12 else:
13     print('NOT found 42')
14
15 # found apple
16 # NOT found a
17 # NOT found 42
```

Where is the element in the list

```
1 words = ['cat', 'dog', 'snake', 'camel']
2 print(words.index('snake'))
3
4 print(words.index('python'))
```

```
1 2
2 Traceback (most recent call last):
3   File "examples/lists/index.py", line 6, in <module>
4       print(words.index('python'))
5 ValueError: 'python' is not in list
```

Index improved

```
1 words = ['cat', 'dog', 'snake', 'camel']
2
3 name = 'snake'
4 if name in words:
5     print(words.index(name))
6
7 name = 'python'
8 if name in words:
9     print(words.index(name))
```

[] .insert

```
1 words = ['apple', 'banana', 'cat']
2 print(words) # ['apple', 'banana', 'cat']
3
4 words.insert(2, 'zebra')
5 print(words) # ['apple', 'banana', 'zebra', 'cat']
6
7 words.insert(0, 'dog')
8 print(words) # ['dog', 'apple', 'banana', 'zebra',
'cat']
9
10 # Instead of this, use append (next slide)
11 words.insert(len(words), 'olifant')
12 print(words) # ['dog', 'apple', 'banana', 'zebra',
'cat', 'olifant']
```

[] .append

```
1 names = ['Foo', 'Bar', 'Zorg', 'Bambi']
2 print(names) # ['Foo', 'Bar', 'Zorg', 'Bambi']
3
4 names.append('Qux')
5 print(names) # ['Foo', 'Bar', 'Zorg', 'Bambi', 'Qux']
```

[] .remove

```
1 names = ['Joe', 'Kim', 'Jane', 'Bob', 'Kim']
2 print(names)                      # ['Joe', 'Kim', 'Jane',
'Bob', 'Kim']
3
4 print(names.remove('Kim'))        # None
5 print(names)                      # ['Joe', 'Jane', 'Bob',
'Kim']
6
7 print(names.remove('George'))
8     # Traceback (most recent call last):
9     #   File "examples/lists/remove.py", line 9, in
<module>
10    #       print(names.remove('George')) # None
11    # ValueError: list.remove(x): x not in list
```

Remove **first** element from a list given by its value.
Throws an exception if there is no such element in the list.

Remove element by index [] .pop

```
1 planets = ['Mercury', 'Venus', 'Earth', 'Mars',
'Jupiter']
2 print(planets)                  # ['Mercury', 'Venus',
'Earth', 'Mars', 'Jupiter']
3
4 third = planets.pop(2)
5 print(third)                   # Earth
6 print(planets)                  # ['Mercury', 'Venus', 'Mars',
'Jupiter']
7
8 last = planets.pop()
9 print(last)                     # Jupiter
10 print(planets)                 # ['Mercury', 'Venus', 'Mars']
11
12 # planets.pop(4)               # IndexError: pop index out
of range
13
14 jupyter_landers = []
```

```
15 # jupyter_landers.pop()      # IndexError: pop from empty
list
```

Remove and return the last element of a list. Throws an exception if the list was empty.

Remove first element of list

To remove an element by its index, use the slice syntax:

```
1 names = ['foo', 'bar', 'baz', 'moo']
2
3 first = names.pop(0)
4 print(first)      # foo
5 print(names)      # ['bar', 'baz', 'moo']
```

Remove several elements of list by index

To remove an element by its index, use the slice syntax:

```
1 names = ['foo', 'bar', 'baz', 'moo', 'qux']
2
3 names[2:4] = []
4 print(names)      # ['foo', 'bar', 'qux']
```

Use list as a queue

```
1 a_queue = []
2 print(a_queue)
3
4 a_queue.append('Moo')
5 print(a_queue)
6
7 a_queue.append('Bar')
8 print(a_queue)
9
10 first = a_queue.pop(0)
11 print(first)
12 print(a_queue)
```

```
1 []
2 ['Moo']
3 ['Moo', 'Bar']
4 Moo
5 ['Bar']
```

Queue using deque from collections

```
1 from collections import deque
2
3 # items = deque([])
4 items = deque(['foo', 'bar'])
5
6 print(type(items))    # <type 'collections.deque'>
7 print(items)           # deque(['foo', 'bar'])
8
9 items.append('zorg')
10 print(items)          # deque(['foo', 'bar', 'zorg'])
11 print(len(items))    # 3
12
13 items.append('zorg')
14 print(items)          # deque(['foo', 'bar', 'zorg',
15 'zorg'])
16 nxt = items.popleft()
17 print(nxt)            # 'foo'
18 print(items)          # deque(['bar', 'zorg', 'zorg'])
19
```

```
20 print(len(items))      # 3
21
22 if items:
23     print("The queue has items")
24 else:
25     print("The queue is empty")
```

- .append
- .popleft
- len() number of elements
- if q: to see if it has elements or if it is empty
- dequeue

Fixed size queue

```
1 from collections import deque
2
3 queue = deque([], maxlen = 3)
4 print(len(queue))      # 0
5 print(queue maxlen)    # 3
6
7 queue.append("Foo")
8 queue.append("Bar")
9 queue.append("Baz")
10 print(queue)           # deque(['Foo', 'Bar', 'Baz'],
maxlen=3)
11
12 queue.append("Zorg")   # Automatically removes the
left-most (first) element
13 print(queue)           # deque(['Bar', 'Baz', 'Zorg'],
maxlen=3)
```

List as a stack

```
1 stack = []
2
3 stack.append("Joe")
4 print(stack)
```

```
5 stack.append("Jane")
6 print(stack)
7 stack.append("Bob")
8 print(stack)
9
10 while stack:
11     name = stack.pop()
12     print(name)
13     print(stack)
```

```
1 ['Joe']
2 ['Joe', 'Jane']
3 ['Joe', 'Jane', 'Bob']
4 Bob
5 ['Joe', 'Jane']
6 Jane
7 ['Joe']
8 Joe
9 []
```

stack with deque

```
1 from collections import deque
2 stack = deque()
3
4 stack.append("Joe")
5 stack.append("Jane")
6 stack.append("Bob")
7
8 while stack:
9     name = stack.pop()
10    print(name)
11
12 # Bob
13 # Jane
14 # Joe
```

Exercises: Queue

The application should manage a queue of people.

- It will prompt the user for a new name by printing :, the user can type in a name and press ENTER. The app will add the name to the queue.
 - If the user types in “n” then the application will remove the first name from the queue and print it.
 - If the user types in “x” then the application will print the list of users who were left in the queue and it will exit.
 - If the user types in “s” then the application will show the current number of elements in the queue.
-

```
1 : Foo
2 : Bar
3 : Moo
4 : n
5   next is Foo
6 : n
7   next is Bar
8 : Peter
9 : n
10  next is Moo
11 : n
12  next is Peter
13 : n
14  the queue is empty
```

Exercise: Stack

Implement a Reverse Polish Calculator

```
1 2
2 3
3 4
4 +
5 *
6 =
7 14
```

```
1 x = eXit, s = Show, [+-*/=]
2 :23
3 :19
4 :7
5 :8
6 :+
7 :3
8 :-.
9 :/
10 :s
11 [23.0, -0.631578947368421]
12 :+
13 :=
14 22.36842105263158
15 :s
16 []
17 :x
```

Solution: Queue with list

```
1 queue = []
2
3 while True:
4     inp = input(":")
5     inp = inp.rstrip("\n")
6
7     if inp == 'x':
8         for name in queue:
9             print(name)
10        exit()
11
12    if inp == 's':
13        print(len(queue))
14        continue
15
16    if inp == 'n':
17        if len(queue) > 0:
18            print("next is {}".format(queue.pop(0)))
19        else:
20            print("the queue is empty")
21        continue
22
23    queue.append(inp)
```

Solution: Queue with deque

```
1 from collections import deque
2
3 queue = deque()
4
5 while True:
6     inp = input(":")
7     inp = inp.rstrip("\n")
8
9     if inp == 'x':
10         for name in queue:
11             print(name)
12         exit()
13
14     if inp == 's':
15         print(len(queue))
16         continue
17
18     if inp == 'n':
19         if len(queue) > 0:
20             print("next is
{}".format(queue.popleft()))
21         else:
22             print("the queue is empty")
23         continue
24
25     queue.append(inp)
```

Solution: Reverse Polish calculator (stack) with lists

```
1 stack = []
2
3 print("x = eXit, s = Show, [+*/=] ")
4 while True:
5     val = input(':')
6
7     if val == 's':
```

```

8         print(stack)
9         continue
10
11    if val == 'x':
12        break
13
14    if val == '+':
15        a = stack.pop()
16        b = stack.pop()
17        stack.append(a+b)
18        continue
19
20    if val == '-':
21        a = stack.pop()
22        b = stack.pop()
23        stack.append(a-b)
24        continue
25
26    if val == '*':
27        a = stack.pop()
28        b = stack.pop()
29        stack.append(a*b)
30        continue
31
32    if val == '/':
33        a = stack.pop()
34        b = stack.pop()
35        stack.append(a/b)
36        continue
37
38    if val == '=':
39        print(stack.pop())
40        continue
41
42    stack.append(float(val))

```

Solution: Reverse Polish calculator (stack) with deque

```

1 from collections import deque
2
3 stack = deque()

```

```
4
5 while True:
6     val = input(':')
7
8     if val == 'x':
9         break
10
11    if val == '+':
12        a = stack.pop()
13        b = stack.pop()
14        stack.append(a+b)
15        continue
16
17    if val == '*':
18        a = stack.pop()
19        b = stack.pop()
20        stack.append(a*b)
21        continue
22
23
24    if val == '=':
25        print(stack.pop())
26        continue
27
28    stack.append(float(val))
```

Debugging Queue

The following implementation has a bug. (Even though the `n` was supposed to remove the element and the code seems to mean that it does, we still see two items after we removed the first.)

The question is how to debug this?

```
1 q = []
2
3 while True:
4     name=input("your name: ")
5
6     if name=="n":
```

```
7     print(q.pop(0))
8
9     if name=="x":
10        print(q)
11        exit()
12
13    if name=="s":
14        print(len(q))
15        exit()
16    else:
17        q.append(name)
18        continue
```

```
1 your name: Foo
2 your name: Bar
3 your name: n
4 Foo
5 your name: s
6 2
```

sort

```
1 planets = ['Mercury', 'Venus', 'Earth', 'Mars',
2 'Jupiter', 'Saturn']
3 print(planets)      # ['Mercury', 'Venus', 'Earth',
4 'Mars', 'Jupiter', 'Saturn']
5 planets.sort()
6 print(planets)      # ['Earth', 'Jupiter', 'Mars',
7 'Mercury', 'Saturn', 'Venus']
8
9 planets.sort(reverse=True)
10 print(planets)     # ['Venus', 'Saturn', 'Mercury',
11 'Mars', 'Jupiter', 'Earth']
```

sort numbers

```
4 print(numbers)                      # [-4, 2, 7, 8,
19]
5
6 numbers.sort(reverse=True)
7 print(numbers)                      # [19, 9, 7, 2,
-4]
8
9 numbers.sort(key=abs, reverse=True)
10 print(numbers)                     # [19, 9, 7, -4,
2]
```

sort mixed

```
1 mixed = [100, 'foo', 42, 'bar']
2 print(mixed)
3 mixed.sort()
4 print(mixed)
```

In Python 2 puts the numbers first in numerical order and then the strings in ASCII order.

```
1 [100, 'foo', 42, 'bar']
2 [42, 100, 'bar', 'foo']
```

In Python 3 it throws an exception.

```
1 [100, 'foo', 42, 'bar']
2 Traceback (most recent call last):
3   File "examples/lists/sort_mixed.py", line 5, in
<module>
4     mixed.sort()
5 TypeError: unorderable types: str() < int()
```

key sort

- Another example to using a **key**.
- To sort the list according to length

```
1 animals = ['chicken', 'cow', 'snail', 'elephant']
2 print(animals)
3
4 animals.sort()
5 print(animals)
6
7 animals.sort(key=len)
8 print(animals)
9
10 animals.sort(key=len, reverse=True)
11 print(animals)
```

```
1 ['chicken', 'cow', 'snail', 'elephant']
2 ['chicken', 'cow', 'elephant', 'snail']
3 ['cow', 'snail', 'chicken', 'elephant']
4 ['elephant', 'chicken', 'snail', 'cow']
```

Sort tuples

Sorting tuples or list, or other complex structures

```
1 students = [
2     ('John', 'A', 2),
3     ('Zoro', 'C', 1),
4     ('Dave', 'B', 3),
5 ]
6 print(students)
7 # [ ('John', 'A', 2), ('Zoro', 'C', 1), ('Dave', 'B',
8 3) ]
9 print(sorted(students))
10 # [ ('Dave', 'B', 3), ('John', 'A', 2), ('Zoro', 'C',
11 1) ]
12 # sort by the first element of each tuple
13 print(sorted(students, key=lambda s : s[1]))
14 # [ ('John', 'A', 2), ('Dave', 'B', 3), ('Zoro', 'C',
15 1) ]
16 # sort by the 2nd element of the tuples (index 1)
17 print(sorted(students, key=lambda s : s[2]))
```

```
18     # [('Zoro', 'C', 1), ('John', 'A', 2), ('Dave', 'B',
3)]
19     # sort by the 3rd element of the tuples (index 2)
20
21
22 from operator import itemgetter
23 print(sorted(students, key=itemgetter(2)))
24     # [('Zoro', 'C', 1), ('John', 'A', 2), ('Dave', 'B',
3)]
25     # maybe this is more simple than the lambda version
26     # and probably faster
```

sort with sorted

```
1 animals = ['chicken', 'cow', 'snail', 'elephant']
2 print(animals)          # ['chicken', 'cow', 'snail',
'elephant']
3
4 s = sorted(animals)
5 print(s)                # ['chicken', 'cow',
'elephant', 'snail']
6 print(animals)          # ['chicken', 'cow', 'snail',
'elephant']
7
8 r = sorted(animals, reverse=True, key=len)
9 print(r)                # ['elephant', 'chicken',
'snail', 'cow']
10 print(animals)         # ['chicken', 'cow', 'snail',
'elephant']
```

sort vs. sorted

The `sort()` method will sort a list in-place and return `None`.
The built-in `sorted()` function will return the sorted list and leave the original list intact.

key sort with sorted

To sort the list according to length using `sorted`

```
1 animals = ['snail', 'cow', 'elephant', 'chicken']
2 animals_in_abc = sorted(animals)
3
4 print(animals)
5 print(animals_in_abc)
6
7 animals_by_length = sorted(animals, key=len)
8 print(animals_by_length)
```

```
1 ['snail', 'cow', 'elephant', 'chicken']
2 ['chicken', 'cow', 'elephant', 'snail']
3 ['cow', 'snail', 'chicken', 'elephant']
```

Sorting characters of a string

```
1 letters = 'axzb'
2 print(letters)          # 'axzb'
3 s = sorted(letters)
4 print(s)                # ['a', 'b', 'x', 'z']
5 print(letters)          # 'axzb'
6
7 r = ''.join(sorted(letters))
8 print(r)                # abxz
```

range

```
1 for i in range(11, 18, 2):
2     print(i)
3 # 11
4 # 13
5 # 15
6 # 17
7
8 for i in range(5, 7):
9     print(i)
10 # 5
11 # 6
12
13 for i in range(3):
14     print(i)
```

```
15 # 0  
16 # 1  
17 # 2
```

Looping over index

```
1 things = ['abc', 'def', 'ghi', 42]  
2 for var in things:  
3     print(var)
```

```
1 things = ['abc', 'def', 'ghi', 42]  
2 for i in range(len(things)):  
3     print(i, things[i])  
4  
5 # 0 abc  
6 # 1 def  
7 # 2 ghi  
8 # 3 42
```

Enumerate lists

```
1 planets = ['Mercury', 'Venus', 'Earth', 'Mars',  
'Jupiter', 'Saturn']  
2 for idx, planet in enumerate(planets):  
3     print(idx, planet)  
4  
5 print()  
6 enu = enumerate(planets)  
7 print(enu.__class__.__name__)  
8 print(enu)
```

```
1 0 Mercury  
2 1 Venus  
3 2 Earth  
4 3 Mars  
5 4 Jupiter  
6 5 Saturn  
7
```

```
8 enumerate
9 <enumerate object at 0x7f2c2402adc8>
```

List operators

```
1 a = ['one', 'two']
2 b = ['three']
3
4 print(a)      # ['one', 'two']
5 print(a * 2)  # ['one', 'two', 'one', 'two']
6 print(a + b)  # ['one', 'two', 'three']
```

List of lists

```
1 x = ['abc', 'def']
2 print(x)      # ['abc', 'def']
3
4 y = [x, 'xyz']
5 print(y)      # [[['abc', 'def']], 'xyz']
6 print(y[0])   # ['abc', 'def']
7
8 print(x[0])   # abc
9 print(y[0][0]) # abc
```

List assignment

List assignment works in “parallel” in Python.

```
1 x, y = 1, 2
2 print(x)      # 1
3 print(y)      # 2
4
5 x, y = y, x
6 print(x)      # 2
7 print(y)      # 1
```

```
1 x, y = f()  # works if f returns a list of 2 elements
```

It will throw a run-time `ValueError` exception if the number of values in the returned list is not 2. (Both for fewer and for more return values).

List documentation

- [datastructures](#)

tuple

Tuple

- A tuple is a fixed-length immutable list. It cannot change its size or content.
- A tuple is denoted with parentheses: (1,2,3)

```
1 t = ('a', 'b', 'c')
2 print(t)  # ('a', 'b', 'c')
```

List

- Elements of a list can be changed via their index or via the list slice notation.
- A list can grow and shrink using **append** and **pop** methods or using the **slice** notation.
- A list is denoted with square brackets: [1, 2, 3]

```
1 l = ['abc', 'def', 'qqrq']
2 t = tuple(l)
3 print(l)  # ['abc', 'def', 'qqrq']
4 print(t)  # ('abc', 'def', 'qqrq')
```

Tuples are rarely used. There are certain places where Python or some module require tuple (instead of list) or return a tuple (instead of a list)

and in each place it will be explained. Otherwise you don't need to use tuples.

e.g. keys of dictionaries can be tuple (but not lists).

Exercise: color selector menu

- In a script have a list of colors. Write a script that will display a menu (a list of numbers and the corresponding color) and prompts the user for a number. The user needs to type in one of the numbers. That's the selected color.
 1. blue
 2. green
 3. yellow
 4. white
- For extra credit make sure the system is user-proof and it won't blow up on various incorrect input values. (e.g Floating point number. Number that is out of range, non-number)
- For more credit allow the user to supply the number of the color on the command line. **python color.py 3**. If that is available, don't prompt.
- For further credit allow the user to provide the name of the color on the command line: **python color.py yellow** Can you handle color names that are not in the expected case (e.g. Yellow)?
- Any more ideas for improvement?

Exercise: count digits

Given a list of numbers `numbers = [1203, 1256, 312456, 98]`, count how many times each digit appears? The output will look like this:

```
1 0 1
2 1 3
3 2 3
4 3 2
5 4 1
6 5 2
7 6 2
8 7 0
9 8 1
10 9 1
```

Exercise: Create list

Given a list of strings with words separated by spaces, create a single list of all the words.

```
1 lines = [
2     'grape banana mango',
3     'nut orange peach',
4     'apple nut banana apple mango',
5 ]
6
7 fruits = ['grape', 'banana', 'mango', 'nut', 'orange',
8 'peach', 'apple', 'nut', 'banana',
9 'apple', 'mango']
```

Then create a list of unique values sorted by abc.

```
1 unique_fruites = ['apple', 'banana', 'grape', 'mango',
2 'nut', 'orange', 'peach']
```

Exercise: Count words

```
1 celestial_objects = [  
2     'Moon', 'Gas', 'Asteroid', 'Dwarf', 'Asteroid',  
'Moon', 'Asteroid'  
3 ]
```

Expected output:

```
1 Moon      2  
2 Gas       1  
3 Asteroid  3  
4 Dwarf     1
```

Exercise: Check if number is prime

Write a program that gets a number on the command line and prints “True” if the number is a prime number or “False” if it isn’t.

```
1 python is_prime.py 42  
2 False  
3 python is_prime.py 19  
4 True
```

Exercise: DNA sequencing

- A, C, T, G are called bases or nucleotides
- Given a sequence like ‘ACCGXXCXXGTTACTGGGCXTTGT’ (nucleoids mixed up with other elements) return the sequences containing only ACTG ordered by length.
- The above string can be split up to [‘ACCG’, ‘C’, ‘GTTACTGGGC’, ‘TTGT’] and then it can be sorted to get the following:

- Expected result: ['GTTACTGGGC', 'ACCG', 'TTGT', 'C']

Solution: menu

```

1 colors = ['blue', 'yellow', 'black', 'purple']
2 for ix in range(len(colors)):
3     print("{} {}) {}".format(ix+1, colors[ix]))
4
5 selection = input("Select color: ")
6 if not selection.isdecimal():
7     exit(f"We need a number between 1 and
{len(colors)}")
8
9 if int(selection) < 1 or int(selection) > len(colors):
10    exit(f"The number must be between 1 and
{len(colors)}")
11
12 col = int(selection) - 1
13 print(colors[col])

```

- We would like to show a menu where each number corresponds to one element of the list so this is one of the places where we need to iterate over the indexes of a list.
- `len(colors)` gives us the length of the list (in our case 4)
- `range(len(colors))` is the range of numbers between 0 and 4 (in our case), meaning 0, 1, 2, 3.
- (Sometimes people explicitly write 4 in this solution, but if later we change the list and include another color we'll have to remember updating this number as well. This is error prone and it is very easy to deduct this number from the data we already have. (The list.))
- We start the list from 0, but when we display the menu we would like to show the numbers 1-4 to make it more human

friendly. Therefore we show `ix+1` and the color from locations `ix`.

- We ask for input and save it in a variable.
- We use the `isdecimal` method to check if the user typed in a decimal number. We give an error and exit if not.
- Then we check if the users provided a number in the correct range of values. We give an error and exit if not.
- then we convert the value to the correct range of numbers (remember, the user sees and selects numbers between 1-4 and we need them between 0-3).

Solution: count digits

```
1 numbers = [1203, 1256, 312456, 98]
2
3 count = [0] * 10 # same as [0, 0, 0, 0, 0, 0, 0, 0, 0,
0]
4
5 for num in numbers:
6     for char in str(num):
7         count[int(char)] += 1
8
9 for d in range(0, 10):
10    print("{} {}".format(d, count[d]))
```

First we have to decide where are we going to store the counts. A 10 element long list seems to fit our requirements so if we have 3 0s and 2 8s we would have `[3, 0, 0, 0, 0, 0, 0, 2, 0]`.

- We have a list of numbers.
- We need a place to store the counters. For this we create a variable called counter which is a list of 10 0s. We are going to count the number of times the digit 3 appears in `counters[3]`.

- We iterate over the numbers so `num` is the current number. (e.g. 1203)
- We would like to iterate over the digits in the current number now, but if we write `for var in num` we will get an error `TypeError: 'int' object is not iterable` because `num` is a number, but numbers are not iterables, so we cannot iterate over them. So we need to convert it to a string using `str`.
- On each iteration `char` will be one character (which in our case we assume that will be a digit, but still stored as a string).
- `int(char)` will convert the string to a number so for example “2” will be converted to 2.
- `count[int(char)]` is going to be `char[2]` if `char` is “2”. That’s the location in the list where we count how many times the digit 2 appears in our numbers.
- We increment it by one as we have just encountered a new copy of the given digit.
- That finished the data collection.
- The second for-loop iterates over all the “possible digits” that is from 0-9, prints out the digit and the counter in the respective place.

Solution: Create list

```

1 lines = [
2     'grape banana mango',
3     'nut orange peach',
4     'apple nut banana apple mango',
5 ]
6
7 one_line = ' '.join(lines)
8 print(one_line)

```

```
9 fruits = one_line.split()
10 print(fruits)
11
12 unique_fruits = []
13 for word in fruits:
14     if word not in unique_fruits:
15         unique_fruits.append(word)
16 print(sorted(unique_fruits))
17
18
19 # a simpler way using a set, but we have not learned
sets yet.
20 unique = sorted(set(fruits))
21 print(unique)
```

Solution: Count words

```
1 celestial_objects = [
2     'Moon', 'Gas', 'Asteroid', 'Dwarf', 'Asteroid',
'Moon', 'Asteroid'
3 ]
4
5 names    = []
6 counter = []
7
8 for name in celestial_objects:
9     if name in names:
10         idx = names.index(name)
11         counter[idx] += 1
12     else:
13         names.append(name)
14         counter.append(1)
15
16 for i in range(len(names)):
17     print("{:12} {}".format(names[i], counter[i]))
```

Solution: Check if number is prime

```
1 import sys
2
3 n = int(sys.argv[1])
```

```
4
5 #print(n)
6
7 is_prime = True
8 for i in range(2, int(n ** 0.5) + 1):
9     if n % i == 0:
10         is_prime = False
11         break
12
13 print(is_prime)
14
15
16 # math.sqrt(n) might be clearer than n ** 0.5
```

Solution: DNA sequencing

```
1 dna = 'ACCGXXCXXGTTACTGGGCXTTGT'
2 sequences = dna.split('X')
3 sequences.sort(key=len, reverse=True)
4
5 new_seq = []
6 for w in sequences:
7     if len(w) > 0:
8         new_seq.append(w)
9
10 print(sequences)
11 print(new_seq)
```

Solution: DNA sequencing with filter

```
1 dna = 'ACCGXXCXXGTTACTGGGCXTTGT'
2 sequences = dna.split('X')
3 sequences.sort(key=len, reverse=True)
4
5 def not_empty(x):
6     return len(x) > 0
7
8 print(sequences)
9 sequences = list(filter(not_empty, sequences))
10 print(sequences)
```

Solution: DNA sequencing with filter and lambda

```
1 dna = 'ACCGXXCXXGTTACTGGGCXTTGT'
2 sequences = dna.split('X')
3 sequences.sort(key=len, reverse=True)
4
5 print(sequences)
6 sequences = list(filter(lambda x: len(x) > 0,
7 sequences))
print(sequences)
```

[] .extend

```
1 names = ['Foo Bar', 'Orgo Morgo']
2
3 names.extend(['Joe Doe', 'Jane Doe'])
4 print(names) # ['Foo Bar', 'Orgo Morgo', 'Joe Doe',
'Jane Doe']
```

append vs. extend

What is the difference between [].append and [].extend ?

The method **append** adds its parameter as a single element to the list, while **extend** gets a list and adds its content.

```
1 names = ['Foo Bar', 'Orgo Morgo']
2 more = ['Joe Doe', 'Jane Doe']
3 names.extend(more)
4 print(names) # ['Foo Bar', 'Orgo Morgo', 'Joe Doe',
'Jane Doe']
5
6 names = ['Foo Bar', 'Orgo Morgo']
7 names.append(more)
8 print(names) # ['Foo Bar', 'Orgo Morgo', ['Joe Doe',
```

```
'Jane Doe']]  
9  
10 names = ['Foo', 'Bar']  
11 names.append('Qux')  
12 print(names)    # ['Foo', 'Bar', 'Qux']  
13  
14 names = ['Foo', 'Bar']  
15 names.extend('Qux')  
16 print(names)    # ['Foo', 'Bar', 'Q', 'u', 'x']
```

split and extend

When collecting data which is received from a string via splitting,
we would like to add the new elements to the existing list:

```
1 lines = [  
2     'abc def ghi',  
3     'hello world',  
4 ]  
5  
6 collector = []  
7  
8 for l in lines:  
9     collector.extend(l.split())  
10    print(collector)  
11  
12 # ['abc', 'def', 'ghi']  
13 # ['abc', 'def', 'ghi', 'hello', 'world']
```

Files

Open and read file

```
1 filename = 'examples/files/numbers.txt'  
2  
3 with open(filename, 'r') as fh:  
4     for line in fh:  
5         print(line)           # duplicate newlines  
6  
7 # close is called when we leave the 'with'
```

Filename on the command line

```
1 import sys  
2  
3 def main():  
4     if len(sys.argv) != 2:  
5         exit("Usage: " + sys.argv[0] + " FILENAME")  
6     filename = sys.argv[1]  
7     with open(filename) as fh:  
8         print("Working on the file", filename)  
9  
10 main()
```

```
1 $ python single.py  
2 Usage: single.py FILENAME  
3  
4 $ python single.py numbers.txt  
5 Working on the file numbers.txt
```

Filehandle with and without

```
1 filename = 'examples/files/numbers.txt'
2
3 fh = open(filename, 'r')
4 print(fh)      # <open file 'numbers.txt', mode 'r' at
0x107084390>
5 data = fh.read()
6 # do something with the data
7 fh.close()
8 print(fh)      # <closed file 'numbers.txt', mode 'r'
at 0x107084390>
9
10
11
12 with open(filename, 'r') as fh:
13     print(fh)  # <open file 'numbers.txt', mode 'r' at
0x1070840c0>
14     data = fh.read()
15 print(fh)      # <closed file 'numbers.txt', mode 'r'
at 0x1070840c0>
```

Filehandle with return

```
1 import sys
2
3 def process_file(filename):
4     with open(filename, 'r') as fh:
5
6         for line in fh:
7             line = line.rstrip("\n")
8             if len(line) > 0:
9                 if line[0] == '#':
10                     return
11 # some comment
12
13         if len(line) > 1:
14             if line[0:2] == '//':
15                 return
16
17         print(line)
18
19
20 process_file(sys.argv[0])
```

Read file remove newlines

```
1 filename = 'examples/files/numbers.txt'  
2  
3 with open(filename, 'r') as fh:  
4     for line in fh:  
5         line = line.rstrip("\n")  
6         print(line)
```

Read all the lines into a list

```
1 filename = 'examples/files/numbers.txt'  
2  
3 with open(filename, 'r') as fh:  
4     lines_list = fh.readlines()      # reads all the  
lines into a list  
5  
6 # print number of lines  
7 print(len(lines_list))  
8  
9 for line in lines_list:  
10    print(line, end="")
```

Read all the characters into a string (slurp)

```
1 filename = 'examples/files/numbers.txt'  
2  
3 with open(filename, 'r') as fh:  
4     lines_str = fh.read()      # reads all the lines into  
a string  
5  
6 print(len(lines_str))      # number of characters in file  
7  
8 print(lines_str)           # the content of the file
```

read(20) will read 20 bytes.

Not existing file

```
1 filename = 'examples/files/unicorns.txt'  
2  
3 with open(filename, 'r') as fh:  
4     lines = fh.read()  
5 print("still running")  
6  
7 # Traceback (most recent call last):  
8 #   File "examples/files/open_file.py", line 5, in  
<module>  
9 #       with open(filename, 'r') as fh:  
10 # IOError: [Errno 2] No such file or directory:  
'examples/files/unicorns.txt'
```

Open file exception handling

Exception handling

```
1 filename = 'examples/files/unicorns.txt'  
2  
3 try:  
4     with open(filename, 'r') as fh:  
5         lines = fh.read()  
6 except Exception as err:  
7     print('There was some error in the file  
operations.')  
8     print(err)  
9     print(type(err).__name__)  
10  
11 print('Still running.')
```

Open many files - exception handling

```
1 import sys  
2  
3  
4 def main():  
5     for filename in sys.argv[1:]:  
6         try:
```

```
7     #do some stuff(filename)
8     with open(filename) as fh:
9         total = 0
10        count = 0
11        for line in fh:
12            number = float(line)
13            total += number
14            count += 1
15            print("Average: ", total/count)
16    except Exception:
17        print("trouble with {}".format(filename))
18
19 main()
```

```
1 23
2 1
3 192
4 17
```

```
1
```

```
1 python average_from_files.pyt number_per_line.txt
empty.txt number_per_line2.txt
```

```
1 Average: 58.25
2 trouble with empty.txt
3 Average: 3.5
```

Writing to file

```
1 filename = 'data.txt'
2
3 with open(filename, 'w') as out:
4     out.write('text\n')
```

Append to file

```
1 filename = 'data.txt'
2
3 with open(filename, 'a') as out:
4     out.write('append more text\n')
```

Binary mode

```
1 filename = 'README'
2
3 try:
4     with open(filename, 'rb') as fh:
5         while True:
6             binary_str = fh.read(5000)
7             print(len(binary_str))
8             if len(binary_str) == 0:
9                 break
10            # do something with the content of the
binary_str
11 except Exception:
12     pass
13
14 # 5000
15 # 5000
16 # 5000
17 # 1599
18 # 0
```

Does file exist? Is it a file?

- [os.path.exists](#)
- [os.path.isfile](#)
- [os.path.isdir](#)

Exercise: count numbers

```
1 23 345 12345
2 67 189 23 17
```

1. Given the file **examples/files/numbers.txt** (or a similar file), count how many times each digit appears? The output will look like this. Just different values.
 2. Save the results in a file called **report.txt**.
-

```
1 0 0
2 1 3
3 2 3
4 3 4
5 4 2
6 5 2
7 6 1
8 7 2
9 8 1
10 9 1
```

Exercise: strip newlines

How to read all the lines of a file into a list and remove trailing newlines?

Exercise: color selector

Create a file similar to the colors.txt file and use it as the list of colors in the earlier example where we prompted for a color.

```
1 blue
2 yellow
3 white
4 green
```

Extend the previous example by letting the user provide the name of the file on the command line:

```
python color.py examples/files/color.txt
```

Exercise: ROT13

Implement [ROT13](#):

- Create a function that given a string return the rot13 of it.
- Create a script that given a file it will replace with the rot13 of it.

How to check if it works properly:

```
1 txt = "any text"
2 encrypted = rot13(txt)
3 decrypted = rot13(encrypted)
4 assert decrypted == text
```

Exercise: Combine lists

```
1 Tomato=78
2 Avocado=23
3 Pumpkin=100
```

```
1 Cucumber=17
2 Avocado=10
3 Cucumber=10
```

Write a script that takes the two files and combines them adding the values for each vegetable. The expected result is:

```
1 Avocado=33
2 Cucumber=27
3 Pumpkin=100
4 Tomato=78
```

Solution: count numbers

```
1 import sys
2
3 if len(sys.argv) < 2:
4     exit("Need name of file.")
5
6 counter = [0] * 10
7 filename = sys.argv[1]
8 with open(filename) as fh:
9     for line in fh:
10         for c in line.rstrip("\n"):
11             if c == ' ':
12                 continue
13
14             c = int(c)
15             counter[c] += 1
16
17 for i in range(10):
18     print("{} {}".format(i, counter[i]))
```

Solution: strip newlines

```
1 import sys
2 filename = sys.argv[0]
3 with open(filename) as fh:
4     lines = []
5     for line in fh:
6         lines.append(line.rstrip("\n"))
7     print(lines)
```

Solution: color selector

```
1 def main():
2     try:
3         with open('colors.txt') as fh:
4             colors = []
5             for line in fh:
6                 colors.append(line.rstrip("\n"))
7     except IOError:
8         print("Could not open colors.txt")
9         exit()
```

```

10
11     for i in range(len(colors)):
12         print("{} {}) {}".format(i, colors[i]))
13
14     c = int(input("Select color: "))
15     print(colors[c])
16
17 main()

```

Solution: Combine lists

```

1 a_names = []
2 a_values = []
3 with open('examples/files/a.txt') as fh:
4     for line in fh:
5         k, v = line.rstrip("\n").split("=")
6         a_names.append(k)
7         a_values.append(int(v))
8
9 b_names = []
10 b_values = []
11 with open('examples/files/b.txt') as fh:
12     for line in fh:
13         k, v = line.rstrip("\n").split("=")
14         b_names.append(k)
15         b_values.append(int(v))
16
17 c_names = []
18 c_values = []
19
20 for i in range(len(a_names)):
21     if a_names[i] in c_names:
22         j = c_names.index(a_names[i])
23         c_values[j] += a_values[i]
24     else:
25         c_names.append( a_names[i] )
26         c_values.append( a_values[i] )
27
28 for i in range(len(b_names)):
29     if b_names[i] in c_names:
30         j = c_names.index(b_names[i])
31         c_values[j] += b_values[i]
32     else:

```

```
33         c_names.append( b_names[i] )
34         c_values.append( b_values[i] )
35
36
37 with open('out.txt', 'w') as fh:
38     for i in range(len(c_names)):
39         fh.write("{}={}\n".format(c_names[i],
c_values[i]))
```

Read text file

```
1 filename = 'examples/files/numbers.txt'
2
3 with open(filename, 'r') as fh:      # open(filename)
would be enough
4     for line in fh:
5         print(line)                  # duplicate newlines
6         #print(line, end="")        # eliminte the trailing
newline of print
```

Open and read file

In some code you will encounter the following way of opening files.

This was used before “with” was added to the language.

It is not a recommended way of opening a file as you might easily forget

to call “close” and that might cause trouble. For example you might loose data.

Don’t do that.

```
1 filename = 'examples/files/numbers.txt'
2
3 fh = open(filename, 'r')
4 for line in fh:
```

```
5     print(line)                      # duplicate newlines
6 fh.close()
```

Direct access of a line in a file

```
1 names = ['Foo', 'Bar', 'Baz']
2 for name in names:
3     print(name)
4 print(names[1])
5
6
7 filename = 'data/README'
8 with open(filename, 'r') as fh:
9     for line in fh:
10         print(line)
11
12 with open(filename, 'r') as fh:
13     print(fh[2])
```

```
1 Traceback (most recent call last):
2   File "examples/files/fh_access.py", line 14, in
<module>
3       print(fh[2])
4 TypeError: '_io.TextIOWrapper' object is not
subscriptable
```

This does NOT work because files can only be accessed sequentially.

Example

```
1 begin test
2 do something
3 report
4 total: 42
5 more things
```

```
6 more
7 another total: 100
8 more data
```

```
1 import sys
2 import os
3
4 #print(sys.argv)
5 if len(sys.argv) < 2:
6     #exit()
7     exit(f"Usage: {sys.argv[0]} FILENAME")
8
9 # print(sys.argv[0])
10 # print(sys.argv[1])
11
12 filename = 'sample.txt'
13
14 #filename = input("type in filename: ")
15
16 filename = sys.argv[1]
17
18 #if not os.path.exists(filename):
19 #    exit(f"File {filename} does not exist")
20
21 with open(filename, 'r') as fh:
22     for line in fh:
23         line = line.rstrip("\n")
24         print(line)
25         #if "total" in line:
26             #    print(line)
27
```

Dictionary (hash)

What is a dictionary

- Unordered key-value pairs.
- Keys are immutables (numbers, strings, tuples).
- Values can be any object.

When to use dictionaries

- ID to Name mapping.
- Object to Count mapping.
- Name of a feature to value of the feature.
- Name of an attribute to value of the attribute.

Dictionary

```
1 user = {}
2 user['name'] = 'Foobar'
3 print(user)          # {'name': 'Foobar'}
4
5 user['email'] = 'foo@bar.com'
6 print(user)          # {'name': 'Foobar', 'email':
'foo@bar.com'}
7
8 the_name = user['name']
9 print(the_name)      # Foobar
10
11 field = 'name'
12 the_value = user[field]
13 print(the_value)    # Foobar
14
15 user['name'] = 'Edith Piaf'
```

```
16 print(user)      # {'name': 'Edith Piaf', 'email':  
'foo@bar.com'}
```

keys

```
1 user = {  
2     'fname': 'Foo',  
3     'lname': 'Bar',  
4 }  
5  
6 print(user)    # {'lname': 'Bar', 'fname': 'Foo'}  
7  
8 print(user.keys())    # ['lname', 'fname']
```

- Keys are returned in seemingly random order.

Loop over keys

```
1 user = {  
2     'fname': 'Foo',  
3     'lname': 'Bar',  
4 }  
5  
6 for k in user.keys():  
7     print(k)  
8  
9 # lname  
10 # fname  
11  
12 for k in user.keys():  
13     print("{} -> {}".format(k, user[k]))  
14  
15 # lname -> Bar  
16 # fname -> Foo
```

Loop using items

```
1 people = {  
2     "foo" : "123",
```

```
3     "bar" : "456",
4     "qux" : "789",
5 }
6
7 for name, uid in people.items():
8     print("{} => {}".format(name, uid))
```

```
1 foo => 123
2 bar => 456
3 qux => 789
```

```
1 user = {
2     'fname': 'Foo',
3     'lname': 'Bar',
4 }
5
6 for t in user.items():      # returns tuples
7     print("{} -> {}".format(t[0], t[1]))
8     #print("{} -> {}".format(*t))
9
10 # lname -> Bar
11 # fname -> Foo
```

values

- Values are returned in the same random order as the keys are.

```
1 user = {
2     'fname': 'Foo',
3     'lname': 'Bar',
4 }
5
6 print(user)    # {'lname': 'Bar', 'fname': 'Foo'}
7
8 print(user.keys())    # ['lname', 'fname']
9
10 print(user.values())   # ['Bar', 'Foo']
```

Not existing key

If we try to fetch the value of a key that does not exist, we get an exception.

```
1 def main():
2     user = {
3         'fname': 'Foo',
4         'lname': 'Bar',
5     }
6
7     print(user['fname'])
8     print(user['email'])
9
10 main()
```

```
1 Foo
2 Traceback (most recent call last):
3   File "examples/dictionary/no_such_key.py", line 11,
4     in <module>
5       main()
6   File "examples/dictionary/no_such_key.py", line 9, in
main
7     print(user['email'])
8 KeyError: 'email'
```

Get key

If we use the `get` method, we get `None` if the key does not exist.

```
1 user = {
2     'fname': 'Foo',
3     'lname': 'Bar',
4     'address': None,
5 }
6
7 print(user.get('fname'))
8 print(user.get('address'))
9 print(user.get('email'))
10
11 print(user.get('answer', 42))
```

```
1 Foo
2 None
3 None
4 42
```

None will be interpreted as False, if checked as a boolean.

Does the key exist?

```
1 user = {
2     'fname': 'Foo',
3     'lname': 'Bar',
4 }
5
6 print('fname' in user)    # True
7 print('email' in user)    # False
8 print('Foo' in user)      # False
9
10 for k in ['fname', 'email', 'lname']:
11     if k in user:
12         print("{} => {}".format(k, user[k]))
13
14 # fname => Foo
15 # lname => Bar
```

```
1 True
2 False
3 False
4 fname => Foo
5 lname => Bar
```

Does the value exist?

```
1 user = {
2     'fname': 'Foo',
3     'lname': 'Bar',
4 }
5
```

```
6 print('fname' in user.values()) # False
7 print('Foo' in user.values()) # True
```

```
1 False
2 True
```

Delete key

```
1 user = {
2     'fname': 'Foo',
3     'lname': 'Bar',
4     'email': 'foo@bar.com',
5 }
6
7 print(user) # {'lname': 'Bar', 'email': 'foo@bar.com',
'fname': 'Foo'}
8
9 fname = user['fname']
10 del user['fname']
11 print(fname) # Foo
12 print(user) # {'lname': 'Bar', 'email': 'foo@bar.com'}
13
14 lname_was = user.pop('lname')
15 print(lname_was) # Bar
16 print(user) # {'email': 'foo@bar.com'}
```

```
1 {'fname': 'Foo', 'lname': 'Bar', 'email':
'foo@bar.com'}
2 Foo
3 {'lname': 'Bar', 'email': 'foo@bar.com'}
4 Bar
5 {'email': 'foo@bar.com'}
```

List of dictionaries

```
1 people = [
2     {
3         'name' : 'Foo Bar',
4         'email' : 'foo@example.com'
```

```
5     },
6     {
7         'name'      : 'Qux Bar',
8         'email'     : 'qux@example.com',
9         'address'   : 'Borg, Country',
10        'children' : [
11            'Alpha',
12            'Beta'
13        ]
14    }
15 ]
16
17 print(people)
18 print(people[0]['name'])
19 print(people[1]['children'][0])
20
21 print(list(map(lambda p: p['name'], people)))
```

```
1 [{ 'name': 'Foo Bar', 'email': 'foo@example.com'},
2 { 'name': 'Qux Bar', 'email': 'qux@\\
3 example.com', 'address': 'Borg, Country', 'children':
4 ['Alpha', 'Beta']}]
3 Foo Bar
4 Alpha
5 ['Foo Bar', 'Qux Bar']
```

Shared dictionary

```
1 people = [
2     {
3         "name" : "Foo",
4         "id"   : "1",
5     },
6     {
7         "name" : "Bar",
8         "id"   : "2",
9     },
10    {
11        "name" : "Moo",
12        "id"   : "3",
13    },
14 ]
```

```
15
16 by_name = {}
17 by_id = {}
18 for p in people:
19     by_name[ p['name'] ] = p
20     by_id[ p['id'] ] = p
21 print(by_name)
22 print(by_id)
23
24 print(by_name["Foo"])
25 by_name["Foo"]['email'] = 'foo@weizmann.ac.il'
26 print(by_name["Foo"])
27
28 print(by_id["1"])
```

```
1 {'Foo': {'name': 'Foo', 'id': '1'}, 'Bar': {'name':
'Bar', 'id': '2'}, 'Moo': {'name\
2 ': 'Moo', 'id': '3'}}
3 {'1': {'name': 'Foo', 'id': '1'}, '2': {'name': 'Bar',
'id': '2'}, '3': {'name': 'Mo\
4 o', 'id': '3'}}
5 {'name': 'Foo', 'id': '1'}
6 {'name': 'Foo', 'id': '1', 'email':
'foo@weizmann.ac.il'}
7 {'name': 'Foo', 'id': '1', 'email':
'foo@weizmann.ac.il'}
```

immutable collection: tuple as dictionary key

```
1 points = {}
2 p1 = (2, 3)
3
4 points[p1] = 'Joe'
5 points[(17, 5)] = 'Jane'
6
7 print(points)
8 for k in points.keys():
9     print(k)
10    print(k.__class__.__name__)
11    print(points[k])
```

```
1 {(2, 3): 'Joe', (17, 5): 'Jane'}
2 (2, 3)
3 tuple
4 Joe
5 (17, 5)
6 tuple
7 Jane
```

immutable numbers: numbers as dictionary key

```
1 number = {
2     23    : "Twenty three",
3     17    : "Seventeen",
4     3.14  : "Three dot fourteen",
5     42    : "The answer",
6 }
7
8 print(number)
9 print(number[42])
10 print(number[3.14])
```

```
1 {23: 'Twenty three', 17: 'Seventeen', 3.14: 'Three dot
fourteen', 42: 'The answer'}
2 The answer
3 Three dot fourteen
```

Sort dictionary by value

```
1 scores = {
2     'Foo' : 10,
3     'Bar' : 34,
4     'Miu' : 88,
5 }
6
7 print(scores) # {'Miu': 88, 'Foo': 10, 'Bar': 34}
8
9 sorted_names = sorted(scores)
10 print(sorted_names) # ['Bar', 'Foo', 'Miu']
```

```

11 for s in sorted_names:
12     print("{} {}".format(s, scores[s]))
13
14 # sort the values, but we cannot get the keys back!
15 print(sorted(scores.values())) # [10, 34, 88]
16
17 print('')
18
19 # sort using a lambda expression
20 sorted_names = sorted(scores, key=lambda x: scores[x])
21 for k in sorted_names:
22     print("{} : {}".format(k, scores[k]))
23
24 # Foo : 10
25 # Bar : 34
26 # Miu : 88
27
28 print('')
29
30 # sort the keys according to the values:
31 sorted_names = sorted(scores, key=scores.__getitem__)
32 for k in sorted_names:
33     print("{} : {}".format(k, scores[k]))
34
35 # Foo : 10
36 # Bar : 34
37 # Miu : 88

```

Sort dictionary keys by value

```

1 scores = {
2     "Jane"      : 30,
3     "Joe"       : 20,
4     "George"    : 30,
5     "Hellen"    : 90,
6 }
7
8 for name in scores.keys():
9     print(f"{name:8} {scores[name]}")
10
11 print('')
12 for name in sorted(scores.keys()):
13     print(f"{name:8} {scores[name]}")

```

```
14
15 print('')
16 for val in sorted(scores.values()):
17     print(f"{val:8}")
18
19 print('')
20 for name in sorted(scores.keys(), key=lambda x:
21 scores[x]):
22     print(f"{name:8} {scores[name]}")
```

```
1 Jane      30
2 Joe       20
3 George    30
4 Hellena   90
5
6 George    30
7 Hellena   90
8 Jane      30
9 Joe       20
10
11        20
12        30
13        30
14        90
15
16 Joe      20
17 Jane    30
18 George   30
19 Hellena 90
```

Insertion Order is kept

Since Python 3.7

```
1 d = {}
2 d['a'] = 1
3 d['b'] = 2
4 d['c'] = 3
5 d['d'] = 4
6 print(d)
```

```
1 { 'a': 1, 'b': 2, 'c': 3, 'd': 4}
```

Change order of keys in dictionary - OrderedDict

```
1 from collections import OrderedDict
2
3 d = OrderedDict()
4 d['a'] = 1
5 d['b'] = 2
6 d['c'] = 3
7 d['d'] = 4
8
9 print(d)
10 d.move_to_end('a')
11
12 print(d)
13 d.move_to_end('d', last=False)
14
15 print(d)
16
17 for key in d.keys():
18     print(key)
```

```
1 OrderedDict([('a', 1), ('b', 2), ('c', 3), ('d', 4)])
2 OrderedDict([('b', 2), ('c', 3), ('d', 4), ('a', 1)])
3 OrderedDict([('d', 4), ('b', 2), ('c', 3), ('a', 1)])
4 d
5 b
6 c
7 a
```

Set order of keys in dictionary - OrderedDict

```
1 from collections import OrderedDict
2
3 d = {}
4 d['a'] = 1
5 d['b'] = 2
```

```

6 d['c'] = 3
7 d['d'] = 4
8 print(d)
9
10 planned_order = ('b', 'c', 'd', 'a')
11 e = OrderedDict(sorted(d.items(), key=lambda x:
planned_order.index(x[0])))
12 print(e)
13
14 print('-----')
15 # Create index to value mapping dictionary from a list
of values
16 planned_order = ('b', 'c', 'd', 'a')
17 plan = dict(zip(planned_order,
range(len(planned_order))))
18 print(plan)
19
20 f = OrderedDict(sorted(d.items(), key=lambda x:
plan[x[0]]))
21 print(f)

```

```

1 {'a': 1, 'b': 2, 'c': 3, 'd': 4}
2 OrderedDict([('b', 2), ('c', 3), ('d', 4), ('a', 1)])
3 -----
4 {'b': 0, 'c': 1, 'd': 2, 'a': 3}
5 OrderedDict([('b', 2), ('c', 3), ('d', 4), ('a', 1)])

```

Exercise: count characters

Given a long text, count how many times each character appears?

```

1 text = """
2 This is a very long text.
3 OK, maybe it is not that long after all.
4 """

```

Extra credit: Change the code so it will be able to count characters of a file.

Exercise: count words

Part of the code:

```
1 words = ['Wombat', 'Rhino', 'Sloth', 'Tarantula',
'Sloth', 'Rhino', 'Sloth']
```

Expected output: (the order is not important)

```
1 Wombat:1
2 Rhino:2
3 Sloth:3
4 Tarantula:1
```

Exercise: count words from a file

Given a file with words and spaces and newlines only, count how many times each word appears.

```
1 Lorem ipsum dolor qui ad labor ad labor sint dolor
tempor incididunt ut labor ad do\l
2 lore lorem ad
3 Ut labor ad dolor lorem qui ad ut labor    ut ad commodo
commodo
4 Lorem ad dolor in reprehenderit in lorem ut labor ad
dolore eu in labor dolor
5 sint occaecat ad labor proident sint in in qui labor ad
dolor ad in ad labor
```

- Based on [Lorem Ipsum](#)

Expected result for the above file:

1 ad	13
2 commodo	2
3 dolor	6
4 dolore	2
5 eu	1

```
6 in          6
7 incididuntut 1
8 ipsum        1
9 labor        10
10 lorem       5
11 occaecat    1
12 proident    1
13 qui         3
14 reprehenderit 1
15 sint        3
16 tempor      1
17 ut          5
```

Exercise: Apache log

Every web server logs the visitors and their requests in a log file. The Apache web server has a log file similar to the following file. (Though I have trimmed the lines for the exercise.) Each line is a “hit”, a request from the browser of a visitor.

Each line starts with the IP address of the visitor. e.g. 217.0.22.3.

Given such a log file from Apache, report how many hits (line were from each IP address.

```
1 127.0.0.1 - - [10/Apr/2007:10:39:11] "GET / HTTP/1.1"
500 606 "-"
2 127.0.0.1 - - [10/Apr/2007:10:39:11] "GET /favicon.ico
HTTP/1.1" 200 766 "-"
3 139.12.0.2 - - [10/Apr/2007:10:40:54] "GET / HTTP/1.1"
500 612 "-"
4 139.12.0.2 - - [10/Apr/2007:10:40:54] "GET
/favicon.ico HTTP/1.1" 200 766 "-"
5 127.0.0.1 - - [10/Apr/2007:10:53:10] "GET / HTTP/1.1"
500 612 "-"
6 127.0.0.1 - - [10/Apr/2007:10:54:08] "GET / HTTP/1.0"
200 3700 "-"
7 127.0.0.1 - - [10/Apr/2007:10:54:08] "GET /style.css
HTTP/1.1" 200 614
```

```
8 127.0.0.1 - - [10/Apr/2007:10:54:08] "GET /img/pti-round.jpg HTTP/1.1" 200 17524
9 127.0.0.1 - - [10/Apr/2007:10:54:21] "GET /unix_sysadmin.html HTTP/1.1" 200 3880
10 217.0.22.3 - - [10/Apr/2007:10:54:51] "GET / HTTP/1.1"
200 34 "-"
11 217.0.22.3 - - [10/Apr/2007:10:54:51] "GET /favicon.ico HTTP/1.1" 200 11514 "-"
12 217.0.22.3 - - [10/Apr/2007:10:54:53] "GET /cgi/pti.pl
HTTP/1.1" 500 617
13 127.0.0.1 - - [10/Apr/2007:10:54:08] "GET / HTTP/0.9"
200 3700 "-"
14 217.0.22.3 - - [10/Apr/2007:10:58:27] "GET / HTTP/1.1"
200 3700 "-"
15 217.0.22.3 - - [10/Apr/2007:10:58:34] "GET /unix.html
HTTP/1.1" 200 3880
16 217.0.22.3 - - [10/Apr/2007:10:58:45] "GET
/talks/read.html HTTP/1.1" 404 311
17 127.0.0.1 - - [10/Apr/2007:10:54:08] "GET /img/pti-round.jpg HTTP/1.1" 200 17524
18 127.0.0.1 - - [10/Apr/2007:10:54:08] "GET /img/pti-round.jpg HTTP/1.1" 200 17524
19 127.0.0.1 - - [10/Apr/2007:10:54:21] "GET
/unix_sysadmin.html HTTP/1.1" 200 3880
20 127.0.0.1 - - [10/Apr/2007:10:54:21] "GET
/unix_sysadmin.html HTTP/1.1" 200 3880
21 217.0.22.3 - - [10/Apr/2007:10:54:51] "GET / HTTP/1.1"
200 34 "-"
```

Expected output:

1	127.0.0.1	12
2	139.12.0.2	2
3	217.0.22.3	7

Exercise: Combine lists again

See the same exercise in the previous chapter.

Exercise: counting DNA bases

Given a sequence like this:

“ACTNGTGCTYGATRGTAGCYXGTN”,

print out the distribution of the elements to get the following result:

1	A	3	-	12.50	%
2	C	3	-	12.50	%
3	G	6	-	25.00	%
4	N	2	-	8.33	%
5	R	1	-	4.17	%
6	T	6	-	25.00	%
7	X	1	-	4.17	%
8	Y	2	-	8.33	%

Exercise: Count Amino Acids

- Each sequence consists of many repetition of the 4 bases represented by the ACTG characters.
- There are 64 codons (sets of 3 bases following each other)
- There are 22 [Amino Acids](#) each of them are represented by 3 bases.
- Some of the Amino Acids can be represented in multiple ways. For example Histidine can be encoded by both CAU, CAC)
- We have a DNA sequence
- Count the Amino acids form the sequence. (For our purposes feel free to generate a DNA sequence with a random number generator.

Exercise: List of dictionaries

Given the following file build a list of dictionaries where each dictionary represents one person.

The keys in the dictionary are the names of the columns (fname, lname, born) the values are the respective values from each row.

```
1 fname, lname, born
2 Graham, Chapman, 8 January 1941
3 Eric, Idle, 29 March 1943
4 Terry, Gilliam, 22 November 1940
5 Terry, Jones, 1 February 1942
6 John, Cleese, 27 October 1939
7 Michael, Palin, 5 May 1943
```

```
1 print(people[1]['fname']) # Eric
```

Exercise: Dictinoary of dictionaries

Given the following file build a dictionary of dictionaries where each internal dictionary represents one person.

The keys in the internal dictionaries are the names of the columns (fname, lname, born) the values are the respective values from each row.

In the outer dictionary the keys are the (fname, lname) tuples.

```
1 fname, lname, born
2 Graham, Chapman, 8 January 1941
3 Eric, Idle, 29 March 1943
4 Terry, Gilliam, 22 November 1940
5 Terry, Jones, 1 February 1942
6 John, Cleese, 27 October 1939
7 Michael, Palin, 5 May 1943
```

```
1 print(people[('Eric', 'Idle')]['born']) # 29 March 1943
```

Solution: count characters

```
1 text = """
2 This is a very long text.
```

```
3 OK, maybe it is not that long after all.  
4 """  
5  
6 # print(text)  
7 count = {}  
8  
9 for char in text:  
10     if char == '\n':  
11         continue  
12     if char not in count:  
13         count[char] = 1  
14     else:  
15         count[char] += 1  
16  
17 for key in sorted(count.keys()):  
18     print("{} {}".format(key, count[key]))
```

- We need to store the counter somewhere. We could use two lists for that, but that would give a complex solution that runs in $O(n^{**}2)$ time.
- Besides, we are in the chapter about dictionaries so probably we better use a dictionary.
- In the `count` dictionary we each key is going to be one of the characters and the respective value will be the number of times it appeared.
- So if our string is “aabx” then we’ll end up with

```
1 {  
2     "a": 2,  
3     "b": 1,  
4     "x": 1,  
5 }
```

- The `for in` loop on a string will iterate over its character by character (even if we don’t call our variable `char`).
- We check if the current character is a newline `\n` and if it we call `continue` to skip the rest of the iteration. We don’t want

to count newlines.

- Then we check if we have already seen this character. That is, it is already one of the keys in the `count` dictionary. If not yet, then we add it and put 1 as the values. After all we saw one copy of this character. If we have already seen this character (we get to the `else` part) then we increment the counter for this character.
- We are done now with the data collection.
- In the second loop we go over the keys of the dictionary, that is the characters we have encountered. We sort them in ASCII order.
- Then we print each one of them and the respective value, the number of times the character was found.

Solution: count characters with default dict

```
1 from collections import defaultdict
2
3 text = """
4 This is a very long text.
5 OK, maybe it is not that long after all.
6 """
7
8 # print(text)
9 count = defaultdict(int)
10
11 for char in text:
12     if char == '\n':
13         continue
14     count[char] += 1
15
16 for key in sorted( count.keys() ):
17     print("{} {}".format(key, count[key]))
```

- The previous solution can be slightly improved by using `defaultdict` from the `collections` module.
- `count = defaultdict(int)` creates an empty dictionary that has the special feature that if you try to use a key that does not exist, it pretends that it exists and that it has a value 0.
- This allows us to remove the condition checking if the character was already seen and just increment the counter. The first time we encounter a character the dictionary will pretend that it was already there with value 0 so everything will work out nicely.

Solution: count words

```

1 words = ['Wombat', 'Rhino', 'Sloth', 'Tarantula',
'Sloth', 'Rhino', 'Sloth']
2
3 counter = {}
4 for word in words:
5     if word not in counter:
6         counter[word] = 0
7     counter[word] += 1
8
9 for word in counter:
10    print("{}:{}".format(word, counter[word]))

```

```

1 from collections import Counter
2
3 words = ['Wombat', 'Rhino', 'Sloth', 'Tarantula',
'Sloth', 'Rhino', 'Sloth']
4
5 cnt = Counter()
6 for word in words:
7     cnt[word] += 1
8
9 print(cnt)
10 for w in cnt.keys():
11     print("{}:{}".format(w, cnt[w]))

```

```
1 from collections import defaultdict
2
3 words = ['Wombat', 'Rhino', 'Sloth', 'Tarantula',
'Sloth', 'Rhino', 'Sloth']
4
5 dd = defaultdict(lambda : 0)
6 for word in words:
7     dd[word] += 1
8
9 print(dd)
10 for word in dd.keys():
11     print("{}:{}".format(word, dd[word]))
```

Solution: count words in file

```
1 import sys
2
3 filename = 'README'
4 if len(sys.argv) > 1:
5     filename = sys.argv[1]
6 print(filename)
7
8 count = {}
9
10 with open(filename) as fh:
11     for full_line in fh:
12         line = full_line.rstrip('\n')
13         line = line.lower()
14         for word in line.split():
15             if word == '':
16                 continue
17             if word not in count:
18                 count[word] = 0
19
20             count[word] += 1
21
22 for word in sorted(count):
23     print("{}:{}".format(word, count[word]))
```

Solution: Apache log

```
1 filename = 'examples/apache_access.log'
2
3 count = {}
4
5 with open(filename) as fh:
6     for line in fh:
7         space = line.index(' ')
8         ip = line[0:space]
9         if ip in count:
10             count[ip] += 1
11         else:
12             count[ip] = 1
13
14 for ip in count:
15     print("{} {:>3}".format(ip, count[ip]))
```

Solution: Combine lists again

```
1 c = {}
2 with open('examples/files/a.txt') as fh:
3     for line in fh:
4         k, v = line.rstrip("\n").split("=")
5         if k in c:
6             c[k] += int(v)
7         else:
8             c[k] = int(v)
9
10 with open('examples/files/b.txt') as fh:
11     for line in fh:
12         k, v = line.rstrip("\n").split("=")
13         if k in c:
14             c[k] += int(v)
15         else:
16             c[k] = int(v)
17
18
19 with open('out.txt', 'w') as fh:
20     for k in sorted(c.keys()):
21         fh.write("{}={}\n".format(k, c[k]))
```

Solution: counting DNA bases

```
1 seq = "ACTNGTGCTYGATRGTAGCYXGTN"
2 count = {}
3 for c in seq:
4     if c not in count:
5         count[c] = 0
6     count[c] += 1
7
8 for c in sorted(count.keys()):
9     print("{} {} - {:.2f} %".format(c, count[c], 100
* count[c]/len(seq)))
10
11 # >5 is the right alignment of 5 places
12 # .2f is the floating point with 2 digits after the
floating point
```

Solution: Count Amino Acids

Generate random DNA sequence

```
1 import sys
2 import random
3
4 if len(sys.argv) != 2:
5     exit("Need a number")
6 count = int(sys.argv[1])
7
8 dna = []
9 for _ in range(count):
10    dna.append(random.choice(['A', 'C', 'T', 'G']))
11 print(''.join(dna))
```

```
1 dna = 'CACCCATGAGATGTCTAACGCTGCTTCATTATAGCCG'
2
3 aa_by_codon = {
4     'ACG' : '?',
5     'CAC' : 'Histidin',
6     'CAU' : 'Histidin',
7     'CCA' : 'Proline',
8     'CCG' : 'Proline',
9     'GAT' : '?',
10    'GTC' : '?',
11    'TGA' : '?'}
```

```
12     'TTA' : '?',
13     'CTG' : '?',
14     'CTT' : '?',
15     'TCA' : '?',
16     'TAG' : '?',
17     #...
18 }
19
20 count = {}
21
22 for i in range(0, len(dna)-2, 3):
23     codon = dna[i:i+3]
24     #print(codon)
25     aa = aa_by_codon[codon]
26     if aa not in count:
27         count[aa] = 0
28     count[aa] += 1
29
30 for aa in sorted(count.keys()):
31     print("{} {}".format(aa, count[aa]))
```

Loop over dictionary keys

Looping over the “dictionary” is just like looping over the keys.

```
1 user = {
2     'fname': 'Foo',
3     'lname': 'Bar',
4 }
5
6 for k in user:
7     print("{} -> {}".format(k, user[k]))
8
9 # lname -> Bar
10 # fname -> Foo
```

Do not change dictionary in loop

```
1 user = {
2     'fname': 'Foo',
3     'lname': 'Bar',
```

```
4 }
5
6 for k in user.keys():
7     user['email'] = 'foo@bar.com'
8     print(k)
9
10 print('-----')
11
12 for k in user:
13     user['birthdate'] = '1991'
14     print(k)
15
16 # lname
17 # fname
18 # -----
19 # lname
20 # Traceback (most recent call last):
21 #   File "examples/dictionary/change_in_loop.py", line
22 #     13, in <module>
22 #       for k in user:
23 # RuntimeError: dictionary changed size during
iteration
```

Default Dict

```
1 counter = {}
2
3 word = 'eggplant'
4
5 counter[word] += 1
6 # counter[word] = counter[word] + 1
```

```
1 Traceback (most recent call last):
2   File "counter.py", line 5, in <module>
3     counter[word] += 1
4 KeyError: 'eggplant'
```

```
1 counter = {}
2
3 word = 'eggplant'
4
```

```
5 if word not in counter:  
6     counter[word] = 0  
7 counter[word] += 1  
8  
9 print(counter)
```

```
1 {'eggplant': 1}
```

```
1 from collections import defaultdict  
2  
3 counter = defaultdict(int)  
4  
5 word = 'eggplant'  
6  
7 counter[word] += 1  
8  
9 print(counter)
```

```
1 defaultdict(<class 'int'>, {'eggplant': 1})
```

Sets

sets

- Sets in Python are used when we are primarily interested in operations that we know from the [sets theory](#).
- See also the [Venn diagrams](#).
- In day to day speech we often use the word “group” instead of “set” even though they are not the same.
- What are the common elements of two set (two groups).
- Is one group (set) the subset of the other?
- What are all the elements that exist in both groups (sets)?
- What are the elements that exist in exactly one of the groups (sets)?

set operations

- set
- issubset
- intersection
- symmetric difference
- union
- relative complement
- [stdtypes: set](#)

set intersection

```
1 english = set(['door', 'car', 'lunar', 'era'])
2 spanish = set(['era', 'lunar', 'hola'])
3
4 print('english: ', english)
5 print('spanish: ', spanish)
6
7 both = english.intersection(spanish)
8 print(both)
```

- intersection returns the elements that are in both sets.

```
1 english:  {'car', 'lunar', 'era', 'door'}
2 spanish:  {'lunar', 'era', 'hola'}
3 {'lunar', 'era'}
```

set subset

```
1 english = set(['door', 'car', 'lunar', 'era'])
2 spanish = set(['era', 'lunar', 'hola'])
3
4 words = set(['door', 'lunar'])
5
6
7 print('issubset: ', words.issubset( english ))
8 print('issubset: ', words.issubset( spanish ))
```

- intersection returns the elements that are in both sets.

```
1 issubset:  True
2 issubset:  False
```

set symmetric difference

```
1 english = set(['door', 'car', 'lunar', 'era'])
2 spanish = set(['era', 'lunar', 'hola'])
3
4 diff = english.symmetric_difference(spanish)
5 print('symmetric_difference: ', diff)
```

-
- Symmetric difference is all the elements in either one of the sets, but not in both. “the ears of the elephant”.
-

```
1 symmetric_difference: { 'door', 'hola', 'car' }
```

set union

```
1 english = set(['door', 'car', 'lunar', 'era'])
2 spanish = set(['era', 'lunar', 'hola'])
3
4 all_the_words = english.union(spanish)
5
6 print(english)
7 print(spanish)
8 print(all_the_words)
```

```
1 {'era', 'door', 'lunar', 'car'}
2 {'era', 'hola', 'lunar'}
3 {'era', 'door', 'car', 'hola', 'lunar'}
```

set relative complement

```
1 english = set(['door', 'car', 'lunar', 'era'])
2 spanish = set(['era', 'lunar', 'hola'])
3
4
5 eng = english - spanish
6 spa = spanish - english
7
8 print(spa)
9 print(eng)
10
11 print(english)
12 print(spanish)
```

```
1 {'hola'}
2 {'door', 'car'}
```

```
3 {'door', 'era', 'car', 'lunar'}
4 {'hola', 'era', 'lunar'}
```

set examples

```
1 things = set(['table', 'chair', 'door', 'chair',
'chair'])
2 print(things)
3 print(things.__class__)
4 print(things.__class__.__name__)
5
6 if 'table' in things:
7     print("has table")
```

```
1 {'door', 'chair', 'table'}
2 <class 'set'>
3 set
4 has table
```

defining an empty set

```
1 objects = set()
2 print(objects)
```

```
1 set()
```

```
1 set([])
```

Adding an element to a set (add)

```
1 objects = set()
2 print(objects)
3
4 objects.add('Mars')
5 print(objects)
6
7 objects.add('Mars')
```

```
8 print(objects)
9
10 objects.add('Neptun')
11 print(objects)
```

```
1 set()
2 {'Mars'}
3 {'Mars'}
4 {'Neptun', 'Mars'}
```

In Python 2:

```
1 set([])
2 set(['Mars'])
3 set(['Mars'])
4 set(['Neptun', 'Mars'])
```

Merging one set into another set (update)

```
1 set(['Neptun', 'Mars'])
2
3
4 objects = set(['Mars', 'Jupiter', 'Saturn'])
5 internal = set(['Mercury', 'Venus', 'Earth', 'Mars'])
6
7 objects.update(internal)
8 print(objects)
9 print(internal)
```

```
1 {'Mars', 'Jupiter', 'Earth', 'Mercury', 'Saturn',
  'Venus'}
2 {'Mars', 'Earth', 'Mercury', 'Venus'}
```

Functions (subroutines)

Defining simple function

```
1 def add(x, y):  
2     z = x + y  
3     return z  
4  
5 a = add(2, 3)  
6 print(a)      # 5  
7  
8 q = add(23, 19)  
9 print(q)      # 42
```

The function definition starts with the word “dev” followed by the name of the function (“add” in our example), followed by the list of parameters

in a pair of parentheses, followed by a colon “:”. Then the body of the function is indented to the right. The depth of indentation does not matter

but it must be the same for all the lines of the function. When we stop the indentation and start a new expression on the first column, that’s what tells

Python that the function defintion has ended.

Defining a function

```
1 def sendmail(From, To, Subject, Content):  
2     print('From:', From)  
3     print('To:', To)  
4     print('Subject:', Subject)
```

```
5     print('')
6     print(Content)
7
8 sendmail('gabor@szabgab.com',
9       'szabgab@gmail.com',
10      'self message',
11      'Has some content too')
```

Positional parameters.

Parameters can be named

```
1 def sendmail(From, To, Subject, Content):
2     print('From:', From)
3     print('To:', To)
4     print('Subject:', Subject)
5     print('')
6     print(Content)
7
8 sendmail(
9     Subject = 'self message',
10    Content = 'Has some content too',
11    From = 'gabor@szabgab.com',
12    To = 'szabgab@gmail.com',
13 )
```

The parameters of every function can be passed either as positional parameters or as named parameters.

Mixing positional and named parameters

```
1 def sendmail(From, To, Subject, Content):
2     print('From:', From)
3     print('To:', To)
4     print('Subject:', Subject)
5     print('')
6     print(Content)
7
8 sendmail(
9     Subject = 'self message',
10    Content = 'Has some content too',
11    To = 'szabgab@gmail.com',
12    'gabor@szabgab.com',
13 )
```

```
1 def sendmail(From, To, Subject, Content):
2     print('From:', From)
3     print('To:', To)
4     print('Subject:', Subject)
5     print('')
6     print(Content)
7
8 sendmail(
9     'gabor@szabgab.com',
10    Subject = 'self message',
11    Content = 'Has some content too',
12    To = 'szabgab@gmail.com',
13 )
```

```
1 File
"examples/functions/named_and_positional_params.py", line
14
2     'gabor@szabgab.com',
3     ^
4 SyntaxError: positional argument follows keyword
argument
```

Default values

```
1 def prompt(question, retry=3):
2     while retry > 0:
3         inp = input('{} ({})'.format(question,
```

```
retry)
4         if inp == 'my secret':
5             return True
6         retry -= 1
7     return False
8
9 print(prompt("Type in your password"))
10
11 print(prompt("Type in your secret", 1))
```

Function parameters can have default values. In such case the parameters are optional.

In the function declaration, the parameters with the default values must come last.

In the call, the order among these arguments does not matter, and they are optional anyway.

Several defaults, using names

Parameters with defaults must come at the end of the parameter declaration.

```
1 def f(a, b=2, c=3):
2     print(a, b , c)
3
4 f(1)          # 1 2 3
5 f(1, b=0)      # 1 0 3
6 f(1, c=0)      # 1 2 0
7 f(1, c=0, b=5) # 1 5 0
8
9 # f(b=0, 1)
10 # would generate:
11 # SyntaxError: non-keyword arg after keyword arg
```

```
12  
13 f(b=0, a=1)      # 1 0 3
```

There can be several parameters with default values.
They are all optional and can be given in any order after the positional arguments.

Arbitrary number of arguments *

The values arrive as tuple.

```
1 def mysum(*numbers):  
2     print(numbers)  
3     total = 0  
4     for s in numbers:  
5         total += s  
6     return total  
7  
8 print(mysum(1))  
9 print(mysum(1, 2))  
10 print(mysum(1, 1, 1))  
11  
12 x = [2, 3, 5, 6]  
13 print(mysum(*x))
```

```
1 (1,)  
2 1  
3 (1, 2)  
4 3  
5 (1, 1, 1)  
6 3
```

Fixed parameters before the others

The `*numbers` argument can be preceded by any number of regular arguments

```
1 def mysum(op, *numbers):
2     print(numbers)
3     if op == '+':
4         total = 0
5     elif op == '*':
6         total = 1
7     else:
8         raise Exception('invalid operator
{}{}'.format(op))
9
10    for s in numbers:
11        if op == '+':
12            total += s
13        elif op == '*':
14            total *= s
15
16    return total
17
18 print(mysum('+', 1))
19 print(mysum('+', 1, 2))
20 print(mysum('+', 1, 1, 1))
21 print(mysum('*', 1, 1, 1))
```

```
1 (1,)
2 1
3 (1, 2)
4 3
5 (1, 1, 1)
6 3
7 (1, 1, 1)
8 1
```

Arbitrary key-value pairs in parameters **

```
1 def f(**kw):
2     print(kw)
3
4 f(a = 23, b = 12)
```

```
1 {'a': 23, 'b': 12}
```

Extra key-value pairs in parameters

```
1 def f(name, **kw):
2     print(name)
3     print(kw)
4
5 f(name="Foo", a = 23, b = 12)
6
7 # Foo
8 # {'a': 23, 'b': 12}
```

```
1 Foo
2 {'a': 23, 'b': 12}
```

Every parameter option

```
1 def f(op, count = 0, *things, **kw):
2     print(op)
3     print(count)
4     print(things)
5     print(kw)
6
7 f(2, 3, 4, 5, a = 23, b = 12)
8
9 # 2
10 # 3
11 # (4, 5)
12 # {'a': 23, 'b': 12}
```

Duplicate declaration of functions (multiple signatures)

```
1 def add(x, y):
2     return x*y
3
4 print(add(2, 3)) # 6
5
6 def add(x):
7     return x+x
8
9 # add(2, 3)
10 # TypeError: add() takes exactly 1 argument (2 given)
11
12 print(add(2)) # 4
```

The second declaration silently overrides the first declaration.

- [pylint](#) can find such problems, along with a bunch of others.

Recursive factorial

```
1 n! = n * (n-1) ... * 1
2
3 0! = 1
4 n! = n * (n-1) !
5
6 f(0) = 1
7 f(n) = n * f(n-1)
```

```
1 def f(n):
2     if n == 0:
3         return 1
4     return n * f(n-1)
5
6 print(f(1)) # 1
7 print(f(2)) # 2
8 print(f(3)) # 6
9 print(f(4)) # 24
```

Recursive Fibonacci

```
1 fib(1) = 1
2 fib(2) = 1
3 fib(n) = fib(n-1) + fib(n-2)
```

```
1 def fib(n):
2     if n == 1:
3         return 1
4     if n == 2:
5         return 1
6     return fib(n-1) + fib(n-2)
7
8 print(3, fib(3))    # 2
9 print(30, fib(30)) # 832040
```

Python also supports recursive functions.

Non-recursive Fibonacci

```
1 def fib(n):
2     if n == 1:
3         return [1]
4     if n == 2:
5         return [1, 1]
6     fibs = [1, 1]
7     for i in range(2, n):
8         fibs.append(fibs[-1] + fibs[-2])
9     return fibs
10
11 print(fib(1)) # [1]
12 print(fib(2)) # [1, 1]
13 print(fib(3)) # [1, 1, 2]
14 print(fib(10)) # [1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

Unbound recursion

- In order to protect us from unlimited recursion, Python limits the depth of recursion:

```
1 def recursion(n):
2     print(f"In recursion {n}")
3     recursion(n+1)
4
5 recursion(1)
```

```
1 ...
2 In recursion 995
3 In recursion 996
4 Traceback (most recent call last):
5   File "recursion.py", line 7, in <module>
6     recursion(1)
7   File "recursion.py", line 5, in recursion
8     recursion(n+1)
9   File "recursion.py", line 5, in recursion
10    recursion(n+1)
11  File "recursion.py", line 5, in recursion
12    recursion(n+1)
13  [Previous line repeated 992 more times]
14  File "recursion.py", line 4, in recursion
15    print(f"In recursion {n}")
16 RecursionError: maximum recursion depth exceeded while
calling a Python object
```

Variable assignment and change - Immutable

Details showed on the next slide

```
1 a = 42      # number or string
2 b = a        # This is a copy
3 print(a)    # 42
4 print(b)    # 42
5 a = 1
6 print(a)    # 1
7 print(b)    # 42
```

```
8
9 a = (1, 2)      # tuple
10 b = a          # this is a copy
11 print(a)       # (1, 2)
12 print(b)       # (1, 2)
13 # a[0] = 42   TypeError: 'tuple' object does not
support item assignment
14 a = (3, 4, 5)
15 print(a)       # (3, 4, 5)
16 print(b)       # (1, 2)
```

Variable assignment and change - Mutable

```
1 a = [5, 6]
2 b = a          # this is a copy of the *reference* only
3                      # if we change the list in a, it will
4                      # change the list connected to b as well
5 print(a)       # [5, 6]
6 print(b)       # [5, 6]
7 a[0] = 1
8 print(a)       # [1, 6]
9 print(b)       # [1, 6]
10
11
12 a = {'name' : 'Foo'}
13 b = a          # this is a copy of the *reference* only
14                      # if we change the dictionary in a, it
will
15                      # change the dictionary connected to b as
well
16 print(a)       # {'name' : 'Foo'}
17 print(b)       # {'name' : 'Foo'}
18 a['name'] = 'Jar Jar'
19 print(a)       # {'name' : 'Jar Jar'}
20 print(b)       # {'name' : 'Jar Jar'}
```

Parameter passing of functions

```
1 x = 3
2
3 def inc(n):
```

```
4     n += 1
5     return n
6
7 print(x)          # 3
8 print(inc(x))    # 4
9 print(x)          # 3
```

Passing references

```
1 numbers = [1, 2, 3]
2
3 def update(x):
4     x[0] = 23
5
6 def change(y):
7     y = [5, 6]
8     return y
9
10 print(numbers)      # [1, 2, 3]
11
12 update(numbers)
13 print(numbers)      # [23, 2, 3]
14
15 print(change(numbers)) # [5, 6]
16 print(numbers)      # [23, 2, 3]
```

Function documentation

```
1 def f(name):
2     """
3     The documentation
4     should have more than one lines.
5     """
6     print(name)
7
8
9 f("hello")
10 print(f.__doc__)
```

Immediately after the definition of the function, you can add a string - it can be a “”” string to spread multiple lines - that will include the documentation of the function. This string can be accessed via the **doc** (2+2 underscores) attribute of the function. Also, if you ‘import’ the file - as a module - in the interactive prompt of Python, you will be able to read this documentation via the **help()** function.
help(mydocs) or **help(mydocs.f)**
in the above case.

Sum ARGV

```
1 import sys
2
3 def mysum(*numbers):
4     print(numbers)
5     total = 0
6     for s in numbers:
7         total += s
8     return total
9
10 v = [int(x) for x in sys.argv[1:] ]
11 r = mysum(*v )
12 print(r)
```

Copy-paste code

```
1 a = [2, 3, 93, 18]
2 b = [27, 81, 11, 35]
3 c = [32, 105, 1]
4
5 total_a = 0
6 for v in a:
7     total_a += v
8 print("sum of a: {} average of a: {}".format(total_a,
```

```
total_a / len(a)))
9
10 total_b = 0
11 for v in b:
12     total_b += v
13 print("sum of b: {} average of b: {}".format(total_b,
14 total_b / len(b)))
15 total_c = 0
16 for v in c:
17     total_c += v
18 print("sum of c: {} average of c: {}".format(total_c,
19 total_c / len(a)))
```

```
1 sum of a: 116 average of a: 29.0
2 sum of b: 154 average of b: 38.5
3 sum of c: 138 average of c: 34.5
```

Did you notice the bug?

Copy-paste code fixed

```
1 a = [2, 3, 93, 18]
2 b = [27, 81, 11, 35]
3 c = [32, 105, 1]
4
5 def calc(numbers):
6     total = 0
7     for v in numbers:
8         total += v
9     return total, total / len(numbers)
10
11 total_a, avg_a = calc(a)
12 print("sum of a: {} average of a: {}".format(total_a,
13 avg_a))
14 total_b, avg_b = calc(b)
15 print("sum of b: {} average of b: {}".format(total_b,
16 avg_b))
17
```

```
18 total_c, avg_c = calc(c)
19 print("sum of c: {} average of c: {}".format(total_c,
avg_c))
```

```
1 sum of a: 116 average of a: 29.0
2 sum of b: 154 average of b: 38.5
3 sum of c: 138 average of c: 46.0
```

Copy-paste code further improvement

```
1 data = {
2     'a': [2, 3, 93, 18],
3     'b': [27, 81, 11, 35],
4     'c': [32, 105, 1],
5 }
6
7 def calc(numbers):
8     total = 0
9     for v in numbers:
10         total += v
11     return total, total / len(numbers)
12
13 total = {}
14 avg = {}
15 for name, numbers in data.items():
16     total[name], avg[name] = calc(numbers)
17     print("sum of {}: {} average of {}:{}".
18          format(name, total[name], name, avg[name]\
```

Palindrome

An iterative and a recursive solution

```
1 def is_palindrome(s):
2     if s == '':
3         return True
4     if s[0] == s[-1]:
5         return is_palindrome(s[1:-1])
6     return False
```

```

7
8 def iter_palindrome(s):
9     for i in range(0, int(len(s) / 2)):
10         if s[i] != s[-(i+1)]:
11             return False
12     return True
13
14 print(is_palindrome(''))      # True
15 print(is_palindrome('a'))     # True
16 print(is_palindrome('ab'))    # False
17 print(is_palindrome('aa'))    # True
18 print(is_palindrome('aba'))   # True
19 print(is_palindrome('abc'))   # False
20
21 print()
22 print(iter_palindrome(''))    # True
23 print(iter_palindrome('a'))    # True
24 print(iter_palindrome('ab'))   # False
25 print(iter_palindrome('aa'))   # True
26 print(iter_palindrome('aba'))  # True
27 print(iter_palindrome('abc'))  # False

```

Exercise: statistics

Write a function that will accept any number of numbers and return a list of values:

- The sum
- Average
- Minimum
- Maximum

Exercise: recursive

Give a bunch of files that has list of requirement in them. Process them recursively and print the resulting full list of requirements

```
1 b  
2 c  
3 d
```

```
1 e  
2 d
```

```
1 f  
2 g
```

```
1 $ python traversing_dependency_tree.py a  
2  
3 Processing a  
4 Processing b  
5 Processing e  
6 Processing d  
7 Processing c  
8 Processing f  
9 Processing g  
10 Processing d
```

Exercise: Tower of Hanoi

[Tower of Hanoi](#)

Exercise: Merge and Bubble sort

- Implement [bubble sort](#)
- Implement [merge sort](#)

Solution: statistics

```
1 def stats(*numbers):  
2     total = 0  
3  
4     average = None    # there might be better solutions
```

```

here!
5     minx = None
6     maxx = None
7
8     for val in numbers:
9         total += val
10        if minx == None:
11            minx = maxx = val
12        if minx > val:
13            minx = val
14        if maxx < val:
15            maxx = val
16
17    if len(numbers):
18        average = total / len(numbers)
19
20
21    return total, average, minx, maxx
22
23
24 ttl, avr, smallest, largest = stats(3, 5, 4)
25
26 print(ttl)
27 print(avr)
28 print(smallest)
29 print(largest)

```

Solution: recursive

```

1 import sys
2 import os
3
4 if len(sys.argv) < 2:
5     exit("Usage: {} NAME".format(sys.argv[0]))
6
7 start = sys.argv[1]
8
9 def get_dependencies(name):
10    print("Processing {}".format(name))
11
12    deps = set(name)
13    filename = name + ".txt"
14    if not os.path.exists(filename):

```

```

15     return deps
16
17     with open(filename) as fh:
18         for line in fh:
19             row = line.rstrip("\n")
20             deps.add(row)
21             deps.update(get_dependencies(row))
22
23     return deps
24
25 dependencies = get_dependencies(start)
26 print(dependencies)

```

Solution: Tower of Hanoi

```

1 def check():
2     for loc in hanoi.keys():
3         if hanoi[loc] != sorted(hanoi[loc],
reverse=True):
4             raise Exception(f"Incorrect order in
{loc}: {hanoi[loc]}")
5
6 def move(depth, source, target, helper):
7     if depth > 0:
8         move(depth-1, source, helper, target)
9
10    val = hanoi[source].pop()
11    hanoi[target].append(val)
12    print(f"Move {val} from {source} to {target}
Status A:{str(hanoi['A']):10}\n
B:{str(hanoi['B']):10}  C:{str(hanoi['C']):10}")
13    check()
14
15    move(depth-1, helper, target, source)
16    check()
17
18
19 hanoi = {
20     'A': [4, 3, 2, 1],
21     'B': [],
22     'C': [],
23 }
24
25 check()

```

```
26 move(len(hanoi['A']), 'A', 'C', 'B')
27 check()
```

Solution: Merge and Bubble sort

```
1 def recursive_bubble_sort(data):
2     data = data[:]
3     if len(data) == 1:
4         return data
5
6     last = data.pop()
7     sorted_data = recursive_bubble_sort(data)
8     for i in range(len(sorted_data)):
9         if last > sorted_data[i]:
10            sorted_data.insert(i, last)
11            break
12        else:
13            sorted_data.append(last)
14    return sorted_data
15
16 def iterative_bubble_sort(data):
17     data = data[:]
18     for end in (range(len(data)-1, 0, -1)):
19         for i in range(end):
20             if data[i] < data[i+1]:
21                 data[i], data[i+1] = data[i+1],
22                 data[i]
23
24
25 old = [1, 5, 2, 4, 8]
26 new1 = recursive_bubble_sort(old)
27 new2 = iterative_bubble_sort(old)
28 print(old)
29 print(new1)
30 print(new2)
```

Modules

Before modules

```
1 def add(a, b):  
2     return a + b  
3  
4  
5 z = add(2, 3)  
6 print(z)      # 5
```

Create modules

A module is just a Python file with a set of functions that us usually not used by itself. For example the “my_calculator.py”.

```
1 def add(a, b):  
2     return a + b
```

A user made module is loaded exactly the same way as the built-in module.

The functions defined in the module are used as if they were methods.

```
1 import my_calculator  
2  
3 z = my_calculator.add(2, 3)
```

```
4  
5 print(z) # 5
```

We can import specific functions to the current name space (symbol table) and then we don't need to prefix it with the name of the file every time we use it. This might be shorter writing, but if we import the same function name from two different modules then they will overwrite each other. So I usually prefer loading the module as in the previous example.

```
1 from my_calculator import add  
2  
3 print(add(2, 3)) # 5
```

path to load modules from - The module search path

1. The directory where the main script is located.
2. The directories listed in PYTHONPATH environment variable.
3. Directories of standard libraries.
4. Directories listed in .pth files.
5. The site-packages home of third-party extensions.

sys.path - the module search path

```
1 import sys  
2  
3 print(sys.path)
```

```
1 ['/Users/gabor/work/training/python/examples/package',  
2  '/Users/gabor/python/lib/python2.7/site-  
packages/crypto-1.1.0-py2.7.egg',  
3  ...  
4  '/Library/Python/2.7/site-packages',  
5  '/usr/local/lib/python2.7/site-packages']  
5 [Finished in 0.112s]
```

Flat project directory structure

If our executable scripts and our modules are all in the same directory then we don't have to worry ad the directory of the script is included in the list of places where "import" is looking for the files to be imported.

```
1 project/  
2     script_a.py  
3     script_b.py  
4     my_module.py
```

Absolute path

If we would like to load a module that is not installed in one of the standard locations, but we know where it is located on our disk,

we can set the "sys.path" to the absolute path to this directory. This works on the specific computer, but if you'd like to distribute

the script to other computers you'll have to make sure the module to be loaded is installed in the same location or you'll have to update the script to point to the location of the module in each computer. Not an ideal solution.

```
1 import sys
2 sys.path.insert(0, "/home/foobar/python/libs")
3
4 # import module_name
```

Relative path

```
1 ../project_root/
2     bin/relative_path.py
3     lib/my_module.py
```

We can use a directory structure that is more complex than the flat structure we had earlier. In this case the location of the modules relatively to the scripts

is fixed. In this case it is “`../lib`”. We can compute the relative path in each of our scripts. That will ensure we pick up the right module every time we run the script.

Regardless of the location of the whole project tree.

```
1 print("Importing my_module")
```

```
1 import os, sys
2
3 # import my_module    # ImportError: No module named
my_module
4
5 print(__file__)      # examples/sys/bin/relative_path.py
6 project_root =
os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
)
7
8 mypath = os.path.join(project_root, 'lib')
9 print(mypath)  #
/Users/gabor/work/training/python/examples/sys/../lib
10 sys.path.insert(0, mypath)
```

```
11  
12 import my_module      # Importing my_module
```

Python modules are compiled

When libraries are loaded they are automatically compiled to .pyc files.

This provides moderate code-hiding and load-time speed-up. Not run-time speed-up.

Starting from Python 3.2 the pyc files are saved in the __pycache__ directory.

How “import” and “from” work?

1. Find the file to load.
2. Compile to bytecode if necessary and save the bytecode if possible.
3. Run the code of the file loaded.
4. Copy names from the imported module to the importing namespace.

Runtime loading of modules

```
1 def hello():  
2     print("Hello World")  
3  
4 print("Loading mygreet")
```

```
1 print("Start running")    # Start running  
2  
3 import mygreet           # Loading mygreet  
4  
5 mygreet.hello()          # Hello World  
6  
7 print("DONE")            # DONE
```

Conditional loading of modules

```
1 import random
2
3 print("Start running")
4 name = input("Your name:")
5
6 if name == "Foo":
7     import mygreet
8     mygreet.hello()
9 else:
10    print('No loading')
11
12
13 print("DONE")
```

Duplicate importing of functions

```
1 from mycalc import add
2 print(add(2, 3))  # 5
3
4 from mymath import add
5 print(add(2, 3))  # 6
6
7
8 from mycalc import add
9 print(add(2, 3))  # 5
```

The second declaration silently overrides the first declaration.

[pylint](#) can find such problems, along with a bunch of others.

Script or library

We can have a file with all the functions implemented and then launch the run() function only if the file was executed as a stand-alone script.

```
1 def run():
2     print("run in ", __name__)
3
4 print("Name space in mymodule.py ", __name__)
5
6 if __name__ == '__main__':
7     run()
```

```
1 $ python mymodule.py
2 Name space in mymodule.py __main__
3 run in __main__
```

Script or library - import

If it is imported by another module then it won't run automatically. We have to call it manually.

```
1 import mymodule
2
3 print("Name space in import_mymodule.py ", __name__)
4 mymodule.run()
```

```
1 $ python import_mymodule.py
2 Name space in mymodule.py mymodule
3 Name space in import_mymodule.py __main__
4 run in mymodule
```

Script or library - from import

```
1 from mymodule import run
2
3 print("Name space in import_mymodule.py ", __name__)
4 run()
```

```
1 $ python import_from_mymodule.py
2 Name space in mymodule.py mymodule
3 Name space in import_mymodule.py __main__
4 run in mymodule
```

assert to verify values

```
1 def add(x, y):
2     return x * y
3
4 for x, y, z in [(2, 2, 4), (9, 2, 11), (2, 3, 5)]:
5     print(f"add({x}, {y}) == {z}")
6     if add(x, y) != z:
7         raise Exception(f"add({x}, {y}) != {z}")
8     #raise AssertionError
```

```
1 add(2, 2) == 4
2 add(9, 2) == 11
3 Traceback (most recent call last):
4   File "examples/functions/raise_exception.py", line 7,
in <module>
5       raise Exception(f"add({x}, {y}) != {z}")
6 Exception: add(9, 2) != 11
```

```
1 def add(x, y):
2     return x * y
3
4 for x, y, z in [(2, 2, 4), (9, 2, 11), (2, 3, 5)]:
5     print(f"add({x}, {y}) == {z}")
6     assert add(x, y) == z
```

```
1 add(2, 2) == 4
2 add(9, 2) == 11
3 Traceback (most recent call last):
4   File "examples/functions/assert.py", line 6, in
<module>
5     assert add(x, y) == z
6 AssertionError
```

mycalc as a self testing module

```
1 import mycalc
2 print(mycalc.add(19, 23))
```

```
1 $ python use_mycalc.py
2 42
```

```
1 def test_add():
2     print('Testing {}'.format(__file__))
3     assert add(1, 1) == 2
4     assert add(-1, 1) == 0
5     # assert add(-99, 1) == 0 # AssertionError
6
7 def add(a, b):
8     return a + b
9
10 if __name__ == '__main__':
11     test_add()
```

```
1 $ python mycalc.py
2 Self testing mycalc.py
```

doctest

```
1 def fib(n):
2     """
3     Before the tests
4     >>> fib(3)
5     2
```

```
6      >>> fib(10)
7      55
8      >>> [fib(n) for n in range(11)]
9      [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
10
11     >>> fib(11)
12     89
13     After the tests
14     ''
15     values = [0, 1]
16
17     if n == 11:
18         return 'bug'
19
20     while( n > len(values) -1 ):
21         values.append(values[-1] + values[-2])
22     return values[n]
23
24 #if __name__ == "__main__":
25 #    import doctest
26 #    doctest.testmod()
```

```
1 python -m doctest fibonacci_doctest.py
```

```
1 python examples/functions/fibonacci_doctest.py
2
3
*****
4 File ".../examples/functions/fibonacci_doctest.py",
line 12, in __main__.fib
5 Failed example:
6     fib(11)
7 Expected:
8     89
9 Got:
10    'bug'
11
*****
12 1 items had failures:
13    1 of   4 in __main__.fib
14 ***Test Failed*** 1 failures.
```

doctest

Scope of import

```
1 def div(a, b):  
2     return a/b
```

```
1 from __future__ import print_function  
2 from __future__ import division  
3  
4 import mydiv  
5  
6 print(mydiv.div(3, 2))      # 1  
7  
8 print(3/2)                  # 1.5
```

The importing of functions, and the changes in the behavior of the compiler are file specific.

In this case the change in the behavior of division is only visible in the division.py script, but not in the mydiv.py module.

Export import

- from mod import a,b,_c - import ‘a’, ‘b’, and ‘_c’ from ‘mod’
- from mod import * - import every name listed in **all** of ‘mod’ if **all** is available.
- from mod import * - import every name that does NOT start with _ (if **all** is not available)

- import mod - import ‘mod’ and make every name in ‘mod’ accessible as ‘mod.a’, and ‘mod._c’
-

```
1 def a():
2     return "in a"
3
4 b = "value of b"
5
6 def _c():
7     return "in _c"
8
9 def d():
10    return "in d"
```

```
1 from my_module import a,b,_c
2
3 print(a())      # in a
4 print(b)        # value of b
5 print(_c())     # in _c
6
7 print(d())
8 # Traceback (most recent call last):
9 #   File ".../examples/modules/x.py", line 7, in
<module>
10 #       print(d())
11 # NameError: name 'd' is not defined
```

```
1 from my_module import *
2
3 print(a())      # in a
4 print(b)        # value of b
5
6 print(d())      # in d
7
8
9 print(_c())
10
11 # Traceback (most recent call last):
12 #   File ".../examples/modules/y.py", line 9, in
<module>
```

```
13 #     print(_c())      # in _c
14 # NameError: name '_c' is not defined
```

Export import with all

```
1 __all__ = ['a', '_c']
2
3 def a():
4     return "in a"
5
6 b = "value of b"
7
8 def _c():
9     return "in _c"
10
11 def d():
12     return "in d"
```

```
1 from my_module2 import *
2
3 print(a())      # in a
4 print(_c())      # in _c
5
6 print(b)
7
8 # Traceback (most recent call last):
9 #   File ".../examples/modules/z.py", line 7, in
<module>
10 #       print(b)          # value of b
11 # NameError: name 'b' is not defined
```

import module

```
1 import my_module
2
3 print(my_module.a())      # in a
4 print(my_module.b)          # value of b
5 print(my_module._c())      # in _c
6 print(my_module.d())          # in d
```

Execute at import time

```
1 import lib
2
3 print("Hello")
```

```
1 print("import lib")
2
3 def do_something():
4     print("do something")
```

```
1 import lib
2 Hello
```

Import multiple times

```
1 import one
2 import two
3
4 print("Hello")
```

```
1 import common
2 print("loading one")
```

```
1 import common
2 print("loading two")
```

```
1 print("import common")
```

```
1 import common
2 loading one
3 loading two
4 Hello
```

Exercise: Number guessing

Take the number guessing game from the earlier chapter and move the internal while() loop to a function.

Once that's done, move the function out to a separate file and use it as a module.

Exercises: Scripts and modules

Take the number guessing game: if I run it as a script execute the whole game with repeated hidden numbers.

If I load it as a module, then let me call the function that runs a single game with one hidden number.

We should be able to even pass the hidden number as a parameter.

Exercise: Module my_sum

- Create a file called `my_simple_math.py` with two functions: `div(a, b)`, `add(a, b)`, that will divide and add the two numbers respectively.
- Add another two functions called `test_div` and `test_add` that will test the above two functions using assert.
- Add code that will run the tests if someone execute `python my_simple_math.py` running the file as if it was a script.
- Create another file called `use_my_simple_math.py` that will use the functions from `my_math` module to calculate $2 + 5 * 7$
- Make sure when you run `python use_my_simple_math.py` the tests won't run.

- Add documentation to the “add” and “div” functions to examples that can be used with doctest.
- Can you run the tests when the file is loaded as a module?

Exercise: Convert your script to module

- Take one of your real script (from work). Create a backup copy.
- Change the script so it can be import-ed as a module and then it won’t automatically execute anything, but that it still works when executed as a script.
- Add a new function to it called `self_test` and in that function add a few test-cases to your code using ‘assert’.
- Write another script that will load your real file as a module and will run the `self_test`.
- Let me know what are the difficulties!

Exercise: Add doctests to your own code

- Pick a module from your own code and create a backup copy. (from work)
- Add a function called ‘`self_test`’ that uses ‘assert’ to test some of the real functions of the module.
- Add code that will run the ‘`self_test`’ when the file is executed as a script.
- Add documentation to one of the functions and convert the ‘assert’-based tests to doctests.
- Convert the mechanism that executed the ‘`self_test`’ to run the doctests as well.
- Let me know what are the difficulties!

Solution: Module my_sum

```
1 def div(a, b):
2     """
3     >>> div(8, 2)
4     4
5     """
6     return a/b
7
8 def add(a, b):
9     """
10    >>> add(2, 2)
11    4
12    """
13    return a * b    # bug added on purpose!
14
15 def test_div():
16     assert div(6, 3) == 2
17     assert div(0, 10) == 0
18     assert div(-2, 2) == -1
19     #assert div(10, 0) == ???
20
21 def test_add():
22     assert add(2, 2) == 4
23     #assert add(1, 1) == 2
24
25
26 if __name__ == "__main__":
27     test_div()
28     test_add()
```

```
1 import my_simple_math
2 print(my_simple_math.my_sum(2, 3, 5))
3
4 print(dir(my_simple_math))
5 #my_sum_as_function.test_my_sum()
```

Regular Expressions

What are Regular Expressions (aka. Regexes)?

- An idea on how to match some pattern in some text.
- A tool/language that is available in many places.
- Has many different “dialects”
- Has many different modes of processing.
- The grand concept is the same.
- Uses the following symbols:

`^ $. * + ? ^ $ | - \ \d \s \w \A \Z \1 \2 \3`

What are Regular Expressions good for?

- Decide if a string is part of a larger string.
- Validate the format of some value (string) (e.g. is it a decimal number?, is it a hex?)
- Find if there are repetitions in a string.
- Analyze a string and fetch parts of if given some loose description.
- Cut up a string into parts.
- Change parts of a string.

Examples

¹ Is the input given by the user a number?

²

```
3 (BTW which one is a number: 23, 2.3, 2.3.4, 2.4e3,  
abc ?)  
4  
5 Is there a word in the file that is repeated 3 or more  
times?  
6  
7 Replaces all occurrences of Python or python by Java  
...  
8 ... but avoid replacing Monty Python.  
9  
10  
11 Given a text message fetch all the phone numbers:  
12 Fetch numbers that look like 09-1234567  
13 then also fetch +972-2-1234567  
14 and maybe also 09-123-4567  
15  
16  
17 Check if in a given text passing your network there  
are credit card numbers....  
18  
19  
20 Given a text find if the word "password" is in it and  
fetch the surrounding text.  
21  
22  
23 Given a log file like this:  
24  
25 [Tue Jun 12 00:01:00 2019] - (3423) - INFO - ERROR log  
restarted  
26 [Tue Jun 12 09:08:17 2019] - (3423) - INFO - System  
starts to work  
27 [Tue Jun 13 08:07:16 2019] - (3423) - ERROR -  
Something is wrong  
28  
29 provide statistics on how many of the different levels  
of log messages  
30 were seen. Separate the log messages into files.
```

Where can I use it ?

- grep, egrep
- Unix tools such as sed, awk, procmail

- vi, emacs, other editors
- text editors such as Multi-Edit
- .NET languages: C#, C++, VB.NET
- Java
- Perl
- **Python**
- PHP
- Ruby
- ...
- Word, Open Office ...
- PCRE

grep

grep gets a regex and one or more files. It goes over line-by-line all the files and displays the lines where the regex matched.
A few examples:

```

1 grep python file.xml      # lines that have the string
python in them in file.xml.
2 grep [34] file.xml        # lines that have either 3 or 4
(or both) in file.xml.
3 grep [34] *.xml           # lines that have either 3 or 4
(or both) in every xml file.
4 grep [0-9] *.xml          # lines with a digit in them.
5 egrep '\b[0-9]\b' *.xml    # only highlight digits that
are at the beginning of a number
6 r.
```

Regexes first match

```

1 import re
2
```

```
3 text = 'The black cat climed'
4 match = re.search(r'lac', text)
5 if match:
6     print("Matching")          # Matching
7     print(match.group(0))      # lac
8
9 match = re.search(r'dog', text)
10 if match:
11     print("Matching")
12 else:
13     print("Did NOT match")
14     print(match)             # None
```

The search method returns an object or **None**, if it could not find any match.

If there is a match you can call the **group()** method. Passing 0 to it will return the actual substring that was matched.

Match numbers

```
1 import re
2
3 line = 'There is a phone number 12345 in this row and
an age: 23'
4
5 match = re.search(r'\d+', line)
6 if match:
7     print(match.group(0))    # 12345
```

Use raw strings for regular expression: r'a\d'. Especially because \ needs it.

- \d matches a digit.

- `+` is a quantifier and it tells `\d` to match one or more digits.

It matches the first occurrence.

Here we can see that the `group(0)` call is much more interesting than earlier.

Capture

```

1 import re
2
3 line = 'There is a phone number 12345 in this row and
an age: 23'
4
5 match = re.search(r'age: \d+', line)
6 if match:
7     print(match.group(0))    # age: 23
8
9
10 match = re.search(r'age: (\d+)', line)
11 if match:
12     print(match.group(0))   # age: 23
13     print(match.group(1))   # 23           the first group
of parentheses
14
15 print(match.groups())   # ('23',)
16 print(len(match.groups())) # 1

```

Parentheses in the regular expression can enclose any sub-expression.

Whatever this sub-expression matches will be saved and can be accessed using the `group()` method.

Capture more

```
1 import re
2
3 line = 'There is a phone number 12345 in this row and
an age: 23'
4
5 match = re.search(r'(\w+): (\d+)', line)
6 if match:
7     print(match.group(0))    # age: 23
8     print(match.group(1))    # age      the first group
of parentheses
9     print(match.group(2))    # 23       the second group
of parentheses
10
11    # print(match.group(3))  # IndexError: no such
group
12    print(match.groups())   # ('age', '23')
13    print(len(match.groups())) # 2
```

Some groups might match ‘’ or even not match at all, in which case we get None in the appropriate match.group() call and in the match.groups() call

Capture even more

```
1 import re
2
3 line = 'There is a phone number 12345 in this row and
an age: 23'
4
5 match = re.search(r'((\w+): (\d+))', line)
6 if match:
7     print(match.group(0))    # age: 23
8     print(match.group(1))    # age: 23
9     print(match.group(2))    # age
10    print(match.group(3))   # 23
11
```

```
12 print(match.groups()) # ('age: 23', 'age', '23')
13 print(len(match.groups())) # 3
```

findall

```
1 import re
2
3 line1 = 'There is a phone number 12345 in this row and
another 42 number'
4 numbers1 = re.findall(r'\d+', line1)
5 print(numbers1) # ['12345', '42']
6
7 line2 = 'There are no numbers in this row. Not even
one.'
8 numbers2 = re.findall(r'\d+', line2)
9 print(numbers2) # []
```

`re.findall` returns the matched substrings.

findall with capture

```
1 import re
2
3 line = 'There is a phone number 12345 in this row and
another 42 number'
4 match = re.search(r'\w+ \d+', line)
5 if match:
6     print(match.group(0)) # number 12345
7
8 match = re.search(r'\w+ (\d+)', line)
9 if match:
10    print(match.group(0)) # number 12345
11    print(match.group(1)) # 12345
12
13 matches = re.findall(r'\w+ \d+', line)
14 print(matches) # ['number 12345', 'another 42']
15
16 matches = re.findall(r'\w+ (\d+)', line)
17 print(matches) # ['12345', '42']
```

findall with capture more than one

```
1 import re
2
3 line = 'There is a phone number 12345 in this row and
another 42 number'
4 match = re.search(r'(\w+) (\d+)', line)
5 if match:
6     print(match.group(1))      # number
7     print(match.group(2))      # 12345
8
9 matches = re.findall(r'(\w+) (\d+)', line)
10 print(matches)  # [('number', '12345'), ('another', '42')]
```

If there are multiple capture groups then The returned list will consist of tuples.

Any Character

- . matches any one character except newline.

For example: #.#

```
1 import re
2
3 strings = [
4     'abc',
5     'text: #q#',
6     'str: #a#',
7     'text #b# more text',
8     '#a and this? #c#',
9     '#a and this? # c#',
10    '#@#',
11    '#.#',
12    '# #' ,
```

```
13     '# #'  
14     '# ##'  
15 ]  
16  
17 for s in strings:  
18     print('str: ', s)  
19     match = re.search(r'#.#+', s)  
20     if match:  
21         print('match:', match.group(0))
```

If `re.DOTALL` is given newline will be also matched.

Match dot

```
1 import re  
2  
3 cases = [  
4     "hello!",  
5     "hello world.",  
6     "hello. world",  
7     ".,",  
8 ]  
9  
10 for case in cases:  
11     print(case)  
12     match = re.search(r'\.', case)      # Match any  
character  
13     if match:  
14         print(match.group(0))  
15  
16 print("----")  
17  
18 for case in cases:  
19     print(case)  
20     match = re.search(r'\.\.', case)    # Match a dot  
21     if match:  
22         print(match.group(0))  
23  
24 print("----")  
25  
26 for case in cases:
```

```
27     print(case)
28     match = re.search(r'\.', case) # Match a dot
29     if match:
30         print(match.group(0))
```

Character classes

We would like to match any string that has any of the #a#, #b#, #c#, #d#, #e#, #f#, #@# or #.#

```
1 import re
2
3 strings = [
4     'abc',
5     'text: #q#',
6     'str: #a#',
7     'text #b# more text',
8     '#ab#',
9     '#@#',
10    '#.#',
11    '# #',
12    '##'
13    '###'
14 ]
15
16
17 for s in strings:
18     print('str: ', s)
19     match = re.search(r'#[abcdef@.]+', s)
20     if match:
21         print('match:', match.group(0))
```

```
1 r'#[abcdef@.]+'
2 r'#[a-f@.]+'
```

Common character classes

- \d digit: [0-9] Use stand alone: \d or as part of a bigger character class: [abc\d]
- \w word character: [0-9a-zA-Z_]
- \s white space: [\f\t\n\r] form-feed, tab, newline, carriage return and SPACE

Negated character class

- [^abc] matches any one character that is not ‘a’, not ‘b’ and not ‘c’.
- D not digit [^d]
- W not word character [^w]
- S not white space [^s]

Optional character

Match the word **color** or the word **colour**

1 Regex: r'colou?r'

1 Input: color
2 Input: colour
3 Input: colouur

Regex 0 or more quantifier

Any line with two - -es with anything in between.

1 Regex: r'-.*-'
2 Input: "ab"
3 Input: "ab - cde"
4 Input: "ab - qqqrq -"
5 Input: "ab -- cde"
6 Input: "--"

Quantifiers

Quantifiers apply to the thing in front of them

```
1 r'ax*a'      # aa, axa, axxa, axxxxa, ...
2 r'ax+a'      #      axa, axxa, axxxxa, ...
3 r'ax?a'      # aa, axa
4 r'ax{2,4}a'  #          axxa, axxxxa, axxxxxa
5 r'ax{3,}a'   #                  axxxxa, axxxxxa, ...
6 r'ax{17}a'   #
axxxxxxxxxxxxxxxxxx
```

*	0-
+	1-
?	0-1
	n-m
	n-
	n

Quantifiers limit

```
1 import re
2
3 strings = (
4     "axxxa",
5     "axxxxaa",
6     "axxxxxaa",
7 )
8
9 for text in strings:
10     match = re.search(r'ax{4}', text)
11     if match:
12         print("Match")
13         print(match.group(0))
14     else:
15         print("NOT Match")
```

Quantifiers on character classes

```
1 import re
2
3 strings = (
4     "-a-",
5     "-b-",
6     "-x-",
7     "-aa-",
8     "-ab-",
9     "--",
10 )
11
12 for line in strings:
13     match = re.search(r'-[abc]-', line)
14     if match:
15         print(line)
16 print('=====')
17
18 for line in strings:
19     match = re.search(r'-[abc]++', line)
20     if match:
21         print(line)
22 print('=====')
23
24 for line in strings:
25     match = re.search(r'-[abc]*-', line)
26     if match:
27         print(line)
```

Greedy quantifiers

```
1 import re
2
3 match = re.search(r'xa*', 'xaaab')
4 print(match.group(0))
5
6 match = re.search(r'xa*', 'xabxaab')
7 print(match.group(0))
8
9 match = re.search(r'a*', 'xabxaab')
10 print(match.group(0))
```

```
11
12 match = re.search(r'a*', 'aaaxaabxaab')
13 print(match.group(0))
```

They match ‘xaaa’, ‘xa’ and ‘’ respectively.

Minimal quantifiers

```
1 import re
2
3 match = re.search(r'a.*b', 'axbzb')
4 print(match.group(0))
5
6 match = re.search(r'a.*?b', 'axbzb')
7 print(match.group(0))
8
9
10 match = re.search(r'a.*b', 'axy121413413bq')
11 print(match.group(0))
12
13 match = re.search(r'a.*?b', 'axyb121413413q')
14 print(match.group(0))
```

Anchors

- A matches the beginning of the string
 - Z matches the end of the string
 - ^ matches the beginning of the row (see also re.MULTILINE)
 - \$ matches the end of the row but will accept a trailing newline (see also re.MULTILINE)
-

```
1 import re
2
3 lines = [
```

```
4     "text with cat in the middle",
5     "cat with dog",
6     "dog with cat",
7 ]
8
9 for line in lines:
10    if re.search(r'cat', line):
11        print(line)
12
13
14 print("---")
15 for line in lines:
16    if re.search(r'^cat', line):
17        print(line)
18
19 print("---")
20 for line in lines:
21    if re.search(r'\Acat', line):
22        print(line)
23
24 print("---")
25 for line in lines:
26    if re.search(r'cat$', line):
27        print(line)
28
29 print("---")
30 for line in lines:
31    if re.search(r'cat\z', line):
32        print(line)
```

```
1 text with cat in the middle
2 cat with dog
3 dog with cat
4 ---
5 cat with dog
6 ---
7 cat with dog
8 ---
9 dog with cat
10 ---
11 dog with cat
```

Anchors on both end

```
1 import re
2
3 strings = [
4     "123",
5     "hello 456 world",
6     "hello world",
7 ]
8
9 for line in strings:
10     if re.search(r'\d+', line):
11         print(line)
12
13 print('---')
14
15 for line in strings:
16     if re.search(r'^\d+$', line):
17         print(line)
18
19
20 print('---')
21
22 for line in strings:
23     if re.search(r'\A\d+\Z', line):
24         print(line)
```

```
1 123
2 hello 456 world
3 ---
4 123
5 ---
6 123
```

Match ISBN numbers

1

```
1 import re
2
```

```
3 strings = [
4     '99921-58-10-7',
5     '9971-5-0210-0',
6     '960-425-059-0',
7     '80-902734-1-6',
8     '85-359-0277-5',
9     '1-84356-028-3',
10    '0-684-84328-5',
11    '0-8044-2957-X',
12    '0-85131-041-9',
13    '0-943396-04-2',
14    '0-9752298-0-X',
15
16    '0-975229-1-X',
17    '0-9752298-10-X',
18    '0-9752298-0-Y',
19    '910975229-0-X',
20    '-----',
21    '00000000000000',
22 ]
23 for isbn in strings:
24     print(isbn)
25
26     if (re.search(r'^[\dX-]{13}$', isbn)):
27         print("match 1")
28
29     if (re.search(r'^\d{1,5}-\d{1,7}-\d{1,5}-[\dX]$', isbn) and len(isbn) == 13):
30         print("match 2")
```

Matching a section

```
1 import re
2
3 text = "This is <a string> with some sections <marked>
4 with special characters"
5 m = re.search(r'<.*>', text)
6 if m:
7     print(m.group(0))
```

Matching a section - minimal

```
1 import re
2
3 text = "This is <a string> with some sections <marked>
with special characters"
4
5 m = re.search(r'<.*?>', text)
6 if m:
7     print(m.group(0))
```

Matching a section negated character class

```
1 import re
2
3 text = "This is <a string> with some sections <marked>
with special characters"
4
5 m = re.search(r'<[^>]*>', text)
6 if m:
7     print(m.group(0))
```

DOTALL S (single line)

if re.DOTALL is given, . will match any character. Including newlines.

```
1 import re
2
3 line = 'Before <div>content</div> After'
4
5 text = '''
6 Before
7 <div>
8 content
9 </div>
10 After
11 '''
12
13 if (re.search(r'<div>.*</div>', line)):
14     print('line');
15 if (re.search(r'<div>.*</div>', text)):
16     print('text');
17
18 print('-' * 10)
19
20 if (re.search(r'<div>.*</div>', line, re.DOTALL)):
21     print('line');
22 if (re.search(r'<div>.*</div>', text, re.DOTALL)):
23     print('text');
```

MULTILINE M

if re.MULTILINE is given, ^ will match beginning of line and \$ will match end of line

```
1 import re
2
3 line = 'Start    blabla End'
4
5 text = '''
6 prefix
7 Start
8 blabla
9 End
10 postfix
11 '''
12
13 regex = r'^Start[\d\D]*End$'
14 m = re.search(regex, line)
15 if (m):
16     print('line')
17
18 m = re.search(regex, text)
19 if (m):
20     print('text')
21
22 print('-' * 10)
23
24 m = re.search(regex, line, re.MULTILINE)
25 if (m):
26     print('line')
27
28 m = re.search(regex, text, re.MULTILINE)
29 if (m):
30     print('text')
```

```
1 line
2 -----
3 line
4 text
```

```
1 re.MULTILINE | re.DOTALL
```

Two regex with logical or

All the rows with either ‘apple pie’ or ‘banana pie’ in them.

```
1 import re
2
3 strings = [
4     'apple pie',
5     'banana pie',
6     'apple'
7 ]
8
9 for s in strings:
10    #print(s)
11    match1 = re.search(r'apple pie', s)
12    match2 = re.search(r'banana pie', s)
13    if match1 or match2:
14        print('Matched in', s)
```

Alternatives

Alternatives

```
1 import re
2
3 strings = [
4     'apple pie',
5     'banana pie',
6     'apple'
7 ]
8
9 for s in strings:
10    match = re.search(r'apple pie|banana pie', s)
11    if match:
12        print('Matched in', s)
```

Grouping and Alternatives

Move the common part in one place and limit the alternation to the part within the parentheses.

```
1 import re
2
3 strings = [
4     'apple pie',
5     'banana pie',
6     'apple'
7 ]
8
9 for s in strings:
10     match = re.search(r'(apple|banana) pie', s)
11     if match:
12         print('Matched in', s)
```

Internal variables

```
1 import re
2
3 strings = [
4     'banana',
5     'apple',
6     'infinite loop',
7 ]
8
9 for s in strings:
10     match = re.search(r'(. )\1', s)
11     if match:
12         print(match.group(0), 'matched in', s)
13         print(match.group(1))
```

More internal variables

```
1 (.) (. )\2\1
2
3 (\d\d).*\1
4
```

```
5 (\d\d).*\1.*\1
6
7 (.{{5}}).*\1
```

Regex DNA

- DNA is built from G, A, T, C
 - Let's create a random DNA sequence
 - Then find the longest repeated sequence in it
-

```
1 import re
2 import random
3
4 chars = ['G', 'A', 'T', 'C']
5 dna = ''
6 for i in range(100):
7     dna += random.choice(chars)
8
9 print(dna)
10
11 '''
12 Generating regexes:
13
14 ([GATC]{1}).*\1
15 ([GATC]{2}).*\1
16 ([GATC]{3}).*\1
17 ([GATC]{4}).*\1
18 '''
19 length = 1
20 result = ''
21 while True:
22     regex = r'([GATC]{' + str(length) + r'}).*\1'
23     #print(regex)
24     m = re.search(regex, dna)
25     if m:
26         result = m.group(1)
27         length += 1
28     else:
29         break
30
```

```
31 print(result)
32 print(len(result))
```

Regex IGNORECASE

```
1 import re
2
3 s = 'Python'
4
5 if (re.search('python', s)):
6     print('python matched')
7
8 if (re.search('python', s, re.IGNORECASE)):
9     print('python matched with IGNORECASE')
```

Regex VERBOSE X

```
1 import re
2
3 email = "foo@bar.com"
4
5 m = re.search(r'\w[\w.-]*@\([\w-]+\.)+'
6 (com|net|org|uk|hu|il)', email)
7 if (m):
8     print('match')
9
10 m = re.search(r'''
11             \w[\w.-]*                      # username
12             \@
13             ([\w-]+\.)+                  # domain
14             (com|net|org|uk|hu|il)       # gTLD
15             ''', email, re.VERBOSE)
16 if (m):
17     print('match')
```

Substitution

```
1 import re
2
3 line = "abc123def"
4
5 print(re.sub(r'\d+', ' ', line)) # "abc def"
6 print(line) # "abc123def"
7
8 print(re.sub(r'x', 'y', line)) # "abc123def"
9 print(line) # "abc123def"
10
11 print(re.sub(r'([a-z]+)(\d+)([a-z]+)', r'\3\2\1', line)) # "def123abc"
12 print(re.sub(r'''
13 ([a-z]+)      # letters
14 (\d+)         # digits
15 ([a-z]+)      # more letters
16 ''', r'\3\2\1', line, flags=re.VERBOSE)) # "def123abc"
17
18 print(re.sub(r'...', 'x', line)) # "xxx"
19 print(re.sub(r'...', 'x', line, count=1)) # "x123def"
20
21 print(re.sub(r'(.)(.)', r'\2\1', line)) # "ba1c32edf"
22 print(re.sub(r'(.)(.)', r'\2\1', line, count=2)) # "ba1c23def"
```

findall capture

If there are parentheses in the regex, it will return tuples of the matches

```
1 import re
2
3 line = 'There is a phone number 83795 in this row and
another 42 number'
4 print(line)
5
6 search = re.search(r'(\d)(\d)', line)
7 if search:
```

```

8   print(search.group(1))    # 8
9   print(search.group(2))    # 3
10
11 matches = re.findall(r'(\d)(\d)', line)
12 if matches:
13     print(matches)  # [(8, '3'), ('7', '9'), ('4', '2')]
14
15 matches = re.findall(r'(\d)\D*', line)
16 if matches:
17     print(matches)  # [(8, '3', '7', '9', '5', '4', '2')]
18
19 matches = re.findall(r'(\d)\D*(\d?)', line)
20 print(matches)  # [(8, '3'), ('7', '9'), ('5', '4'), ('2', '')]
21
22 matches = re.findall(r'(\d).*?(\d)', line)
23 print(matches)  # [(8, '3'), ('7', '9'), ('5', '4')]
24
25 matches = re.findall(r'(\d+)\D+(\d+)', line)
26 print(matches)  # [(83795, '42')]
27
28 matches = re.findall(r'(\d+).*?(\d+)', line)
29 print(matches)  # [(83795, '42')]
30
31 matches = re.findall(r'\d', line)
32 print(matches)  # [8, '3', '7', '9', '5', '4', '2']

```

Fixing dates

In the input we get dates like this

2010-7-5 but we would like to make sure we have two digits
for both days and months: 2010-07-05

```

1 import re
2
3 def fix_date(date):
4     return re.sub(r'-(\d)\b', r'-0\1', date)
5
6

```

```
7 dates = {  
8     '2010-7-5' : '2010-07-05',  
9     '2010-07-5' : '2010-07-05',  
10    '2010-07-05' : '2010-07-05',  
11    '2010-7-15' : '2010-07-15',  
12 }  
13  
14 for original in sorted(dates.keys()):  
15     result = fix_date(original)  
16  
17     assert result == dates[original]  
18  
19     print(f"      old: {original}")  
20     print(f"      new: {result}")  
21     print(f" expected: {dates[original]}")  
22     print("")
```

```
1      old: 2010-07-05  
2      new: 2010-07-05  
3 expected: 2010-07-05  
4  
5      old: 2010-07-5  
6      new: 2010-07-05  
7 expected: 2010-07-05  
8  
9      old: 2010-7-15  
10     new: 2010-07-15  
11 expected: 2010-07-15  
12  
13     old: 2010-7-5  
14     new: 2010-07-05  
15 expected: 2010-07-05
```

Duplicate numbers

```
1 import re  
2  
3 text = "This is 1 string with 3 numbers: 34"  
4 new_text = re.sub(r'(\d+)', r'\1\1', text)  
5 print(new_text)    # This is 11 string with 33 numbers:  
3434  
6
```

```
7 double_numbers = re.sub(r'(\d+)', lambda match: str(2 *  
int(match.group(0))), text)  
8 print(double_numbers) # This is 2 string with 6  
numbers: 68
```

Remove spaces

```
1 line = " ab cd "  
2  
3 res = line.lstrip(" ")  
4 print(f"'{res}'") # 'ab cd '  
5  
6 res = line.rstrip(" ")  
7 print(f"'{res}'") # ' ab cd'  
8  
9 res = line.strip(" ")  
10 print(f"'{res}'") # 'ab cd'  
11  
12 res = line.replace(" ", "")  
13 print(f"'{res}'") # 'abcd'
```

Replace string in assembly code

```
1 mv A, R3  
2 mv R2, B  
3 mv R1, R3  
4 mv B1, R4  
5 add A, R1  
6 add B, R1  
7 add R1, R2  
8 add R3, R3  
9 add R21, X  
10 add R12, Y  
11 mv X, R2
```

```
1 import sys  
2 import re  
3  
4 if len(sys.argv) != 2:  
5     exit(f"Usage: {sys.argv[0]} FILENAME")
```

```
6
7 filename = sys.argv[1]
8
9 with open(filename) as fh:
10     code = fh.read()
11
12 # assuming there are no R4 values then 4 substitutions
13 # will do
13 code = re.sub(r'R1', 'R4', code)
14 code = re.sub(r'R3', 'R1', code)
15 code = re.sub(r'R2', 'R3', code)
16 code = re.sub(r'R4', 'R2', code)
17
18 print(code)
```

```
1 import sys
2 import re
3
4 if len(sys.argv) != 2:
5     exit(f"Usage: {sys.argv[0]} FILENAME")
6
7 filename = sys.argv[1]
8
9 with open(filename) as fh:
10     code = fh.read()
11
12 # or without any assumption and in one substitution:
13 mapping = {
14     'R1' : 'R2',
15     'R2' : 'R3',
16     'R3' : 'R1',
17 }
18
19
20 code = re.sub(r'\b(R[123])\b', lambda match:
mapping[match.group(1)], code)
21
22 print(code)
```

```
1 import sys
2 import re
3
```

```
4 if len(sys.argv) != 2:
5     exit(f"Usage: {sys.argv[0]} FILENAME")
6
7 filename = sys.argv[1]
8
9 with open(filename) as fh:
10     code = fh.read()
11
12
13 # or without any assumption and in one substitution:
14 mapping = {
15     'R1' : 'R2',
16     'R2' : 'R3',
17     'R3' : 'R1',
18     'R12' : 'R21',
19     'R21' : 'R12',
20 }
21
22 code = re.sub(r'\b(R1|R2|R3|R12)\b', lambda match:
mapping[match.group(1)], code)
23
24 print(code)
```

```
1 import sys
2 import re
3
4 if len(sys.argv) != 2:
5     exit(f"Usage: {sys.argv[0]} FILENAME")
6
7 filename = sys.argv[1]
8
9 with open(filename) as fh:
10     code = fh.read()
11
12
13 # or without any assumption and in one substitution:
14 mapping = {
15     'R1' : 'R2',
16     'R2' : 'R3',
17     'R3' : 'R1',
18     'R12' : 'R21',
19     'R21' : 'R12',
20 }
21
```

```
22 regex = r'\b(' + '|'.join(mapping.keys()) + r')\b'
23
24 code = re.sub(regex, lambda match:
mapping[match.group(1)], code)
25
26 print(code)
```

Full example of previous

```
1 import sys
2 import os
3 import time
4 import re
5
6 if len(sys.argv) <= 1:
7     exit(f"Usage: {sys.argv[0]} INFILES")
8
9 conversion = {
10     'R1' : 'R2',
11     'R2' : 'R3',
12     'R3' : 'R1',
13     'R12' : 'R21',
14     'R21' : 'R12',
15 }
16 #print(conversion)
17
18 def replace(mapping, files):
19     regex = r'\b(' + '|'.join(mapping.keys()) + r')\b'
20     #print(regex)
21     ts = time.time()
22
23     for filename in files:
24         with open(filename) as fh:
25             data = fh.read()
26             data = re.sub(regex, lambda match:
mapping[match.group(1)], data)
27             os.rename(filename, f"{filename}.{ts}")
# backup with current timestamp
28         with open(filename, 'w') as fh:
29             fh.write(data)
30
31 replace(conversion, sys.argv[1:]);
```

Split with regex

```
1 fname      =      Foo
2 lname      = Bar
3 email=foo@bar.com
```

```
1 import sys
2 import re
3
4 # data: field_value_pairs.txt
5 if len(sys.argv) != 2:
6     exit(f"Usage: {sys.argv[0]} filename")
7
8 filename = sys.argv[1]
9
10 with open(filename) as fh:
11     for line in fh:
12         line = line.rstrip("\n")
13         field, value = re.split(r'\s*=\s*', line)
14         print(f"{value}={field}")
```

```
1 Foo=fname
2 Bar=lname
3 foo@bar.com=email
```

Exercises: Regexes part 1

Pick up a file with some text in it. Write a script (one for each item) that prints out every line from the file that matches the requirement. You can use the script at the end of the page as a starting point but you will have to change it!

- has a ‘q’
- starts with a ‘q’
- has ‘th’
- has an ‘q’ or a ‘Q’
- has a ‘*’ in it
- starts with an ‘q’ or an ‘Q’
- has both ‘a’ and ‘e’ in it
- has an ‘a’ and somewhere later an ‘e’
- does not have an ‘a’
- does not have an ‘a’ nor ‘e’
- has an ‘a’ but not ‘e’
- has at least 2 consecutive vowels (a,e,i,o,u) like in the word “bear”
- has at least 3 vowels
- has at least 6 characters
- has at exactly 6 characters
- all the words with either ‘Bar’ or ‘Baz’ in them
- all the rows with either ‘apple pie’ or ‘banana pie’ in them
- for each row print if it was apple or banana pie?
- Bonus: Print if the same word appears twice in the same line
- Bonus: has a double character (e.g. ‘oo’)

¹ `import sys`

² `import re`

```
3
4 if len(sys.argv) != 2:
5     print("Usage:", sys.argv[0], "FILE")
6     exit()
7
8 filename = sys.argv[1]
9 with open(filename, 'r') as fh:
10     for line in fh:
11         print(line, end=" ")
12
13     match = re.search(r'REGEX1', line)
14     if match:
15         print("    Matching 1", match.group(0))
16
17     match = re.search(r'REGEX2', line)
18     if match:
19         print("    Matching 2", match.group(0))
```

Exercise: Regexes part 2

Write functions that returns true if the given value is a

- Hexadecimal number
- Octal number
- Binary number

Write a function that given a string it return true if the string is a number.

As there might be several definitions of what is the number create several solutions
one for each definition:

- Non negative integer.
- Integer. (Will you also allow + in front of the number or only - ?)
- Real number. (Do you allow .3 ? What about 2. ?)
- In scientific notation. (something like this: 2.123e4)

```
1 23
2 2.3
3 2.3.4
4 2.4e3
5 abc
```

Exercise: Sort SNMP numbers

Given a file with SNMP numbers (one number on every line)
print them in sorted order comparing the first number of
each SNMP number first.

If they are equal then comparing the second number, etc...

input:

```
1 1.2.7.6
2 4.5.7.23
3 1.2.7
4 1.12.23
5 2.3.5.7.10.8.9
6 1.2.7.5
```

output:

```
1 1.2.7
2 1.2.7.5
3 1.2.7.6
4 1.12.23
5 2.3.5.7.10.8.9
6 4.5.7.23
```

Exercise: parse hours log file and give report

The log file looks like this

```
1 09:20 Introduction
2 11:00 Exercises
3 11:15 Break
4 11:35 Numbers and strings
5 12:30 Lunch Break
6 13:30 Exercises
7 14:10 Solutions
8 14:30 Break
9 14:40 Lists
10 15:40 Exercises
11 17:00 Solutions
12 17:30 End
13
14 09:30 Lists and Tuples
15 10:30 Break
16 10:50 Exercises
17 12:00 Solutions
18 12:30 Dictionaries
19 12:45 Lunch Break
```

```
20 14:15 Exercises
21 16:00 Solutions
22 16:15 Break
23 16:30 Functions
24 17:00 Exercises
25 17:30 End
```

the report should look something like this:

```
1 09:20-11:00 Introduction
2 11:00-11:15 Exercises
3 11:15-11:35 Break
4 11:35-12:30 Numbers and strings
5 12:30-13:30 Lunch Break
6 13:30-14:10 Exercises
7 14:10-14:30 Solutions
8 14:30-14:40 Break
9 14:40-15:40 Lists
10 15:40-17:00 Exercises
11 17:00-17:30 Solutions
12
13 09:30-10:30 Lists and Tuples
14 10:30-10:50 Break
15 10:50-12:00 Exercises
16 12:00-12:30 Solutions
17 12:30-12:45 Dictionaries
18 12:45-14:15 Lunch Break
19 14:15-16:00 Exercises
20 16:00-16:15 Solutions
21 16:15-16:30 Break
22 16:30-17:00 Functions
23 17:00-17:30 Exercises
24
25 Break           65 minutes   6%
26 Dictionaries    15 minutes   1%
27 Exercises       340 minutes 35%
28 Functions        30 minutes   3%
29 Introduction     100 minutes 10%
30 Lists            60 minutes   6%
31 Lists and Tuples 60 minutes   6%
32 Lunch Break      150 minutes 15%
33 Numbers and strings 55 minutes 5%
34 Solutions        95 minutes   9%
```

Exercise: Parse ini file

An ini file has sections starting by the name of the section in square brackets and within each section there are key = value pairs with optional spaces around the “=” sign.

The keys can only contain letters, numbers, underscore or dash. In addition there can be empty lines and lines starting with # which are comments.

Given a filename, generate a 2 dimensional hash and then print it out.

Example ini file:

```
1 # comment
2 [alpha]
3
4 base    = moon
5 ship= alpha 3
6
7 [earth]
8     # ?
9 base=London
10 ship= x-wing
```

If you print it, it should look like this (except of the nice formatting).

```
1 {
2     'alpha': {
3         'base': 'moon',
4         'ship': 'alpha 3'
5     },
6     'earth': {
7         'base': 'London',
8         'ship': 'x-wing'
9     }
10 }
```

Exercise: Replace Python

1 Replace all occurrences of Python or python by Java ...
2 ... but avoid replacing Monty Python.

Exercise: Extract phone numbers

1 Given a text message fetch all the phone numbers:
2 Fetch numbers that look like 09-1234567
3 then also fetch +972-2-1234567
4 and maybe also 09-123-4567
5 This 123 is not a phone number.

Solution: Sort SNMP numbers

```
1 import sys
2
3 def process(filename):
4     snmps = []
5     with open(filename) as fh:
6         for row in fh:
7             snmps.append({
8                 'orig': row.rstrip(),
9                 })
10    #print(snmps)
11
12    max_number_of_parts = 0
13    max_number_of_digits = 0
14    for snmp in snmps:
15        snmp['split'] = snmp['orig'].split('.')
16        max_number_of_parts = max(max_number_of_parts,
17                                   len(snmp['split']))
18        for part in snmp['split']:
19            max_number_of_digits =
20            max(max_number_of_digits, len(part))
21
22    padding = "{:0" + str(max_number_of_digits) + "}"
23    #print(padding)
24    for snmp in snmps:
25        padded = [
```

```

24         padded_split = snmp['split'] + ['0'] * 
(max_number_of_parts - len(snmp['split']\ 
])
25
26
27     for part in padded_split:
28         padded.append(padding.format( int(part) ))
29     snmp['padded'] = padded
30     snmp['joined'] = '.'.join(padded)
31
32
33 #print(snmps)
34 #print(max_number_of_parts)
35 #print(max_number_of_digits)
36
37 snmps.sort(key = lambda e: e['joined'])
38 sorted_snmps = []
39 for snmp in snmps:
40     sorted_snmps.append( snmp['orig'] )
41 for snmp in sorted_snmps:
42     print(snmp)
43
44 # get the max number of all the snmp parts
45 # make each snmp the same length
46 # pad each part to that length with leading 0s
47
48 if len(sys.argv) < 2:
49     exit("Usage: {} FILENAME".format(sys.argv[0]))
50 process(sys.argv[1])

```

Solution: parse hours log file and give report

```

1 import sys
2
3
4 if len(sys.argv) < 2:
5     exit("Usage: {} FILENAME".format(sys.argv[0]))
6
7
8
9 data = {}
10
11 def read_file(filename):
12     entries = []

```

```

13     with open(filename) as fh:
14         for row in fh:
15             row = row.rstrip("\n")
16             if row == '':
17                 process_day(entries)
18                 entries = []
19             continue
20             #print(row)
21             time, title = row.split(" ", 1)
22             #print(time)
23             #print(title)
24             #print('')
25
26             entries.append({
27                 'start': time,
28                 'title': title,
29             })
30         process_day(entries)
31
32 def process_day(entries):
33     for i in range(len(entries)-1):
34         start = entries[i]['start']
35         title = entries[i]['title']
36         end = entries[i+1]['start']
37         print("{}-{} {}".format(start, end, title))
38
39         # manual way to parse timestamp and calculate
40         # elapsed time
41         # as we have not learned to use the datetim
42         # module yet
43         start_hour, start_min = start.split(':')
44         end_hour, end_min = end.split(':')
45         start_in_min = 60*int(start_hour) +
46         int(start_min)
47         end_in_min = 60*int(end_hour) + int(end_min)
48         elapsed_time = end_in_min - start_in_min
49         #print(elapsed_time)
50
51         if title not in data:
52             data[title] = 0
53             data[title] += elapsed_time
54
55     print('')

```

```
54
55 def print_summary():
56     total = 0
57     for val in data.values():
58         total += val
59
60     for key in sorted( data.keys() ):
61         print("{:20}    {:4} minutes"
62           .format(key, data[key], int(100 * data[key] / total)))
63
64
65 read_file( sys.argv[1] )
66 print_summary()
```

Solution: Processing INI file manually

```
1 # comment
2
3     # deep comment
4
5 outer = 42
6
7 [person]
8 fname = Foo
9 lname=Bar
10 phone =      123
11
12 [company]
13 name = Acme Corp.
14 phone = 456
```

```
1 import sys
2 import re
3
4 # Sample input data.ini
5
6 def parse():
7     if len(sys.argv) != 2:
8         exit("Usage: {} FILENAME".format(sys.argv[0]))
9     filename = sys.argv[1]
10    data = {}
```

```

11     # print("Dealing with " + filename)
12     with open(filename) as fh:
13         section = '__DEFAULT__'
14         for line in fh:
15             if re.match(r'^\s*(#.*)?$', line):
16                 continue
17             match = re.match(r'^\[((^\s]+)+\]\s*$', line)
18             if (match):
19                 # print('Section "{}".format(m.group(1))')
20                 section = match.group(1)
21                 continue
22             match = re.match(r'^\s*(.+?)\s*=\s*(.*?)\s*$', line)
23             if match:
24                 # print 'field :"{}"  value: "{}".format(m.group(1), m.group(2))'
25                 if not data.get(section):
26                     data[section] = {}
27                 data[section][match.group(1)] = match.group(2)
28
29     return data
30
31 if __name__ == '__main__':
32     ini = parse()
33     print(ini)

```

Solution: Processing config file

```

1 [person]
2 fname = Foo
3 lname=Bar
4 phone = 123
5
6 # comment
7
8     # deep comment
9
10
11 [company]

```

```
12 name = Acme Corp.  
13 phone = 456
```

```
1 import configparser  
2 import sys  
3  
4 def parse():  
5     if len(sys.argv) != 2:  
6         print("Usage: " + sys.argv[0] + " FILENAME")  
7         exit()  
8     filename = sys.argv[1]  
9  
10    cp = configparser.RawConfigParser()  
11    cp.read(filename)  
12    return cp  
13  
14 ini = parse()  
15  
16 for section in ini.sections():  
17     print(section)  
18     for v in ini.items(section):  
19         print(" {} = {}".format(v[0], v[1]))
```

Solution: Extract phone numbers

```
1 import re  
2  
3 filename = "phone.txt"  
4 with open(filename) as fh:  
5     for line in fh:  
6         match = re.search(r'''\\b  
7             (  
8                 \\d\\d-\\d{7}  
9                 |  
10                \\d\\d\\d-\\d-\\d{7}  
11                |  
12                \\d\\d-\\d\\d\\d-\\d\\d\\d\\d  
13            )\\b''', line, re.VERBOSE)  
14     if match:  
15         print(match.group(1))
```

Regular Expressions Cheat sheet

Expression	Meaning
a	Just an ‘a’ character
.	any character except new-line
[bgh.]	one of the chars listed in the character class b,g,h or .
[b-h]	The same as [bcdefgh]
[a-z]	Lower case letters
[b-]	The letter b or -
[^bx]	Anything except b or x
\w	Word characters: [a-zA-Z0-9_]
\d	Digits: [0-9]
\s	[\\f\\t\\n\\r] form-feed, tab, newline, carriage return and SPACE
W	[^\\w]
D	[^\\d]
S	[^\\s]
a*	0-infinite ‘a’ characters
a+	1-infinite ‘a’ characters
a?	0-1 ‘a’ characters
a	n-m ‘a’ characters
()	Grouping and capturing
	Alternation
\1, \2	Capture buffers
^ \$	Beginning and end of string anchors

[re](#)

Fix bad JSON

```
1 {
2     subscriptions : [
3         {
4             name : "Foo Bar",
5             source_name : "pypi",
6             space_names : [
7                 "Foo", "Bar"
8             ]
9         }
10    ]
11 }
```

```
1 import re, json, os
2
3 json_file = os.path.join(
4     os.path.dirname(__file__),
5     'bad.json'
6 )
7 with open(json_file) as fh:
8     data = json.load(fh)
9     # ValueError: Expecting property name: line 2
column 4 (char 5)
```

```
1 import re, json, os
2
3 def fix(s):
4     return re.sub(r'(\s)([^:\s][^:]+[^:\s])(\s+)',',
r'\1"\2"\3', s)
5
6 json_file = os.path.join(
7     os.path.dirname(__file__),
8     'bad.json'
9 )
10 with open(json_file) as fh:
11     bad_json_rows = fh.readlines()
12     json_str = ''.join(map(fix, bad_json_rows))
13     print(json_str)
14     data = json.loads(json_str)
15     print(data)
```

Fix very bad JSON

```
1 [
2 {
3     TID : "t-0_login_sucess"
4     Test :
5     [
6         {SetValue : { uname : "Zorg", pass : "Rules" }
7     ,
8             {DoAction : "login"}, {CheckResult: [0, LOGGED_IN] }
9         ]
10 },
11 { TID : "t-1_login_failure", Test : [ {SetValue :
12 { uname : "11", pass : "im2happy78" } },
13 {DoAction : "login"}, {CheckResult: [-1000,
LOGGED_OUT] } ] }
14 ]
```

```
1 import re, json, os
2
3 json_file = os.path.join(
4     os.path.dirname(__file__),
5     'very_bad.json'
6 )
7 with open(json_file, 'r') as fh:
8     bad_json = fh.read()
9     #print(bad_json)
10    improved_json = re.sub(r'"\\s*$', ' ', bad_json,
flags=re.MULTILINE)
11    #print(improved_json)
12
13    # good_json = re.sub(r'(?<! ") (?P<word>
[\w-]+) \b(?!") ', '"\g<word>"',
14        # improved_json)
15    # good_json = re.sub(r'(?<[\{\s]) (?P<word>[\w-]+)
(?=[:\s]) ', '\g<word>',
16        # improved_json)
17    # good_json = re.sub(r'([\{\s]) (?P<word>[\w-]+)
([:, \]\s]) ', '\1"\g<word>"\3',
18        # improved_json)
19    good_json = re.sub(r'(?<=[\{\s]) (?P<word>[\w-]+)
(?=[:, \]\s]) ', '"\g<word>"',
```

```
20     improved_json)
21     #print(good_json)
22
23 # with open('out.js', 'w') as fh:
24 #     fh.write(good_json)
25
26 data = json.loads(good_json)
27 print(data)
```

Raw string or escape

Let's try to check if a string contains a back-slash?

```
1 import re
2
3 txt = 'text with slash \ and more text'
4 print(txt)          # text with slash \ and more text
5
6 # m0 = re.search('\\', txt)
7 #     # SyntaxError: EOL while scanning string literal
8
9 # m0 = re.search('\\\\', txt)
10 #     # Exception: sre_constants.error: bogus escape
11 # (end of line)
12 # because the regex engine does not know what to
13 # do with a single \
14
15 m1 = re.search('\\\\\\', txt)
16 if m1:
17     print('m1')      # m1
18
19 m2 = re.search(r'\\', txt)
20 if m2:
21     print('m2')      # m2
```

Remove spaces regex

This is not necessary as we can use rstrip, lstrip, and replace.

```
1 import re
2
```

```
3 line = " ab cd "
4
5 res = re.sub(r'^\s+', '', line) # leading
6 print(f'{res}')
7
8 res = re.sub(r'\s+$', '', line) # trailing
9 print(f'{res}')
```

both ends:

```
1 re.sub(r'\s*(.*)\s*$', r'\1', line) # " abc " =>
"abc" because of the greediness
```

```
1 re.sub('^\s*(.*?)\s*$', '\1', line) # " abc " =>
"abc" minimal match
```

Regex Unicode

Python 3.8 required

```
1 print("\N{GREEK CAPITAL LETTER DELTA}")
2
3 print("\u05E9")
4 print("\u05DC")
5 print("\u05D5")
6 print("\u05DD")
7 print("\u262E")
8 print("\u1F426") # "bird"
9
10 print("\u05E9\u05DC\u05D5\u05DD \u262E")
```

```
1 Hello World!
2 Szia Világ!
3 שָׁלוֹם עֲוָלָם!
```

```
1 import re
2
3 filename = "mixed.txt"
```

```
4
5 with open(filename) as fh:
6     lines = fh.readlines()
7 for line in lines:
8     if re.search('\N{IN HEBREW}', line):
9         print(line)
```

Anchors Other example

```
1 import re
2
3 strings = [
4     "123-XYZ-456",
5     "a 123-XYZ-456 b",
6     "a 123-XYZ-456",
7     "123-XYZ-456 b",
8     "123-XYZ-456\n",
9 ]
10
11 regexes = [
12     r'\d{3}-\w+-\d{3}',
13     r'^\d{3}-\w+-\d{3}',
14     r'\d{3}-\w+-\d{3}$',
15     r'^\d{3}-\w+-\d{3}$',
16     r'^\d{3}-\w+-\d{3}\Z',
17     r'\A\d{3}-\w+-\d{3}\Z',
18 ]
19
20 for r in regexes:
21     print(r)
22     for s in strings:
23         #print(r, s)
24         if (re.search(r, s)):
25             print(' ', s)
26     print(' - ' * 10)
```

Python standard modules

Some Standard modules

- [sys](#) - System specific
- [os](#) - Operating System
- [stat](#) - inode table
- [shutil](#) - File Operations
- [glob](#) - Unix style pathname expansion
- [subprocess](#) - Processes
- [argparse](#) - Command Line Arguments
- [re](#) - Regexes
- [math](#) - Mathematics
- [time](#) - timestamp and friends
- [datetime](#) - time management
- [random](#) - Random numbers

sys

```
1 import sys,os
2
3 print(sys.argv) # the list of the values
4     # on the command line sys.argv[0] is the name of
the Python script
5
6 print(sys.executable) # path to the python
interpreter
7
8 # print(sys.path)
9     # list of file-system path strings for searching
for modules
```

```

10      # hard-coded at compile time but can be changed
11      via the PYTHONPATH
12      # environment variable or during execution by
13      modifying sys.path
14
15      print(sys.version_info)
16      # sys.version_info(major=2, minor=7, micro=12,
17      # releaselevel='final', serial=0)
18
19      print(sys.version_info.major)    # 2 or 3
20
21      print(sys.platform)          # darwin   or   linux2   or
22      win32
23
24      print(os.uname())
25      # On Mac:
26      # ('Darwin', 'air.local', '16.3.0', 'Darwin Kernel
27      Version 16.3.0: Thu Nov 17 20:23:\'
28      58 PST 2016; root:xnu-3789.31.2~1/RELEASE_X86_64',
29      'x86_64')
30
31      # On Linux:
32      # posix.uname_result(sysname='Linux',
33      nodename='thinkpad', release='5.0.0-32-generic\
34      ', version='#34-Ubuntu SMP Wed Oct 2 02:06:48 UTC
35      2019', machine='x86_64')

```

```

1  ['examples/sys/mysys.py']
2
3  /usr/bin/python
4
5  ['/Users/gabor/work/training/python/examples/sys',
6  '/Users/gabor/python/lib/python2.7/site-
7  packages/crypto-1.1.0-py2.7.egg',
8  ...,
9  '/Users/gabor/python',
10  '/Users/gabor/python/lib/python2.7/site-packages',
11  ...]

```

Writing to standard error (stderr)

```
1 import sys
2
3 print("on stdout (Standard Output)")
4 print("on stderr (Standard Error)", file=sys.stderr)
5 sys.stderr.write("in stderr again\n")
```

Redirection:

```
1 python stderr.py > out.txt 2> err.txt
2 python stderr.py 2> /dev/null
3 python stderr.py > out.txt 2>&1
```

Current directory (getcwd, pwd, chdir)

```
1 import os
2
3 this_dir = os.getcwd()
4 print(this_dir)
5
6 # os.chdir('/path/to/some/dir')
7 os.chdir('..')
```

OS dir (mkdir, makedirs, remove, rmdir)

```
1 os.mkdir(path_to_new_dir)
2 os.makedirs(path_to_new_dir)
3
4 os.remove()           remove a file
5 os.unlink()          (the same)
6
7 os.rmdir()           single empty directory
8 os.removedirs()      empty subdirectories as well
9 shutil.rmtree()      rm -rf
```

python which OS are we running on (os, platform)

```
1 import os
2 import platform
3
4 print(os.name)
5 print(platform.system())
6 print(platform.release())
7
8 # posix
9 # Linux
10 # 5.3.0-24-generic
```

Get process ID

```
1 import os
2
3 print(os.getpid())
4 print(os.getppid())
```

```
1 93518
2 92859
```

```
1 echo $$
```

OS path

```
1 import os
2
3 os.path.basename(path_to_thing)
4 os.path.dirname(path_to_thing)
5 os.path.abspath(path_to_file)
6
7 os.path.exists(path_to_file)
8 os.path.isdir(path_to_thing)
9
10 os.path.expanduser('~/')
```

Traverse directory tree - list directories recursively

```
1 import os
2 import sys
3
4 if len(sys.argv) != 2:
5     exit("Usage: {}  
PATH_TO_DIRECTORY".format(sys.argv[0]))
6
7 root = sys.argv[1]
8
9 for dirname, dirs, files in os.walk(root):
10    #print(dirname)      # relative path (from cwd) to  
the directory being processed
11    #print(dirs)        # list of subdirectories in the  
currently processed directory
12    #print(files)        # list of files in the  
currently processed directory
13
14    for filename in files:
15        print(os.path.join(dirname, filename))    #  
relative path to the "current" fi\  
16 le
```

os.path.join

```
1 import os
2
3 path = os.path.join('home', 'foo', 'work')
4 print(path)  # home/foo/work
```

Directory listing

```
1 import os
2 import sys
3
4 if len(sys.argv) != 2:
5     exit("Usage: {} directory".format(sys.argv[0]))
```

```
6  
7 path = sys.argv[1]  
8 files = os.listdir(path)  
9 for name in files:  
10     print(name)  
11     print(os.path.join(path, name))
```

expanduser - handle tilde ~

```
1 import os  
2  
3 print( os.path.expanduser("~/") )  
4 print( os.path.expanduser("~/work") )  
5 print( os.path.expanduser("~/other") )  
6 print(  
    os.path.expanduser("some/other/dir/no/expansion") )
```

Listing specific files using glob

```
1 import glob  
2  
3 files = glob.glob("*.py")  
4 print(files)  
5  
6 files = glob.glob("/usr/bin/*.sh")  
7 print(files)
```

External command with system

```
1 import os  
2  
3 command = 'ls -l'  
4  
5 os.system(command)
```

If you wanted to list the content of a directory in an os independent way you'd use `os.listdir('..')`

or you could use the `glob.glob("*.py")` function to have a subset of files.

subprocess

Run external command and capture the output

```
1 import time
2 import sys
3
4 for i in range(3):
5     print("OUT {}".format(i))
6     print("ERR {}".format(i), file=sys.stderr)
7     time.sleep(1)
```

```
1 import subprocess
2 import sys
3
4 command = [sys.executable, 'slow.py']
5
6 proc = subprocess.Popen(command,
7                         stdout = subprocess.PIPE,
8                         stderr = subprocess.PIPE,
9                         )
10
11 out,err = proc.communicate()    # runs the code
12
13 # out and err are two strings
14
15 print('exit code:', proc.returncode)
16
17 print('out:')
18 for line in out.decode('utf8').split('\n'):
19     print(line)
20
21 print('err:')
22 for line in err.decode('utf8').split('\n'):
23     print(line)
```

In this example `p` is an instance of the `subprocess.PIPE` class. The command is executed when the object is created.

subprocess in the background

```
1 import subprocess
2 import sys
3 import time
4
5
6 proc = subprocess.Popen([sys.executable, 'slow.py'],
7     stdout = subprocess.PIPE,
8     stderr = subprocess.PIPE,
9 )
10
11 #out, err = proc.communicate()    # this is when the
12 #code starts executing
13 #print(out)
14 #print(err)
15
16 timeout = 6
17 while True:
18     poll = proc.poll()
19     print(poll)
20     time.sleep(0.5)
21     timeout -= 0.5
22     if timeout <= 0:
23         break
24     if poll is not None:
25         break
26
27 print("Final: {}".format(poll))
28 if poll is None:
29     pass
30 else:
31     out, err = proc.communicate()
32     print(out)
33     print(err)
```

Accessing the system environment variables from Python

```
1 import os
2
3 print(os.environ['HOME']) # /Users/gabor
4 print(os.environ.get('HOME')) # /Users/gabor
5
6 for k in os.environ.keys():
7     print("{:30} {}".format(k, os.environ[k]))
```

os.environ is a dictionary where the keys are the environment variables and the values are, well, the values.

Set env and run command

```
1 import os
2
3 os.system("echo hello")
4 os.system("echo $HOME")
5
6 os.system("echo Before ${MY_TEST}")
7 os.environ['MY_TEST'] = 'qqrq'
8 os.system("echo After ${MY_TEST}")
```

We can change the environment variables and that change will be visible in subprocesses, but once we exit from our Python program, the change will not persist.

shutil

```
1 import shutil
2
3 shutil.copy(source, dest)
4 shutil.copytree(source, dest)
5 shutil.move(source, dest)
6 shutil.rmtree(path)
```

time

```
1 import time
2
3 print(time.time())          # 1351178170.85
4
5 print(time.timezone)       # 7200 = 2*60*60 (GMT + 2)
6 print(time.daylight)        # 1 (DST or Daylight Saving
Time)
7
8 print(time.gmtime())       # time.struct_time
9      # time.struct_time(tm_year=2012, tm_mon=10,
tm_mday=25,
10     # tm_hour=17, tm_min=25, tm_sec=34, tm_wday=3,
tm_yday=299, tm_isdst=0)
11
12 t = time.gmtime()
13 print(t.tm_year) # 2012
14
15 print(time.strftime('%Y-%m-%d %H:%M:%S')) # with
optional timestamp
```

sleep in Python

```
1 import time
2
3 start = time.time()
4 print("hello " + str(start))
5
6 time.sleep(3.5)
7
```

```
8 end = time.time()
9 print("world " + str(end))
10 print("Elapsed time:" + str(end-start))
```

```
1 hello 1475217162.472256
2 world 1475217165.973437
3 Elapsed time:3.501181125640869
```

timer

More time-related examples.

```
1 import random
2 import time
3
4 #
https://docs.python.org/3/library/time.html#time.struct\_time
5
6 print(time.time())      # time since the epoch in
seconds
7 print(time.asctime())   # current local time in human-
readable format
8 print(time.strftime("%Y-%m-%d %H:%M:%S"))  # create
your own human-readable format
9
10 print(time.gmtime(0))  # epoch
11 print(time.asctime(time.gmtime(0)))  # epoch in human-
readable format
12
13 print(time.localtime()) # local time now
14 print(time.gmtime())  # time in London
15
16
17
18 print(time.process_time())
19 print(time.process_time_ns())
20
21 s = time.perf_counter()
22 ps = time.process_time()
23 print(time.monotonic())
```

```
24 time.sleep(0.1)
25 print(time.monotonic())
26 e = time.perf_counter()
27 for _ in range(100000):
28     random.random()
29 pe = time.process_time()
30 print(s)
31 print(e)
32 print(e-s)
33 print(pe-ps)
34
35 # print(time.get_clock_info('monotonic'))
```

Current date and time datetime now

```
1 import datetime
2
3 now = datetime.datetime.now()
4 print(now)                      # 2015-07-02 16:28:01.762244
5 print(type(now))                # <type 'datetime.datetime'>
6
7 print(now.year)                 # 2015
8 print(now.month)                # 7
9 print(now.day)                  # 2
10 print(now.hour)                 # 16
11 print(now.minute)               # 28
12 print(now.second)               # 1
13 print(now.microsecond)         # 762244
14
15 print(now.strftime("%Y%m%d-%H%M%S-%f")) # 20150702-
162801-762244
16 print(now.strftime("%B %b %a %A"))      # July Jul
Thu Thursday
17 print(now.strftime("%c"))            # Thu Jul  2
16:28:01 2015
```

Converting string to datetime

```
1 import datetime
2
```

```
3 usa_date_format = "%m/%d/%Y" # MM/DD/YYYY
4 world_date_format = "%d/%m/%Y" # DD/MM/YYYY
5 other_date_format = "%Y/%m/%d" # YYYY/MM/DD
6
7
8 d = "2012-12-19"
9 some_day = datetime.datetime.strptime(d, '%Y-%m-%d') # YYYY-MM-DD
10 print(some_day) # 2012-12-19
11 print(type(some_day)) # <type 'datetime.datetime'>
12
13 t = "2013-11-04 11:23:45" # YYYY-MM-DD HH:MM:SS
14 some_time = datetime.datetime.strptime(t, '%Y-%m-%d %H:%M:%S')
15 print(type(some_time)) # <type 'datetime.date'>
16 print(some_time) # 2013-11-04
17 print(some_time.minute) # 23
```

datetime arithmeticis

```
1 import datetime
2
3 t1 = "2013-12-29T11:23:45"
4 t2 = "2014-01-02T10:19:49"
5 dt1 = datetime.datetime.strptime(t1, '%Y-%m-%dT%H:%M:%S')
6 dt2 = datetime.datetime.strptime(t2, '%Y-%m-%dT%H:%M:%S')
7 print(dt1) # 2013-12-29 11:23:45
8 print(dt2) # 2014-01-02 10:19:49
9
10 d = dt2-dt1
11 print(d) # 3 days, 22:56:04
12 print(type(d)) # <type 'datetime.timedelta'>
13 print(d.total_seconds()) # 341764.0
14
15 nd = dt1 + datetime.timedelta(days = 3)
16 print(nd) # 2014-01-01 11:23:45
```

Rounding datetime object to nearest second

```
1 import datetime
2
3 d = datetime.datetime.now()
4 x = d - datetime.timedelta(microseconds=d.microsecond)
5 print(d) # 2019-11-01 07:11:19.930974
6 print(x) # 2019-11-01 07:11:19
```

Signals and Python

- [man 7 signal](#) (on Linux)
- Unix: kill PID, kill -9 PID, Ctrl-C, Ctrl-Z
- os.kill
- [signal](#)

Sending Signal

```
1 import signal
2 import os
3
4 print("before")
5 os.kill(os.getpid(), signal.SIGUSR1)
6 print("after")
```

```
1 before
2 User defined signal 1: 30
```

Catching Signal

```
1 import signal
2 import os
3
4 def handler(signum, frame):
5     print('Signal handler called with signal', signum)
6
7 signal.signal(signal.SIGUSR1, handler)
8
9 print("before")
```

```
10 os.kill(os.getpid(), signal.SIGUSR1)
11 print("after")
```

```
1 before
2 ('Signal handler called with signal', 30)
3 after
```

Catching Ctrl-C on Unix

```
1 username = input('Username:')
2 print(username)
```

```
1 $ python ctrl_c.py
```

```
1 Username:^CTraceback (most recent call last):
2   File "ctrl_c.py", line 3, in <module>
3     username = input('Username:')
4 KeyboardInterrupt
```

```
1 import signal
2
3 def handler(signum, frame):
4     print('Signal handler called with signal', signum)
5
6 signal.signal(signal.SIGINT, handler)
7
8 username = input('Username:')
9 print(username)
```

- Cannot stop using Ctrl-C !
- Ctrl-Z and then kill %1
- kill PID

Catching Ctrl-C on Unix confirm

```
1 import signal
2 import time
3
4 def handler(signum, frame):
5     answer = input('We are almost done. Do you really
want to exit? [yes]:')
6     if answer == 'yes':
7         print('bye')
8         exit()
9     print("Then let's keep running")
10
11 signal.signal(signal.SIGINT, handler)
12
13 for _ in range(10):
14     time.sleep(5)
```

Alarm signal and timeouts

```
1 import signal
2
3 class MyTimeout(Exception):
4     pass
5
6 def handler(signum, frame):
7     print('Signal handler called with signal', signum)
8     raise MyTimeout
9
10 try:
11     signal.signal(signal.SIGALRM, handler)
12     signal.alarm(5)
13     number = input("Divide by (5 sec):")
14     signal.alarm(0)
15     print(42/int(number))
16 except MyTimeout:
17     print('timeout')
18 except Exception as e:
19     print(e)
20     #raise
21
22 print("Still working")
```

deep copy list

```
1 a = [
2     {
3         'name': 'Joe',
4         'email': 'joe@examples.com',
5     },
6     {
7         'name': 'Mary',
8         'email': 'mary@examples.com',
9     },
10 ]
11
12
13 b = a
14 a[0]['phone'] = '1234'
15 a[0]['name'] = 'Jane'
16 a.append({
17     'name': 'George'
18 })
19
20 print(a)
21 print(b)
```

```
1 [{"name": "Jane", "email": "joe@examples.com", "phone": "1234"}, {"name": "Mary", "e\
2 mail": "mary@examples.com"}, {"name": "George"}]
3 [{"name": "Jane", "email": "joe@examples.com", "phone": "1234"}, {"name": "Mary", "e\
4 mail": "mary@examples.com"}, {"name": "George"}]
```

```
1 a = [
2     {
3         'name': 'Joe',
4         'email': 'joe@examples.com',
5     },
6     {
7         'name': 'Mary',
8         'email': 'mary@examples.com',
9     },
10 ]
11
```

```
12
13 b = a[:]
14 a[0]['phone'] = '1234'
15 a[0]['name'] = 'Jane'
16 a.append({
17     'name': 'George'
18 })
19
20 print(a)
21 print(b)
```

```
1 [{ 'name': 'Jane', 'email': 'joe@example.com', 'phone':
'1234'}, { 'name': 'Mary', 'e\
2 mail': 'mary@example.com'}, { 'name': 'George'}]
3 [{ 'name': 'Jane', 'email': 'joe@example.com', 'phone':
'1234'}, { 'name': 'Mary', 'e\
4 mail': 'mary@example.com'}]
```

```
1 from copy import deepcopy
2
3 a = [
4     {
5         'name': 'Joe',
6         'email': 'joe@example.com',
7     },
8     {
9         'name': 'Mary',
10        'email': 'mary@example.com',
11    },
12 ]
13
14
15 b = deepcopy(a)
16 a[0]['phone'] = '1234'
17 a[0]['name'] = 'Jane'
18 a.append({
19     'name': 'George'
20 })
21
22 print(a)
23 print(b)
```

```
1 [{ 'name': 'Jane', 'email': 'joe@example.com', 'phone': '1234'}, { 'name': 'Mary', 'e\\
2 mail': 'mary@example.com'}, { 'name': 'George'}]
3 [{ 'name': 'Joe', 'email': 'joe@example.com'}, { 'name': 'Mary', 'email': 'mary@examp\\
4 les.com'}]
```

deep copy dictionary

```
1 a = {
2     'name': 'Foo Bar',
3     'grades': {
4         'math': 70,
5         'art' : 100,
6     },
7     'friends': ['Mary', 'John', 'Jane', 'George'],
8 }
9
10 b = a
11 a['grades']['math'] = 90
12 a['email'] = 'foo@bar.com'
13 print(a)
14 print(b)
```

```
1 {'name': 'Foo Bar', 'grades': {'math': 90, 'art': 100},
2 'friends': ['Mary', 'John', \
3 'Jane', 'George'], 'email': 'foo@bar.com'}
4 {'name': 'Foo Bar', 'grades': {'math': 90, 'art': 100},
5 'friends': ['Mary', 'John', \
6 'Jane', 'George'], 'email': 'foo@bar.com'}
```

- deepcopy

```
1 from copy import deepcopy
2
3 a = {
4     'name': 'Foo Bar',
5     'grades': {
6         'math': 70,
7         'art' : 100,
```

```

8     },
9     'friends': ['Mary', 'John', 'Jane', 'George'],
10 }
11
12 b = deepcopy(a)
13 a['grades']['math'] = 90
14 a['email'] = 'foo@bar.com'
15 print(a)
16 print(b)

```

```

1 {'name': 'Foo Bar', 'grades': {'math': 90, 'art': 100},
2 'friends': ['Mary', 'John', \
3 'Jane', 'George'], 'email': 'foo@bar.com'}
4 {'name': 'Foo Bar', 'grades': {'math': 70, 'art': 100},
5 'friends': ['Mary', 'John', \
6 'Jane', 'George']}

```

Exercise: Catching Ctrl-C on Unix 2nd time

- When Ctrl-C is pressed display: “In order to really kill the application press Ctrl-C again” and keep running. If the user presses Ctrl-C again, then let id die.
- Improve the previous that if 5 sec within the first Ctrl-C there is no 2nd Ctrl-C then any further Ctrl-C will trigger the above message again.

Exercise: Signals

- What signal is sent when you run **kill PID**?
- Write a script that will disable the **kill PID** for your process. How can you kill it then?
- What signal is sent when we press Ctrl-Z ?

Ctrl-z

```
1 import signal
2 import os
3
4 print(os.getpid())
5
6 username = input('Username:')
7 print(username)
```

```
1 kill PID
```

```
1 import signal
2 import os
3
4 print(os.getpid())
5
6 def handler(signum, frame):
7     print('Signal handler called with signum', signum)
8
9 signal.signal(signal.SIGTERM, handler)
10
11 username = input('Username:')
12 print(username)
```

JSON

JSON - JavaScript Object Notation

[JSON](#) is basically the data format used by JavaScript. Because its universal availability it became the de-facto standard for data communication between many different languages. Most dynamic languages have an fairly good mapping between JSON and their own data structures.

Lists and dictionaries in the case of Python.

Documentation of the
[Python json library](#).

```
1 {"lname": "Bar", "email": null, "fname": "Foo",
"children": ["Moo", "Koo", "Roo"] }
```

dumps

```
1 import json
2
3 a = {
4     "fname" : 'Foo',
5     "lname" : 'Bar',
6     "email" : None,
7     "children" : [
8         "Moo",
9         "Koo",
10        "Roo"
11    ]
12 }
13 print(a)
14
15 json_str = json.dumps(a)
```

```
16 print(json_str)
17
18 with open('data.json', 'w') as fh:
19     fh.write(json_str)



---


1 {'lname': 'Bar', 'email': None, 'fname': 'Foo',
2  'children': ['Moo', 'Koo', 'Roo']}
3
4 {"lname": "Bar", "email": null, "fname": "Foo",
5  "children": ["Moo", "Koo", "Roo"]}
```

(lines were broken for readability on the slides)

`dumps` can be used to take a Python data structure and generate a string in JSON format. That string can then be saved in a file, inserted in a database, or sent over the wire.

loads

```
1 import json
2
3 with open('examples/json/data.json') as fh:
4     json_str = fh.read()
5
6 print(json_str)
7 b = json.loads(json_str)
8 print(b)



---


1 {"lname": "Bar", "email": null, "fname": "Foo",
2  "children": ["Moo", "Koo", "Roo"]}
3
4 {u'lname': u'Bar', u'email': None, u'fname': u'Foo',
5  u'children': [u'Moo', u'Koo', u'Roo']}
```

`u` is the Unicode prefix used in Python 2. In Python 3 it won't appear as Unicode is the default there.

dump

```
1 import json
2
3 a = {
4     "fname": 'Foo',
5     "lname": 'Bar',
6     "email": None,
7     "children": [
8         "Moo",
9         "Koo",
10        "Roo"
11    ]
12 }
13
14 print(a)
15
16 with open('data.json', 'w') as fh:
17     json.dump(a, fh)
```

```
1 {'lname': 'Bar', 'email': None, 'fname': 'Foo',
2  'children': ['Moo', 'Koo', 'Roo']}
3
4 {"lname": "Bar", "email": null, "fname": "Foo",
5  "children": ["Moo", "Koo", "Roo"]}
```

(lines were broken for readability on the slides)

As a special case **dump** will save the string in a file or in other stream.

load

```
1 import json
2
3 with open('examples/json/data.json', 'r') as fh:
4     a = json.load(fh)
5 print(a)
```

```
1 {u'lname': u'Bar', u'email': None, u'fname': u'Foo',  
2      u'children': [u'Moo', u'Koo', u'Roo']} }
```

Round trip

```
1 import json  
2 import os  
3 import time  
4  
5 data = {}  
6 filename = 'mydata.json'  
7  
8 if os.path.exists(filename):  
9     with open(filename) as fh:  
10         json_str = fh.read()  
11         print(json_str)  
12         data = json.loads(json_str)  
13  
14 data['name'] = 'Foo Bar'  
15 data['time'] = time.time()  
16  
17  
18 with open(filename, 'w') as fh:  
19     json_str = json.dumps(data)  
20     fh.write(json_str)
```

Pretty print JSON

```
1 import json  
2  
3 data = {  
4     "name" : "Foo Bar",  
5     "grades" : [23, 47, 99, 11],  
6     "children" : {  
7         "Peti Bar" : {  
8             "email": "peti@bar.com",  
9         },  
10        "Jenny Bar" : {  
11            "phone": "12345",  
12        },  
13    }  
14}
```

```
14 }
15
16 print(data)
17 print(json.dumps(data))
18 print(json.dumps(data, indent=4, separators=(',', ':'
'')))
```

```
1 {'name': 'Foo Bar', 'grades': [23, 47, 99, 11],
2 'children': {'Peti Bar': {'email': '\
3 peti@bar.com'}, 'Jenny Bar': {'phone': '12345'}}}
4 {"name": "Foo Bar", "grades": [23, 47, 99, 11],
5 "children": {"Peti Bar": {"email": "\
6 peti@bar.com"}, "Jenny Bar": {"phone": "12345"}}}
7 {
8     "name": "Foo Bar",
9     "grades": [
10         23,
11         47,
12         99,
13         11
14     ],
15     "children": {
16         "Peti Bar": {
17             "email": "peti@bar.com"
18         },
19         "Jenny Bar": {
20             "phone": "12345"
21     }
22 }
```

Sort keys in JSON

```
1 import json
2
3 data = {
4     "name" : "Foo Bar",
5     "grades" : [23, 47, 99, 11],
6     "children" : {
7         "Peti Bar" : {
8             "email": "peti@bar.com",
9         },
10    }
```

```
10         "Jenny Bar" : {
11             "phone": "12345",
12         },
13     }
14 }
15
16 print(json.dumps(data, sort_keys=True, indent=4,
separators=(',', ': ')))
```

```
1 {
2     "children": {
3         "Jenny Bar": {
4             "phone": "12345"
5         },
6         "Peti Bar": {
7             "email": "peti@bar.com"
8         }
9     },
10    "grades": [
11        23,
12        47,
13        99,
14        11
15    ],
16    "name": "Foo Bar"
17 }
```

Set order of keys in JSON - OrderedDict

```
1 from collections import OrderedDict
2
3 d = { }
4 d['a'] = 1
5 d['b'] = 2
6 d['c'] = 3
7 d['d'] = 4
8 print(d)
9
10 planned_order = ('b', 'c', 'd', 'a')
11 e = OrderedDict(sorted(d.items(), key=lambda x:
planned_order.index(x[0])))
12 print(e)
```

```
13
14 print('-----')
15 # Create index to value mapping dictionary from a list
16 # of values
16 planned_order = ('b', 'c', 'd', 'a')
17 plan = dict(zip(planned_order,
18 range(len(planned_order))))
18 print(plan)
19
20 f = OrderedDict(sorted(d.items(), key=lambda x:
21 plan[x[0]]))
21 print(f)
```

```
1 {'a': 1, 'b': 2, 'c': 3, 'd': 4}
2 OrderedDict([('b', 2), ('c', 3), ('d', 4), ('a', 1)])
3 -----
4 {'b': 0, 'c': 1, 'd': 2, 'a': 3}
5 OrderedDict([('b', 2), ('c', 3), ('d', 4), ('a', 1)])
```

Exercise: Counter in JSON

Write a script that will provide several counters. The user can provide an argument on the command line and the script will increment and display that counter. Keep the current values of the counters in a single JSON file. The script should behave like this:

```
1 $ python counter.py foo
2 1
3
4 $ python counter.py foo
5 2
6
7 $ python counter.py bar
8 1
9
10 $ python counter.py foo
11 3
```

Exercise: Phone book

Write a script that acts as a phonebook. As “database” use a file in JSON format.

```
1 $ python phone.py Foo 123
2 Foo added
3
4 $ python phone.py Bar
5 Bar is not in the phnebook
6
7 $ python phone.py Bar 456
8 Bar added
9
10 $ python phone.py Bar
11 456
12
13 $ python phone.py Foo
14 123
```

Can it handle changes in phone numbers?

Can it remove a name from the “database”?

Exercise: Processes

Write a program that will do “some work” that can be run in parallel

and collect the data. Make the code work in a single process by default

and allow the user to pass a number that will be the number of child processes

to be used. When the child process exits it should save the results in a file and the parent process should read them in.

The “some work” can be accessing 10-20 machines using “ssh machine uptime”
and creating a report from the results.

It can be fetching 10-20 URLs and reporting the size of each page.

It can be any other network intensive task.

Measure the time in both cases

Solution: Counter in JSON

```
1 import json
2 import sys
3 import os
4
5 filename = 'counter.json'
6
7 if len(sys.argv) != 2:
8     print("Usage: " + sys.argv[0] + " COUNTER")
9     exit()
10
11 counter = {}
12
13 if os.path.exists(filename):
14     with open(filename) as fh:
15         json_str = fh.read()
16         counter = json.loads(json_str)
17
18 name = sys.argv[1]
19 if name in counter:
20     counter[name] += 1
21 else:
22     counter[name] = 1
23
24 print(counter[name])
25
26 with open(filename, 'w') as fh:
```

```
28     json_str = json.dumps(counter)
29     fh.write(json_str)
```

Solution: Phone book

```
1 import sys
2 import json
3 import os
4
5 def main():
6     filename = 'phonebook.json'
7     phonebook = {}
8     if os.path.exists(filename):
9         with open(filename) as fh:
10             json_str = fh.read()
11             phonebook = json.loads(json_str)
12
13     if len(sys.argv) == 2:
14         name = sys.argv[1]
15         if name in phonebook:
16             print(phonebook[name])
17         else:
18             print("{} is not in the"
19                  "phonebook".format(name))
20         return
21
22     if len(sys.argv) == 3:
23         name = sys.argv[1]
24         phone = sys.argv[2]
25         phonebook[name] = phone
26         with open(filename, 'w') as fh:
27             json_str = json.dumps(phonebook)
28             fh.write(json_str)
29         return
30
31     print("Invalid number of parameters")
32     print("Usage: {} username"
33           "[phone]".format(sys.argv[0]))
34
35 if __name__ == '__main__':
36     main()
```

Command line arguments with argparse

Modules to handle the command line

You would like to allow the user to pass arguments on the command line. For example:

```
1 myprog.py --machine server_name --test name --verbose -  
-debug  
2 myprog.py -v -d  
3 myprog.py -vd  
4 myprog.py file1 file2 file3
```

- [sys.argv](#) manual parsing?
- [optparse](#) (deprecated)
- [argparse](#)

argparse

```
1 import argparse  
2  
3 parser = argparse.ArgumentParser()  
4 parser.add_argument('--name')      # optional named  
parameter that requires a value  
5 parser.add_argument('--name', help="Some description")  
6  
7 parser.add_argument('--max', help='max number of  
somthing', type=int) # check and co\br/>8 nvert to integer  
9 parser.add_argument('--verbose', action='store_true')  
# "flag" no value is expected  
10  
11 parser.add_argument('--color', '-c') # short name also
```

```
accepted
12
13
14 parser.add_argument('files', help="filenames(s)")      #
a required positional argument
15 parser.add_argument('files', nargs="*")      # 0 or more
positional
16 parser.add_argument('files', nargs="+")      # 1 or more
positional
17
18 parser.add_argument('--files', nargs="+")      # --files
a.txt b.txt c.txt
19
20
21 args = parser.parse_args()
22
23 print(args.name)
24 print(args.files)
```

Basic usage of argparse

Setting up the argparse already has some (little) added value.

```
1 import argparse
2
3 parser = argparse.ArgumentParser()
4 parser.parse_args()
5
6 print('the code...')
```

Running the script without any parameter will not interfere...

```
1 $ python argparse_basic.py
2 the code...
```

If the user tries to pass some parameters on the command line, the argparse will print an error message and stop the execution.

```
1 $ python argparse_basic.py foo
2 usage: argparse_basic.py [-h]
3 argparse_basic.py: error: unrecognized arguments: foo
```

```
1 $ python argparse_basic.py -h
2 usage: argparse_basic.py [-h]
3
4 optional arguments:
5   -h, --help  show this help message and exit
```

The minimal set up of the argparse class already provides a (minimally) useful help message.

Positional argument

```
1 import argparse
2
3 parser = argparse.ArgumentParser()
4 parser.add_argument('name', help='your full name')
5 args = parser.parse_args()
6
7 print(args.name)
```

```
1 $ python argparse_positional.py
2 usage: argparse_positional.py [-h] name
3 argparse_positional.py: error: too few arguments
```

```
1 $ python argparse_positional.py -h
2 usage: argparse_positional.py [-h] name
3
```

```
4 positional arguments:  
5   name           your full name  
6  
7 optional arguments:  
8   -h, --help    show this help message and exit
```

```
1 $ python argparse_positional.py Foo  
2 Foo
```

```
1 $ python argparse_positional.py Foo Bar  
2 usage: argparse_positional.py [-h] name  
3 argparse_positional.py: error: unrecognized arguments:  
Bar
```

```
1 $ python argparse_positional.py "Foo Bar"  
2 Foo Bar
```

Many positional argument

```
1 import argparse  
2  
3 parser = argparse.ArgumentParser()  
4 parser.add_argument('files', help='filename(s)',  
nargs='+')  
5 args = parser.parse_args()  
6  
7 print(args.files)
```

```
1 $ python argparse_positional_many.py  
2 usage: argparse_positional_many.py [-h] files [files  
...]  
3 argparse_positional_many.py: error: too few arguments
```

```
1 air:python gabor$ python argparse_positional_many.py  
a.txt b.txt  
2 ['a.txt', 'b.txt']
```

Convert to integers

```
1 import argparse
2
3 parser = argparse.ArgumentParser()
4 parser.add_argument('number', help='the number to take
to the square')
5 args = parser.parse_args()
6
7 print(args.number * args.number)
```

```
1 $ python argparse_number.py abc
```

```
1 Traceback (most recent call last):
2   File "examples/argparse/argparse_number.py", line 10,
in <module>
3     print(args.number * args.number)
4 TypeError: can't multiply sequence by non-int of type
'str'
```

Trying to convert the argument received from the command line as an integer, we get a `TypeError`. The same would happen even if a number was passed, but you could call `int()` on the parameter to convert to an integer.
However there is a better solution.

The same with the following

```
1 $ python argparse_number.py 23
```

```
1 Traceback (most recent call last):
2   File "examples/argparse/argparse_number.py", line 10,
in <module>
3     print(args.number * args.number)
4 TypeError: can't multiply sequence by non-int of type
'str'
```

Convert to integer

```
1 import argparse
2
3 parser = argparse.ArgumentParser()
4 parser.add_argument('number', help='the number to take
5 to the square', type=int)
6 args = parser.parse_args()
7 print(args.number * args.number)
```

```
1 $ argparse_type.py abc
2 usage: argparse_type.py [-h] number
3 argparse_type.py: error: argument number: invalid int
value: 'abc'
```

We got a much better error message as argparse already found out the argument was a string and not a number as expected.

```
1 $ argparse_type.py 23
2 529
```

The `type` parameter can be used to define the type restriction and type conversion of the attributes.

Named arguments

```
1 import argparse
2
3 parser = argparse.ArgumentParser()
4 parser.add_argument('--color', help='The name of the
5 color')
6 args = parser.parse_args()
7 print(args.color)
```

python argparse_named.py –color Blue

```
1 Blue
```

python argparse_named.py

```
1 None
```

Named parameters are optional by default. You can pass the required=True parameter to make them required.

Boolean Flags

```
1 import argparse
2
3 parser = argparse.ArgumentParser()
4 parser.add_argument('--color',    help='The name of the
color')
5 parser.add_argument('--verbose',  help='Print more
data',
6         action='store_true')
7 args = parser.parse_args()
8
9 print(args.color)
10 print(args.verbose)
```

python argparse_boolean.py –color Blue –verbose

```
1 Blue
2 True
```

python argparse_boolean.py

```
1 None
2 False
```

Short names

```
1 import argparse
2
3 parser = argparse.ArgumentParser()
4 parser.add_argument('--color', '-c', help='The name of
the color')
5 parser.add_argument('--verbose', '-v', help='Print
more data',
6     action='store_true')
7 args = parser.parse_args()
8
9 print(args.color)
10 print(args.verbose)
```

python argparse_shortname.py -c Blue -v

python argparse_shortname.py -vc Blue

Exercise: Command line parameters

Take the code from the color selector exercise in the files section and change it so

the user can supply the name of the file where the colors are listed using the

--file filename option.

If the user supplies an incorrect color name (which is not listed among the accepted colors)
give an error message and stop execution.

Allow the user to supply a flag called --force that will override the color-name-validity checking and will allow any color name.

Exercise: argparse positional and named

Create a script that can accept any number of filenames, the named parameter `--machine` and the flag `--verbose`.
Like this:

```
1 python ex.py file1 file2 file3 --machine MACHINE --  
verbose
```

Exception handling

Hierarchy of calls

```
1 main()
2     some_process()
3         for filename in some_list:
4             handle_file(filename)
5                 private_module.deal_with_file(filename)
6
7 private_module._helper_function(filename)
8
9 public_module.process_file(filename)
    with open(filename) as fh:
        pass
```

Handling errors as return values

- Each function that fails returns some error indicator. **None** ? An object that has and attribute “error”?
- None would be bad as that cannot indicate different errors.
- Every called needs to check if the function returned error. If at any point we forget our system might run with hidden failures.

```
1 main()
2     .....
3         result = do_something(filename)
4         if result:
5             do_something_else(result)
```

```
1 main()
2     .....
```

```
3     result = do_something(filename)
4     do_something_else(result)
```

Handling errors as exceptions

- Only need to explicitly check for it at the level where we know what to do with the problem.
 - But: Do we want our pacemaker to stop totally after missing one beat? Probably not. Or better yet: not when it is in production.
-

```
1 main()
2     try:
3         .....
4             result = do_something(filename)
5                 do_something_else(result)
6     except Exception:
7         # decide what to do
```

A simple exception

When something goes wrong, Python throws (raises) an exception. For example, trying to divide a number by 0 won't work. If the exception is not handled, it will end the execution.

In some programming languages we use the expression “throwing an exception” in other languages the expression is “raising an exception”.

I use the two expressions interchangeably.

In the next simple example, Python will print the string before the division, then it will throw an exception, printing it to the standard error that is the screen by default. Then the script stops working and the string “after” is not printed.

```
1 def div(a, b):
2     print("before")
3     print(a/b)
4     print("after")
5
6 div(1, 0)
7
8 # before
9 # Traceback (most recent call last):
10 #   File "examples/exceptions/divide_by_zero.py", line
8, in <module>
11 #     div(1, 0)
12 #   File "examples/exceptions/divide_by_zero.py", line
5, in div
13 #     print(a/b)
14 # ZeroDivisionError: integer division or modulo by
zero
```

Working on a list

In a slightly more interesting example we have a list of values. We would like to divide a number by each one of the values.

As you can see one of the values is 0 which will generate and exception.

The loop will finish early.

```
1 def div(a, b):
2     print("dividing {} by {} is {}".format(a, b, a/b))
3
4 a = 100
5 values = [2, 5, 0, 4]
6
7 for v in values:
8     div(a, v)
9
10 # dividing 100 by 2 is 50.0
11 # dividing 100 by 5 is 20.0
12 # Traceback (most recent call last):
13 # ...
14 # ZeroDivisionError: division by zero
```

We can't repair the case where the code tries to divide by 0, but it would be nice if we could get the rest of the results as well.

Catch `ZeroDivisionError` exception

For that, we'll wrap the critical part of the code in a “try” block.

After the “try” block we need to provide a list of exception that are caught by this try-block.

You could say something like “Try this code and let all the exceptions propagate, except of the ones I listed”.

As we saw in the previous example, the specific error is called `ZeroDivisionError`.

If the specified exception occurs within the `try:` block, instead of the script ending,
only the `try` block end and the `except:` block is executed.

```
1 def div(a, b):
2     print("dividing {} by {} is {}".format(a, b, a/b))
3
4 a = 100
5 values = [2, 5, 0, 4]
6
7 for v in values:
8     try:
9         div(a, v)
10    except ZeroDivisionError:
11        print("Cannot divide by 0")
12
13 # dividing 100 by 2 is 50.0
14 # dividing 100 by 5 is 20.0
15 # Cannot divide by 0
16 # dividing 100 by 4 is 25.0
```

Module to open files and calculate something

Of course in the previous example, it would be probably much easier if we just checked if the number was 0, before trying to divide with it. There are many other cases when this is not possible. For example it is impossible to check if open a file will succeed, without actually trying to open the file.

In this example we open the file, read the first line which is a number and use that for division.

When the open() fails, Python throws an IOError exception.

```
1 def read_and_divide(filename):
2     print("before " + filename)
3     with open(filename, 'r') as fh:
4         number = int(fh.readline())
5         print(100 / number)
6     print("after " + filename)
```

File for exception handling example

If we have a list of files and we would like to make sure we process as many as possible without any problem caused in the middle, we can catch the exception.

We have the following list of files.

Notice that “two.txt” is missing and “zero.txt” has a 0 in it.

1 0

1 1

File two.txt is missing on purpose.

1 3

Open files - exception

```
1 import sys
2 import module
3
4 # python open_list_of_files.py one.txt zero.txt
5 # two.txt three.txt
5 files = sys.argv[1:]
6
7 for filename in files:
8     module.read_and_divide(filename)
9
10 # before one.txt
11 # 100.0
12 # after one.txt
13 # before zero.txt
14 # Traceback (most recent call last):
15 # ...
16 # ZeroDivisionError: division by zero
```

Handle divide by zero exception

Running this code will throw the ZeroDivisionError exception, but it will die with a IOError exception.

```
1 import sys
2 import module
3
4 # python handle_divide_by_zero.py one.txt zero.txt
5 files = sys.argv[1:]
6
7 for filename in files:
8     try:
9         module.read_and_divide(filename)
10    except ZeroDivisionError:
11        print("Cannot divide by 0 in file
12        {}".format(filename))
13
14 # before one.txt
15 # 100.0
16 # after one.txt
17 # before zero.txt
18 # Cannot divide by 0 in file zero.txt
19 # before two.txt
20 # IOError: [Errno 2] No such file or directory:
'two.txt'
```

Handle files - exception

We can add multiple “except” statement at the end of the “try” block and handle several exceptions. Each one in a different way.

```
1 import sys
2 import module
3
4 # python handle_both_exceptions.py one.txt zero.txt
5 files = sys.argv[1:]
6
7 for filename in files:
8     try:
9         module.read_and_divide(filename)
10    except ZeroDivisionError:
11        print("Cannot divide by 0 in file
12 {}".format(filename))
13    except IOError:
14        print("Cannot open file {}".format(filename))
15
16 # before one.txt
17 # 100.0
18 # after one.txt
19 # before zero.txt
20 # Cannot divide by 0 in file zero.txt
21 # before two.txt
22 # Cannot open file two.txt
23 # before three.txt
24 # 33.33333333333336
25 # after three.txt
```

Catch all the exceptions and show their type

We can also use the “except Exception” to catch all exceptions. In this case we might want to also print out the text and the type of the exception by ourselves.

```
1 import sys
2 import module
3
4 # python show_exceptions_type.py one.txt zero.txt
```

```

two.txt three.txt
1 files = sys.argv[1:]
2
3 for filename in files:
4     try:
5         module.read_and_divide(filename)
6     except Exception as err:
7         print("  There was a problem in " + filename)
8         print("  Text: {}".format(err))
9         print("  Name: {}".format(type(err).__name__))
10
11 # before one.txt
12 # 100.0
13 # after one.txt
14 # before zero.txt
15 #     There was a problem in zero.txt
16 #     Text: division by zero
17 #     Name: ZeroDivisionError
18 # before two.txt
19 #     There was a problem in two.txt
20 #     Text: [Errno 2] No such file or directory:
21 #             'two.txt'
22 #     Name: FileNotFoundError
23 # before three.txt
24 # 33.33333333333336
25 # after three.txt

```

List exception types

We can list more than one exceptions to be caught one after the other in a single “except” statement.

```
1 except (IOError, ZeroDivisionError):
```

```
1 import sys
2 import module
3
```

```
 4 # python handle_both_exceptions.py one.txt zero.txt
 5 files = sys.argv[1:]
 6
 7 for filename in files:
 8     try:
 9         module.read_and_divide(filename)
10     except (ZeroDivisionError, IOError):
11         print("We have a problem with file
12             {}".format(filename))
13
14 # before one.txt
15 # 100.0
16 # after one.txt
17 # before zero.txt
18 # We have a problem with file zero.txt
19 # before two.txt
20 # We have a problem with file two.txt
21 # before three.txt
22 # 33.33333333333336
23 # after three.txt
```

Exceptions

There are many kinds of exceptions in Python and each module can define its own exception types as well.

On this page you'll find the list and hierarchy of exceptions in Python.

- [exceptions](#)

How to raise an exception

As you create more and more complex applications you'll reach a point where you write a function, probably in a module that needs to report some error condition.
You can raise an exception in a simple way.

```
1 def some():
2     raise Exception("Some Error")
3
4 def main():
5     try:
6         some()
7     except Exception as err:
8         print(err)
9         print("Type: " + type(err).__name__)
10
11 main()
12
13 # Some Error
14 # Type: Exception
```

Stack trace

```
1 import traceback
2
3 def bar():
4     foo()
5
6 def foo():
7     raise Exception("hi")
8
9 def main():
10    try:
11        bar()
12    except Exception as err:
13        track = traceback.format_exc()
14        print(track)
15
16    print("-----")
```

```
17     bar()
18
19
20 main()
```

```
1 Traceback (most recent call last):
2   File "stack_trace.py", line 11, in main
3     bar()
4   File "stack_trace.py", line 4, in bar
5     foo()
6   File "stack_trace.py", line 7, in foo
7     raise Exception("hi")
8 Exception: hi
9
10 -----
11 Traceback (most recent call last):
12   File "stack_trace.py", line 20, in <module>
13     main()
14   File "stack_trace.py", line 17, in main
15     bar()
16   File "stack_trace.py", line 4, in bar
17     foo()
18   File "stack_trace.py", line 7, in foo
19     raise Exception("hi")
20 Exception: hi
```

Exercises: Exception int conversion

- In the earlier example we learned how to handle both ZeroDivisionError and IOError exceptions. Now try this

```
1 cd examples/exceptions
2 python handle_both_exceptions.py one.txt zero.txt
two.txt text.txt three.txt
```

```
1 before one.txt
2 100.0
3 after one.txt
4 before zero.txt
5 Cannot divide by 0 in file zero.txt
```

```
6 before two.txt
7 Cannot open file two.txt
8 before text.txt
9 Traceback (most recent call last):
10   File "handle_both_exceptions.py", line 9, in
<module>
11     module.read_and_divide(filename)
12   File
"/home/gabor/work/slides/python/examples/exceptions/modul
e.py", line 4, in re\
13 ad_and_divide
14     number = int(fh.readline())
15 ValueError: invalid literal for int() with base 10:
'3.14\n'
```

- This will raise a ValueError exception before handling file three.txt
 - Fix it by capturing the specific exception.
 - Fix by capturing “all other exceptions”.
-
- 1 3.14
-

Exercises: Raise Exception

- Write a function that expects a positive integer as its single parameter.
- Raise exception if the parameter is not a number.
- Raise a different exception if the parameter is not positive.
- Raise a different exception if the parameter is not whole number.

Solution: Exception int conversion (specific)

```
1 import sys
2 import module
3
```

```
4 # python handle_both_exceptions.py one.txt zero.txt  
two.txt three.txt  
5 files = sys.argv[1:]  
6  
7 for filename in files:  
8     try:  
9         module.read_and_divide(filename)  
10    except ZeroDivisionError:  
11        print("Cannot divide by 0 in file  
{})".format(filename))  
12    except IOError:  
13        print("Cannot open file {}".format(filename))  
14    except ValueError as ex:  
15        print("ValueError {} in file {}".format(ex,  
filename))
```

```
1 before one.txt  
2 100.0  
3 after one.txt  
4 before zero.txt  
5 Cannot divide by 0 in file zero.txt  
6 before two.txt  
7 Cannot open file two.txt  
8 before text.txt  
9 ValueError invalid literal for int() with base 10:  
'3.14\n' in file text.txt  
10 before three.txt  
11 33.33333333333336  
12 after three.txt
```

Solution: Exception int conversion (all other)

```
1 import sys  
2 import module  
3  
4 # python handle_both_exceptions.py one.txt zero.txt  
two.txt three.txt  
5 files = sys.argv[1:]  
6  
7 for filename in files:  
8     try:  
9         module.read_and_divide(filename)
```

```
10     except ZeroDivisionError:
11         print("Cannot divide by 0 in file
12             {}".format(filename))
13     except IOError:
14         print("Cannot open file {}".format(filename))
15     except Exception as ex:
16         print("Exception type {} {} in file
17             {}".format(type(ex).__name__, ex, filename))
```

```
1 before one.txt
2 100.0
3 after one.txt
4 before zero.txt
5 Cannot divide by 0 in file zero.txt
6 before two.txt
7 Cannot open file two.txt
8 before text.txt
9 Exception type ValueError invalid literal for int()
with base 10: '3.14\n' in file t\
10 ext.txt
11 before three.txt
12 33.33333333333336
13 after three.txt
```

Solution: Raise Exception

```
1 def positive(num):
2     if type(num).__name__ == 'float':
3         raise Exception("The given parameter {} is a
4 float and not an int.".format(nu\
5 m))
6     if type(num).__name__ != 'int':
7         raise Exception("The given parameter {} is of
8 type {} and not int.".format(nu\
9 m, type(num).__name__))
10    if num < 0:
11        raise Exception("The given number {} is not
12 positive.".format(num))
```

```
13 for val in [14, 24.3, "hi", -10]:  
14     print(val)  
15     print(type(val).__name__)  
16     try:  
17         positive(val)  
18     except Exception as ex:  
19         print("Exception: {}".format(ex))
```

Classes - OOP - Object Oriented Programming

Why Object Oriented Programming?

- Better encapsulation of intent.
- Integration between data and functionality (attributes and methods)
- Better modelling for some part of the world.
- Another level of code-reuse.
- Clearer separation between “usage” and “implementation”.
(Private data in some cases)
- Clearer connection between “classes” of things.
- In reality: avoid using “global”.

Generic Object Oriented Programming terms

- OOP differs a lot among programming languages!
- Classes (blueprints)
- Objects / instances (actual)
- Members: Attributes and Methods
- Attributes / Properties (variables - data)
- Methods (functions) (private, public, virtual)
- Inheritance (is a)
- Composition (has a)
- Constructor
- Destructor

OOP in Python

- Everything is an object
- Numbers, strings, list, ... even classes are objects.
- Class objects
- Instance objects
- Nothing is private.

OOP in Python (numbers, strings, lists)

```
1 # numbers
2 print((255).bit_length())      # 8
3 print((256).bit_length())      # 9
4
5 # strings
6 print( "hello WOrld".capitalize() )  # Hello world
7 print( ":".join(["a", "b", "c"]) )   # a:b:c
8
9
10 # lists
11 numbers = [2, 17, 4]
12 print(numbers)                # [2, 17, 4]
13
14 numbers.append(7)
15 print(numbers)                # [2, 17, 4, 7]
16
17 numbers.sort()
18 print(numbers)                # [2, 4, 7, 17]
```

OOP in Python (argparse)

```
1 import argparse
2 def get_args():
3     parser = argparse.ArgumentParser()
4     parser.add_argument('--name')
5     parser.add_argument('--email')
6
7     print(type(parser).__name__)
8     print(parser.__class__)
```

```
9
10 # print(dir(parser))
11 print( parser.format_help() )
12 parser.print_help()
13
14 return parser.parse_args()
15
16 args = get_args()
17 print(args.__class__)
18 print(args.name)
```

Create a class

```
1 # class Person(object):
2     pass
3
4 class Person:
5     pass
6
7 if __name__ == '__main__':
8     p = Person()
9     print(p)
10    print(type(p))
11    print(p.__class__.__name__)
12
13 members = dir(p)
14 print(members)
```

```
1 <__main__.Person object at 0x7fc4e3ec1da0>
2 <class '__main__.Person'>
3 Person
4['__class__', '__delattr__', '__dict__', '__dir__', '__doc__',
5 '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__',
6 '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__',
7 '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__',
8 '__str__', '__subclasshook__', '__weakref__']
```

In Python 2.x classes needed to inherit from ‘object’ in order to become ‘new style’ classes.

Import module containing class

```
1 import ppl
2
3 p = ppl.Person()
4 print(p)                      # <person.Person object at
0x101a8a190>
5 print(type(p))                # <class 'person.Person'>
6 print(p.__class__.__name__)    # Person
```

```
1 <ppl.Person object at 0x7f973024a780>
2 <class 'ppl.Person'>
3 Person
```

Import class from module

```
1 from ppl import Person
2
3 p = Person()
4 print(p)                      # <person.Person object at
0x101a8a190>
5 print(type(p))                # <class 'person.Person'>
6 print(p.__class__.__name__)    # Person
```

Initialize a class - constructor, attributes

```
1 class Person():
2     def __init__(self, given_name):
3         self.name = given_name
4
5 if __name__ == '__main__':
6     p1 = Person("Joe")
7     print(p1)                  # <__main__.Person
object at 0x0000021EC664B358>
8     print(p1.__class__.__name__) # Person
```

```
9     print(p1.name)                      # Joe
10
11    p2 = Person("Jane")                  # <__main__.Person
12    print(p2)
object at 0x0000021EC664B470>
13    print(p2.name)                      # Jane
14
15    p1.name = "Joseph"                  # <__main__.Person
16    print(p1)
object at 0x0000021EC664B358>
17    print(p1.name)                      # Josheph
```

Attributes are not special

```
1 class Person():
2     def __init__(self, given_name):
3         self.name = given_name
4
5 if __name__ == '__main__':
6     p1 = Person("Joe")
7     print(p1.__class__.__name__)      # Person
8     print(p1.name)                  # Joe
9
10    p2 = Person("Jane")
11    print(p2.name)                  # Jane
12
13    p1.address = "Main street 12"
14    print(p1.address)              # Main street 12
15
16
17    print(p2.address)              # AttributeError:
'Person' object has no attribute\
18    'address'
```

Create Point class

```
1 import shapes
2
3 p = shapes.Point()
4 print(p)                      # <shapes.Point instance at
0x7fb58c31ccb0>
```

```
1 class Point():
2     pass
```

Initialize a class - constructor, attributes

```
1 import shapes
2
3 p1 = shapes.Point(2, 3)
4 print(p1)          # <shapes.Point instance at
0x7fb58c31ccb0>
5 print(p1.x)        # 2
6 print(p1.y)        # 3
7
8 p1.x = 7
9 print(p1.x)        # 7
```

```
1 class Point():
2     def __init__(self, a, b):
3         self.x = a
4         self.y = b
```

Methods

```
1 import shapes
2
3 p1 = shapes.Point(2, 3)
4
5 print(p1.x)      # 2
6 print(p1.y)      # 3
7
8 p1.move(4, 5)
9 print(p1.x)      # 6
10 print(p1.y)     # 8
11
12
13 print(p1)        # <shapes.Point object at
0x7fb0691c3e48>
```

```
1 class Point():
2     def __init__(self, a, b):
3         self.x = a
4         self.y = b
5
6     def move(self, dx, dy):
7         self.x += dx
8         self.y += dy
```

Stringify class

- **repr** “should” return Python-like code
- **str** should return readable representation
- If **str** does not exist, **repr** is called instead.

```
1 import shapes
2
3 p1 = shapes.Point(2, 3)
4 print(p1)      # Point(2, 3)
```

```
1 class Point():
2     def __init__(self, x, y):
3         self.x = x
4         self.y = y
5
6     def __repr__(self):
7         return 'Point({}, {})'.format(self.x, self.y)
8
9     def move(self, dx, dy):
10        self.x += dx
11        self.y += dy
```

Inheritance

```
1 class Point():
2     def __init__(self, x, y):
3         print('__init__ of Point')
4         self.x = x
```

```
5         self.y = y
6
7     def move(self, dx, dy):
8         self.x += dx
9         self.y += dy
10
11 class Circle(Point):
12     def __init__(self, x, y, r):
13         print('__init__ of Circle')
14         super().__init__(x, y)
15         self.r = r
16
17     def area(self):
18         return self.r * self.r * 3.14
```

```
1 import shapes
2
3 c = shapes.Circle(2, 3, 10)      # __init__ of Circle
4                               # __init__ of Point
5 print(c)                      # <shapes.Circle instance at
0x7fb58c31ccb0>
6 print(c.x)                    # 2
7 print(c.y)                    # 3
8 print(c.r)                    # 10
9
10 c.move(4, 5)
11 print(c.x)                   # 6
12 print(c.y)                   # 8
13 print(c.area())              # 314.0
```

Inheritance - another level

```
1 class Point():
2     def __init__(self, x, y):
3         print('__init__ of Point')
4         self.x = x
5         self.y = y
6
7 class Circle(Point):
8     def __init__(self, x, y, r):
9         print('__init__ of Circle')
```

```

10         super().__init__(x, y)
11         self.r = r
12
13     def area(self):
14         return self.r * self.r * 3.14
15
16 class Ball(Circle):
17     def __init__(self, x, y, r, z):
18         print('__init__ of Ball')
19         super().__init__(x, y, r)
20         self.z = z
21
22
23 b = Ball(2, 3, 9, 7)
24 print(b)
25 print(b.area())
26
27 # __init__ of Ball
28 # __init__ of Circle
29 # __init__ of Point
30 # <__main__.Ball object at 0x103dea190>
31 # 254.34

```

Modes of method inheritance

- Implicit
- Override
- Extend
- Delegate - Provide

Modes of method inheritance - implicit

Inherit method

```

1 class Parent():
2     def greet(self):
3         print("Hello World")
4
5 class Child(Parent):
6     pass

```

```
7
8 p = Parent()
9 p.greet()      # Hello World
10
11 c = Child()
12 c.greet()      # Hello World
```

Modes of method inheritance - override

Replace method

```
1 class Parent():
2     def greet(self):
3         print("Hello World")
4
5 class Child(Parent):
6     def greet(self):
7         print("Hi five!")
8
9 p = Parent()
10 p.greet()
11
12 c = Child()
13 c.greet()
14
15 super(Child, c).greet()
```

```
1 Hello World
2 Hi five!
3 Hello World
```

Modes of method inheritance - extend

Extend method before or after calling original.

```
1 class Parent():
2     def greet(self):
3         print("Hello World")
4
5 class Child(Parent):
```

```
6     def greet(self):
7         print("Hi five!")
8         super().greet()
9         print("This is my world!")
10
11 p = Parent()
12 p.greet()      # Hello World
13
14 c = Child()
15 c.greet()
16
17 # Hi five!
18 # Hello World
19 # This is my world!
```

Modes of method inheritance - delegate - provide

Let the child implement the functionality.

```
1 class Parent():
2     def greet(self):
3         print("Hello", self.get_name())
4
5 class Child(Parent):
6     def __init__(self, name):
7         self.name = name
8
9     def get_name(self):
10        return self.name
11
12 # Should not create instance from Parent
13 # p = Parent()
14 # p.greet()      # AttributeError: 'Parent' object has
no attribute 'get_name'
15
16 c = Child('Foo')
17 c.greet()      # Hello Foo
```

- Should we have a version of greet() in the Parent that throws an exception?
- Do we want to allow the creation of instance of the Parent class?
- Abstract Base Class (abc)

Composition - Line

When an object holds references to one or more other objects.

- Pythagorean theorem

```

1 class Point():
2     def __init__(self, x, y):
3         self.x = x
4         self.y = y
5
6 class Line():
7     def __init__(self, a, b):
8         self.a = a
9         self.b = b
10
11    def length(self):
12        return ((self.a.x - self.b.x) ** 2 + (self.a.y
13 - self.b.y) ** 2) ** 0.5
14 p1 = Point(2, 3)
15 p2 = Point(5, 7)
16 blue_line = Line(p1, p2)
17
18 print(blue_line.a) # <__main__.Point object at
0x0000022174B637B8>
19 print(blue_line.b) # <__main__.Point object at
0x0000022174B3C7B8>
20 print(blue_line.length()) # 5.0

```

Some comments

- There are no private attributes. The convention is to use leading underscore to communicate to other developers what is private.
- Using the name **self** for the current object is just a consensus.

Class in function

```
1 def creator():
2     class MyClass():
3         pass
4     o = MyClass()
5     print(o.__class__.__name__) # MyClass
6
7 creator()
8 # MyClass() # NameError: name 'MyClass' is not defined
```

Serialization of instances with pickle

```
1 import pickle
2
3 class aClass(object):
4     def __init__(self, amount, name):
5         self.amount = amount
6         self.name = name
7
8
9 the_instance = aClass(42, "FooBar")
10
11 a = {
12     "name": "Some Name",
13     "address" : ['country', 'city', 'street'],
14     'repr' : the_instance,
15 }
16
17 print(a)
18
19 pickle_string = pickle.dumps(a)
20
21 b = pickle.loads(pickle_string)
22
```

```
23 print(b)
24
25 print(b['repr'].amount)
26 print(b['repr'].name)
```

Quick Class definition and usage

```
1 class Quick(object):
2     def __init__(self, name, email):
3         self.name = name
4         self.email = email
5
6 q = Quick(name = "Foo", email = "foo@bar.com")
7 print(q.name)
8 print(q.email)
```

Exercise: Add move_rad to based on radians

- From the **Python: Methods** take the examples/classes/methods/shapes.py and add a method called **move_rad(dist, angle)** that accpets a distance and an angle and moved the point accordingly.
-

```
1 delta_x = dist * cos(angle)
2 delta_y = dist * sin(angle)
```

Exercise: Improve previous examples

- Take the previous example **Python: Inheritance - another level** and the example file called examples/classes/inheritance/ball_shape.py and change it so the **Ball** class will accept **x, y, z, r**.
- Add a method called **move** to the new Ball class that will accept **dx, dy, dz**.

- Implement a method that will return the volume of the ball.

Exercise: Polygon

- Implement a class representing a Point.
- Make the printing of a point instance nice.
- Implement a class representing a Polygon. (A list of points)
- Allow the user to “move a polygon” calling poly.move(dx, dy) that will change the coordinates of every point by (dx, dy)

```
1 class Point():
2     pass
3
4 class Polygon():
5     pass
6
7 p1 = Point(0, 0)    # Point(0, 0)
8 p2 = Point(5, 7)    # Point(5, 7)
9 p3 = Point(4, 9)    # Point(4, 9)
10 print(p1)
11 print(p2)
12 print(p3)
13 p1.move(2, 3)
14 print(p1)           # Point(2, 3)
15
16 poly = Polygon(p1, p2, p3)
17 print(poly)          # Polygon(Point(2, 3), Point(5, 7),
Point(4, 9))
18 poly.move(-1, 1)
19 print(poly)          # Polygon(Point(1, 4), Point(4, 8),
Point(3, 10))
```

Exercise: Number

Turn the Number guessing game into a class. Replace every print statement with a call to an output method.

Do the same with the way you get the input.

Then create a subclass where you override these methods.
You will be able to launch the game with a hidden value you decide upon.

The input will feed a pre-defined list of values as guesses to the game
and the output methods will collect the values that the game prints in a list.

Exercise: Library

Create a class hierarchy to represent a library that will be able to represent the following entities.

- Author (name, birthdate, books)
- Book (title, author, language, who_has_it_now?,
is_on_waiting_list_for_whom?)
- Reader (name, birthdate, books_currently_lending)

Methods:

- write_book(title, language,)

Exercise: Bookexchange

It is like the library example, but instead of having a central library with books,
each person owns and lends out books to other people.

Exercise: Represent turtle graphics

There is a cursor (or turtle) in the x-y two-dimensional sphere. It has some (x,y) coordinates.

It can go forward n pixels. It can turn left n degrees. It can lift up the pencil or put it down.

Solution - Polygon

```
1 class Point:
2     def __init__(self, x, y):
3         self.x = x
4         self.y = y
5
6     def __repr__(self):
7         return "Point({}, {})".format(self.x, self.y)
8
9     def move(self, dx, dy):
10        self.x += dx
11        self.y += dy
12
13 class Polygon:
14     def __init__(self, *args):
15         self.points = args
16
17     def __repr__(self):
18         return 'Polygon(' + ', '.join(map(lambda p:
19             str(p), self.points)) + ')'
20
21     def move(self, dx, dy):
22         for p in self.points:
23             p.move(dx, dy)
24
25 p1 = Point(0, 0)    # Point(0, 0)
26 p2 = Point(5, 7)    # Point(5, 7)
27 p3 = Point(4, 9)    # Point(4, 9)
28
29 print(p1)
30 print(p2)
31 print(p3)
32 p1.move(2, 3)
33 print(p1)           # Point(2, 3)
34
35 poly = Polygon(p1, p2, p3)
36 print(poly)          # Polygon(Point(2, 3), Point(5, 7),
37 Point(4, 9))
38 poly.move(-1, 1)
```

```
36 print(poly)          # Polygon(Point(1, 4), Point(4, 8),  
Point(3, 10))
```

PyPi - Python Package Index

What is PyPi?

- [pypi](#)

Easy Install

- [setuptools](#)

```
1 $ easy_install module_name
```

pip

```
1 $ pip install package_name
```

Upgrade pip

- **pip install –upgrade pip** Will probably not work on Windows because file is in use...
- **easy_install pip** Will work on Windows as well.

PYTHONPATH

```
1 export PYTHONPATH=~/python
2 easy_install -d ~/python Genshi
```

Virtualenv

```
1 $ pip install virtualenv  
2  
3 $ cd project_dir  
4 $ virtualenv venv  
5 $ source venv/bin/activate  
6 $ ...  
7 $ deactivate
```

On Windows:

```
1 venv\Source\activate.bat
```

The **virtualenv** command will create a copy of python in the given directory inside the current directory.

In the above example it will create the copy in the ‘venv’ directory inside the ‘project_dir’.

After source-ing the ‘activate’ file the PATH will include the local python with a local version of **pip** and **easy_install**. This requires bash or zsh.

See also the [Python guide](#).

Virtualenv for Python 3

```
1 virtualenv -p python3 venv3  
2 source venv3/bin/activate  
3 ...  
4 deactivate
```

SQLite Database Access

SQLite

- [sqlite3](#)

Connecting to SQLite database

```
1 import sqlite3
2
3 conn = sqlite3.connect("sample.db")
4 c = conn.cursor()
5
6 # use the database here
7
8 conn.close()
```

Create TABLE in SQLite

execute and commit

```
1 import sqlite3
2
3 conn = sqlite3.connect("sample.db")
4 c = conn.cursor()
5
6 try:
7     c.execute('''CREATE TABLE companies (
8         id PRIMARY KEY,
9         name VARCHAR(100) UNIQUE NOT NULL,
10        employees INTEGER DEFAULT 0)'''')
11 except sqlite3.OperationalError as e:
12     print('sqlite error:', e.args[0])    # table
13    companies already exists
```

```
14 conn.commit()
15
16 conn.close()
17
18 print('done')
```

INSERT data into SQLite database

Use placeholders (?) supply the data in tuples.

```
1 import sqlite3
2
3 conn = sqlite3.connect("sample.db")
4 c = conn.cursor()
5
6 my_company = 'Acme'
7
8 try:
9     c.execute('''INSERT INTO companies (name) VALUES
10    (?)''', (my_company,))
11 except sqlite3.IntegrityError as e:
12     print('sqlite error: ', e.args[0]) # column name is
13     not unique
14 conn.commit()
15
16 companies = [
17     ('Foo', 12),
18     ('Bar', 7),
19     ('Moo', 99),
20 ]
21
22 try:
23     sql = '''INSERT INTO companies (name, employees)
24    VALUES (?, ?)'''
25     c.executemany(sql, companies)
26 except sqlite3.IntegrityError as e:
27     print('sqlite error: ', e.args[0]) # column name is
28     not unique
29 conn.commit()
30
31
32 conn.close()
```

29

```
30 print('done')
```

UPDATE works quite similar, but it might have a WHERE clause.

SELECT data from SQLite database

```
1 import sqlite3
2
3 conn = sqlite3.connect("sample.db")
4 c = conn.cursor()
5
6 minimum = 0
7
8 sql = '''SELECT * FROM companies WHERE employees >=
?'''
9 for company in c.execute(sql, (minimum,)):
10     print(company)
11
12 sql = '''SELECT COUNT(*) FROM companies WHERE
employees >= ?'''
13 c.execute(sql, (minimum,))
14 print(c.fetchone()[0])
15
16 conn.close()
```

Use the result as an iterator, or call the fetchone method. If the result set might be empty, then the fetchone might return None. Check for it!

A counter

```
1 """
2     Counter using an SQLite backend
3     --list           list all the counters
4     --start name    creates the counter for 'name'
5     name            counts for 'name'
6 """
```

```
7
8 import sys
9 import os
10 import sqlite3
11
12 database_file = "counter.db"
13
14 def usage():
15     print('TODO print doc')
16     conn.close()
17     exit()
18
19 def main():
20     global conn
21     conn = sqlite3.connect(database_file)
22     c = conn.cursor()
23     try:
24         c.execute('''CREATE TABLE counters (
25             id PRIMARY KEY,
26             name VARCHAR(100) UNIQUE NOT NULL,
27             count INTEGER NOT NULL
28         )''' )
29     except sqlite3.OperationalError as e:
30         pass
31         # print('sqlite error:', e.args[0]) # table
32         # already exists
33
34     # print(len(sys.argv))
35     # print(sys.argv)
36
37     if len(sys.argv) == 1:
38         usage()
39
40     if len(sys.argv) == 2:
41         if sys.argv[1] == '--list':
42             print('List counters:')
43             for r in c.execute("SELECT name FROM
44             counters"):
45                 print(r[0])
46                 exit()
47             name = sys.argv[1]
48             c.execute("SELECT count FROM counters WHERE
49             name = ?", (name,))
50             line = c.fetchone()
```

```

48         if line == None:
49             print("Invalid counter name
'{ } '".format(name))
50             exit()
51         value = line[0]
52         value = value +1
53         c.execute("UPDATE counters SET count=? WHERE
name = ?", (value, name))
54         conn.commit()
55         print("{} {}".format(name, value))
56         #print("increment counter {} was:
{} ".format(name, value))
57         exit()
58
59     if len(sys.argv) == 3 and sys.argv[1] == '--
start':
60         name = sys.argv[2]
61         print("Start counter", name)
62         try:
63             c.execute("INSERT INTO counters (name,
count) VALUES(?,?)", (name, 0))
64             conn.commit()
65         except sqlite3.IntegrityError:
66             print("Name '{}' already
exists".format(name))
67             exit()
68
69         exit()
70
71     print('none')
72     usage()
73
74 main()
75
76 #print "TODO get the value of 'name' from the
database"
77 # if it was not there then add
78
79
80 #try:
81 #    c.execute('''INSERT INTO companies (name) VALUES
('Stonehenge')''')
82 #except sqlite3.IntegrityError as e:
83 #    print 'sqlite error: ', e.args[0] # column name is
not unique

```

```
84  
85 #conn.commit()  
86  
87 #conn.close()  
88  
89 #print "done"
```

MySQL

Install MySQL support

- Anaconda on MS Windows: **conda install mysql-connector-python**
- Otherwise: **pip install mysql-connector**

Create database user (manually)

```
1 $ mysql -u root -p
2
3     SHOW DATABASES;
4
5     CREATE USER 'foobar'@'localhost' IDENTIFIED BY 'no
secret';
6     GRANT ALL PRIVILEGES ON fb_db . * TO
'foobar'@'localhost';
7     GRANT ALL PRIVILEGES ON * . * TO 'foobar'@'%'
IDENTIFIED BY 'no secret';
8     FLUSH PRIVILEGES;
9
10    exit
```

```
1 vim /etc/mysql/mysql.conf.d/mysqld.cnf
2 comment out
3 # bind-address          = 127.0.0.1
4
5 service mysql restart
```

Create database (manually)

```
1 $ mysql -u foobar -p
2
```

```
3 CREATE DATABASE fb_db;
4
5 DROP DATABASE fb_db;
6 exit
```

Create table (manually)

```
1 $ mysql -u foobar -p
2
3 USE fb_db;
4 CREATE TABLE person (
5     id INTEGER PRIMARY KEY AUTO_INCREMENT,
6     name VARCHAR(255),
7     birthdate DATE,
8     score REAL
9 );
10
11 INSERT INTO person (name, birthdate, score)
12     VALUES ("Foo Bar", "1998-05-23", 42.1)
```

Connect to MySQL

```
1 import mysql.connector
2
3 def main():
4     conn = mysql.connector.connect(
5         host = 'localhost',
6         database = 'fb_db',
7         user = 'foobar',
8         password='no secret')
9
10    print("Connected:", conn)
11
12    conn.close()
13
14 if __name__ == "__main__":
15     main()
```

```
1 $ python3 examples/mysql/connect.py
```

- Change some of the parameters and try again

Connect to MySQL and Handle exception

```

1 import mysql.connector
2
3 def main():
4     try:
5         conn = mysql.connector.connect(
6             host = 'localhost',
7             database = 'fb_db',
8             user = 'foobar',
9             password='no secret')
10    except mysql.connector.Error as e:
11        print("MySQL exception: ", e)
12        return
13    #except Exception as e:
14    #    print("Other exception", e);
15    #    return
16
17    print("Connected:", conn)
18
19    conn.close()
20
21 if __name__ == "__main__":
22     main()

```

Select data

```

1 import mysql.connector
2
3
4 def main():
5     conn = mysql.connector.connect(
6         host = 'localhost',
7         database = 'fb_db',
8         user = 'foobar',
9         password='no secret')
10
11     cursor = conn.cursor()
12     cursor.execute("SELECT * FROM person")

```

```
13
14     row = cursor.fetchone()
15     print(row)
16
17     # cursor.close()  #
mysql.connector.errors.InternalError: Unread result
found.
18     conn.close()
19
20 if __name__ == "__main__":
21     main()
```

Select more data

```
1 import mysql.connector
2
3
4 def main():
5     conn = mysql.connector.connect(
6         host = 'localhost',
7         database = 'fb_db',
8         user = 'foobar',
9         password='no secret')
10
11    cursor = conn.cursor()
12    cursor.execute("SELECT * FROM person")
13
14    while True:
15        row = cursor.fetchone()
16        if not row:
17            break
18        print(row)
19
20    cursor.close()
21    conn.close()
22
23 if __name__ == "__main__":
24     main()
```

Select all data fetchall

```
1 import mysql.connector
2
3
4 def main():
5     conn = mysql.connector.connect(
6         host = 'localhost',
7         database = 'fb_db',
8         user = 'foobar',
9         password='no secret')
10
11    cursor = conn.cursor()
12    cursor.execute("SELECT * FROM person")
13
14    rows = cursor.fetchall()
15
16    print(len(rows))
17    for row in rows:
18        print(row)
19
20    cursor.close()
21    conn.close()
22
23 if __name__ == "__main__":
24     main()
```

Select some data fetchmany

```
1 import mysql.connector
2
3
4 def main():
5     conn = mysql.connector.connect(
6         host = 'localhost',
7         database = 'fb_db',
8         user = 'foobar',
9         password='no secret')
10
11    cursor = conn.cursor()
12    cursor.execute("SELECT * FROM person")
13
14    size = 2
15
```

```
16 while True:
17     rows = cursor.fetchmany(size)
18     if not rows:
19         break
20     print(len(rows))
21     for row in rows:
22         print(row)
23
24 cursor.close()
25 conn.close()
26
27 if __name__ == "__main__":
28     main()
```

Select some data WHERE clause

Bobby Tables

```
1 import mysql.connector
2
3
4 def main(min_score):
5     conn = mysql.connector.connect(
6         host = 'localhost',
7         database = 'fb_db',
8         user = 'foobar',
9         password='no secret')
10
11     cursor = conn.cursor()
12     cursor.execute("SELECT * FROM person WHERE score >
% s", (min_score,))
13
14     size = 2
15
16     while True:
17         rows = cursor.fetchmany(size)
18         if not rows:
19             break
20         print(len(rows))
21         for row in rows:
22             print(row)
23
```

```
24     cursor.close()
25     conn.close()
26
27 if __name__ == "__main__":
28     main(40)
```

Select into dictionaries

```
1 import mysql.connector
2
3
4 def main():
5     conn = mysql.connector.connect(
6         host = 'localhost',
7         database = 'fb_db',
8         user = 'foobar',
9         password='no secret')
10
11    cursor = conn.cursor(dictionary=True)
12    cursor.execute("SELECT * FROM person")
13
14    for row in cursor:
15        print(row)
16
17    cursor.close()
18    conn.close()
19
20 if __name__ == "__main__":
21     main()
```

Insert data

```
1 import mysql.connector
2
3
4 def main(name, birthdate, score):
5     conn = mysql.connector.connect(
6         host = 'localhost',
7         database = 'fb_db',
8         user = 'foobar',
9         password='no secret')
```

```
10
11     cursor = conn.cursor()
12     cursor.execute(
13         "INSERT INTO person (name, birthdate, score)
14 VALUES (%s, %s, %s)",
15         (name, birthdate, score))
16
17     if cursor.lastrowid:
18         print('last insert id', cursor.lastrowid)
19     else:
20         print('last insert id not found')
21     conn.commit()
22
23     conn.close()
24
25 if __name__ == "__main__":
26     main('Monty Python', '1969-10-05', 100)
```

Update data

```
1 import mysql.connector
2
3
4 def main(uid, score):
5     conn = mysql.connector.connect(
6         host = 'localhost',
7         database = 'fb_db',
8         user = 'foobar',
9         password='no secret')
10
11     cursor = conn.cursor()
12     cursor.execute("UPDATE person SET score=%s WHERE
13 id=%s",
14         (score, uid))
15     conn.commit()
16
17     conn.close()
18
19 if __name__ == "__main__":
20     main(12, 32)
```

Delete data

```
1 import mysql.connector
2
3
4 def main(uid):
5     conn = mysql.connector.connect(
6         host = 'localhost',
7         database = 'fb_db',
8         user = 'foobar',
9         password='no secret')
10
11     cursor = conn.cursor()
12     cursor.execute("DELETE FROM person WHERE id=%s",
13 (uid,))
14     conn.commit()
15
16     conn.close()
17 if __name__ == "__main__":
18     main(11)
```

Exercise MySQL

1. Create a user with a password manually.
2. Create a database manually.
3. Create a table manually for describing fleet of cars: id, license-plate, year-built, brand, owner. (Owner is the name of the owner)
4. Create a program that accepts values on the command line and inserts the data in the database
5. Create another program that lists all the cars.
6. Improve the selector program to accept command line paramter –minage N and –maxage N and show the cars within those age limits (N is a number of years e.g. 3)
7. Create program to delete a car.
8. Create program to change the owner of a car.

Exercise: MySQL Connection

Instead of hard-coding the connection details in the script, let's create an INI file that contains the connection information and use that.

```
1 [development]
2 host      = localhost
3 database  = fb_db
4 user      = foobar
5 password  = no secret
```

Solution: MySQL Connection

```
1 import configparser
2 import mysql.connector
3
4 config_file = 'examples/mysql/connect.ini'
5
6 def read_config(section = 'development'):
7     print(section)
8     cp = configparser.ConfigParser()
9     cp.read(config_file)
10    if not cp.has_section(section):
11        raise Exception("No configuration found for
12'{ }'".format(section))
13
14    return cp[section]
15
16 def main():
17     try:
18         db = read_config()
19         print(db['password'])
20         print(db)
21         conn = mysql.connector.connect(**db)
22     except mysql.connector.Error as e:
23         print("MySQL exception: ", e)
24         return
25     except Exception as e:
26         print("Other exception", e);
27         return
```

```
27
28     if conn.is_connected():
29         print("is connected")
30     print("Connected:", conn)
31
32     conn.close()
33
34 if __name__ == "__main__":
35     main()
```

PostgreSQL

PostgreSQL install

```
1 $ sudo aptitude install postgresql
2
3 $ sudo -i -u postgres
4 $ createuser --interactive
5 Add "ubuntu" as superuser      (we need a username that
matches our Linux username)
6 $ createdb testdb
7
8 $ psql
9 $ sudo -u postgres psql
10
11 $ psql testdb
12 testdb=# CREATE TABLE people (id INTEGER PRIMARY KEY,
name VARCHAR(100));
```

Python and Postgresql

```
1 $ sudo aptitude install python3-postgresql
2 $ sudo aptitude install python3-psycopg2
```

PostgreSQL connect

```
1 import psycopg2
2
3 try:
4     conn = psycopg2.connect("postgresql://testdb")
5     #conn = psycopg2.connect("dbname='testdb'
user='ubuntu' host='localhost' password\
6 d='secret'")
7 except Exception as e:
```

```
8     print("I am unable to connect to the database: ",  
e)
```

INSERT

```
1 import psycopg2  
2  
3 try:  
4     conn = psycopg2.connect("postgresql://testdb")  
5 except Exception as e:  
6     print("I am unable to connect to the database: ",  
e)  
7  
8 cur = conn.cursor()  
9  
10 uid = 1  
11 name = 'Foo'  
12  
13 try:  
14     cur.execute("INSERT INTO people (id, name) VALUES  
(%s, %s)", (uid, name))  
15     conn.commit()  
16 except Exception as e:  
17     print(e)
```

```
1 duplicate key value violates unique constraint  
"people_pkey"  
2 DETAIL: Key (id)=(1) already exists.
```

INSERT (from command line)

```
1 import psycopg2  
2 import sys  
3  
4 if len(sys.argv) != 3:  
5     exit("Usage: {} ID NAME".format(sys.argv[0]))  
6  
7 uid, name = sys.argv[1:]  
8  
9
```

```
10 try:
11     conn = psycopg2.connect("postgresql://testdb")
12 except Exception as e:
13     print("I am unable to connect to the database: ", e)
14
15 cur = conn.cursor()
16
17 try:
18     cur.execute("INSERT INTO people (id, name) VALUES (%s, %s)", (uid, name))
19     conn.commit()
20 except Exception as e:
21     print(e)
```

SELECT

```
1 import psycopg2
2
3 try:
4     conn = psycopg2.connect("postgresql://testdb")
5 except Exception as e:
6     print("I am unable to connect to the database: ", e)
7
8 cur = conn.cursor()
9
10 try:
11     cur.execute("SELECT * from people")
12     for r in cur.fetchall():
13         print(r)
14 except Exception as e:
15     print(e)
```

DELETE

```
1 import psycopg2
2
3 try:
4     conn = psycopg2.connect("postgresql://testdb")
```

```
5 except Exception as e:  
6     print("I am unable to connect to the database: ",  
e)  
7  
8 cur = conn.cursor()  
9  
10 try:  
11     cur.execute("DELETE FROM people")  
12     conn.commit()  
13 except Exception as e:  
14     print(e)  
15  
16 try:  
17     cur.execute("SELECT * from people")  
18     for r in cur.fetchall():  
19         print(r)  
20 except Exception as e:  
21     print(e)
```

SQLAlchemy

SQLAlchemy hierarchy

- ORM
- Table, Metadata, Reflection, DDL - standardized language
- Engine - standardize low-level access (placeholders)

SQLAlchemy engine

```
1 engine = create_engine('sqlite:///test.db')
# relative path
2 engine =
create_engine('sqlite:///full/path/to/test.db') # full
path
3 engine = create_engine('sqlite:///')
# in memory database
```

PostgreSQL

```
1 engine =
create_engine('postgresql://user:password@hostname/dbname')
'
2 engine =
create_engine('postgresql+psycopg2://user:password@hostna
me/dbname')
```

MySQL

```
1 engine =
create_engine("mysql://user:password@hostname/dbname",
encoding='latin1') #\
2 defaults to utf-8
```

SQLAlchemy autocommit

Unlike the underlying database engines, SQLAlchemy uses autocommit.

That is, usually we don't need to call `commit()`, but if we would like to have a transaction we need to start it using `begin()` and end it either with `commit()` or with `rollback()`.

SQLAlchemy engine CREATE TABLE

```
1 import os
2 from sqlalchemy import create_engine
3
4 dbname = 'test.db'
5 if os.path.exists(dbname):
6     os.unlink(dbname)
7
8 engine = create_engine('sqlite:///{} + dbname) # Engine
9
10 engine.execute '''
11     CREATE TABLE person (
12         id INTEGER PRIMARY KEY,
13         name VARCHAR(100) UNIQUE,
14         balance INTEGER NOT NULL
15     );
16 '''
```

SQLAlchemy engine INSERT

```
1 import os
2 from sqlalchemy import create_engine
3
4 dbname = 'test.db'
5
6 engine = create_engine('sqlite:///{} + dbname)
7
8 engine.execute('INSERT INTO person (name, balance)
```

```
VALUES (:name, :balance)', name =\
9  'Joe', balance = 100)
10 engine.execute('INSERT INTO person (name, balance)
VALUES (:name, :balance)', name =\
11  'Jane', balance = 100)
12 engine.execute('INSERT INTO person (name, balance)
VALUES (:name, :balance)', name =\
13  'Melinda', balance = 100)
14 engine.execute('INSERT INTO person (name, balance)
VALUES (:name, :balance)', name =\
15  'George', balance = 100)
```

SQLAlchemy engine SELECT

```
1 from sqlalchemy import create_engine
2
3 dbname = 'test.db'
4
5 engine = create_engine('sqlite:///+' + dbname)
6 result = engine.execute('SELECT * FROM person WHERE
id=:id', id=3)
7
8 print(result)          #
<sqlalchemy.engine.result.ResultProxy object at
0x1013c9d\
9 a0>
10
11 row = result.fetchone()
12 print(row)           # (3, 'Melinda', 100) - Its
a tuple
13 print(row['name'])    # Melinda           - And
a dictionary
14 print(row.name)      # Melinda - and object with
methods for the columns
15
16 for k in row.keys():   # keys also works on it
17     print(k)           # id, name, balance
18
19 result.close()
```

SQLAlchemy engine SELECT all

```
 1 import os
 2 from sqlalchemy import create_engine
 3
 4 dbname = 'test.db'
 5
 6 engine = create_engine('sqlite:///+' + dbname)
 7 result = engine.execute('SELECT * FROM person')
 8
 9 for row in result:
10     print(row)
11
12 result.close()
13
14 # (1, 'Joe', 100)
15 # (2, 'Jane', 100)
16 # (3, 'Melinda', 100)
17 # (4, 'George', 100)
```

SQLAlchemy engine SELECT fetchall

```
 1 from sqlalchemy import create_engine
 2
 3 dbname = 'test.db'
 4
 5 engine = create_engine('sqlite:///+' + dbname)
 6 result = engine.execute('SELECT * FROM person WHERE id
>= :id', id=3)
 7
 8 rows = result.fetchall()
 9 print(rows)          # [(3, 'Melinda', 100), (4,
'George', 100)]
10
11 result.close()
```

SQLAlchemy engine SELECT aggregate

```
 1 from sqlalchemy import create_engine
 2
 3 dbname = 'test.db'
 4
 5 engine = create_engine('sqlite:///+' + dbname)
```

```
6 result = engine.execute('SELECT COUNT(*) FROM person')
7
8 r = result.fetchone()[0]
9 print(r)
10
11 result.close()
```

SQLAlchemy engine SELECT IN

```
1 from sqlalchemy import create_engine
2
3 dbname = 'test.db'
4
5 engine = create_engine('sqlite:///{} + dbname)
6
7 results = engine.execute("SELECT * FROM person WHERE
name IN ('Joe', 'Jane')")
8 print(results.fetchall()) # [(2, 'Jane', 100), (1,
'Joe', 100)]
9
10 # engine.execute("SELECT * FROM person WHERE name IN
(:a0, :a1)", a0 = 'Joe', a1 = '\
11 Jane')
```

SQLAlchemy engine SELECT IN with placeholders

```
1 from sqlalchemy import create_engine
2
3 dbname = 'test.db'
4
5 engine = create_engine('sqlite:///{} + dbname)
6
7
8 names = ['Joe', 'Jane']
9 placeholders = []
10 data = {}
11 for i in range(len(names)):
12     placeholders.append(':a' + str(i))
13     data['a' + str(i)] = names[i]
```

```
14
15 # print(placeholders)    # [':a0', ':a1']
16 # print(data)           # {'a0': 'Joe', 'a1': 'Jane'}
17
18 sql = "SELECT * FROM person WHERE name IN
19     ({}).format(', '.join(placeholders))
20
21 #results = engine.execute(sql, a0 = 'Jane', a1 =
22     'Joe')
23 results = engine.execute(sql, **data)
24 print(results.fetchall()) # [(2, 'Jane', 100), (1,
25     'Joe', 100)]
```

SQLAlchemy engine connection

```
1 from sqlalchemy import create_engine
2
3 dbname = 'test.db'
4
5 engine = create_engine('sqlite:///{}' + dbname)
6
7 conn = engine.connect()
8 results = conn.execute('SELECT balance, name FROM
9 person WHERE id < :id', id = 3)
10 print(results.fetchall())   # [(100, 'Joe'), (100,
11     'Jane')]
12 conn.close()
```

SQLAlchemy engine transaction

```
1 from sqlalchemy import create_engine
2
3 dbname = 'test.db'
4
5 engine = create_engine('sqlite:///{}' + dbname)
6
7 conn = engine.connect()
8
9 trans = conn.begin()
```

```

10
11 src = 'Joe'
12 dst = 'Jane'
13 payment = 3
14
15 results = conn.execute("SELECT balance, name FROM
person WHERE name = :name", name =\
16 src)
17 src_balance = results.fetchone()[0]
18 results.fetchall()
19 print(src_balance)
20
21
22 results = conn.execute("SELECT balance, name FROM
person WHERE name = :name", name =\
23 dst)
24 dst_balance = results.fetchone()[0]
25 results.fetchall()
26 print(dst_balance)
27
28 conn.execute('UPDATE person SET balance = :balance
WHERE name=:name', balance = src_\
29 balance - payment, name = src)
30 conn.execute('UPDATE person SET balance = :balance
WHERE name=:name', balance = dst_\
31 balance + payment, name = dst)
32
33 trans.commit()
34
35 # trans.rollback()
36
37 conn.close()
38
39 results = engine.execute("SELECT * FROM person")
40 print(results.fetchall())

```

SQLAlchemy engine using context managers

```

1 with engine.begin() as trans:
2     conn.execute(...)
3     conn.execute(...)
4     raise Exception()  # for rollback

```

Exercise: Create table

Create the following schema

```
1 CREATE TABLE node (
2     id      INTEGER PRIMARY KEY,
3     name    VARCHAR(100)
4 );
5
6 CREATE TABLE interface (
7     id      INTEGER PRIMARY KEY,
8     node_id INTEGER NOT NULL,
9     ipv4    VARCHAR(15) UNIQUE,
10    ipv6    VARCHAR(80) UNIQUE,
11    FOREIGN KEY (node_id) REFERENCES node(id)
12 );
13
14 CREATE TABLE connection (
15    a      INTEGER NOT NULL,
16    b      INTEGER NOT NULL,
17    FOREIGN KEY (a) REFERENCES interface(id),
18    FOREIGN KEY (b) REFERENCES interface(id)
19 );
```

Insert a few data items. Write a few select statements.

SQLAlchemy MetaData

Describe the Schema, the structure of the database (tables, columns, constraints, etc.) in Python.

- SQL generation from the metadata, generate to a schema.
- Reflection (Introspection) - Create the metadata from an existing database, from an existing schema.

```
1 from sqlalchemy import MetaData
2 from sqlalchemy import Table, Column
3 from sqlalchemy import Integer, String
4
```

```
5 metadata = MetaData()
6 user_table = Table('user', metadata,
7                     Column('id', Integer,
primary_key=True),
8                     Column('name', String(100),
unique=True),
9                     Column('balance', Integer,
nullable=False)
10                    )
11 print(user_table.name)
12 print(user_table.c.name)
13 print(user_table.c.id)
14
15 print(user_table.c)
16 print(user_table.columns) # A bit like a Python
dictionary, but it is an associativ\
17 e array
18
19
20
21 from sqlalchemy import create_engine
22 engine = create_engine('sqlite://')
23 metadata.create_all(engine)
24
25 from sqlalchemy import ForeignKey
26
27 address_table = Table('address', metadata,
28                     Column('id', Integer,
primary_key=True),
29                     Column('stree', String(100)),
30                     Column('user_id', Integer,
ForeignKey('user.id'))
31                    )
32 address_table.create(engine)
33
34 from sqlalchemy import Unicode, UnicodeText,
ForeignKeyConstraint, DateTime
35
36 story_table = Table('story', metadata,
37                     Column('id', Integer,
primary_key=True),
38                     Column('version', Integer,
primary_key=True),
39                     Column('headline', Unicode(100),
nullable=False),
```

```
40                                         Column('body', UnicodeText)
41                                         )
42 published_table = Table('published', metadata,
43                         Column('id', Integer,
44                         primary_key=True),
45                         Column('timestamp', DateTime,
46                         nullable=False),
47                         Column('story_id', Integer,
48                         nullable=False),
49                         Column('version', Integer,
50                         nullable=False),
51                         ForeignKeyConstraint(
52                             ['story_id', 'version_id'],
53                             ['story.story_id',
54                             'story.version_id'])
55                         )
56
57 conn.execute(user_table.insert(), [
58     {'username': 'Jack', 'fullname': 'Jack Burger'},
59     {'username': 'Jane', 'fullname': 'Jane Doe'}
60 ])
61
62 from sqlalchemy import select
63 select_stmt = select([user_table.c.username,
64 user_table.c.fullname]).where(user_table\
65 c.username == 'ed')
66 result = conn.execute(select_stmt)
67 for row in result:
68     print(row)
69
70 select_stmt = select([user_table]).where(
71     or_(
72         user_table.c.username == 'ed',
73         user_table.c.username == 'wendy'
74     )
75
76 joined_obj = user_table.join(address_table,
77 user_table.c.id == address_table.c.user_id)
```

SQLAlchemy types

- Integer() - INT
- String() - ASCII strings - VARCHAR
- Unicode() - Unicode string - VARCHAR or NVARCHAR depending on database
- Boolean() - BOOLEAN, INT, TINYINT depending on db support for boolean type
- DateTime() - DATETIME or TIMESTAMP returns Python datetime() objects.
- Float() - floating point values
- Numeric() - precision numbers using Python Decimal()

SQLAlchemy ORM - Object Relational Mapping

- Domain model
- Mapping between Domain Object - Table Row

SQLAlchemy ORM create

```
 1 import os
 2 from sqlalchemy import Column, ForeignKey, Integer,
String
 3 from sqlalchemy.ext.declarative import
declarative_base
 4 from sqlalchemy.orm import relationship
 5 from sqlalchemy import create_engine
 6
 7 Base = declarative_base()
 8
 9
10 class Person(Base):
11     __tablename__ = 'person'
```

```

12     id      = Column(Integer, primary_key=True)
13     name   = Column(String(250), nullable=False,
unique=True)
14
15 class Genre(Base):
16     __tablename__ = 'genre'
17     id = Column(Integer, primary_key=True)
18     name = Column(String(250), nullable=False,
unique=True)
19
20 class Movie(Base):
21     __tablename__ = 'movie'
22     id = Column(Integer, primary_key=True)
23     title = Column(String(250), nullable=False,
unique=True)
24     genre_id = Column(Integer, ForeignKey('genre.id'))
25     genre = relationship(Genre)
26
27 class Cast(Base):
28     __tablename__ = 'cast'
29     id = Column(Integer, primary_key=True)
30     character = Column(String(250))
31     person_id = Column(Integer,
ForeignKey('person.id'))
32     movie_id = Column(Integer, ForeignKey('movie.id'))
33
34
35
36 if __name__ == '__main__':
37     dbname = 'imdb.db'
38     if os.path.exists(dbname):
39         os.unlink(dbname)
40     engine = create_engine('sqlite:/// ' + dbname)
41     Base.metadata.create_all(engine)

```

SQLAlchemy ORM schema

```
1 echo .schema | sqlite3 imdb.db
```

```
1 CREATE TABLE person (
2     id INTEGER NOT NULL,
3     name VARCHAR(250) NOT NULL,
```

```

4         PRIMARY KEY (id)
5 ) ;
6 CREATE TABLE genre (
7     id INTEGER NOT NULL,
8     title VARCHAR(250),
9     PRIMARY KEY (id)
10) ;
11 CREATE TABLE movie (
12     id INTEGER NOT NULL,
13     title VARCHAR(250),
14     genre_id INTEGER,
15     PRIMARY KEY (id),
16     FOREIGN KEY(genre_id) REFERENCES genre (id)
17) ;
18 CREATE TABLE "cast" (
19     id INTEGER NOT NULL,
20     character VARCHAR(250),
21     person_id INTEGER,
22     movie_id INTEGER,
23     PRIMARY KEY (id),
24     FOREIGN KEY(person_id) REFERENCES person (id),
25     FOREIGN KEY(movie_id) REFERENCES movie (id)
26) ;

```

SQLAlchemy ORM reflection

```

1 from sqlalchemy import create_engine
2 from sqlalchemy.orm import Session
3 from sqlalchemy.ext.automap import automap_base
4
5 Base = automap_base()
6
7 dbname = 'imdb.db'
8 engine = create_engine('sqlite:///{} + dbname)
9
10 Base.prepare(engine, reflect=True)
11 Genre = Base.classes.genre
12
13 print(Genre.metadata.sorted_tables)
14
15 for c in Base.classes:
16     print(c)
17

```

```
18 #session = Session(engine)
19 #session.add(Address(email_address="foo@bar.com",
20 user=User(name="foo")))
21 #session.commit()
```

SQLAlchemy ORM INSERT after automap

```
1 from sqlalchemy import create_engine
2 from sqlalchemy.orm import Session
3 from sqlalchemy.ext.automap import automap_base
4
5 Base = automap_base()
6
7 dbname = 'imdb.db'
8 engine = create_engine('sqlite:///{} + dbname)
9
10 Base.prepare(engine, reflect=True)
11 Genre = Base.classes.genre
12 Movie = Base.classes.movie
13 Person = Base.classes.person
14 Cast = Base.classes.cast
15
16
17
18 session = Session(engine)
19 for name in ('Action', 'Animation', 'Comedy',
'Documentary', 'Family', 'Horror'):
20     session.add(Genre(name = name))
21
22 session.add(Movie(title = "Sing", genre_id=2))
23 session.add(Movie(title = "Moana", genre_id=2))
24 session.add(Movie(title = "Trolls", genre_id=2))
25 session.add(Movie(title = "Power Rangers",
genre_id=1))
26
27 session.commit()
```

SQLAlchemy ORM INSERT

```
1 from sqlalchemy import create_engine
2 from sqlalchemy.orm import Session
```

```

3 from orm_create_db import Base, Genre, Movie, Person,
Cast
4
5 dbname = 'imdb.db'
6 engine = create_engine('sqlite:///+' + dbname)
7
8 Base.metadata.bind = engine
9
10 session = Session(engine)
11 genre = {}
12 for name in ('Action', 'Animation', 'Comedy',
'Documentary', 'Family', 'Horror'):
13     genre[name] = Genre(name = name)
14     session.add(genre[name])
15
16 print(genre['Animation'].name) # Animation
17 print(genre['Animation'].id) # None
18 session.commit()
19
20 print(genre['Animation'].name) # Animation
21 print(genre['Animation'].id) # 2
22 session.add(Movie(title = "Sing", genre =
genre['Animation']))
23 session.commit()

```

SQLAlchemy ORM SELECT

```

1 from sqlalchemy import create_engine
2 from sqlalchemy.orm import Session
3 from orm_create_db import Base, Genre, Movie, Person,
Cast
4
5 dbname = 'imdb.db'
6 engine = create_engine('sqlite:///+' + dbname)
7
8 Base.metadata.bind = engine
9
10 session = Session(engine)
11
12 for g in session.query(Genre).all():
13     print(g.name, g.id)
14
15 print("---")

```

```
16 animation = session.query(Genre).filter(Genre.name ==  
'Animation').one()  
17 print(animation.name, animation.id)
```

SQLAlchemy ORM SELECT cross tables

```
1 from sqlalchemy import create_engine  
2 from sqlalchemy.orm import Session  
3 from orm_create_db import Base, Genre, Movie, Person,  
Cast  
4  
5 dbname = 'imdb.db'  
6 engine = create_engine('sqlite:///+' + dbname)  
7  
8 Base.metadata.bind = engine  
9  
10 session = Session(engine)  
11  
12 movies = session.query(Movie).all()  
13 for m in movies:  
14     print(m.title, "--", m.genre.name)
```

SQLAlchemy ORM SELECT and INSERT

```
1 from sqlalchemy import create_engine  
2 from sqlalchemy.orm import Session  
3 from orm_create_db import Base, Genre, Movie, Person,  
Cast  
4  
5 dbname = 'imdb.db'  
6 engine = create_engine('sqlite:///+' + dbname)  
7  
8 Base.metadata.bind = engine  
9  
10 session = Session(engine)  
11  
12 animation = session.query(Genre).filter(Genre.name ==  
'Animation').one()  
13 session.add(Movie(title = "Moana", genre = animation))  
14 session.add(Movie(title = "Trolls", genre =  
animation))
```

```
15
16 action = session.query(Genre).filter(Genre.name ==
17     'Action').one()
17 session.add(Movie(title = "Power Rangers", genre =
18     action))
18
19 comedy = session.query(Genre).filter(Genre.name ==
20     'Comedy').one()
20 session.add(Movie(title = "Gostbuster", genre =
21     comedy))
21
22
23 session.commit()
```

SQLAlchemy ORM UPDATE

```
1 from sqlalchemy import create_engine
2 from sqlalchemy.orm import Session
3 from orm_create_db import Base, Genre, Movie, Person,
Cast
4
5 dbname = 'imdb.db'
6 engine = create_engine('sqlite:/// ' + dbname)
7
8 Base.metadata.bind = engine
9
10 session = Session(engine)
11
12 movie = session.query(Movie).filter(Movie.title ==
13     'Gostbuster').one()
13 print(movie.title)
14 movie.title = 'Ghostbusters'
15 session.commit()
16
17 print(movie.title)
```

SQLAlchemy ORM logging

```
1 from sqlalchemy import create_engine
2 from sqlalchemy.orm import Session
3 from orm_create_db import Base, Genre, Movie, Person,
```

```

Cast
4
5 import logging
6
7 logging.basicConfig()
8
logging.getLogger('sqlalchemy.engine').setLevel(logging.I
NFO)
9
10 logger = logging.getLogger('demo')
11 logger.setLevel(logging.INFO)
12
13 dbname = 'imdb.db'
14 engine = create_engine('sqlite:/// ' + dbname)
15
16 Base.metadata.bind = engine
17
18 session = Session(engine)
19
20
21 logger.info("Selecting all")
22 movies = session.query(Movie).all()
23 for m in movies:
24     logger.info("-----")
25     #print(m.title, "-", m.genre_id)
26     print(m.title, "-", m.genre.name)

```

Solution: Create table

Create the followig schema

```

1 from sqlalchemy import create_engine
2 from sqlalchemy import MetaData
3 from sqlalchemy import Table, Column
4 from sqlalchemy import Integer, String
5 from sqlalchemy import ForeignKey
6
7 metadata = MetaData()
8
9 node_table = Table('node', metadata,
10                     Column('id', Integer,
primary_key=True),
11                     Column('name', String(100),

```

```
unique=True)
12
13
14 interface_table = Table('interface', metadata,
15                         Column('id', Integer,
16                         primary_key=True),
17                         Column('node_id', Integer,
18                         ForeignKey('node.id'), nullable=False),
19                         Column('ipv4', String(14),
20                         unique=True),
21                         Column('ipv6', String(80),
22                         unique=True),
23
24                         )
25
26 engine = create_engine('sqlite://', echo=True)
27 metadata.create_all(engine)
```

Exercise: Inspector

Use the inspector to list all the tables and all the columns in every table.

```
1 from sqlalchemy import create_engine
2 from sqlalchemy import MetaData
3 from sqlalchemy import Table, Column
4 from sqlalchemy import Integer, String
5 from sqlalchemy import ForeignKey
6
7 metadata = MetaData()
8
9 node_table = Table('node', metadata,
10                     Column('id', Integer,
11                     primary_key=True),
12                     Column('name', String(100),
13                     unique=True)
```

```

12
)
13
14 interface_table = Table('interface', metadata,
15                         Column('id', Integer,
primary_key=True),
16                         Column('node_id', Integer,
ForeignKey('node.id'), nullable=False),
17                         Column('ipv4', String(14),
unique=True),
18                         Column('ipv6', String(80),
unique=True),
19                         )
20
21 connection_table = Table('connection', metadata,
22                           Column('a', Integer,
ForeignKey('interface.id'), nullable=False),
23                           Column('b', Integer,
ForeignKey('interface.id'), nullable=False)
24                           )
25
26 engine = create_engine('sqlite://', echo=True)
27 metadata.create_all(engine)
28
29
30 m2 = MetaData()
31 m2_node_table = Table('node', m2, autoload=True,
autoload_with=engine)
32 m2_interface_table = Table('interface', m2,
autoload=True, autoload_with=engine)
33 print(m2_node_table.columns)
34 print(m2_interface_table.columns)
35 print(m2_node_table.__repr__())
36
37 from sqlalchemy import inspect
38
39 inspector = inspect(engine)
40 inspector.get_columns('address')
41 inspector.get_foreign_keys('address')

```

SQLAlchemy CREATE and DROP

- `metadata.create_all(engine, checkfirst=True|False)` emits CREATE statement for all tables.
- `table.create(engine, checkfirst=False|True)` emits CREATE statement for a single table.
- `metadata.drop_all(engine, checkfirst=True|False)` emits DROPT statement for all the tables.
- `table.drop(engine, checkfirst=False|True)` emits DROPT statement for a single table.

`metada` can create (or drop) the tables in the correct order to maintain the dependencies.

SQLAlchemy Notes

- Multi-column primary key (composite primary key).
- Composite foreign key.

SQLAlchemy Meta SQLite CREATE

```

1 from sqlalchemy import create_engine
2 import os
3 from sqlite_meta_schema import get_meta
4
5 dbname = 'test.db'
6 if os.path.exists(dbname):
7     os.unlink(dbname)
8 engine = create_engine('sqlite:///test.db')
9
10 metadata = get_meta()
11 metadata.create_all(engine)

```

```

1 from sqlalchemy import MetaData
2 from sqlalchemy import Table, Column
3 from sqlalchemy import Integer, String
4 from sqlalchemy import ForeignKey
5

```

```

6
7 def get_meta():
8     metadata = MetaData()
9
10    node_table = Table('node', metadata,
11                         Column('id', Integer,
primary_key=True),
12                         Column('name', String(100),
unique=True)
13                         )
14
15    interface_table = Table('interface', metadata,
16                         Column('id', Integer,
primary_key=True),
17                         Column('node_id', Integer,
ForeignKey('node.id'), nullable\
18 le=False),
19                         Column('ipv4', String(14),
unique=True),
20                         Column('ipv6', String(80),
unique=True),
21                         )
22
23    connection_table = Table('connection', metadata,
24                         Column('a', Integer,
ForeignKey('interface.id'), nullable\
25 le=False),
26                         Column('b', Integer,
ForeignKey('interface.id'), nullable\
27 le=False)
28                         )
29
30    return metadata

```

SQLAlchemy Meta Reflection

```

1 from sqlalchemy import create_engine
2 import os
3 #from sqlalchemy import inspect
4 from sqlalchemy.engine import reflection
5
6 dbname = 'test.db'
7 if not os.path.exists(dbname):
8     exit("Database file '{}' could not be"

```

```
found".format(dbname))
9
10 engine = create_engine('sqlite:///test.db')
11 # inspector = inspect(engine)
12 # print(inspector)
13 # print(inspector.get_columns('address'))
14 # print(inspector.get_foreign_keys('address'))
15
16 insp = reflection Inspector.from_engine(engine)
17 print(insp.get_table_names())
```

SQLAlchemy Meta INSERT

1

SQLAlchemy Meta SELECT

1

NoSQL

Types of NoSQL databases

- Document oriented - MongoDB
- Key-Value store - Redis
- Graph - Neo4j
- Tuple store - Apache River, TIBCO

MongoDB

MongoDB CRUD

- Create, Read, Update, Delete

Install MongoDB support

- Otherwise: `pip install pymongo`

Python MongoDB insert

```
1 from pymongo import MongoClient
2 import datetime
3
4 client = MongoClient()
5 db = client.demo
6
7 foo = {
8     'name' : 'Foo',
9     'email' : 'foo@example.com',
10    'birthdate' : datetime.datetime.strptime('2002-01-
02', '%Y-%m-%d'),
11    'student' : True,
12 }
13
14 bar = {
15     'name' : 'Bar',
16     'email' : 'bar@example.com',
17     'birthdate' : datetime.datetime.strptime('1998-08-
03', '%Y-%m-%d'),
18     'student' : True,
19     'teacher' : False,
20 }
21
```

```

22 zorg = {
23     'name'       : 'Zorg',
24     'email'       : 'zorg@corp.com',
25     'birthdate'   : datetime.datetime.strptime('1995-12-
26     12', '%Y-%m-%d'),
27     'teacher'     : True,
28 }
29
30 db.people.insert(foo)
31 db.people.insert(bar)
32 db.people.insert(zorg)

```

MongoDB CLI

```

1 $ mongo
2 > help
3 ...
4 > show dbs
5 admin    (empty)
6 demo     0.078GB
7 local    0.078GB
8
9 > use demo      (name of db)
10 switched to db demo
11
12 > show collections
13 people
14 system.indexes
15
16 > db.people.find()
17 { "_id" : ObjectId("58a3e9b2962d747a9c6e676c"),
18 "email" : "foo@example.com", "studen\
18 t" : true,
19     "birthdate" : ISODate("2002-01-02T00:00:00Z"),
20     "name" : "Foo" }
20 { "_id" : ObjectId("58a3e9b2962d747a9c6e676d"),
21 "email" : "bar@example.com", "name" \
21 : "Bar", "student" : true,
22     "birthdate" : ISODate("1998-08-03T00:00:00Z"),
23     "teacher" : false }
23 { "_id" : ObjectId("58a3e9b2962d747a9c6e676e"),
24 "email" : "zorg@corp.com",
25     "name" : "Zorg" }
25
26 > exit

```

```
24     "birthdate" : ISODate("1995-12-12T00:00:00Z"),
25     "teacher" : true, "name" : "Zorg"\n26
27 > db.people.drop()          (drop a collection)
28 > db.dropDatabase()        (drop a whole database)
```

Python MongoDB find

```
1 from pymongo import MongoClient
2 import datetime
3
4 client = MongoClient()
5 db = client.demo
6
7 for p in db.people.find():
8     print(p)
```

Python MongoDB find refine

```
1 from pymongo import MongoClient
2 import datetime
3
4 client = MongoClient()
5 db = client.demo
6
7 for p in db.people.find({ 'name' : 'Foo' }):
8     print(p)
```

Python MongoDB update

```
1 from pymongo import MongoClient
2 import datetime
3
4 client = MongoClient()
5 db = client.demo
6
7 db.people.update({ 'name' : 'Zorg' }, { '$set' : {
8     'salary' : 1000 } })
```

```
8 for p in db.people.find({ 'name' : 'Zorg' }):
9     print(p)
```

Python MongoDB remove (delete)

```
1 from pymongo import MongoClient
2 import datetime
3
4 client = MongoClient()
5 db = client.demo
6
7 db.people.remove({ 'name' : 'Zorg' })
8 for p in db.people.find():
9     print(p)
```

Redis

Redis CLI

redis-cli

```
1 $ redis-cli
2 > set name foo
3 > get name
4 > set name "foo bar"
5 > get name
6
7 > set a 1
8 > get a
9 > incr a
10 > get a
11
12 > set b 1
13 > keys *
14 > del b
```

Redis list keys

```
1 import redis
2 r = redis.StrictRedis()
3
4 for k in r.keys('*'):
5     print(k)
```

Redis set get

```
1 import redis
2 r = redis.StrictRedis()
3
```

```
4 r.set("name", "some value")
5 print(r.get("name"))
```

Redis incr

```
1 import redis
2 r = redis.StrictRedis()
3
4 r.set("counter", 40)
5 print(r.get("counter"))
6 print(r.incr("counter"))
7 print(r.incr("counter"))
8 print(r.get("counter"))
```

Redis incrby

```
1 import redis
2 r = redis.StrictRedis()
3
4 r.set("counter", 19)
5 print(r.get("counter"))
6 print(r.incrby("counter", 23))
7 print(r.get("counter"))
```

Redis setex

Set with expiration time in seconds.

```
1 import redis
2 import time
3 r = redis.StrictRedis()
4
5
6 r.setex("login", 2, 'foobar')
7 print(r.get("login"))    # 'foobar'
8 time.sleep(1)
9 print(r.get("login"))    # 'foobar'
10 time.sleep(1)
11 print(r.get("login"))   # None
```


Web client

urllib the web client

```
1 import urllib
2
3 # f is like a filehand for http requests, but it
4 # cannot be user "with"
5 # Only works in Python 2
6 f = urllib.urlopen('http://python.org/')
7 html = f.read()    # is like a get() request
8 f.close()
9 print(html)
10
11
12 # retrieve a file and save it locally:
13
urllib.urlretrieve('http://www.python.org/images/python-
logo.gif', 'logo.gif')
```

urllib2 the web client

urllib2 is better than urllib as it will indicate if there was an error retrieving

```
1 import urllib2
2
3 # f is like a filehand for http requests
4 f = urllib2.urlopen('http://python.org/')
5 html = f.read()    # is like a get() request
6 f.close()
7 print(html)
```

```
8
9
10 try:
11     f =
urllib2.urlopen('http://python.org/some_missing_page')
12     html = f.read()
13     f.close()
14     print(html)
15 except urllib2.HTTPError as e:
16     print(e)    # HTTP Error 404: OK
```

httpbin.org

- httpbin.org
- [source](#)

requests get

```
1 import requests
2
3 r = requests.get('http://httpbin.org/')
4 print(type(r))
5 print(r.status_code)
6 print(r.headers)
7 print(r.headers['content-type'])
```

- [HTTP status codes](#)
- [Python requests](#)

Download image using requests

```
1 import requests
2
3 url =
'https://bloximages.newyork1.vip.townnews.com/wpsdlocal6.
com/content/tncms/ass\
4 ets/v3/editorial/7/22/722f8401-e134-5758-9f4b-
```

```
a542ed88a101/5d41b45d92106.image.jpg'
5 filename = "source.jpg"
6 res = requests.get(url)
7 print(res.status_code)
8 with open(filename, 'wb') as fh:
9     fh.write(res.content)
```

Download image as a stream using requests

```
1 import requests
2 import shutil
3
4 url =
'https://bloximages.newyork1.vip.townnews.com/wpsdlocal6.
com/content/tncms/ass\
5 ets/v3/editorial/7/22/722f8401-e134-5758-9f4b-
a542ed88a101/5d41b45d92106.image.jpg'
6 filename = "source.jpg"
7 res = requests.get(url, stream=True)
8 print(res.status_code)
9 with open(filename, 'wb') as fh:
10    res.raw.decode_content
11    shutil.copyfileobj(res.raw, fh)
```

Download zip file

```
1 import requests
2 import shutil
3
4 url = "https://code-
maven.com/public/developer_survey_2019.zip"
5 filename = "developer_survey_2019.zip"
6
7 res = requests.get(url, stream=True)
8 print(res.status_code)
9 if res.status_code == 200:
10    with open(filename, 'wb') as fh:
11        res.raw.decode_content
12        shutil.copyfileobj(res.raw, fh)
```

Extract zip file

```
1 import zipfile  
2  
3 path = "developer_survey_2019.zip"  
4 z = zipfile.ZipFile(path)  
5 z.extractall()
```

Interactive Requests

```
1 import requests  
2  
3 r = requests.get('http://httpbin.org/')
```

```
4  
5 import code  
6 code.interact(local=locals())
```

requests get JSON

```
1 import requests  
2  
3 r = requests.get('http://httpbin.org/ip')  
4 print(r.headers['content-type'])  
5 print(r.text)  
6 data = r.json()  
7 print(data)  
8 print(data['origin'])
```

requests get JSON UserAgent

```
1 import requests  
2  
3 r = requests.get('http://httpbin.org/user-agent')  
4 print(r.headers['content-type'])  
5 print(r.text)  
6 data = r.json()  
7 print(data)  
8 print(data['user-agent'])
```

requests get JSON UserAgent

```
1 import requests
2
3 r = requests.get('http://httpbin.org/user-agent',
4     headers = { 'User-agent': 'Internet Explorer/2.0' })
5 print(r.headers['content-type'])
6 print(r.text)
7 data = r.json()
8 print(data)
9 print(data['user-agent'])
```

requests get header

```
1 import requests
2
3 r = requests.get('http://httpbin.org/headers')
4 print(r.text)
5
6 # {
7 #     "headers": {
8 #         "Accept": "*/*",
9 #         "Accept-Encoding": "gzip, deflate",
10 #         "Host": "httpbin.org",
11 #         "User-Agent": "python-requests/2.3.0
CPython/2.7.12 Darwin/16.3.0"
12 #     }
13 # }
```

requests change header

```
1 import requests
2
3 r = requests.get('http://httpbin.org/headers',
4     headers = {
5         'User-agent' : 'Internet Explorer/2.0',
6         'SOAPAction' : ''
7         'http://www.corp.net/some/path/CustMsagDown.Check',
8         'Content-type': 'text/xml'
9     }
10 }
```

```
9     )
10    print(r.text)
11
12 # {
13 #     "headers": {
14 #         "Accept": "*/*",
15 #         "Accept-Encoding": "gzip, deflate",
16 #         "Content-Type": "text/xml",
17 #         "Host": "httpbin.org",
18 #         "Soapaction":
19 #             "http://www.corp.net/some/path/CustMsagDown.Check",
20 #             "User-Agent": "Internet Explorer/2.0"
21 #     }
22 }
```

requests post

```
1 import requests
2
3 payload = '''
4 <soapenv:Envelope
5 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
6 xmlns:cu\
7 s="http://www.corp.net/Request.XSD">
8     <soapenv:Header/>
9     <soapenv:Body>
10         <cus:CustMsagDown.Check>
11             <cus:MainCustNum>327</cus:MainCustNum>
12             <cus:SourceSystem></cus:SourceSystem>
13         </cus:CustMsagDown.Check>
14     </soapenv:Body>
15 </soapenv:Envelope>
16 '''
17
18 r = requests.post('http://httpbin.org/post',
19     headers = {
20         'User-agent' : 'Internet Explorer/2.0',
21         'SOAPAction' :
22             'http://www.corp.net/some/path/CustMsagDown.Check',
23         'Content-type': 'text/xml'
24     },
25     data = payload,
26 )
27 )
```

```
24 print(r.headers['content-type'])  
25 print(r.text)
```

Tweet

```
1 import configparser  
2 import twitter  
3 import os  
4  
5 config = configparser.ConfigParser()  
6  
config.read(os.path.join(os.path.dirname(os.path.abspath(  
__file__)), 'api.cfg'));  
7 api = twitter.Api(**config['twitter'])  
8  
9 status = api.PostUpdate('My first Tweet using Python')  
10 print(status.text)
```

API config file

```
1 [twitter]  
2 consumer_key=  
3 consumer_secret=  
4 access_token_key=  
5 access_token_secret=  
6  
7 [bitly]  
8 access_token=
```

bit.ly

```
1 import configparser  
2 import os  
3 import requests  
4  
5 def shorten(uri):  
6     config = configparser.ConfigParser()  
7     #config.read(os.path.join(os.path.expanduser('~'),  
'api.cfg'))
```

```

8
config.read(os.path.join(os.path.dirname(os.path.abspath(
__file__)), 'api.cfg'))
9
10    query_params = {
11        'access_token': bitly_config['bitly']
12    }
13
14    endpoint = 'https://api-ssl.bitly.com/v3/shorten'
15    response = requests.get(endpoint,
16    params=query_params, verify=False)
17
18    data = response.json()
19
20    if not data['status_code'] == 200:
21        exit("Unexpected status_code: {} in bitly
response. {}".format(data['status_\
22 code'], response.text))
23    return data['data']['url']
24
25 print(shorten("http://code-maven.com/"))

```

Exercise: Combine web server and client

Write a web application that can get a site and a text as input (e.g.
<http://cnn.com> and ‘Korea’)
check if on the given site the word appears or not?

Extended version: Only get the URL as the input and create
statistics, which are the most
frequent words on the given page.

Python Web server

Hello world web

```
1 from wsgiref.util import setup_testing_defaults
2 from wsgiref.simple_server import make_server
3
4 import time
5
6 def hello_world(environ, start_response):
7     setup_testing_defaults(environ)
8
9     status = '200 OK'
10    headers = [('Content-type', 'text/plain')]
11
12    start_response(status, headers)
13
14    return "Hello World " + str(time.time())
15
16 port = 8080
17 httpd = make_server('0.0.0.0', port, hello_world)
18 print("Serving on port {}...".format(port))
19 httpd.serve_forever()
```

Dump web environment info

```
1 from wsgiref.util import setup_testing_defaults
2 from wsgiref.simple_server import make_server
3
4 # A relatively simple WSGI application. It's going to
print out the
5 # environment dictionary after being updated by
setup_testing_defaults
6 def simple_app(environ, start_response):
7     setup_testing_defaults(environ)
8
9     status = '200 OK'
```

```

10     headers = [ ('Content-type', 'text/plain') ]
11
12     start_response(status, headers)
13
14     ret = ["{}: {}\n".format(key, value)
15            for key, value in environ.items()]
16
17     return ret
18
19 httpd = make_server('', 8000, simple_app)
20 print("Serving on port 8000...")
21 httpd.serve_forever()
22 # taken from the standard documentation of Python

```

Web echo

```

1 from wsgiref.util import setup_testing_defaults
2 from wsgiref.simple_server import make_server
3
4 import time
5 import cgi
6
7 def hello_world(environ, start_response):
8     setup_testing_defaults(environ)
9
10    status = '200 OK'
11    headers = [ ('Content-type', 'text/html')]
12
13    start_response(status, headers)
14
15    form = cgi.FieldStorage(fp=environ['wsgi.input'],
16                           environ=environ)
17    if 'txt' in form:
18        return 'Echo: ' + form['txt'].value
19
20    return """
21 <form>
22 <input name="txt" />
23 <input type="submit" value="Echo" />
24 """
25 httpd = make_server('', 8000, hello_world)

```

```
26 print("Serving on port 8000...")
27 httpd.serve_forever()
```

Web form

```
1  from wsgiref.util import setup_testing_defaults
2  from wsgiref.simple_server import make_server
3
4  import time
5  import cgi
6
7  def hello_world(environ, start_response):
8      setup_testing_defaults(environ)
9
10     status = '200 OK'
11     headers = [('Content-type', 'text/html')]
12
13     start_response(status, headers)
14
15     form = cgi.FieldStorage(fp=environ['wsgi.input'],
16                           environ=environ)
16     html = ''
17     for f in form:
18         html += f + '==' + form[f].value + '<br>'
19
20     if not html:
21         html = """
22 <a href="?fname=Foo&lname=Bar">click</a>
23 <form>
24 Username: <input name="username" /><br>
25 Password: <input type="password" name="pw" /><br>
26 Age group: Under 18 <input type="radio" name="age"
27 value="kid" >
28 18-30 <input type="radio" name="age" value="young" >
29 30- <input type="radio" name="age" value="old" >
30 <input type="submit" value="Send" />
31 </form>
31 """
32     return html
33
34 httpd = make_server(' ', 8000, hello_world)
35 print("Serving on port 8000...")
36 httpd.serve_forever()
```

Resources

- [wsgi tutorial](#)

Python Flask

Python Flask intro

- [Flask](#)
- [Jinja](#)
- [Werkzeug](#)

Python Flask installation

```
1 virtualenv venv -p python3
2 source venv/bin/activate
3
4 pip install flask
```

Flask: Hello World

```
1 from flask import Flask
2 app = Flask(__name__)
3
4 @app.route("/")
5 def hello():
6     return "Hello World!"
7
8 if __name__ == "__main__":
9     app.run()
```

```
1 $ python hello_world.py
2 Running on http://127.0.0.1:5000/ (Press CTRL+C to
quit)
```

Flask hello world + test

```
1 from flask import Flask
2
3 app = Flask(__name__)
4
5 @app.route("/")
6 def hello():
7     return "Hello World!"
```

```
1 FLASK_APP=app FLASK_DEBUG=1 flask run
2
3 Visit: http://127.0.0.1:5000/
4 curl http://localhost:5000/
```

Windows on the command line or in the terminal of Pycharm.

```
1 set FLASK_APP=app
2 set FLASK_DEBUG=1
3 flask run
```

```
1 import app
2
3 def test_app():
4     web = app.app.test_client()
5
6     rv = web.get('/')
7     assert rv.status == '200 OK'
8     assert rv.data == b'Hello World!'
```

```
1 pytest
```

Flask generated page - time

```
1 from flask import Flask
2 import time
3
4 app = Flask(__name__)
5
6 @app.route("/")
```

```
7 def main():
8     return '<a href="/time">time</a>'
9
10 @app.route("/time")
11 def echo():
12     return str(time.time())
```

```
1 import app
2 import re
3
4 def test_home():
5     web = app.app.test_client()
6
7     rv = web.get('/')
8     assert rv.status == '200 OK'
9     assert rv.data == b'<a href="/time">time</a>'
10
11 def test_time():
12     web = app.app.test_client()
13
14     rv = web.get('/time')
15     assert rv.status == '200 OK'
16     assert re.search(r'\d+\.\d+$',
rv.data.decode('utf-8'))
```

Flask: Echo GET

```
1 from flask import Flask, request
2 app = Flask(__name__)
3
4 @app.route("/")
5 def main():
6     return '''
7         <form action="/echo" method="GET">
8             <input name="text">
9             <input type="submit" value="Echo">
10        </form>
11        '''
12
13 @app.route("/echo")
```

```
14 def echo():
15     return "You said: " + request.args.get('text', '')
```

```
1 import app
2
3 def test_app():
4     web = app.app.test_client()
5
6     rv = web.get('/')
7     assert rv.status == '200 OK'
8     assert '<form action="/echo" method="GET">' in
rv.data.decode('utf-8')
9
10    rv = web.get('/echo')
11    assert rv.status == '200 OK'
12    assert b"You said: " == rv.data
13
14    rv = web.get('/echo?text=foo+bar')
15    assert rv.status == '200 OK'
16    assert b"You said: foo bar" == rv.data
```

```
1 curl http://localhost:5000/
2 curl http://localhost:5000/echo?text=Sanch+Panza
```

```
1 import requests
2
3 res = requests.get('http://localhost:5000/')
4 print(res.status_code)
5 print(res.text)
6
7 res = requests.get('http://localhost:5000/echo?
text=Hello World!')
8 print(res.status_code)
9 print(res.text)
```

Flask: Echo POST

```
1 from flask import Flask, request
2
```

```
3 app = Flask(__name__)
4
5 @app.route("/")
6 def main():
7     return """
8         <form action="/echo" method="POST">
9             <input name="text">
10            <input type="submit" value="Echo">
11        </form>
12    """
13
14 @app.route("/echo", methods=['POST'])
15 def echo():
16     if 'text' in request.form:
17         return "You said: " + request.form['text']
18     else:
19         return "Nothing to say?"
```

```
1 import app
2
3 def test_app():
4     web = app.app.test_client()
5
6     rv = web.get('/')
7     assert rv.status == '200 OK'
8     assert '<form action="/echo" method="POST">' in
rv.data.decode('utf-8')
9
10    rv = web.get('/echo')
11    assert rv.status == '405 METHOD NOT ALLOWED'
12    assert '<title>405 Method Not Allowed</title>' in
rv.data.decode('utf-8')
13
14    rv = web.post('/echo')
15    assert rv.status == '200 OK'
16    assert b"Nothing to say?" == rv.data
17
18    rv = web.post('/echo', data={ "text": "foo bar" })
19    assert rv.status == '200 OK'
20    assert b"You said: foo bar" == rv.data
```

```
1 curl --data "text=Sancho Panza"
http://localhost:5000/echo
```

```
1 import requests
2
3 res = requests.get('http://localhost:5000/')
4 print(res.status_code)
5 print(res.text)
6
7
8 res = requests.post('http://localhost:5000/echo',
9 data={"text": "Hello World!"})
9 print(res.status_code)
10 print(res.text)
```

Flask: templates

```
1 from flask import Flask, request
2 app = Flask(__name__)
3
4 @app.route("/")
5 def main():
6     return render_template('index.html')
7
8 @app.route("/echo", methods=['POST'])
9 def echo():
10     return "You said: " + request.form['text']
```

```
1 <form action="/echo" method="POST">
2 <input name="text">
3 <input type="submit" value="Echo">
4 </form>
```

```
1 FLASK_APP=echo_post FLASK_DEBUG=0 flask run
```

Internal Server Error

```
1 FLASK_APP=echo_post FLASK_DEBUG=1 flask run
```

Flask: templates

```
1 from flask import Flask, request, render_template
2 app = Flask(__name__)
3
4 @app.route("/")
5 def main():
6     return render_template('index.html')
7
8 @app.route("/echo", methods=['POST'])
9 def echo():
10    return "You said: " + request.form['text']
```

Flask: templates with parameters

```
1 from flask import Flask, request, render_template
2 app = Flask(__name__)
3
4 @app.route("/")
5 def main():
6     return render_template('echo.html')
7
8 @app.route("/echo", methods=['POST'])
9 def echo():
10    return render_template('echo.html',
11                           text=request.form['text'])
```

```
1 <form action="/echo" method="POST">
2   <input name="text">
3   <input type="submit" value="Echo">
4 </form>
5
6 {%- if text %}
7   You said: {{ text }}
8 {%- endif %}
```

```
1 import echo
2
3 def test_app():
4     web = echo.app.test_client()
5
6     rv = web.get('/')
7     assert rv.status == '200 OK'
8     assert '<form action="/echo" method="POST">' in
rv.data.decode('utf-8')
9
10    rv = web.post('/echo', data={ "text": "foo bar" })
11    assert rv.status == '200 OK'
12    assert "You said: foo bar" in rv.data.decode('utf-
8')
```

Flask: runner

```
1 $ cd examples/flask/params
```

```
1 $ export FLASK_APP=echo
2 $ export FLASK_DEBUG=1
3 $ flask run
```

or

```
1 $ FLASK_APP=echo.py FLASK_DEBUG=1 flask run
```

on windows

```
1 > set FLASK_APP=echo
2 > set FLASK_DEBUG=1
3 > flask run
```

Other parameters

```
1 $ FLASK_APP=echo.py FLASK_DEBUG=1 flask run --port
8080 --host 0.0.0.0
```

Exercise: Flask calculator

Write a web application that has two entry boxes and a button and that will add the two numbers inserted into the entry boxes.

Static files

```
1 from flask import Flask, request, render_template,
2 url_for
3 app = Flask(__name__)
4
5 @app.route("/")
6 def main():
7     return render_template('main.html')
8
9 @app.route("/other")
10 def other():
11     return render_template('other.html',
12                           img_path = url_for('static',
13                           filename='img/python.png'))
```

```
1 <h1>Main page</h1>
2 
3 <p>
4 <a href="/other">other</a>
```

```
1 <h2>Other page</h2>
2 img_path: {{ img_path }}
3 <p>
4 
5 <p>
6 <a href="/">main</a>
```

```
1 .
2 └── app.py
```

```
3 └── static
4     └── img
5         └── python.png
6 └── templates
7     ├── main.html
8     └── other.html
```

Flask Logging

```
1 from flask import Flask
2 app = Flask(__name__)
3
4 @app.route("/")
5 def main():
6     app.logger.debug("Some debug message")
7     app.logger.warning("Some warning message")
8     app.logger.error("Some error message")
9     return "Hello World"
```

Flask: Counter

```
1 from flask import Flask
2 app = Flask(__name__)
3
4 counter = 1
5
6 @app.route("/")
7 def main():
8     global counter
9     counter += 1
10    return str(counter)
```

Access the page from several browsers. There is one single counter that lives as long as the process lives.

Color selector without session

```
1 from flask import Flask, request, render_template
2 import re
3 app = Flask(__name__)
4
5 @app.route("/", methods=['GET', 'POST'])
6 def main():
7     color = "FFFFFF"
8     new_color = request.form.get('color', '')
9     if re.search(r'^[0-9A-F]{6}$', new_color):
10         color = new_color
11
12     return render_template('main.html', color = color)
```

```
1 <style>
2 * {
3     background-color: #{{ color }};
4 }
5 </style>
6
7 <form method="POST">
8 <input name="color" value="{{ color }}">
9 <input type="submit" value="Set">
10 </form>
11 <p>
12 <a href="/">home</a>
```

Session management

```
1 from flask import Flask, request, render_template,
session
2 import re
3 app = Flask(__name__)
4 app.secret_key = 'blabla'
5
6 @app.route("/", methods=['GET', 'POST'])
7 def main():
8     color = session.get('color', 'FFFFFF')
9     app.logger.debug("Color: " + color)
10
11     new_color = request.form.get('color', '')
12     if re.search(r'^[0-9A-F]{6}$', new_color):
```

```
13         app.logger.debug('New color: ' + new_color)
14         session['color'] = new_color
15         color = new_color
16
17     return render_template('main.html', color = color)
```

Flask custom 404 page

```
1 from flask import Flask
2 app = Flask(__name__)
3
4 @app.route("/")
5 def main():
6     return '''
7 Main
8 <a href="/not">404 page</a>
9 '''
```

```
1 from flask import Flask
2 app = Flask(__name__)
3
4 @app.route("/")
5 def main():
6     return '''
7 Main
8 <a href="/not">404 page</a>
9 '''
10
11 @app.errorhandler(404)
12 def not_found(e):
13     return "Our Page not found", 404
```

```
1 curl -I http://localhost:5000/not
2
3 HTTP/1.0 404 NOT FOUND
```

Flask Error page

```
1 from flask import Flask
2 app = Flask(__name__)
3
4 @app.route("/")
5 def main():
6     return '''
7 Main
8 <a href="/bad">bad</a>
9 '''
10
11 @app.route("/bad")
12 def bad():
13     raise Exception("This is a bad page")
14     return 'Bad page'
```

Will not trigger in debug mode!

```
1 $ FLASK_APP=echo.py FLASK_DEBUG=0 flask run
```

```
1 curl -I http://localhost:5000/not
2
3 HTTP/1.0 500 INTERNAL SERVER ERROR
```

```
1 from flask import Flask
2 app = Flask(__name__)
3
4 @app.route("/")
5 def main():
6     return '''
7 Main
8 <a href="/bad">bad</a>
9 '''
10
11 @app.route("/bad")
12 def bad():
13     raise Exception("This is a bad page")
14     return 'Bad page'
15
16 @app.errorhandler(500)
17 def not_found(err):
```

```
18     #raise Exception("Oups")
19     return "Our Page crashed", 500
```

Flask URL routing

The mapping of the path part of a URL, so the one that comes after the domain name and after the port number (if it is included) is the path. Mapping that to a function call is called routing.

In the following pages we are going to see several examples on how to map routes to functions.

It is also called “url route registration”.

Flask Path params

```
1 from flask import Flask, jsonify
2 app = Flask(__name__)
3
4 @app.route("/")
5 def main():
6     return '''
7 Main<br>
8 <a href="/user/23">23</a><br>
9 <a href="/user/42">42</a><br>
10 <a href="/user/Joe">Joe</a><br>
11 '''
12
13 @app.route("/user/<uid>")
14 def api_info(uid):
15     return uid
```

```
1 FLASK_APP=app.py FLASK_DEBUG=0 flask run
```

Flask Path params (int)

```
1 from flask import Flask, jsonify
2 app = Flask(__name__)
3
4 @app.route("/")
5 def main():
6     return '''
7 Main<br>
8 <a href="/user/23">23</a><br>
9 <a href="/user/42">42</a><br>
10 <a href="/user/Joe">Joe</a><br>
11 '''
12
13 @app.route("/user/<int:uid>")
14 def api_info(uid):
15     return str(uid)
```

```
1 FLASK_APP=app.py FLASK_DEBUG=0 flask run
```

Flask Path params add (int)

```
1 from flask import Flask, jsonify
2 app = Flask(__name__)
3
4 @app.route("/")
5 def main():
6     return '''
7 Main <a href="/add/23/19">add</a>
8 '''
9
10 @app.route("/add/<int:a>/<int:b>")
11 def api_info(a, b):
12     return str(a+b)
```

```
1 FLASK_APP=app.py FLASK_DEBUG=0 flask run
```

Flask Path params add (path)

- Accept any path, including slashes:

```
1 from flask import Flask, jsonify
2 app = Flask(__name__)
3
4 @app.route("/")
5 def main():
6     return """
7 Main<br>
8 <a href="/user/name">/user/name</a><br>
9 <a href="/user/other/dir">/user/other/dir</a><br>
10 <a href="/user/hi.html">/usre/hi.html</a><br>
11 """
12
13 @app.route("/user/<path:fullpath>")
14 def api_info(fullpath):
15     return fullpath
```

```
1 FLASK_APP=app.py FLASK_DEBUG=0 flask run
```

Jinja loop, conditional, include

```
1 .
2   └── app.py
3   └── templates
4     ├── incl
5     │   ├── footer.html
6     │   ├── header.html
7     └── main.html
```

```
1 from flask import Flask, render_template
2 app = Flask(__name__)
3
4 @app.route("/")
5 def main():
6     languages = [
7         {
```

```
8         "name": "Python",
9         "year": 1991,
10        },
11        {
12            "name": "JavaScript",
13            "year": 1995,
14        },
15        {
16            "name": "C",
17        }
18    ]
19    return render_template('main.html',
20        title = "Code Maven Ninja example",
21        languages = languages,
22    )
```

```
1 { % include 'incl/header.html' %}
2
3
4 <h2>Languages</h2>
5 <ul>
6     { % for lang in languages %}
7         <li>{{ lang.name }}
8             { % if lang.year %}
9                 {{ lang.year }}
10            { % else %}
11                Timeless
12            { % endif %}
13        </li>
14    { % endfor %}
15 </ul>
16
17 { % include 'incl/footer.html' %}
```

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="utf-8">
5     <meta name="viewport" content="width=device-width,
initial-scale=1, user-scalable\
6 e=yes">
7         <title>{{ title }}</title>
```

```
8 </head>
9 <body>
10 <h1>{{ title }}</h1>
```

```
1 </body>
2 </html>
```

Exercise: Flask persistent

Create a Flask-based application with a persistent counter that even after restarting the application the counter will keep increasing.

Exercise: Flask persistent

Create a Flask-based application with a persistent counter that even after restarting the application the counter will keep increasing. For each user have its own counter as identified by the username they type in.

Flask Exercises

- [Shopping list](#)
- [TODO](#)

Flask login

```
1 from flask import Flask, render_template, url_for,
redirect, request, session
2 app = Flask(__name__)
3 app.secret_key = 'loginner'
4
5 users = {
6     'admin' : 'secret',
7     'foo'   : 'myfoo',
```

```

8 }
9
10 @app.route("/")
11 def main():
12     return render_template('main.html')
13
14 @app.route('/login')
15 def login_form():
16     return render_template('login.html')
17
18 @app.route('/login', methods=['POST'])
19 def login():
20     username = request.form.get('username')
21     password = request.form.get('password')
22     if username and password and username in users and
users[username] == password:
23         session['logged_in'] = True
24         return redirect(url_for('account'))
25
26     return render_template('login.html')
27
28 @app.route("/account")
29 def account():
30     if not session.get('logged_in'):
31         return redirect(url_for('login'))
32
33     return render_template('account.html')
34
35 @app.route('/logout')
36 def logout():
37     if not session.get('logged_in'):
38         return "Not logged in"
39     else:
40         session['logged_in'] = False
41     return render_template('logout.html')

```

```

1 {%
2 include 'header.html' %}
```

2 Account information.

```

1 <div>
2 <a href="/">home</a> | <a href="/login">login</a> | <a
href="/logout">logout</a> | <\
```

```
3 a href="/account">account</a>
4 </div>
```

```
1 {%- include 'header.html' %}
```

```
2 Home page
```

```
1 {%- include 'header.html' %}
```

```
2 <form method="POST">
```

```
3 <input name="username" placeholder="username">
```

```
4 <input name="password" placeholder="password"
type="password">
```

```
5 <input type="submit" value="Login">
```

```
6 </form>
```

```
1 {%- include 'header.html' %}
```

```
2 Bye bye
```

```
1 {%- include 'header.html' %}
```

```
2 Home
```

Flask JSON API

```
1 from flask import Flask, jsonify
2 app = Flask(__name__)
3
4 @app.route("/")
5 def main():
6     return '''
7 Main
8 <a href="/api/info">info</a>
9 '''
10
11 @app.route("/api/info")
12 def api_info():
13     info = {
14         "ip" : "127.0.0.1",
15         "hostname" : "everest",
16         "description" : "Main server",
17         "load" : [ 3.21, 7, 14 ]
```

```
18     }
19     return jsonify(info)
```

```
1 $ curl -I http://localhost:5000/api/info
2 HTTP/1.0 200 OK
3 Content-Type: application/json
```

Flask and AJAX

```
1 from flask import Flask, jsonify, render_template,
request
2 import time
3 app = Flask(__name__)
4
5 @app.route("/")
6 def main():
7     return render_template('main.html', reload =
time.time())
8
9 @app.route("/api/info")
10 def api_info():
11     info = {
12         "ip" : "127.0.0.1",
13         "hostname" : "everest",
14         "description" : "Main server",
15         "load" : [ 3.21, 7, 14 ]
16     }
17     return jsonify(info)
18
19 @app.route("/api/calc")
20 def add():
21     a = int(request.args.get('a', 0))
22     b = int(request.args.get('b', 0))
23     div = 'na'
24     if b != 0:
25         div = a/b
26     return jsonify({
27         "a" : a,
28         "b" : b,
29         "add" : a+b,
30         "multiply" : a*b,
31         "subtract" : a-b,
```

```
32         "divide"    :  div,
33     } )


---


1 (function() {
2     var ajax_get = function(url, callback) {
3         xmlhttp = new XMLHttpRequest();
4         xmlhttp.onreadystatechange = function() {
5             if (xmlhttp.readyState == 4 &&
xmlhttp.status == 200) {
6                 console.log('responseText:' +
xmlhttp.responseText);
7                 try {
8                     var data =
JSON.parse(xmlhttp.responseText);
9                 } catch(err) {
10                     console.log(err.message + " in " +
xmlhttp.responseText);
11                 return;
12             }
13             callback(data);
14         }
15     };
16
17     xmlhttp.open("GET", url, true);
18     xmlhttp.send();
19 };
20
21     ajax_get('/api/info', function(data) {
22         console.log('get info');
23         document.getElementById('info').innerHTML =
JSON.stringify(data, null, '    \'\\
24 ');
25
document.getElementById('description').innerHTML =
data['description'];
26 });
27
28     var calc = document.getElementById('calc');
29     calc.addEventListener('click', function() {
30         document.getElementById('info').style.display
= "none";
31
document.getElementById('description').style.display =
"none";

```

```
32         var url = '/api/calc?a=' +
33 document.getElementById('a').value + '&b=' + docu\
34 ment.getElementById('b').value;
35         //console.log(url);
36         ajax_get(url, function(data) {
37             document.getElementById('add').innerHTML =
38 data['a'] + ' + ' + data['b']\
39 + ' = ' + data['add'];
40
41 document.getElementById('subtract').innerHTML = data['a']
42 + ' - ' + data\
43 ['b'] + ' = ' + data['subtract'];
44
45 document.getElementById('multiply').innerHTML = data['a']
46 + ' * ' + data\
47 ['b'] + ' = ' + data['multiply'];
48
49 document.getElementById('divide').innerHTML = data['a'] +
50 ' / ' + data['\
51 b'] + ' = ' + data['divide'];
52
53     });
54
55 })();
56 })()
```

```
1 <html>
2 <head>
3 </head>
4 <body>
5 <input type="number" id="a">
6 <input type="number" id="b">
7 <button id="calc">Calc</button>
8 <div id="results">
9     <div id="add"></div>
10    <div id="subtract"></div>
11    <div id="multiply"></div>
12    <div id="divide"></div>
13 </div>
14
15 <pre id="info"></pre>
16 <div id="description"></div>
17
18 <script src="/static/math.js?r={ {reload} }"></script>
```

```
19 </body>
20 </html>
```

Flask and AJAX

```
 1 from flask import Flask, jsonify, render_template,
request
 2 import time
 3 app = Flask(__name__)
 4
 5 @app.route("/")
 6 def main():
 7     return render_template('main.html', reload =
time.time())
 8
 9 @app.route("/api/info")
10 def api_info():
11     info = {
12         "ip" : "127.0.0.1",
13         "hostname" : "everest",
14         "description" : "Main server",
15         "load" : [ 3.21, 7, 14 ]
16     }
17     return jsonify(info)
18
19 @app.route("/api/calc")
20 def add():
21     a = int(request.args.get('a', 0))
22     b = int(request.args.get('b', 0))
23     div = 'na'
24     if b != 0:
25         div = a/b
26     return jsonify({
27         "a" : a,
28         "b" : b,
29         "add" : a+b,
30         "multiply" : a*b,
31         "subtract" : a-b,
32         "divide" : div,
33     })
```

```
1 $(function() {
2     $.ajax({
3         url: '/api/info',
4         success: function(data) {
5             console.log('get info');
6             $('#info').html(JSON.stringify(data, null,
7                 ' '));
8         }
9     });
10
11     $('#calc').click(function() {
12         $('#info').css('display', "none");
13         $('#description').css('display', "none");
14         //console.log(url);
15         $.ajax({
16             url : '/api/calc?a=' +
document.getElementById('a').value + '&b=' + docu\
17 ment.getElementById('b').value,
18             success: function(data) {
19                 $('#add').html(data['a'] + ' + ' +
data['b'] + ' = ' + data['add']);
20                 $('#subtract').html(data['a'] + ' - ' +
data['b'] + ' = ' + data['su\
21 btract']);
22                 $('#multiply').html(data['a'] + ' * ' +
data['b'] + ' = ' + data['mu\
23 ltiply']);
24                 $('#divide').html(data['a'] + ' / ' +
data['b'] + ' = ' + data['divi\
25 de']);
26             }
27         });
28     });
29 })
```

```
1 <html>
2 <head>
3 </head>
4 <body>
5 <input type="number" id="a">
6 <input type="number" id="b">
```

```
7 <button id="calc">Calc</button>
8 <div id="results">
9   <div id="add"></div>
10  <div id="subtract"></div>
11  <div id="multiply"></div>
12  <div id="divide"></div>
13 </div>
14
15 <pre id="info"></pre>
16 <div id="description"></div>
17
18 <script src="/static/jquery-3.1.1.min.js"></script>
19 <script src="/static/math.js?r={ reload }"></script>
20 </body>
21 </html>
```

passlib

```
1 from passlib.hash import pbkdf2_sha256
2 import sys
3
4 if len(sys.argv) != 2:
5     exit("Usage: {} PASSWORD".format(sys.argv[0]))
6
7 pw = sys.argv[1]
8
9 hash1 = pbkdf2_sha256.hash(pw)
10 print(hash1)
11
12 hash2 = pbkdf2_sha256.hash(pw)
13 print(hash2)
14
15 print(pbkdf2_sha256.verify(pw, hash1))
16 print(pbkdf2_sha256.verify(pw, hash2))
```

```
1 $ python use_passlib.py "my secret"
2 $pbkdf2-
sha256$29000$svZ.7z2HEEJIiVHqPeecMw$QAWd8P7MaPDx1Ewtsv9Aq
hFEP2hp8MvZ9QxasIw4\
3 Pgw
4 $pbkdf2-
```

```
sha256$29000$XQuh9N57r9W69x6jtDaG0A$VtD35zfeZhXsE/jxG16wB  
7Mjwj/5iDGZv6QC7XBJ\  
5 jrI  
6 True  
7 True
```

Flask Testing

```
1 from flask import Flask, jsonify  
2 myapp = Flask(__name__)  
3  
4 @myapp.route("/")  
5 def main():  
6     return ''  
7 Main <a href="/add/23/19">add</a>  
8 ''  
9  
10 @myapp.route("/add/<int:a>/<int:b>")  
11 def api_info(a, b):  
12     return str(a+b)
```

```
1 from app import myapp  
2 import unittest  
3  
4 # python -m unittest test_app  
5  
6  
7 class TestMyApp(unittest.TestCase):  
8     def setUp(self):  
9         self.app = myapp.test_client()  
10  
11     def test_main(self):  
12         rv = self.app.get('/')  
13         assert rv.status == '200 OK'  
14         assert b'Main' in rv.data  
15         #assert False  
16  
17     def test_add(self):  
18         rv = self.app.get('/add/2/3')  
19         self.assertEqual(rv.status, '200 OK')  
20         self.assertEqual(rv.data, '5')
```

```
21
22     rv = self.app.get('/add/0/10')
23     self.assertEqual(rv.status, '200 OK')
24     self.assertEqual(rv.data, '10')
25
26     def test_404(self):
27         rv = self.app.get('/other')
28         self.assertEqual(rv.status, '404 NOT FOUND')
```

Flask Deploy app

```
1 from flask import Flask
2 myapp = Flask(__name__)
3
4 @myapp.route("/")
5 def main():
6     return 'Main'
```

uwsgi

```
1 [uwsgi]
2 socket      = :9091
3 plugin      = python
4 wsgi-file   = /home/gabor/work/my/app.py
5 process     = 3
6 callable    = myapp
```

nginx

```
1 server {
2     server_name example.com;
3
4
5     access_log  /var/log/nginx/example.log  main;
6     error_log   /var/log/nginx/example.error.log;
7
8     location ~ /.git/ {
9         deny all;
10    }
11}
```

```
12     #error_page 404 /404.html;
13
14     location '/' {
15         include uwsgi_params;
16         uwsgi_pass 127.0.0.1:9091;
17     }
18
19     root /home/gabor/work/example.com/html/;
20 }
```

Flask Simple Authentication + test

```
1 from flask import Flask
2 from flask_httpauth import HTTPBasicAuth
3 from werkzeug.security import generate_password_hash,
check_password_hash
4
5 app = Flask(__name__)
6 auth = HTTPBasicAuth()
7
8 users = {
9     "john": generate_password_hash("nhoj"),
10    "jane": generate_password_hash("enaj")
11 }
12
13 @app.route("/")
14 def hello():
15     return "Hello World!"
16
17 @auth.verify_password
18 def verify_password(username, password):
19     if username in users:
20         return
21     check_password_hash(users.get(username), password)
22     return False
23
24 @app.route("/admin")
25 @auth.login_required
26 def admin():
27     return "Hello Admin"
```

```

1 import app
2 import base64
3
4 def test_app():
5     web = app.app.test_client()
6
7     rv = web.get('/')
8     assert rv.status == '200 OK'
9     assert rv.data == b'Hello World!'
10
11 def test_admin_unauth():
12     web = app.app.test_client()
13
14     rv = web.get('/admin')
15     assert rv.status == '401 UNAUTHORIZED'
16     assert rv.data == b'Unauthorized Access'
17     assert 'WWW-Authenticate' in rv.headers
18     assert rv.headers['WWW-Authenticate'] == 'Basic realm="Authentication Required"'
19
20 def test_admin_auth():
21     web = app.app.test_client()
22
23     credentials =
24         base64.b64encode(b'john:nhoj').decode('utf-8')
25     rv = web.get('/admin', headers={
26         'Authorization': 'Basic ' + credentials
27     })
28
29     assert rv.status == '200 OK'
30     assert rv.data == b'Hello Admin'

```

Flask REST API

- [flask-restful](#)

Flask REST API - Echo

```

1 from flask import Flask, request
2 from flask_restful import Api, Resource
3

```

```
4 app = Flask(__name__)
5
6 api = Api(app)
7
8 class Echo(Resource):
9     def get(self):
10         return { "prompt": "Type in something" }
11     def post(self):
12         return { "echo": "This" }
13
14 api.add_resource(Echo, '/echo')
```

```
1 import api
2
3 def test_echo():
4     web = api.app.test_client()
5
6     rv = web.get('/echo')
7     assert rv.status == '200 OK'
8     assert rv.headers['Content-Type'] ==
'application/json'
9     assert rv.json == {"prompt": "Type in something"}
10
11
12     rv = web.post('/echo')
13     assert rv.status == '200 OK'
14     assert rv.headers['Content-Type'] ==
'application/json'
15     assert rv.json == {"echo": "This"}
```

Flask REST API - parameters in path

```
1 from flask import Flask, request
2 from flask_restful import Api, Resource
3
4 app = Flask(__name__)
5
6 api = Api(app)
7
8 class Echo(Resource):
9     def get(self, me):
```

```
10         return { "res": f"Text: {me}" }
11
12     def post(self, me):
13         return { "Answer": f"You said: {me}" }
14
15
16 api.add_resource(Echo, '/echo/<me>')
```

```
1 import api
2
3 def test_echo():
4     web = api.app.test_client()
5
6     rv = web.get('/echo/hello')
7     assert rv.status == '200 OK'
8     assert rv.headers['Content-Type'] ==
'application/json'
9     assert rv.json == { 'res': 'Text: hello'}
10
11
12     rv = web.post('/echo/ciao')
13     assert rv.status == '200 OK'
14     assert rv.headers['Content-Type'] ==
'application/json'
15     assert rv.json == { 'Answer': 'You said: ciao'}
```

Flask REST API - parameter parsing

```
1 from flask import Flask, request
2 from flask_restful import Api, Resource, reqparse
3
4 app = Flask(__name__)
5
6 api = Api(app)
7
8
9 class Echo(Resource):
10     def get(self):
11         parser = reqparse.RequestParser()
12         parser.add_argument('text', help='Type in some
text')
```

```
13         args = parser.parse_args()
14     return { "res": f"Text: {args['text']}"} }
15
16 def post(self):
17     parser = reqparse.RequestParser()
18     parser.add_argument('text', help='Type in some
text')
19     args = parser.parse_args()
20     return { "Answer": f"You said: {args['text']}"} }
21
22
23 api.add_resource(Echo, '/echo')
```

```
1 import api
2
3 def test_echo():
4     web = api.app.test_client()
5
6     rv = web.get('/echo?text=hello')
7     assert rv.status == '200 OK'
8     assert rv.headers['Content-Type'] ==
'application/json'
9     assert rv.json == { 'res': 'Text: hello'}
10
11    rv = web.post('/echo', data={'text': 'ciao'})
12    assert rv.status == '200 OK'
13    assert rv.headers['Content-Type'] ==
'application/json'
14    assert rv.json == { 'Answer': 'You said: ciao'}
15
16
17     # If the parameter is missing the parser just
returns None
18    rv = web.get('/echo')
19    assert rv.status == '200 OK'
20    assert rv.headers['Content-Type'] ==
'application/json'
21    assert rv.json == { 'res': 'Text: None'}
```

Flask REST API - parameter parsing - required

```
1 from flask import Flask, request
2 from flask_restful import Api, Resource, reqparse
3
4 app = Flask(__name__)
5
6 api = Api(app)
7
8
9 class Echo(Resource):
10     def get(self):
11         parser = reqparse.RequestParser()
12         parser.add_argument('text', help='Type in some
text', required=True)
13         args = parser.parse_args()
14         return { "res": f"Text: {args['text']}" }
15
16     def post(self):
17         parser = reqparse.RequestParser()
18         parser.add_argument('text', help='Type in some
text')
19         args = parser.parse_args()
20         return { "Answer": f"You said: {args['text']}" }
21
22
23 api.add_resource(Echo, '/echo')
```

```
1 import api
2
3 def test_echo():
4     web = api.app.test_client()
5
6     rv = web.get('/echo?text=hello')
7     assert rv.status == '200 OK'
8     assert rv.headers['Content-Type'] ==
'application/json'
9     assert rv.json == { 'res': 'Text: hello'}
10
11    rv = web.post('/echo', data={'text': 'ciao'})
```

```
12     assert rv.status == '200 OK'
13     assert rv.headers['Content-Type'] ==
14         'application/json'
15
16
17     # If the parameter is missing the parser just
18     # returns None
18     rv = web.get('/echo')
19     assert rv.status == '400 BAD REQUEST'
20     assert rv.headers['Content-Type'] ==
21         'application/json'
21     assert rv.json == {'message': {'text': 'Type in
some text'}}}
```

Networking

Secure shell

- subprocess + external ssh client
- [Paramiko](#)
- [Fabric](#)

ssh

- On Windows install [putty](#)

```
1 import subprocess
2 import sys
3
4 if len(sys.argv) !=2:
5     exit("Usage: " + sys.argv[0] + " hostname")
6
7 host = sys.argv[1]
8 command = "uname -a"
9
10 ssh = subprocess.Popen(["ssh", host, command],
11                         shell=False,
12                         stdout=subprocess.PIPE,
13                         stderr=subprocess.PIPE)
14 result = ssh.stdout.readlines()
15 error = ssh.stderr.readlines()
16 if error:
17     for err in error:
18         sys.stderr.write("ERROR: {}\\n".format(err))
19 if result:
20     print(result)
```

ssh from Windows

```
1 $ ssh foobar@hostname-or-ip
2   -o "StrictHostKeyChecking no"
3
4 $ plink.exe -ssh foobar@hostname-or-ip -pw "password" -
C "uname -a"
5 $ plink.exe", "-ssh", "foobar@username-or-ip", "-pw",
"no secret", "-C", "uname -a"
```

```
1 import subprocess
2 import sys
3
4 ssh =
subprocess.Popen([r"c:\Users\foobar\download\plink.exe",
"-ssh",
      "foobar@username-or-ip",
      "-pw", "password",
      "-C", "uname -a"],
      shell=False,
      stdout=subprocess.PIPE,
      stderr=subprocess.PIPE)
11 result = ssh.stdout.readlines()
12 error = ssh.stderr.readlines()
13 if error:
14     for err in error:
15         sys.stderr.write("ERROR: {}\\n".format(err))
16 if result:
17     print(result)
```

Parallel ssh

- [parallel-ssh](#)
- **pip install parallel-ssh**

```
1 from pssh import ParallelSSHClient
2 hosts = ['myhost1', 'myhost2']
3 client = ParallelSSHClient(hosts)
4 output = client.run_command('ls -ltrh /tmp/',
sudo=True)
```

telnet

```
1 import telnetlib
2
3 hostname = '104.131.87.33'
4 user = 'gabor'
5 password = 'robag'
6
7 tn = telnetlib.Telnet(hostname)
8 tn.read_until("login: ")
9 tn.write(user + "\n")
10
11 tn.read_until("Password: ")
12 tn.write(password + "\n")
13 tn.read_until("~$")
14
15 tn.write("hostname\n")
16 print(tn.read_until("~$"))
17 print("-----");
18
19
20 tn.write("uptime\n")
21 print(tn.read_until("~$"))
22 print("-----");
23
24
25 print("going to exit")
26 tn.write("exit\n")
27
28 print("-----")
29 print(tn.read_all())
```

prompt for password

```
1 import getpass
2
3 password = getpass.getpass("Password:")
4
5 print(password)
```

Python nmap

```
1 import nmap
2 nm = nmap.PortScanner()
3 nm.scan('127.0.0.1', '20-1024')
4 print(nm.command_line())
5
6 for host in nm.all_hosts():
7     print('-----')
8     print('Host : {} ({})'.format(host,
nm[host].hostname()))
9     print('State : {}'.format(nm[host].state()))
10    for proto in nm[host].all_protocols():
11        print('-----')
12        print('Protocol : {}'.format(proto))
13
14        lport = nm[host][proto].keys()
15        for port in lport:
16            print ('port : {}\\tstate :
{}{}'.format(port, nm[host][proto][port]['state'],
17 '']))
```

```
1 nmap -oX - -p 10-1024 -sV 127.0.0.1
2 -----
3 Host : 127.0.0.1 ()
4 State : up
5 -----
6 Protocol : tcp
7 port : 21      state : open
8 port : 22      state : open
9 port : 23      state : open
```

ftp

```
1 $ sudo aptitude install proftpd
2 $ sudo /etc/init.d/proftpd start
3 $ sudo adduser    (user: foo pw: bar)
```

```
 1 from ftplib import FTP
 2 ftp = FTP('localhost')
 3 ftp.login("foo", "bar")
 4
 5 print(ftp.retrlines('LIST'))
 6
 7 print('-----')
 8 for f in ftp.nlst():
 9     print("file: " + f)
10
11 filename = 'ssh.py'
12
13 ftp.storlines("STOR " + filename, open(filename))
14
15 print('-----')
16 for f in ftp.nlst():
17     print("file: " + f)
18
19 ftp.delete(filename)
20
21 print('-----')
22 for f in ftp.nlst():
23     print("file: " + f)
24
25
26
```

```
 1 -rw-rw-r--    1 foo        foo          6 Feb 18 19:18
a.txt
 2 -rw-rw-r--    1 foo        foo          6 Feb 18 19:18
b.txt
 3 226 Transfer complete
 4 -----
 5 file: b.txt
 6 file: a.txt
 7 -----
 8 file: b.txt
 9 file: a.txt
10 file: ssh.py
11 -----
12 file: b.txt
13 file: a.txt
```

Interactive shell

The Python interactive shell

Type `python` without any arguments on the command line and you'll get into the Interactive shell of Python.

In the interactive shell you can type:

```
1 >>> print "hello"
2 hello
3
4 >>> "hello"
5 'hello'
6
7 >>> 6
8 6
9
10 >>> len("abc")
11 3
12
13 >>> "abc" + 6
14 Traceback (most recent call last):
15   File "<stdin>", line 1, in <module>
16 TypeError: cannot concatenate 'str' and 'int' objects
17
18 >>> "abc" + str(6)
19 'abc6'
```

REPL - Read Evaluate Print Loop

A variable comes to existence the first time we assign a value to it.

It points to an object and that object knows about its type.

```
1 >>> a = "abc"
2 >>> len(a)
3 3
4
5 >>> a = '3'
6 >>> a + 3
7 Traceback (most recent call last):
8   File "<stdin>", line 1, in &lt;module>
9 TypeError: cannot concatenate 'str' and 'int' objects
10
11 >>> int(a) + 3
12 6
13
14 >>> a = '2.3'
15 >>> float(a) + 1
16 3.3
```

Using Modules

Python has lots of standard (and not standard) modules. You can load one of them using the

`import` keyword. Once loaded, you can use functions from the module

or access its objects. For example the `sys` module has a `sys.version` and a `sys.executable` variable.

```
1 >>> import sys
2 >>> sys.version
3 '2.7.3 (default, Apr 10 2012, 23:24:47) [MSC v.1500 64
bit (AMD64)]'
```

```
1 >>> sys.executable
2 'c:\\Python27\\python.exe'
```

You can also load specific object directly into your code.

```
1 >>> from sys import executable
2 >>> executable
3 'c:\\Python27\\python.exe'
```

To quit the interpreter call the `exit()` function.

```
1 >>> exit
2 Use exit() or Ctrl-Z plus Return to exit
```

The `import` binds the word `sys` to whatever it loaded from the file.

Getting help

```
1 >>> help
2 Type help() for interactive help, or help(object) for
help about object.
3 >>> help()      - entering an internal shell:
4 ...
5 help> dir      - explains about the dir command.
Navigate using SPACE/ENTER/q
6 help> Ctrl-D   - to quite, (Ctrl-Z ENTER on Windows)
7 >>> help(dir)  - the same explanation as before
8
9 >>> dir()
10['__builtins__', '__doc__', '__name__', '__package__']
11>>> dir("")    - list of string related methods
12['__add__', '__class__', ... 'upper', 'zfill']
13
14>>> dir(1)     - list of integer related methods
15['__abs__', '__add__', ... 'numerator', 'real']
16
17>>> dir(__builtins__)
18...                  - functions available in python
19
20>>> help(abs)      - explain how abs() works
21>>> help(sum)
22>>> help(zip)
23>>> help(int)
24>>> help(str)
```

```
25
26 >>> help("") .upper)      - explain how the upper method of
strings work
27
28 >>> import sys
29 >>> dir(sys)
30 >>> help(sys)
31 >>> help(sys)
32 >>> help(sys.path)
33 >>> help(sys.path.pop)
```

Exercise: Interactive shell

- Start the REPL and check the examples.
- Check the documentation in the REPL.

Testing Demo

How do you test your code?

* What kind of things do you test?

- Web application?
- Command line application?
- Databases?
- ...

What is testing?

- Fixture + Input = Expected Output

What is testing really?

- Fixture + Input = Expected Output + Bugs

Testing demo - AUT - Application Under Test

Given the following module with a single function, how can we use this function and how can we test it?

```
1 def add(x, y):  
2     return x * y  
3  
4 # Yes, I know there is a bug in this code!
```

Testing demo - use the module

```
1 import mymath
2 import sys
3
4 if len(sys.argv) != 3:
5     exit(f"Usage {sys.argv[0]} NUMBER NUMBER")
6
7 a = int(sys.argv[1])
8 b = int(sys.argv[2])
9
10 print(mymath.add(a, b))
```

```
1 python use_mymath.py 2 2
2 4
```

Testing demo: doctests

```
1 def add(x, y):
2     """
3     >>> add(2, 2)
4     4
5     """
6     return x * y
```

```
1 python -m doctest mymath_doctest_first.py
2 echo $?
3 0
4
5 echo %ERRORLEVEL%
6 0
```

```
1 def add(x, y):
2     """
3     >>> add(2, 2)
4     4
5     >>> add(3, 3)
6     6
```

```
7     """
8     return x * y


---



```
1 python -m doctest mymath_doctest.py
2 echo $?
3 1

```
1
*****
***** File "/home/gabor/work/slides/python/examples/testing-
demo/mymath_doctest.py", line \
3 5, in mymath_doctest.add
4 Failed example:
5     add(3, 3)
6 Expected:
7     6
8 Got:
9     9
10
*****
***** 1 items had failures:
12     1 of    2 in mymath_doctest.add
13 ***Test Failed*** 1 failures.
```

```


```

Testing demo: Unittest success

```
1 import unittest
2 import mymath
3
4 class TestMath(unittest.TestCase):
5     def test_math(self):
6         self.assertEqual(mymath.add(2, 2), 4)


---


```

```
1 python -m unittest test_one_with_unittest.py
2 echo $?
3 0
```

```
1 .
2 -----
3 -----
4 Ran 1 test in 0.000s
5
5 OK
```

Testing demo: Unittest failure

```
1 import unittest
2 import mymath
3
4 class TestMath(unittest.TestCase):
5     def test_math(self):
6         self.assertEqual(mymath.add(2, 2), 4)
7
8     def test_more_math(self):
9         self.assertEqual(mymath.add(3, 3), 6)
```

```
1 python -m unittest test_with_unittest.py
2 echo $?
3 1
```

```
1 .F
2 -----
3 =====
3 FAIL: test_more_math (test_with_unittest.TestMath)
4 -----
4 -----
5 Traceback (most recent call last):
6   File
"/home/gabor/work/slides/python/examples/testing-
demo/test_with_unittest.py",
7   line 9, in test_more_math
8       self.assertEqual(mymath.add(3, 3), 6)
9 AssertionError: 9 != 6
10
11 -----
```

```
12 Ran 2 tests in 0.000s
13
14 FAILED (failures=1)
```

Testing demo: pytest using classes

```
1 import mymath
2
3 class TestMath():
4     def test_math(self):
5         assert mymath.add(2, 2) == 4
6
7     def test_more_math(self):
8         assert mymath.add(3, 3) == 6
```

```
1 pytest test_with_pytest_class.py
```

```
1 ===== test session starts =====
2 platform linux -- Python 3.7.3, pytest-5.1.1, py-
1.8.0, pluggy-0.13.0
3 rootdir:
/home/gabor/work/slides/python/examples/testing-demo
4 plugins: flake8-1.0.4
5 collected 2 items
6
7 test_with_pytest_class.py .F
[100%]
8
9 ===== FAILURES =====
10 _____ TestMath.test_more_math
11
12 self = <test_with_pytest_class.TestMath object at
0x7fc1ea617828>
13
14     def test_more_math(self):
15 >         assert mymath.add(3, 3) == 6
16 E             assert 9 == 6
17 E                 + where 9 = <function add at 0x7fc1ea6caf28>
```

```
(3, 3)
18 E           +   where <function add at 0x7fc1ea6caf28> =
mymath.add
19
20 test_with_pytest_class.py:8: AssertionError
21 ===== 1 failed, 1 passed in 0.03s
=====
```

Testing demo: pytest without classes

```
1 import mymath
2
3 def test_math():
4     assert mymath.add(2, 2) == 4
5
6 def test_more_math():
7     assert mymath.add(3, 3) == 6
```

```
1 pytest test_with_pytest.py
```

```
1 ===== test session starts
=====
2 platform linux -- Python 3.7.3, pytest-5.1.1, py-
1.8.0, pluggy-0.13.0
3 rootdir:
/home/gabor/work/slides/python/examples/testing-demo
4 plugins: flake8-1.0.4
5 collected 2 items
6
7 test_with_pytest.py .F
[100%]
8
9 ===== FAILURES
=====
10 _____ test_more_math
=====
11
12     def test_more_math():
13 >         assert mymath.add(3, 3) == 6
14 E         assert 9 == 6
15 E             +   where 9 = <function add at 0x7f36e78db0d0>
```

```
(3, 3)
16 E           +   where <function add at 0x7f36e78db0d0> =
mymath.add
17
18 test_with_pytest.py:7: AssertionError
19 ===== 1 failed, 1 passed in 0.02s
=====
```

Testing demo: pytest run doctests

```
1 pytest --doctest-modules mymath_doctest_first.py
2 pytest --doctest-modules mymath_doctest.py
```

Testing demo: pytest run unittest

```
1 pytest -v test_with_unittest.py
```

Exercise: Testing demo

- An anagram is a pair of words that are created from exactly the same set of characters, but of different order.
- For example **listen** and **silent**
- Or **bad credit** and **debit card**
- Given the following module with the **is_anagram** function write tests for it. (in a file called `test_anagram.py`)
- Write a failing test as well.
- Try **doctest**, **unittest**, and **pytest** as well.

```
1 def is_anagram(a_word, b_word):
2     return sorted(a_word) == sorted(b_word)
```

Sample code to use the Anagram module.

```
1 from anagram import is_anagram
2 import sys
3
4 if len(sys.argv) != 3:
5     exit(f"Usage {sys.argv[0]} WORD WORD")
6
7 if is_anagram(sys.argv[1], sys.argv[2]):
8     print("Anagram")
9 else:
10    print("NOT")
```

Solution: Testing demo

```
1 from anagram import is_anagram
2
3 def test_anagram():
4     assert is_anagram("silent", "listen")
5     assert is_anagram("bad credit", "debit card")
6
7 def test_not_anagram():
8     assert not is_anagram("abc", "def")
9
10 def test_should_be_anagram_spaces():
11     assert is_anagram("anagram", "nag a ram")
12
13
14 def test_should_be_anagram_case():
15     assert is_anagram("Silent", "Listen")
```

Types in Python

mypy

- [mypy](#)
 - [Type Checking](#)
 - [type hints](#)
-

¹ pip install mypy

Types of variables

```
1 x :int = 0
2
3 x = 2
4 print(x)
5
6 x = "hello"
7 print(x)
```

python variables.py

```
1 2
2 hello
```

mypy variables.py

```
1 variables.py:7: error: Incompatible types in assignment
(expression has type "str", \
2 variable has type "int")
3 Found 1 error in 1 file (checked 1 source file)
```

Types of function parameters

```
1 def add(a :int, b :int) -> int:  
2     return a+b  
3  
4 print(add(2, 3))  
5 print(add("Foo", "Bar"))
```

```
1 5  
2 FooBar
```

```
1 function.py:6: error: Argument 1 to "add" has  
incompatible type "str"; expected "int"  
2 function.py:6: error: Argument 2 to "add" has  
incompatible type "str"; expected "int"  
3 Found 2 errors in 1 file (checked 1 source file)
```

Types used properly

```
1 def add(a :int, b :int) -> int:  
2     return a+b  
3  
4 print(add(2, 3))  
5  
6 x :int = 0  
7  
8 x = 2  
9 print(x)
```

```
1 5  
2 2
```

```
1 Success: no issues found in 1 source file
```

TODO: mypy

- Complex data structures?
- My types/classes?
- Allow None (or not) for a variable.

Testing Intro

The software testing equasion

`1 INPUT + PROCESS = EXPECTED_OUTPUT`

The software testing equasion (fixed)

`1 INPUT + PROCESS = EXPECTED_OUTPUT + BUGS`

The pieces of your software?

- Web application with HTML interface?
- Web application with HTML + JavaScript? Which frameworks?
- Web application with JSON interface? (API)
- What kind of databases do you have in the system? SQL? NoSQL? What size is the database?
- Source and the format of your input? Zip? CSV? XML? SQL Dump? JSON?
- The format of your output?
HTML/PDF/CSV/JSON/XML/Excel/Email/..?
- Are you pushing out your results or are your clients pulling them? e.g. via an API?
- What external dependencies do you have? Slack, Twilio,
What kind of APIs?

Manual testing

- How do you check your application now?

What to tests?

- Happy path
- Sad path
- Valid input
- Valid edge cases (0, -1, empty string, etc.)
- Broken input (string instead of number, invalid values, too long input, etc.)
- Extreme load
- System failure (power failure, network outage, lack of memory, lack of disk, ...)
- Third-party error or failure - How does your system work if the 3rd party API returns rubbish?

Continuous Integration

- Reduce feedback cycle
- Avoid regression
- On every push
- Every few hours full coverage

Functional programming

Functional programming

- Immutability (variables don't change)
- Separation of data and functions.
- First-class functions (you can assign function to another name and you can pass function to other functions and return them as well. We can also manipulate functions)
- Higher order functions: a functions that either takes a function as a parameter or returns a function as a parameter.

Iterators (Iterables)

You already know that there are all kinds of objects in Python that you can iterate over using the **for in** construct.

For example you can iterate over the characters of a string, or the elements of a list, or whatever **range()** returns.

You can also iterate over the lines of a file

and you have probably seen the **for in** construct in other cases as well. The objects that can be iterated over are collectively called

iterables.

You can do all kind of interesting things on such **iterables**. We'll see a few now.

- A few data type we can iterate over using the **for ... in ...** construct. (strings, files, tuples, lists, list comprehension)

```
1 numbers = [101, 2, 3, 42]
2 for num in numbers:
3     print(num)
4 print(numbers)
5
6 print()
7
8 name = "FooBar"
9 for cr in name:
10    print(cr)
11 print(name)
12
13 print()
14
15 rng = range(3, 9, 2)
16 for num in rng:
17    print(num)
18 print(rng)
```

```
1 101
2 2
3 3
4 42
5 [101, 2, 3, 42]
6
7 F
8 O
9 O
10 B
11 a
12 r
13 FooBar
14
15 3
16 5
17 7
18 range(3, 9, 2)
```

range

So what does **range** really return?

Instead of returning the list of numbers (as it used to do in Python 2), now it returns a **range object** that provides “the opportunity to go over the specific series of numbers” without actually creating the **list** of numbers. Getting an object instead of the whole list has a number of advantages.

One is space. In the next example we’ll see how much memory is needed for the object returned by the **range** function and how much would it take to have the corresponding list of numbers in memory. For now let’s see how can we use it:

- `range(start, end, step)`
- `range(start, end) - step defaults to 1`
- `range(end) - start defaults to 0, step defaults to 1`

```
1 rng = range(3, 9, 2)
2 print(rng)
3 print(type(rng).__name__)
4
5 print()
6
7 for num in rng:
8     print(num)
9
10 print()
11
12 for num in rng:
13     print(num)
```

```
14  
15 print()  
16  
17 print(rng[2])
```

```
1 range(3, 9, 2)  
2 range  
3  
4 3  
5 5  
6 7  
7  
8 3  
9 5  
10 7  
11  
12 7
```

range with list

Using the **list** function we can tell the **range** object to generate the whole list immediately. Either using the variable that holds the **range** object, or wrapping the **range()** call in a **list()** call.

You might recall at the beginning of the course we saw the **sys.getsizeof()** function that returns the size of a Python object in the memory. If you don't recall, no problem, we'll see it used now. As you can see the size of the range object is only 48 bytes while the size of the 3-element list is already 112 bytes. It seems the range object is better than even such a short lists.
On the next page we'll see a more detailed analysis.

```
1 import sys
2
3 rng = range(3, 9, 2)
4 numbers = list(rng)
5 print(rng)          # range(3, 9, 2)
6 print(numbers)      # [3, 5, 7]
7
8 others = list(range(2, 11, 3))
9 print(others)      # [2, 5, 8]
10
11 print(sys.getsizeof(rng))      # 48
12 print(sys.getsizeof(numbers)) # 112
```

range vs. list size

In this example we have a loop iterating over **range(21)**, but that's only for the convenience, the interesting part is inside the loop.

On every iteration call **range()** with the current number, then we convert the resulting object into a list of numbers. Finally we print out

the current number and the size of both the object returned by **range()** and the list generated from the object. As you can see the memory usage

of the **range** object remains the same 48 bytes, while the memory usage of the list growth as the list gets longer.

```
1 import sys
2
3 for ix in range(21):
4     rng = range(ix)
5     numbers = list(rng)
6     print("{:>3} {:>3} {:>4}".format(ix,
sys.getsizeof(rng), sys.getsizeof(numbers)))
```

```
1 0 48 64
2 1 48 96
3 2 48 104
4 3 48 112
5 4 48 120
6 5 48 128
7 6 48 136
8 7 48 144
9 8 48 160
10 9 48 192
11 10 48 200
12 11 48 208
13 12 48 216
14 13 48 224
15 14 48 232
16 15 48 240
17 16 48 256
18 17 48 264
19 18 48 272
20 19 48 280
21 20 48 288
```

for loop with transformation

There are many cases when we have a list of some values and we need to apply some transformation to each value. At the end we would like to have the list of the resulting values.

A very simple such transformation would be to double each value. Other, more interesting examples might be reversing each string, computing some more complex function on each number, etc.)

In this example we just double the values and use **append** to add each value to the list containing the results.

```
1 def double(n):
2     return 2 * n
3
4 numbers = [1, 2, 3, 4]
5 name = "FooBar"
6
7 double_numbers = []
8
9 for num in numbers:
10    double_numbers.append( double(num) )
11 print(double_numbers)
12
13 double_letters = []
14 for cr in name:
15    double_letters.append( double(cr) )
16 print(double_letters)
```

```
1 [2, 4, 6, 8]
2 ['FF', 'oo', 'oo', 'BB', 'aa', 'rr']
```

There are better ways to do this.

map

- `map(function, iterable, ...)`

The `map` function of Python applies a function to every item in an iterable and returns an iterator that can be used to iterate over the results. Wow, how many times I repeated the word iter...something. Instead of trying to untangle that sentence, let's look at the following example:

We have a list of numbers in the brilliantly named variable `numbers` with 1, 2, 3, 4 as the content. We could like to create a list of all the doubles (so that would be 2, 4, 6, 8 in this case) and then iterate over them printing them on the screen. Sure, you probably have some more complex operation to do on the numbers than simple double them, but in this example I did not want to complicate that part. Suffice to say that you have some computation to do in every element.

So you encapsulate your computation in a regular Python function (in our case the function is called `double`). Then you call `map` and pass to it two parameters. The first parameter is the `double` function itself, the second parameter is the list of the values you would like to work on. `map` will go over all the values in the `numbers` list, call the `double` function with each number and provide allow you to iterate over the results. Something like this:

```
double_numbers = [ double(1), double(2), double(3),
double(4)]
```

Except, that the above is not true.

When Python executes the `double_numbers = map(double, numbers)` line, no computation happens and no resulting list is created. Python only prepares “the possibility to do the computations”. In the upcoming examples we’ll see what does this sentence really mean, for now let’s see what do we have in this example: `double_numbers` contains a **map object**, but when you iterate over it using the **for num in double_numbers** construct you get the expected values.

In the second half of the example you can see the same works on strings as well.

```
1 def double(n):
2     return 2 * n
3
4 numbers = [1, 2, 3, 4]
5 name = "FooBar"
6
7 double_numbers = map(double, numbers)
8 print(double_numbers)    # <map object at
```

```
0x7f8eb2d849e8>
9 for num in double_numbers:
10     print(num)
11
12 double_letters = map(double, name)
13 print(double_letters)    # <map object at
0x7f8eb2d84cc0>
14 for cr in double_letters:
15     print(cr)
```

```
1 <map object at 0x7ff0c0d89da0>
2 2
3 4
4 6
5 8
6 <map object at 0x7ff0c0d89a20>
7 FF
8 oo
9 oo
10 BB
11 aa
12 rr
```

map delaying function call

In this example we have added a call to `print` in the `double` function in order to see when it really executed. You can see that the first output comes from the `print` statement that was **after** the `map` call. Then on each iteration we see the output from inside the “`double`” function and then the result from the loop. In a nutshell Python does not execute the “`double`” function at the point where we called `map`. It only executes it when we iterate over the resulting object.

```
1 def double(n):
2     print(f"double {n}")
3     return 2 * n
4
5 numbers = [1, 4, 2, -1]
6
7 double_numbers = map(double, numbers)
8 print(double_numbers)
9
10 for num in double_numbers:
11     print(num)
```

```
1 <map object at 0x7f90df760f98>
2 double 1
3 2
4 double 4
5 8
6 double 2
7 4
8 double -1
9 -2
```

map on many values

Now imagine you have a very long list. I know this is not such a long list, but I trust you can imagine a long list of numbers. We would like to run some function on each element and then iterate over the results, but what if at one point in the iteration we decide to break out of the loop?

```
1 import sys
2
3 def double(n):
4     print(f"double {n}")
5     return 2 * n
```

```
6
7 numbers = [1, 4, 2, -1, 23, 12, 5, 6, 34, 143123, 98,
213]
8
9 double_numbers = map(double, numbers)
10 print(double_numbers)
11 for num in double_numbers:
12     print(num)
13     if num > 42:
14         break
15
16 print()
17 print(sys.getsizeof(numbers))
18 print(sys.getsizeof(double_numbers))
```

```
1 <map object at 0x7fe5c5270d68>
2 double 1
3 2
4 double 4
5 8
6 double 2
7 4
8 double -1
9 -2
10 double 23
11 46
12
13 160
14 56
```

You can see that it did not need to waste time calculating the doubles of all the values, as it was calculating on-demand. You can also see that the object returned from `map` takes up only 56 bytes. Regardless of the size of the original array.

map with list

Here too you can use the **list** function to convert all the values at once, but there is an advantage of keeping it as a **map object**. Not only the size that we already saw with the **range** case, but also the processing time saved by not calculating the results till you actually need it.

Imagine a case where you apply several expensive (time consuming) transformations to some original list and then you iterate over the end-results

looking for the first value that matches some condition. What if you find the value you were looking for after only a few iteration. Then

making all that expensive calculations to the whole list was a waste of time.

This lazy evaluation can help you save both memory and time and you always have the option to force the immediate calculation by calling the **list** function.

```
1 def double(num):
2     return 2 * num
3
4 numbers = [1, 2, 3, 4]
5 name = "FooBar"
6
7 double_numbers = list(map(double, numbers))
8 print(double_numbers)
9
```

```
10 double_letters = list(map(double, name))  
11 print(double_letters)
```

```
1 [2, 4, 6, 8]  
2 ['FF', 'oo', 'oo', 'BB', 'aa', 'rr']
```

double with lambda

There are many other cases besides **map** where we need to pass a function as a parameter to some other function.

Many cases the function we pass is some almost trivial function with a single operation in it.

In those cases creating a named function like the “double” function in the previous examples is an overkill.

In this example we also used the **list** function to force the full evaluation of the map object to make it easier to show the results. Normally you probably would not use the **list** function here.

```
1 numbers = [1, 2, 3, 4]  
2 name = "FooBar"  
3  
4  
5 double_numbers = list( map( lambda n: n * 2, numbers)  
)  
6 print(double_numbers)  
7  
8  
9 double_letters = map( lambda n: n * 2, name)  
10 for cr in double_letters:  
11     print(cr)
```

```
1 [2, 4, 6, 8]
2 FF
3 oo
4 oo
5 BB
6 aa
7 rr
```

What is lambda in Python?

Lambda creates simple anonymous function. It is simple because it can only have one statement in its body. It is anonymous because usually it does not have a name.

The usual use is as we saw earlier when we passed it as a parameter to the `map` function. However, in the next example we show that you can assign the lambda-function to a name and then you could used that name just as any other function you would define using `def`.

```
1 def dbl(n):
2     return 2*n
3 print(dbl(3))
4
5 double = lambda n: 2*n
6 print(double(3))
```

```
1 6
2 6
```

lambda returning tuple

A lambda function can return complex data structures as well.
e.g. a tuple.

```
1 dbl = lambda n: (n, 2*n)
2
3 ret = dbl(12)
4
5 print(ret)
```

```
1 (12, 24)
```

map returning tuples

```
1 numbers = [1, 2, 3, 4]
2
3 pairs = map(lambda n: (n, 2*n), numbers)
4 print(pairs)
5
6 for pair in pairs:
7     print(pair)
```

```
1 <map object at 0x7fcd264a15f8>
2 (1, 2)
3 (2, 4)
4 (3, 6)
5 (4, 8)
```

lambda with two parameters

A **lambda**-function can have more than one parameters:

```
1 add = lambda x,y: x+y
2 print(add(2, 3))
```

```
1 5
```

map for more than one iterable

Lets “add” together two lists of numbers. Using + will just join the two lists together, but we can use the “map” function to add the values pair-wise.

```
1 v1 = [1, 3, 5, 9]
2 v2 = [2, 6, 4, 8]
3
4 v3 = v1 + v2
5 print(v3)
6
7 sums = map(lambda x,y: x+y, v1, v2)
8 print(sums)
9 print(list(sums))
```

```
1 [1, 3, 5, 9, 2, 6, 4, 8]
2 <map object at 0x7fcbecc8c668>
3 [3, 9, 9, 17]
```

map on uneven lists

In **Python 3** the iterator stops when the shortest iterable is exhausted.

In **Python 2** it used to extend the shorter lists by **None** values.

```
1 v1 = [1, 3, 5, 9]
2 v2 = [2, 6, 4, 8, 10]
3
4 sums = map(lambda x,y: x+y, v1, v2)
5 print(sums)
6
7 print(list(sums))
```

```
1 <map object at 0x7ff9469a8da0>
2 [3, 9, 9, 17]
```

replace None (for Python 2)

In Python 2 map used to extend the shorter lists by **None** values. So to avoid exceptions, we had some extra code replacing the None values by 0, using the ternary operator.

```
1 v1 = [1, 3, 5, 9]
2 v2 = [2, 6, 4, 8, 10]
3
4 print(map(lambda x,y: (0 if x is None else x) + (0 if y
is None else y), v1, v2))
5 # [3, 9, 9, 17, 10]
```

map on uneven lists - fixed (for Python 2)

A nicer fix was this:

```
1 v1 = [1, 3, 5, 9]
2 v2 = [2, 6, 4, 8, 10]
3
```

```
4 print(map(lambda x,y: (x or 0) + (y or 0), v1, v2))
5 # [3, 9, 9, 17, 10]
```

map mixed iterators

map works on any iterable, so we might end up passing one list and one string to it.

```
1 v1 = ['foo', 'bar', 'baz']
2 v2 = 'abc'
3
4 result = map(lambda x,y: x+y, v1, v2)
5 print(result)
6 print( list(result) )
```

```
1 <map object at 0x7fc5e9ff4e80>
2 ['fooa', 'barb', 'bazc']
```

map fetch value from dict

```
1 people = [
2     {
3         'name': 'Foo',
4         'phone': '123',
5     },
6     {
7         'name': 'Bar',
8         'phone': '456',
9     },
10    {
11        'name': 'SnowWhite',
12        'phone': '7-dwarfs',
13    }
14 ]
15
16 names = map(lambda d: d['name'], people)
```

```
17  
18 print(names)  
19 print(list(names))
```

```
1 <map object at 0x7f5afffaeb00>  
2 ['Foo', 'Bar', 'SnowWhite']
```

Exercise: string to length

Given a list of strings, create an iterator that will provide the length of each string.

Exercise: row to length

Given a file, create an iterator that will provide the length of each row. Can you do it without actually reading the file?

Exercise: compare rows

Create an iterator that given two files will return true for each line where the first space in the first file is earlier than the first space in the second file. So

- given: “ab cd” vs “abc d” the value is true
- given: “ab cd” vs “ab cd” the value is false
- given: “ab cd” vs “a bcd” the value is false

Solution: string to length

```
1 animals = ['chicken', 'cow', 'snail', 'elephant',  
'pig', 'zebra', 'gnu', 'praying ma\  
2 ntiss', 'snake']  
3  
4 length = map(len, animals)
```

```
5 print(length)
6 print(list(length))
```

Solution: row to length

```
1 filename = __file__ # just being lazy and using
2         ourselves as the input file
3
4 with open(filename) as fh:
5     length = map(len, fh)
6     print(length)
7     for ln in length:
8         print(ln)
9         # if ln > 10:
10            #      break
```

Solution: compare rows

```
1 import sys
2
3 file_a = 'map_string_to_len.py'
4 file_b = 'map_row_to_length.py'
5
6 def compare(row_a, row_b):
7     a = row_a.find(' ')
8     b = row_b.find(' ')
9     return a < b
10
11 with open(file_a) as fh_a, open(file_b) as fh_b:
12     results = map(compare, fh_a, fh_b)
13     print(results)
14     print(sys.getsizeof(results))
15
16     truth = list(results)
17     print(truth)
18     print(sys.getsizeof(truth))
```

```
1 <map object at 0x7f0858d3f8d0>
2 56
```

```
3 [False, True, False, True, True]
4 128
```

filter

- `filter(function, iterable)`

`filter` will return an iterable object that will return all the items of the original iterable that evaluate the function to **True**. This can have only one iterable!

```
1 numbers = [1, 3, 27, 10, 38]
2 def big(n):
3     return n > 10
4
5 reduced = filter(big, numbers)
6 print(reduced)
7 print(list(reduced))
```

```
1 <filter object at 0x7f4bc37355c0>
2 [27, 38]
```

filter with lambda

```
1 numbers = [1, 3, 27, 10, 38]
2
3 reduced = filter(lambda n: n > 10, numbers)
4 print(reduced)
5 print(list(reduced))
```

```
1 <filter object at 0x7faed0fe57b8>
2 [27, 38]
```

filter - map example

```
1 numbers = [1, 7, 19, 5, 57, 23, 8]
2
3 def big(x):
4     print(f"filtering {x}")
5     return x > 10
6
7 def double(y):
8     print(f"double {y}")
9     return 2*y
10
11 big_numbers = filter(big, numbers)
12 print(big_numbers)
13
14 doubles = map(double, big_numbers)
15 print(doubles)
16
17 for num in doubles:
18     print(num)
```

```
1 <filter object at 0x7ffad9f82f28>
2 <map object at 0x7ffad9f829e8>
3 filtering 1
4 filtering 7
5 filtering 19
6 double 19
7 38
8 filtering 5
9 filtering 57
10 double 57
11 114
12 filtering 23
13 double 23
14 46
15 filtering 8
```

filter - map in one expression

```
1 numbers = [1, 7, 19, 5, 57, 23, 8]
2
```

```
3 def big(x):
4     print(f"filtering {x}")
5     return x > 10
6
7 def double(y):
8     print(f"double {y}")
9     return 2*y
10
11
12 for num in map(double, filter(big, numbers)):
13     print(num)
```

```
1 filtering 1
2 filtering 7
3 filtering 19
4 double 19
5 38
6 filtering 5
7 filtering 57
8 double 57
9 114
10 filtering 23
11 double 23
12 46
13 filtering 8
```

Get indexes of values

`filter` can help us get a sublist of values from an iterable, eg. from a list that match some condition.

In this example we see how to get all the names that are exactly 3 characters long.

What if, however if instead of the values themselves, you would like to know their location? The indexes of the places where these value can be found. In that case, you would run the `filter` on the indexes from 0 till the last valid index of the list. You can do that using the `range` function.

Finally there is another example that shows how to get the indexes of all the names that have an “e” in them.
Just to show you that we can use any arbitrary condition there.

```
1 names = ["Helen", "Ann", "Mary", "Harry", "Joe",
"Peter"]
2 names3 = filter(lambda w: len(w) == 3, names)
3 print( list(names3) )
4
5 loc3 = filter(lambda i: len(names[i]) == 3,
range(len(names)))
6 print( list(loc3) )
7
8
9 has_e = filter(lambda i: "e" in names[i],
range(len(names)))
10
11 print( list(has_e) )
```

```
1 ['Ann', 'Joe']
2 [1, 4]
3 [0, 4, 5]
```

reduce

In Python 2 it was still part of the language.

```
reduce(function, iterable[, initializer])
```

```
1 from functools import reduce
2
3 numbers = [1, 2, 3, 4]
4
5 print(reduce(lambda x,y: x+y, numbers))      # 10 =
((1+2)+3)+4
6 print(reduce(lambda x,y: x*y, numbers))      # 24 =
((1*2)*3)*4
7 print(reduce(lambda x,y: x/y, [8, 4, 2]))    # 1.0
8
9 print(reduce(lambda x,y: x+y, [2]))          # 2
10 print()
11
12 # print(reduce(lambda x,y: x+y, []))
13     # TypeError: reduce() of empty sequence with no
14     initial value
14 print(reduce(lambda x,y: x+y, [], 0))        # 0
15 print(reduce(lambda x,y: x+y, [2,4], 1))      # 7
16 print()
17
18 mysum = 0
19 for num in numbers:
20     mysum += num
21 print(mysum)                                # 10
22
23 mymultiple = 1
24 for num in numbers:
25     mymultiple *= num
26 print(mymultiple)                          #24
```

```
1 10
2 24
3 1.0
4 2
5
6 0
7 7
8
9 10
10 24
```

The initializer is used as the 0th element returned by the iterable. It is mostly interesting in case the iterable is empty.

reduce with default

```
1 from functools import reduce
2
3 print( reduce(lambda x,y: x+y, [], 0) )          # 0
4 print( reduce(lambda x,y: x+y, [1, 2], 0) )      # 3
5
6 print( reduce(lambda x,y: x*y, [1, 2], 0) )      # 0
7 print( reduce(lambda x,y: x*y, [2, 3], 1) )      # 6
8 print( reduce(lambda x,y: x*y, [], 0) )          # 0
```

```
1 0
2 3
3 0
4 6
5 0
```

zip

```
1 fname = ['Graham',           'Eric',           'Terry',
2             'Terry',           'John',           'Michael']
3 lname = ['Chapman',          'Idle',           'Palin']
4             'Gilliam',          'Cleese',         'Jones'
5 born   = ['8 January 1941',  '29 March 1943',  '22
6 November 1940',
7             '1 February 1942',  '27 October 1939', '5 May
1943']
```

```
8 for f_name, l_name, b_date in zip(fname, lname, born):
9     print("{} {} was born {}".format(f_name,
10        l_name, b_date))
```

```
1 Graham      Chapman      was born 8 January 1941
2 Eric        Idle        was born 29 March 1943
```

```
3 Terry Gilliam was born 22 November 1940
4 Terry Jones was born 1 February 1942
5 John Cleese was born 27 October 1939
6 Michael Palin was born 5 May 1943
```

Monty Python

Creating dictionary from two lists using zip

```
1 names = ['Jan', 'Feb', 'Mar', 'Apr']
2 days = [31, 28, 31, 30]
3
4 zipped = zip(names, days)
5 print(zipped)
6
7 pairs = list(zipped)
8 print(pairs)
9
10 month = dict(zipped)
11 print(month) # this is empty because zipped was
already exhausted by the "list" ca\
12 ll
13
14 zipped = zip(names, days)
15 month = dict(zipped)
16 print(month)
```

```
1 <zip object at 0x7ff021949788>
2 [('Jan', 31), ('Feb', 28), ('Mar', 31), ('Apr', 30)]
3 {}
4 {'Jan': 31, 'Feb': 28, 'Mar': 31, 'Apr': 30}
```

all, any

- `all(iterable)` - returns True if all the elements of iterable return True
- `any(iterable)` - returns True if any of the elements in iterable return True

```
1 a = [True, True]
2 b = [True, False]
3 c = [False, False]
4
5 print(all(a))      # True
6 print(all(b))      # False
7 print(all(c))      # False
8 print()
9 print(any(a))      # True
10 print(any(b))     # True
11 print(any(c))     # False
```

Compare elements of list with scalar

```
1 print(2 > 1) # True
2 print(0 > 1) # False
3 print()
4
5 numbers = [2, 4]
6 # Comparing different types does not make sense, but
nevertheless Python 2 would stil\
7 ll do it.
8 # Python 3 raises exception:
9 # TypeError: '>' not supported between instances of
'list' and 'int'
10 # print(numbers > 1) # True
11 # print(numbers > 7) # True
12 # print()
13
14 # compare each element with the scalar and then check
if 'all' were True
15 print(all(map(lambda x: x > 1, numbers))) # True
16 print(all(map(lambda x: x > 2, numbers))) # False
```

List comprehension - double

We take the original example where we had a function called double, and this time we

write a different expression to run the function on every element of an iterable.

```
1 def double(n):
2     return 2*n
3
4 numbers = [1, 2, 3, 4]
5 name = "FooBar"
6
7 double_numbers = [double(n) for n in numbers]
8 print(double_numbers) # [2, 4, 6, 8]
9
10
11 double_chars = [double(n) for n in name]
12 print(double_chars) # ['FF', 'oo', 'oo', 'BB', 'aa',
'rr']
```

List comprehension - simple expression

```
1 import sys
2
3 numbers = [0, 1, 2, 3]
4
5 sqrs = map(lambda n: n*n, numbers)
6 print(sqrs)          # <map object at 0x7fdcab2f5940>
7 print(list(sqrs))   # [0, 1, 4, 9]
8 print(sys.getsizeof(sqrs))
9 print()
10
11 squares = [n*n for n in numbers]
12 print(squares)    # [0, 1, 4, 9]
13 print(sys.getsizeof(squares))
```

```
1 <map object at 0x7fa9cf2eb9e8>
2 [0, 1, 4, 9]
3 56
4
5 [0, 1, 4, 9]
6 96
```

List generator

Going over the values of the generator will empty the generator.

```
1 import sys
2
3 numbers = [0, 1, 2, 3, 4, 5, 6]
4
5 gn = (n*n for n in numbers)
6 print(gn)
7 print(sys.getsizeof(gn))
8 print()
9
10 for num in gn:
11     print(num)
12 print()
13
14 gn = (n*n for n in numbers)
15 squares = list(gn)
16 print(sys.getsizeof(squares))
17 print(squares)
18
19 print(list(gn)) # the generator was already exhausted
```

```
1 <generator object <genexpr> at 0x7f8c0bda2930>
2 120
3
4 0
5 1
6 4
7 9
8 16
9 25
10 36
11
12 160
13 [0, 1, 4, 9, 16, 25, 36]
14 []
```

List comprehension

```
1 text = ['aaaa', 'bb', 'ccc ccc']
2
3 length_1 = map(lambda x: len(x), text)
4 print(length_1)          # <map object at 0x7f60ceb90f98>
5 print(list(length_1)) # [4, 2, 7]
6
7
8 length_2 = map(len, text)
9 print(length_2)          # <map object at 0x7f60ceb90fd0>
10 print(list(length_2)) # [4, 2, 7]
11
12
13 length_3 = [ len(s)   for s in text ]
14 print(length_3) # [4, 2, 7]
```

In LISP this would be a `mapcar`.

Dict comprehension

```
1 people = {
2     'Foo':           123,
3     'Bar':           456,
4     'SnowWhite':    7,
5 }
6
7 doubles = { k:v**2 for (k, v) in people.items() }
8 print(doubles)  # {'Foo': 246, 'Bar': 912, 'SnowWhite':
14}
```

Lookup table with lambda

```
1 import sys
2
3 table = {
4     "cat"  : lambda : print("miau"),
5     "dog"  : lambda : print("hauhau"),
```

```
6     "duck" : lambda : print("hap hap"),
7 }
8
9
10 def main():
11     if len(sys.argv) != 2:
12         exit(f"Usage: {sys.argv[0]} {NAME}")
13
14     animal = sys.argv[1]
15     if animal in table:
16         table[animal]()
17
18 main()
```

Read lines without newlines

```
1 import sys
2
3 if len(sys.argv) != 2:
4     exit(f"Usage: {sys.argv[0]}")
5
6 filename = sys.argv[1]
7
8 with open(filename) as fh:
9     rows = map(lambda s: s.rstrip("\n"),
10 fh.readlines())
11 for row in rows:
12     print(row)
```

Read key-value pairs

```
1 name=Foo Bar
2 email=foo@bar.com
3 address=Foo street 42
```

```
1 import sys
2
3 if len(sys.argv) != 2:
4     exit(f"Usage: {sys.argv[0]}")
```

```
5
6 filename = sys.argv[1]
7
8 with open(filename) as fh:
9     pairs = dict(map(lambda x: x.split('='),
10                      map(lambda s: s.rstrip("\n"), fh.readlines())))
11
12 print(pairs)
```

```
1 {'name': 'Foo Bar', 'email': 'foo@bar.com', 'address':
2  'Foo street 42'}
```

Create index-to-value mapping in a dictionary based on a list of values

```
1 planned_order = ('b', 'c', 'd', 'a')
2 plan = dict(zip(range(len(planned_order)),
3                  planned_order))
4 print(plan)
```

```
1 {0: 'b', 1: 'c', 2: 'd', 3: 'a'}
```

Exercise: min, max, factorial

- Implement an expression to calculate “min”, and another expression to calculate “max” of lists.
- Implement an expression that will calculate factorial. $f(n)$ should return the value of $n!$ ($n! = n * (n-1) * (n-2) * \dots * 1$)
- Implement an expression that given 2 lists will return a new list in which each element is the `max()` for each pair from the input lists. E.g. given [1, 3, 6] and [2, 4, 5] the result is [2, 4, 6]
- Use `reduce`, `map`, `lambda`

Exercise: Prime numbers

Calculate and print the prime numbers between 2 and N. Use filter.

Exercise: Many validator functions

Given several validator functions (that get a parameter and return True or False),
and given a list of values, return a sublist of values that pass all
the validation
checks. See the sekeleton:

```
1 def is_big(x):
2     return x > 100
3
4 def is_even(x):
5     return not x % 2
6
7 numbers = [90, 102, 101, 104]
8
9 cond = [is_big, is_even]
10
11 # z = ...
12 print(z) # [102, 104]
```

Exercise: Calculator using lookup table

Write a script that will accept a math expression such as `python calc.py 2 + 3` and will print the result.

Use lookup tables select the implementation of the actual computation. (supporting +, - , *, /) is enought

Exercise: parse file

In the following file we have lines:

```
1 SOURCE/Filename.json, TARGET
```

read in the file and create

- a single dictionary where the SOURCE/Filename.json is the key and the TARGET is the value.
- list of dictionaries in which the keys are ‘source’, ‘filename’, and ‘target’ and the values are from the respective columns (SOURCE, FILENAME.json, and TARGET)

You can solve this for-loop or with map and list-comprehensions.
Do it in both ways.

```
1 agile/agile.json,agile
2 ansible/ansible.json,ansible
3 ansible-intro/ansible.json,ansible-intro
4 aws-lambda/aws.json,aws-lambda
5 bash/bash.json,bash
6 css/css.json,css
7 collab-dev/collab.json,collab-dev
8 data-science/data.json,data-science
9 dart-programming/dart.json,dart-programming
10 docker/docker.json,docker
11 google-gcp/gcp.json,google-gcp
12 git/git.json,git
13 git-intro/git.json,git-intro
14 github-ci/github-ci.json,github-ci
15 golang/go.json,golang
16 groovy/groovy.json,groovy
17 java-programming/java.json,java-programming
18 javascript-programming/javascript.json,javascript-
programming
19 jenkins/jenkins.json,jenkins
20 jenkins-intro/jenkins.json,jenkins-intro
21 linux/linux.json,linux
22 linux-intro/linux.json,linux-intro
23 mobile/mobile.json,mobile
24 mojolicious/mojolicious.json,mojolicious
25 mongodb/mongodb.json,mongodb
26 nodejs/nodejs.json,nodejs
```

```

27 nosql/nosql.json,nosql
28 pair-programming/pair.json,pair-programming
29 perl-intro/perl.json,perl-intro
30 perl-programming/perl.json,perl-programming
31 perl-programming/testing.json,test-automation-using-
perl
32 php-programming/php.json,php-programming
33 programming/programming.json,programming
34 python-mocking/python.json,python-mocking
35 python-programming/python.json,python-programming
36 ruby-programming/ruby.json,ruby=programming
37 sql/sql.json,sql
38 value/value.json,value
39 vim/vim.json,vim
40 web/web.json,web
41 windows-cmd/windows.json,windows-cmd
42 talks/real_world.json,real-world
43 talks/github-pages.json,github-pages
44 talks/python-pair-programming-and-tdd-
workshop.json,python-pair-programming-and-tdd-\
45 workshop

```

Solution: min, max, factorial

```

1 from functools import reduce
2
3 numbers = [2, 1, 4, 3]
4
5 # min
6 print(reduce(lambda x,y: x if x < y else y, numbers))
# 1
7 # max
8 print(reduce(lambda x,y: x if x > y else y, numbers))
# 4
9
10 # factorial
11 n = 4
12 print(reduce(lambda x,y: x*y, range(1, n+1), 1))    #
24
13 # The 1 at the end is the initializer of reduce to
provide
14 # correct results for n = 0.
15

```

```
16 a = [1, 3, 6]
17 b = [2, 4, 5]
18 c = map(lambda x,y: x if x > y else y, a, b)
19 print(list(c)) # [2, 4, 6]
```

Solution: Prime numbers

Calculating the prime numbers

```
1 n = 50
2
3 nums = range(2, n)
4 for i in range(2, 1+int(n ** 0.5)):
5     nums = filter(lambda x: x == i or x % i, nums)
6
7 print(nums)
```

Solution: Many validator functions

```
1 def is_big(x):
2     return x > 100
3
4 def is_even(x):
5     return not x % 2
6
7 numbers = [90, 102, 101, 104]
8
9 cond = [is_big, is_even]
10
11 z = filter( lambda n: all([f(n) for f in cond]), numbers)
12 print(z) # [102, 104]
```

Solution: Calculator using lookup table

```
1 import sys
2
3 table = {
4     "+" : lambda x, y: x+y,
```

```
5     "-": lambda x, y: x-y,
6     "*": lambda x, y: x*y,
7     "/": lambda x, y: x/y,
8 }
9
10
11 def main():
12     if len(sys.argv) != 4:
13         exit(f"Usage: {sys.argv[0]} NUMBER OP NUMBER")
14     action = table[sys.argv[2]]
15     print( action(int(sys.argv[1]), int(sys.argv[3])))
16
17 main()
```

map with condition

The conversion function can do anything. It can have a condition inside.

```
1 numbers = [1, 2, 3, 4]
2
3 def cond_double(n):
4     if n % 2:
5         return 2*n
6     else:
7         return n
8
9 cd = map(cond_double, numbers)
10 print(cd) # [2, 2, 6, 4]
```

map with lambda

```
1 numbers = [1, 2, 3, 4]
2
3 def dbl(x):
4     return 2*x
```

```
5 d1 = map(dbl, numbers)
6 print(d1)  # [2, 4, 6, 8]
7
8 double = lambda x: 2*x
9 d2 = map(double, numbers)
10 print(d2)  # [2, 4, 6, 8]
11
12 d3 = map(lambda n: 2*n, numbers)
13 print(d3)  # [2, 4, 6, 8]
```

map with lambda with condition

```
1 numbers = [1, 2, 3, 4]
2
3 a = map(lambda n: 2*n if n % 2 else n, numbers)
4 print(a)  # [2, 2, 6, 4]
```

List comprehension - complex

```
1 numbers = [1, 3, 2, 4]
2
3 t = filter(lambda n: n > 2, numbers)
4 print(t)  # [3, 4]
5
6 n1 = map(lambda n: n*n, t)
7 print(n1) # [9, 16]
8
9
10 n2 = map(lambda n: n*n, filter(lambda n: n > 2,
numbers))
11 print(n2)  # [9, 16]
12
13
14
15 n3 = [ n*n for n in numbers if n > 2 ]
16 print(n3) # [9, 16]
```

Iterators - with and without Itertools

Advantages of iterators and generators

- Lazy evaluation
- Save processing (or at least delay the use)
- Save memory
- Handle an infinite series of information
- Turn complex operations into a simple matter of `for` loop.

The Fibonacci research institute

- We have a bunch of mathematicians who research the Fibonacci series.
- We have a bunch of people who research a series of DNA sequences.
- ???

Fibonacci plain

- We don't call this as this has an infinite loop

```
1 def fibonacci():
2     a, b = 0, 1
3     while True:
4         a, b = b, a+b
5
6 # fibonacci()
```

Fibonacci copy-paste

```
1 def fibonacci():
2     a, b = 0, 1
3     while True:
4         a, b = b, a+b
5
6         print(a)
7         if a % 17 == 0:
8             print('found')
9             break
10
11        if a > 200:
12            print('not found')
13            break
14
15 fibonacci()
```

Iterators Glossary

- [iterable](#) (Can be iterated over using a `for` loop.)
- [iterator](#)
- Every iterator is also iterable
- Iterators (and iterables) are not necessarily addressable like lists with the `thing[index]` construct.
- [Iterator Types](#)
- [The standard type hierarchy](#)

What are iterators and iterables?

- All of them are iterables
- A filehandle and the map object are also iterators. (Side note: You should always open files using the `with` statement and not like this.)
- `iter()` would return the iterator from an iterable. We don't need this.

```
1 from collections.abc import Iterator, Iterable
2
3 a_string = "Hello World"
4 a_list = ["Tiger", "Mouse"]
5 a_tuple = ("Blue", "Red")
6 a_range = range(10)
7 a_fh = open(__file__)
8 a_map = map(lambda x: x**2, a_list)
9
10 for thing in [a_string, a_list, a_tuple, a_range,
a_map, a_fh]:
11     print(thing.__class__.__name__)
12     print(issubclass(thing.__class__, Iterator))
13     print(issubclass(thing.__class__, Iterable))
14     zorg = iter(thing)
15     print(zorg.__class__.__name__)
16     print(issubclass(zorg.__class__, Iterator))
17
18     print()
19
20 a_fh.close()
```

```
1 str
2 False
3 True
4 str_iterator
5 True
6
7 list
8 False
9 True
10 list_iterator
11 True
12
13 tuple
14 False
15 True
16 tuple_iterator
17 True
18
19 range
20 False
21 True
```

```
22 range_iterator
23 True
24
25 TextIOWrapper
26 True
27 True
28 TextIOWrapper
29 True
```

A file-handle is an iterator

This slightly a repetition of the previous statement, that filehandles are iterators.

```
1 from collections.abc import Iterator, Iterable
2 from io import TextIOWrapper
3
4 with open(__file__) as fh:
5     print(fh.__class__.__name__)
6     print(issubclass(fh.__class__, TextIOWrapper))
7     print(issubclass(fh.__class__, Iterator))
8     print(issubclass(fh.__class__, Iterable))
9
10    for line in fh:
11        pass
12        #print(line, end="")
```

```
1 TextIOWrapper
2 True
3 True
4 True
```

range is iterable but it is not an iterator

Just as a string or a list, the `range` function in Python is also an “iterable” but it is not an “iterator”.

In many aspects it behaves as an iterator. Specifically it allows us to iterate over numbers.

Range Is Not An Iterator

- range

```
1 for n in range(2, 12, 3):
2     print(n)
3 print()
4
5 for n in range(3):
6     print(n)
7 print()
8
9 for n in range(2, 5):
10    print(n)
11 print()
12
13 from collections.abc import Iterator, Iterable
14 rng = range(2, 5)
15 print(issubclass(rng.__class__, Iterator))
16 print(issubclass(rng.__class__, Iterable))
```

```
1 2
2 5
3 8
4 11
5
6 0
7 1
8 2
9
10 2
11 3
12 4
13
```

```
14 False  
15 True
```

Iterator: a counter

We can create a iterator using a class. We are required to implement the `__iter__` method that returns the iterator object and the `__next__` method that returns the next element in our iteration. We can indicated that the iteration was exhausted by raising a `StopIteration` exception.

The instance-object that is created from this class-object is the iterator, not the class-object itself!

- `__iter__`
- `__next__` (in Python 2 this used to called `next`)
- `raise StopIteration`

```
1 class Counter():  
2     def __init__(self):  
3         self.count = 0  
4  
5     def __iter__(self):  
6         return self  
7  
8     def __next__(self):  
9         self.count += 1  
10        if self.count > 3:  
11            raise StopIteration  
12        return self.count
```

Using iterator

The class returned an iterator, we could use a `for` loop to iterate over the element.

We tried to run through the iterator again, but it did not print anything. It was exhausted.

```
1 from counter import Counter  
2  
3 cnt = Counter()  
4 for c in cnt:  
5     print(c)  
6  
7 for c in cnt:  
8     print(c)
```

```
1 1  
2 2  
3 3
```

Iterator without temporary variable

```
1 from counter import Counter  
2  
3 for c in Counter():  
4     print(c)
```

```
1 1  
2 2  
3 3
```

The type of the iterator

How can we know it is an iterator? We check it.

```
1 from collections.abc import Iterator, Iterable
2 from counter import Counter
3
4 cnt = Counter()
5 print(cnt.__class__.__name__)
6 print(issubclass(cnt.__class__, Iterator))
7 print(issubclass(cnt.__class__, Iterable))
```

```
1 Counter
2 True
3 True
```

Using iterator with next

A feature of any iterator is that we could iterate over it using the `next` call.

```
1 from counter import Counter
2
3 cnt = Counter()
4
5 while True:
6     try:
7         a = next(cnt)
8         print(a)
9     except Exception as ex:
10        print(ex.__class__.__name__)
11        break
```

```
1 1
2 2
3 3
4 StopIteration
```

Mixing for and next

You can even use `next` inside a `for` loop, but then you will have to handle the `StopIteration` exception that might happen during your call of `next`.

I am not really sure when would we want to use this.

```
1 from counter import Counter
2
3 cnt = Counter()
4
5 for i in cnt:
6     print(f"i: {i}")
7     try:
8         n = next(cnt)
9         print(f"n: {n}")
10    except Exception as ex:
11        print(ex.__class__.__name__)
12        break
```

```
1 i: 1
2 n: 2
3 i: 3
4 StopIteration
```

Iterable which is not an iterator

```
1 from counter import Counter
2
3 class GetMyIterable():
4     def __init__(self):
5         pass
6     def __iter__(self):
7         return Counter()
8
```

```
9
10 thing = GetMyIterable()
11
12 from collections.abc import Iterator, Iterable
13 print(issubclass(thing.__class__, Iterator))
14 print(issubclass(thing.__class__, Iterable))
15
16 for i in thing:
17     print(i)
```

```
1 False
2 True
3 1
4 2
5 3
```

Iterator returning multiple values

```
1 class SquareCounter():
2     def __init__(self):
3         self.count = 0
4
5     def __iter__(self):
6         return self
7
8     def __next__(self):
9         self.count += 1
10        if self.count > 5:
11            raise StopIteration
12        return self.count, self.count ** 2
13
14 for cnt, sqr in SquareCounter():
15     print(f"{cnt} {sqr}")
```

```
1 1 1
2 2 4
3 3 9
4 4 16
5 5 25
```

Range-like iterator

```
1 class Range():
2     def __init__(self, start, end):
3         self.current = start
4         self.end = end
5
6     def __iter__(self):
7         return self
8
9     def __next__(self):
10        if self.current >= self.end:
11            raise StopIteration
12        v = self.current
13        self.current += 1
14        return v
```

```
1 import it
2
3 r = it.Range(1, 4)
4 for n in r:
5     print(n)
6
7 print('---')
8
9 for n in it.Range(2, 5):
10    print(n)
```

```
1 1
2 2
3 3
4 ---
5 2
6 3
7 4
```

Unbound or infinite iterator

So far each iterator had a beginning and an end. However we can also create infinite or unbounded iterators.

The nice thing about them is that we can pass them around as we do with any other object and we can execute operations on them without burning our CPU.

Of course the user will have to be careful not to try to flatten the iterator, not to try to get all the values from it, as that will only create an infinite loop or a never ending operation.

In this very simple example we count from 0 and we never stop.

When we use the `Counter` in the `for` loop we need to include a stop-condition, otherwise our loop will never end.

```
1 class Counter():
2     def __init__(self):
3         self.count = 0
4
5     def __iter__(self):
6         return self
7
8     def __next__(self):
9         self.count += 1
10        return self.count
11
12 for c in Counter():
13     print(c)
14     if c > 10:
15         break
```

```
1 1
2 2
3 3
4 4
5 5
```

```
6 6
7 7
8 8
9 9
10 10
11 11
```

Unbound iterator Fibonacci

Now we can get back to our original problem, the slightly more complex Fibonacci series. In this example we created an unbounded iterator that on every iteration will return the next element of the Fibonacci series.

```
1 class Fibonacci():
2     def __init__(self):
3         self.values = []
4
5     def __iter__(self):
6         return self
7
8     def __next__(self):
9         if len(self.values) == 0:
10             self.values.append(1)
11             return 1
12
13         if len(self.values) == 1:
14             self.values.append(1)
15             return 1
16
17         self.values.append(self.values[-1] +
self.values[-2])
18         self.values.pop(0)
19
20         return self.values[-1]
```

```
1 from fibonacci import Fibonacci
2 for v in Fibonacci():
3     print(v)
4     if v > 10:
5         break
```

```
1 1
2 1
3 2
4 3
5 5
6 8
7 13
```

Operations on Unbound iterator

```
1 from fibonacci import Fibonacci
2
3 fib = Fibonacci()
4
5 #odd = [x for x in fib if x % 2 == 1]
6 odd = filter(lambda x: x % 2 == 1, fib)
7
8 print("Let's see")
9
10 for v in odd:
11     print(v)
12     if v > 10:
13         break
```

```
1 Let's see
2 1
3 1
4 3
5 5
6 13
```

itertools

- [itertools](#)

itertools is a standard Python library that provides a number of interesting iterators.

We are going to see a few examples here:

itertools - count

- Unbound counter: Count from N to infinity.

```
1 import itertools
2
3 for c in itertools.count(start=19, step=1):
4     print(c)
5     if c > 23:
6         break
7
8 # 19
9 # 20
10 # 21
11 # 22
12 # 23
13 # 24
```

itertools - cycle

```
1 import itertools
2
3 ix = 0
4 for c in itertools.cycle(['A', 'B', 'C']):
5     print(c)
6     ix += 1
7     if ix >= 5:
8         break
9
```

```
10 print(' ')
11
12 ix = 0
13 for c in itertools.cycle('DEF'):
14     print(c)
15     ix += 1
16     if ix >= 5:
17         break
```

```
1 A
2 B
3 C
4 A
5 B
6
7 D
8 E
9 F
10 D
11 E
```

Exercise: iterators - reimplement the range function

In one of the first slides of this chapter we saw a partial implementation of the `range` function.

Change that code to have a full implementation, that can accept 1, 2, or 3 parameters.

Exercise: iterators - cycle

- Reimplement the cycle functions of `itertools` using iterator class.

Exercise: iterators - alter

- Implement the alter functions as an iterator that will return
-

```
1 1
2 -2
3 3
4 -4
5 5
6 -6
7 ...
```

- Optionally provide a start and end parameters
- start defaults to 1
- end defaults to unlimited

Exercise: iterators - limit Fibonacci

Change the Iterator version of the Fibonacci series so optionally you will be able to provide a parameter called “limit” to the constructor. If the limit is provided, the iterator should stop when the value passes the limit.

Exercise: iterators - Fibonacci less memory

Change the Iterator version of the Fibonacci series so it will NOT hold the previous values in memory.

Exercise: read char

Create an iterator that given a filename will return an object that on every iteration will return a single character. As an option let the user skip newlines, or maybe any pre-defined character.

Exercise: read section

- Create an iterator that given the name of a file like the following, will return one section at a time.
- It will return a list one each iteration and each element of the list will be a line from the current section.
- Other ideas what should be returned on each iteration?

```
1 name      = Mercury
2 distance  = 0.4
3 mass      = 0.055
4
5
6 name      = Venus
7 distance  = 0.7
8 mass      = 0.815
9
10
11 name     = Earth
12 distance = 1
13 mass     = 1
14
15 name     = Mars
16 distance = 1.5
17 mass     = 0.107
```

Exercise: collect packets

- You get a series of packets (e.g. lines in a file)
- In each line you have several fields: id, seqid, maxseq, content
- id is a unique identifier of a series of packets (lines)
- seqid is the sequence id of a packet in a series. (an integer)
- maxseq is the length of the sequence.
- content is the actual content.

In each iteration return a message that is built up from all the packages in the given sequence.

```
1 12,1,5,First of Twelve
2 12,2,5,Second of Twelve
3 12,3,5,Third of Twelve
4 12,4,5,Fourth of Twelve
5 12,5,5,Fifth of Twelve
6
7 9,1,4,First of Nine
8 9,2,4,Second of Nine
9 9,3,4,Third of Nine
10 9,4,4,Fourth of Nine
11
12 11,1,3,First of Eleven
13 11,2,3,Second of Eleven
14 11,3,3,Third of Eleven
```

```
1 ['First of Twelve', 'Second of Twelve', 'Third of
Twelve', 'Fourth of Twelve', 'Fift\
2 h of Twelve']
3 ['First of Nine', 'Second of Nine', 'Third of Nine',
'Fourth of Nine']
4 ['First of Eleven', 'Second of Eleven', 'Third of
Eleven']
```

```
1 12,1,5,First of Twelve
2 11,1,3,First of Eleven
3 9,1,4,First of Nine
4 12,2,5,Second of Twelve
5 9,2,4,Second of Nine
6 11,2,3,Second of Eleven
7 12,3,5,Third of Twelve
8 9,3,4,Third of Nine
9 12,4,5,Fourth of Twelve
10 12,5,5,Fifth of Twelve
11 9,4,4,Fourth of Nine
12 11,3,3,Third of Eleven
```

```
1 11,2,3,Second of Eleven
2 11,1,3,First of Eleven
```

```
3 9,1,4,First of Nine
4 12,1,5,First of Twelve
5 9,3,4,Third of Nine
6 9,2,4,Second of Nine
7 12,3,5,Third of Twelve
8 12,4,5,Fourth of Twelve
9 12,2,5,Second of Twelve
10
11 12,5,5,Fifth of Twelve
12 9,4,4,Fourth of Nine
13 11,3,3,Third of Eleven
```

Exercise: compare files

Compare two files line-by-line, and create a 3rd file listing the lines that are different.

```
1 One
2 Two
3 Three
4 Four
5 Five
```

```
1 One
2 Two
3 Tree
4 Four
5 Five
```

Expected output:

```
1 2,Three,Tree
```

Solution: iterators - limit Fibonacci

```
1 class Fibonacci:
2     def __init__(self, limit=0):
3         self.values = []
```

```
4         self.limit = limit
5     def __iter__(self):
6         return self
7     def next(self):
8         if self.limit and len(self.values) >=
self.limit:
9             raise StopIteration
10        if len(self.values) == 0:
11            self.values.append(1)
12            return 1
13        if len(self.values) == 1:
14            self.values.append(1)
15            return 1
16        self.values.append(self.values[-1] +
self.values[-2])
17        return self.values[-1]
```

```
1 import fibonacci
2 f = fibonacci.Fibonacci(limit = 10)
3 print(f)
4 for v in f:
5     print(v)
6
7 print('-----')
8 f = fibonacci.Fibonacci()
9 for v in f:
10    print(v)
11    if v > 30:
12        break
```

Solution: iterators - Fibonacci less memory

```
1 class Fibonacci:
2     def __init__(self, limit=0):
3         self.values = []
4         self.limit = limit
5     def __iter__(self):
6         return self
7     def next(self):
8         if self.limit and len(self.values) and
self.values[-1] >= self.limit:
```

```
9         raise StopIteration
10        if len(self.values) == 0:
11            self.values = (1, )
12        return 1
13        if len(self.values) == 1:
14            self.values = (1, 1)
15        return 1
16        self.values = (self.values[-1],
17                      self.values[-1] + self.values[-2])
18    return self.values[-1]
```

```
1 import fibonacci
2 f = fibonacci.Fibonacci(limit = 10)
3 print(f)
4 for v in f:
5     print(v)
6
7 print('-----')
8 f = fibonacci.Fibonacci()
9 for v in f:
10    print(v)
11    if v > 30:
12        break
```

Solution: read section

```
1 import re
2
3 class SectionReader():
4     def __init__(self, filename):
5         self.filename = filename
6         self.fh       = open(filename)
7
8     def __iter__(self):
9         return self
10
11    def __next__(self):
12        self.section = []
13        while True:
14            line = self.fh.readline()
15            if not line:
```

```

16         if self.section:
17             return self.section
18     else:
19         self.fh.close()
20         raise StopIteration
21     if re.search(r'\A\s*\Z', line):
22         if self.section:
23             return self.section
24         else:
25             continue
26     self.section.append(line)
27
28
29 filename = 'planets.txt'
30 for sec in SectionReader(filename):
31     print(sec)

```

Solution: compare files

```

1 import sys
2
3 def main():
4     if len(sys.argv) != 4:
5         exit(f"Usage: {sys.argv[0]} IN_FILE IN_FILE
OUT_FILE")
6     infile_a, infile_b = sys.argv[1:3]
7     outfile = sys.argv[3]
8
9     with open(outfile, 'w') as out_fh, open(infile_a)
as in_a, open(infile_b) as in_\
10 b:
11         cnt = 0
12         for lines in zip(in_a, in_b):
13             #print(lines)
14             lines = list(map(lambda s: s.rstrip('\n'),
lines))
15             #print(lines)
16             if lines[0] != lines[1]:
17                 out_fh.write(f"{cnt},{lines[0]},\
{lines[1]}\n")
18             cnt += 1

```

```
19
20 main()


---


1 python diff.py first.txt second.txt diff.txt
```

Solution: collect packets

The implementation

```
1 class Packets():
2     def __init__(self, filename):
3         self.filename = filename
4         self.fh = open(filename)
5         self.packets = {}
6         self.max = {}
7
8     def __iter__(self):
9         return self
10
11    def __next__(self):
12        while True:
13            line = self.fh.readline()
14            #print(f"line: {line}")
15            if line == '':
16                raise StopIteration
17
18            line = line.rstrip("\n")
19            if line == '':
20                continue
21
22            pid, seqid, maxseq, content =
23            line.split(',')
24            pid = int(pid)
25            seqid = int(seqid)
26            maxseq = int(maxseq)
27            if pid not in self.packets:
28                self.packets[pid] = {}
29                self.max[pid] = maxseq
30            if seqid in self.packets[pid]:
31                raise Exception("pid arrived twice")
32            if maxseq != self.max[pid]:
```

```
32             raise Exception("maxseq changed")
33         self.packets[pid][seqid] = content
34         if len(self.packets[pid].keys()) ==
35             self.max[pid]:
36             content = list(map(lambda i:
37                 self.packets[pid][i+1], range(self.max[\
38                     pid])))
39             del(self.max[pid])
40             del(self.packets[pid])
41             return content
```

The use:

```
1 import sys
2 from packets import Packets
3
4 if len(sys.argv) < 2:
5     exit(f"Usage: {sys.argv[0]} FILENAME")
6
7 for packet in Packets(sys.argv[1]):
8     print(packet)
```

The test to verify it

```
1 import os
2 import json
3 import pytest
4
5 from packets import Packets
6
7 root = os.path.dirname(os.path.abspath(__file__))
8
9 with open(os.path.join(root, 'packets.json')) as fh:
10     expected_results = json.load(fh)
11
12 @pytest.mark.parametrize('filename', ['packets.txt',
13 'packets1.txt', 'packets2.txt'])
13 def test_packetes(filename):
14     filepath = os.path.join(root, filename)
15
16     results = []
```

```
17     for packet in Packets(filepath):
18         results.append(packet)
19     assert results == expected_results
```

Expected result:

```
1 [[{"First of Twelve", "Second of Twelve", "Third of
Twelve", "Fourth of Twelve", "Fif\
2 th of Twelve"}, {"First of Nine", "Second of Nine",
"Third of Nine", "Fourth of Nine\
3 "}, {"First of Eleven", "Second of Eleven", "Third of
Eleven"}]]
```

Generators and Generator Expressions

Generators Glossary

- generator (a function that returns a “generator iterator”)
 - generator-iterator (an object created by a generator)
 - Generator types
 - generator-expression
-
- Generators are basically a way to create iterators without a class.

Iterators vs Generators

- a generator is an iterator
- an iterator is an iterable

```
1 from collections.abc import Iterator, Iterable
2 from types import GeneratorType
3
4 print( issubclass(GeneratorType, Iterator) )    # True
5 print( issubclass(Iterator, Iterable) )           # True
```

- Generators are a simpler way to create an iterable object than iterators, but iterators allow for more complex iterables.
- To create an iterator we need a class with two methods: `__iter__` and `__next__`, and a `raise StopIteration`.

- To create a generator we only need a single function with `yield` .

List comprehension and Generator Expression

However, before learning about `yield` let's see an even simpler way to create a generator. What we call a **generator expression**.

You are probably already familiar with list comprehensions where you have a `for` expression inside square brackets. That returns a `list` of values.

If you replace the square brackets with parentheses then you get a **generator expression**.

You can iterate over either of those. So what's the difference?

```
1 a_list = [i*2 for i in range(3)]
2 print(a_list)
3 for x in a_list:
4     print(x)
5 print()
6
7 a_generator = (i*2 for i in range(3))
8 print(a_generator)
9 for x in a_generator:
10    print(x)
```

```
1 [0, 2, 4]
2 0
3 2
4 4
5
6 <generator object <genexpr> at 0x7f0af6f97a50>
```

```
7 0  
8 2  
9 4
```

List comprehension vs Generator Expression - less memory

Let's use a bigger range of numbers and create the corresponding list and generator. Then check the size of both of them.

You can see the list is much bigger. That's because the list already contains all the elements, while the generator contains only the promise to give you all the elements.

As we could see in the previous example, this is not an empty promise, you can indeed iterate over the elements of a generator just as you can iterate over the elements of a list.

However, you cannot access an arbitrary element of a generator because the generator is not **subscriptable**.

```
1 import sys  
2  
3 lst = [n*2 for n in range(1000)] # List comprehension  
4 gen = (n*2 for n in range(1000)) # Generator  
expression  
5  
6 print(sys.getsizeof(lst))  
7 print(sys.getsizeof(gen))  
8 print()  
9  
10 print(type(lst))  
11 print(type(gen))  
12 print()  
13
```

```
14 print(lst[4])
15 print()
16
17 print(gen[4])
```

```
1 9016
2 112
3
4 <class 'list'>
5 <class 'generator'>
6
7 8
8
9 Traceback (most recent call last):
10   File "generator_expression.py", line 17, in <module>
11     print(gen[4])
12 TypeError: 'generator' object is not subscriptable
```

List Comprehension vs Generator Expressions

List comprehension vs Generator Expression - lazy evaluation

The second big difference between list comprehension and generator expressions is that the latter has lazy evaluation. In this example you can see that once we assign to list comprehension to a variable the `sqr` function is called on each element.

In the case of the generator expression, only when we iterate over the elements will Python call the `sqr` function. If we exit from the loop before we go over all the values than we saved time by not executing the expression on every element up-front. If the computation is complex and if our list is long, this can have a substantial impact.

```
1 def sqr(n):
2     print(f"sqr {n}")
3     return n ** 2
4
5 numbers = [1, 3, 7]
6
7 # list comprehension
8 n1 = [sqr(n) for n in numbers]
9 print("we have the list")
10 for i in n1:
11     print(i)
12 print("-----")
13
14 # generator expression
15 n2 = (sqr(n) for n in numbers)
16 print("we have the generator")
17 for i in n2:
18     print(i)
```

```
1 sqr 1
2 sqr 3
3 sqr 7
4 we have the list
5 1
6 9
7 49
8 -----
9 we have the generator
10 sqr 1
11 1
12 sqr 3
13 9
14 sqr 7
15 49
```

Generator: function with yield - call next

We can create a function that has multiple `yield` expressions inside.

We call the function and what we get back is a generator.

A generator is also an iterator so we can call the `next` function on it and it will give us the next `yield` value.

If we call it one too many times we get a `StopIteration` exception.

```
1 def number():
2     yield 42
3     yield 19
4     yield 23
5
6 num = number()
7 print(type(num))
8 print(next(num))
9 print(next(num))
10 print(next(num))
11 print(next(num))
```

```
1 <class 'generator'>
2 42
3 19
4 23
5 Traceback (most recent call last):
6   File "simple_generator_next.py", line 11, in <module>
7       print(next(num))
8 StopIteration
```

Generators - call `next`

We can also use a `for` loop on the generator and then we don't need to worry about the exception.

```
1 def number():
2     yield 42
3     yield 19
4     yield 23
5
6 num = number()
7 print(type(num))
8 for n in num:
9     print(n)
```

```
1 <class 'generator'>
2 42
3 19
4 23
```

Generator with yield

We don't even need to use a temporary variable for it.

```
1 def number():
2     yield 42
3     yield 19
4     yield 23
5
6 for n in number():
7     print(n)
```

```
1 42
2 19
3 23
```

Generators - fixed counter

```
1 def counter():
2     n = 1
3     yield n
4
5     n += 1
6     yield n
7
8     n += 1
9     yield n
10
11 for c in counter():
12     print(c)
```

```
1 1
2 2
3 3
```

Generators - counter

```
1 def counter():
2     n = 1
3     while True:
4         yield n
5         n += 1
6
7 for c in counter():
8     print(c)
9     if c >= 10:
10        break
```

```
1 1
2 2
3 3
4 4
5 5
6 6
7 7
8 8
9 9
10 10
```

Generators - counter with parameter

```
1 def counter(n = 1):
2     while True:
3         yield n
4         n += 1
5
6 for c in counter():
7     print(c)
8     if c >= 4:
9         break
10 print()
11
12 for c in counter(8):
13     print(c)
14     if c >= 12:
15         break
```

```
1 1
2 2
3 3
4 4
5
6 8
7 9
8 10
9 11
10 12
```

Generators - my_range

```
1 import sys
2
3 def my_range(limit = 1):
4     n = 0
5     while n < limit:
6         yield n
7         n += 1
8
9 for i in my_range(5):
10    print(i)
```

```
11 print()
12
13 print(sum(my_range(10)))
14 print()
15
16 x = my_range(10000)
17 print(x)
18 print(sys.getsizeof(x))
```

```
1 0
2 1
3 2
4 3
5 4
6
7 45
8
9 <generator object my_range at 0x7f36f6089930>
10 120
```

Fibonacci - generator

```
1 def fibonacci():
2     a, b = 0, 1
3     while True:
4         a, b = b, a+b
5         yield a
6
7 for a in fibonacci():
8     print(a)
9     if a % 17 == 0:
10        print('found')
11        break
12
13    if a > 200:
14        print('not found')
15        break
```

The fibonacci() function is called 5 times. When it reached the ‘yield’ command it returns the value as if it was a normal return call, but when the function is called again, it will be executed starting from the next statement. Hence the word ‘after’ will be printed after each call.

Infinite series

- The Fibonacci was already infinite, let’s see a few more.

Integers

```
1 from series import integers
2
3 for i in integers():
4     print(i)
5     if i >= 10:
6         break
```

```
1 1
2 2
3 3
4 4
5 5
6 6
7 7
8 8
9 9
10 10
```

Integers + 3

```
1 from series import integers
2
3 n3 = (n+3 for n in integers())
4 # n3 = integers(3)
5 for i in n3:
6     print(i)
7     if i >= 10:
8         break
```

```
1 4
2 5
3 6
4 7
5 8
6 9
7 10
```

Integers + Integers

```
1 from series import integers
2
3 def mysum(nums):
4     print(nums)
5     total = 0
6     for n in nums:
7         total += n
8     return total
9
10 n3 = integers(3)
11 n7 = integers(7)
12 d = (mysum(p) for p in zip(n3, n7))
13
14 print("start")
15 for i in d:
16     print(i)
17     if i >= 20:
18         break
```

```
1 start
2 (3, 7)
```

```
3 10
4 (4, 8)
5 12
6 (5, 9)
7 14
8 (6, 10)
9 16
10 (7, 11)
11 18
12 (8, 12)
13 20
```

Filtered Fibonacci

```
1 from series import fibonacci
2
3 even = ( fib for fib in fibonacci() if fib % 2 == 0 )
4 for e in even:
5     print(e)
6     if e > 40:
7         break
```

```
1 0
2 2
3 8
4 34
5 144
```

The series.py

This is the module behind the previous examples.

```
1 def integers(n = 1):
2     while True:
3         yield n
4         n += 1
5
6 def fibonacci():
7     a, b = 0, 1
8     while True:
```

```

9         yield a
10        a, b = b, a+b
11
12
13 def gfibonacci(size = 2):
14     """Generalized Fibonacci. """
15     values = [0]
16     while True:
17         yield values[-1]
18         if len(values) < size:
19             values.append(1)
20         else:
21             values.append(sum(values))
22             values = values[1:]
23
24 def pascal():
25     values = [1]
26     while True:
27         yield values
28         new = [1]
29         for i in range(0, len(values)-1):
30             new.append(values[i] + values[i+1])
31         new.append(1)
32         values = new

```

generator - unbound count (with yield)

```

1 def count(start=0, step=1):
2     n = start
3     while True:
4         yield n
5         n += step
6
7
8 for c in count(start=19, step=1):
9     print(c)
10    if c > 23:
11        break

```

```

1 19
2 20
3 21

```

```
4 22
5 23
6 24
```

iterator - cycle

```
1 def cycle(values=[]):
2     my_values = []
3     for v in values:
4         my_values.append(v)
5         yield v
6     while True:
7         for v in my_values:
8             yield v
9
10 i = 0
11 for c in cycle(['A', 'B', 'C']):
12     print(c)
13     i += 1
14     if i >= 4:
15         break
```

```
1 A
2 B
3 C
4 A
```

Exercise: Alternator

Create a generator for the following number series: 1, -2, 3, -4, 5, -6, ...

Exercise: Prime number generator

Create a generator that will return the prime numbers:
2, 3, 5, 7, 11, 13, 17, ...

Exercise: generator

Take the two generator examples (increment number and Fibonacci) and change them to provide infinite iterations. Then try to run them in a for loop. Just make sure you have some other condition to leave the for-loop.

Exercise: Tower of Hanoi

There are 3 sticks. On the first stick there are n rings of different sizes. The smaller the ring the higher it is on the stick.

Move over all the rings to the 3rd stick by always moving only one ring and making sure that never will there be a large ring on top of a smaller ring.

- [Tower of Hanoi](#)

Exercise: Binary file reader

Create a generator that given a filename and a number n will return the content of the file in chunks of n characters.

Exercise: File reader with records

In a file we have “records” of data. Each record starts with three bytes in which we have the length of the record. Then the content.

1 8 ABCDEFGH 5 XYZQR

Given this source file

```
1 First line
2 Second record
3 Third row of the records
4 Fourth
5 5
6 END
```

using this code

```
1 filename = "rows.txt"
2 records = "records.txt"
3
4 with open(filename) as in_fh:
5     with open(records, 'w') as out_fh:
6         for line in in_fh:
7             line = line.rstrip("\n")
8             out_fh.write("{}{:>3}{}".format(len(line),
line))
```

we can create this file:

```
1 10First line 13Second record 24Third row of the
records 6Fourth 15 3END
```

The exercise is to create an iterator/generator that can read such a file record-by-record.

Logging

Simple logging

```
1 import logging
2
3 logging.debug("debug")
4 logging.info("info")
5 logging.warning("warning")
6 logging.error("error")
7 logging.critical("critical")
8
9 logging.log(logging.WARNING, "another warning")
10 logging.log(40, "another error")
```

```
1 WARNING:root:warning
2 ERROR:root:error
3 CRITICAL:root:critical
4 WARNING:root:another warning
5 ERROR:root:another error
```

- Written on STDERR

Simple logging - set level

```
1 import logging
2
3 logging.basicConfig(level = logging.INFO)
4
5 logging.debug("debug")
6 logging.info("info")
7 logging.warning("warning")
8 logging.error("error")
9 logging.critical("critical")
```

```
1 INFO:root:info
2 WARNING:root:warning
3 ERROR:root:error
4 CRITICAL:root:critical
```

Simple logging to a file

```
1 import logging
2 import time
3
4 logging.basicConfig(level = logging.INFO, filename =
time.strftime("my-%Y-%m-%d.log")\
5 )
6
7 logging.debug("debug")
8 logging.info("info")
9 logging.warning("warning")
10 logging.error("error")
11 logging.critical("critical")
```

Simple logging format

```
1 import logging
2
3 logging.basicConfig( format = '%(asctime)s  %
%(levelname)-10s %(processName)s  %(name\
4 )s %(message)s')
5
6 logging.debug("debug")
7 logging.info("info")
8 logging.warning("warning")
9 logging.error("error")
10 logging.critical("critical")
```

Simple logging change date format

```
1 import logging
2
3 logging.basicConfig( format = '%(asctime)s  %
```

```
(levelname)-10s %(processName)s  %(name\
4 )s %(message)s', datefmt = "%Y-%m-%d-%H-%M-%S")  
5  
6 logging.debug("debug")  
7 logging.info("info")  
8 logging.warning("warning")  
9 logging.error("error")  
10 logging.critical("critical")
```

```
1 2020-04-22-18-59-16  WARNING      MainProcess  root  
warning  
2 2020-04-22-18-59-16  ERROR        MainProcess  root error  
3 2020-04-22-18-59-16  CRITICAL    MainProcess  root  
critical
```

getLogger

```
1 import logging  
2  
3 logger = logging.getLogger(__name__)
4 logger.setLevel(logging.DEBUG)
5  
6 fh = logging.FileHandler('my.log')
7 fh.setLevel(logging.INFO)
8 fh.setFormatter(logging.Formatter('%(asctime)s - %
(name)s - %(levelname)-10s - %(me\
9 ssage)s'))
10 logger.addHandler(fh)
11  
12  
13 sh = logging.StreamHandler()
14 sh.setLevel(logging.DEBUG)
15 sh.setFormatter(logging.Formatter('%(asctime)s - %
(levelname)-10s - %(message)s'))
16 logger.addHandler(sh)
17  
18  
19  
20 log = logging.getLogger(__name__)
21 log.debug("debug")
22 log.info("info")
23 log.warning("warning")
```

```
24 log.error("error")
25 log.critical("critical")
```

Time-based logrotation

```
1 import logging
2
3 log_file = "my.log"
4
5 logger = logging.getLogger(__name__)
6 logger.setLevel(logging.DEBUG)
7
8 ch =
logging.handlers.TimedRotatingFileHandler(log_file,
when='M', backupCount=2)
9 ch.setLevel(logging.INFO)
10 ch.setFormatter(logging.Formatter('%(asctime)s - %
%(name)s - %(levelname)-10s - %(me\
11 ssage)s'))
12 logger.addHandler(ch)
13
14
15 log = logging.getLogger(__name__)
16 log.debug("debug")
17 log.info("info")
18 log.warning("warning")
19 log.error("error")
20 log.critical("critical")
```

Size-based logrotation

```
1 import logging
2
3 log_file = "my.log"
4
5 logger = logging.getLogger(__name__)
6 logger.setLevel(logging.DEBUG)
7
8 ch = logging.handlers.RotatingFileHandler(log_file,
maxBytes=100, backupCount=2)
9 ch.setLevel(logging.INFO)
```

```
10 ch.setFormatter( logging.Formatter('%(asctime)s - %  
11 %(name)s - %(levelname)-10s - %(me\  
12 ssage)s') )  
13 logger.addHandler(ch)  
14  
15 log = logging.getLogger(__name__)  
16 log.debug("debug")  
17 log.info("info")  
18 log.warning("warning")  
19 log.error("error")  
20 log.critical("critical")
```

Closures

Counter local - not working

```
1 def counter():
2     count = 0
3     count += 1
4     return count
5
6 print(counter())
7 print(counter())
8 print(counter())
```

```
1 1
2 1
3 1
```

Counter with global

```
1 count = 0
2 def counter():
3     global count
4     count += 1
5     return count
6
7 print(counter())
8 print(counter())
9 print(counter())
10
11 count = -42
12 print(counter())
```

```
1 1
2 2
```

```
3 3
4 -41
```

Create incrementors

In order to use in various map-expressions, we need a couple of functions that - for simplicity - need to increment a number:

```
1 def f3(x):
2     return x + 3
3
4 def f7(x):
5     return x + 7
6
7 def f23(x):
8     return x + 23
9
10 print(f3(2))
11 print(f7(3))
12 print(f3(4))
13 print(f7(10))
14 print(f23(19))
```

```
1 5
2 10
3 7
4 17
5 42
```

Create internal function

```
1 def create_func():
2     def internal():
3         print("Hello world")
4     internal()
5
6
7 func = create_func()
8 internal()
```

```
1 Hello world
2 Traceback (most recent call last):
3   File "create_internal_func.py", line 8, in <module>
4     internal()
5 NameError: name 'internal' is not defined
```

Create function by a function

```
1 def create_func():
2     def internal():
3         print("Hello world")
4     #internal()
5
6     return internal
7
8 func = create_func()
9 #internal()
10 func()
```

```
1 Hello world
```

Create function with parameters

```
1 def create_func(name):
2     def internal():
3         print(f"Hello {name}")
4
5     return internal
6
7 foo = create_func("Foo")
8 foo()
9
10
11 bar = create_func("Bar")
12 bar()
```

```
1 Hello Foo
2 Hello Bar
```

Counter closure

```
1 def create_counter():
2     count = 0
3     def internal():
4         nonlocal count
5         count += 1
6         return count
7     return internal
8
9 counter = create_counter()
10
11 print(counter())
12 print(counter())
13 print(counter())
14 print()
15
16 other = create_counter()
17 print(counter())
18 print(other())
19 print(counter())
20 print(other())
21
22 print()
23 print(count)
```

```
1 1
2 2
3 3
4
5 4
6 1
7 5
8 2
9
10 Traceback (most recent call last):
11   File "counter.py", line 23, in <module>
12     print(count)
13 NameError: name 'count' is not defined
```

Make incrementor with def (closure)

```
1 def make_incremator(n):
2     def inc(x):
3         return x + n
4     return inc
5
6 f3 = make_incremator(3)
7 f7 = make_incremator(7)
8
9 print(f3(2))
10 print(f7(3))
11 print(f3(4))
12 print(f7(10))
```

```
1 5
2 10
3 7
4 17
```

Make incrementor with lambda

```
1 def make_incremator(n):
2     return lambda x: x + n
3
4 f3 = make_incremator(3)
5 f7 = make_incremator(7)
6
7 print(f3(2))
8 print(f7(3))
9 print(f3(4))
10 print(f7(10))
```

```
1 5
2 10
3 7
4 17
```

Exercise: closure bank

- Create a closure that returns a function that holds a number (like a bank account) that can be incremented or decremented as follows:
 - Allow for an extra parameter called `prev` that defaults to `False`. If `True` is passed then instead of returning the new balance, return the old balance.
-

```
1 bank = create_bank(20)
2
3 print(bank())      # 20
4 print(bank(7))    # 27
5 print(bank())      # 27
6 print(bank(-3))   # 24
7 print(bank())      # 24
8
9
10 print(bank(10, prev=True))   # 24
11 print(bank())      # 34
```

Exercise: counter with parameter

Change the counter example to accept a parameter and start counting from that number.

Solution: closure bank

```
1 def create_bank(n = 0):
2     balance = n
3     def bnk(change = 0, prev=False):
4         nonlocal balance
5         prev_balance = balance
6         balance += change
7         if prev:
8             return prev_balance
9         else:
10            return balance
11     return bnk
12
13
```

```
14 bank = create_bank(20)
15
16 print(bank())      # 20
17 print(bank(7))    # 27
18 print(bank())      # 27
19 print(bank(-3))   # 24
20 print(bank())      # 24
21
22
23 print(bank(10, prev=True))  # 24
24 print(bank())      # 34
```

```
1 20
2 27
3 27
4 24
5 24
6 24
7 34
```

Solution: counter with parameter

```
1 def create_counter(count=0):
2     def internal():
3         nonlocal count
4         count += 1
5         return count
6     return internal
7
8 counter = create_counter()
9
10 print(counter())
11 print(counter())
12 print(counter())
13 print()
14
15 other = create_counter(42)
16 print(counter())
17 print(other())
18 print(counter())
19 print(other())
```

1 1

2 2

3 3

4

5 4

6 43

7 5

8 44

Decorators

Function assignment

Before we learn about decorators let's remember that we can assign function names to other names and then use the new name:

```
1 say = print
2 say("Hello World")
3
4 print = lambda n: n**n
5 res = print(3)
6 say("Hi")
7 say(res)
8
9
10 def add(x, y):
11     return x + y
12
13 combine = add
14
15 say( combine(2, 3) )
```

```
1 Hello World
2 Hi
3 27
4 5
```

Function inside other function

Let's also remember that we can define a function inside another function
and then the internally defined function only exists in the scope
of the function
where it was defined in. Not outside.

```
1 def f():
2     def g():
3         print("in g")
4     print("start f")
5     g()
6     print("end f")
7
8 f()
9 g()
```

```
1 start f
2 in g
3 end f
4 Traceback (most recent call last):
5   File "examples/decorators/function_in_function.py",
line 9, in <module>
6     g()
7 NameError: name 'g' is not defined
```

Decorator

- A function that changes the behaviour of other functions.
- The input of a decorator is a function.
- The returned value of a decorator is a modified version of the same function.

```
1 from some_module import some_decorator
2
3 @some_decorator
4 def f(...):
5     ...
```

```
1 def f(...):
2     ...
3
4 f = some_decorator(f)
```

Use cases for decorators in Python

- Common decorators are classmethod() and staticmethod().
- Flask uses them to mark and configure the routes.
- Pytest uses them to add marks to the tests.
- Logging calls with parameters.
- Logging elapsed time of calls.
- Access control in Django or other web frameworks. (e.g. login required)
- Memoization (caching)
- Retry
- Function timeout
- Locking for thread safety
- [Decorator Library](#)

A recursive Fibonacci

```
1 def fibo(n):
2     if n in (1,2):
3         return 1
4     return fibo(n-1) + fibo(n-2)
5
6 print(fibo(5)) # 5
```

trace fibo

```
1 import decor
2
3 @decor.tron
4 def fibo(n):
5     if n in (1,2):
6         return 1
7     return fibo(n-1) + fibo(n-2)
8
9 print(fibo(5))
```

```
1 Calling fibo(5)
2 Calling fibo(4)
3 Calling fibo(3)
4 Calling fibo(2)
5 Calling fibo(1)
6 Calling fibo(2)
7 Calling fibo(3)
8 Calling fibo(2)
9 Calling fibo(1)
10 5
```

tron decorator

```
1 def tron(func):
2     def new_func(v):
3         print("Calling {}({})".format(func.__name__, v))
4         return func(v)
5     return new_func
```

Decorate with direct call

```
1 import decor
2
3 def fibo(n):
4     if n in (1,2):
5         return 1
6     return fibo(n-1) + fibo(n-2)
7
```

```
8 fibo = decor.tron(fibo)
9
10 print(fibo(5))
```

Decorate with parameter

```
1 import decor_param
2
3 @decor_param.tron('foo')
4 def fibo(n):
5     if n in (1, 2):
6         return 1
7     return fibo(n-1) + fibo(n-2)
8
9 print(fibo(5))
```

```
1 foo Calling fibo(5)
2 foo Calling fibo(4)
3 foo Calling fibo(3)
4 foo Calling fibo(2)
5 foo Calling fibo(1)
6 foo Calling fibo(2)
7 foo Calling fibo(3)
8 foo Calling fibo(2)
9 foo Calling fibo(1)
10 5
```

Decorator accepting parameter

```
1 def tron(prefix):
2     def real_tron(func):
3         def new_func(v):
4             print("{} Calling {}({})".format(prefix,
5                 func.__name__, v))
6             return func(v)
7         return new_func
8     return real_tron
```

Decorate function with any signature

- How can we decorate a function that is flexible on the number of arguments?
- Accept *args and **kwargs and pass them on.

```
1 from decor_any import tron
2
3
4 @tron
5 def one(param):
6     print(f"one({param})")
7
8 @tron
9 def two(first, second = 42):
10    print(f"two({first}, {second})")
11
12
13 one("hello")
14 one(param = "world")
15
16 two("hi")
17 two(first = "Foo", second = "Bar")
```

Decorate function with any signature - implementation

```
1 def tron(func):
2     def new_func(*args, **kw):
3         params = list(map(lambda p: str(p), args))
4         for (k, v) in kw.items():
5             params.append(f"{k}={v}")
6         print(f"Calling {func.__name__}({params})")
7         return func(*args, **kw)
8     return new_func
```

```
1 Calling one(hello)
2 one(hello)
```

```
3 Calling one(param=world)
4 one(world)
5 Calling two(hi)
6 two(hi, 42)
7 Calling two(first=Foo, second=Bar)
8 two(Foo, Bar)
```

Exercise: Logger decorator

- In the previous pages we created a decorator that can decorate arbitrary function logging the call and its parameters.
- Add time measurement to each call to see how long each function took.

Exercise: memoize decorator

Write a function that gets a functions as attribute and returns a new functions while memoizing (caching) the input/output pairs. Then write a unit test that checks it.

You probably will need to create a subroutine to be memoized.

- Write tests for the fibonacci functions.
- Implement the memoize decorator for a function with a single parameter.
- Apply the decorator.
- Run the tests again.
- Check the speed differences.
- or decorate with tron to see the calls...

Solution: Logger decorator

```
1 import time
2 def tron(func):
```

```

3     def new_func(*args, **kwargs):
4         start = time.time()
5         print("Calling {}({},"
6             .format(func.__name__, args, kwargs))
6         out = func(*args, **kwargs)
7         end = time.time()
8         print("Finished {}({})".format(func.__name__,
out))
9         print("Elapsed time: {}".format(end - start))
10        return out
11    return new_func

```

Solution: Logger decorator (testing)

```

1 from logger_decor import tron
2
3 @tron
4 def f(a, b=1, *args, **kwargs):
5     print('a:', a)
6     print('b:', b)
7     print('args:', args)
8     print('kwargs:', kwargs)
9     return a + b
10
11 f(2, 3, 4, 5, c=6, d=7)
12 print()
13 f(2, c=5, d=6)
14 print()
15 f(10)

```

```

1 Calling f((2, 3, 4, 5), {'c': 6, 'd': 7})
2 a:      2
3 b:      3
4 args:   (4, 5)
5 kwargs: {'c': 6, 'd': 7}
6 Finished f(5)
7 Elapsed time: 1.3589859008789062e-05
8
9 Calling f((2,), {'c': 5, 'd': 6})
10 a:      2
11 b:      1

```

```
12 args:    ()
13 kwargs: {'c': 5, 'd': 6}
14 Finished f(3)
15 Elapsed time: 5.245208740234375e-06
16
17 Calling f((10,), {})
18 a:      10
19 b:      1
20 args:   ()
21 kwargs: {}
22 Finished f(11)
23 Elapsed time: 4.291534423828125e-06
```

Solution memoize decorator

```
1 import sys
2 import memoize_attribute
3 import memoize_nonlocal
4 import decor_any
5
6 #@memoize_attribute.memoize
7 #@memoize_nonlocal.memoize
8 #@decor_any.tron
9 def fibonacci(n):
10     if n == 1:
11         return 1
12     if n == 2:
13         return 1
14     return fibonacci(n-1) + fibonacci(n-2)
15
16 if __name__ == '__main__':
17     if len(sys.argv) != 2:
18         sys.stderr.write("Usage: {}"
N\n".format(sys.argv[0]))
19         exit(1)
20     print(fibonacci(int(sys.argv[1])))
```

```
1 def memoize(f):
2     data = {}
3     def caching(n):
4         nonlocal data
```

```
5         key = n
6         if key not in data:
7             data[key] = f(n)
8         return data[key]
9
10    return caching
```

```
1 def memoize(f):
2     def caching(n):
3         key = n
4         #if 'data' not in caching.__dict__:
5         #    caching.data = {}
6         if key not in caching.data:
7             caching.data[key] = f(n)
8         return caching.data[key]
9         caching.data = {}
10
11    return caching
```

Before

```
1 $ time python fibonacci.py 35
2 9227465
3
4 real    0m3.850s
5 user    0m3.832s
6 sys     0m0.015s
```

After

```
1 $ time python fibonacci.py 35
2 9227465
3
4 real    0m0.034s
5 user    0m0.019s
6 sys     0m0.014s
```

Context managers (with statement)

Why use context managers?

In certain operations you might want to ensure that when the operation is done there will be an opportunity to clean up after it. Even if decided to end the operation early or if there is an exception in the middle of the operation.

In the following pseudo-code example you can see that `cleanup` must be called both at the end and before the `early-end`, but that still leaves the bad-code that raises exception avoiding the cleanup. That forces us to wrap the whole section in a try-block.

```
1 start
2 do
3 do
4 do
5 do
6 cleanup
```

What is we have some conditions for early termination?

```
1 start
2 do
3 do
4 if we are done early:
5   cleanup
6   early-end
7 do
8 do
9 cleanup
```

What if we might have an exception in the code?

```
1 start
2 try:
3     do
4     do
5     if we are done early:
6         cleanup
7         early-end
8     do
9     bad-code      (raises exception)
10    do
11    cleanup
12 finally:
13     cleanup
```

It is a lot of unnecessary code duplication and we can easily forget to add it in every location where we early-end our code.

Context Manager examples

A few examples where context managers can be useful:

- Opening a file - close it once we are done with it so we don't leak file descriptors.
- Changing directory - change back when we are done.
- Create temporary directory - remove when we are done.
- Open connection to database - close connection.
- Open SSH connection - close connection.
- More information about [context managers](#)

cd in a function

In this example we have a function in which we change to a directory and then when we are done we change back to the original directory.

For this to work first we save the current working directory using the `os.getcwd` call. Unfortunately in the middle of the code there

is a conditional call to `return`. If that condition is `True` we won't change back to the original directory. We could fix this by calling `os.chdir(start_dir)` just before calling `return`.

However this would still not solve the problem if there is an exception
in the function.

```
1 import sys
2 import os
3
4 def do_something(path):
5     start_dir = os.getcwd()
6     os.chdir(path)
7
8     content = os.listdir()
9     number = len(content)
10    print(number)
11    if number < 15:
12        return
13
14    os.chdir(start_dir)
15
16 def main():
17     if len(sys.argv) != 2:
18         exit(f"Usage: {sys.argv[0]} PATH")
19     path = sys.argv[1]
20     print(os.getcwd())
21     do_something(path)
22     print(os.getcwd())
23
24 main()
```

```
1 $ python no_context_cd.py /tmp/
2
3 /home/gabor/work/slides/python-
programming/examples/advanced
4 19
5 /home/gabor/work/slides/python-
programming/examples/advanced
```

```
1 $ python no_context_cd.py /opt/
2
3 /home/gabor/work/slides/python-
programming/examples/advanced
4 9
5 /opt
```

- In the second example `return` was called and thus we stayed on the `/opt` directory.:w

open in function

This is not the recommended way to open a file, but this is how it was done before the introduction of the `with` context manager. Here we have the same issue. We have a conditional call to `return` where we forgot to close the file.

```
1 import sys
2 import re
3
4 def do_something(filename):
5     fh = open(filename)
6
7     while True:
8         line = fh.readline()
9         if line is None:
10             break
```

```
11         line = line.rstrip("\n")
12
13     if re.search(r'\A\s*\Z', line):
14         return
15     print(line)
16
17 fh.close()
18
19 def main():
20     if len(sys.argv) != 2:
21         exit(f"Usage: {sys.argv[0]} FILENAME")
22     filename = sys.argv[1]
23     do_something(filename)
24
25 main()
```

open in for loop

Calling `write` does not immediately write to disk. The Operating System provides buffering as an optimization to avoid frequent access to the disk. In this case it means the file has not been saved before we already check its size.

```
1 import os
2
3 for ix in range(10):
4     filename = f'data{ix}.txt'
5     fh = open(filename, 'w')
6     fh.write('hello')
7     if ix == 0:
8         break
9     fh.close()
10 stat = os.stat(filename)
11 print(stat.st_size)    # 0,    the file has not been
                           saved yet
```

open in function using with

If we open the file in the recommended way using the `with` statement then we can be sure that the `close` method of the `fh` object will be called when we leave the context of the `with` statement.

```
1 import sys
2 import re
3
4 def do_something(filename):
5     with open(filename) as fh:
6
7         while True:
8             line = fh.readline()
9             if line is None:
10                 break
11             line = line.rstrip("\n")
12
13             if re.search(r'\A\s*\Z', line):
14                 return
15             print(line)
16
17
18 def main():
19     if len(sys.argv) != 2:
20         exit(f"Usage: {sys.argv[0]} FILENAME")
21     filename = sys.argv[1]
22     do_something(filename)
23
24 main()
```

Plain context manager

```
1 from contextlib import contextmanager
2
3 @contextmanager
```

```
4 def my_plain_context():
5     print("start context")
6     yield
7     print("end context")
8
9 print("START")
10 with my_plain_context():
11     print("  In plain context")
12     print("  More work")
13
14 print("END")
```

```
1 START
2 start context
3   In plain context
4   More work
5 end context
6 END
```

Param context manager

```
1 from contextlib import contextmanager
2
3 @contextmanager
4 def my_param_context(name):
5     print(f"start {name}")
6     yield
7     print(f"end {name}")
8
9 with my_param_context("foo"):
10    print("In param context")
```

```
1 start foo
2 In param context
3 end foo
```

Context manager that returns a value

```
 1 from contextlib import contextmanager
 2
 3 import time
 4 import random
 5 import os
 6 import shutil
 7
 8
 9 @contextmanager
10 def my_tempdir():
11     print("start return")
12     tmpdir = '/tmp/' + str(time.time()) +
13         str(random.random())
14     os.mkdir(tmpdir)
15     try:
16         yield tmpdir
17     finally:
18         shutil.rmtree(tmpdir)
19         print("end return")
```

```
 1 import os
 2 from my_tempdir import my_tempdir
 3
 4 with my_tempdir() as tmp_dir:
 5     print(f"In return context with {tmp_dir}")
 6     with open(tmp_dir + '/data.txt', 'w') as fh:
 7         fh.write("hello")
 8     print(os.listdir(tmp_dir))
 9
10 print('')
11 print(tmp_dir)
12 print(os.path.exists(tmp_dir))
```

```
 1 start return
 2 In return context with
 3 /tmp/1578211890.49409370.6063140788762365
 4 ['data.txt']
 5 end return
 6
 7 /tmp/1578211890.49409370.6063140788762365
 8 False
```

Use my tempdir - return

```
1 import os
2 from my_tempdir import my_tempdir
3
4 def some_code():
5     with my_tempdir() as tmp_dir:
6         print(f"In return context with {tmp_dir}")
7         with open(tmp_dir + '/data.txt', 'w') as fh:
8             fh.write("hello")
9         print(os.listdir(tmp_dir))
10    return
11
12 print('')
13 print(tmp_dir)
14 print(os.path.exists(tmp_dir))
15
16 some_code()
```

```
1 start return
2 In return context with
3 /tmp/1578211902.3545020.7667694368935928
4 end return
```

Use my tempdir - exception

```
1 import os
2 from my_tempdir import my_tempdir
3
4 with my_tempdir() as tmp_dir:
5     print(f"In return context with {tmp_dir}")
6     with open(tmp_dir + '/data.txt', 'w') as fh:
7         fh.write("hello")
8     print(os.listdir(tmp_dir))
9     raise Exception('trouble')
10
11 print('')
12 print(tmp_dir)
13 print(os.path.exists(tmp_dir))
```

```
1 start return
2 In return context with
/tmpp/1578211921.12552210.9000097350821897
3 ['data.txt']
4 end return
5 Traceback (most recent call last):
6   File "use_my_tempdir_exception.py", line 9, in
<module>
7     raise Exception('trouble')
8 Exception: trouble
```

cwd context manager

```
1 import os
2 from contextlib import contextmanager
3
4 @contextmanager
5 def cwd(path):
6     oldpwd = os.getcwd()
7     os.chdir(path)
8     try:
9         yield
10    finally:
11        os.chdir(oldpwd)
```

```
1 import sys
2 import os
3 from mycwd import cwd
4
5 def do_something(path):
6     with cwd(path):
7         content = os.listdir()
8         if len(content) < 10:
9             return
10
11 def main():
12     if len(sys.argv) != 2:
13         exit(f"Usage: {sys.argv[0]} PATH")
14     path = sys.argv[1]
15     print(os.getcwd())
16     do_something(path)
```

```
17     print(os.getcwd())
18
19 main()
```

```
1 $ python context_cd.py /tmp
2 /home/gabor/work/slides/python/examples/advanced
3 /home/gabor/work/slides/python/examples/advanced
4
5 $ python context_cd.py /opt
6 /home/gabor/work/slides/python/examples/advanced
7 /home/gabor/work/slides/python/examples/advanced
```

tempdir context manager

```
1 import os
2 from contextlib import contextmanager
3 import tempfile
4 import shutil
5
6 @contextmanager
7 def tmpdir():
8     dd = tempfile.mkdtemp()
9     try:
10         yield dd
11     finally:
12         shutil.rmtree(dd)
```

```
1 from mytmpdir import tmpdir
2 import os
3
4 with tmpdir() as temp_dir:
5     print(temp_dir)
6     with open( os.path.join(temp_dir, 'some.txt'),
7 'w') as fh:
8         fh.write("hello")
9     print(os.path.exists(temp_dir))
10    print(os.listdir(temp_dir))
11 print(os.path.exists(temp_dir))
```

```
1 /tmp/tmpprpuywa3_
2 True
3 ['some.txt']
4 False
```

Context manager with class

```
1 class MyCM:
2     def __init__(self, name):
3         self.name = name
4
5     def __enter__(self):
6         print(f'__enter__ {self.name}')
7         return self
8
9     def __exit__(self, exception_type, exception,
traceback):
10        print(f'__exit__ {self.name}')
11
12    def something(self):
13        print(f'something {self.name}')
14
15 def main():
16     with MyCM('Foo') as cm:
17         print(cm.name)
18         cm.something()
19         #raise Exception('nono')
20     print('in main - after')
21
22 main()
23 print('after main')
```

Context managers with class

Even if there was an exception in the middle of the process,
the **exit** methods of each object will be called.

```
1 class MyCM:
2     def __init__(self, n):
3         self.name = n
4
```

```

5      def __enter__(self):
6          print('__enter__', self.name)
7
8      def __exit__(self, exception_type, exception,
9                  traceback):
10         print('__exit__', self.name)
11
12     def something(self):
13         print('something', self.name)
14
15 def main():
16     a = MyCM('a')
17     b = MyCM('b')
18     with a, b:
19         a.partner = b
20         b.partner = a
21         a.something()
22         raise Exception('nono')
23         b.something()
24     print('in main - after')
25
26 main()
27 print('after main')

```

```

1 __enter__ a
2 __enter__ b
3 something a
4 __exit__ b
5 __exit__ a
6 Traceback (most recent call last):
7   File "context-managers.py", line 27, in <module>
8     main()
9   File "context-managers.py", line 23, in main
10    raise Exception('nono')
11 Exception: nono

```

Context manager: with for file

```

1 import sys
2
3 if len(sys.argv) != 2:
4     sys.stderr.write('Usage: {}'

```

```
FILENAME\n'.format(sys.argv[0]))  
    exit()  
file = sys.argv[1]  
print(file)  
with open(file) as f:  
    for line in f:  
        val = 30/int(line)  
print('done')
```

With - context managers

```
class WithClass:  
    def __init__(self, name='default'):  
        self.name = name  
  
    def __enter__(self):  
        print('entering the system')  
        return self.name  
  
    def __exit__(self, exc_type, exc_value,  
traceback):  
        print('exiting the system')  
  
    def __str__(self):  
        return 'WithObject:' + self.name  
  
x = WithClass()  
with x as y:  
    print(x,y)
```

Exercise: Context manager

Create a few CSV file likes these:

```
a11,a12  
a21,a22
```

```
1 b13,b14
2 b23,b24
```

```
1 c15,c16
2 c25,c26
```

Merge them horizontally to get this:

```
1 a11,a12,b13,b14,c15,c16
2 a21,a22,b23,b24,c25,c26
```

- Do it without your own context manager
- Create a context manager called myopen that accepts N filenames. It opens the first one to write and the other N-1 to read

```
1 with myopen(outfile, infile1, infile2, infile3) as out,
ins:
2     ...
```

Exercise: Tempdir on Windows

Make the tempdir context manager example work on windows as well. Probably need to cd out of the directory.

Solution: Context manager

```
1 import sys
2 from contextlib import contextmanager
3
4 if len(sys.argv) < 3:
5     exit(f"Usage: {sys.argv[0]} OUTFILE INFILeS")
6
7 outfile = sys.argv[1]
8 infiles = sys.argv[2:]
```

```
 9 #print(outfile)
10 #print(infiles)
11
12 @contextmanager
13 def myopen(outfile, *infiles):
14     #print(len(infiles))
15     out = open(outfile, 'w')
16     ins = []
17     for filename in infiles:
18         ins.append(open(filename, 'r'))
19     try:
20         yield out, ins
21     except Exception as ex:
22         print(ex)
23         pass
24     finally:
25         out.close()
26         for fh in ins:
27             fh.close()
28
29
30 with myopen(outfile, *infiles) as (out_fh, input_fhs):
31     #print(out_fh.__class__.__name__)
32     #print(len(input_fhs))
33     while True:
34         row = ''
35         done = False
36         for infh in input_fhs:
37             line = infh.readline()
38             #print(f'{line}')
39             if not line:
40                 done = True
41                 break
42             if row:
43                 row += ','
44             row += line.rstrip("\n")
45         if done:
46             break
47         out_fh.write(row)
48         out_fh.write("\n")
```

Advanced lists

Change list while looping: endless list

```
1 numbers = [1, 1]
2 for n in numbers:
3     print(n)
4     numbers.append(numbers[-1] + numbers[-2])
5
6     if n > 100:
7         break
8
9 print(numbers)
```

Creating a Fibonacci series in a crazy way.

Change list while looping

Probably not a good idea...

```
1 numbers = [1, 2, 3, 4]
2 for n in numbers:
3     print(n)
4     if n == 2:
5         numbers.remove(2)
6
7
8 print(numbers)
```

```
1 1
2 2
3 4
4 [1, 3, 4]
```

Note, the loop only iterated 3 times, and it skipped value 3

Copy list before iteration

It is better to copy the list using list slices before the iteration starts.

```
1 numbers = [1, 2, 3, 4]
2 for n in numbers[:]:
3     print(n)
4     if n == 2:
5         numbers.remove(2)
6
7
8 print(numbers)
```

```
1 1
2 2
3 3
4 4
5 [1, 3, 4]
```

for with flag

```
1 names = ['Foo', 'Bar', 'Baz']
2
3 ok = False
4 for i in range(3):
5     name = input('Your name please: ')
6     if name in names:
7         ok = True
8         break
9
10 if not ok:
11     print("Not OK")
12     exit()
13
14 print("OK....")
```

for else

The else statement of the for loop is executed when the iteration ends normally. (without calling break)

```
1 names = ['Foo', 'Bar', 'Baz']
2
3
4 for i in range(3):
5     name = input('Your name please: ')
6     if name in names:
7         break
8 else:
9     print("Not OK")
10    exit()
11
12 print("OK....")
```

enumerate

```
1 names = ['Foo', 'Bar', 'Baz']
2
3 for i in range(len(names)):
4     print(i, names[i])
5
6 print('')
7
8 for i, n in enumerate(names):
9     print(i, n)
```

```
1 0 Foo
2 1 Bar
3 2 Baz
4
5 0 Foo
6 1 Bar
7 2 Baz
```

do while

There is no do-while in Python, but you can emulate it:

```
1 while True:  
2     do_stuff()  
3     if not loop_condition():  
4         break
```

```
1 x = 0  
2  
3 while True:  
4     x += 1  
5     print(x)  
6     if x > 0:  
7         break
```

list slice is copy

```
1 x = [1, 1, 2, 3, 5, 8, 13, 21, 34]  
2 y = x[2:5]  
3 print(y)      # [2, 3, 5]  
4  
5 x[2] = 20  
6 print(x)      # [1, 1, 20, 3, 5, 8, 13, 21, 34]  
7 print(y)      # [2, 3, 5]
```

Advanced Exception handling

Exceptions else

- The else part will be execute after each successful “try”. (So when there was no exception.)

```
1 import sys
2 import module
3
4 # python else.py one.txt zero.txt two.txt three.txt
5 files = sys.argv[1:]
6
7 for filename in files:
8     try:
9         module.read_and_divide(filename)
10    except ZeroDivisionError as err:
11        print("Exception {} of type {} in file
{}.".format(err, type(err).__name__, f\
ilename))
12    else:
13        print("In else part after trying file {} and
succeeding".format(filename))
14        # Will run only if there was no exception.
15    print()
```

```
1 before one.txt
2 100.0
3 after one.txt
4 In else part after trying file one.txt and succeeding
5
6 before zero.txt
7 Exception division by zero of type ZeroDivisionError
in file zero.txt
8
9 before two.txt
10 Traceback (most recent call last):
```

```
11     File "else.py", line 9, in <module>
12         module.read_and_divide(filename)
13     File "/home/gabor/work/slides/python-
programming/examples/exceptions/module.py", 1\
14 ine 3, in read_and_divide
15         with open(filename, 'r') as fh:
16 FileNotFoundError: [Errno 2] No such file or
directory: 'two.txt'
```

Exceptions finally

- We can add a “finally” section to the end of the “try” - “except” construct.
- The code in this block will be executed after **every** time we enter the **try**.
- When we finish it successfully. When we catch an exception. (In this case a ZeroDivisionError exception in file zero.txt)
- Even when we don’t catch an exception. Before the exception propagates up in the call stack, we still see the “finally” section executed.

```
1 import sys
2 import module
3
4 # python finally.py one.txt zero.txt two.txt three.txt
5 files = sys.argv[1:]
6
7 for filename in files:
8     try:
9         module.read_and_divide(filename)
10    except ZeroDivisionError as err:
11        print("Exception {} of type {} in file
{}".format(err, type(err).__name__, f\
12 ilename))
13    finally:
14        print("In finally after trying file
{}".format(filename))
15    print('')
```

```
1 before one.txt
2 100.0
3 after one.txt
4 In finally after trying file one.txt
5
6 before zero.txt
7 Exception division by zero of type ZeroDivisionError
in file zero.txt
8 In finally after trying file zero.txt
9
10 before two.txt
11 In finally after trying file two.txt
12 Traceback (most recent call last):
13   File "finally.py", line 9, in <module>
14     module.read_and_divide(filename)
15   File "/home/gabor/work/slides/python-
programming/examples/exceptions/module.py", 1\
16 ine 3, in read_and_divide
17     with open(filename, 'r') as fh:
18 FileNotFoundError: [Errno 2] No such file or
directory: 'two.txt'
```

Exit and finally

The “finally” part will be called even if we call “return” or “exit” in the “try” block.

```
1 def f():
2     try:
3         return
4     finally:
5         print("finally in f")
6
7 def g():
8     try:
9         exit()
10    finally:
11        print("finally in g")
```

```
12
13 print("before")
14 f()
15 print("after f")
16 g()
17 print("after g")
18
19 # before
20 # finally in f
21 # after f
22 # finally in g
```

Catching exceptions

```
1 def divide(x, y):
2     return x/y
3
4 def main():
5     cnt = 6
6     for num in [2, 0, 'a']:
7         try:
8             divide(cnt, num)
9         except ZeroDivisionError:
10             pass
11         except (IOError, MemoryError) as err:
12             print(err)
13         else:
14             print("This will run if there was no
exception at all")
15         finally:
16             print("Always executes. {} / {}"
ended.".format(cnt, num))
17
18     print("done")
19
20
21 main()
```

¹ This will run if there was no exception at all

² Always executes. $6/2$ ended.

³ Always executes. $6/0$ ended.

⁴ Always executes. $6/a$ ended.

```
5 Traceback (most recent call last):
6   File "try.py", line 22, in <module>
7     main()
8   File "try.py", line 9, in main
9     divide(cnt, num)
10  File "try.py", line 3, in divide
11    return x/y
12 TypeError: unsupported operand type(s) for /: 'int'
and 'str'
```

Home made exception

You can create your own exception classes that will allow the user to know what kind of an exception was caught or to capture only the exceptions of that type.

```
1 class MyException(Exception):
2     pass
3
4 def some():
5     raise MyException("Some Error")
6
7 def main():
8     try:
9         some()
10    except Exception as err:
11        print(err)
12        print("Type: " + type(err).__name__)
13
14    try:
15        some()
16    except MyException as err:
17        print(err)
18
19 main()
```

```
1 Some Error
2 Type: MyException
3 Some Error
```

Home made exception with attributes

```
1 class MyException(Exception):
2     def __init__(self, name, address):
3         self.name = name
4         self.address = address
5     def __str__(self):
6         return 'Have you encountered problems? name:{}\naddress:{}'.format(self.name\
7 , self.address)
8
9
10 def some():
11     raise MyException(name = "Foo Bar", address =
12 "Somewhere deep in the code")
13
14 def main():
15     try:
16         some()
17     except Exception as err:
18         print(err)
19         print("Type: " + type(err).__name__)
20         print(err.name)
21         print(err.address)
22
23 main()
24 # Have you encountered problems? name:Foo Bar
25 # address:Somewhere deep in the code
26 # Type: MyException
27 # Foo Bar
28 # Somewhere deep in the code
```

Home made exception hierarchy

```
1 class MyError(Exception):
2     pass
```

```
3
4 class MyGreenError(MyError):
5     pass
6
7 class MyBlueError(MyError):
8     pass
9
10
11 def green():
12     raise MyGreenError('Hulk')
13
14 def blue():
15     raise MyBlueError('Frozen')
16
17 def red():
18     red_alert()
```

Home made exception hierarchy - 1

```
1 import colors as cl
2
3 def main():
4     print("start")
5     try:
6         cl.green()
7     except Exception as err:
8         print(err)
9         print(type(err).__name__)
10    print("done")
11
12
13 main()
```

```
1 start
2 Hulk
3 MyGreenError
4 done
```

Home made exception hierarchy - 2

```
1 import colors as cl
2
3 def main():
4     print("start")
5     try:
6         cl.green()
7     except cl.MyGreenError as err:
8         print(err)
9         print(type(err).__name__)
10    print("done")
11
12
13 main()
```

```
1 start
2 Hulk
3 MyGreenError
4 done
```

Home made exception hierarchy - 3

```
1 import colors as cl
2
3 def main():
4     print("start")
5
6     try:
7         cl.green()
8     except cl.MyError as err:
9         print(err)
10        print(type(err).__name__)
11
12     try:
13         cl.blue()
14     except cl.MyError as err:
15         print(err)
16         print(type(err).__name__)
17
18     try:
19         cl.red()
20     except cl.MyError as err:
```

```
21         print(err)
22         print(type(err).__name__)
23
24
25
26
27     print("done")
28
29
30 main()
```

```
1 start
2 Hulk
3 MyGreenError
4 Frozen
5 MyBlueError
6 Traceback (most recent call last):
7   File "hierarchy3.py", line 30, in <module>
8     main()
9   File "hierarchy3.py", line 19, in main
10    cl.red()
11   File
12   "/home/gabor/work/slides/python/examples/exceptions/colors.py", line 18, in r\
13     red_alert()
14 NameError: name 'red_alert' is not defined
```

Exercise: spacefight with exceptions

Take the number guessing game (or one-dimensional space-fight) and add exceptions for cases when the guess is out of space (0-200 by default), or when the guess is not a number.

```
1 import random
2
3 class Game:
4     def __init__(self):
```

```
5         self.lower_limit = 0
6         self.upper_limit = 200
7
8         self.number =
random.randrange(self.lower_limit, self.upper_limit)
9         self.is_debug = False
10        self.running = True
11
12    def debug(self):
13        self.is_debug = not self.is_debug
14
15    def guess(self, num):
16        if num == 'd':
17            self.debug()
18            return
19
20        if self.is_debug:
21            print("Hidden number {}. Your guess is
{}".format(self.number, num))
22
23        if num < self.number:
24            print("Too small")
25        elif num > self.number:
26            print("Too big")
27        else:
28            print("Bingo")
29            self.running = False
30
31
32 g = Game()
33 g.guess('d')
34
35 try:
36     g.guess('z')
37 except Exception as e:
38     print(e)
39
40 try:
41     g.guess('201')
42 except Exception as e:
43     print(e)
44
45 try:
46     g.guess('-1')
```

```
47 except Exception as e:  
48     print(e)
```

Exercises: Raise My Exception

This is very similar to the exercise the first chapter about exceptions, but in this case you need to create your own hierarchy of exception classes.

- Write a function that expects a positive integer as its single parameter.
- Raise exception if the parameter is not a number.
- Raise a different exception if the parameter is not positive.
- Raise a different exception if the parameter is not whole number.
- In each case make sure both the text and the type of the exceptions are different.
- Include the actual value received as an attribute in the exception object.

Solution: spacefight with exceptions

```
1 import random  
2  
3 class SpaceShipError(Exception):  
4     def __init__(self, inp):  
5         self.inp = inp  
6  
7 class NumberTooBigError(SpaceShipError):  
8     def __str__(self):  
9         return "Number {} is too big".format(self.inp)  
10  
11 class NumberTooSmallError(SpaceShipError):  
12     def __str__(self):  
13         return "Number {} is too
```

```
small".format(self.inp)
14
15
16 class NotANumberError(SpaceShipError):
17     def __str__(self):
18         return "Not a Number {}".format(self.inp)
19
20
21 class Game:
22     def __init__(self):
23         self.lower_limit = 0
24         self.upper_limit = 200
25
26         self.number =
random.randrange(self.lower_limit, self.upper_limit)
27         self.is_debug = False
28         self.running = True
29
30     def debug(self):
31         self.is_debug = not self.is_debug
32
33     def guess(self, num):
34         if num == 'd':
35             self.debug()
36             return
37
38         if self.is_debug:
39             print("Hidden number {}. Your guess is
{}".format(self.number, num))
40
41         try:
42             num = int(num)
43         except Exception:
44             raise NotANumberError(num)
45
46         if num > self.upper_limit:
47             raise NumberTooBigError(num)
48
49         if num < self.upper_limit:
50             raise NumberTooSmallError(num)
51
52         if num < self.number:
53             print("Too small")
54         elif num > self.number:
```

```

55         print("Too big")
56     else:
57         print("Bingo")
58         self.running = False
59
60
61 g = Game()
62 g.guess('d')
63
64 try:
65     g.guess('z')
66 except Exception as e:
67     print(e)
68
69 try:
70     g.guess('201')
71 except Exception as e:
72     print(e)
73
74 try:
75     g.guess('-1')
76 except Exception as e:
77     print(e)
78
79
80
81 #while g.running:
82 #    guess = input("Please type in your guess: ")
83 #    g.guess(int(guess))

```

```

1 Hidden number 137. Your guess is z
2 Not a Number z
3 Hidden number 137. Your guess is 201
4 Number 201 is too big
5 Hidden number 137. Your guess is -1
6 Number -1 is too small

```

Solution: Raise My Exception

```

1 class MyValueError(ValueError):
2     def __init__(self, val):
3         self.value = val

```

```
4
5 class MyFloatError(MyValueError):
6     def __str__(self):
7         return "The given parameter {} is a float and
not an int.".format(self.value)
8
9 class MyTypeError(MyValueError):
10    def __init__(self, val, val_type):
11        self.value_type = val_type
12        super(MyTypeError, self).__init__(val)
13
14    def __str__(self):
15        return "The given parameter {} is of type {}"
and not int.".format(self.value,\n16 self.value_type)
17
18 class MyNegativeError(MyValueError):
19     def __str__(self):
20         return "The given number {} is not
positive.".format(self.value)
21
22 def positive(num):
23     if type(num).__name__ == 'float':
24         raise MyFloatError(num)
25
26     if type(num).__name__ != 'int':
27         raise MyTypeError(num, type(num).__name__)
28
29     if num < 0:
30         raise MyNegativeError(num)
31
32 for val in [14, 24.3, "hi", -10]:
33     print(val)
34     print(type(val).__name__)
35     try:
36         positive(val)
37     except MyValueError as ex:
38         print("Exception: {}".format(ex))
39         print("Exception type\n{}".format(type(ex).__name__))
40
41     # Exception, ValueError
```

Exception finally return

```
1 def div(a, b):
2     try:
3         print("try")
4         c = a / b
5     except Exception:
6         print("exception")
7         return
8     finally:
9         print("finally")
10
11 div(2, 1)
12 print('---')
13 div(2, 0)
```

Warnings

Warnings

```
1 from warnings import warn
2
3 def foo():
4     warn("foo will be deprecated soon. Use bar()
instead", DeprecationWarning)
5     print("foo still works")
6
7
8 def main():
9     foo()
10    print("afterfoo")
11
12 main()
```

CSV

Reading CSV the naive way

```
1 Tudor;Vidor;10;Hapci
2 Szundi;Morgo;7;Szende
3 Kuka;Hofeherke;100;Kiralyno
4 Boszorkany;Herceg;9;Meselo
```

```
1 import sys, csv
2
3 if len(sys.argv) != 2:
4     sys.stderr.write("Usage: {}"
FILENAME\n".format(sys.argv[0]))
5     exit()
6
7 file = sys.argv[1]
8 fh = open(file, 'rb')
9
10 count = 0
11 for line in fh:
12     line = line.rstrip("\n")
13     row = line.split(';')
14     print(row)
15     count += int(row[2])
16
17 print("Total: {}".format(count))
```

```
python examples/csv/read_csv_split.py
examples/csv/process_csv_file.csv
```

CSV with quotes and newlines

```
1 Tudor;Vidor;10;Hapci
2 Szundi;Morgo;7;Szende
```

```
3 Kuka;"Hofeherke; alma";100;Kiralyno  
4 Boszorkany;Herceg;9;Meselo
```

```
1 Tudor;Vidor;10;Hapci  
2 Szundi;Morgo;7;Szende  
3 Kuka;"Hofeherke;  
4 alma";100;Kiralyno  
5 Boszorkany;Herceg;9;Meselo
```

Reading a CSV file

```
1 import sys, csv  
2  
3 if len(sys.argv) != 2:  
4     sys.stderr.write("Usage: {}  
FILENAME\n".format(sys.argv[0]))  
5     exit()  
6  
7 file = sys.argv[1]  
8 count = 0  
9 with open(file) as fh:    # Python 2 might need 'rb'  
10    rd = csv.reader(fh, delimiter=';')  
11  
12    for row in rd:  
13        print(row)  
14        count += int(row[2])  
15  
16 print("Total: {}".format(count))
```

python examples/csv/read_csv.py
examples/csv/process_csv_file.csv

Dialects of CSV files. See also:

[CSV](#)

CSV dialects

```
1 import csv
2
3 for dname in csv.list_dialects():
4     print(dname)
5     d = csv.get_dialect(dname)
6     for n in ['delimiter', 'doublequote',
7               'escapechar',
8               'lineterminator', 'quotechar',
9               'quoting', 'skipinitialspace', 'strict']:
10        attr = getattr(d, n)
11        if attr == '\t':
12            attr = '\\t'
13        if attr == '\r\n':
14            attr = '\\r\\n'
15        print("  {:16} {}".format(n, attr))
```

```
1 excel-tab
2   delimiter      '\t'
3   doublequote    '1'
4   escapechar     'None'
5   lineterminator '\r\n'
6   quotechar      ''
7   quoting         '0'
8   skipinitialspace '0'
9   strict          '0'
10 excel
11   delimiter      ','
12   doublequote    '1'
13   escapechar     'None'
14   lineterminator '\r\n'
15   quotechar      ''
16   quoting         '0'
17   skipinitialspace '0'
18   strict          '0'
```

CSV to dictionary

```
1 fname, lname, born
2 Graham, Chapman, 8 January 1941
3 Eric, Idle, 29 March 1943
4 Terry, Gilliam, 22 November 1940
```

```
5 Terry, Jones, 1 February 1942
6 John, Cleese, 27 October 1939
7 Michael, Palin, 5 May 1943
```

```
1 import csv
2
3 file = 'examples/csv/monty_python.csv'
4 with open(file) as fh:
5     rd = csv.DictReader(fh, delimiter=',')
6     for row in rd:
7         print(row)
```

```
1 {'lname': 'Chapman', 'born': '8 January 1941', 'fname':
'Graham'}
2 {'lname': 'Idle', 'born': '29 March 1943', 'fname':
'Eric'}
3 {'lname': 'Gilliam', 'born': '22 November 1940',
'fname': 'Terry'}
4 {'lname': 'Jones', 'born': '1 February 1942', 'fname':
'Terry'}
5 {'lname': 'Cleese', 'born': '27 October 1939', 'fname':
'John'}
6 {'lname': 'Palin', 'born': '5 May 1943', 'fname':
'Michael'}
```

Exercise: CSV

Given the CSV file of Monty Python troupe, create a dictionary where we can look up information about them based on the first name. For example:

```
1 people = read_csv_file()
2 print(people["Graham"]["lname"])    # Champman
3 print(people["John"]["born"])        # 27 October 1939
4 print(people["Michael"])
5     # {'lname': 'Palin', 'born': '5 May 1943',
'fname': 'Michael'}
6 print(people["Terry"]["lname"])    # Gilliam
```

For extra bonus create another dictionary where we can look up the information based on their fname and lname.

Solution: CSV

```
1 import csv
2
3 def read_csv_file():
4     file = 'examples/csv/monty_python.csv'
5     name_of = {}
6     with open(file) as fh:
7         rd = csv.DictReader(fh, delimiter=',')
8         for row in rd:
9             name_of[ row['fname'] ] = row
10    print(name_of)
11    return name_of
12
13 people = read_csv_file()
14 print(people["Graham"]["lname"]) # Chapman
15 print(people["John"]["born"]) # 27 October 1939
16 print(people["Michael"])
17     # {'lname': 'Palin', 'born': '5 May 1943',
18     # 'fname': 'Michael'}
19 print(people["Terry"]["lname"]) # Gilliam
```

Excel

Spreadsheets

- CSV files - use the standard csv library
- Microsoft Excel files (various versions and formats)
- Open Office / Libre Office Calc

Python Excel

- [Python Excel](#)
- [openpyxl](#)
- [xlsxwriter](#)
- [xlrd](#)
- [xlwt](#)
- [xlutils](#) using xlrd and xlwt. Mostly obsolete.

Create an Excel file from scratch

```
1 import openpyxl
2 import datetime
3
4 wb = openpyxl.Workbook()
5
6 ws = wb.active
7
8 ws['A1'] = 42
9
10 ws['A2'] = datetime.datetime.now()
11 #ws.column_dimensions['A'].width = 20.0
12
13 wb.save("first.xlsx")
```

Worksheets in Excel

```
1 import openpyxl
2 import datetime
3
4 wb = openpyxl.Workbook()
5 ws = wb.active
6 ws['A1'] = 42
7 ws.title = "First"
8
9 ws2 = wb.create_sheet()
10 ws2.title = "Second sheet"
11 ws2['A1'] = datetime.datetime.now()
12 ws2.sheet_properties.tabColor = "1072BA"
13
14 wb.save("two_worksheets.xlsx")
```

Add expressions to Excel

Nothing special needed.

```
1 import openpyxl
2 import datetime
3
4 wb = openpyxl.Workbook()
5
6 ws = wb.active
7
8 ws['A1'] = 19
9 ws['A2'] = 23
10
11 ws['A3'] = "=A1+A2"
12
13 wb.save("expression.xlsx")
```

Format field

```
1 import openpyxl
2 import datetime
3
```

```

4 wb = openpyxl.Workbook()
5
6 ws = wb.active
7
8 ws['A1'] = 123456.78
9 ws['A2'] = 123456.78
10 ws['A3'] = 123456.78
11 ws['A4'] = -123456.78
12 ws['A5'] = datetime.datetime.now()
13 ws.column_dimensions['A'].width = 20.0
14
15 ws['A2'].number_format = '0.00E+00'
16 ws['A3'].number_format = '#,##0_);[RED](#,##0)'
17 ws['A4'].number_format = '#,##0_);[RED](#,##0)'
18
19 wb.save("format.xlsx")

```

Number series and chart

```

1 import openpyxl
2
3 wb = openpyxl.Workbook()
4
5 ws = wb.active
6 ws.title = "Chart"
7
8 a = ["First", 20, 28, 30, 37, 18, 47]
9 b = ["Second", 35, 30, 40, 40, 38, 35]
10
11 # write them as columns
12 for i in range(len(a)):
13     ws.cell(row=i+1, column=1).value = a[i]
14     ws.cell(row=i+1, column=2).value = b[i]
15
16 lc = openpyxl.chart.LineChart()
17 lc.title = "Two Lines Chart"
18 #lc.style=13
19 data = openpyxl.chart.Reference(ws,
20                                     min_col=1,
21                                     min_row=1,
22                                     max_col=2,
23                                     max_row=len(a))
24 lc.add_data(data, titles_from_data=True)

```

```
25  
26 ws.add_chart(lc, "D1")  
27 wb.save("chart.xlsx")
```

Read Excel file

```
1 import openpyxl  
2 wb = openpyxl.load_workbook(filename = 'chart.xlsx')  
3 for ws in wb.worksheets:  
4     print(ws.title)  
5  
6 ws = wb.worksheets[0]  
7 print(ws['A1'].value)
```

Update Excel file

```
1 import openpyxl  
2  
3 wb = openpyxl.load_workbook(filename = 'chart.xlsx')  
4 for ws in wb.worksheets:  
5     print(ws.title)  
6  
7 ws = wb.worksheets[0]  
8 c = ["Third", 40, 20, 35, 25, 20, 35]  
9  
10 for i in range(len(c)):  
11     ws.cell(row=i+1, column=3).value = c[i]  
12  
13 lc = openpyxl.chart.LineChart()  
14 lc.title = "Three Lines Chart"  
15 data = openpyxl.chart.Reference(ws,  
16                                 min_col=1,  
17                                 min_row=1,  
18                                 max_col=3,  
19                                 max_row=len(c))  
20 lc.add_data(data, titles_from_data=True)  
21  
22 ws.add_chart(lc, "D1")  
23  
24 wb.save("chart.xlsx")
```

Exercise: Excel

- Create a series of 10 random numbers between 1 and 100 and save them in an Excel file in a column.
- Create a graph showing the values.
- Add a second series of 10 random numbers, add them to the Excel file as a second column next to the first one.
- Add a 3rd column containing the average of the first two columns.
- Update the graph to include all 3 number serieses

XML

XML Data

```
1 <?xml version="1.0"?>
2 <main>
3   <person id="1">
4     <fname>Foo</fname>
5     <lname>Bar</lname>
6   </person>
7   <person id="3">
8     <fname>Moo</fname>
9     <lname>Zorg</lname>
10    <email id="home">moo@zorghome.com</email>
11    <email id="work">moo@work.com</email>
12  </person>
13 </main>
```

Expat - Callbacks

```
1 import xml.parsers.expat
2
3 file = 'examples/xml/data.xml'
4
5
6 def start_element(name, attrs):
7     print('Start element: {} {}'.format(name, attrs))
8
9
10 def end_element(name):
11     print('End element: {}'.format(name))
12
13
14 def char_data(data):
15     print('Character data: {}'.format(repr(data)))
16
```

```
17
18 p = xml.parsers.expat.ParserCreate()
19
20 p.StartElementHandler = start_element
21 p.EndElementHandler = end_element
22 p.CharacterDataHandler = char_data
23
24 p.ParseFile(open(file, 'rb'))
25
26 print('done')
```

XML DOM - Document Object Model

```
1 import xml.dom.minidom
2
3 file = 'examples/xml/data.xml'
4
5 dom = xml.dom.minidom.parse(file)
6
7 root = dom.firstChild
8 print(root.tagName)
9
10 print('')
11
12 for node in root.childNodes:
13     if node.nodeType != node.TEXT_NODE:
14         print('name: ', node.tagName)
15         print('id: ', node.getAttribute('id'))
16
17 print('')
18
19 emails = dom.getElementsByTagName("email")
20 for e in emails:
21     print('email', e.getAttribute('id'),
e.firstChild.data)
```

```
1 main
2
3 name: person
4 id: 1
5 name: person
6 id: 3
```

```
7  
8 email home moo@zorghome.com  
9 email work moo@work.com
```

- [xml.dom](#)
- [xml.dom.minidom](#)

XML SAX - Simple API for XML

```
1 import xml.sax  
2  
3 file = 'examples/xml/data.xml'  
4  
5  
6 class EventHandler(xml.sax.ContentHandler):  
7     def startElement(self, name, attrs):  
8         print('start', (name, attrs._attrs))  
9  
10    def characters(self, text):  
11        if not text.isspace():  
12            print('text', text)  
13  
14    def endElement(self, name):  
15        print('end', name)  
16  
17  
18 xml.sax.parse(file, EventHandler())
```

```
1 start (u'main', {})  
2 start (u'person', {u'id': u'1'})  
3 start (u'fname', {})  
4 text Foo  
5 end fname  
6 start (u'lname', {})  
7 text Bar  
8 end lname  
9 end person  
10 start (u'person', {u'id': u'3'})  
11 start (u'fname', {})  
12 text Moo
```

```

13 end fname
14 start (u'lname', {})
15 text Zorg
16 end lname
17 start (u'email', {u'id': u'home'})
18 text moo@zorghome.com
19 end email
20 start (u'email', {u'id': u'work'})
21 text moo@work.com
22 end email
23 end person
24 end main

```

- [xml.sax](#)
- [xml.sax.handler](#)
- [xml.sax.reader](#)

SAX collect

```

1 import xml.sax
2
3 file = 'examples/xml/data.xml'
4
5 class EventHandler(xml.sax.ContentHandler):
6     def __init__(self, c):
7         self.path = []
8         self.collector = c
9
10    def startElement(self, name, attrs):
11        self.path.append({ 'name' : name, 'attr' :
12 attrs._attrs })
13
14    def characters(self, text):
15        self.path[-1]['text'] = text
16
17    def endElement(self, name):
18        element = self.path.pop()
19        print('End name: ', name)
20        if element['name'] == 'email':
21            collector.append(element)

```

```
22 collector = []
23 xml.sax.parse(file, EventHandler(collector))
24 print(collector)
```

```
1 End name: fname
2 End name: lname
3 End name: person
4 End name: fname
5 End name: lname
6 End name: email
7 End name: email
8 End name: person
9 End name: main
10 [{"text": u'moo@zorghome.com', 'name': u'email',
11   'attr': {u'id': u'home'}},
12  {"text": u'moo@work.com', 'name': u'email', 'attr':
13    {u'id': u'work'}}]
```

XML elementtree

```
1 import xml.etree.ElementTree as ET
2
3 file = 'examples/xml/data.xml'
4
5 tree = ET.parse(file)
6 root = tree.getroot()
7 print(root.tag)
8
9 for p in root.iter('person'):
10     print(p.attrib)
11
12 print('')
13
14 for p in root.iter('email'):
15     print(p.attrib, p.text)
16
17 print('')
18
19 elements = tree.findall('.//*[@id=\'home\'])')
20 for e in elements:
21     print(e.tag, e.attrib)
```

```
1 main
2 {'id': '1'}
3 {'id': '3'}
4
5 {'id': 'home'} moo@zorghome.com
6 {'id': 'work'} moo@work.com
7
8 email {'id': 'home'}
```

- [xml.etree.ElementTree](#)

SciPy - for Scientific Computing in Python

Data Science tools in Python

- [SciPy](#) ecosystem of open-source software for mathematics, science, and engineering.
- [Biopython](#) tools for biological computation.
- [NumPy](#) to handle N-dimensional arrays.
- [Pandas](#) Python Data Analysis Library. (Data Frames)
- [Matplotlib](#) a 2D plotting library.
- [Seaborn](#) data visualization library based on matplotlib.
- [Bokeh](#) interactive visualization library.
- [SciKit-Learn](#) Machine Learning in Python.
- [TensorFlow](#) Machine learning framework. (developed by Google engineer)
- [Keras](#) Python Deep learning (neural-network) library. (On top of Tensorflow.)
- [Orange](#) machine learning and data visualization tool. Written partially in Python.
- [Airflow](#) Workflow management platform
- [Luigi](#) Data pipelines (from Spotify)
- [Showing speed improvement using a GPU with CUDA and Python with numpy on Nvidia Quadro 2000D](#)
- [Octave](#) (Open Source Matlab replacement - not related to Python)

Data Analysis resources

- [Exploratory data analysis](#) by John Tukey
- [Think Bayes - Bayesian Statistics Made Simple](#)
- [Statistical Signal Extraction and Filtering: Structural Time Series Models](#)
- [Panel Data](#)

For Econometrics

- [Econometric Analysis](#)
- [Microeconometric Modeling and Discrete Choice Analysis with Cross Section and Panel Data](#)

For Intro Stats,

- [Applied Statistics with R](#)
- [Statistics: A Fresh Approach](#)

Datasets

- [Climate](#)
- [Open Weather map](#)
- [PRB](#)

Python and Biology

Biopython

- [Biopython](#)
- [Biopython GitHub project](#)
- [Biopython Tutorial and Cookbook](#)

Biopython background

- [Sequence formats](#) (FASTA, FASTQ, EMBL, ...)
- [FASTA](#)
- [FASTQ](#)
- [EMBL](#) European Molecular Biology Laboratory
- [Gene names symbols](#)

Bio python sequences

```
1 from Bio.Seq import Seq
2
3 # Nucleotide Sequences
4 my_dna = Seq("AGTACACTGGTAGGCCTTACAG_T")
5 print(my_dna)                                #
AGTACACTGGTAGGCCTTACAG_T
6 print(my_dna.complement())                   #
TCATGTGACCATCCGGAATGTC_A
7 print(my_dna.reverse_complement())      #
A_CTGTAAGGCCCTACCAAGTGTACT
8 print(my_dna.transcribe())                  #
AGUACACUGGUAGGCCUUACAG_U
9
10 my_rna = Seq("GAC_U")
11 print(my_rna)                            # GAC_U
```

```
12 print(my_rna.reverse_complement())    # A_GUC
13 print(my_rna.reverse_complement())    # A_GUC
14 print(my_rna.transcribe())           # GAC_U
```

```
1 from Bio.Seq import Seq
2
3 what_is_this = Seq("AGTC_U")
4 what_is_this.complement()  # ValueError: Mixed RNA/DNA
found
```

Download data

Use the NCBI (National Center for Biotechnology Information) database to search manually for [nucleotide](#) or tons of other types of data. Then one can download the files manually from the web site.

Read FASTA, GenBank files

For example the data about Orchids in two formats:

- [ls orchid.fasta](#) in FASTA format
- [ls orchid.gbk](#) in GenBank format

Download those files and use them:

```
1 from Bio import SeqIO
2 import requests
3
4 def get_file(url, filename):
5     res = requests.get(url)
6     if res.status_code != 200:
7         raise Exception("Could not get file")
8
9     with open(filename, 'w') as fh:
10         fh.write(res.text)
11
```

```
12
13 def process_file(filename, file_type):
14     for seq_record in SeqIO.parse(filename,
file_type):
15         print(seq_record.id)
16         print(repr(seq_record.seq))
17         print(len(seq_record))
18
19
20 fasta_url =
'https://raw.githubusercontent.com/biopython/biopython/master/Doc/examples/ls_orchid.fasta'
21 filename = "ls_orchid.fasta"
22 file_type = "fasta"
23 get_file(fasta_url, filename)
24 process_file(filename, file_type)
25
26
27
28 genbank_url =
"https://raw.githubusercontent.com/biopython/biopython/master/Doc/examples/ls_orchid.gbk"
29 filename = "ls_orchid.gbk"
30 file_type = "genbank"
31 get_file(genbank_url, filename)
32 process_file(filename, file_type)
```

Search nucleotids

You can also search the same database programmatically.

```
1 from Bio import Entrez
2 Entrez.email = "gabor@szabgab.com"
3
4 term = "Cypripedioideae[Orgn] AND matK[Gene]"
5
6 handle = Entrez.esearch(db="nucleotide", term=term,
idtype="acc", retmax=30)
7 record = Entrez.read(handle)
8 print(record["Count"])          # 538
9 print(record["IdList"])        # ['MK792700.1',
'MK792699.1', 'MK792698.1', ..., 'MK79\
```

```
10 2681.1']
11 print(len(record["IdList"])) # 30
12 handle.close()
13
14
15 # term = "Orchid"
16 # 530077
17 # ['NZ_SELD00000000.2', 'NZ_SELD02000072.1',
```

Download nucleotids

```
1 from Bio import Entrez, SeqIO
2
3 Entrez.email = "gabor@szabgab.com"
4
5 #doc_id = 'MK792700.1'
6 doc_id = "EU490707"
7
8 # rettype="fasta"
9 handle = Entrez.efetch(db="nucleotide", id=doc_id,
rettype="gb", retmode="text")
10 data = handle.read()
11 handle.close()
12 #print(data)
13
14 filename = "temp.data"
15 with open(filename, 'w') as fh:
16     fh.write(data)
17
18 file_type = "genbank"
19 for seq_record in SeqIO.parse(filename, file_type):
20     print(seq_record.id)
21     print(repr(seq_record.seq)) # A short part of the
sequence
22     print()
23     print(seq_record.seq) # The full sequence
24     print()
25     print(len(seq_record.seq))
26     print()
27     print(seq_record.name)
28     print()
29     print(seq_record.annotations)
```

```
30     #print()  
31     #print(dir(seq_record))
```

Exercise: Nucleotid

- Search for your favorite nucleotid
- Print out the number of results
- Download the 3 different sequences from the list (using the id) in GeneBank format and save them in files using the id as the name of the file and .gb as the extension
- Write a separate script that reads and displays the sequences.

Biology background

- [Genetics - inheritance](#)
- [Genetic inheritance](#)
- [What's a genome Chp2 1](#)
- [What's a genome Chp4 1](#)
- alleles, genotype, phenotype

Chemistry

Chemistry links

- [Python for Chemistry students](#)
- [Open Babel](#) The Open Source Chemistry Toolbox
- [Chemical table file](#) to describe molecules and chemical reactions.
- [Pytim](#) Interfacial Analysis of Molecular Simulations
- [Awesome Python Chemistry](#) (article)
- [Awesome Python Chemistry](#) (list on GitHub)
- [downloads](#)
- [Open Babel module](#)
- [Pybel](#)

```
1 import sdf
2 import pybel
```

Bond length

- [Bond length](#)
- Distance between two points [Pythagorean theorem](#)
- [Video](#)
- [XYZ fileformat](#) to specify the molecule geometry.

Covalent radius

- [Covalent radius](#)
- [Video](#)
- [tmpchem/computational_chemistry](#)

Python energy landscape explorer

- [Python energy landscape explorer](#)

Other chemistry links

- [Periodic table](#)
- [Diatomc molecule](#)
- [VMD - Visual Molecular Dynamics](#) and application to visualize molecules.

numpy

What is NumPy

- [numpy](#)
- High-level mathematical functions to operate on large, multi-dimensional arrays and matrices. **ndarray**

Numpy - vector

```
1 import numpy as np
2
3 a = np.array([3, 4, 7])
4 print(a)          # [3 4 7]
5 print(a * 3)      # [ 9 12 21]
6 print(a + 4)      # [ 7  8 11]
7 print(a.dtype)    # int64
8 print(a.ndim)     # 1
9 print(a.shape)    # (3,)
10
11 b = np.array([2, 3.14, -1])
12 print(b.dtype)   # float64
13 print(b.shape)   # (3,)
14
15 c = np.array(['one', 'two', 'three'])
16 print(c.dtype)   # <U5      (Unicode less than 5
characters)
```

- [Basic types](#)
- [dtypes](#)

NumPy 2D arrays

```
1 import numpy as np
2
3 a = np.array([
4     [ 1, 2, 3, 4, 5],
5     [ 2, 3, 4, 5, 6]
6 ])
7
8 print(a)
9 # [[1 2 3 4 5]
10 #  [2 3 4 5 6]]
11
12 print(a.shape) # (2, 5)
13 print(a.ndim) # 2
14
15
16 print(a * 3)
17 # [[ 3  6  9 12 15]
18 #  [ 6  9 12 15 18]]
19
20 print(a + 7)
21 # [[ 8  9 10 11 12]
22 #  [ 9 10 11 12 13]]
```

Numpy - set type

```
1 import numpy as np
2
3 a = np.array([3, 4, 7], dtype='int8')
4 print(a) # [3 4 7]
5 print(a * 3) # [ 9 12 21]
6 print(a + 4) # [ 7  8 11]
7 print(a.dtype) # int8
```

NumPy arrays: ones and zeros

```
1 import numpy as np
2
3 c = np.ones(4, dtype='int32')
4 print(c) # [1 1 1 1]
5 print(c.dtype) # int32
```

```
6 print(c.shape)      # (4, )
7 print()
8
9
10 d = np.zeros(3, dtype='float32')
11 print(d)           # [ 0.  0.  0.]
12 print(d.dtype)     # float32
13 print(d.shape)     # (3,)
14 print()
15
16
17 a = np.ones([2, 3])
18 print(a)
19 # [[1., 1., 1.],
20 # [1., 1., 1.]]
21 print(a.dtype)     # float64
22 print(a.shape)     # (2, 3)
```

Numpy: eye

```
1 import numpy as np
2
3 a = np.eye(4)
4 print(a)
5 print()
6
7 b = np.eye(3, 5)
8 print(b)
```

```
1 [[1. 0. 0. 0.]
2 [0. 1. 0. 0.]
3 [0. 0. 1. 0.]
4 [0. 0. 0. 1.]]
5
6 [[1. 0. 0. 0. 0.]
7 [0. 1. 0. 0. 0.]
8 [0. 0. 1. 0. 0.]]
```

NumPy array random

```
1 import numpy as np
2
3 a = np.random.random((2, 5))    # in the range [0.0, 1.0)
4 print(a)
5 print()
6
7 rng = np.random.default_rng()
8 b = rng.random(size=(3, 4))
9 print(b)
```

```
1 [[0.32151126 0.07688622 0.95666894 0.42396291
0.93592235]
2 [0.71406863 0.95152079 0.20199695 0.72628099
0.33545885]]
3
4 [[0.46643834 0.71350899 0.40279583 0.85148985]
5 [0.19367868 0.53288449 0.97181597 0.86311691]
6 [0.70687485 0.78534671 0.16654183 0.9371896 ]]
```

- random sampling

NumPy Random integers

```
1 import numpy as np
2
3 a = np.random.randint(10, size=(3, 4))
4 print(a)
5
6 rng = np.random.default_rng()
7 b = rng.integers(42, size=(3, 4))
8 print(b)
```

```
1 [[1 2 2 6]
2 [2 2 9 8]
3 [8 8 9 5]]
4 [[13 31 7 11]
5 [22 2 6 18]
6 [24 10 12 0]]
```

- [integer generator](#)

NumPy array type change by division (int to float)

```
1 import numpy as np
2
3 a = np.array([3, 4, 7])
4 print(a.dtype) # int64
5 print(a.shape) # (3,)
6
7 x = (a / 2)
8 print(x) # [ 1.5  2.   3.5]
9 print(x.dtype) # float64
10 print(x.shape) # (3,)
```

Numpy: Array methods: transpose

```
1 import numpy
2
3 a = numpy.array([
4     [ 1, 2, 3, 4, 5],
5     [ 2, 3, 4, 5, 6]
6 ])
7
8 b = a.transpose()
9
10 print(b)
11 # [[1 2]
12 #  [2 3]
13 #  [3 4]
14 #  [4 5]
15 #  [5 6]]
16
17 print(a)
18 # [[1 2 3 4 5]
19 #  [2 3 4 5 6]]
```

Numpy: reference, not copy

```
1 import numpy
2
3 a = numpy.array([
4     [ 1, 2, 3, 4, 5],
5     [ 2, 3, 4, 5, 6]
6 ])
7
8 b = a.transpose()
9 a[0][0] = 42
10
11 print(b)
12 # [[42 2]
13 #  [2 3]
14 #  [3 4]
15 #  [4 5]
16 #  [5 6]]
17
18 print(a)
19 # [[42 2 3 4 5]
20 #  [2 3 4 5 6]]
```

Numpy: copy array

```
1 import numpy
2
3 a = numpy.array([
4     [ 1, 2, 3, 4, 5],
5     [ 2, 3, 4, 5, 6]
6 ])
7
8 b = a.copy().transpose()
9 a[0][0] = 42
10
11 print(b)
12 # [[1 2]
13 #  [2 3]
14 #  [3 4]
15 #  [4 5]
16 #  [5 6]]
17
18 print(a)
```

```
19 # [[42 2 3 4 5]
20 # [2 3 4 5 6]]
```

Numpy: Elementwise Operations on Arrays

```
1 import numpy as np
2
3 a = np.array([
4     [ 1, 2, 3, 4, 5],
5     [ 2, 3, 4, 5, 6]
6 ])
7 b = np.array([
8     [ 7, 3, 8, 9, 4],
9     [ 1, 3, 6, 1, 2]
10 ])
11
12 print(a+b)
13 # [[ 8  5 11 13  9]
14 # [ 3  6 10  6  8]]
15
16 print(a*b)
17 # [[ 7  6 24 36 20]
18 # [ 2  9 24  5 12]]
```

Numpy: multiply, matmul, dot for vectors

- [multiply](#)
- [matmul](#)
- [dot](#)

```
1 import numpy as np
2
3 a = np.array([3, 4, 7])
4 b = np.array([6, 5, 2])
5 print(a) # [3 4 7]
6 print(b) # [6 5 2]
7
8 c = np.multiply(a, b)
9 print(c) # [18 20 14]
```

```
10
11 d = np.dot(a, b)
12 print(d)      # 52
13
14 m = np.matmul(a, b)
15 print(m)      # 52
```

Numpy: multiply, matmul, dot for vector and matrix

```
1 import numpy as np
2
3 a = np.array([[1, 2, 3], [4, 5, 6]])
4 b = np.array([1, 2, 4])
5 print(a)
6 print(b)
7 print()
8
9 print(a*b)
10 print(b*a)
11 print()
12
13 print(np.multiply(a, b))
14
15 print()
16 print( np.dot(a, b) )
17 print( np.matmul(a, b) )
```

```
1 [[1 2 3]
2  [4 5 6]]
3 [1 2 4]
4
5 [[ 1   4 12]
6  [ 4 10 24]]
7 [[ 1   4 12]
8  [ 4 10 24]]
9
10 [[ 1   4 12]
11  [ 4 10 24]]
```

```
13 [17 38]  
14 [17 38]
```

Numpy: multiply, matmul, dot for matrices

```
1 import numpy as np  
2  
3 a = np.array([[1, 2, 3], [4, 5, 6]])  
4 b = np.array([[1, 3, 4], [7, 8, 0]])  
5 print(a)  
6 print(b)  
7 print()  
8  
9 print(a*b)  
10 print(b*a)  
11 print()  
12  
13 print(np.multiply(a, b))  
14  
15 print()  
16 print( np.dot(a, b.transpose()) )  
17 print( np.matmul(a, b.transpose()) )  
18  
19 print()  
20 print( np.dot(a.transpose(), b) )  
21 print( np.matmul(a.transpose(), b) )
```

```
1 [[1 2 3]  
2  [4 5 6]]  
3 [[1 3 4]  
4  [7 8 0]]  
5  
6 [[ 1   6 12]  
7  [28 40  0]]  
8 [[ 1   6 12]  
9  [28 40  0]]  
10  
11 [[ 1   6 12]  
12  [28 40  0]]  
13  
14 [[19 23]]
```

```
15 [43 68]
16 [[19 23]
17 [43 68]
18
19 [[29 35 4]
20 [37 46 8]
21 [45 57 12]]
22 [[29 35 4]
23 [37 46 8]
24 [45 57 12]]
```

Numpy: casting - converting from strings to integer.

```
1 import numpy as np
2
3 a = np.array([
4     ["12", "23", "3", "4"],
5     ["2", "3", "4", "5"]
6 ])
7
8 print(a)
9 #[['12' '23' '3' '4']
10 # ['2' '3' '4' '5']]
11
12 try:
13     b = a + 1
14 except Exception as e:
15     print(e)
16 # TypeError: ufunc 'add' did not contain a loop with
17 #      signature matching types dtype('<U3')
18 #      dtype('<U3') dtype('<U3')
19
20 c = a.astype(np.int) + 1
21 print(c)
22 # [[13 24 4 5]
23 # [ 3 4 5 6]]
```

Numpy: indexing 1d array

```
1 import numpy as np
2
3 a = np.array([1, 1, 2, 3, 5, 8, 13, 21, 34])
4 print(a)          # [ 1  1  2  3  5  8 13 21 34]
5
6 print(a[4])      # 5
7 print(a[2:5])    # [2 3 5]
```

Numpy: slice is a reference

The slice in numpy does not copy the data structure

```
1 import numpy as np
2
3 a = np.array([1, 1, 2, 3, 5, 8, 13, 21, 34])
4 print(a)          # [ 1  1  2  3  5  8 13 21 34]
5
6 b = a[2:5]
7 print(b)          # [2 3 5]
8
9 a[2] = 20
10 print(a)         # [ 1  1 20  3  5  8 13 21 34]
11 print(b)          # [2 3 5]
```

Numpy: slice - copy

```
1 import numpy as np
2
3 a = np.array([1, 1, 2, 3, 5, 8, 13, 21, 34])
4 print(a)          # [ 1  1  2  3  5  8 13 21 34]
5
6 b = a[2:5].copy()
7 print(b)          # [2 3 5]
8
9 a[2] = 20
10 print(a)         # [ 1  1 20  3  5  8 13 21 34]
11 print(b)          # [2 3 5]
```

Numpy: abs value on a Numpy array

```
1 import numpy as np
2
3 a = np.array([-1, 2, -3], [-4, 5, -7])
4 print(a)
5 print(a.dtype)
6 print()
7
8 abs_a = np.absolute(a)
9 print(abs_a)
10 print(abs_a.dtype)
```

```
1 [[-1  2 -3]
2  [-4  5 -7]]
3 int64
4
5 [[1  2  3]
6  [4  5  7]]
7 int64
```

- absolute

Numpy: Logical not on a Numpy array

```
1 import numpy as np
2
3 a = np.array([True, True, False])
4 print(a.dtype)
5 print(a)
6 print()
7
8 not_a = np.logical_not(a)
9 print(not_a.dtype)
10 print(not_a)
11 print()
12
13 b = np.array([True, True, False, 0, 42])
14 print(b.dtype)
15 print(b)
```

```
16 print()
17
18 not_b = np.logical_not(b)
19 print(not_b.dtype)
20 print(not_b)
21 print()
```

```
1 bool
2 [ True  True False]
3
4 bool
5 [False False  True]
6
7 int64
8 [ 1  1  0  0 42]
9
10 bool
11 [False False  True  True False]
```

- [logical not](#)

Numpy: Vectorize a function

```
1 import numpy as np
2
3 def fibo(n):
4     if n == 1 or n == 2:
5         return 1
6     a, b = 1, 1
7     for _ in range(n-2):
8         a, b = b, a + b
9     return b
10
11 vfibo = np.vectorize(fibo)
12 a = np.array([
13     [1, 2, 3, 4, 5, 6],
14     [7, 8, 9, 10, 11, 12],
15 ])
16 print(a)
17 print(a.dtype)
18 print()
```

```
19
20 b = vfibo(a)
21 print(b)
22 print(b.dtype)
```

```
1 [[ 1  2  3  4  5  6]
2  [ 7  8  9 10 11 12]]
3 int64
4
5 [[ 1    1    2    3    5    8]
6  [ 13   21   34   55   89  144]]
7 int64
```

- [vectorize](#)

Numpy: Vectorize len

```
1 import numpy as np
2
3 animals = np.array(['Cow', 'Elephant', 'Snake',
'Camel', 'Praying Mantis'])
4 print(animals)
5
6 vlen = np.vectorize(len)
7 print(vlen(animals))
```

```
1 ['Cow' 'Elephant' 'Snake' 'Camel' 'Praying Mantis']
2 [ 3  8  5  5 14]
```

Numpy: Vectorize lambda

```
1 import numpy as np
2
3 animals = np.array(['Cow', 'Elephant', 'Snake',
'Camel', 'Praying Mantis'])
4 print(animals)
5
6 longer_than_5 = np.vectorize(lambda x: len(x) > 5)
```

```
7 long_animals_bool = longer_than_5(animals)
8 print(long_animals_bool)
```

```
1 ['Cow' 'Elephant' 'Snake' 'Camel' 'Praying Mantis']
2 [False True False False True]
```

Numpy: Filtering array

```
1 import numpy as np
2
3 animals = np.array(['Cow', 'Elephant', 'Snake',
'Camel', 'Praying Mantis'])
4 print(animals)
5
6 longer_than_5 = np.vectorize(lambda x: len(x) > 5)
7 long_animals_bool = longer_than_5(animals)
8 print(long_animals_bool)
9
10 long_animals = animals[long_animals_bool]
11 print(long_animals)
```

```
1 ['Cow' 'Elephant' 'Snake' 'Camel' 'Praying Mantis']
2 [False True False False True]
3 ['Elephant' 'Praying Mantis']
```

Numpy: Filter matrix values

```
1 import numpy as np
2 import re
3
4 scores = np.array([
5     [23, 37, 18, 97, 13, 40],
6     [10, 15, 20, 30, 39, 50],
7     [99, 20, 83, 42, 19, 31],
8     [19, 11, 55, 78, 39, 27]
9 ])
10 print(scores)
11 print()
12
```

```
13 high_scores_boolean = (scores > 20)
14 print(high_scores_boolean)
15 print()
16
17 high_scores = scores[high_scores_boolean]
18 print(high_scores)
```

```
1 [[23 37 18 97 13 40]
2 [10 15 20 30 39 50]
3 [99 20 83 42 19 31]
4 [19 11 55 78 39 27]]
5
6 [[ True  True False  True False  True]
7 [False False False  True  True  True]
8 [ True False  True  True False  True]
9 [False False  True  True  True  True]]
10
11 [23 37 97 40 30 39 50 99 83 42 31 55 78 39 27]
```

Numpy: Filter matrix rows

```
1 import numpy as np
2
3 names = np.array(['Mary', 'Bar', 'Joe', 'Jane'])
4 print(names)
5 print()
6
7 def has_ar(text):
8     return "ar" in text
9     # if "ar" in text:
10         # return True
11     # else:
12         # return False
13
14 names_with_ar_selector = np.vectorize(has_ar)
15 names_with_ar_bool = names_with_ar_selector(names)
16 print(names_with_ar_bool)
17 print()
18
19 scores = np.array([
20     [23, 37, 18, 97, 13, 40],
```

```
21      [10, 15, 20, 30, 39, 50],  
22      [99, 20, 83, 42, 19, 31],  
23      [19, 11, 55, 78, 39, 27]  
24  ])  
25  
26 print(scores[names_with_ar_bool])
```

```
1 ['Mary' 'Bar' 'Joe' 'Jane']  
2  
3 [ True  True False False]  
4  
5 [[23 37 18 97 13 40]  
6  [10 15 20 30 39 50]]  
7  
8 [[23 37 18 97 13 40]  
9  [10 15 20 30 39 50]]
```

Numpy: Stat

```
1 import numpy as np  
2  
3 scores = np.array([23, 37, 18, 97, 13, 40])  
4 print(scores.sum())          # 228  
5 print(len(scores))         # 6  
6 print(scores.mean())        # 38.0  
7  
8 print(scores.std())         # 28.0950766743 standard  
deviation  
9 print(scores.var())          # 789.333333333 variance  
10 print(np.median(scores))     # 30.0  
11 print(scores.max())          # 97  
12 print(scores.min())          # 13  
13  
14 print(scores.cumsum())       # [ 23   60   78  175  188  
228]
```

Numpy: Serialization

```
1 import numpy as np
2
3 scores = np.array([
4     [23, 37, 18, 97, 13, 40],
5     [10, 15, 20, 30, 39, 50],
6     [99, 20, 83, 42, 19, 31],
7     [19, 11, 55, 78, 39, 27]
8 ])
9 filename = 'scores.npy'
10 np.save(filename, scores)
11
12 s = np.load(filename)
13 print(s)
```

Numpy: Load from Matlab file

```
1 import scipy.io
2
3 file_path = 'data.mat'
4 mat = scipy.io.loadmat(file_path)
5 data = mat['data']
6 print(type(data))
7 print(data)
```

- numpy.ndarray

Numpy: Save as Matlab file

```
1 import scipy.io
2 import numpy as np
3
4 data = np.random.random((2, 5))
5 print(data)
6
7 file_path = 'data.mat'
8 scipy.io.savemat(file_path, {'data': data})
```

Numpy: Horizontal stack vectors (hstack)

```
1 import numpy as np
2
3 a = np.array([1, 2, 3])
4 b = np.array([4, 5, 6])
5 c = np.array([7, 8, 9])
6 print(a)
7 print(b)
8 print(c)
9 print()
10
11 d = np.hstack([a, b])
12 print(d)
13 print()
14
15 e = np.hstack([d, c])
16 print(e)
```

```
1 [1 2 3]
2 [4 5 6]
3 [7 8 9]
4
5 [1 2 3 4 5 6]
6
7 [1 2 3 4 5 6 7 8 9]
```

Numpy: Append or vertically stack vectors and matrices (vstack)

```
1 import numpy as np
2
3 a = np.array([1, 2, 3])
4 b = np.array([4, 5, 6])
5 c = np.array([7, 8, 9])
6 print(a)
7 print(b)
8 print(c)
9 print()
10
11 m = np.vstack([a, b])
12 print(m)
```

```
13 print()  
14  
15 d3 = np.vstack([m, c])  
16 print(d3)
```

```
1 [1 2 3]  
2 [4 5 6]  
3 [7 8 9]  
4  
5 [[1 2 3]  
6 [4 5 6]]  
7  
8 [[1 2 3]  
9 [4 5 6]  
10 [7 8 9]]
```

Numpy uint8

```
1 import numpy as np  
2  
3 a = np.array([127], 'uint8')  
4 print(a.dtype)    # uint8  
5 print(a)          # [127]  
6  
7 a[0] += 1           # [128]  
8 print(a)  
9  
10 a[0] -= 1         # [127]  
11 print(a)  
12  
13 a[0] = 255  
14 print(a)          # [255]  
15  
16 a[0] += 1  
17 print(a)          # [0]
```

Numpy int8

```
1 import numpy as np
2
3 a = np.array([127], 'int8')
4 print(a.dtype)      # int8
5 print(a)           # [127]
6
7 a[0] += 1          # [-128]
8 print(a)
9
10 a[0] -= 1         # [127]
11 print(a)
12
13 a[0] = 255
14 print(a)          # [-1]
15
16 a[0] += 1
17 print(a)          # [0]
```

Pandas

Pandas

- [Pandas](#) Python Data Analysis Library
- Handle data sequences
- [A Beginner's Guide to Optimizing Pandas Code for Speed](#)

Planets

```
1 name,distance,mass
2 Mercury,0.4,0.055
3 Venus,0.7,0.815
4 Earth,1,1
5 Mars,1.5,0.107
6 Ceres,2.77,0.00015
7 Jupiter,5.2,318
8 Saturn,9.5,95
9 Uranus,19.6,14
10 Neptune,30,17
11 Pluto,39,0.00218
12 Charon,39,0.000254
```

Pandas Planets - Dataframes

```
1 import pandas as pd
2
3 df = pd.read_csv('planets.csv', index_col='name')
4 print(type(df))    # <class
'pandas.core.frame.DataFrame'
5 print(df)
6
7 df['dm'] = df['distance'] * df['mass']
8 print(df.head())
9
```

```
10 big = df[ df['mass'] > 20 ]
11 print(big)
```

	distance	mass
1		
2	name	
3	Mercury	0.40
4	Venus	0.70
5	Earth	1.00
6	Mars	1.50
7	Ceres	2.77
8	Jupiter	5.20
9	Saturn	9.50
10	Uranus	19.60
11	Neptune	30.00
12	Pluto	39.00
13	Charon	39.00

	distance	mass	dm
1			
2	name		
3	Mercury	0.40	0.022000
4	Venus	0.70	0.570500
5	Earth	1.00	1.000000
6	Mars	1.50	0.160500
7	Ceres	2.77	0.000415

	distance	mass	dm
1			
2	name		
3	Jupiter	5.2	1653.6
4	Saturn	9.5	902.5

Pandas Stocks

```
1 import pandas
2 import pandas_datareader.data as web
3 all_data = { ticker: web.get_data_yahoo(ticker) for
ticker in ['AAPL', 'IBM', 'MSFT' \
4 , 'GOOG']}
5
6 print(all_data.keys()) # dict_keys(['MSFT',
'IBM', 'AAPL', 'GOOG'])
```

```
7 print(all_data['MSFT'].keys()) # Index(['Open',  
'High', 'Low', 'Close', 'Volume', '\\  
8 Adj Close'], dtype='object')  
9  
10 price = pandas.DataFrame({ticker: data['Adj Close']  
for ticker, data in all_data.items()  
11 ms()})  
12  
13 print(price.head())  
14  
15 volume = pandas.DataFrame({ticker: data['Volume'] for  
ticker, data in all_data.items()  
16 ()})  
17  
18 print(volume.tail())  
19  
20 returns = price.pct_change() # change in percentage  
21 print(returns.head())  
22  
23 # correlation  
24 print(returns.MSFT.corr(returns.IBM)) #  
0.49532932971  
25 print(returns.MSFT.corr(returns.AAPL)) #  
0.389551383559  
26  
27 # covariance  
28 print(returns.MSFT.cov(returns.IBM)) #  
8.50115754064e-05  
29 print(returns.MSFT.cov(returns.AAPL)) #  
9.15254855961e-05
```

Pandas Stocks

```
1 import pandas  
2 prices = pandas.read_csv('stock_prices.csv')  
3 print(prices)
```

Merge Dataframes

```
1 import pandas as pd  
2 import numpy as np
```

```

3 import matplotlib.pyplot as plt
4
5 # s = pd.Series([1,3,5,np.nan,6,8])
6 # dates = pd.date_range('20130101', periods=6)
7 # x = pd.date_range('20130101', periods=6, freq='3D')
8 # df = pd.DataFrame(np.random.randn(6,4), index=dates,
columns=list('ABCD'))
9 # df = pd.DataFrame(np.random.randn(6,4), index=dates,
columns=list('ABCD'))
10 # df = pd.DataFrame(np.random.randn(6,4), index=dates,
columns=list('ABC'))
11 # df2 = pd.DataFrame({ 'A' : 1.,
12 #                      'B' : pd.Timestamp('20130102'),
13 #                      'C' :
pd.Series(1,index=list(range(4)),dtype='float32'),
14 #                      'D' : np.array([3] *
4,dtype='int32'),
15 #                      'E' :
pd.Categorical(["test","train","test","train"]),
16 #                      'F' : 'foo' })
17 a = pd.DataFrame({ 'A' : ['Joe', 'Jane', 'Foo',
'Bar'], 'B' : [1, 23, 12, 5] })
18 b = pd.DataFrame({ 'A' : ['Joe', 'Jane', 'Foo',
'Bar'], 'B' : [7, 10, 27, 1] })
19 #c = pd.DataFrame({ 'A' : ['Jane', 'Joe', 'Foo',
'Bar'], 'B' : [10, 7, 27, 1] })
20 c = b.sort_values(by = 'A')
21 print(a)
22 print(b)
23 print(c)
24 print('---')
25 #print(a+b)
26 x = pd.merge(a, b, on='A')
27 z = pd.DataFrame({ 'A' : x.A, 'B' : x.B_x + x.B_y })
28 print(z)
29
30
31
32 #sa = a.sort_values(by = 'A')
33 #sc = c.sort_values(by = 'A')
34 print('-----')
35 #print(sa)
36 #print(sc)
37 y = pd.merge(a, c, on='A')
38 #print(x)

```

```
39 q = pd.DataFrame({ 'A' : y.A, 'B' : y.B_x + y.B_y })
40 print(z)
```

Analyze Alerts

```
1 import pandas
2 alerts = pandas.read_csv('../data/alerts.csv')
3 print(alerts.head())
4 #print(alerts.count())
```

Analyze IFMetrics

```
1 import pandas
2 data = pandas.read_csv('../data/ifmetrics.csv',
na_values=['(null)'])
3 data.fillna(0, inplace=True)
4 # , parse_dates=True )
5 # print(type(data)) # pandas.core.frame.DataFrame
6 print(data.columns) # Index([ ... ], dtype='object',
length=135)
7
8 #print(data['Utilization In - Threshold Exception
Rate'].head(3))
9
10 for col in ['Utilization In - Threshold Exception
Rate', 'Overall Exception Rate']:
11     dt = data[col]
12     print(dt[dt != 0])
13
14
15 #print(data.head(1))
16 #print(data.get_values())
```

Create Excel file for experiment with random data

Input is an excel file with the following columns:

```
1 genome name, c1, c2, c3, c4, c5, c6
```

- c1-c3 are numbers of cond1
- c4-c6 are numbers of cond2

We would like to filter to the lines that fulfill the following equations:

```
1 log2(avg(1-3) / avg(4-6)) > limit
2 other_limit > p.value()
```

```
1 import numpy as np
2 import pandas as pd
3 import datetime
4 import sys
5
6 if len(sys.argv) < 2:
7     exit("Need number of rows")
8
9 rows_num = int(sys.argv[1])
10 cols_num = 6
11
12 start = datetime.datetime.now()
13 x = np.random.rand(rows_num, cols_num)
14
15 genome_names = list(map(lambda i: f'g{i}', range(rows_num)))
16 column_names = list(map(lambda i: f'm{i}', range(cols_num)))
17
18 df = pd.DataFrame(x, index=genome_names,
columns=column_names)
19 df.index.name = 'genome name'
20
21 print(df.head())
22 print(datetime.datetime.now() - start)
23 df.to_excel('raw_data.xlsx')
24 print(datetime.datetime.now() - start)
```

Calculate Genome metrics

```
1 import pandas as pd
2 import numpy as np
3 import datetime
4 import sys
5
6 if len(sys.argv) < 2:
7     exit("Need filename")
8 filename = sys.argv[1]
9
10
11 def calculate_averages(row):
12     v1 = row.iloc[0:3].mean()
13     v2 = row.iloc[3:6].mean()
14     return np.log2(v1/v2)
15
16 start = datetime.datetime.now()
17 df = pd.read_excel(filename, index_col='genome name')
18 print(df.head())
19 print(datetime.datetime.now() - start)
20
21 calculated_value = df.apply(calculate_averages,
axis=1)
22 print(datetime.datetime.now() - start)
23
24 threshold = 0.2
25 filtered_df = df[calculated_value > threshold]
26 print(filtered_df.head())
27 print(datetime.datetime.now() - start)
```

Calculate Genome metrics - add columns

```
1 import pandas as pd
2 import numpy as np
3 import datetime
4 import sys
5
6 if len(sys.argv) < 2:
7     exit("Need filename")
8 filename = sys.argv[1]
9
```

```

10
11 def calculate_averages(row):
12     v1 = row.iloc[0:3].mean()
13     v2 = row.iloc[3:6].mean()
14     return np.log2(v1/v2)
15
16 start = datetime.datetime.now()
17 df = pd.read_excel(filename, index_col='genome name')
18 print(df.head())
19 print(datetime.datetime.now() - start)
20
21 # create a new column of the calculated value
22 df['calculated_value'] = df.apply(calculate_averages,
axis=1)
23 print(datetime.datetime.now() - start)
24
25 threshold = 0.2
26 filtered_df = df[df['calculated_value'] > threshold]
27 print(filtered_df.head())
28 print(datetime.datetime.now() - start)

```

Calculate Genome metrics - vectorized

```

1 import pandas as pd
2 import numpy as np
3 import datetime
4 import sys
5
6 if len(sys.argv) < 2:
7     exit("Need filename")
8 filename = sys.argv[1]
9
10 def calculate_averages(df):
11     v1 = df.iloc[:, 0:3].mean(axis=1)    # axis=1 ->
calculate the mean row-wise
12     v2 = df.iloc[:, 3:6].mean(axis=1)
13     return np.log2(v1/v2)
14
15 start = datetime.datetime.now()
16 df = pd.read_excel(filename, index_col='genome name')
17 print(df.head())
18 print(datetime.datetime.now() - start)

```

```
19
20 calculated_value = calculate_averages(df)
21 print(datetime.datetime.now() - start)
22
23 threshold = 0.2
24 filtered_df = df[calculated_value > threshold]
25 print(filtered_df.head())
26 print(datetime.datetime.now() - start)
```

Calculate Genome metrics - vectorized numpy

```
1 import pandas as pd
2 import numpy as np
3 import datetime
4 import sys
5
6 if len(sys.argv) < 2:
7     exit("Need filename")
8 filename = sys.argv[1]
9
10 def calculate_averages(df_numpy):
11     v1 = df_numpy[:, 0:3].mean(axis=1)
12     v2 = df_numpy[:, 3:6].mean(axis=1)
13     return np.log2(v1/v2)
14
15 start = datetime.datetime.now()
16 df = pd.read_excel(filename, index_col='genome name')
17 print(df.head())
18 print(datetime.datetime.now() - start)
19
20 # the .values attribute changes from Pandas to numpy array
21 # (no more iloc, no headers, no index)
22 calculated_value = calculate_averages(df.values)
23 print(datetime.datetime.now() - start)
24
25 threshold = 0.2
26 filtered_df = df[calculated_value > threshold]
27 print(filtered_df.head())
28 print(datetime.datetime.now() - start)
```

Genes using Jupyter

```
1 cd examples/pandas/  
2 jupyter notebook genes.ipynb
```

Combine columns

```
1 fname, lname, age  
2 Foo, Bar, 100  
3 Alma, Matter, 78  
4 Buzz, Lightyear, 23
```

```
1 import pandas as pd  
2  
3 filename = 'data.csv'  
4 df = pd.read_csv(filename)  
5 print(df)  
6  
7  
8 def combine(row):  
9     return row['lname'] + ' ' + row['fname']  
10  
11  
12 df['combined'] = df.apply(combine, axis=1)  
13 print(df)  
14  
15  
16 def new_column(row):  
17     columns = ['lname', 'age', 'fname']  
18     return '_'.join(map(lambda name: str(row[name]),  
19     columns))  
20 df['combined'] = df.apply(new_column, axis=1)  
21 print(df)
```

	fname	lname	age	
2 0	Foo	Bar	100	
3 1	Alma	Matter	78	
4 2	Buzz	Lightyear	23	
5	fname	lname	age	combined

```
6 0     Foo          Bar   100          Bar_Foo
7 1   Alma      Matter    78      Matter_Alma
8 2   Buzz  Lightyear    23  Lightyear_Buzz
9   fname      lname   age           combined
10 0    Foo          Bar   100        Bar_100_Foo
11 1   Alma      Matter    78  Matter_78_Alma
12 2   Buzz  Lightyear    23  Lightyear_23_Buzz
```

Pandas more

```
1 df.iloc[:, 4:10].sum(axis=1)
2
3 # rearrange order of columns
4 cols = list(df.columns)
5 df = df[ cols[0:4], cols[-1], cols[4:20] ]
6
7 to_csv('file.csv', index=False)
8 to_excel()
9
10 read_csv(filename, delimiter='\t')
11 to_csv(filename, sep='\t')
12
13
14 # after filtering out some rows:
15 df = df.reset_index()
16 df.reset_index(drop=True, inplace=True)
17
18
19 fileter with
20 df.loc[ ~df['Name'].str.contains('substring') ]
21
22 can also have regex=True parameter
23
24 # replace values
25 df[ df['Name'] == 'old', 'Name' ] = 'new'
```

Pandas Series

```
1 import pandas
2
3 s = pandas.Series([1, 1, 2, 3, 5, 8])
```

```

4 print(s)
5
6 # 0      1
7 # 1      1
8 # 2      2
9 # 3      3
10 # 4     5
11 # 5     8
12 # dtype: int64
13
14 print(s.values)    # [1 1 2 3 5 8]
15 print(s.index)     # RangeIndex(start=0, stop=6, step=1)
16
17 print('----')
18 print(s.sum())      # 20
19 print(s.count())     # 6
20 print(s.mean())      # 3.33333333333
21 print(s.median())    # 2.5
22 print(s.std())       # 2.73252020426
23 print(s.cumsum())
24
25 # 0      1
26 # 1      2
27 # 2      4
28 # 3      7
29 # 4     12
30 # 5     20
31 # dtype: int64

```

Pandas Series with names

```

1 import pandas
2
3 planets      = ['Mercury', 'Venus', 'Earth', 'Mars']
4 distances_raw = [      0.4,      0.7,          1,     1.5 ]
5 masses_raw   = [      0.055,    0.815,         1,    0.107]
6
7 distance = pandas.Series(distances_raw, index =
planets)
8 mass      = pandas.Series(masses_raw,      index =
planets)
9
10 print(distance)

```

```
11
12 # Mercury      0.40
13 # Venus        0.70
14 # Earth        1.00
15 # Mars         1.50
16 # dtype: float64
17
18
19 print(distance.index)
20 # Index(['Mercury', 'Venus', 'Earth', 'Mars'],
21 #          dtype='object')
22
23 print(distance[distance < 0.8])
24 # Mercury      0.4
25 # Venus        0.7
26 # dtype: float64
27
28 print('-----')
29 print(distance/mass)
30 # Mercury      7.272727
31 # Venus        0.858896
32 # Earth        1.000000
33 # Mars         14.018692
34 # dtype: float64
```

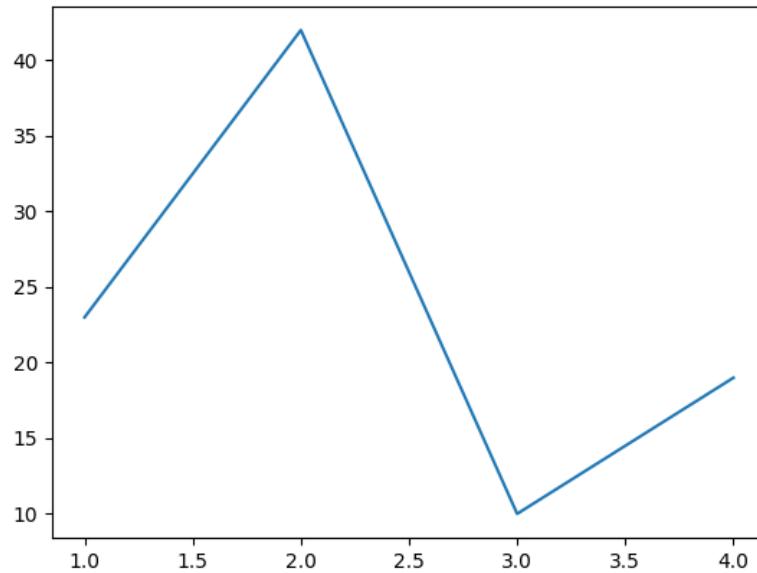
Matplotlib

About Matplotlib

- [matplotlib](#)

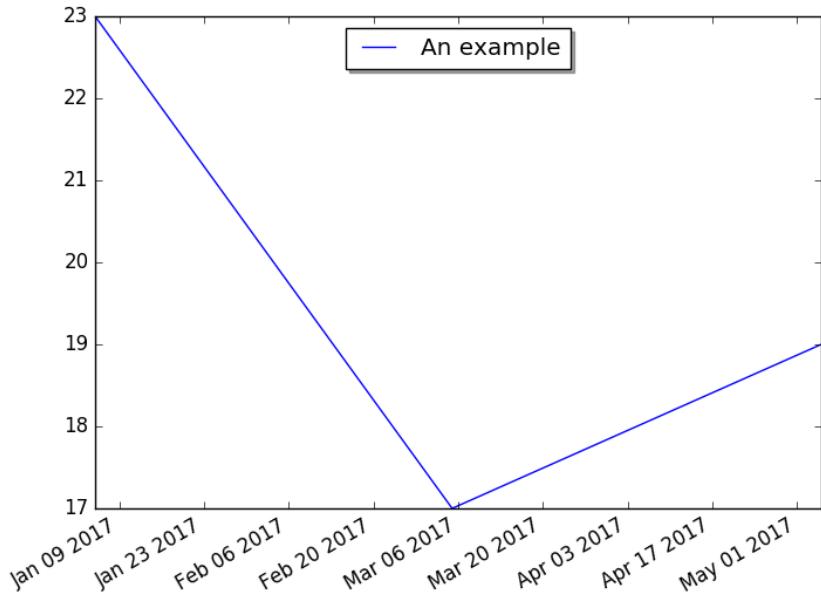
Matplotlib Line

```
1 import matplotlib.pyplot as plt
2
3 plt.plot([ 1, 2, 3, 4 ], [ 23, 42, 10, 19 ])
4 #fig, ax = plt.subplots()
5 #ax.plot(
6 #    [ 1, 2, 3, 4 ],
7 #    [ 23, 42, 10, 19 ],
8 #)
9 plt.show()
10 #plt.savefig('line.png')
```



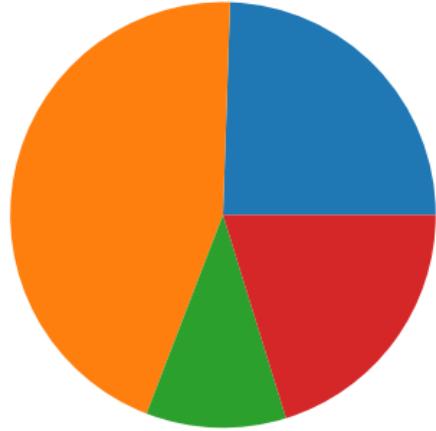
Matplotlib Line with dates

```
1 import datetime
2 import matplotlib.pyplot as plt
3
4 fig, subplots = plt.subplots()
5 subplots.plot(
6     [datetime.date(2017, 1, 5), datetime.date(2017, 3,
5), datetime.date(2017, 5, 5) \
7 ],
8     [ 23, 17, 19 ],
9     label='An example',
10 )
11 subplots.legend(loc='upper center', shadow=True)
12 fig.autofmt_xdate()
13 plt.show()
14 #plt.savefig('line_with_dates.png')
```



Matplotlib Simple Pie

```
1 import matplotlib.pyplot as plt
2
3 plt.pie([ 23, 42, 10, 19 ])
4
5 plt.show()
6 #plt.savefig('simple_pie.png')
```



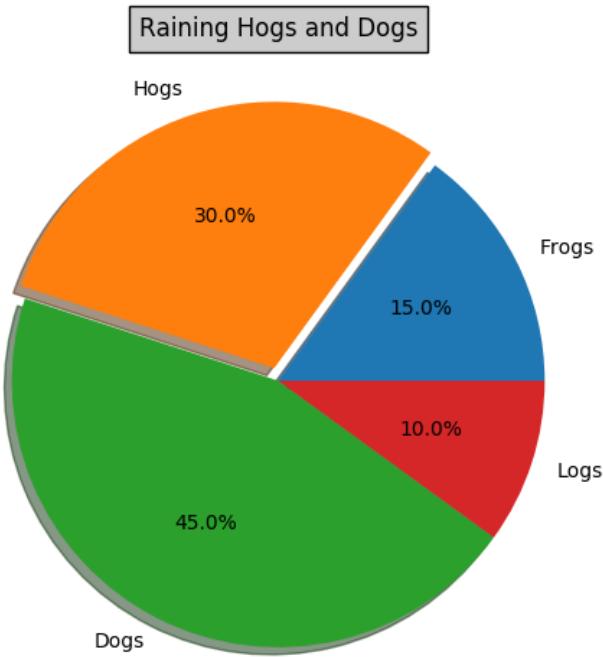
Matplotlib Simple Pie with params

```
1 import matplotlib.pyplot as plt
2
3 plt.pie(
4     x = [ 23, 42, 10, 19 ],
5     #explode = [0, 0, 0.1, 0.3],
6     #labels = ["failure", "success", "maybe", "what"],
7     #colors = ["red", "green", "blue", "#A395C1"],
8     #shadow = True,
9     #radius = 2,
10 )
11
12 plt.show()
```

- [pyplot pie](#)

Matplotlib Pie

```
1 import matplotlib.pyplot as plt
2
3
4 # Make a square figure and axes
5 plt.figure(1, figsize=(6, 6))
6 #ax = plt.axes([0.1, 0.1, 0.8, 0.8])
7
8 labels = 'Frogs', 'Hogs', 'Dogs', 'Logs'
9 fracs = [15, 30, 45, 10]
10
11 explode = (0, 0.05, 0, 0)
12 plt.pie(fracs,
13         explode=explode,
14         labels=labels,
15         autopct='%.1f%%',
16         shadow=True)
17 plt.title('Raining Hogs and Dogs',
18           bbox={'facecolor': '0.8', 'pad': 5})
19
20 plt.show()
21 #plt.savefig('pie.png')
22 #plt.savefig('pie.pdf')
```



Matplotlib Pie 2

```
1 import matplotlib.pyplot as plt
2
3 cases = {
4     'success': 38,
5     'failure': 7,
6     'skipped': 3,
7     'xfailed': 8,
8     'xpassed': 4,
9 }
10
11 explode = (0, 0.1, 0.1, 0.1, 0.1)
12 labels = cases.keys()
13 sizes = cases.values()
14
15 fig1, ax1 = plt.subplots()
```

```
16 ax1.pie(sizes, explode=explode, labels=labels,  
17 autopct='%.1f%%', shadow=True, start\  
18 angle=90)  
19  
20 plt.tight_layout()  
21 plt.show()
```

Plot, scatter, histogram

- plot - line
- scatter - just the values
- histogram (to group the values into bins)
- plt.hist(data, bin=10)

Seaborn

Seaborn use examples

seaborn

In Jupyter notebook type %matplotlib before writing the seaborn code.

In plain Python import matplotlib, then assign the result of the plotting function to a variable, and call matplotlib.pyplot.show(r).

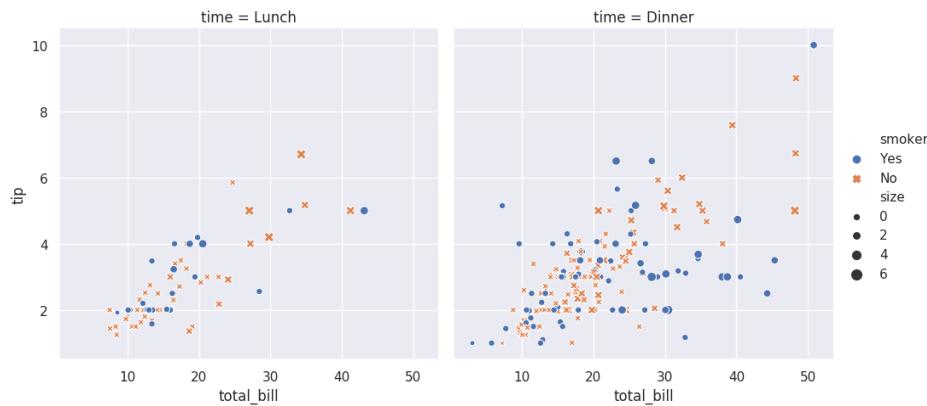
Seaborn tip

```
1 """
2 Source : https://seaborn.pydata.org/introduction.html
3 """
4
5 import seaborn as sns
6
7 sns.set() # Apply the default default seaborn theme,
8 # scaling, and color palette. Optional.
9
10 tips = sns.load_dataset("tips") # Load example
11 # dataset into Pandas DataFrame
12 #print(type(tips))
13 # print(tips)
14
15 plot = sns.relplot(
16     x = "total_bill",
17     y = "tip",
18     col = "time",
```

```

19     hue = "smoker",
20     style = "smoker",
21     size = "size",
22     data = tips)
23
24 # print(type(plot))      # seaborn.axisgrid.FacetGrid
25 plot.savefig("tips.png")

```



Seaborn Anscombes Quartet

```

1 """
2 Anscombe's quartet
3 =====
4
5 _thumb: .4, .4
6
7 Source:
https://seaborn.pydata.org/examples/anscombes\_quartet.html
1
8 """
9 import seaborn as sns
10 import matplotlib
11 sns.set(style="ticks")
12
13 # Load the example dataset for Anscombe's quartet
14 df = sns.load_dataset("anscombe")
15
16 # Show the results of a linear regression within each

```

```
dataset
17 r = sns.lmplot(
18     x="x",
19     y="y",
20     col="dataset",
21     hue="dataset",
22     data=df,
23     col_wrap=2,
24     ci=None,
25     palette="muted",
26     height=4,
27     scatter_kws={"s": 50, "alpha": 1})
28
29 matplotlib.pyplot.show(r)
```

Jupyter notebooks

Jupyter on Windows

On Windows install [Anaconda](#)

and then you'll be able to run Jupyter notebook from the start menu.

Jupyter on Linux and OSX

Install

For Linux and OSX I recommend using **virtualenv** and installing with **pip**.

```
1 virtualenv -p python3 ~/venv3
2 source ~/venv3/bin/activate
3 pip install jupyter
```

Run

```
1 cd examples/jupyter/
2 jupyter notebook
```

- Your browser should open. If not, there is a link in the terminal.

Jupyter add

- Open an existing notebook (ipynb file). e.g examples/jupyter/add.ipynb
 - Create new notebook.
 - File - Save As
 - ...
 - Quit - shut down the notebook server.
-

```
1 def add(x, y):  
2     return x+y  
3  
4 add(2, 3)
```

Planets

```
1 Planet name,Distance (AU),Mass  
2 Mercury,0.4,0.055  
3 Venus,0.7,0.815  
4 Earth,1,1  
5 Mars,1.5,0.107  
6 Ceres,2.77,0.00015  
7 Jupiter,5.2,318  
8 Saturn,9.5,95  
9 Uranus,19.6,14  
10 Neptune,30,17  
11 Pluto,39,0.00218  
12 Charon,39,0.000254
```

Jupyter notebook Planets

```
1 %config IPCompleter.greedy=True
2 import pandas as pd
3 import numpy as np
4 import matplotlib.pyplot as plt
5
6
7 planets = pd.read_csv('planets.csv')
8 planets
9
10 planets.__class__.__name__
11 planets.columns
12 planets.dtypes
13 planets.index
14 planets.values
15 planets.describe()
16
17 #planets.sort_values('Mass', ascending=False)
18 planets.sort_values('Planet name', ascending=False)
19
20 planets.Mass
21 planets['Planet name']
22 planets[2:5]
23 planets.loc[3:6, ['Mass', 'Planet name']]
24 planets.Mass > 1
25
26 planets[planets.Mass > 1]
27 planets['Planet name'].isin(['Earth', 'Mars'])
28 planets[ planets['Planet name'].isin(['Earth',
29 'Mars']) ]
30 planets[(planets.Mass > 1) & (planets.Mass < 100)]
31 # element-wise boolean and
32
33 center = 'Earth'
34 this = planets[ planets['Planet name'] == center ]
35 mass = this.iloc[0]['Mass']
36 dist = this.iloc[0]['Distance (AU)']
37
38 # gravitational force is  $F = G * (mass1 * mass2) / D^{**2}$ 
39 G = 6
40 D = abs(dist - planets['Distance (AU)'])
```

```
41 D
42
43 forces = planets.copy()
44 forces
45
46 G * (planets.Mass * mass) / D**2
47 forces['F'] = G * (planets.Mass * mass) / D**2
48 forces.drop(columns = 'Mass', inplace=True)
49 forces.drop(columns = 'Distance (AU)', inplace=True)
50 forces
```

Jupyter StackOverflow

- Download the latest dataset from the [survey](#).
 - unzip the file. Feel free to remove the __MACOSX/ directory.
-

```
1 %config IPCompleter.greedy=True
2 import pandas as pd
3 import numpy as np
4 import matplotlib.pyplot as plt
5 import seaborn as sns
6
7
8 # The following might not work on your computer if it
does not have enough free memo\
9 try
10 df = pd.read_csv('survey_results_public.csv')
11 df
12
13 df.size # size in memory 7,555,055 it is too big if
you only have 8gb memory
14
15 df.count()
16
17 df.info()
18
19 df.describe() # only few columns were identified to
have numeric values
20
21 df.head(3)
```

Jupyter StackOverflow - selected columns

```
1 df = pd.read_csv('survey_results_public.csv', usecols=[  
2     'Country', 'OpenSourcer', 'Co\  
2 mpTotal'])
```

Jupyter processing chunks

```
1 for chunk in pd.read_csv('survey_results_public.csv',  
2     chunksize=chunksize):  
2     process(chunk)
```

Jupyter StackOverflow - selected rows

```
1 # Load only data from a specific country.  
2  
3 country_name = 'Israel'  
4 df = None  
5 for chunk in pd.read_csv('survey_results_public.csv',  
6     chunksize=10000):  
7     part = chunk[ chunk['Country'] == country_name ]  
8     if df is None:  
9         df = part.copy(deep = True)  
10    else:  
11        df = df.append(part.copy(deep = True),  
12        ignore_index = True)  
13  
14 df.count()  
14 df.size
```

Jupyter StackOverflow - biggest countries (in terms of number of responses)

```
1 country_count = df['Country'].value_counts()  
2 country_count  
3  
4 type(country_count) # pandas.core.series.Series
```

```
5 # country_count.__class__.__name__ # Series
6
7 # We can use it either as a dictionary or as a list
8 country_count['United States'] # 20949
9 # country_count[0] # 20949
10 # country_count['Israel']
11
12 # Take the top 20 countries
13 first20 = country_count.head(20)
14 first20
15 # type(first20) # Series
16
17 # first20 = country_count.iloc[0:20] # part of the
Series
18 # first20
19 # type(first20) # Series
20
21 #first20 = country_count[0:20]
22 # first20
23 # type(first20) # Series
24
25 # Select rows of the "biggest" countries
26 first20.keys()
```

Jupyter StackOverflow - histogram

```
1 # Histogram of the top 20 countries
2 first20.hist(bins = 20)
3
4 # Plot using Seaborn
5 plot = sns.relplot(data = first20)
6 plot.set_xticklabels(rotation=90)
```

Jupyter StackOverflow - filter by country

```
1 df['Country'] == 'Israel'
2 df[ df['Country'] == 'Israel' ]
3
4 df[ df['Country'].isin( ['India', 'Israel'] ) ]
5 df[ df['Country'].isin( first20.keys() ) ]
```

Jupyter StackOverflow - OpenSourcer

```
1 df['OpenSourcer'].value_counts()  
2  
3 df['OpenSourcer'].unique()
```

Jupyter StackOverflow - cross tabulation

```
1 # Crosstabulation  
2 first10 = country_count.head(10)  
3 subset = df[ df['Country'].isin( first10.keys() ) ]  
4 # subset.count()  
5  
6 # subset['OpenSourcer'].value_counts()  
7 grouped = subset.groupby('Country')  
['OpenSourcer'].value_counts()  
8 # grouped.plot.bar(figsize=(15,15))  
9  
10 pd.crosstab(subset['Country'], df['OpenSourcer'])  
11  
12 ct = pd.crosstab(subset['Country'],  
df['OpenSourcer']).apply(lambda r: 100 * r/r.sum\  
13 (), axis=1)  
14 ct  
15  
16 ct.transpose().hist(figsize=(15, 15))
```

Jupyter StackOverflow - salaries

```
1 # Try to show the average salary by country  
2 grp = df.groupby('Country').mean().round({'CompTotal' :  
0})  
3 #grp['CompTotal']  
4 pd.set_option('display.float_format', lambda x:  
'{:,.}{}'.format(x))  
5 grp.sort_values('CompTotal', ascending=False)
```

Jupyter StackOverflow - replace values

```
1 nd = df.replace({'OpenSourcer' : {
2     'Never' : 0,
3     'Less than once per year' : 1,
4     'Less than once a month but more than once per
year' : 2,
5     'Once a month or more often' : 3,
6     } })
7 nd
8 nd.describe()
9 nd.groupby('Country').mean().sort_values('OpenSourcer',
ascending=False)
```

Jupyter StackOverflow - selected rows

```
1 # Distribution of responses among countries.
2 # Relation of Open Source contribution to experience.
3 # Open Source contribution by country.
4 # Look at the pdf file and create similar reports for a
specific country
```

Jupyter notebook Intellisense (TAB completion)

```
1 %config IPCompleter.greedy=True
```

Jupyter examples

```
1 examples/jupyter/planets.csv  
2  
3 examples/jupyter/planets.ipynb  
4  
5 examples/jupyter/numpy_matrix.ipynb  
6  
7 examples/jupyter/seaborn_tips.ipynb
```

IPy Widgets

- [Interact](#)
- [Widget list](#)

Testing

Traditional Organizations

- Months of planning
- Many months of development
- Many months of testing / qa
- Release once every few months or once a year
- (Waterfall)

Quality Assurance

- Nightly build
- Testing new features
- Testing bug fixes
- Maybe testing critical features again and again...
- ...or maybe not.
- Regression testing?
- Testing / qa has a huge boring repetitive part.
- It is also very slow and expensive.

Web age Organizations

- Very frequent releases (20-30 / day!)
- Very little time for manual testing
- CI - Continuous Integration
- CD - Continuous Delivery
- CD - Continuous Deployment

TDD vs Testing as an Afterthought

- TDD - Test Driven Development.

*

- Testing as an afterthought:
- Existing product
- Mostly works
- Hard to test

Why test?

- Business Value
- Avoid regression
- Better Software Design (TDD)
- Your Sanity

Testing Modes

- Functional testing
- Unit testing
- Integration testing
- Acceptance testing (BDD Behavior-driven development?)
- White box
- Black box
- Regression testing
- Usability testing
- Performance testing
- Load testing
- Security testing
- ...

Testing Applications

- Web site
- Web application
- Web API / Microservice (JSON, XML)
- Mobile Application
- Desktop Application (GUI)
- Command-line tool (CLI)
- Batch process

Testing What to test?

- How would you check that they work as expected?
- What if they get invalid input?
- Edge cases? (e.g. 0, -1, 131314134141)
- A value that is too big or two small.
- Invalid or no response from third-party system.

Testing in Python

- Doctest
- Unittest
- Pytest
- Nose
- Nimoy
- Hypothesis
- Selenium
- Tox

Testing Environment

- Git (or other VCS)
- Virtualenv
- Docker
- ...

Testing Setup - Fixture

- Web server
- Databases
- Other machines
- Devices
- External services

Testing Resources

- [AB Testing](#) Alan and Brent talk about Modern Testing

Testing with unittest

Use a module

We have a module called mymath that has two methods: add and div.

```
1 import mymath
2 print( mymath.add(2, 3) )
3 print( mymath.div(6, 2) )
```

```
1 import mymath
2 import sys
3
4 if len(sys.argv) != 4:
5     exit("Usage: {} [add|div] INT
INT".format(sys.argv[0]))
6
7 if sys.argv[1] == 'add':
8     print(mymath.add(int(sys.argv[2]),
int(sys.argv[3])))
9 if sys.argv[1] == 'div':
10    print(mymath.div(int(sys.argv[2]),
int(sys.argv[3])))
```

Test a module

```
1 import unittest
2 import mymath
3
4 class TestMath(unittest.TestCase):
5
6     def test_match(self):
7         self.assertEqual(mymath.add(2, 3), 5)
8         self.assertEqual(mymath.div(6, 3), 2)
```

```
9         self.assertEqual(mymath.div(42, 1), 42)
10        self.assertEqual(mymath.add(-1, 1), 0)
11
12 if __name__ == '__main__':
13     unittest.main()
```

The tested module

```
1 def add(x, y):
2     """Adding two numbers
3
4     >>> add(2, 3)
5     5
6
7     """
8     return x + y
9
10 def div(x, y):
11     """Dividing two numbers
12
13     >>> div(8, 2)
14     4
15     >>> div(8, 0)
16     Traceback (most recent call last):
17     ...
18     ZeroDivisionError: integer division or modulo by
19     zero
20
21     """
22
23
24 #print add(2, 3, 4)
```

Testing - skeleton

```
1 import unittest
2
3 def add(x, y):
4     return x+y
5
```

```

6 class Something(unittest.TestCase):
7
8     def setUp(self):
9         pass
10        #print("setup")
11
12    def tearDown(self):
13        pass
14        #print("teardown")
15
16    def test_something(self):
17        self.assertEqual(add(2, 3), 5)
18        self.assertEqual(add(0, 3), 3)
19        self.assertEqual(add(0, 3), 2)
20
21
22    def test_other(self):
23        self.assertEqual(add(-3, 3), 0)
24        self.assertEqual(add(-3, 2), 7)
25        self.assertEqual(add(-3, 2), 0)
26
27
28 if __name__ == '__main__':
29     unittest.main()

```

Testing

```

1 import unittest
2
3 class TestReg(unittest.TestCase):
4
5     def setUp(self):
6         self.str_number = "123"
7         self.str_not_number = "12x"
8
9     def test_match1(self):
10        self.assertEqual(1, 1)
11        self.assertRegexpMatches(self.str_number,
12            r'^\d+$')
13
14    def test_match2(self):
15        self.assertEqual(1, 1)
16        self.assertRegexpMatches(self.str_not_number,

```

```
r'^^\\d+$')  
16  
17 if __name__ == '__main__':  
18     unittest.main()  
19
```

Test examples

- [pylev](#) unittest
- [weighted-levenshtein](#)

Testing with PyTest

Pytest features

- Organize and run test per directory (test discovery)
- Run tests by name matching
- Run tests by mark (smoke, integration, db)
- Run tests in parallel with the xdist plugin.
- Create your own fixtures and distribute them.
- Create your own plugins and distribute them.

Pytest setup

Python 2

```
1 virtualenv venv2
2 source venv2/bin/activate
3 pip install pytest
```

Python 3

```
1 virtualenv venv3 -p python3
2 source venv3/bin/activate
3 pip install pytest
```

Python 3 Debian/Ubuntu

```
1 apt-get install python3-pytest
```

Python 3 RedHat/Centos

```
1 yum install python3-pytest
```

Testing with Pytest

A module called mymath with two functions: add and div.

```
1 def add(x, y):
2     """Adding two numbers
3
4     >>> add(2, 3)
5
6
7     """
8     return x + y
9
10 def div(x, y):
11     """Dividing two numbers
12
13     >>> div(8, 2)
14     4
15     >>> div(8, 0)
16     Traceback (most recent call last):
17     ...
18     ZeroDivisionError: integer division or modulo by
19     zero
20
21     """
22     return x / y
```

Testing functions

```
1 import mymath
2
3 def test_math():
4     assert mymath.add(2, 3) == 5
5     assert mymath.div(6, 3) == 2
6     assert mymath.div(42, 1) == 42
7     assert mymath.add(-1, 1) == 0
```

Testing class and methods

```
1 import mymath  
2  
3 class TestMath():  
4     def test_math(self):  
5         assert mymath.add(2, 3) == 5  
6         assert mymath.div(6, 3) == 2  
7         assert mymath.div(42, 1) == 42  
8         assert mymath.add(-1, 1) == 0
```

Pytest - execute

```
1 pytest test_mymath.py  
  
1 ===== test session starts  
=====  
2 platform darwin -- Python 3.6.3, pytest-3.3.0, py-  
1.5.2, pluggy-0.6.0  
3 rootdir: /Users/gabor/work/training/python, inifile:  
4 collected 1 item  
5  
6 examples/pytest/test_mymath.py .  
[100%]  
7  
8 ===== 1 passed in 0.01 seconds  
=====
```

Pytest - execute

```
1 pytest  
2 python -m pytest
```

Pytest simple module to be tested

An anagram is a pair of words containing the exact same letters in different order. For example:

- listen silent
 - elvis lives
-

```
1 def is_anagram(a_word, b_word):  
2     return sorted(a_word) == sorted(b_word)
```

Pytest simple tests - success

```
1 from mymod_1 import is_anagram  
2  
3 def test_anagram():  
4     assert is_anagram("elvis", "lives")  
5     assert is_anagram("silent", "listen")  
6     assert not is_anagram("one", "two")
```

Pytest simple tests - success output

```
1 $ pytest test_mymod_1.py  
2  
3 ===== test session starts  
=====  
4 platform darwin -- Python 3.5.2, pytest-3.0.7, py-  
1.4.33, pluggy-0.4.0  
5 rootdir: /examples/python/pt, ini file:  
6 collected 1 items  
7  
8 test_mymod_1.py .  
9  
10 ===== 1 passed in 0.03 seconds  
=====
```

Pytest simple tests - failure

- Failure reported by user: `is_anagram("anagram", "nag a ram")` is expected to return true.

- We write a test case to reproduce the problem. It should fail now.

```
1 from mymod_1 import is_anagram
2
3 def test_anagram():
4     assert is_anagram("elvis", "lives")
5     assert is_anagram("silent", "listen")
6     assert not is_anagram("one", "two")
7
8 def test_multiword_anagram():
9     assert is_anagram("ana gram", "naga ram")
10    assert is_anagram("anagram", "nag a ram")
```

Pytest simple tests - failure output

```
1 $ pytest test_mymod_2.py
2
3 ===== test session starts
4 =====
5 platform darwin -- Python 3.5.2, pytest-3.0.7, py-
6 1.4.33, pluggy-0.4.0
7 rootdir: /examples/python/pt, inifile:
8 collected 2 items
9
10 ===== FAILURES
11 =====
12
13     def test_multiword_anagram():
14         assert is_anagram("ana gram", "naga ram")
15 >     assert is_anagram("anagram", "nag a ram")
16 E         AssertionError: assert False
17 E             +   where False = is_anagram('anagram', 'nag a
18 ram')
19 test_mymod_2.py:10: AssertionError
20 ===== 1 failed, 1 passed in 0.09 seconds
=====
```

Exercise: test math functions

- Test methods of the [math](#) module.
- ceil
- factorial
- gcd

Exercise: test this app

Write tests for the `swap` and `average` functions of the `app` module. Can you find a bug?

```
1 def swap(txt):  
2     '''  
3     >>> half("abcd")  
4     cdab  
5     '''  
6     return txt[int(len(txt)/2):] +  
txt[:int(len(txt)/2)]  
7  
8 def average(*numbers):  
9     '''  
10    >>> average(2, 4, 6)  
11    4  
12    '''  
13    s = 0  
14    c = 0  
15    for n in numbers:  
16        s += n  
17        c += 1  
18    return s/c
```

Exercise: test the csv module

- [csv](#)

- Create a CSV file, read it and check if the results are as expected!
- Test creating a CSV file?
- Test round trip?

Solution: Pytest test math functions

```

1 import math
2
3 def test_gcd():
4     assert math.gcd(6, 9) == 3
5     assert math.gcd(17, 9) == 1
6
7 def test.ceil():
8     assert math.ceil(0) == 0
9     assert math.ceil(0.1) == 1
10    assert math.ceil(-0.1) == 0
11
12 def test_factorial():
13    assert math.factorial(0) == 1
14    assert math.factorial(1) == 1
15    assert math.factorial(2) == 2
16    assert math.factorial(3) == 6

```

```

1 import math
2 import pytest
3
4 def test_math():
5     with pytest.raises(Exception) as exinfo:
6         math.factorial(-1)
7     assert exinfo.type == ValueError
8     assert str(exinfo.value) == 'factorial() not
defined for negative values'
9
10    with pytest.raises(Exception) as exinfo:
11        math.factorial(1.2)
12    assert exinfo.type == ValueError
13    assert str(exinfo.value) == 'factorial() only
accepts integral values'

```

Solution: Pytest test this app

```
1 import app
2
3 def test_swap():
4     assert app.swap("abcd") == "cdab"
5     assert app.swap("abc") == "bca"
6     assert app.swap("abcde") == "cdeab"
7     assert app.swap("a") == "a"
8     assert app.swap("") == ""
9
10 def test_average():
11     assert app.average(2, 4) == 3
12     assert app.average(2, 3) == 2.5
13     assert app.average(42) == 42
14     #assert app.average() == 0
```

Solution: test the csv module

- 1 Tudor;Vidor;10;Hapci
- 2 Szundi;Morgo;7;Szende
- 3 Kuka;"Hofeherke;
- 4 alma";100;Kiralyno
- 5 Boszorkany;Herceq;9;Meselo

```
1 import csv
2
3
4 def test_csv():
5     filename =
'../../examples/csv/process_csv_file_newline.csv'
6     with open(filename) as fh:
7         rd = csv.reader(fh, delimiter=';')
8         assert rd.__next__() == ['Tudor', 'Vidor',
'10', 'Hapci']
9         assert rd.__next__() == ['Szundi', 'Morgo',
'7', 'Szende']
10        assert rd.__next__() == ['Kuka', 'Hofeherke;
\nalma', '100', 'Kiralyno']
```

```
11         assert rd.__next__() == ['Boszorkany',
'Herceg', '9', 'Meselo']
```

PyTest bank deposit

```
1 class NegativeDeposite(Exception):
2     pass
3
4 class Bank:
5     def __init__(self, start):
6         self.balance = start
7
8     def deposit(self, money):
9         if money < 0:
10             raise NegativeDeposite('Cannot deposit
negative sum')
11         self.balance += money
12         return
```

PyTest expected exceptions (bank deposit)

```
1 import pytest
2 from banks import Bank, NegativeDeposite
3
4
5 def test_negative_deposit():
6     b = Bank(10)
7     with pytest.raises(Exception) as exinfo:
8         b.deposit(-1)
9     assert exinfo.type == NegativeDeposite
10    assert str(exinfo.value) == 'Cannot deposit
negative sum'
```

```
1 pytest test_bank.py
2
3 test_bank.py .
```

PyTest expected exceptions (bank deposit) - no exception happens

Pytest properly reports that there was no exception where an exception was expected.

```
1 class NegativeDeposite(Exception):
2     pass
3
4 class Bank:
5     def __init__(self, start):
6         self.balance = start
7
8     def deposit(self, money):
9         #if money < 0:
10            #    raise NegativeDeposite('Cannot deposit
negative sum')
11        self.balance += money
12        return
```

```
1     def test_negative_deposit():
2         b = Bank(10)
3         with pytest.raises(NegativeDeposite) as e:
4             b.deposit(-1)
5             Failed: DID NOT RAISE <class 'Exception'>
```

PyTest expected exceptions (bank deposit) - different exception is raised

```
1 class NegativeDeposite(Exception):
2     pass
3
4 class Bank:
5     def __init__(self, start):
6         self.balance = start
7
8     def deposit(self, money):
9         if money < 0:
10            raise ValueError('Cannot deposit negative
```

```
sum')
11         self.balance += money
12     return
```

```
1 def test_negative_deposit():
2     b = Bank(10)
3     with pytest.raises(Exception) as exinfo:
4         b.deposit(-1)
5 >     assert exinfo.type == NegativeDeposite
6 E         AssertionError: assert <class 'ValueError'> ==
NegativeDeposite
7 E             + where <class 'ValueError'> = <ExceptionInfo
ValueError tb[=2].type
```

PyTest expected exceptions

```
1 import pytest
2
3 def divide(a, b):
4     if b == 0:
5         raise ValueError('Cannot divide by Zero')
6     return a / b
7
8 def test_zero_division():
9     with pytest.raises(ValueError) as e:
10         divide(1, 0)
11     assert str(e.value) == 'Cannot divide by Zero'
```

PyTest expected exceptions output

```
1 $ pytest test_exceptions.py
2
3 test_exceptions.py .
```

PyTest expected exceptions (text changed)

```
1 import pytest
2
```

```
3 def divide(a, b):
4     if b == 0:
5         raise ValueError('Cannot divide by Null')
6     return a / b
7
8 def test_zero_division():
9     with pytest.raises(ValueError) as e:
10         divide(1, 0)
11     assert str(e.value) == 'Cannot divide by Zero'
```

PyTest expected exceptions (text changed) output

```
1 $ pytest test_exceptions_text_changed.py
2
3
4     def test_zero_division():
5         with pytest.raises(ValueError) as e:
6             divide(1, 0)
7 >     assert str(e.value) == 'Cannot divide by Zero'
8 E     AssertionError: assert 'Cannot divide by Null'
== 'Cannot divide by Zero'
9 E             - Cannot divide by Null
10 E             ?           ^
11 E             + Cannot divide by Zero
12 E             ?           ^^^^
```

PyTest expected exceptions (other exception)

```
1 import pytest
2
3 def divide(a, b):
4     # if b == 0:
5     #     raise ValueError('Cannot divide by Zero')
6     return a / b
7
8 def test_zero_division():
9     with pytest.raises(ValueError) as e:
```

```
10         divide(1, 0)
11     assert str(e.value) == 'Cannot divide by Zero'
```

PyTest expected exceptions (other exception) output

```
1   $ pytest test_exceptions_failing.py
2
3   def test_zero_division():
4       with pytest.raises(ValueError) as e:
5 >           divide(1, 0)
6
7 test_exceptions_failing.py:10:
8 -----
9
10 a = 1, b = 0
11
12 def divide(a, b):
13     # if b == 0:
14     #     raise ValueError('Cannot divide by Zero')
15 >         return a / b
16 E     ZeroDivisionError: division by zero
```

PyTest expected exceptions (no exception)

```
1 import pytest
2
3 def divide(a, b):
4     if b == 0:
5         return None
6     return a / b
7
8 def test_zero_division():
9     with pytest.raises(ValueError) as e:
10         divide(1, 0)
11     assert str(e.value) == 'Cannot divide by Zero'
```

PyTest expected exceptions (no exception) output

```
1 def test_zero_division():
2     with pytest.raises(ValueError) as e:
3 >         divide(1, 0)
4 E             Failed: DID NOT RAISE <class 'ValueError'>
```

PyTest: Multiple Failures

```
1 def test_one():
2     assert True
3     print('one')
4
5 def test_two():
6     assert False
7     print('two')
8
9 def test_three():
10    assert True
11    print('three')
12
13 def test_four():
14    assert False
15    print('four')
16
17 def test_five():
18    assert True
19    print('five')
```

PyTest: Multiple Failures output

```
1 test_failures.py .F.F.
```

```
1 $ pytest -v test_failures.py
2
3 test_failures.py::test_one PASSED
4 test_failures.py::test_two FAILED
5 test_failures.py::test_three PASSED
```

```
6 test_failures.py::test_four FAILED
7 test_failures.py::test_five PASSED
```

```
1 $ pytest -s test_failures.py
2
3 one
4 three
5 five
```

PyTest Selective running of test functions

```
1 pytest test_mymod_2.py::test_anagram
2
3 pytest test_mymod_2.py::test_multiword_anagram
```

PyTest: stop on first failure

```
1           pytest -x
2           pytest --maxfail 42
```

Pytest: expect a test to fail (xfail or TODO tests)

Use the `@pytest.mark.xfail` decorator to mark the test.

```
1 from mymod_1 import is_anagram
2 import pytest
3
4 def test_anagram():
5     assert is_anagram("abc", "acb")
6     assert is_anagram("silent", "listen")
7     assert not is_anagram("one", "two")
8
9 @pytest.mark.xfail(reason = "Bug #42")
10 def test_multiword_anagram():
11     assert is_anagram("ana gram", "naga ram")
12     assert is_anagram("anagram", "nag a ram")
```

Pytest: expect a test to fail (xfail or TODO tests)

```
1 $ pytest test_mymod_3.py
```

```
1 ===== test session starts =====
2 platform darwin -- Python 3.5.2, pytest-3.0.7, py-
1.4.33, pluggy-0.4.0
3 Using --random-order-bucket=module
4 Using --random-order-seed=557111
5
6 rootdir:
/Users/gabor/work/training/python/examples/pytest,
inifile:
7 plugins: xdist-1.16.0, random-order-0.5.4
8 collected 2 items
9
10 test_mymod_3.py .x
11
12 ===== 1 passed, 1 xfailed in 0.08 seconds =====
```

PyTest: show xfailed tests with -rx

```
1 $ pytest -rx test_mymod_3.py
```

```
1 ===== test session starts =====
2 platform darwin -- Python 3.5.2, pytest-3.0.7, py-
1.4.33, pluggy-0.4.0
3 Using --random-order-bucket=module
4 Using --random-order-seed=557111
5
6 rootdir:
/Users/gabor/work/training/python/examples/pytest,
inifile:
7 plugins: xdist-1.16.0, random-order-0.5.4
8 collected 2 items
9
10 test_mymod_3.py .x
11
12 ===== short test summary info =====
```

```
13 XFAIL test_mymod_3.py::test_multiword_anagram
14     Bug #42
15
16 ===== 1 passed, 1 xfailed in 0.08 seconds =====
```

Pytest: skipping tests

```
1 import sys
2 import pytest
3
4 @pytest.mark.skipif(sys.platform != 'darwin',
5 reason="Mac tests")
6 def test_mac():
7     assert True
8
9 @pytest.mark.skipif(sys.platform != 'linux',
10 reason="Linux tests")
11 def test_linux():
12     assert True
13
14 @pytest.mark.skipif(sys.platform != 'win32',
15 reason="Windows tests")
16 def test_windows():
17     assert True
18
19 @pytest.mark.skip(reason="To show we can skip tests
20 without any condition.")
21 def test_any():
22     assert True
```

```
1 pytest test_on_condition.py
```

```
1 collected 4 items
2
3 test_on_condition.py ss.s
4
5 ===== 1 passed, 3 skipped in 0.02 seconds =====
```

Pytest: show skipped tests woth -rs

```
1 $ pytest -rs test_on_condition.py
```

```
1 collected 4 items
2
3 test_on_condition.py s.ss
4
5 ===== short test summary info =====
6 SKIP [1] test_on_condition.py:15: To show we can skip
tests without any condition.
7 SKIP [1] test_on_condition.py:7: Linux tests
8 SKIP [1] test_on_condition.py:11: Windows tests
9
10 ===== 1 passed, 3 skipped in 0.03 seconds =====
```

Pytest: show extra test summary info with -r

- (f)ailed
- (E)rror
- (s)kipped
- (x)failed
- (X)passed
- (p)assed
- (P)assed with output
- (a)ll except pP

```
1 pytest -h
```

Pytest: skipping tests output in verbose mode

```
1 $ pytest -v test_on_condition.py
2
3 test_on_condition.py::test_mac PASSED
4 test_on_condition.py::test_any SKIPPED
```

```
5 test_on_condition.py::test_windows SKIPPED
6 test_on_condition.py::test_linux SKIPPED
7
8 ===== 1 passed, 3 skipped in 0.01 seconds =====
```

Pytest verbose mode

```
1 $ pytest -v test_mymod_1.py
2
3 test_mymod_1.py::test_anagram PASSED
```

```
1 $ pytest -v test_mymod_2.py
2
3 test_mymod_2.py::test_anagram PASSED
4 test_mymod_2.py::test_multiword_anagram FAILED
```

Pytest quiet mode

```
1 $ pytest -q test_mymod_1.py
2 .
3 1 passed in 0.01 seconds
```

```
1 $ pytest -q test_mymod_2.py
2
3 .F
4 ===== FAILURES =====
5 ----- test_multiword_anagram -----
6
7     def test_multiword_anagram():
8         assert is_anagram("ana gram", "naga ram")
9 >         assert is_anagram("anagram", "nag a ram")
10 E         AssertionError: assert False
11 E             +   where False = is_anagram('anagram', 'nag a
12 ram')
13 test_mymod_2.py:10: AssertionError
14 1 failed, 1 passed in 0.09 seconds
```

PyTest print STDOUT and STDERR using -s

```
1 import sys
2
3 def test_hello():
4     print("hello testing")
5     print("stderr during testing", file=sys.stderr)
6     assert True
```

```
1 $ pytest -s -q test_stdout_stderr.py
2 hello testing
3 stderr during testing
4 .
5 1 passed in 0.01 seconds
```

PyTest failure reports

- Reporting success is boring
- Reporting failure can be interesting: assert + introspection

PyTest compare numbers

```
1 def double(n):
2     #return 2*n
3     return 2+n
4
5 def test_string_equal():
6     assert double(2) == 4
7     assert double(21) == 42
```

```
1     $ pytest test_number_equal.py
2
3     def test_string_equal():
4         assert double(2) == 4
5 >         assert double(21) == 42
6 E         assert 23 == 42
7 E             + where 23 = double(21)
```

PyTest compare numbers relatively

```
1 def get_number():
2     return 23
3
4 def test_string_equal():
5     assert get_number() < 0
```

```
1 $ pytest test_number_less_than.py
```

```
1     def test_string_equal():
2 >         assert get_number() < 0
3 E         assert 23 < 0
4 E             +  where 23 = get_number()
```

PyTest compare strings

```
1 def get_string():
2     return "abc"
3
4 def test_string_equal():
5     assert get_string() == "abd"
```

```
1 $ pytest test_string_equal.py
```

```
1     def test_string_equal():
2 >         assert get_string() == "abd"
3 E         AssertionError: assert 'abc' == 'abd'
4 E             - abc
5 E             + abd
```

PyTest compare long strings

```
1 import string
2
3 def get_string(s):
4     return string.printable + s + string.printable
```

```
5
6 def test_long_strings():
7     assert get_string('a') == get_string('b')
```

```
1 $ pytest test_long_strings.py
```

```
1     def test_long_strings():
2 >         assert get_string('a') == get_string('b')
3 E         AssertionError: assert
'0123456789ab...t\n\r\x0b\x0c' == '0123456789abc...t\\\
4 n\r\x0b\x0c'
5 E             Skipping 90 identical leading characters in
diff, use -v to show
6 E             Skipping 91 identical trailing characters in
diff, use -v to show
7 E             { | }~
8 E
9 E             - a012345678
10 E             ? ^
11 E             + b012345678
12 E             ? ^
```

PyTest is one string in another strings

Shows ~250 characters

```
1 import string
2
3 def get_string():
4     return string.printable * 30
5
6 def test_long_strings():
7     assert 'hello' in get_string()
```

```
1     def test_long_strings():
2 >         assert 'hello' in get_string()
3 E         assert 'hello' in
'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
U\
4 VWXYZ!"#$%&\'()*+,-.:/;<=>?@[\\" ]^_`{|}~
```

```
\t\n\r\x0b\x0c012345...x0b\x0c0123456789abcd\
5
efghijklmnopqrstuvwxyzABCDEFGHIJKLMNPQRSTUVWXYZ!"#$%&\'(\
)*+,-./:;<=>?@[\\"[^_`{|}~\\\
6 t\n\r\x0b\x0c'
7 E           +   where
'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNPQRSTUVWXYZ!
8 $"%&\'(()*+,-./:;<=>?@[\\"[^_`{|}~\\
\t\n\r\x0b\x0c012345...x0b\x0c0123456789abcdefghijkl\
9
mnopqrstuvwxyzABCDEFGHIJKLMNPQRSTUVWXYZ!"#$%&\'(()*+,-./:\
;=>?@[\\"[^_`{|}~\t\n\r\x0c\
10 b\x0c' = get_string()
```

PyTest test any expression

```
1 def test_expression_equal():
2     a = 3
3     assert a % 2 == 0
```

```
1 $ pytest test_expression_equal.py
2
3     def test_expression_equal():
4         a = 3
5 >         assert a % 2 == 0
6 E         assert (3 % 2) == 0
```

PyTest element in list

```
1 def get_list():
2     return ["monkey", "cat"]
3
4 def test_in_list():
5     assert "dog" in get_list()
```

```
1 $ pytest test_in_list.py
2
3     def test_in_list():
```

```
4 >         assert "dog" in get_list()
5 E             AssertionError: assert 'dog' in ['monkey',
'cat']
6 E                 +  where ['monkey', 'cat'] = get_list()
```

PyTest compare lists

```
1 import string
2 import re
3
4 def get_list(s):
5     return list(string.printable + s +
string.printable)
6
7 def test_long_lists():
8     assert get_list('a') == get_list('b')
```

```
1 $ pytest test_lists.py
2
3     def test_long_lists():
4 >         assert get_list('a') == get_list('b')
5 E             AssertionError: assert ['0', '1', '2...', '4',
'5', ...]
6                     == ['0', '1', '2...', '4', '5', ...]
7 E             At index 100 diff: 'a' != 'b'
8 E             Use -v to get the full diff
```

PyTest compare short lists

```
1 import string
2 import re
3
4 def get_lista():
5     return 'a', 'b', 'c'
6 def get_listx():
7     return 'x', 'b', 'y'
8
9 def test_short_lists():
10    assert get_lista() == get_listx()
```

```
1 $ pytest test_short_lists.py
```

```
1     def test_short_lists():
2 >         assert get_lista() == get_listx()
3 E         AssertionError: assert ('a', 'b', 'c') == ('x',
'b', 'y')
4 E             At index 0 diff: 'a' != 'x'
5 E             Use -v to get the full diff
```

PyTest compare short lists - verbose output

```
1 $ pytest -v test_short_lists.py
```

```
1     def test_short_lists():
2 >         assert get_lista() == get_listx()
3 E         AssertionError: assert ('a', 'b', 'c') == ('x',
'b', 'y')
4 E             At index 0 diff: 'a' != 'x'
5 E             Full diff:
6 E             - ('a', 'b', 'c')
7 E             ?   ^       ^
8 E             + ('x', 'b', 'y')
9 E             ?   ^       ^
```

PyTest compare dictionaries

```
1 import string
2 import re
3
4 def get_dictionary(k, v):
5     d = dict([x, ord(x)] for x in string.printable)
6     d[k] = v
7     return d
8
9 def test_big_dictionary_different_value():
10    assert get_dictionary('a', 'def') ==
get_dictionary('a', 'abc')
11
12 def test_big_dictionary_differnt_keys():
```

```
13     assert get_dictionary('abc', 1) ==
get_dictionary('def', 2)
```

PyTest compare dictionaries output

```
1 $ pytest test_dictionaries.py
2
3     test_big_dictionary_different_value
-----
4
5     def test_big_dictionary_different_value():
6 >         assert get_dictionary('a', 'def') ==
get_dictionary('a', 'abc')
7 E         AssertionError: assert {'\t': 9, '\n...x0c':
12, ...}
8             == {'\t': 9, '\n'...x0c': 12, ...}
9 E             Omitting 99 identical items, use -v to show
10 E             Differing items:
11 E                 {'a': 'def'} != {'a': 'abc'}
12 E                 Use -v to get the full diff
13
14     test_big_dictionary_differnt_keys
-----
15
16     def test_big_dictionary_differnt_keys():
17 >         assert get_dictionary('abc', 1) ==
get_dictionary('def', 2)
18 E         AssertionError: assert {'\t': 9, '\n...x0c':
12, ...}
19             == {'\t': 9, '\n'...x0c': 12, ...}
20 E             Omitting 100 identical items, use -v to show
21 E             Left contains more items:
22 E                 {'abc': 1}
23 E             Right contains more items:
24 E                 {'def': 2}
25 E                 Use -v to get the full diff
```

PyTest Fixtures

- In generally we call [test fixture](#) the environment in which a test is expected to run.
- Pytest uses the same word for a more generic concept. All the techniques that make it easy to set up the environment and to tear it down after the tests.

PyTest Fixture setup and teardown

```
1 def setup_module():
2     print("setup_module")
3
4 def teardown_module():
5     print("teardown_module")
6
7
8 def setup_function():
9     print("    setup_function")
10
11 def teardown_function():
12     print("    teardown_function")
13
14
15 def test_one():
16     print("        test_one")
17     assert True
18     print("        test_one after")
19
20 def test_two():
21     print("        test_two")
22     assert False
23     print("        test_two after")
24
25 def test_three():
26     print("        test_three")
27     assert True
28     print("        test_three after")
```

See next slide for the output.

PyTest Fixture setup and teardown output

```
1 test_fixture.py .F.
```

```
1 $ pytest test_fixture.py -s
2
3 setup_module
4
5     setup_function
6         test_one
7         test_one after
8     teardown_function
9
10    setup_function
11        test_two
12    teardown_function
13
14    setup_function
15        test_three
16        test_three after
17    teardown_function
18
19 teardown_module
```

Note, the `teardown_function` is executed even after failed tests.

PyTest: Class setup and teardown

```
1 class TestClass():
2     def setup_class(self):
3         print("setup_class called once for the class")
4
5     def teardown_class(self):
6         print("teardown_class called once for the
class")
7
8
9     def setup_method(self):
10         print("setup_method called for every method")
11
```

```
12     def teardown_method(self):
13         print("teardown_method called for every
method")
14
15
16     def test_one(self):
17         print("one")
18         assert True
19         print("one after")
20
21     def test_two(self):
22         print("two")
23         assert False
24         print("two after")
25
26     def test_three(self):
27         print("three")
28         assert True
29         print("three after")
```

PyTest: Class setup and teardown output

```
1 $ pytest -s test_class.py
2
3 setup_class called once for the class
4
5 setup_method called for every method
6 one
7 one after
8 teardown_method called for every method
9
10 setup_method called for every method
11 two
12 teardown_method called for every method
13
14 setup_method called for every method
15 three
16 three after
17 teardown_method called for every method
18
19 teardown_class called once for the class
```

Pytest Dependency injection

```
1 def function(thingy):  
2     pass
```

1. Find function.
2. Check parameters of the function.
3. Create the appropriate instances.
4. Call the function with the instances.

Pytest fixture - tmpdir

```
1 import os  
2  
3  
4 def test_something(tmpdir):  
5     print(tmpdir)      #  
/private/var/folders/ry/z60xxmw0000gn/T/pytest-of-  
gabor/pyt\  
6 est-14/test_read0  
7  
8     d = tmpdir.mkdir("subdir")  
9     fh = d.join("config.ini")  
10    fh.write("Some text")  
11  
12    filename = os.path.join( fh.dirname, fh.basename )  
13  
14    temp_dir = str(tmpdir)  
15  
16    # ...
```

Pytest capture STDOUT and STDERR with capsys

Captures everything that is printed to STDOUT and STDERR so we can compare that to the expected output and error.

```
1 import sys
2
3 def greet(to_out, to_err=None):
4     print(to_out)
5     if to_err:
6         print(to_err, file=sys.stderr)
7
8
9 def test_myoutput(capsys):
10    greet("hello", "world")
11    out, err = capsys.readouterr()
12    assert out == "hello\n"
13    assert err == "world\n"
14
15    greet("next")
16    out, err = capsys.readouterr()
17    assert out == "next\n"
```

Pytest Fixture - home made fixtures

```
1 import pytest
2 import application
3
4
5 @pytest.fixture()
6 def getapp():
7     print('getapp starts')
8     app = application.App()
9
10    yield app
11
12    app.shutdown()
13    print('getapp ends')
14
15 def test_add_user_foo(getapp):
16     getapp.add_user("Foo")
17     assert True
18
19 def test_add_user_bar(getapp):
20     getapp.add_user("Bar")
21     assert True
```

```
1 class App:
2     def __init__(self):
3         self.pi = 3.14
4         # .. set up database
5         print("__init__ of App")
6
7
8     def shutdown(self):
9         print("shutdown of App cleaning up database")
10
11
12     def add_user(self, name):
13         print("Working on add_user({})".format(name))
```

```
1 $ pytest -s -q fixtures.py
2
3 getapp starts
4 __init__ of App
5 Working on add_user(Bar)
6 .shutdown of App cleaning up database
7 getapp ends
8
9 getapp starts
10 __init__ of App
11 Working on add_user(Foo)
12 .shutdown of App cleaning up database
13 getapp ends
```

More fixtures

```
1 import pytest
2
3 @pytest.fixture(autouse = True, scope="module")
4 def fix_module():
5     print("\nFix module setup")
6     yield
7     print("\nFix module teardown")
8
9
10 @pytest.fixture(autouse = True, scope="function")
11 def fix_function():
```

```
12     print("\nFix function setup")
13     yield
14     print("\nFix function teardown")
15
16
17 @pytest.fixture()
18 def blue():
19     print("\nFix blue setup")
20     yield
21     print("\nFix blue teardown")
22
23 @pytest.fixture()
24 def green():
25     print("\nFix green setup")
26     yield
27     print("\nFix green teardown")
28
29
30 def test_one(blue, green):
31     print("Test one")
32
33
34 def test_two(green, blue):
35     print("Test two")
```

```
1 ===== test session starts =====
2 platform linux -- Python 3.7.3, pytest-5.1.1, py-
1.8.0, pluggy-0.13.0 -- /home/gabor\
3 /venv3/bin/python3
4 cachedir: .pytest_cache
5 rootdir:
6 /home/gabor/work/slides/python/examples/pytest
7 plugins: flake8-1.0.4
8 collecting ... collected 2 items
9
10 more_fixtures.py::test_one
11 Fix module setup
12
13 Fix function setup
14
15 Fix blue setup
16
17 Fix green setup
```

```
17 Test one
18 PASSED
19 Fix green teardown
20
21 Fix blue teardown
22
23 Fix function teardown
24
25 more_fixtures.py::test_two
26 Fix function setup
27
28 Fix green setup
29
30 Fix blue setup
31 Test two
32 PASSED
33 Fix blue teardown
34
35 Fix green teardown
36
37 Fix function teardown
38
39 Fix module teardown
40
41
42 ====== 2 passed in 0.01s =====
```

- We can't add fixtures to test_functions as decorators (as I think was the case in NoseTest), we need to use dependency injection.

Pytest: Mocking - why?

- Independent testing environment.
- Faster tests (mock remote calls, mock whole database)
- Fake some code/application/API that does not exist yet.
- Test error conditions in a system not under our control.

Pytest: Mocking - what?

- External dependency (e.g. an API)
- STDIN/STDOUT/STDERR
- Random values
- Methods of a database

Pytest: One dimensional spacefight

```
1 import random
2
3 def play():
4     debug = False
5     move = False
6     while True:
7         print("\nWelcome to another Number Guessing
game")
8         hidden = random.randrange(1, 201)
9         while True:
10            if debug:
11                print("Debug: ", hidden)
12
13            if move:
14                mv = random.randrange(-2, 3)
15                hidden = hidden + mv
16
17            user_input = input("Please enter your
guess [x|s|d|m|n]: ")
18            print(user_input)
19
20            if user_input == 'x':
21                print("Sad to see you leave early")
22                return
23
24            if user_input == 's':
25                print("The hidden value is ", hidden)
26                continue
27
28            if user_input == 'd':
29                debug = not debug
```

```

30         continue
31
32     if user_input == 'm':
33         move = not move
34         continue
35
36     if user_input == 'n':
37         print("Giving up, eh?")
38         break
39
40     guess = int(user_input)
41     if guess == hidden:
42         print("Hit!")
43         break
44
45     if guess < hidden:
46         print("Your guess is too low")
47     else:
48         print("Your guess is too high")
49
50
51 if __name__ == '__main__':
52     play()

```

Pytest: Mocking input and output

```

1 import game
2
3 def test_immediate_exit():
4     input_values = ['x']
5     output = []
6
7     def mock_input(s):
8         output.append(s)
9         return input_values.pop(0)
10    game.input = mock_input
11    game.print = lambda s : output.append(s)
12
13    game.play()
14
15    assert output == [
16        '\nWelcome to another Number Guessing game',

```

```
17         'Please enter your guess [x|s|d|m|n]: ',
18         'x',
19         'Sad to see you leave early',
20     ]
```

Pytest: Mocking random

```
1 import game
2 import random
3
4 def test_immediate_exit():
5     input_values = ['30', '50', '42', 'x']
6     output = []
7
8     def mock_input(s):
9         output.append(s)
10        return input_values.pop(0)
11    game.input = mock_input
12    game.print = lambda s : output.append(s)
13    random.randrange = lambda a, b : 42
14
15    game.play()
16
17    assert output == [
18        '\nWelcome to another Number Guessing game',
19        'Please enter your guess [x|s|d|m|n]: ',
20        '30',
21        'Your guess is too low',
22        'Please enter your guess [x|s|d|m|n]: ',
23        '50',
24        'Your guess is too high',
25        'Please enter your guess [x|s|d|m|n]: ',
26        '42',
27        'Hit!',
28        '\nWelcome to another Number Guessing game',
29        'Please enter your guess [x|s|d|m|n]: ',
30        'x',
31        'Sad to see you leave early',
32    ]
```

Pytest: Flask echo

```
1 from flask import Flask, request
2 eapp = Flask(__name__)
3
4 @eapp.route("/")
5 def hello():
6     return '''
7 <form action="/echo" method="GET">
8 <input name="text">
9 <input type="submit" value="Echo">
10 </form>
11 '''
12
13 @eapp.route("/echo")
14 def echo():
15     answer = request.args.get('text')
16     if answer:
17         return "You said: " + answer
18     else:
19         return "Nothing to say?"
20
21
22 if __name__ == "__main__":
23     eapp.run()
```

Pytest: testing Flask echo

```
1 import flask_echo
2
3 class TestEcho:
4     def setup_method(self):
5         self.app = flask_echo.eapp.test_client()
6         print("setup")
7
8     def test_main(self):
9         rv = self.app.get('/')
10        assert rv.status == '200 OK'
11        assert b'<form action="/echo" method="GET">' in rv.data
12
13    def test_echo(self):
14        rv = self.app.get('/echo?text=Hello')
15        assert rv.status == '200 OK'
```

```
16         assert b'You said: Hello' in rv.data
17
18     def test_empty_echo(self):
19         rv = self.app.get('/echo')
20         assert rv.status == '200 OK'
21         assert b'Nothing to say?' in rv.data
```

PyTest: Run tests in parallel with xdist

```
1             $ pip install pytest-xdist
2             $ pytest -n NUM
```

PyTest: Order of tests

Pytest runs the test in the same order as they are found in the test module:

```
1 def test_one():
2     assert True
3
4 def test_two():
5     assert True
6
7 def test_three():
8     assert True
```

```
1 test_order.py::test_one PASSED
2 test_order.py::test_two PASSED
3 test_order.py::test_three PASSED
```

PyTest: Randomize Order of tests

Install [pytest-random-order](#)

```
1 pip install pytest-random-order
```

And from now on all the test will run in a random order.

PyTest: Force default order

If for some reason we would like to make sure the order remains the same,
we can add the following two lines of code.

```
1 import pytest
2 pytestmark = pytest.mark.random_order(disabled=True)
```

```
1 import pytest
2 pytestmark = pytest.mark.random_order(disabled=True)
3
4 def test_one():
5     assert True
6
7 def test_two():
8     assert True
9
10 def test_three():
11     assert True
```

PyTest: no random order

```
1 pytest -p no:random-order -v
```

Anagram on the command line

```
1 from mymod_1 import is_anagram
2 import sys
3
4 if len(sys.argv) != 3:
5     exit("Usage {} STR STR".format(sys.argv[0]))
6
7 print(is_anagram(sys.argv[1], sys.argv[2]))
```

PyTest testing CLI

```
1 import subprocess
2
3 def capture(command):
4     proc = subprocess.Popen(command,
5         stdout = subprocess.PIPE,
6         stderr = subprocess.PIPE,
7     )
8     out,err = proc.communicate()
9     return out, err, proc.returncode
10
11
12 def test_anagram_no_param():
13     command = ["python3",
14     "examples/pytest/anagram.py"]
15     out, err, exitcode = capture(command)
16     assert exitcode == 1
17     assert out == b''
18     assert err == b'Usage examples/pytest/anagram.py
STR STR\n'
19
20 def test_anagram():
21     command = ["python3",
22     "examples/pytest/anagram.py", "abc", "cba"]
23     out, err, exitcode = capture(command)
24     assert exitcode == 0
25     assert out == b'True\n'
26     assert err == b''
27
28 def test_no_anagram():
29     command = ["python3",
30     "examples/pytest/anagram.py", "abc", "def"]
31     out, err, exitcode = capture(command)
32     assert exitcode == 0
33     assert out == b'False\n'
34     assert err == b''
```

PyTest test discovery

Running `py.test` will find test files and in the files test functions.

- test_*.py files
 - *_test.py files
 - test_* functions
 - ...
-

```
1 $ pytest
2 ===== test session starts
=====
3 platform darwin -- Python 2.7.5 -- py-1.4.20 --
pytest-2.5.2
4 collected 3 items
5
6 test_fibo.py F
7 test_fibonacci.py F
8 test_fibonacci_ok.py .
9
10 ===== FAILURES
=====
11 -----
12
13     def test_fibo():
14         assert mymath.fibo(1) == [1]
15         assert mymath.fibo(2) == [1, 1]
16 >       assert mymath.fibo(3) == [1, 1, 2]
17 E           assert [1, 1, 5] == [1, 1, 2]
18 E               At index 2 diff: 5 != 2
19
20 test_fibo.py:6: AssertionError
21 -----
22
23     def test_fibonacci():
24         assert mymath.fibonacci(1) == 1
25         assert mymath.fibonacci(2) == 1
26 >       assert mymath.fibonacci(3) == 2
27 E           assert 5 == 2
28 E               + where 5 = <function fibonacci at
0x107f90488>(3)
29 E               + where <function fibonacci at
0x107f90488> = mymath.fibonacci
30
31 test_fibonacci.py:6: AssertionError
```

```
32 ===== 2 failed, 1 passed in 0.04
seconds =====
```

PyTest test discovery - ignore some tests

```
1 $ pytest
2
3
4 $ pytest --ignore venv3/
```

```
1 test_mymod_1.py .
2 test_mymod_2.py .F
```

- test_*.py files
- *_test.py files
- TestClasses
- test_* functions
- ...

PyTest select tests by name

- –collect-only - only list the tests, don't run them yet.
- -k select by name

```
1 def test_database_read():
2     assert True
3
4 def test_database_write():
5     assert True
6
7 def test_database_forget():
8     assert True
9
10 def test_ui_access():
11     assert True
12
```

```
13 def test_ui_forget():
14     assert True

---


1 pytest --collect-only -k database test_by_name.py
2     test_database_forget
3     test_database_read
4     test_database_write

---


1 pytest --collect-only -k ui test_by_name.py
2     test_ui_access
3     test_ui_forget

---


1 pytest --collect-only -k forget test_by_name.py
2     test_database_forget
3     test_ui_forget

---


1 pytest --collect-only -k "forget or read"
test_by_name.py
2     test_database_read
3     test_database_forget
4     test_ui_forget
```

PyTest select tests by marker

Use the `@pytest.mark.name` decorator to tag the tests.

```
1 import pytest
2
3 @pytest.mark.smoke
4 def test_database_read():
5     assert True
6
7 @pytest.mark.security
8 @pytest.mark.smoke
9 def test_database_write():
10    assert True
11
12 @pytest.mark.security
13 def test_database_forget():
```

```
14     assert True
15
16 @pytest.mark.smoke
17 def test_ui_access():
18     assert True
19
20 @pytest.mark.security
21 def test_ui_forget():
22     assert True
```

```
1 pytest --collect-only -m security test_by_marker.py
2     test_ui_forget
3     test_database_write
4     test_database_forget
```

```
1 pytest --collect-only -m smoke test_by_marker.py
2     test_database_read
3     test_ui_access
4     test_database_write
```

PyTest: Test Coverage

```
1 pip install pytest-cov
2
3 pytest --cov=my --cov-report html --cov-branch
4
5 Open htmlcov/index.html
```

Try werkzeug

```
1 pytest --cov=werkzeug --cov-report html --cov-branch
2 xdg-open htmlcov/index.html
```

Exercise: module

Pick one of the modules and write a test for it.

- [algo](#)
- [editdistance](#) Levenshtein distance implemented in C
- [python-Levenshtein](#) implemented in C
- [pylev](#)
- [pyxdameraulevenshtein](#)
- [weighted-levenshtein](#)
- OpenPyXL

Exercise: Open Source

- Visit the [stats](#) on PyDigger.com
- List the packages that [have GitHub no Travis-CI](#).
- Pick one that sounds simple. Visit its GitHub page and check if it has tests.
- If it does not, write one.
- Send Pull Request

Pytest resources

- [pytest.org](#)
- [Python Testing with pytest by Brian Okken](#) (The Pragmatic Bookshelf)
- [Python Testing by Brian Okken](#)
- [Talk Python to me by Michael Kennedy](#)
- [Python Bytes](#) podcast by Brian Okken and Michael Kennedy

Pytest and tempdir

```

1 import re
2
3 def parse_file(filename):
4     data = {}
5     with open(filename) as fh:
```

```

6     for row in fh:
7         row = row.rstrip("\n")
8         if re.search(r'=', row):
9             k, v = re.split(r'\s*=\s*', row)
10            data[k] = v
11        else:
12            pass # error reporting?
13    return data
14
15 def save_file(filename, data):
16     with open(filename, 'w') as fh:
17         for k in data:
18             fh.write("{}={}\n".format(k, data[k]))
19
20 if __name__ == '__main__':
21     print(parse_file('a.cfg'))

```

```

1 name=Foo Bar
2 email = foo@bar.com

```

```

1 import mycfg
2 import os
3
4 class TestMe:
5     def test_parse(self):
6         data = mycfg.parse_file('a.cfg')
7         assert data, {
8             'name' : 'Foo Bar',
9             'email' : 'foo@bar.com',
10            }
11
12     def test_example(self, tmpdir):
13         original = {
14             'name' : 'My Name',
15             'email' : 'me@home.com',
16             'home' : '127.0.0.1',
17            }
18         filename = str(tmpdir.join('abc.cfg'))
19         assert not os.path.exists(filename)
20         mycfg.save_file(filename, original)
21         assert os.path.exists(filename)

```

```
22         new = mycfg.parse_file(filename)
23         assert new == original
```

PyTest compare short lists - output

```
1 import configparser
2 import os
3
4
5 def test_read_ini(tmpdir):
6     print(tmpdir)      #

7 est-14/test_read0
8     d = tmpdir.mkdir("subdir")
9     fh = d.join("config.ini")
10    fh.write("""
11 [application]
12 user = foo
13 password = secret
14 """)
15
16    print(fh.basename) # data.txt
17    print(fh.dirname)  #

18 est-14/test_read0/subdir
19     filename = os.path.join( fh.dirname, fh.basename )
20
21     config = configparser.ConfigParser()
22     config.read(filename)
23
24     assert config.sections() == ['application']
25     assert config['application'], {
26         "user" : "foo",
27         "password" : "secret"
28     }
```

PyTest with parameter

```
1 import pytest
2
3 @pytest.mark.parametrize("name", ["Foo", "Bar"])
4 def test_cases(name):
5     print(f"name={name}")
6     assert len(name) == 3
```

```
1 ===== test session starts =====
2 platform linux -- Python 3.7.3, pytest-5.3.2, py-
1.8.0, pluggy-0.13.0
3 rootdir: /home/gabor/work/slides/python-
programming/examples/pytest
4 plugins: flake8-1.0.4
5 collected 2 items
6
7 test_with_param.py name=Foo
8 .name=Bar
9 .
10
11 ===== 2 passed in 0.00s =====
```

PyTest with parameters

```
1 import pytest
2
3 @pytest.mark.parametrize("name, email", [
4     ("Foo", "foo@email.com"),
5     ("Bar", "bar@email.com"),
6 ])
7 def test_cases(name, email):
8     print(f"name={name} email={email}")
9     assert email.lower().startswith(name.lower())
```

```
1 ===== test session starts =====
2 platform linux -- Python 3.7.3, pytest-5.3.2, py-
1.8.0, pluggy-0.13.0
3 rootdir: /home/gabor/work/slides/python-
programming/examples/pytest
4 plugins: flake8-1.0.4
5 collected 2 items
```

```
6
7 test_with_params.py name=Foo email=foo@email.com
8 .name=Bar email=bar@email.com
9 .
10
11 ===== 2 passed in 0.01s =====
```

Pytest reporting in JUnit XML format

```
1 pytest --junitxml report.xml
```

- [pytest-json-report](#)

```
1 pip install pytest-json-report
2
3 pytest --json-report --json-report-file=report.json
```

Recommended to also add

```
1 --json-report-omit=log
```

```
1 pytest -s --json-report --json-report-file=report.json
--log-cli-level=INFO
```

```
1 import logging
2
3 def add(x, y):
4     #     logger = logging.getLogger("mytest")
5     logging.basicConfig(level = logging.INFO)
6     logging.info("Just some info log")
7     return x * y
8
9 def test_one():
10     assert add(2, 2) == 4
```

No test selected

If you run pytest and it cannot find any tests, for example because you used some selector and not test matched it, then Pytest will exit with exit code 5.

This is considered a failure by every tool, including Jenkins and other CI systems.

On the other hand you won't see any failed test reported. After all if no tests are run, then none of them fails. This can be confusing.

Advanced functions

Variable scopes

- Local (inside a def)
- Enclosing (in the enclosing def, aka. nonlocal)
- Global (outside of all defs)

Name resolution order (LEGB)

1. Local
2. Enclosing
3. Global
4. Built-in

Scoping: global seen from function

```
1 a = 42
2 def f():
3     print(a)
4
5 f()
```

42

Assignment creates local scope

```
1 a = 42
2 def f():
3     a = 23
4     print(a)
```

```
5
6 print('ok')
7 print(a)
8 f()
9 print(a)
```

```
1 ok
2 42
3 23
4 42
```

Local scope gone wrong

```
1 a = 42
2 def f():
3     print(a)
4     a = 23
5
6 print('ok')
7 print(a)
8 f()
9 print(a)
```

```
1 ok
2 42
3 Traceback (most recent call last):
4   File "scoping_external_variable.py", line 8, in
<module>
5     f()
6   File "scoping_external_variable.py", line 3, in f
7     print(a)
8 UnboundLocalError: local variable 'a' referenced before
assignment
```

Accessing a global variable inside a function works, but if I change it (make it refer to another piece of data), then it is disallowed. If I only change the data inside (for mutable variables), that works, but is a bad practice.

Changing global variable from a function

```
1 a = 42
2 def f():
3     global a
4     print(a)
5     a = 23
6
7 print(a)    # 42
8 f()         # 42
9 print(a)    # 23
```

Does not need to be created outside

```
1 def f():
2     global a
3     a = 23
4
5 f()
6 print(a)    # 23
```

Global variables mutable in functions

```
1 a = [2]
2
3 def f():
4     print(a)          # [2]
5     a.append(3)
6     print(a)          # [2, 3]
7     a[0] = 4
8
9 f()
10 print(a)           # [4, 3]
```

Scoping issues

```
1 text = ['aaaa', 'bb', 'ccc ccc']
2
3 length_1 = [ len(s) for s in text ]
```

```
4 print(length_1) # [4, 2, 7]
5
6
7 length_2 = [ len(s) for x in text ]
8 print(length_2) # [7, 7, 7]
```

List comprehensions don't create their own scope!

sub in sub

Functions can be defined inside functions.

```
1 def f():
2     print("in f")
3     def g():
4         print("in g")
5     g()
6
7 f()
8 #g() # does not exist here
```

They are scoped locally

Scoping sub in sub (enclosing scope)

```
1 def external_func():
2     the_answer = 42
3
4     def func(args):
5         print(args, "the_answer:", the_answer)
6
7         # the_answer = 'what was the question?'
8         # enabling this would give:
9         # UnboundLocalError: local variable
10        'the_answer'
11        #           referenced before assignment
12
13        func("first")
14        func("second")
```

```
14  
15 external_func()
```

```
1 first the_answer: 42  
2 second the_answer: 42
```

Function objects

```
1 The difference between  
2 x = foo  
3 y = foo()
```

```
1 c = 0  
2  
3 def foo():  
4     global c  
5     c += 1  
6     return c  
7  
8  
9 print(foo())    # 1  
10 print(foo())   # 2  
11 x = foo        # assigning the function object  
12 y = foo()      # assigning the return value of the  
function  
13 print(foo())   # 4  
14 print(x())     # 5  
15 print(y)       # 3
```

Functions are created at run time

def and class are run-time
Everything is runtime. Even compilation is runtime.

foo() will return a random value every time, but when
bar is defined it freezes the specific value that foo

returned when bar was created.

```
1 import random
2
3 def foo():
4     return random.random()
5
6
7 print(foo())
8 print(foo())
9
10 def bar(a, b = foo()):
11     return [a, b]
12
13 print(bar(1))
14 print(bar(2))
```

```
1 0.0756804810689
2 0.350692064113
3 [1, 0.7401995987184571]
4 [2, 0.7401995987184571]
```

Mutable default

The default list assigned to b is created when the f functions is defined.

After that, each call to f() (that does not get a “b” parameter) uses this common list.

```
1 def f(a, b = []):
2     b.append(a)
3     return b
4
5 print(f(1))
6 print(f(2))
7 print(f(3))
```

```
1 [1]
2 [1, 2]
3 [1, 2, 3]
```

Use None instead:

Use None as default parameter

```
1 def f(a, b = None):
2     if b == None:
3         b = []
4     b.append(a)
5     return b
6
7 print(f(1))
8 print(f(2))
9 print(f(3))
```

```
1 [1]
2 [2]
3 [3]
```

Inner function created every time the outer function runs

Also defined during run-time, but in every call of bar() the inner_func is redefined again and again.

```
1 import random
2
3 def foo():
4     return random.random()
```

```
5
6 print(foo())
7 print(foo())
8
9 def bar(a, b = foo()):
10
11     def inner_func(x, y = foo()):
12         return [x, y]
13
14     print('inner', inner_func(a))
15     return [a, b]
16
17 print(bar(1))
18 print(bar(2))
```

```
1 0.821210904648
2 0.925337844251
3 inner [1, 0.9243163421154859]
4 [1, 0.38535850141949013]
5 inner [2, 0.5665772632462458]
6 [2, 0.38535850141949013]
```

Static variable

There are no function-level static variables in Python, but you can fake it quite easily

```
1 def counter():
2     if 'cnt' not in counter.__dict__:
3         counter.cnt = 0
4     counter.cnt += 1
5     return counter.cnt
6
7 print(counter())      # 1
8 print(counter())      # 2
9 print(counter())      # 3
10
11 print(counter.cnt)    # 3
12
13 counter.cnt = 6
14 print(counter())      # 7
```

Static variable in generated function

```
1 def create():
2     def func():
3         func.cnt += 1
4         return func.cnt
5     func.cnt = 0
6     return func
7
8 a = create()
9 b = create()
10 print(a())      # 1
11 print(a())      # 2
12 print(b())      # 1
13 print(a())      # 3
14
15 b.cnt = 7
16 print(a.cnt)   # 3
17 print(b.cnt)   # 7
```

Inspect

The [inspect](#) module provides introspection to Python runtime. `inspect.stack` returns the stack-trace. Element 0 is the deepest (where we called `inspect.stack`).
Each level has several values. A representation of the frame, filename, linenum, subroutine-name.

```
1 import inspect
2 import sys
3
4 level = int(sys.argv[1])
5
6
7 def f():
8     print("in f before g")
9     g()
10    print("in f after g")
11
12 def g():
13
```

```
13     print("in g")
14     PrintFrame()
15
16
17 def PrintFrame():
18     st = inspect.stack()
19
20     frame = st[level][0]
21     info = inspect.getframeinfo(frame)
22     print('__file__:', info.filename)
23     print('__line__:', info.lineno)
24     print('__function__:', info.function)
25
26     print('* file', st[level][1])
27     print('* line', st[level][2])
28     print('* sub', st[level][3])
29
30 f()
```

python caller.py 1

```
1 in f before g
2 in g
3 __file__:      caller.py
4 __line__:      15
5 __function__:  g
6 * file caller.py
7 * line 15
8 * sub g
9 in f after g
```

Variable number of function arguments

Python function arguments - a reminder

- Order of parameter
- Arguments with default values are optional (and come at the end of the definition)
- Number of arguments is known at the time of function definition. The only flexibility is provided by the optional arguments.

```
1 def f(a, b = 42):
2     print(a)
3     print(b)
4
5 f(23)
6     # 23
7     # 42
8
9 f(19, 11)
10    # 19
11    # 11
12
13 f(b=7, a=8)
14    # 8
15    # 7
16
17 # f()          # (runtime) TypeError: f() takes at
18 least 1 argument (0 given)
19 # f(1, 2, 3)  # (runtime) TypeError: f() takes at
most 2 arguments (3 given)
20 # f(b=10, 23) # SyntaxError: non-keyword arg after
keyword arg
21
22 # def g(a=23, b):
23 #     pass
```

```
23 #      SyntaxError: non-default argument follows
default argument
```

Functions with unknown number of arguments

- `sum(a, b, c, ...)`
- `reduce(function, a, b, c, ...)`
- `report(function, foo = 23, bar = 19, moo = 70, ...)`
- `report(function, a, b, c, ..., foo = 23, bar = 19, moo = 70, ...)`

Variable length argument list with * and **

```
1 def f(a, b=1, *args, **kwargs):
2     print('a:', a)
3     print('b:', b)
4     print('args:', args)
5     print('kwargs:', kwargs)
6     return a + b
7
8 f(2, 3, 4, 5, c=6, d=7)
9 print()
10 f(2, c=5, d=6)
11 print()
12 f(10)
```

```
1 a:      2
2 b:      3
3 args:   (4, 5)
4 kwargs: {'c': 6, 'd': 7}
5
6 a:      2
7 b:      1
8 args:   ()
9 kwargs: {'c': 5, 'd': 6}
10
11 a:      10
```

```
12 b:      1
13 args:   ()
14 kwargs: { }
```

Passing arguments as they were received (but incorrectly)

What if we need to pass the list of individual arguments (or pairs) to another function?

```
1 def f(*args, **kwargs):
2     print('f args: ', args)
3     print('f kwargs: ', kwargs)
4     g(args, kwargs)
5
6 def g(*args, **kwargs):
7     print('g args: ', args)
8     print('g kwargs: ', kwargs)
9
10 f(1, 2, a=3, b=4)
```

```
1 f args: (1, 2)
2 f kwargs: {'a': 3, 'b': 4}
3 g args: ((1, 2), {'a': 3, 'b': 4})
4 g kwargs: {}
```

g() received 2 individual parameters, the first was a tuple, the second a dictionary

Unpacking args before passing them on

```
1 def f(*args, **kwargs):
2     print('f: ', args)
3     print('f: ', kwargs)
4     g(*args, **kwargs)
5
6 def g(*args, **kwargs):
7     print('g: ', args)
```

```
8     print('g: ', kwargs)
9
10 f(1, 2, a=3, b=4)
```

```
1 f:  (1, 2)
2 f:  {'a': 3, 'b': 4}
3 g:  (1, 2)
4 g:  {'a': 3, 'b': 4}
```

Exercise: implement the my_sum function

- my_sum should be able to accept any number of values and return their sum.
- my_sum() should return 0 or None. Decide yourself!
- my_sum(2, 3) should return 5. etc.

Solution: implement the my_sum function

```
1 def my_sum(*numbers):
2     s = 0
3     for n in numbers:
4         s += n
5     return s
6
7 print(my_sum())          # 0
8 print(my_sum(2, 3))      # 5
9 print(my_sum(-1, 2, -1,)) # 0
```

Exercise: implement the reduce function

```
1 my_reduce(function, a, b, c, ...)
```

- ‘function’ is expected to be a function that receives two arguments and returns a result.
- If only the function is given, return None.

- If only one value is given, return that value.
- Take the first two values, run the function on them. Then take the result and the next value and run the function on them. etc. When no more values are left, return the last result.

```
1 # print(my_reduce()) # TypeError: my_reduce() takes at least 1 argument (0 given)
2 print(my_reduce(lambda x,y: x+y)) # None
3 print(my_reduce(lambda x,y: x+y, 3)) # 3
4 print(my_reduce(lambda x,y: x+y, -1, 4, -2)) # 1
5
6 print(my_reduce(lambda x,y: x*y, -1, 4, -2)) # 8
```

Soluton: implement the reduce function

```
1 def my_reduce(f, *args):
2     if len(args) == 0:
3         return None
4     result = args[0]
5     for i in range(1, len(args)):
6         result = f(result, args[i])
7     return result
8
9 # print(my_reduce()) # TypeError: my_reduce() takes at least 1 argument (0 given)
10 print(my_reduce(lambda x,y: x+y)) # None
11 print(my_reduce(lambda x,y: x+y, 3)) # 3
12 print(my_reduce(lambda x,y: x+y, -1, 4, -2)) # 1
13
14 print(my_reduce(lambda x,y: x*y, -1, 4, -2)) # 8
```

Exercise: sort pairs

Create a function called `sort_pairs`, that would receive a sorting method, e.g. the word ‘keys’ or the word ‘values’ and will receive an arbitrary

number of key-value pairs
and will return a list of tuples.

```
1 sort_pairs( 'keys', foo = 23, bar = 47)
2 [('bar', 47), ('foo', 23)]
3
4 sort_pairs( 'values', foo = 23, bar = 47)
5 [('foo', 23), ('bar', 47)]
```

Solution: sort pairs

```
1 def sort_pairs(how, **kwargs):
2     if how == 'keys':
3         sort_function = lambda s : s[0];
4     elif how == 'values':
5         sort_function = lambda s : s[1];
6     else:
7         raise Exception("Invalid sort function")
8     return sorted(kwargs.items(), key=sort_function)
9
10
11
12 k = sort_pairs( 'keys', foo = 23, bar = 47)
13 print(k)
14 v = sort_pairs( 'values', foo = 23, bar = 47)
15 print(v)
```

Python Packages

Why Create package

As a module gets larger and larger it will be more and more difficult to maintain.

It might be easier if we split it up into multiple files and put those files inside a directory. A ‘package’ is just that. A bunch of Python modules that belong together and are placed in a directory hierarchy. In order to tell Python that you really mean these files to be a package one must add a file called `init.py` in each directory of the project. In the most simple case the file can be empty.

- Code reuse
- Separation of concerns
- Easier distribution

Create package

```
1 mymath/
2     __init__.py
3     calc.py
4     ...
5     internal_use.py
```

```
1 def add(x, y):  
2     return x+y
```

```
1 # empty
```

Internal usage

```
1 import calc  
2 print(calc.add(7, 8))    # 15  
3  
4 from calc import add  
5 print(add(3, 5))        # 8
```

```
1 cd examples/package  
2 python 1/mymath/internal_use.py
```

use module in package - relative path

```
1 import sys  
2 import os  
3  
4 path =  
os.path.join(os.path.dirname(os.path.dirname(os.path.abspath(__file__))), '1')  
5 # print(path)  # /home/gabor/work/slides/python-  
# programming/examples/package/1  
6 sys.path.insert(0, path)  
7  
8 import mymath.calc  
9 print(mymath.calc.add(2, 5))  
10  
11 from mymath.calc import add  
12 print(add(2, 3))
```

```
1 7  
2 5
```

use package (does not work)

```
1 import sys
2 import os
3
4 sys.path.insert(0, os.path.join(
5
6     os.path.dirname(os.path.dirname(os.path.abspath(__file__))),
7     '1' )
8
9 import mymath
9 print(mymath.calc.add(4, 7))
```

```
1 Traceback (most recent call last):
2   File "use_project/proj1_2.py", line 9, in <module>
3     print(mymath.calc.add(4, 7))
4 AttributeError: module 'mymath' has no attribute 'calc'
```

If we import the main package name, it does not have access to the module inside.

package importing (and exporting) module

Put import (and thus re-export) in `init.py`

```
1 def add(x, y):
2     return x+y
```

```
1 import mymath.calc
```

use package (module) with import

Still works...

```
1 import sys
2 import os
```

```
3
4 path = os.path.join(
os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
), '2\
5 ')
6 # print(path)
7 sys.path.insert(0, path)
8
9 import mymath.calc
10 print(mymath.calc.add(2, 5)) # 7
11
12 from mymath.calc import add
13 print(add(2, 3)) # 5
```

use package with import

Now we can import the module from the package and use that.

```
1 import sys
2 import os
3
4 sys.path.insert(0, os.path.join(
5
os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
),
6 '2' )
7
8 import mymath
9 print(mymath.calc.add(4, 7)) # 11
10
11 from mymath import calc
12 print(calc.add(5, 9)) # 14
```

Creating an installable Python package

The directory layout of a package:

```
1 └── mymath
2   ├── calc.py
3   └── __init__.py
4   └── setup.py
```

```
1 from setuptools import setup
2
3
4
5
6
7 setup(name='mymath',
8       version='0.1',
9       description='The best math library',
10      url='http://github.com/szabgab/mymath',
11      author='Foo Bar',
12      author_email='foo@bar.com',
13      license='MIT',
14      packages=['mymath'],
15      zip_safe=False,
16      )
```

Create tar.gz file

```
1 $ python setup.py sdist
```

- mymath.egg-info/
- dist/mymath-0.1.tar.gz

```
1 running sdist
2 running egg_info
3 creating mymath.egg-info
4 writing mymath.egg-info/PKG-INFO
5 writing top-level names to mymath.egg-
info/top_level.txt
6 writing dependency_links to mymath.egg-
info/dependency_links.txt
7 writing manifest file 'mymath.egg-info/SOURCES.txt'
8 reading manifest file 'mymath.egg-info/SOURCES.txt'
9 writing manifest file 'mymath.egg-info/SOURCES.txt'
10 warning: sdist: standard file not found: should have
one of README, README.txt
11
12 creating mymath-0.1
```

```
13 creating mymath-0.1/mymath
14 creating mymath-0.1/mymath.egg-info
15 making hard links in mymath-0.1...
16 hard linking setup.py -> mymath-0.1
17 hard linking mymath/__init__.py -> mymath-0.1/mymath
18 hard linking mymath.egg-info/PKG-INFO -> mymath-
0.1/mymath.egg-info
19 hard linking mymath.egg-info/SOURCES.txt -> mymath-
0.1/mymath.egg-info
20 hard linking mymath.egg-info/dependency_links.txt ->
mymath-0.1/mymath.egg-info
21 hard linking mymath.egg-info/not-zip-safe -> mymath-
0.1/mymath.egg-info
22 hard linking mymath.egg-info/top_level.txt -> mymath-
0.1/mymath.egg-info
23 Writing mymath-0.1/setup.cfg
24 creating dist
25 Creating tar archive
26 removing 'mymath-0.1' (and everything under it)
```

Install Package

```
1 $ pip install dist/mymath-0.1.tar.gz
```

```
1 $ easy_install --prefix ~/python/ dist/mymath-
0.1.tar.gz
```

```
1 $ python setup.py install --prefix ~/python/
```

Upload to [PyPi](#) or distribute to your users.

Dependencies

```
1 requires=[
2     'lawyerup',
3 ],
```

To list them

```
1 $ python setup.py --requires
```

In the setup.py file we only need to change the version number and we can release a new version of the package.

Add README file

```
1 .
2   └── bin
3     ├── runmymath.bat
4     └── runmymath.py
5   └── MANIFEST.in
6   └── mymath
7     └── test
8       ├── __init__.py
9       ├── test_all.py
10      └── test_calc.py
11   └── README.rst
12   └── setup.py
```

```
1 mymath
2 -----
3
4 Super awesome Python module to compute the sum of
numbers.
5
6 To use:
7
8   import mymath
9   mymath.sum(1, 2, 3)
```

```
1 include README.rst
```

Add README file (setup.py)

In the setup.py add the following function:

```
1 def readme():
2     with open('README.rst') as f:
3         return f.read()
```

and in the setup() call include the following parameter:

```
1     long_description=readme(),
```

This will display the README file when called at

```
1 $ python setup.py --long-description
```

Include executables

```
1 root/
2   setup.py
3   README.rst
4   MANIFEST.in
5   bin/
6     runmymath.py
7     runmymath.bat
8   mymath/
9     __init__.py
10    calc.py
```

```
1 import mymath
2
3 def main():
4     print("running")
5
6 main()
```

```
1 echo "hi"
```

`setup.py` will need to get

```
1     scripts=['bin/runmymath.py', 'bin/runmymath.bat'],
```

Add tests

```
1   root/
2       setup.py
3       README.rst
4       MANIFEST.in
5       bin/
6           runmymath.py
7           runmymath.bat
8           mymath/
9               __init__.py
10              calc.py
11              test/
12                  __init__.py
13                  test_all.py
14                  test_calc.py
```

```
1 #empty (needed for unittest discover)
```

```
1 python mymath/test/test_calc.py
2 python mymath/test/test_all.py
```

```
1 python -m unittest discover
```

Add tests calc

```
1 from os.path import dirname, abspath
2 import sys
3
4 sys.path.insert(0,
5     dirname(dirname(abspath(__file__))))
6 from mymath.calc import add
7 import unittest
```

```
8 class AddTest(unittest.TestCase):
9     def test_add(self):
10         self.assertEqual(add(2, 3), 5)
11         self.assertEqual(add(2, -2), 0)
12         #self.assertEqual(add(1, 1), 1)
13
14 if __name__ == '__main__':
15     unittest.main()
```

Add tests all

```
1 from os.path import dirname, abspath
2 import sys
3
4 sys.path.insert(0,
5                 dirname(dirname(abspath(__file__))))
6 from mymath.calc import *
7 import unittest
8
9 class AllTest(unittest.TestCase):
10    def test_sum(self):
11        self.assertEqual(add(2, 3), 5)
12        #self.assertEqual(sum(1, 1), 2)
13        #self.assertEqual(div(6, 2), 3)
14
15 if __name__ == '__main__':
16     unittest.main()
```

setup.py

```
1 from setuptools import setup
2
3 def readme():
4     with open('README.rst') as f:
5         return f.read()
6
7 setup(name='mymath',
8       version='0.2',
9       description='The best math library',
10      url='http://github.com/szabgab/mymath',
11      author='Foo Bar',
```

```
12     author_email='foo@bar.com',
13     license='MIT',
14     packages=['mymath'],
15     zip_safe=False,
16     requires=[
17         'lawyerup',
18     ],
19     long_description=readme(),
20     scripts=['bin/runmymath.py',
21               'bin/runmymath.bat'],
22 )
```

Run tests and create package

```
1 python setup.py test
2 python setup.py sdist
```

Packaging applications (creating executable binaries)

- [py2exe](#) on Windows (discontinued)
- [Freeze](#) on Linux
- [py2app](#) on Mac
- [cx_Freeze](#) cross-platform
- [PyInstaller](#) cross-platform
- [Auto Py To Exe](#)

Using PyInstaller

```
1 print("hello world")
```

```
1 pip install pyinstaller
2 pyinstaller myscript.py
3 pyinstaller --onefile hello_world.py
```

- See the results in dist/

Other PyInstaller examples

Use this to see where does the packaged version of our code look for modules:

```
1 import sys
2
3 print(sys.path)
```

Use this to see how to pass command line parameters to the packaged exe:

```
1 import sys
2
3 print(sys.argv)
```

Other

```
1 pyinstaller --onefile --windowed myscript.py
```

Py2app for Mac

```
1 pip install py2app
2 py2applet examples/basics/hello.py
```

Exercise: package

- Go to [PyPi](#), find some interesting module and install it in a non-standard location (or in a virtualenv)
- Check if it was installed (try to import it in a python script).

- Take one of the previously created modules, and create a package for it.
 - Install this new package in a non-standard location.
 - Check if it works from some other place in your file-system.
-
- Take the mymath package, add another method, add tests and create the distubtable zip file.

Exercise: create executable

- Go over some of the examples in the course and package that.
- Package a script using some of your favorite modules.

Ctypes

ctypes - hello

```
1 #include <stdio.h>
2
3 char * echo(char * what)
4 {
5     return what;
6 }
7
8 int add_int(int a, int b)
9 {
10     int sum = a+b;
11     return sum;
12 }
13
14 int add_int(int a, int b)
15 {
16     int sum = a+b;
17     return sum;
18 }
19
20
21 int main(void)
22 {
23     printf("hello\n");
24     printf("%d\n", add_int(2, 3));
25     printf("%s\n", echo("Foo"));
26     return 0;
27 }
```

```
1 gcc -o hello hello.c
2 gcc -o hello.so -shared -fPIC hello.c
```

```
1 from ctypes import cdll
2 from ctypes import c_char_p
```

```
3
4 hello_lib = cdll.LoadLibrary("hello.so")
5
6 print(hello_lib.add_int(4, 5))           # 9
7
8 print(hello_lib.echo('Hello World'))    # 153977204
9
10
11 hello_lib.echo.restype = c_char_p
12 print(hello_lib.echo('Hello World'))    # Hello World
```

concat

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4
5 int len(char * s)
6 {
7     return strlen(s);
8 }
9
10 char * concat(char * a, char * b)
11 {
12     char * res;
13     int leng = strlen(a) + strlen(b);
14     res = (char *)malloc(leng);
15     strcpy(res, a);
16     strcat(res, b);
17     return res;
18 }
19
20
21 int main(void)
22 {
23     printf("concat\n");
24     printf("%d\n", len("abc"));
25     printf("%d\n", len(""));
26     printf("%d\n", len("xxxxxxxxxx"));
27     printf("%s\n", concat("Fool", "Bar"));
28     return 0;
29 }
```

```
1 from ctypes import cdll
2 from ctypes import c_char_p
3
4 more_lib = cdll.LoadLibrary("more.so")
5
6 print(more_lib.len("abcd"))          # 4
7 print(more_lib.len(""))             # 0
8 print(more_lib.len("x" * 123))     # 123
9
10
11 more_lib.concat.restype = c_char_p
12 print(more_lib.concat("abc", "def"))
```

links

- [ctypes](#)
- [Python Ctypes Tutorial](#)

Advanced OOP

Class count instances

```
1 class Person:  
2     count = 0  
3     def __init__(self, name):  
4         self.name = name  
5         #Person.count += 1  
6         #self.count += 1  
7         self.count = self.count + 1  
8  
9  
10    print(Person.count)  
11    joe = Person("Joe")  
12    print(Person.count)  
13    print(joe.count)  
14  
15    jane = Person("Jane")  
16    print(Person.count)  
17    print(jane.count)
```

```
1 0  
2 0  
3 1  
4 0  
5 1
```

Class Attributes

- Class attributes can be created inside a class.
- Assign to class attribute and fetch from it
- Class attributes can be also created from the outside.

```
1 class Person:
2     name = 'Joseph'
3
4 print(Person.name)      # Joseph
5
6 Person.name = 'Joe'
7 print(Person.name)      # Joe
8
9 Person.email = 'joe@foobar.com'
10 print(Person.email)     # joe@foobar.com
```

Class Attributes in Instances

```
1 class Person:
2     name = 'Joe'
3
4 # Class Attributes are inherited by object instances
when accessing them.
5 x = Person()
6 print(x.name)          # Joe
7 y = Person()
8 print(y.name)          # Joe
9
10 # Changes to class attribute are reflected in existing
instances as well
11 Person.name = 'Bar'
12 print(Person.name)    # Bar
13 print(x.name)         # Bar
14
15 # Setting the attribute via the instance will create
an instance attribute that
16 # shadows the class attribute
17 x.name = 'Joseph'
18 print(x.name)          # Joseph
19 print(Person.name)     # Bar
20 # Nor does it impact the instance attribute of other
instances:
21 print(y.name)          # Bar
22
23 # Both instance and class have a dictionary containing
its members:
24 print(x.__dict__)      # {'name': 'Joseph'}
```

```
25 print(y.__dict__)          # {}
26 print(Person.__dict__)    # {..., 'name': 'Bar'}
```

Attributes with method access

- Use a method (show) to access it.

```
1 class Person():
2     name = 'Joe'
3     print(f'Hello {name}')
4
5     def show(self):
6         print(Person.name)
7
8 x = Person()           # Hello Joe
9 x.show()                # Joe
10 print(x.name)          # Joe
11 print(Person.name)      # Joe
12
13 Person.name = 'Jane'
14 print(x.name)          # Jane
15 print(Person.name)      # Jane
16 x.show()                # Jane
17
18 x.name = 'Hilda'       # creating and setting the
19 instance attribute
20 print(x.name)          # Hilda
21 print(Person.name)      # Jane
22 x.show()                # Jane
```

Instance Attribute

The attributes of the instance object can be set via ‘self’ from within the class.

```
1 class Person():
2     name = 'Joseph'
3
4     def __init__(self, given_name):
```

```

5         self.name = given_name
6
7     def show_class(self):
8         return Person.name
9
10    def show_instance(self):
11        return self.name
12
13 print(Person.name)           # Joseph
14
15 Person.name = 'Classy'
16 print(Person.name)          # Classy
17 # print(Person.show_class()) # TypeError: show_class()
missing 1 required positional\
18 argument: 'self'
19
20 x = Person('Joe')
21 print(x.name)               # Joe
22 print(Person.name)          # Classy
23 print(x.show_class())       # Classy
24 print(x.show_instance())    # Joe
25
26 Person.name = 'General'
27 print(x.name)               # Joe
28 print(Person.name)          # General
29 print(x.show_class())       # General
30 print(x.show_instance())    # Joe
31
32 x.name = 'Zorg'            # changing the instance
attribute
33 print(x.name)               # Zorg
34 print(Person.name)          # General
35 print(x.show_class())       # General
36 print(x.show_instance())    # Zorg

```

Methods are class attributes

In this example we are going to replace the method in the class by a newly created function.
(monkey patching)

```
1 class Person():
2     def __init__(self, name):
3         self.name = name
4
5     def show(self):
6         return self.name
7
8 y = Person('Jane')
9 print(y.show())          # Jane
10
11 def new_show(some_instance):
12     print("Hello " + some_instance.name)
13     return some_instance
14
15 Person.show = new_show
16 y.show()                 # Hello Jane
```

Monkey patching

```
1 class Person():
2     def __init__(self, name):
3         self.name = name
4
5     def show(self):
6         return self.name
7
8 x = Person('Joe')
9 print(x.show())          # Joe
10
11 def patch(class_name):
12     temp = class_name.show
13     def debug(*args, **kwargs):
14         print("in debug")
15         return temp(*args, **kwargs)
16     class_name.show = debug
17
18 patch(Person)
19
20 print(x.show())
21     # in debug
22     # Joe
```

Classes: instance method

Regular functions (methods) defined in a class are “instance methods”. They can only be called on “instance objects” and not on the “class object” as seen in the 3rd example.

The attributes created with “self.something = value” belong to the individual instance object.

```
1 class Date:  
2     def __init__(self, Year, Month, Day):  
3         self.year = Year  
4         self.month = Month  
5         self.day = Day  
6  
7     def __str__(self):  
8         return 'Date({}, {}, {})'.format(self.year,  
self.month, self.day)  
9  
10    def set_date(self, y, m, d):  
11        self.year = y  
12        self.month = m  
13        self.day = d
```

```
1 from mydate import Date  
2  
3 d = Date(2013, 11, 22)  
4 print(d)  
5  
6 # We can call it on the instance  
7 d.set_date(2014, 1, 27)  
8 print(d)  
9
```

```
10 # If we call it on the class, we need to pass an
11 # instance.
12 Date.set_date(d, 2000, 2, 1)
13 print(d)
14
15
16 # If we call it on the class, we get an error
17 Date.set_date(1999, 2, 1)
```

`set_date` is an instance method. We cannot properly call it on a class.

```
1 Date(2013, 11, 22)
2 Date(2014, 1, 27)
3 Date(2000, 2, 1)
4 Traceback (most recent call last):
5   File "run.py", line 17, in <module>
6     Date.set_date(1999, 2, 1)
7 TypeError: set_date() missing 1 required positional
argument: 'd'
```

Class methods and class attributes

“total” is an attribute that belongs to the class. We can access it using `Date.total`. We can create a `@classmethod` to access it, but actually we can access it from the outside even without the class method, just using the “class object”

```
1 class Date:
2     total = 0
3
4     def __init__(self, Year, Month, Day):
5         self.year = Year
6         self.month = Month
7         self.day = Day
8         Date.total += 1
9
10    def __str__(self):
11        return 'Date({}, {}, {})'.format(self.year,
```

```
self.month, self.day)
12
13     def set_date(self, y, m, d):
14         self.year = y
15         self.month = m
16         self.day = d
17
18     @classmethod
19     def get_total(class_object):
20         print(class_object)
21         return class_object.total
```

```
1 from mydate import Date
2
3 d1 = Date(2013, 11, 22)
4 print(d1)
5 print(Date.get_total())
6 print(Date.total)
7 print('')
8
9 d2 = Date(2014, 11, 22)
10 print(d2)
11 print(Date.get_total())
12 print(Date.total)
13 print('')
14
15 d1.total = 42
16 print(d1.total)
17 print(d2.total)
18 print(Date.get_total())
19 print(Date.total)
```

```
1 Date(2013, 11, 22)
2 <class 'mydate.Date'>
3 1
4 1
5
6 Date(2014, 11, 22)
7 <class 'mydate.Date'>
8 2
9 2
10
```

```
11 42
12 2
13 <class 'mydate.Date'>
14 2
15 2
```

Classes: constructor

- The “class” keyword creates a “class object”. The default constructor of these classes are their own names.
- The actual code is implemented in the `__new__` method of the object.
- Calling the constructor will create an “instance object”.

Class methods - alternative constructor

Class methods are used as Factory methods, they are usually good for alternative constructors. In order to be able to use a method as a class-method

(Calling `Date.method(...)` one needs to mark the method with the `@classmethod` decorator)

```
1 class Date:
2     def __init__(self, Year, Month, Day):
3         self.year = Year
4         self.month = Month
5         self.day = Day
6
7     def __str__(self):
8         return 'Date({}, {}, {})'.format(self.year,
self.month, self.day)
9
10    def set_date(self, y, m, d):
11        self.year = y
12        self.month = m
13        self.day = d
14
15    @classmethod
```

```
16     def from_str(class_object, date_str):
17         '''Call as
18             d = Date.from_str('2013-12-30')
19         '''
20         print(class_object)
21         year, month, day = map(int, date_str.split('-'
22 ))  
22         return class_object(year, month, day)
```

```
1 from mydate import Date
2
3 d = Date(2013, 11, 22)
4 print(d)
5
6 d.set_date(2014, 1, 27)
7 print(d)
8
9 print('')
10 dd = Date.from_str('2013-10-20')
11 print(dd)
12
13 print('')
14 z = d.from_str('2012-10-20')
15 print(d)
16 print(z)
```

```
1 Date(2013, 11, 22)
2 Date(2014, 1, 27)
3
4 <class 'mydate.Date'>
5 Date(2013, 10, 20)
6
7 <class 'mydate.Date'>
8 Date(2014, 1, 27)
9 Date(2012, 10, 20)
```

Abstract Base Class

- Create a class object that cannot be used to create an instance object. (It must be subclassed)

- The subclass must implement certain methods required by the base-class.

```
1 class NotImplementedException(Exception):
2     pass
3
4 class Base():
5     def foo(self):
6         raise NotImplemented()
7
8     def bar(self):
9         raise NotImplemented()
10
11 class Real(Base):
12     def foo(self):
13         print('foo in Real')
14     def bar(self):
15         print('bar in Real')
16     def other(self):
17         pass
18
19 class Fake(Base):
20     def foo(self):
21         print('foo in Fake')
22
23 r = Real()
24 r.foo()
25 r.bar()
26 f = Fake()
27 f.foo()
28 f.bar()
```

```
1 foo in Real
2 bar in Real
3 foo in Fake
4 Traceback (most recent call last):
5   File "no_abc.py", line 28, in <module>
6     f.bar()      # NotImplemented()
7   File "no_abc.py", line 9, in bar
8     raise NotImplemented()
9 __main__.NotImplemented()
```

Abstract Base Class with abc

- [abc](#)

```
1 from abc import ABC, abstractmethod
2
3 class Base(ABC):
4     def __init__(self, name):
5         self.name = name
6
7     @abstractmethod
8     def foo(self):
9         pass
10
11    @abstractmethod
12    def bar(self):
13        pass
```

ABC working example

```
1 from with_abc3 import Base
2
3 class Real(Base):
4     def foo(self):
5         print('foo in Real')
6
7     def bar(self):
8         print('bar in Real')
9
10    def other(self):
11        pass
12
13 r = Real('Jane')
14 print(r.name)      # Jane
```

1 Jane

ABC - cannot instantiate the base-class

```
1 from with_abc3 import Base  
2  
3 b = Base('Boss')
```

```
1 Traceback (most recent call last):  
2   File "with_abc3_base.py", line 3, in <module>  
3     b = Base('Boss')  
4 TypeError: Can't instantiate abstract class Base with  
abstract methods bar, foo
```

ABC - must implement methods

```
1 from with_abc3 import Base  
2  
3 class Fake(Base):  
4     def foo(self):  
5         print('foo in Fake')  
6  
7 f = Fake('Joe')
```

```
1 Traceback (most recent call last):  
2   File "with_abc3_fake.py", line 7, in <module>  
3     f = Fake('Joe')  
4 TypeError: Can't instantiate abstract class Fake with  
abstract methods bar
```

Use Python @property to fix bad interface (the bad interface)

When we created the class the first time we wanted to have a field representing the age of a person. (For simplicity of the example we only store the years.)

```
1 class Person():  
2     def __init__(self, age):  
3         self.age = age
```

```
4
5 p = Person(19)
6 print(p.age)          # 19
7
8 p.age = p.age + 1
9 print(p.age)          # 20
```

Only after releasing it to the public have we noticed the problem.
Age changes.

We would have been better off storing birthdate and if necessary calculating the age.

How can we fix this?

Use Python `@property` to fix bad interface (first attempt)

This might have been a good solution, but now we cannot use this as a “fix” as this would change the public interface from `p.age` to `p.age()`

```
1 from datetime import datetime
2 class Person():
3     def __init__(self, years):
4         self.set_birthyear(years)
5
6     def get_birthyear(self):
7         return datetime.now().year - self._birthyear
8
9     def set_birthyear(self, years):
10        self._birthyear = datetime.now().year - years
11
12    def age(self, years=None):
13        if (years):
14            self.set_birthyear(years)
15        else:
16            return self.get_birthyear()
```

```
17
18
19
20 p = Person(19)
21 print(p.age())           # 19
22
23 p.age(p.age() + 1)
24 print(p.age())           # 20
```

Use Python @property to fix bad API

```
1 property(fget=None, fset=None, fdel=None, doc=None)

1 from datetime import datetime
2 class Person():
3     def __init__(self, years):
4         self.age = years
5
6     def get_birthyear(self):
7         return datetime.now().year - self.birthyear
8
9     def set_birthyear(self, years):
10        self.birthyear = datetime.now().year - years
11
12    age = property(get_birthyear, set_birthyear)
13
14 p = Person(19)
15 print(p.age)           # 19
16
17 p.age = p.age + 1
18 print(p.age)           # 20
19
20 p.birthyear = 1992
21 print(p.age)           # 28
22     # warning: this will be different if you run the
23     # example in a year different from \
24     # 2020 :)
```

Use Python @property decorator to fix bad API

```
1 from datetime import datetime
2 class Person():
3     def __init__(self, years):
4         self.age = years
5
6     # creates "getter"
7     @property
8     def age(self):
9         return datetime.now().year - self.birthyear
10
11    # creates "setter"
12    @age.setter
13    def age(self, years):
14        self.birthyear = datetime.now().year - years
15
16 p = Person(19)
17 print(p.age)      # 19
18
19 p.age = p.age + 1
20 print(p.age)      # 20
21
22
23 p.birthyear = 1992
24 print(p.age)      # 28
25    # warning: this will be different if you run the
26    # example in a year different from \
27    # 2020 :)
```

- [property article](#)
- [property docs](#)

Use Python @property for value validation

```
1 from datetime import datetime
2 class Person():
3     def __init__(self, years):
4         self.age = years
```

```
5
6     @property
7     def age(self):
8         return datetime.now().year - self.birthyear
9
10    @age.setter
11    def age(self, years):
12        if years < 0:
13            raise ValueError("Age cannot be negative")
14        self.birthyear = datetime.now().year - years
```

```
1 from person5 import Person
2
3 p = Person(19)
4 print(p.age)          # 19
5
6 p.age = p.age + 1
7 print(p.age)          # 20
8
9 p.birthyear = 1992
10 print(p.age)          # 28
11     # warning: this will be different if you run the
example in a year different from\
12 2020 :)
```

```
1 from person5 import Person
2
3 print("Hello")
4
5 p = Person(-1)
```

```
1 Hello
2 Traceback (most recent call last):
3   File "person5_bad_init.py", line 5, in <module>
4     p = Person(-1)
5   File "/home/gabor/work/slides/python-
programming/examples/classes/person/person5.p\
6 y", line 4, in __init__
7     self.age = years
8   File "/home/gabor/work/slides/python-
programming/examples/classes/person/person5.p\
```

```
9 y", line 13, in age
10      raise ValueError("Age cannot be negative")
11 ValueError: Age cannot be negative
```

```
1 Hello
2 10
3 Traceback (most recent call last):
4   File "person5_bad_setter.py", line 7, in <module>
5     p.age = -1
6   File "/home/gabor/work/slides/python-
programming/examples/classes/person/person5.p\
7 y", line 13, in age
8      raise ValueError("Age cannot be negative")
9 ValueError: Age cannot be negative
```

class and static methods

Static methods are used when no “class-object” and no “instance-object” is required.

They are called on the class-object, but they don’t receive it as a parameter.

They might be better off placed in a module, like the other_method.

```
1 def other_method(val):
2     print(f"other_method: {val}")
3
4 class Date(object):
5     def __init__(self, Year, Month, Day):
6         self.year = Year
7         self.month = Month
8         self.day = Day
9
10    def __str__(self):
11        return 'Date({}, {}, {})'.format(self.year,
12                                         self.month, self.day)
13
14    @classmethod
15    def from_str(class_object, date_str):
```

```
15         '''Call as
16             d = Date.from_str('2013-12-30')
17         '''
18     print(f"from_str: {class_object}")
19     year, month, day = map(int, date_str.split('-'
20 ))
21
22     other_method(43)
23
24     if class_object.is_valid_date(year, month,
25     day):
26         return class_object(year, month, day)
27     else:
28         raise Exception("Invalid date")
29
30     @staticmethod
31     def is_valid_date(year, month, day):
32         if 0 <= year <= 3000 and 1 <= month <= 12 and
33         1 <= day <= 31:
34             return True
35         else:
36             return False
```

```
1 import mydate
2
3 dd = mydate.Date.from_str('2013-10-20')
4 print(dd)
5
6 print('')
7 print(mydate.Date.is_valid_date(2013, 10, 20))
8 print(mydate.Date.is_valid_date(2013, 10, 32))
9 print('')
10
11 x = mydate.Date.from_str('2013-10-32')
```

```
1 from_str: <class 'mydate.Date'>
2 other_method: 43
3 Date(2013, 10, 20)
4
5 True
6 False
7
```

```
8 from _str: <class 'mydate.Date'>
9 other_method: 43
10 Traceback (most recent call last):
11   File "run.py", line 11, in <module>
12     x = mydate.Date.from_str('2013-10-32')
13   File "/home/gabor/work/slides/python-
programming/examples/classes/mydate4/mydate.p\
14 y", line 26, in from_str
15     raise Exception("Invalid date")
16 Exception: Invalid date
```

Destructor: del

```
1 class Person:
2     def __init__(self):
3         print('__init__')
4     def __del__(self):
5         print('__del__')
6
7 def main():
8     a = Person()
9     print('in main - after')
10
11 main()
12 print('after main')
```

```
1 __init__
2 in main - after
3 __del__
4 after main
```

Destructor delayed

Becasue the object has a reference to itself. (Python uses both reference count and garbage collection.)

```
1 class Person:
2     def __init__(self, name):
3         self.name = name
4         print(f'__init__ {name}')
```

```
5
6     def __del__(self):
7         print(f'__del__ {self.name}')
8
9 def main():
10    a = Person('A')
11    b = Person('B')
12    a.partner = b
13    print('in main - after')
14
15 main()
16 print('after main')
```

```
1 __init__ A
2 __init__ B
3 in main - after
4 __del__ B
5 after main
6 __del__ A
```

Destructor delayed for both

Because the instances reference each other

```
1 class Person:
2     def __init__(self, name):
3         self.name = name
4         print(f'__init__ for {self.name}')
5     def __del__(self):
6         print(f'__del__ for {self.name}')
7
8 def main():
9     a = Person('Joe')
10    b = Person('Jane')
11    a.partner = b
12    b.partner = a
13    print('in main - after')
14
15 main()
16 print('after main')
```

```
1 __init__ for Joe
2 __init__ for Jane
3 in main - after
4 after main
5 __del__ for Joe
6 __del__ for Jane
```

Opearator overloading

```
1 import copy
2
3 class Rect:
4     def __init__(self, w, h):
5         self.width = w
6         self.height = h
7
8     def __str__(self):
9         return 'Rect[{}, {}]'.format(self.width,
self.height)
10
11    def __mul__(self, other):
12        o = int(other)
13        new = copy.deepcopy(self)
14        new.height *= o
15        return new
```

```
1 import shapes
2
3 r = shapes.Rect(10, 20)
4 print(r)
5 print(r * 3)
6 print(r)
7
8 print(4 * r)
```

```
1 Rect[10, 20]
2 Rect[10, 60]
3 Rect[10, 20]
4 Traceback (most recent call last):
5   File "rect.py", line 8, in <module>
```

```
6     print(4 * r)
7 TypeError: unsupported operand type(s) for *: 'int' and
'Rect'
```

In order to make the multiplication work in the other direction, one needs to implement the **rmul** method.

Operator overloading methods

```
1 *      __mul__,    __rmul__
2 +      __add__,   __radd__
3 +=     __iadd__
4 <      __lt__
5 <=    __le__
6 ...
```

- [see all of them in datamodel](#)

Exercise: rectangular

Take the Rect class in the shapes module. Implement **rmul**, but in that case multiply the width of the rectangular.

Implement the addition of two rectangulairs. I think this should be defined only if one of the sides is the same, but if you have an idea how to add two rectangualars of different sides, then go ahead, implement that.

Also implement all the comparision operators when comparing two rectangulairs, compare the area of the two. (like less-than)
Do you need to implement all of them?

Exercise: SNMP numbers

- SNMP numbers are strings consisting a series of integers separated by dots: 1.5.2, 3.7.11.2
- Create a class that can hold such an snmp number. Make sure we can compare them with less-than (the comparision is pair-wise for each number until we find two numbers that are different. If one SNMP number is the prefix is the other then the shorter is “smaller”).
- Add a class-method, that can tell us how many SNMP numbers have been created.
- Write a separate file to add unit-tests

Exercise: Implement a Gene inheritance model combining DNA

- A class representing a person. It has an attribute called “genes” which is string of letters. Each character is a gene.
- Implement the + operator on genes that will create a new “Person” and for the gene will select one randomly from each parent.

```

1 a = Person('ABC')
2 b = Person('DEF')
3
4 c = a + b
5 print(c.gene) # ABF

```

Exercise: imaginary numbers - complex numbers

Create a class that will represent imaginary numbers ($x, y*i$) and has methods to add and multiply two imaginary numbers.

```
1 The math:  
2  
3 z1 = (x1 + y1*i)  
4 z2 = (x2 + y2*i)  
5 z1+z2 = (x1 + x2 + (y1 + y2)*i)  
6  
7 z1*z2 = x1*y1 + x2*y2*i*i + x1*y2*i + x2*y1*i
```

Add operator overloading so we can really write code like:

```
1 z1 = Z(2, 3)  
2 z2 = Z(4, 7)  
3  
4 zz = z1*z2
```

- See [cmath](#)

```
1 z = complex(2, 3)  
2 print(z)  
3 print(z.real)  
4 print(z.imag)  
5  
6 imag = (-1) ** 0.5  
7 print(imag)  
8  
9 i = complex(0, 1)  
10 print(i)  
11 print(i ** 2)
```

```
1 (2+3j)  
2 2.0  
3 3.0  
4 (6.123233995736766e-17+1j)  
5 1j  
6 (-1+0j)
```

Solution: Rectangular

```
1 import copy
2 import shapes
3
4 class Rectangular(shapes.Rect):
5
6     def __rmul__(self, other):
7         o = int(other)
8         new = copy.deepcopy(self)
9         new.width *= o
10        return new
11
12    def area(self):
13        return self.width * self.height
14
15    def __eq__(self, other):
16        return self.area() == other.area()
17
18    def __add__(self, other):
19        new = copy.deepcopy(self)
20        if self.width == other.width:
21            new.height += other.height
22        elif self.height == other.height:
23            new.width += other.width
24        else:
25            raise Exception('None of the sides are
equal')
26        return new
```

```
1 import shape2
2 import unittest
3
4 class TestRect(unittest.TestCase):
5
6     def assertEqualsSides(self, left, right):
7         if isinstance(right, tuple):
8             right = shape2.Rectangular(*right)
9
10        if left.width != right.width:
11            raise AssertionError('widths are
different')
12        if left.height != right.height:
13            raise AssertionError('heights are
```

```

different')

14
15    def setUp(self):
16        self.a = shape2.Rectangular(4, 10)
17        self.b = shape2.Rectangular(2, 20)
18        self.c = shape2.Rectangular(1, 30)
19        self.d = shape2.Rectangular(4, 10)
20
21    def test_sanity(self):
22        self.assertEqualSides(self.a, self.a)
23        self.assertEqualSides(self.a, self.d)
24        try:
25            self.assertEqualSides(self.a, self.b)
26        except AssertionError as e:
27            self.assertEqual(e.args[0], 'widths are
different')
28
29        try:
30            self.assertEqualSides(self.a,
shape2.Rectangular(4, 20))
31        except AssertionError as e:
32            self.assertEqual(e.args[0], 'heights are
different')
33
34        self.assertEqualSides(self.a, (4, 10))
35
36    def test_str(self):
37        self.assertEqual(str(self.a), 'Rect[4, 10]')
38        self.assertEqual(str(self.b), 'Rect[2, 20]')
39        self.assertEqual(str(self.c), 'Rect[1, 30]')
40
41    def test_mul(self):
42        self.assertEqual(str(self.a * 3), 'Rect[4,
30]')
43        self.assertEqual(str(self.b * 7), 'Rect[2,
140]')
44
45    def test_rmul(self):
46        self.assertEqual(str(3 * self.a), 'Rect[12,
10]')
47        self.assertEqualSides(3 * self.a, (12, 10))
48
49    def test_area(self):
50        self.assertEqual(self.a.area(), 40)

```

```

51         self.assertEqual(self.b.area(), 40)
52         self.assertEqual(self.c.area(), 30)
53
54     def test_equal(self):
55         self.assertEqual(self.a, self.d)
56         self.assertEqual(self.a, self.b)
57
58     def test_add(self):
59         self.assertEqualSides(self.a +
shape2.Rectangular(4, 20), (4, 30))
60
61
62
63
64 if __name__ == '__main__':
65     unittest.main()

```

Solution: Implement a Gene inheritance model combining DNA

```

1 import random
2
3 class Person(object):
4     def __init__(self, DNA):
5         self.DNA = DNA
6
7     def gene(self):
8         return list(self.DNA)
9
10    def print_genes(self):
11        print(list(self.DNA))
12
13    def __add__(self, other):
14        DNA_father = self.gene()
15        DNA_mother = other.gene()
16        if len(DNA_father) != len(DNA_mother):
17            raise Exception("Incompatible couple")
18
19        DNA_childPossible_sequence = DNA_father +
DNA_mother
20        DNA_child = ""
21        for i in range(len(self.gene())):

```

```
22             DNA_child += random.choice([DNA_father[i],
23 DNA_mother[i]])
24
25
26
27 a = Person("ABCD")
28 b = Person("1234")
29 c = a + b
30 print(c.DNA)
```

Instance counter

```
1 class Bike:
2     count = 0
3     def __init__(self):
4         Bike.count += 1
5
6     def __del__(self):
7         Bike.count -= 1
8
9     def bike_trip():
10        print(Bike.count)      # 0
11        a = Bike()
12        print(Bike.count)      # 1
13        b = Bike()
14        print(Bike.count)      # 2
15        c = Bike()
16        print(Bike.count)      # 3
17        b = None
18        print(Bike.count)      # 2
19
20
21 bike_trip()
22 print(Bike.count)      # 0
```

2to3

Converting from Python 2 to Python 3

from **future** import ...

division

```
1 print 3/2    # 1


---


1 from __future__ import division
2
3 print 3/2    # 1.5
```

print in Python 2

```
1 fname = 'Foo'
2 lname = 'Bar'
3 print("Name: %s %s" % (fname, lname))
4 print("Name: {} {}".format(fname, lname))
5 print(fname, lname)
6 print fname, lname
```

```
1 Name: Foo Bar
2 Name: Foo Bar
3 ('FOO', 'Bar')
4 Foo Bar
```

print in Python 3

print now requires print()

```
1 from __future__ import print_function
2
3 fname = 'Foo'
4 lname = 'Bar'
5 print("Name: %s %s" % (fname, lname))
6 print("Name: {} {}".format(fname, lname))
7 print(fname, lname)
```

```
1 Name: Foo Bar
2 Name: Foo Bar
3 Foo Bar
```

input and raw_input

`raw_input()` was renamed to `input()`

In Python 2 `raw_input()` returned the raw string. `input()`, on the other hand ran `eval(raw_input())` which meant it tried to execute the input string as a piece of Python code. This was dangerous and was not really used.

In Python 3 `raw_input()` is gone. `input()` behaves as the old `raw_input()` returning the raw string. If you would like to get the old, and dangerous, behavior of `input()` you can call `eval(input())`.

Code that works on both 2 and 3

```
1 import platform
2
3 def my_input(text):
4     if platform.python_version_tuple()[0] == 3:
5         return input(text)
6     else:
7         return raw_input(text)
```

Compare different types

```
1 x = 3
2 y = '3'
3
4                                     # Python 2      Python 3
5 print( x > y )      # False      TypeError:
unorderable types: int() > str()
6 print( x < y )      # True       TypeError:
unorderable types: int() < str()
7 print( x == y )     # False      False
```

Octal numbers

Octal numbers in 2.x was 011 in 3.x is: 0o11

2to3 Resources

- [python3porting book](#)
- [wiki](#)
- [Dive into Python 3](#)
- [The future module](#)
- [The third-party future module](#)
- [The six module](#)
- [docs of 2to3](#)

Design Patterns

What are Design Patterns?

Not all the Design Patterns discussed for Java or C++ are interesting, relevant or even needed in Python.

Design Patterns are formal descriptions of how people do things, and not how you should do things.

The formal description makes it easy to talk about them.

Some of the DPs exists to overcome problems in that specific language.

Other DPs are more general, solving classes of problem that are generic.

Don't replace built-in objects

```
1 import sys
2
3 print = 'hello'
4 sys.stdout.write(print)
5 sys.stdout.write('\n')
```

```
1 pip install flake8-builtins
2 flake8 --ignore=    replace_print.py
3
4 replace_print.py:3:1: A001 "print" is a python builtin
and is being shadowed, consider renaming the variable
```

Facade - simple interface to complex system

Facade, a structural design pattern. - Provide a simple interface (maybe a single class with few methods) to some complex system behind it.

This gives flexibility for the implementation of the complex system while users gain simplicity in using it in certain subsets of operations.

```
1 os.path.basename, os.path.dirname are faced for
os.path.split + indexing in the list
2 os.path.basename = os.path.split() [-1]
3 os.path.split = split with os.sep
4 os.path.join(names) = os.sep.join(names)
5 os.path.isdir(path) = stat.S_ISDIR(os.stat(path))
```

- [](<http://docs.python.org/library/os.path.html>)
- [](<http://docs.python.org/library/os.html>)
- [](<http://docs.python.org/library/stat.html>)

Monkey Patching

```
1 import real_class
2 class faker(object): pass
3 fake = faker
4 real_class.time = fake
5 fake.sleep =
6 fake.time =
```

- handy in emergencies
- easily abused for NON-emergencies - gives dynamic languages a bad name

- subtle hidden “communication” via secret obscure pathways (explicit is better)
-

```
1 class Monkey:  
2  
3     def __init__(self, count):  
4         self.bananas = count  
5  
6     def is_hungry(self):  
7         hungry = True  
8         if hungry:  
9             self.eat()  
10  
11    def eat(self):  
12        self.bananas -= 1  
13  
14  
15 m = Monkey(10)  
16 print(m.bananas)      # 10  
17 print(m.is_hungry())  # None  
18 print(m.bananas)      # 9  
19  
20 Monkey.eat = lambda self: True  
21  
22 om = Monkey(10)  
23 print(om.bananas)      # 10  
24 print(om.is_hungry())  # None  
25 print(om.bananas)      # 10
```

Creation DPs “Just One”

we want just one instance to exist

- Singleton - subclassing can never be really smooth
- Use a module instead of a class (no inheritance, no special methods)
- make just one instance (self discipline, no enforcement), need to decide to “when” (in which part of the code) to make it

- monostate (borg)

Singleton

```
1  class Singleton(object):
2      def __new__(cls, *a, **kw):
3          if not hasattr(cls, '_inst'):
4              cls._inst = super(Singleton,
5                  cls).__new__(*a, **kw)
6          return cls._inst
```

the problem

```
1  class Foo(Singleton): pass
2  class Bar(Foo): pass
3  f = Foo()
4  b = Bar()
5  # what class is b now? is that a Bar or a Foo
6  instance?
```

Monostate (Borg)

- Monostate Pattern

```
1 class Monostate(object):
2     _shared_state = {}
3     def __new__(cls, *a, **kw):
4         obj = super(Monostate, cls).__new__(*a, **kw)
5         obj.__dict__ = _shared_state
6         return obj
7
8 class Foo(Monostate): pass
9 class Bar(Foo): pass
10 f = Foo()
11 b = Bar()
```

Better than singleton, data overriding to the rescue:
But what if two calls to the constructor provide different initial
data?

Dispatch table

```
1 calls = []
2 calls.append( lambda x: x+1 )
3 calls.append( lambda x: x*2 )
4
5 others = [
6     lambda x: x-1,
7     lambda x: 0
8 ]
9
10 def do_something( call_list ):
11     for c in call_list:
12         print(c(3))
13
14
15 do_something( calls )
16 do_something( others )
```

Parallel

Types of Problems

- CPU intensive application - use more of the cores to reduce the wallclock time.
- IO intensive applications - don't waste the CPU and wallclock time while waiting for the IO process.
- Interactive applications - make sure they are responsive during long operations.

Types of solutions

- Number of processes (forking on Unix or spawning)
- Number of threads (Single threaded vs Multi-threaded)
- Asynchronous, non-blocking or synchronous vs blocking (aka “normal”) Cooperative Multitasking

How many parallels to use?

* First of all, I call them “parallels” as this applies to forks, threads, spawns, and even to async code.

- Overhead of creating new parallel.
- Overhead of communication (sending job input to parallel, receiving results).
- Total number of items to process.
- Time it takes to process an item.

- Distribution of processing times. (e.g. one long and many short jobs.)
- Number of cores (CPUs).

Dividing jobs

- N items to process
- K in parallel
- Divide the items in K groups of size $\text{int}(N/K)$ and $\text{int}(N/K)+1$.
- Create K parallels with one item each. When it is done, give it another item.
- Create K parallels with one item each. When done let it stop and create a new parallel.

Performance Monitoring

- Linux, OSX: htop
- Windows: Performance Monitor

Threads

Python Threading docs

- [threading](#)
- [Real Python](#)
- [Wikibooks](#)

Threaded counters

```
1 import threading
2 import sys
3
4 class ThreadedCount(threading.Thread):
5     def run(self):
6         for cnt in range(6):
7             print(f'{cnt}')
{threading.current_thread().name})
8         return
9
10 a = ThreadedCount()
11 b = ThreadedCount()
12 c = ThreadedCount()
13
14 a.start()
15 b.start()
16 c.start()
17 print('main - Running {}'
threads'.format(threading.active_count()))
18
19 a.join()
20 b.join()
21 c.join()
22 print("main - thread is done")
```

```
1 0 Thread-1
2 1 Thread-1
3 0 Thread-2
4 2 Thread-1
5 1 Thread-2
6 0 Thread-3
7 3 Thread-1
8 2 Thread-2
9 main - Running 4 threads
10 3 Thread-2
11 1 Thread-3
12 4 Thread-2
13 2 Thread-3
14 5 Thread-2
15 3 Thread-3
16 4 Thread-1
17 4 Thread-3
18 5 Thread-1
19 5 Thread-3
20 main - thread is done
```

Simple threaded counters

```
1 import threading
2 import sys
3
4 class ThreadedCount(threading.Thread):
5     def run(self):
6         thread = threading.current_thread()
7         print('{} - start'.format(thread.name))
8         for c in range(10):
9             print('{} - count {}'.format(thread.name,
c))
10        print('{} - end'.format(thread.name))
11        return
12
13 a = ThreadedCount()
14 b = ThreadedCount()
15 c = ThreadedCount()
16 a.start()
17 b.start()
18 c.start()
```

```
19
20 print('main - running {}'
21     threads'.format(threading.active_count()))
22 a.join()
23 b.join()
24 c.join()
25 print("main - thread is done")
```

```
1 Thread-1 - start
2 Thread-1 - count 0
3 Thread-1 - count 1
4 Thread-2 - start
5 Thread-1 - count 2
6 Thread-2 - count 0
7 Thread-1 - count 3
8 Thread-3 - start
9 main - running 4 threads
10 Thread-2 - count 1
11 Thread-1 - count 4
12 Thread-2 - count 2
13 Thread-1 - count 5
14 Thread-2 - count 3
15 Thread-1 - count 6
16 Thread-2 - count 4
17 Thread-1 - count 7
18 Thread-2 - count 5
19 Thread-1 - count 8
20 Thread-2 - count 6
21 Thread-1 - count 9
22 Thread-2 - count 7
23 Thread-1 - end
24 Thread-2 - count 8
25 Thread-2 - count 9
26 Thread-2 - end
27 Thread-3 - count 0
28 Thread-3 - count 1
29 Thread-3 - count 2
30 Thread-3 - count 3
31 Thread-3 - count 4
32 Thread-3 - count 5
33 Thread-3 - count 6
34 Thread-3 - count 7
35 Thread-3 - count 8
```

```
36 Thread-3 - count 9
37 Thread-3 - end
38 main - thread is done
```

Simple threaded counters (parameterized)

The same as the previous one, but with parameters controlling the numbers of threads and the range of the counter.

```
1 import threading
2 import sys
3
4 num_threads, count_till = 3, 5
5
6 class ThreadedCount(threading.Thread):
7     def run(self):
8         thread = threading.current_thread()
9         print(f'{thread.name} - start')
10        for cnt in range(count_till):
11            print(f'{thread.name} - count {cnt}')
12        print(f'{thread.name} - end')
13        return
14
15 threads = []
16 for ix in range(num_threads):
17     threads.append(ThreadedCount())
18
19 for th in threads:
20     th.start()
21
22 print('main - running {}'
23       .format(threading.active_count()))
24
25 for th in threads:
26     th.join()
27 print("main - thread is done")
```

```
1 Thread-1 - start
2 Thread-1 - count 0
```

```
3 Thread-1 - count 1
4 Thread-1 - count 2
5 Thread-1 - count 3
6 Thread-1 - count 4
7 Thread-1 - end
8 Thread-2 - start
9 Thread-2 - count 0
10 Thread-2 - count 1
11 Thread-2 - count 2
12 Thread-2 - count 3
13 Thread-2 - count 4
14 Thread-2 - end
15 Thread-3 - start
16 Thread-3 - count 0
17 Thread-3 - count 1
18 Thread-3 - count 2
19 Thread-3 - count 3
20 Thread-3 - count 4
21 Thread-3 - end
22 main - running 1 threads
23 main - thread is done
```

Pass parameters to threads - Counter with attributes

```
1 import threading
2 import sys
3
4 class ThreadedCount(threading.Thread):
5     def __init__(self, name, start, stop):
6         super().__init__()
7         self.name = name
8         self.counter = start
9         self.limit = stop
10        print('__init__ of {} in
{}'.format(self.name, threading.current_thread()))
11
12    def run(self):
13        print('start run of {} in
{}'.format(self.name, threading.current_thread()))
14        while self.counter < self.limit:
15            print('count {} of {}'.format(self.name,
```

```
self.counter))
16         self.counter += 1
17         print('end run    of {} in {}'
18             .format(self.name,
threading.current_thread()))
19     return
20
21 foo = ThreadedCount("Foo", 1, 11)
22 bar = ThreadedCount("Bar", 1, 11)
23 foo.start()
24 bar.start()
25 print('main - running {}'
threads'.format(threading.active_count()))
26 foo.join()
27 bar.join()
28 print("main - thread is done")
```

```
1 __init__ of Foo in <_MainThread(MainThread, started
139645405484864)>
2 __init__ of Bar in <_MainThread(MainThread, started
139645405484864)>
3 start run of Foo in <ThreadedCount(Foo, started
139645391374080)>
4 count Foo of 1
5 count Foo of 2
6 start run of Bar in <ThreadedCount(Bar, started
139645382981376)>
7 count Bar of 1
8 main - running 3 threads
9 count Foo of 3
10 count Bar of 2
11 count Foo of 4
12 count Bar of 3
13 count Foo of 5
14 count Bar of 4
15 count Foo of 6
16 count Bar of 5
17 count Foo of 7
18 count Bar of 6
19 count Foo of 8
20 count Bar of 7
21 count Foo of 9
22 count Bar of 8
23 count Foo of 10
```

```
24 count Bar of 9
25 end run    of Foo in <ThreadedCount(Foo, started
139645391374080)>
26 count Bar of 10
27 end run    of Bar in <ThreadedCount(Bar, started
139645382981376)>
28 main - thread is done
```

Create a central counter

```
1 import threading
2 import sys
3 import time
4
5 cnt = 0
6 num = 30
7 limit = 100000
8
9 class ThreadedCount(threading.Thread):
10     def __init__(self):
11         threading.Thread.__init__(self)
12         self.counter = 0
13
14     def run(self):
15         global cnt
16         while self.counter < limit:
17             self.counter += 1
18             cnt += 1
19         return
20
21 start = time.time()
22 threads = [ ThreadedCount() for n in range(num) ]
23 [ t.start() for t in threads ]
24 [ t.join() for t in threads ]
25 end = time.time()
26
27 print("Expected: {}".format(num * limit))
28 print("Received: {}".format(cnt))
29 print("Elapsed: {}".format(end-start))
30
31 # Expected: 3000000
```

```
32 # Received: 2659032
33 # Elapsed: 0.437514066696167
```

Lock - acquire - release

```
1 import threading
2 import sys
3 import time
4
5 cnt = 0
6 num = 30
7 limit = 100000
8
9 locker = threading.Lock()
10
11 class ThreadedCount(threading.Thread):
12     def __init__(self):
13         threading.Thread.__init__(self)
14         self.counter = 0
15     def run(self):
16         global cnt
17         while self.counter < limit:
18             self.counter += 1
19             locker.acquire()
20             cnt += 1
21             locker.release()
22         return
23
24 start = time.time()
25 threads = [ ThreadedCount() for n in range(num) ]
26 [ t.start() for t in threads ]
27 [ t.join() for t in threads ]
28 end = time.time()
29
30 print("Expected: {}".format(num * limit))
31 print("Received: {}".format(cnt))
32 print("Elapsed: {}".format(end-start))
33
34 # Expected: 3000000
35 # Received: 3000000
36 # Elapsed: 12.333643198013306
```

Counter - plain

```
1 import sys
2 import time
3
4 cnt = 0
5 num = 30
6 limit = 100000
7
8 class Count():
9     def __init__(self):
10         self.counter = 0
11     def run(self):
12         global cnt
13         while self.counter < limit:
14             self.counter += 1
15             cnt += 1
16         return
17
18 start = time.time()
19 for _ in range(num):
20     c = Count()
21     c.run()
22 end = time.time()
23
24 print("Expected: {}".format(num * limit))
25 print("Received: {}".format(cnt))
26 print("Elapsed: {}".format(end-start))
27
28 # Expected: 3000000
29 # Received: 3000000
30 # Elapsed: 0.4130408763885498
```

GIL - Global Interpreter Lock

- Solves the problem introduced by having reference count.
- Not going away any time soon.
- [GIL wiki](#)
- [GIL realpython](#)

Thread load

```
1 import threading
2 import sys
3 import time
4 import random
5
6
7 results = []
8 locker = threading.Lock()
9
10 class ThreadedCount(threading.Thread):
11     def __init__(self, n):
12         threading.Thread.__init__(self)
13         self.n = n
14
15     def run(self):
16         count = 0
17         total = 0
18         while count < 40000000 / self.n:
19             rnd = random.random()
20             total += rnd
21             count += 1
22
23             locker.acquire()
24             results.append({'count': count, 'total':
25 total})
26             locker.release()
27         return
28
29 def main():
30     if len(sys.argv) != 2:
31         exit("Usage: {} POOL_SIZE")
32     size = int(sys.argv[1])
33     start = time.time()
34     threads = [ ThreadedCount(n=size) for i in
35 range(size) ]
36     [ t.start() for t in threads ]
37     [ t.join() for t in threads ]
38     print("Results: {}".format(results))
39     totals = map(lambda r: r['total'], results)
40     print("Total: {}".format(sum(totals)))
41     end = time.time()
```

```
40     print(end - start)
41
42 if __name__ == '__main__':
43     main()
```

```
1 $ time python thread_load.py 1
2 Results: [{'count': 40000000, 'total':
19996878.531261113}]
3 Total: 19996878.531261113
4 6.478948354721069
5
6 real    0m6.539s
7 user    0m6.491s
8 sys     0m0.012s
```

```
1 $ time python thread_load.py 4
2 Results: [{'count': 10000000, 'total':
5000680.7382364655}, {'count': 10000000, 'tot\
al': 5000496.15077697}, {'count': 10000000, 'total':
5000225.747780174}, {'count': 1\
0000000, 'total': 4999503.803068357}]
3 Total: 20000906.43986197
4 6.180345296859741
5
6 real    0m6.241s
7 user    0m6.283s
8 sys     0m0.029s
```

Exercise: thread files

- Get a list of files (from the current directory or from all the files in the “slides” repository).
- Process each file:
 - 1. get size of file
 - 2. count how many times each character appear in the file.
- The script should accept the number of threads to use.

Exercise: thread URL requests.

In the following script we fetch the URLs listed in a file:

```
1 https://google.com/
2 https://youtube.com/
3 https://facebook.com/
4 https://baidu.com/
5 https://twitter.com/
6 https://instagram.com/
7 https://wikipedia.com/
8 https://www.amazon.com/
9 https://yahoo.com/
10 https://yandex.ru/
11 https://vk.com/
12 https://live.com/
13 https://naver.com/
14 https://yahoo.co.jp/
15 https://google.com.br/
16 https://netflix.com/
17 https://reddit.com/
18 https://ok.ru/
19 https://mail.ru/
20 https://ebay.com/
21 https://linkedin.com/
22 https://qq.com/
23 https://pinterest.com/
24 https://bing.com/
25 https://whatsapp.com/
26 https://office.com/
27 https://amazon.de/
28 https://aliexpress.com/
29 https://amazon.co.jp/
30 https://msn.com/
31 https://google.de/
32 https://paypal.com/
33 https://rakuten.co.jp/
34 https://amazon.co.uk/
35 https://daum.net/
36 https://google.co.jp/
37 https://taobao.com/
38 https://bilibili.com/
39 https://imdb.com/
40 https://booking.com/
```

```
41 https://roblox.com/
42 https://9apps.com/
43 https://globo.com/
44 https://duckduckgo.com/
45 https://www.nttdocomo.co.jp/
```

It takes about 1.5-2 sec / URL from home. (It depends on a lot of factors including your network connection.)

```
1 import time
2 import requests
3 import sys
4 from bs4 import BeautifulSoup
5
6 def get_urls(limit):
7     with open('urls.txt') as fh:
8         urls = list(map(lambda line:
line.rstrip("\n"), fh))
9     if len(urls) > limit:
10        urls = urls[:limit]
11
12    return urls
13
14 def get_title(url):
15    try:
16        resp = requests.get(url)
17        if resp.status_code != 200:
18            return None, f"Incorrect status_code
{resp.status_code} for {url}"
19    except Exception as err:
20        return None, f"Error: {err} for {url}"
21
22    soup = BeautifulSoup(resp.content, 'html.parser')
23    return soup.title.string, None
24
25 def main():
26    if len(sys.argv) < 2:
27        exit(f"Usage: {sys.argv[0]} LIMIT")
28    limit = int(sys.argv[1])
29    urls = get_urls(limit)
30    print(urls)
31    start = time.time()
```

```

32
33     titles = []
34     for url in urls:
35         #print(f"Processing {url}")
36         title, err = get_title(url)
37         if err:
38             print(err)
39         else:
40             print(title)
41         titles.append({
42             "url": url,
43             "title": title,
44             "err": err,
45         })
46     end = time.time()
47     print("Elapsed time: {} for {} pages.".format(end-
start, len(urls)))
48     print(titles)
49
50
51 if __name__ == '__main__':
52     main()

```

Create a version of the above script that can use K threads.

Exercise: thread queue

Write an application that handles a queue of jobs in N=5 threads.
Each job contains a number between 0-5.

Each thread takes the next element from the queue and sleeps for
the given amount
of second (as an imitation of actual work it should be doing).
When finished it checks
for another job. If there are no more jobs in the queue, the thread
can close itself.

```

1 import threading
2 import random
3 import sys

```

```
4
5 thread_count = 5
6
7 counter = 0
8 queue = map(lambda x: ('main', random.randrange(5)),
range(20))
9 print(queue)
```

If that's done, change the code so that each thread will generate a random number between 0-5 (for sleep-time) and in 33% of the cases it will add it to the central queue as a new job.

Another extension to this exercise is to change the code to limit the number of jobs each thread can execute in its lifetime. When the thread has finished that many jobs it will quit and the main thread will create a new worker thread.

Solution: thread queue

```
1 import threading
2 import random
3 import sys
4 import time
5
6 thread_count = 5
7
8 counter = 0
9 queue = list(map(lambda x: ('main',
random.randrange(5)), range(20)))
10 #print(queue)
11
12 locker = threading.Lock()
13
14 class ThreadedCount(threading.Thread):
15     def run(self):
```

```

16     global counter
17     my_counter = 0
18     thread = threading.current_thread()
19     print('{} - start thread'.format(thread.name))
20     while (True):
21         locker.acquire()
22         job = None
23         if len(queue) > 0:
24             counter += 1
25             my_counter += 1
26             job = queue[0]
27             queue[0:1] = []
28         locker.release()
29         if job == None:
30             print('{} - no more
31 jobs'.format(thread.name))
32             break
33
34         print('{} - working on job {} ({}) from {}
35 sleep for {}'.format(thread.name, counter,
36 my_counter, job[0], job[1]))
37         time.sleep(job[1])
38
39     return
40
41 threads = []
42 for i in range(thread_count):
43     threads.append(ThreadedCount())
44 for t in threads:
45     t.start()
46 for t in threads:
47     t.join()

```

Solution: thread URL requests.

```

1 import time
2 import threading
3 import requests
4 import sys
5 from bs4 import BeautifulSoup
6

```

```

7 from fetch_urls import get_urls, get_title
8
9 titles = []
10 locker = threading.Lock()
11
12 class GetURLs(threading.Thread):
13     def __init__(self, urls):
14         threading.Thread.__init__(self)
15         self.urls = urls
16
17     def run(self):
18         my_titles = []
19         for url in self.urls:
20             title, err = get_title(url)
21             my_titles.append({
22                 'url': url,
23                 'title': title,
24                 'err': err,
25             })
26         locker.acquire()
27         titles.extend(my_titles)
28         locker.release()
29     return
30
31 def main():
32     if len(sys.argv) < 3:
33         exit(f"Usage: {sys.argv[0]} LIMIT THREADS")
34     limit = int(sys.argv[1])
35     threads_count = int(sys.argv[2])
36
37     urls = get_urls(limit)
38     print(urls)
39     start_time = time.time()
40     batch_size = int(limit/threads_count)
41     left_over = limit % threads_count
42     batches = []
43     end = 0
44     for ix in range(threads_count):
45         start = end
46         end = start + batch_size
47         if ix < left_over:
48             end += 1
49         batches.append(urls[start:end])
50

```

```
51     threads = [ GetURLs(batches[ix]) for ix in
52         range(threads_count) ]
53         [ t.start() for t in threads ]
54         [ t.join() for t in threads ]
55
56     end_time = time.time()
57     print("Elapsed time: {} for {}"
58         pages.format(end_time-start_time, len(urls)))
59     print(titles)
60 if __name__ == '__main__':
61     main()
```

Forking

Fork

- fork

```
1 import os
2 import time
3
4 print('{} - start running'.format(os.getpid()))
5
6 pid = os.fork()
7 if not pid:
8     print('{} - in child. Parent is
{}'.format(os.getpid(), os.getppid()))
9     time.sleep(1)
10    exit(3)
11
12 print('{} - in parent (child pid is
{})'.format(os.getpid(), pid))
13
14 child_pid, exit_code = os.wait()
15 print('{} - Child with pid {} exited. Exit code
{}'.format(os.getpid(), child_pid, e\
xit_code))
16 print('Real exit code {}'.format(int(exit_code/256)))
# The upper byte
17 print('Also known as {}'.format(exit_code >> 8)) # Right shift 8 bits
```

```
1 10278 - start running
2 10279 - in child. Parent is 10278
3 10278 - start running
4 10278 - in parent (child pid is 10279)
5 10278 - Child with pid 10279 exited. Exit code 768
6 Real exit code 3
7 Also known as 3
```

Forking

```
1 import os
2 import time
3
4 name = "common"
5
6 def child():
7     print("In Child of {}".format(name))
8     print("In Child PID: {} PPID: {}".
9           format(os.getpid(), os.getppid()))
10    time.sleep(5)
11    exit(3)
12
13 def parent(child_pid):
14     print("In Parent ({}) The child is: {}".
15           format(name, child_pid))
16     print("In Parent PID: {} PPID: {}".
17           format(os.getpid(), os.getppid()))
18     r = os.wait()
19     print(r)
20
21 pid = os.fork()
22 print(pid)
23 if pid == 0:
24     child()
25 else:
26     parent(pid)
```

```
1 0
2 In Child of common
3 In Child PID: 11212 PPID: 11211
4 11212
5 In Parent (common) The child is: 11212
6 In Parent PID: 11211 PPID: 4195
7 (11212, 768)
```

Fork skeleton

```
1 import os
2 import glob
3
4 files = glob.glob("*.py")
5 # print(files)
6 count = len(files)
7 print(f"Number of items to process: {count}")
8
9 parallel = 4      # How many in parallel
10
11 batch = int(count/parallel)
12 leftover = count % parallel
13 print(f"batch size: {batch}  leftover: {leftover}")
14
15 def parent(pid):
16     print(f"parent {pid}")
17
18 def child(files):
19     print(f"{os.getpid()} {files}")
20     exit()
21
22 end = 0
23 for ix in range(parallel):
24     start = end
25     end   = start + batch
26     if ix < leftover:
27         end += 1
28     print(f"start={start} end={end}")
29
30 pid = os.fork()
31 if pid:
32     parent(pid)
33 else:
34     child(files[start:end])
35
36 print(f"In parent {os.getpid()}")
37 for ix in range(parallel):
38     r = os.wait()
39     print(r)
```

Fork with load

```
1 import os
2 import random
3 import sys
4
5 if len(sys.argv) != 2:
6     exit("Usage: {} N".format(sys.argv[0]))
7 n = int(sys.argv[1])
8 for p in range(0, n):
9     pid = os.fork()
10    if not pid:
11        print('In Child')
12        i = 0
13        while i < 40000000/n:
14            x = random.random()
15            y = random.random()
16            z = x+y
17            i += 1
18        exit(3)
19    print('In Parent of', pid)
20
21 for p in range(0, n):
22     r = os.wait()
23     print(r)
```

Fork load results

```
1 $ time python fork_load.py 1
```

```
1 In Parent of 96355
2 In Child
3 (96355, 768)
4
5 real    0m26.391s
6 user    0m25.893s
7 sys    0m0.190s
```

```
1 $ time python fork_load.py 8
```

```
1 In Parent of 96372
2 In Parent of 96373
3 In Parent of 96374
4 In Child
5 In Child
6 In Parent of 96375
7 In Child
8 In Child
9 In Parent of 96376
10 In Child
11 In Parent of 96377
12 In Child
13 In Child
14 In Parent of 96378
15 In Parent of 96379
16 In Child
17 (96374, 768)
18 (96372, 768)
19 (96375, 768)
20 (96373, 768)
21 (96376, 768)
22 (96377, 768)
23 (96378, 768)
24 (96379, 768)
25
26 real      0m12.754s
27 user      0m45.196s
28 sys       0m0.164s
```

Marshalling / Serialization

Marshalling (or serialization) is the operation when we take an arbitrary data structure and convert it into a string in a way that we can convert the string back to the same data structure.

Marshalling can be used to save data persistent between execution of the same script, to transfer data between processes, or even between

machines.

In some cases it can be used to communicate between two processes written in different programming languages.

The [marshal](#) module

provides such features but it is not recommended as it was built for internal object serialization for python.

The [pickle](#) module was designed for this task.

The [json](#) module can be used too.

Fork with random

When the **random** module is loaded it automatically calls `random.seed()` to initialize the random generator. When we create a fork this is not called again and thus all the processes will return the same random numbers. We can fix this by calling `random.seed()` manually.

```
1 import os, time, random
2
3 print('{} - start running'.format(os.getpid()))
4
5 pid = os.fork()
6 if not pid:
7     #random.seed()
8     print('{} - in child'.format(os.getpid()))
9     print(random.random())
10    time.sleep(1)
11    exit(3)
12
13 print('{} - in parent (child pid is
14 {}'.format(os.getpid(), pid))
15 print(random.random())
```

```
15
16 done = os.wait()
17 print('{} - Child exited {}'.format(os.getpid(), done))
```

Exercise: fork return data

Create a script that will go over a list of numbers and does some computation on each number.

```
1 import sys
2 import time
3 from mymodule import calc
4
5 def main(n):
6     results = {}
7     print(f"do 1-{n}")
8     for ix in range(1, n):
9         results[ix] = calc(ix)
10    return results
11
12 if __name__ == '__main__':
13     if len(sys.argv) < 2:
14         exit(f"Usage: {sys.argv[0]} NUMBER")
15
16     start = time.time()
17     results = main(1+int(sys.argv[1]))
18     end = time.time()
19     total = sum(results.values())
20     print(f"Total: {total}")
21     print("Elapsed time: {}".format(end-start))
```

Allow the child process to return data to the parent process.
Before exiting from the child process, serialize the data-structure you want to send back and save in a file that corresponds to the parent process and the child process. (eg. created from the PID of the parent process and the PID of the child process)

In the parent process, when one of the children exits, check if there is a file corresponding to this child process, read the file and de-serialize it.

Solution: fork return data

```
1 import sys
2 import os
3 import json
4 import time
5 from mymodule import calc
6
7 def child(start, end):
8     results = {}
9     for ix in range(start, end):
10         results[ix] = calc(ix)
11     filename = str(os.getpid()) + '.json'
12     with open(filename, 'w') as fh:
13         json.dump(results, fh)
14     exit()
15
16 def main(total_number, parallels):
17     results = {}
18
19     processes = []
20     a_range = int(total_number / parallels)
21     for cnt in range(parallels):
22         start = 1 + cnt * a_range
23         end = start + a_range
24         if cnt == parallels - 1:
25             end = total_number + 1
26         print(f"do: {start}-{end}")
27         pid = os.fork()
28         if pid:
29             processes.append(pid) # parent
30         else:
31             child(start, end)
32     for _ in range(len(processes)):
33         pid, exit_code = os.wait()
34         #print(pid, exit_code)
35         filename = str(pid) + '.json'
36         with open(filename) as fh:
```

```
37         res = json.load(fh)
38         print(f"pid: {res}")
39         results.update(res)
40         os.unlink(filename)
41     return results
42
43 if __name__ == '__main__':
44     if len(sys.argv) < 3:
45         exit(f"Usage: {sys.argv[0]} NUMBER PARALLEL")
46
47     start = time.time()
48     results = main(int(sys.argv[1]), int(sys.argv[2]))
49     print(f"results: {results}")
50     end = time.time()
51     total = sum(results.values())
52     print(f"Total: {total}")
53     print("Elapsed time: {}".format(end-start))
```

Asynchronous programming with AsyncIO

Sync chores

We have a number of household chores to do. Each takes a couple of seconds for a machine to do while we have time to do something else. We also have one task, cleaning potatoes, that requires our full attention. It is a CPU-intensive process.

We also have two processes depending each other. We can turn on the dryer only after the washing machine has finished.

```
1 import time
2
3 def boil_water(sec):
4     print(f"Start boiling water for {sec} seconds")
5     time.sleep(sec)
6     print(f"End boiling water for {sec} seconds")
7
8 def washing_machine(sec):
9     print("Start washing machine")
10    time.sleep(sec)
11    print("End washing machine")
12
13 def dryer(sec):
14     print("Start dryer")
15     time.sleep(sec)
16     print("End dryer")
17
18 def dishwasher(sec):
19     print("Start dishwasher")
```

```
20     time.sleep(sec)
21     print("End dishwasher")
22
23 def clean_potatoes(pieces):
24     print("Start cleaning potatoes")
25     for ix in range(pieces):
26         print(f"Cleaning potato {ix}")
27         time.sleep(0.5)
28     print("End cleaning potatoes")
29
30 def main():
31     dishwasher(3)
32     washing_machine(3)
33     dryer(3)
34     boil_water(4)
35     clean_potatoes(14)
36
37 start = time.time()
38 main()
39 end = time.time()
40 print(f"Elapsed {end-start}")
```

```
1 Start dishwasher
2 End dishwasher
3 Start washing machine
4 End washing machine
5 Start dryer
6 End dryer
7 Start boiling water for 4 seconds
8 End boiling water for 4 seconds
9 Start cleaning potatoes
10 Cleaning potato 0
11 Cleaning potato 1
12 Cleaning potato 2
13 Cleaning potato 3
14 Cleaning potato 4
15 Cleaning potato 5
16 Cleaning potato 6
17 Cleaning potato 7
18 Cleaning potato 8
19 Cleaning potato 9
20 Cleaning potato 10
21 Cleaning potato 11
22 Cleaning potato 12
```

```
23 Cleaning potato 13
24 End cleaning potatoes
25 Elapsed 20.017353534698486
```

Async chores

```
1 import time
2 import asyncio
3
4 async def boil_water(sec):
5     print(f"Start boiling water for {sec} seconds")
6     await asyncio.sleep(sec)
7     print(f"End boiling water for {sec} seconds")
8
9 async def washing_machine(sec):
10    print(f"Start washing machine for {sec} seconds")
11    await asyncio.sleep(sec)
12    print(f"End washing machine for {sec} seconds")
13    await dryer(3)
14
15 async def dryer(sec):
16    print(f"Start dryer for {sec} seconds")
17    await asyncio.sleep(sec)
18    print(f"End dryer for {sec} seconds")
19
20 async def dishwasher(sec):
21    print(f"Start dishwasher for {sec} seconds")
22    await asyncio.sleep(sec)
23    print(f"End dishwasher for {sec} seconds")
24
25 async def clean_potatoes(pieces):
26    print(f"Start cleaning potatoes for {pieces} pieces")
27    for ix in range(pieces):
28        print(f"Cleaning potato {ix}")
29        time.sleep(0.5)
30        #await asyncio.sleep(0.0001)
31    print(f"End cleaning potatoes for {pieces} pieces")
32
33 async def main():
34     await asyncio.gather(dishwasher(3),
```

```
washing_machine(3), boil_water(4), clean_pot\  
35 atoes(14))  
36  
37 start = time.time()  
38 asyncio.run(main())  
39 end = time.time()  
40 print(f"Elapsed {end-start}")
```

From the output you can see that we noticed that the washing machine has finished only after we have finished all the potatoes. That's because our potato cleaning process was a long-running CPU-intensive process. This means the dryer only starts working after the potatoes are clean.

```
1 Start dishwasher for 3 seconds  
2 Start washing machine for 3 seconds  
3 Start boiling water for 4 seconds  
4 Start cleaning potatoes for 14 pieces  
5 Cleaning potato 0  
6 Cleaning potato 1  
7 Cleaning potato 2  
8 Cleaning potato 3  
9 Cleaning potato 4  
10 Cleaning potato 5  
11 Cleaning potato 6  
12 Cleaning potato 7  
13 Cleaning potato 8  
14 Cleaning potato 9  
15 Cleaning potato 10  
16 Cleaning potato 11  
17 Cleaning potato 12  
18 Cleaning potato 13  
19 End cleaning potatoes for 14 pieces  
20 End dishwasher for 3 seconds  
21 End washing machine for 3 seconds  
22 Start dryer for 3 seconds  
23 End boiling water for 4 seconds  
24 End dryer for 3 seconds  
25 Elapsed 10.01340126991272
```

If after cleaning each potato we look up for a fraction of a second, if we let the main loop run, then we can notice that the washing machine has ended and we can turn on the dryer before continuing with the next potato. This will allow the dryer to work while we are still cleaning the potatoes.

```
1 Start dishwasher for 3 seconds
2 Start washing machine for 3 seconds
3 Start boiling water for 4 seconds
4 Start cleaning potatoes for 14 pieces
5 Cleaning potato 0
6 Cleaning potato 1
7 Cleaning potato 2
8 Cleaning potato 3
9 Cleaning potato 4
10 Cleaning potato 5
11 End dishwasher for 3 seconds
12 End washing machine for 3 seconds
13 Start dryer for 3 seconds
14 Cleaning potato 6
15 Cleaning potato 7
16 End boiling water for 4 seconds
17 Cleaning potato 8
18 Cleaning potato 9
19 Cleaning potato 10
20 Cleaning potato 11
21 End dryer for 3 seconds
22 Cleaning potato 12
23 Cleaning potato 13
24 End cleaning potatoes for 14 pieces
25 Elapsed 7.02296781539917
```

Explanation

- Single thread
- Single process
- The feeling of parallelism
- Coroutines

- * `async/await`
- * event loop
- * Cooperative Multitasking
 - Asynchronous
 - non-blocking or synchronous vs blocking (aka “normal”)

Coroutines

- * Functions that can be suspended mid-way and allow other functions to run (a generator)
 - `async def` is a native coroutine or asynchronous generator
 - `async with`
 - `async for`

More about `asyncio`

- [AsyncIO in Real Python](#)
- [asyncio](#)
- [aiohttp](#)

Async files

```

1 import aiohttp
2 import asyncio
3
4 async def fetch(session, url):
5     async with session.get(url) as response:
6         return await response.text()
7
8 async def main():
9     async with aiohttp.ClientSession() as session:

```

```
10         html = await fetch(session,
11             'http://python.org')
12             print(html)
13             print("OK")
14 asyncio.run(main())
```

```
1 import aiofiles
```

Asynchronous programming with Twisted

About Twisted

- [Twisted](#)

Echo

```
1 from twisted.internet import protocol,reactor
2
3 port = 8000
4
5 class Echo(protocol.Protocol):
6     def dataReceived(self, data):
7         text = data.decode('utf8')
8         print(f"Received: {text}")
9         self.transport.write("You said:
{ }".format(text).encode('utf8'))
10
11 class EchoFactory(protocol.Factory):
12     def buildProtocol(self, addr):
13         return Echo()
14
15 print(f"Listening on port {port}")
16 reactor.listenTCP(port, EchoFactory())
17 reactor.run()
```

```
1 from twisted.internet import reactor,protocol
2 import sys
3
4 if len(sys.argv) < 2:
5     exit("Usage: {sys.argv[0]} TEXT")
6
7 message = sys.argv[1]
```

```

8 port = 8000
9
10 class EchoClient(protocol.Protocol):
11     def connectionMade(self):
12         self.transport.write(message.encode('utf8'))
13
14     def dataReceived(self, data):
15         print(f"Server said: {data}")
16         self.transport.loseConnection()
17
18 class EchoFactory(protocol.ClientFactory):
19     def buildProtocol(self, addr):
20         return EchoClient()
21
22     def clientConnectionFailed(self, connector,
23 reason):
24         print("connection failed")
25         reactor.stop()
26
27     def clientConnectionLost(self, connector, reason):
28         print("connection lost")
29         reactor.stop()
30
31 reactor.connectTCP("localhost", port, EchoFactory())
32 reactor.run()

```

Echo with log

```

1 from twisted.internet import protocol,reactor
2
3 port = 8000
4
5 class Echo(protocol.Protocol):
6     def dataReceived(self, data):
7         print("Received: {}".format(data))
8         self.transport.write(data)
9
10 class EchoFactory(protocol.Factory):
11     def buildProtocol(self, addr):
12         print(f"Connection established with {addr}")
13         return Echo()
14

```

```
15 print(f"Started to listen on port {port}")  
16 reactor.listenTCP(port, EchoFactory())  
17 reactor.run()
```

Simple web client

The code behind this example was deprecated. Need to be fixed.

- getPage() returns a “deferred”
 - addCallbacks(on_success, on_failure)
 - addBoth(on_both) adds callbock to both success and failure callback chain
-

```
1 from twisted.internet import reactor  
2 from twisted.web.client import getPage  
3 import sys  
4  
5 def printPage(result):  
6     print("Page")  
7     print('Size of the returned page is  
{ }'.format(len(result)))  
8  
9 def printError(error):  
10    print("Error")  
11    print(f"Error: {error}")  
12    #sys.stderr.write(error)  
13  
14 def stop(result):  
15     print('stop')  
16     reactor.stop()  
17  
18 if (len(sys.argv) != 2):  
19     sys.stderr.write("Usage: python " + sys.argv[0] +  
" <URL>\n")  
20     exit(1)  
21  
22 d = getPage(sys.argv[1])  
23 d.addCallbacks(printPage, printError)  
24 d.addBoth(stop)  
25
```

```
26 reactor.run()
27
28 # getPage(sys.argv[1], method='POST', postdata="My
test data").
```

Web client

```
1 from twisted.internet import reactor
2 from twisted.web.client import getPage
3 import sys
4 import re
5 import time
6
7 queue = [
8     'http://docs.python.org/3/',
9     'http://docs.python.org/3/whatsnew/3.3.html',
10    'http://docs.python.org/3/tutorial/index.html',
11    'http://docs.python.org/3/library/index.html',
12    'http://docs.python.org/3/reference/index.html'
13    'http://docs.python.org/3/howto/index.html',
14    'http://docs.python.org/3/howto/pyporting.html',
15    'http://docs.python.org/3/howto/cporting.html',
16    'http://docs.python.org/3/howto/curses.html',
17    'http://docs.python.org/3/howto/descriptor.html',
18    'http://docs.python.org/3/howto/functional.html',
19    'http://docs.python.org/3/howto/logging.html',
20    'http://docs.python.org/3/howto/logging-
cookbook.html',
21    'http://docs.python.org/3/howto/regex.html',
22    'http://docs.python.org/3/howto/sockets.html',
23    'http://docs.python.org/3/howto/sorting.html',
24    'http://docs.python.org/3/howto/unicode.html',
25    'http://docs.python.org/3/howto/urllib2.html',
26    'http://docs.python.org/3/howto/webservers.html',
27    'http://docs.python.org/3/howto/argparse.html',
28    'http://docs.python.org/3/howto/ipaddress.html',
29 ]
30
31 max_parallel = 3
32 current_parallel = 0
33 if len(sys.argv) == 2:
34     max_parallel = int(sys.argv[1])
35
```

```
36 def printPage(result):
37     print("page size: ", len(result))
38     global current_parallel
39     current_parallel -= 1
40     print("current_parallel: ", current_parallel)
41     #urls = re.findall(r'href="([^\"]+)"', result)
42     #for u in urls:
43     #    queue.append(u)
44     #queue.extend(urls)
45     process_queue()
46
47 def printError(error):
48     print("Error: ", error)
49     global current_parallel
50     current_parallel -= 1
51     process_queue()
52
53
54 def stop(result):
55     reactor.stop()
56
57 def process_queue():
58     global current_parallel, max_parallel, queue
59     print("process_queue cs: {} max: {}".
60           format(current_parallel, max_parallel))
61     while True:
62         if current_parallel >= max_parallel:
63             print("No empty slot")
64             return
65         if len(queue) == 0:
66             print("queue is empty")
67             if current_parallel == 0:
68                 reactor.stop()
69             return
70         url = queue[0] + '?' + str(time.time())
71         queue[0:1] = []
72         current_parallel += 1
73         d = getPage(url)
74         d.addCallbacks(printPage, printError)
75
76 process_queue()
77 reactor.run()
78 print("----done ---")
```

Multiprocess

Multiprocess CPU count

- [multiprocessing](#)

```
1 import multiprocessing as mp
2 print(mp.cpu_count())
```

Multiprocess Process

```
1 import multiprocessing as mp
2 print(mp.cpu_count())
```

Multiprocess N files: Pool

Analyze N files in parallel.

```
1 from multiprocessing import Pool
2 import os
3 import sys
4 import re
5
6 def analyze(filename):
7     print("Process {:>5} analyzing
{}".format(os.getpid(), filename))
8     digits = 0
9     spaces = 0
10    total = 0
11    with open(filename) as fh:
12        for line in fh:
13            for char in line:
14                total += 1
15                if re.search(r'^\d$', char):
```

```
16             digits += 1
17             if char == ' ':
18                 spaces += 1
19         return {
20             'filename': filename,
21             'total': total,
22             'digits': digits,
23             'spaces': spaces,
24         }
25
26 def main():
27     if len(sys.argv) < 3:
28         exit("Usage: {} POOL_SIZE FILES")
29     size = int(sys.argv[1])
30     files = sys.argv[2:]
31
32     with Pool(size) as p:
33         results = p.map(analyze, files)
34     for res in results:
35         print(res)
36
37 if __name__ == '__main__':
38     main()
```

```
1 $ python multiprocess_files.py 3 multiprocess_*
2
3 Process 22688 analyzing multiprocess_files.py
4 Process 22689 analyzing multiprocess_load.py
5 Process 22690 analyzing multiprocess_pool_async.py
6 Process 22688 analyzing multiprocess_pool.py
7 {'filename': 'multiprocess_files.py', 'total': 833,
8 'digits': 10, 'spaces': 275}
9 {'filename': 'multiprocess_load.py', 'total': 694,
10 'digits': 14, 'spaces': 163}
11 {'filename': 'multiprocess_pool_async.py', 'total':
12 695, 'digits': 8, 'spaces': 161}
13 {'filename': 'multiprocess_pool.py', 'total': 397,
14 'digits': 3, 'spaces': 80}
```

We asked it to use 3 processes, so looking at the process ID you can see one of them worked twice.

The returned results can be any Python datastructure. A dictionary is usually a good idea.

Multiprocess load

```
1 import random
2 import multiprocessing
3 import time
4 import sys
5 # Works only in Python 3
6
7 def calc(n):
8     count = 0
9     total = 0
10    while count < 40000000 / n:
11        rnd = random.random()
12        total += rnd
13        count += 1
14    return {'count': count, 'total': total}
15
16 def main():
17    if len(sys.argv) != 2:
18        exit("Usage: {} POOL_SIZE")
19
20    start = time.time()
21    size = int(sys.argv[1])
22    with multiprocessing.Pool(size) as pool:
23        results = pool.map(calc, [size] * size)
24        print("Results: {}".format(results))
25        totals = map(lambda r: r['total'], results)
26        print("Total: {}".format(sum(totals)))
27    end = time.time()
28    print(end - start)
29
30 if __name__ == '__main__':
31     main()
```

Multiprocess: Pool

Pool(3) creates 3 child-processes and let's them compute the values. map returns the results in the same order as the input came in.

```
1 from multiprocessing import Pool
2 import os
3 import sys
4
5 def f(x):
6     print("Input {} in process {}".format(x,
os.getpid()))
7     #print(x)
8     return x*x
9
10 def main():
11     if len(sys.argv) != 3:
12         exit("Usage: {} NUMBERS POOL_SIZE")
13     numbers = int(sys.argv[1])
14     size     = int(sys.argv[2])
15
16     with Pool(size) as p:
17         results = p.map(f, range(numbers))
18     print(results)
19
20 if __name__ == '__main__':
21     main()
```

```
1 python multiprocess_pool.py 11 3
2 python multiprocess_pool.py 100 5
```

Multiprocess load async

```
1 from multiprocessing import Pool
2 import os
3
4
5 def f(x):
6     print("Input {} in process {}".format(x,
```

```

os.getpid()))
7     return x*x
8
9 def prt(z):
10    print(z)
11
12 def main():
13     with Pool(5) as p:
14         results = p imap(f, range(11)) #
<multiprocessing.pool.IMapIterator object
15         print(results)
16         print('---')
17         for r in results:
18             print(r)
19
20         #results = p.map_async(f, range(11)) #
<multiprocessing.pool.MapResult obje\
21 ct>, not iterable
22
23         #results = []
24         #p.map_async(f, range(11)) #
<multiprocessing.pool.MapResult object>, not i\
25 terable
26         #print(results)
27         #for r in results:
28             #    print(r)
29
30
31 if __name__ == '__main__':
32     main()

```

Multiprocess and logging

Tested on Windows

```

1 from multiprocessing import Pool
2 import os
3 import logging
4 import logging.handlers
5
6 count = 0
7 def f(x):

```

```
8     global count
9     count += 1
10    #print("Input {} in process {}".format(x,
11        os.getpid())))
11    logger = logging.getLogger("app")
12    logger.info("f({}) count {} in PID {}".format(x,
13        count, os.getpid())))
14    return x*x
15
16 def prt(z):
17     print(z)
18
19 def setup_logger():
20     level = logging.DEBUG
21     logger = logging.getLogger("app")
22     logger.setLevel(level)
23     log_file = 'try.log'
24     formatter = logging.Formatter('%(asctime)s - %
25 (%levelname)-8s - %(filename)-20s:%(\n
26 lineno)-5d - %(funcName)-22s - %(message)s')
27     ch = logging.FileHandler(log_file)
28     #ch =
29     #logging.handlers.TimedRotatingFileHandler(log_file,
30     #when='D', backupCount=2)
31     ch.setLevel(level)
32     ch.setFormatter(formatter)
33     logger.addHandler(ch)
34     logger.info("Setup logger in PID
35     {}".format(os.getpid())))
36
37 def main():
38     logger = logging.getLogger('app')
39     logger.info("main")
40
41     with Pool(5) as p:
42         results = p imap(f, range(110)) #
43             <multiprocessing.pool.IMapIterator object
44         print(results)
45         print('--')
46         for r in results:
47             print(r)
48
49 setup_logger()
```

```
45 if __name__ == '__main__':
46     main()
```

Exercise: Process N files in parallel

Create N=100 files 1.txt - N.txt

In each file put L random strings of up to X characters

Write a script that will read all the files for each file and count how many times each digit appears. Then provide a combined report. First write the script in a single process way.
Then convert it to be able to work with multiprocess.

Exercise: Process N Excel files in parallel

- Create N Excel files with random 10 random numbers in the first row of each file.
- Write a process that reads the N Excel files and sums up the numbers in each one of them and then sums up the numbers of all the files.

Exercise: Fetch URLs in parallel

- [top-websites](#)
- Given a file with a list of URLs, collect the title of each site.

```
1 https://google.com/
2 https://youtube.com/
3 https://facebook.com/
4 https://baidu.com/
5 https://twitter.com/
6 https://instagram.com/
7 https://wikipedia.com/
8 https://www.amazon.com/
9 https://yahoo.com/
```

```
10 https://yandex.ru/
11 https://vk.com/
12 https://live.com/
13 https://naver.com/
14 https://yahoo.co.jp/
15 https://google.com.br/
16 https://netflix.com/
17 https://reddit.com/
18 https://ok.ru/
19 https://mail.ru/
20 https://ebay.com/
21 https://linkedin.com/
22 https://qq.com/
23 https://pinterest.com/
24 https://bing.com/
25 https://whatsapp.com/
26 https://office.com/
27 https://amazon.de/
28 https://aliexpress.com/
29 https://amazon.co.jp/
30 https://msn.com/
31 https://google.de/
32 https://paypal.com/
33 https://rakuten.co.jp/
34 https://amazon.co.uk/
35 https://daum.net/
36 https://google.co.jp/
37 https://taobao.com/
38 https://bilibili.com/
39 https://imdb.com/
40 https://booking.com/
41 https://roblox.com/
42 https://9apps.com/
43 https://globoplay.globo.com/
44 https://duckduckgo.com/
45 https://www.nttdocomo.co.jp/
```

```
1 import time
2 import requests
3 import sys
4 from bs4 import BeautifulSoup
5
6 def get_urls(limit):
7     with open('urls.txt') as fh:
```

```
8         urls = list(map(lambda line:
9             line.rstrip("\n"), fh))
10            if len(urls) > limit:
11                urls = urls[:limit]
12
13
14 def get_title(url):
15     try:
16         resp = requests.get(url)
17         if resp.status_code != 200:
18             return None, f"Incorrect status_code
{resp.status_code} for {url}"
19     except Exception as err:
20         return None, f"Error: {err} for {url}"
21
22     soup = BeautifulSoup(resp.content, 'html.parser')
23     return soup.title.string, None
24
25 def main():
26     if len(sys.argv) < 2:
27         exit(f"Usage: {sys.argv[0]} LIMIT")
28     limit = int(sys.argv[1])
29     urls = get_urls(limit)
30     print(urls)
31     start = time.time()
32
33     titles = []
34     for url in urls:
35         #print(f"Processing {url}")
36         title, err = get_title(url)
37         if err:
38             print(err)
39         else:
40             print(title)
41         titles.append({
42             "url": url,
43             "title": title,
44             "err": err,
45         })
46     end = time.time()
47     print("Elapsed time: {} for {} pages.".format(end-
start, len(urls)))
48     print(titles)
```

```
49
50
51 if __name__ == '__main__':
52     main()
```

Exercise: Fetch URLs from one site.

Download the [sitemap](#) or the other [sitemap](#) file and fetch the first N URLs from there. Collecting the titles.

```
1 import time
2 import requests
3 import xml.etree.ElementTree as ET
4 from bs4 import BeautifulSoup
5
6 def get_urls(content):
7     urls = []
8     root = ET.fromstring(content)
9     for child in root:
10         for ch in child:
11             if ch.tag.endswith('loc'):
12                 urls.append(ch.text)
13 #print(len(urls)) # 2653
14 MAX = 20
15 if len(urls) > MAX:
16     urls = urls[:MAX]
17
18 return urls
19
20 def main():
21     start = time.time()
22     url = 'https://code-maven.com/slides/sitemap.xml'
23     resp = requests.get(url)
24     if resp.status_code != 200:
25         exit(f"Incorrect status_code {resp.status_code}")
26
27     urls = get_urls(resp.content)
28
29     titles = []
30     for url in urls:
31         resp = requests.get(url)
```

```

32         if resp.status_code != 200:
33             print(f"Incorrect status_code
{resp.status_code} for {url}")
34             continue
35
36         soup = BeautifulSoup(resp.content,
37             'html.parser')
37         print(soup.title.string)
38         titles.append(soup.title.string)
39     end = time.time()
40     print("Elapsed time: {} for {} pages.".format(end-
start, len(urls)))
41     print(titles)
42
43
44 if __name__ == '__main__':
45     main()

```

Solution: Fetch URLs in parallel

- First create function and use regular map.
- Deal with encoding.
- Replace continue by return, include None in results.
- It has some 2 sec overhead, but then 20 items reduced from 18 sec to 7 sec using pool of 5.

```

1 import time
2 import requests
3 import xml.etree.ElementTree as ET
4 from bs4 import BeautifulSoup
5 from multiprocessing import Pool
6 import os
7
8
9 def get_urls(content):
10     urls = []
11     root = ET.fromstring(content)
12     for child in root:
13         for ch in child:
14             if ch.tag.endswith('loc'):

```

```
15             urls.append(ch.text)
16
17     #print(len(urls)) # 2653
18     MAX = 20
19     if len(urls) > MAX:
20         urls = urls[:MAX]
21
22     return urls
23
24 def get_title(url):
25     resp = requests.get(url)
26     if resp.status_code != 200:
27         print(f"Incorrect status_code
{resp.status_code} for {url}")
28     return
29
30     soup = BeautifulSoup(resp.content, 'html.parser')
31     print(soup.title.string)
32     return soup.title.string.encode('utf-8')
33
34
35 def main():
36     start = time.time()
37     url = 'https://code-maven.com/slides/sitemap.xml'
38     resp = requests.get(url)
39     if resp.status_code != 200:
40         exit(f"Incorrect status_code
{resp.status_code}")
41
42     urls = get_urls(resp.content)
43
44     titles = []
45     #     for url in urls:
46     #         titles.append(get_title(url))
47     #     titles = list(map(get_title, urls))
48     with Pool(5) as pool:
49         results = pool.map(get_title, urls)
50     for r in results:
51         titles.append(r)
52     end = time.time()
53     print("Elapsed time: {} for {} pages.".format(end-
start, len(urls)))
54     print(list(titles))
55     print("DONE")
```

```
56  
57  
58 if __name__ == '__main__':  
59     main()  
-----
```

Multitasking

What is Multitasking?

- [Multitasking](#)
- A wrapper around threading and os.fork by Ran Aroussi

```
1 pip install multitasking
```

Multitasking example

```
1 import multitasking
2 import time
3 import random
4
5 multitasking.set_max_threads(2)
6
7 @multitasking.task
8 def work(ix, sec):
9     print(f"Start {ix} sleeping for {sec}s")
10    time.sleep(sec)
11    print(f"Finish {ix}")
12
13 if __name__ == "__main__":
14     tasks = (6, 0.7, 0.8, 0.3, 0.4, 3, 0.1)
15     for ix, sec in enumerate(tasks):
16         work(ix+1, sec)
17
18     print("do some work after all the jobs are done")
```

```
1 Start 1 sleeping for 6s
2 Start 2 sleeping for 0.7s
3 do some work after all the jobs are done
4 Finish 2
5 Start 3 sleeping for 0.8s
```

```
6 Finish 3
7 Start 4 sleeping for 0.3s
8 Finish 4
9 Start 5 sleeping for 0.4s
10 Finish 5
11 Start 6 sleeping for 3s
12 Finish 6
13 Start 7 sleeping for 0.1s
14 Finish 7
15 Finish 1
```

Multitasking example with wait

```
1 import multitasking
2 import time
3 import random
4
5 multitasking.set_max_threads(2)
6
7 @multitasking.task
8 def work(ix, sec):
9     print(f"Start {ix} sleeping for {sec}s")
10    time.sleep(sec)
11    print(f"Finish {ix}")
12
13 if __name__ == "__main__":
14     tasks = (6, 0.7, 0.8, 0.3, 0.4, 3, 0.1)
15     for ix, sec in enumerate(tasks):
16         work(ix+1, sec)
17         multitasking.wait_for_tasks()
18
19     print("do some work after all the jobs are done")
```

```
1 Start 1 sleeping for 6s
2 Start 2 sleeping for 0.7s
3 Finish 2
4 Start 3 sleeping for 0.8s
5 Finish 3
6 Start 4 sleeping for 0.3s
7 Finish 4
8 Start 5 sleeping for 0.4s
9 Finish 5
```

```
10 Start 6 sleeping for 3s
11 Finish 6
12 Start 7 sleeping for 0.1s
13 Finish 7
14 Finish 1
15 do some work after all the jobs are done
```

Multitaksing - second loop waits for first one

```
1 import multitasking
2 import time
3 import random
4
5 @multitasking.task
6 def first(count):
7     sleep = random.randint(1,10)/2
8     if count == 10:
9         sleep = 10
10    print("Start First {} (sleeping for
11    {}.format(count, sleep))
12    time.sleep(sleep)
13    print("finish First {} (after for
14    {}.format(count, sleep))
15
16 @multitasking.task
17 def second(count):
18     sleep = random.randint(1,10)/2
19     print("Start Second {} (sleeping for
20    {}.format(count, sleep))
21     time.sleep(sleep)
22     print("finish Second {} (after for
23    {}.format(count, sleep))
24
25 if __name__ == "__main__":
26     for i in range(0, 10):
27         first(i+1)
28         multitasking.wait_for_tasks()
29         print('first done')
30
31     for i in range(0, 10):
32         second(i+1)
```

```
30     multitasking.wait_for_tasks()
31     print('second done')
```

Multitasking counter

```
1 import multitasking
2 import time
3
4
5 multitasking.set_max_threads(10)
6 counter = 0
7
8
9 @multitasking.task
10 def count(n):
11     global counter
12     for _ in range(n):
13         counter += 1
14
15
16 if __name__ == "__main__":
17     start = time.time()
18     k = 10
19     n = 1000000
20     for _ in range(k):
21         count(n)
22     multitasking.wait_for_tasks()
23     end = time.time()
24     expected = k * n
25     print(f'done actual: {counter} expected:
{expected}. Missing: {expected-counter}\n')
26
27     print(f'Elapsed time {end-start}')
```

```
1 done actual: 3198547 expected: 10000000. Missing:
6801453
2 Elapsed time 0.5210244655609131
```

Multitasking counter with thread locking

```
1 import multitasking
2 import time
3 import threading
4
5
6 multitasking.set_max_threads(10)
7 counter = 0
8
9
10 locker = threading.Lock()
11
12
13 @multitasking.task
14 def count(n):
15     global counter
16     for _ in range(n):
17         locker.acquire()
18         counter += 1
19         locker.release()
20
21
22 if __name__ == "__main__":
23     start = time.time()
24     k = 10
25     n = 1000000
26     for _ in range(k):
27         count(n)
28     multitasking.wait_for_tasks()
29     end = time.time()
30     expected = k * n
31     print(f'done actual: {counter} expected: {expected}. Missing: {expected-counter}\n')
32 ')
33     print(f'Elapsed time {end-start}')
```

```
1 done actual: 10000000 expected: 10000000. Missing: 0
2 Elapsed time 37.231414556503296
```

Improving Performance - Optimizing code

Problems

- Speed
- Memory usage
- I/O (disk, network, database)

Optimization strategy

The 3 rules of optimization

- Don't do it!
- Don't do it!
- Don't do it yet!

Premature optimization is the root of all evil ~ Donald Knuth

Locate the source of the problem

- I/O is expensive! Database access, file access, GUI update
- If memory is full swapping starts - speed decreases

Optimizing tactics

- Choose the Right Data Structure (Dictionary?, Set?, List?)

- Sorting: Decorate Sort Undecorate (DSU) aka. [Schwartzian Transform](#).
- String Concatenation: avoid extensive concatenation.
- Loops: for, list comprehension: use generators and iterators.
- Delay expanding range, map, filter, etc. iterables.
- Caching results, memoizing.

Read more [performance tips](#)

DSU: Decorate Sort Undecorate

In Perl it is called Schwartzian transform

```

1 animals = ['chicken', 'cow', 'snail', 'elephant']
2 print(sorted(animals))
3 print(sorted(animals, key=len))
4
5 decorated = [(len(w), w) for w in animals]
6 print(decorated)
7
8 decorated.sort()
9 result = [ d[1] for d in decorated]
10 print(result)
11
12 # at once
13 print( [ d[1] for d in sorted( [(len(w), w) for w in
animals] ) ] )

```

```

1 ['chicken', 'cow', 'elephant', 'snail']
2 ['cow', 'snail', 'chicken', 'elephant']
3 [(7, 'chicken'), (3, 'cow'), (5, 'snail'), (8,
'elephant')]
4 ['cow', 'snail', 'chicken', 'elephant']
5 ['cow', 'snail', 'chicken', 'elephant']

```

Profile code

Always profile before starting to optimize!

- [profile](#)

Slow example

This code does some stuff which was deemed to be “too slow” by some client.
The actual content is not that interesting.

```
1 import random
2
3 def f():
4     n = 0
5     for i in range(30):
6         n += random.random()
7     return n
8
9 def g():
10    return random.random() * 30
11
12
13 def main(n):
14     text = get_str(n)
15
16     #print(str)
17     text_sorted = sort(text)
18     return text_sorted
19
20 def sort(s):
21     chars = list(s)
22     for i in reversed(range(len(chars))):
23         a = f()
24         b = g()
25         for j in range(i, len(chars)-1):
26             swap(chars, j)
```

```

28     return ''.join(chars)
29
30 def get_str(n):
31     text = ''
32     for i in range(1, n):
33         text += chr(65 + random.randrange(0, 26))
34     return text
35
36 def swap(lst, loc):
37     if lst[loc] > lst[loc + 1]:
38         lst[loc], lst[loc + 1] = lst[loc + 1],
39         lst[loc]
40 if __name__ == '__main__':
41     print(main(1000))

```

profile slow code

```

1 import slow
2 import profile
3
4 profile.run('slow.main(1000)')

```

```

1          537471 function calls in 3.078 seconds
2
3      Ordered by: standard name
4
5      ncalls  tottime  percall  cumtime  percall
filename:lineno(function)
6          999    0.003    0.000    0.003    0.000 :0(chr)
7             1    0.000    0.000    0.000    0.000 :0(join)
8         1000    0.003    0.000    0.003    0.000 :0(len)
9        31968    0.083    0.000    0.083    0.000
:0(random)
10        1999    0.009    0.000    0.009    0.000
:0(range)
11             1    0.001    0.001    0.001    0.001 :0(setprofile)
12             1    0.000    0.000    3.076    3.076 <string>:1(<module>)
13             0    0.000                  0.000
profile:0(profiler)

```

```
14      1    0.000    0.000    3.078    3.078
profile:0(slow.main(1000))
15    999    0.009    0.000    0.012    0.000
random.py:173(randrange)
16    999    0.005    0.000    0.008    0.000
slow.py:10(g)
17      1    0.000    0.000    3.076    3.076
slow.py:14(main)
18      1    1.410    1.410    3.053    3.053
slow.py:21(sort)
19      1    0.008    0.008    0.023    0.023
slow.py:31(get_str)
20  498501    1.456    0.000    1.456    0.000
slow.py:37	swap)
21    999    0.090    0.000    0.171    0.000
slow.py:4(f)
```

cProfile slow code

```
1 import slow
2 import cProfile
3
4 cProfile.run('slow.main(1000)')
```

```
1      537470 function calls in 0.325 seconds
2
3 Ordered by: standard name
4
5 ncalls  tottime  percall  cumtime  percall
filename:lineno(function)
6      1    0.000    0.000    0.325    0.325
<string>:1(<module>)
7    999    0.002    0.000    0.002    0.000
random.py:173(randrange)
8    999    0.000    0.000    0.000    0.000
slow.py:10(g)
9      1    0.000    0.000    0.325    0.325
slow.py:14(main)
10     1    0.119    0.119    0.322    0.322
slow.py:21(sort)
11     1    0.001    0.001    0.003    0.003
slow.py:31(get_str)
```

```
12 498501      0.189      0.000      0.189      0.000
slow.py:37 (swap)
13    999      0.008      0.000      0.010      0.000
slow.py:4 (f)
14    999      0.000      0.000      0.000      0.000 {chr}
15   1000      0.000      0.000      0.000      0.000 {len}
16     1      0.000      0.000      0.000      0.000 {method
'disable' of '_lsprof.Profiler' o\
17 bjects}
18     1      0.000      0.000      0.000      0.000 {method
'join' of 'str' objects}
19  31968      0.003      0.000      0.003      0.000 {method
'random' of '_random.Random' obje\
20 cts}
21  1999      0.003      0.000      0.003      0.000 {range}
```

Benchmarking

- [benchmark](#)
-

```
1 import timeit
2 from functools import reduce
3 import random
4
5 chars = []
6 for i in range(200):
7     chars.append(chr(65 + random.randrange(0, 26)))
8
9 print(timeit.timeit('string = "".join(chars)',
10                     setup="from __main__ import chars", number=10000))
11
12 print(timeit.timeit('reduce(lambda x, y: x+y, chars)',
13                     setup="from __main__ import chars, reduce",
number=10000))
```

```
1 0.01576369699614588
2 0.15464225399773568
```

Benchmarking subs

```

1 import timeit
2
3 def one_by_one():
4     import random
5     text = ""
6     for i in range(200):
7         text += chr(65 + random.randrange(0, 26))
8     return text
9
10 def at_once():
11     import random
12     chars = []
13     for i in range(200):
14         chars.append(chr(65 + random.randrange(0,
15 26)))
16     text = ''.join(chars)
17     return text
18 print(timeit.timeit('one_by_one',
19     setup="from __main__ import one_by_one",
number=10000))
20
21 print(timeit.timeit('at_once',
22     setup="from __main__ import at_once",
number=10000))

```

```

1 1.5248507579963189
2 1.5566942970035598

```

Levenshtein distance

- [editdistance](#) Levenshtein distance implemented in C
- [python-Levenshtein](#) implemented in C
- [pylev](#)
- [pyxdameraulevenshtein](#)
- [weighted-levenshtein](#)

Generate words

```

1 import sys
2 import random
3 import string
4
5 # TODO: set min, max word length
6 # TODO: set filename
7 # TODO: set character types
8 # TODO: allow spaces?
9
10 def main():
11     filename = "words.txt"
12     min_len = 6
13     max_len = 6
14
15     if len(sys.argv) != 2:
16         exit(f"Usage: {sys.argv[0]} WORD_COUNT")
17     count = int(sys.argv[1])
18     with open(filename, 'w') as fh:
19         for _ in range(count):
20             word = ''
21             length = random.randrange(min_len,
max_len+1)
22             for _ in range(length):
23                 word +=
random.choice(string.ascii_lowercase)
24             fh.write(word + "\n")
25
26 main()

```

Levenshtein - pylev

```

1 import sys
2 import pylev
3
4 def main():
5     if len(sys.argv) != 2:
6         exit(f"Usage: {sys.argv[0]} filename")
7     filename = sys.argv[1]
8     outfile = 'out.txt'
9
10    rows = []
11    with open(filename) as fh:

```

```
12     for row in fh:
13         rows.append(row.rstrip("\n"))
14     with open(outfile, 'w') as fh:
15         for a in rows:
16             for b in rows:
17                 dist = pylev.levenshtein(a, b)
18                 fh.write(f"{a},{b},{dist}\n")
19
20 main()
```

Levenshtein - editdistance

```
1 import sys
2 import editdistance
3
4 def main():
5     if len(sys.argv) != 2:
6         exit(f"Usage: {sys.argv[0]} filename")
7     filename = sys.argv[1]
8     outfile = 'out.txt'
9
10    rows = []
11    with open(filename) as fh:
12        for row in fh:
13            rows.append(row.rstrip("\n"))
14    with open(outfile, 'w') as fh:
15        for a in rows:
16            for b in rows:
17                dist = editdistance.eval(a, b)
18                fh.write(f"{a},{b},{dist}\n")
19
20 main()
```

Editdistance benchmark

- [editdistance](#)

A Tool to Generate text files

```
 1 import sys
 2 import string
 3 import random
 4 import argparse
 5 import os
 6
 7 # Generate n file of size S with random letters
 8
 9 def get_args():
10     parser = argparse.ArgumentParser()
11     parser.add_argument('--dir',
help="Directory where to create the fil\
12 es", default="."))
13     parser.add_argument('--files', type=int,
help="Number of files to create", defau\
14 lt=1)
15     parser.add_argument('--size', type=int,
help="Size of files", defau\
16 lt=10)
17     args = parser.parse_args()
18     return args
19
20 def main():
21     args = get_args()
22     chars = list(string.ascii_lowercase) + [' '] * 5 +
['\n']
23
24     for ix in range(args.files):
25         all_chars = []
26         for _ in range(args.size):
27             all_chars.extend(random.sample(chars, 1))
28             #print(len(all_chars))
29
30             #print(all_chars)
31             filename = os.path.join(args.dir, str(ix) +
'.txt')
32             with open(filename, 'w') as fh:
33                 fh.write(''.join(all_chars))
34
35
36 def old_main():
37     if len(sys.argv) < 2:
38         exit(f"Usage: {sys.argv[0]} {NUMBER_OF_ROWS}")
39
```

```

40     row_count = int(sys.argv[1])
41     min_width = 30
42     max_width = 50
43     filename = 'data.log'
44
45     chars = list(string.ascii_lowercase) + [' '] * 5
46     all_chars = chars * max_width
47
48     with open(filename, 'w') as fh:
49         for i in range(row_count):
50             width = random.randrange(min_width,
51                                       max_width+1)
51             row = ''.join(random.sample(all_chars,
52                                         width))
52             fh.write(row + "\n")
53
54 main()

```

Count characters

```

1 # changes chars and counter
2 def add_char(chars, counter, ch, cnt=1):
3     for ix in range(len(chars)):
4         if chars[ix] == ch:
5             counter[ix] += cnt
6             break
7     else:
8         chars.append(ch)
9         counter.append(cnt)
10
11
12 def count_in_file(filename):
13     #print(filename)
14     chars = []
15     counter = []
16     with open(filename) as fh:
17         for row in fh:
18             for ch in row:
19                 #print(ch)
20                 if ch == ' ':
21                     continue
22                 if ch == '\n':
23                     continue

```

```

24             add_char(chars, counter, ch)
25
26     #print(chars)
27     #print(counter)
28     return chars, counter
29
30 def merge(chars1, counter1, chars2, counter2):
31     chars = []
32     counter = []
33     for ix in range(len(chars1)):
34         add_char(chars, counter, chars1[ix],
35                  cnt=counter1[ix])
36         for ix in range(len(chars2)):
37             add_char(chars, counter, chars2[ix],
38                      cnt=counter2[ix])
39     return chars, counter
40
41
42
43
44
45 def print_results(chars, counter):
46     print("Results")
47     for ix in range(len(chars)):
48         print("{} {}".format(chars[ix], counter[ix]))
49
50
51
52
53
54
55
56 if __name__ == '__main__':
57     import sys
58     chars, counter = count_in(sys.argv[1:])
59     print_results(chars, counter)

```

```

1 import count_characters as count
2 import cProfile
3 import sys
4

```

```
5 cProfile.run('chars, counter =
count.count_in(sys.argv[1:])')
```

Memory leak

```
1 import random
2
3 def alloc():
4     a = {
5         'data': str(random.random()) + "a" * 10000000,
6     }
7     b = {
8         'data': str(random.random()) + "b" * 10000000,
9     }
10    a['other'] = b
11    b['other'] = a
```

```
1 import sys
2 from mymem import alloc
3
4 if len(sys.argv) < 2:
5     exit(f"Usage: {sys.argv[0]} N")
6
7 count = int(sys.argv[1])
8
9 for _ in range(count):
10    alloc()
11 input("End the script")
```

Garbage collection

- `gc`
-

```
1 import sys
2 from mymem import alloc
3 import gc
4
5 if len(sys.argv) < 2:
6     exit(f"Usage: {sys.argv[0]} N")
```

```
7
8 count = int(sys.argv[1])
9
10 for _ in range(count):
11     alloc()
12 input("Run gc")
13
14 gc.collect()
15 input("End the script")
```

Weak reference

- weakref

```
1 import random
2 import weakref
3
4 def alloc():
5     a = {
6         'data': str(random.random()) + "a" * 10000000,
7     }
8     b = {
9         'data': str(random.random()) + "b" * 10000000,
10    }
11    #a['other'] = weakref.WeakKeyDictionary(b)
12    z = weakref.ref(b)
13    #a['other'] =
14    #weakref.ref(a['other'])
15    #b['other'] = a
16    #weakref.ref(b['other'])
```

```
1 import sys
2 from weakmymem import alloc
3
4 if len(sys.argv) < 2:
5     exit(f"Usage: {sys.argv[0]} N")
6
7 count = int(sys.argv[1])
8
9 for _ in range(count):
```

```
10     alloc()  
11 input("End the script")
```

Exercise: benchmark list-comprehension, map, for

- Create several functions that accept a list of numbers from 1 to 1000 and calculate their square:
- A function with a `for`-loop.
- A function that uses `map`.
- A function that uses list-comprehension.
- Feel free to have any other calucaltion and measure that.
- Send me the code and the results!

Exercise: Benchmark Levenshtein

- Take the implementation of the Levenshtein distance calculations and check which one is faster.

Exercise: sort files

Write a script that given a path to a directory will print the files sorted by date.

If you don't have one large folder, then use `os.walk` to get the path to the files of a whole directory tree.

- Write a simple solution.
- Profile.
- Use [DSU](#).

Exercise: compare split words:

We have three ways of splitting a string into words. Using `split`, using `re.split` and by going over it character-by-character.
Which one is the fastest?

```
1 import sys
2 import re
3
4 def split_to_words_by_regex(text):
5     return re.split(' ', text)
6
7 def split_to_words_by_split(text):
8     return text.split()
9
10 def split_to_words_by_chars(text):
11     words = []
12     word = ''
13     for ch in text:
14         if ch == ' ':
15             words.append(word)
16             word = ''
17         else:
18             word += ch
19     if word:
20         words.append(word)
21     return words
22
23
24 if __name__ == '__main__':
25     if len(sys.argv) < 2:
26         exit(f"Usage: {sys.argv[0]} FILENAME")
27
28     filename = sys.argv[1]
29     with open(filename) as fh:
30         text = fh.read()
31     res1 = split_to_words_by_split(text)
32     res2 = split_to_words_by_chars(text)
33     res3 = split_to_words_by_regex(text)
34     #print(res1)
35     #print(res2)
36     assert res1 == res2
37     assert res1 == res3
```

Exercise: count words

Given a file count how many times each word appears.

Have two implementations. One using two list and one using a dictionary.

Profile the code and benchmark the two solutions.

See `examples/lists/count_words_two_lists.py` and
`examples/dictionary/count_words.py`

GUI with Python/Tk

Sample Tk app

```
 1 import tkinter as tk
 2 from tkinter import ttk, messagebox, filedialog
 3 import os
 4
 5
 6 def scary_action():
 7     messagebox.showerror(title="Scary",
message="Deleting hard disk. Please wait...")
 8
 9
10 def run_code():
11     text = ""
12     text += "Name: {}\n".format(name.get())
13     text += "Password: {}\n".format(password.get())
14     text += "Animal: {}\n".format(animal.get())
15     text += "Country: {}\n".format(country.get())
16     text += "Colors: "
17     for ix in range(len(colors)):
18         if colors[ix].get():
19             text += color_names[ix] + " "
20     text += "\n"
21
22     selected = list_box.curselection() # returns a
tuple
23     text += "Animals: "
24     text += ', '.join([list_box.get(idx) for idx in
selected])
25     text += "\n"
26
27     text += "Filename:
{}{}\n".format(os.path.basename(filename_entry.get()))
28
29     resp = messagebox.askquestion(title="Running
with", message=f"Shall I start runn\
```

```
30 ing with the following values?\n\n{text})")
31     if resp == 'yes':
32         output_window['state'] = 'normal' # allow
editing of the Text widget
33         output_window.insert('end', f"{text}\n-----\n")
34         output_window['state'] = 'disabled' # disable
editing
35         output_window.see('end') # scroll to the end
as we make progress
36         app.update()
37
38
39 def close_app():
40     app.destroy()
41
42
43 app = tk.Tk()
44 app.title('Simple App')
45
46 menubar = tk.Menu(app)
47 app.config(menu=menubar)
48
49 menu1 = tk.Menu(menubar, tearoff=0)
50 menubar.add_cascade(label="File", underline=0,
menu=menu1)
51 menu1.add_separator()
52 menu1.add_command(label="Exit", underline=1,
command=close_app)
53
54 top_frame = tk.Frame(app)
55 top_frame.pack(side="top")
56 pw_frame = tk.Frame(app)
57 pw_frame.pack(side="top")
58
59 # Simple Label widget:
60 name_title = tk.Label(top_frame, text=" Name:", width=10, anchor="w")
61 name_title.pack({"side": "left"})
62
63 # Simple Entry widget:
64 name = tk.Entry(top_frame)
65 name.pack({"side": "left"})
66 # name.insert(0, "Your name")
67
```

```

68 # Simple Label widget:
69 password_title = tk.Label(pw_frame, text=" Password:",
70 width=10, anchor="w")
71 password_title.pack({"side": "left"})
72
72 # In order to hide the text as it is typed (e.g. for
73 # set the "show" parameter:
74 password = tk.Entry(pw_frame)
75 password["show"] = "*"
76 password.pack({"side": "left"})
77
78 radios = tk.Frame(app)
79 radios.pack()
80 animal = tk.StringVar()
81 animal.set("Red")
82 my_radio = []
83 animals = ["Cow", "Mouse", "Dog", "Car", "Snake"]
84 for animal_name in animals:
85     radio = tk.Radiobutton(radios, text=animal_name,
variable=animal, value=animal_n\
ame)
86     radio.pack({"side": "left"})
87     my_radio.append(radio)
88
89
90
91 checkboxes = tk.Frame(app)
92 checkboxes.pack()
93 colors = []
94 my_checkbox = []
95 color_names = ["Red", "Blue", "Green"]
96 for color_name in color_names:
97     color_var = tk.BooleanVar()
98     colors.append(color_var)
99     checkbox = tk.Checkbutton(checkboxes,
text=color_name, variable=color_var)
100    checkbox.pack({"side": "left"})
101    my_checkbox.append(checkbox)
102
103 countries = ["Japan", "Korea", "Vietnam", "China"]
104
105 def country_change(event):
106     pass
107     #selection = country.current()
108     #print(selection)

```

```
109     #print(countries[selection])
110
111 def country_clicked():
112     pass
113     #print(country.get())
114
115 country = ttk.Combobox(app, values=countries)
116 country.pack()
117 country.bind("<<ComboboxSelected>>", country_change)
118
119
120
121
122 list_box = tk.Listbox(app, selectmode=tk.MULTIPLE,
height=4)
123 animal_names = ['Snake', 'Mouse', 'Elephant', 'Dog',
'Cat', 'Zebra', 'Camel', 'Spide\
124 r']
125 for val in animal_names:
126     list_box.insert(tk.END, val)
127 list_box.pack()
128
129 def open_filename_selector():
130     file_path = filedialog.askopenfilename(filetypes=\
((("Any file", "*"),))
131     filename_entry.delete(0, tk.END)
132     filename_entry.insert(0, file_path)
133
134
135 filename_frame = tk.Frame(app)
136 filename_frame.pack()
137 filename_label = tk.Label(filename_frame,
text="Filename:", width=10)
138 filename_label.pack({"side": "left"})
139 filename_entry = tk.Entry(filename_frame, width=60)
140 filename_entry.pack({"side": "left"})
141 filename_button = tk.Button(filename_frame,
text="Select file", command=open_filenam\
142 e_selector)
143 filename_button.pack({"side": "left"})
144
145 output_frame = tk.Frame(app)
146 output_frame.pack()
147 output_window = tk.Text(output_frame,
state='disabled')
```

```
148 output_window.pack()
149
150
151 buttons = tk.Frame(app)
152 buttons.pack()
153
154 scary_button = tk.Button(buttons, text="Don't click
here!", fg="red", command=scary_\
155 action)
156 scary_button.pack({"side": "left"})
157
158 action_button = tk.Button(buttons, text="Run",
command=run_code)
159 action_button.pack()
160
161 app.mainloop()
162
163 # TODO: key binding?
164 # TODO: Option Menu
165 # TODO: Scale
166 # TODO: Progressbar (after the deleting hard disk pop-
up)
167 # TODO: Frame (with border?)
```

GUI Toolkits

When creating an application there are several ways to interact with the user. You can accept command line parameters. You can interact on the Standard Output / Standard Input running in a Unix Shell or in the Command Prompt of Windows.

Many people, especially those who are using MS Windows, will frown upon both of those. They expect a Graphical User Interface (GUI) or maybe a web interface via their browser. In this chapter we are going to look at the possibility to create a desktop GUI.

There are plenty of ways to create a GUI in Python. The major ones were listed here, but there are many more. See the

additional links.

In this chapter we are going to use the Tk Toolkit.

- [Tk](#)
- [GTK](#)
- [Qt](#)
- [wxWidgets](#)
- [GUI FAQ](#)
- [GUI Programming](#)

Installation

Tk in Python is actually a wrapper around the implementation in Tcl.

Tcl/Tk usually comes installed with Python. All we need is basically the Tkinter Python module.

In some Python installations (e.g. Anaconda), Tkinter is already installed. In other cases you might need to install it yourself. For example on Ubuntu you can use `apt` to install it.

¹ `sudo apt-get install python3-tk`

Python Tk Documentation

The documentation of Tk in Python does not cover all the aspects of Tk. If you are creating a complex GUI application you might need to dig in the documentation written for Tcl/Tk.

- [Tk](#)
- The [Tk Command of Tcl 8.6](#)
- [Python GUI Geeks for Geeks](#)

In the Unix world where Tk came from the various parts of a GUI application are called widgets. In the MS Windows world they are usually called controls. There are several commonly used Widgets. For example, Label, Button, Entry, Radiobutton, Checkbox.

First we are going to see small examples with each one of these Widgets. Then we'll see how to combine them.

Python Tk Button

- [Button](#)

```
1 import tkinter as tk
2
3 app = tk.Tk()
4 app.title('Single Button')
5
6 button = tk.Button(app, text='Close', width=25,
7 command=app.destroy)
8 button.pack()
9 app.mainloop()
```

Python Tk Button with action

```
1 import tkinter as tk
2
3 def run_action():
4     print("clicked")
5
6 app = tk.Tk()
7 app.title('Single Button')
8
9 action_button = tk.Button(app, text='Action',
width=25, command=run_action)
10 action_button.pack()
11 #action_button.pack(side="left")
12
13 exit_button = tk.Button(app, text='Close', width=25,
command=app.destroy)
14 exit_button.pack()
15
16 app.mainloop()
```

Python Tk Label

- Label
-

```
1 import tkinter as tk
2
3 app = tk.Tk()
4 #app.title('Simple Label')
5
6 label = tk.Label(app, text='Some fixed text')
7 label.pack()
8
9 app.mainloop()
```

Python Tk Label - font size and color

```
1 import tkinter as tk
2
3 app = tk.Tk()
4 app.title('Label with font')
5
6 label = tk.Label(app, text='Some text with larger
letters')
7 label.pack()
8 label.config(font=("Courier", 44))
9 label.config(fg="#0000FF")
10 label.config(bg="yellow")
11
12 app.mainloop()
```

Python Tk Keybinding

```
1 import tkinter as tk
2
3 app = tk.Tk()
4 app.title('Key binding')
5
6 label = tk.Label(app, text='Use the keyboard: (a, Ctrl-
b, Alt-c, F1, Alt-F4)')
7 label.config(font=("Courier", 44))
8 label.pack()
9
10 def pressed_a(event):
11     print("pressed a")
12
13 def pressed_control_b(event):
14     print("pressed Ctrl-b")
15
16 def pressed_alt_c(event):
17     print("pressed Alt-c")
18
19 def pressed_f1(event):
20     print("pressed F1")
21
22 app.bind("<a>", pressed_a)
23 app.bind("<Control-b>", pressed_control_b)
24 app.bind("<Alt-c>", pressed_alt_c)
25 app.bind("<F1>", pressed_f1)
```

```
26  
27  
28 app.mainloop()
```

Python Tk Entry (one-line text entry)

- [Entry](#)

```
1 import tkinter as tk  
2  
3 app = tk.Tk()  
4 app.title('Text Entry')  
5  
6 entry = tk.Entry(app)  
7 entry.pack()  
8  
9 def clicked():  
10     print(entry.get())  
11  
12 button = tk.Button(app, text='Show', width=25,  
command=clicked)  
13 button.pack()  
14  
15 exit_button = tk.Button(app, text='Close', width=25,  
command=app.destroy)  
16 exit_button.pack()  
17  
18 app.mainloop()
```

Python Tk Entry for passwords and other secrets (hidden text)

```
1 import tkinter as tk  
2  
3 app = tk.Tk()  
4 app.title('Text Entry')  
5  
6 entry = tk.Entry(app)  
7 entry['show'] = '*'  
8 entry.pack()
```

```
9
10 def clicked():
11     print(entry.get())
12
13 button = tk.Button(app, text='Show', width=25,
14 command=clicked)
14 button.pack()
15
16 exit_button = tk.Button(app, text='Close', width=25,
17 command=app.destroy)
17 exit_button.pack()
18
19 app.mainloop()
```

Python Tk Checkbox

```
1 import tkinter as tk
2
3 app = tk.Tk()
4 app.title('Checkbox')
5
6 var1 = tk.BooleanVar()
7 cb1 = tk.Checkbutton(app, text='male', variable=var1)
8 cb1.pack()
9
10 var2 = tk.BooleanVar()
11 cb2 = tk.Checkbutton(app, text='female',
12 variable=var2)
12 cb2.pack()
13
14 def clicked():
15     print(var1.get())
16     print(var2.get())
17
18 button = tk.Button(app, text='Show', width=25,
19 command=clicked)
19 button.pack()
20
21 exit_button = tk.Button(app, text='Close', width=25,
22 command=app.destroy)
22 exit_button.pack()
23
24 app.mainloop()
```

-
- [Variables](#)

Python Tk Radiobutton

```
1 import tkinter as tk
2
3 def run_action():
4     print("clicked")
5     print(count.get())
6
7 app = tk.Tk()
8 app.title('Radio button')
9
10 count = tk.IntVar()
11 #count.set(2)
12
13 my_radios = []
14 values = [(1, "One"), (2, "Two"), (3, "Three")]
15 for ix in range(len(values)):
16     my_radios.append(tk.Radiobutton(app,
17         text=values[ix][1], variable=count, value=v\
18         alues[ix][0]))
19     my_radios[ix].pack()
20
21 action_button = tk.Button(app, text='Action',
22     width=25, command=run_action)
23 action_button.pack()
24
25 exit_button = tk.Button(app, text='Close', width=25,
26     command=app.destroy)
27 exit_button.pack()
28
29 app.mainloop()
```

Python Tk Listbox

```
1 import tkinter as tk
2
3 app = tk.Tk()
4 app.title('List box')
```

```

5
6
7 def clicked():
8     print("clicked")
9     selected = box.curselection()    # returns a tuple
10    if selected:
11        first = selected[0]
12        color = box.get(first)
13        print(color)
14
15 box = tk.Listbox(app)
16 values = ['Red', 'Green', 'Blue', 'Purple']
17 for val in values:
18     box.insert(tk.END, val)
19 box.pack()
20
21 button = tk.Button(app, text='Show', width=25,
command=clicked)
22 button.pack()
23
24 exit_button = tk.Button(app, text='Close', width=25,
command=app.destroy)
25 exit_button.pack()
26
27 app.mainloop()

```

Python Tk Listbox Multiple

```

1 import tkinter as tk
2
3 app = tk.Tk()
4 app.title('List box')
5
6
7 def clicked():
8     print("clicked")
9     selected = box.curselection()    # returns a tuple
10    for idx in selected:
11        print(box.get(idx))
12
13 box = tk.Listbox(app, selectmode=tk.MULTIPLE,
height=4)
14 values = ['Red', 'Green', 'Blue', 'Purple', 'Yellow',

```

```
'Orange', 'Black', 'White']
15 for val in values:
16     box.insert(tk.END, val)
17 box.pack()
18
19 button = tk.Button(app, text='Show', width=25,
command=clicked)
20 button.pack()
21
22 exit_button = tk.Button(app, text='Close', width=25,
command=app.destroy)
23 exit_button.pack()
24
25 app.mainloop()
```

Python Tk Menubar

- [Menubar](#)
 - [Menu](#)
-
- `underline` sets the hot-key.
 - `tearoff=` (the default) allows floating menu by clicking on the dashed line.
 - enable/disable menu items.
 - Set actions via `command` on the menu items.

```
1 import tkinter as tk
2
3 app = tk.Tk()
4 app.title('Menu')
5
6 def run_new():
7     print("new")
8
9 def run_exit():
10    print("exit")
11    app.destroy()
12
13 def enable_languages():
```

```

14     menu2.entryconfig("Klingon", state="normal")
15 def disable_languages():
16     menu2.entryconfig("Klingon", state="disabled")
17
18 def set_language(lang):
19     print(lang)
20
21
22 menubar = tk.Menu(app)
23
24 menu1 = tk.Menu(menubar, tearoff=0)
25 menu1.add_command(label="New", command=run_new)
26 menu1.add_command(label="Enable language",
command=enable_languages)
27 menu1.add_command(label="Disable language",
command=disable_languages)
28 menu1.add_separator()
29 menu1.add_command(label="Exit", underline=1,
command=run_exit)
30
31 menubar.add_cascade(label="File", underline=0,
menu=menu1)
32
33 menu2 = tk.Menu(menubar, tearoff=1)
34 menu2.add_command(label="English")
35 menu2.add_command(label="Hebrew")
36 menu2.add_command(label="Spanish")
37 menu2.add_command(label="Klingon", state="disabled",
command=lambda : set_language('\
38 Klingon'))
39 menu2.add_command(label="Hungarian")
40
41 menubar.add_cascade(label="Language", menu=menu2)
42
43 app.config(menu=menubar)
44
45 app.mainloop()

```

Python Tk Text

```

1 import tkinter as tk
2
3 app = tk.Tk()

```

```
4 app.title('Text Editor')
5
6 text = tk.Text(app)
7 text.pack({"side": "bottom"})
8
9 app.mainloop()
```

- `text.delete(1.0, tk.END)`
- `text.insert('end', content)`
- `content = text.get(1.0, tk.END)`
- [tk text](#)

Python Tk Dialogs

- [Dialogs](#)
- Filedialogs
- Message boxes

Python Tk Filedialog

- [file dialogs](#)
- [dialog](#)
- `askopenfilename` - returns path to file
- `asksaveasfilename` - returns path to file
- `askopenfile` - returns filehandle opened for reading
- `asksaveasfile` - retutns filehandle opened for writing
- Allow the listing of file-extension filters.

```
1 import tkinter as tk
2 from tkinter import filedialog
3
```

```
4 input_file_path = None
5 output_file_path = None
6
7 def run_process():
8     print("Parameters:")
9     print(f"in: {input_file_path}")
10    print(f"out: {output_file_path}")
11
12 def close_app():
13     print("Bye")
14     app.destroy()
15
16 def select_input_file():
17     global input_file_path
18     input_file_path =
filedialog.askopenfilename(filetypes=(( "Excel files",
"*.xlsx"\n),
("CSV files", "*.csv"), ("Any file", "*")))
19     print(input_file_path)
20
21
22 def select_output_file():
23     global output_file_path
24     output_file_path =
filedialog.asksaveasfilename(filetypes=(( "Excel files",
"*.xl\n",
"*.xlsx"\n),
("CSV files", "*.csv"), ("Any file", "*")))
25     print(output_file_path)
26
27
28 app = tk.Tk()
29 app.title('Convert file')
30
31 input_button = tk.Button(app, text='Select input
file', command=select_input_file)
32 input_button.pack()
33
34 output_button = tk.Button(app, text='Select output
file', command=select_output_file)
35 output_button.pack()
36
37 process_button = tk.Button(app, text='Process',
width=25, command=run_process)
38 process_button.pack()
39
40 exit_button = tk.Button(app, text='Close', width=25,
```

```
command=close_app)
41 exit_button.pack()
42
43 app.mainloop()
```

Python Tk messagebox

```
1 import tkinter as tk
2 from tkinter import messagebox
3
4 app = tk.Tk()
5 app.title('Menu')
6
7 def run_show_info():
8     messagebox.showinfo(title = "Title", message =
9 "Show info text")
10
10 def run_show_warning():
11     messagebox.showwarning(title = "Title", message =
12 "Show warning text")
12
13 def run_show_error():
14     messagebox.showerror(title = "Title", message =
15 "Show error text")
15
16 def run_ask_question():
17     resp = messagebox.askquestion(title = "Title",
18 message = "Can I ask you a questi\
18 on?")
19     print(resp) # "yes" / "no" (default "no")
20
21 def run_ask_okcancel():
22     resp = messagebox.askokcancel(title = "Title",
23 message = "Shall I do it?")
23     print(resp) # True / False (default = False)
24
25 def run_ask_retrycancel():
26     resp = messagebox.askretrycancel(title = "Title",
27 message = "Shall retry it?")
27     print(resp) # True / False (default = False)
28
29 def run_ask_yesno():
30     resp = messagebox.askyesno(title = "Title",
```

```
message = "Yes or No?")
31     print(resp)    # True / False (default = False)
32
33 def run_ask_ynocancel():
34     resp = messagebox.askynocancel(title = "Title",
35     message = "Yes, No, or Cancel?\\" )
36     print(resp)    # True / False / None (default =
37     None)
38
39 def run_exit():
40     app.destroy()
41
42 menubar = tk.Menu(app)
43
44 menu1 = tk.Menu(menubar, tearoff=0)
45 menu1.add_command(label="Info",      underline=0,
46 command=run_show_info)
46 menu1.add_command(label="Warning",   underline=0,
47 command=run_show_warning)
47 menu1.add_command(label="Error",    underline=0,
48 command=run_show_error)
48 menu1.add_separator()
49 menu1.add_command(label="Exit",     underline=1,
50 command=run_exit)
51
51 menubar.add_cascade(label="Show",   underline=0,
52 menu=menu1)
52
53 menu2 = tk.Menu(menubar, tearoff=0)
54 menu2.add_command(label="Question",
55 underline=0, command=run_ask_question)
55 menu2.add_command(label="OK Cancel",
56 underline=0, command=run_ask_okcancel)
56 menu2.add_command(label="Retry Cancel",
57 underline=0, command=run_ask_retrycancel)
57 el)
58 menu2.add_command(label="Yes or No",
59 underline=0, command=run_ask_yesno)
59 menu2.add_command(label="Yes, No, or Cancel",
60 underline=5, command=run_ask_yesnocancel)
60 el)
61
62 menubar.add_cascade(label="Ask",   underline=0,
```

```
menu=menu2)
63
64 app.config(menu=menubar)
65
66 app.mainloop()
```

- [Tk messagebox](#)

Python Tk Combobox

```
1 import tkinter as tk
2 from tkinter import ttk
3
4 countries = ["Japan", "Korea", "Vietnam", "China"]
5
6 app = tk.Tk()
7 app.title('Combo box')
8
9
10 def change(event):
11     # VirtualEvent
12     print("change")
13     selection = country.current()
14     print(selection)
15     print(countries[selection])
16
17 def clicked():
18     print("clicked")
19     print(country.get())
20
21 country = ttk.Combobox(app, values=countries)
22 country.pack()
23 country.bind("<<ComboboxSelected>>", change)
24
25 button = tk.Button(app, text='Run', width=25,
26 command=clicked)
27 button.pack()
28
29 app.mainloop()
```

Python Tk OptionMenu

```
1 import tkinter as tk
2
3 def run_action():
4     color = color_var.get()
5     print(color)
6
7     size = size_var.get()
8     print(size)
9
10 app = tk.Tk()
11 app.title('Option Menu')
12
13 color_var = tk.StringVar(app)
14 color_selector = tk.OptionMenu(app, color_var, "Red",
15 "Green", "Blue")
15 color_selector.pack()
16
17 sizes = ("Small", "Medium", "Large")
18 size_var = tk.StringVar(app)
19 size_selector = tk.OptionMenu(app, size_var, *sizes)
20 size_selector.pack()
21
22 action_button = tk.Button(app, text='Action',
23 width=25, command=run_action)
23 action_button.pack()
24
25 app.mainloop()
```

Python Tk Scale

```
1 import tkinter as tk
2
3 def run_action():
4     h = scale_h.get()
5     print(h)
6
7     v = scale_v.get()
8     print(v)
9
10 app = tk.Tk()
```

```
11 app.title('Scale')
12
13 scale_h = tk.Scale(app, from_=0, to=42,
orient=tk.HORIZONTAL)
14 scale_h.pack()
15
16 scale_v = tk.Scale(app, from_=1, to=100,
orient=tk.VERTICAL)
17 scale_v.pack()
18 scale_v.set(23)
19
20 action_button = tk.Button(app, text='Action',
width=25, command=run_action)
21 action_button.pack()
22
23 app.mainloop()
```

Python Tk Progressbar

```
1 import tkinter as tk
2 from tkinter import ttk
3
4 app = tk.Tk()
5 app.title('Single Button')
6
7 progressbar = ttk.Progressbar(app)
8 progressbar.pack()
9
10 def stop():
11     progressbar.stop()
12
13 def start():
14     app.after(10000, stop)
15     progressbar.start(100)
16
17
18 button = tk.Button(app, text='Start', width=25,
command=start)
19 button.pack()
20
21 exit_button = tk.Button(app, text='Close', width=25,
command=app.destroy)
22 exit_button.pack()
```

```
23  
24 app.mainloop()
```

Python Tk Frame

```
1 import tkinter as tk  
2  
3 def close():  
4     app.destroy()  
5  
6 def clicked(val):  
7     entry.insert(tk.END, val)  
8  
9 app = tk.Tk()  
10 app.title('Frame')  
11  
12 entry = tk.Entry(app)  
13 entry.pack()  
14  
15 frames = {}  
16 frames[1] = tk.Frame(app)  
17 frames[1].pack(side="top")  
18 frames[2] = tk.Frame(app)  
19 frames[2].pack(side="top")  
20 frames[3] = tk.Frame(app)  
21 frames[3].pack(side="top")  
22  
23 btn = {}  
24  
25 btn["a"] = tk.Button(frames[1], text="a", width=25,  
command=lambda : clicked("a"))  
26 btn["a"].pack(side="left")  
27  
28 btn["b"] = tk.Button(frames[1], text="b", width=25,  
command=lambda : clicked("b"))  
29 btn["b"].pack(side="left")  
30  
31 btn["c"] = tk.Button(frames[2], text="c", width=25,  
command=lambda : clicked("c"))  
32 btn["c"].pack(side="left")  
33  
34 btn["d"] = tk.Button(frames[2], text="d", width=25,  
command=lambda : clicked("d"))
```

```
35 btn["d"].pack(side="left")
36
37 close_btn = tk.Button(frames[3], text='Close',
width=25, command=close)
38 close_btn.pack(side="right", expand=0)
39
40 app.mainloop()
```

- width
- side: left, right, top, bottom

Not so Simple Tk app with class

```
1 from tkinter import Tk, Frame, BOTH
2
3
4 class Example(Frame):
5     def __init__(self, parent):
6         Frame.__init__(self, parent,
background="white")
7         self.parent = parent
8         self.initUI()
9
10    def initUI(self):
11        self.parent.title("Simple")
12        self.pack(fill=BOTH, expand=1)
13
14
15 def main():
16     root = Tk()
17     root.geometry("250x150+300+300")
18     app = Example(parent=root)
19
20     # move the window to the front (needed on Mac
only?)
21     root.lift()
22     root.call('wm', 'attributes', '.', '-topmost',
True)
23     root.after_idle(root.call, 'wm', 'attributes',
'.', '-topmost', False)
24
25     root.mainloop()
```

```
27 main()
```

Tk: Hello World

```
1 import tkinter as tk
2
3 class Example(tk.Frame):
4     def __init__(self, parent=None):
5         super().__init__(parent)
6         self.pack()
7         self.createWidgets()
8
9     def createWidgets(self):
10        # Simple Label widget:
11        self.name_title = tk.Label(self, text="Hello
12        World!")
13        self.name_title.pack({"side": "left"})
14
15    def main():
16        root = tk.Tk()
17        app = Example(parent=root)
18        app.mainloop()
19
20 main()
```

Tk: Quit button

```
1 import tkinter as tk
2
3 class Example(tk.Frame):
4     def __init__(self, parent=None):
5         super().__init__(parent)
6         self.pack()
7         self.createWidgets()
8
9     def createWidgets(self):
10        self.QUIT = tk.Button(self)
11        self.QUIT["text"] = "QUIT"
12        self.QUIT["fg"]   = "red"
13        self.QUIT["command"] = self.quit
```

```

14         self.QUIT.pack({"side": "left"})
15
16 def main():
17     root = tk.Tk()
18     app = Example(parent=root)
19
20     app.mainloop()
21
22 main()

```

Tk: File selector

```

1 import tkinter as tk
2 from tkinter import filedialog
3
4 class Example(tk.Frame):
5     def __init__(self, parent=None):
6         super().__init__(parent)
7         self.pack()
8         self.createWidgets()
9
10    def get_file(self):
11        file_path = filedialog.askopenfilename()
12        print(file_path)
13        self.filename.delete(0, tk.END)
14        self.filename.insert(0, file_path)
15
16    def run_process(self):
17        print("Running a process on file
{}".format(self.filename.get()))
18
19    def createWidgets(self):
20        self.QUIT = tk.Button(self)
21        self.QUIT["text"] = "QUIT"
22        self.QUIT["fg"] = "red"
23        self.QUIT["command"] = self.quit
24        self.QUIT.pack({"side": "right"})
25
26        # Simple Label widget:
27        self.filename_title = tk.Label(self,
28                                       text="Fileame:")
29        self.filename_title.pack({"side": "left"})

```

```

30     # Simple Entry widget:
31     self.filename = tk.Entry(self, width=120)
32     self.filename.pack({"side": "left"})
33     self.filename.delete(0, tk.END)
34
35     self.selector = tk.Button(self)
36     self.selector["text"] = "Select",
37     self.selector["command"] = self.get_file
38     self.selector.pack({"side": "left"})
39
40     self.process = tk.Button(self)
41     self.process["text"] = "Process",
42     self.process["command"] = self.run_process
43     self.process.pack({"side": "left"})
44
45
46 def main():
47     root = tk.Tk()
48     app = Example(parent=root)
49
50     root.lift()
51     root.call('wm', 'attributes', '.', '-topmost',
52               True)
52     root.after_idle(root.call, 'wm', 'attributes',
53                     '.', '-topmost', False)
53
54     app.mainloop()
55
56 main()

```

Tk: Checkbox

```

1 import tkinter as tk
2
3 class Example(tk.Frame):
4     def __init__(self, parent=None):
5         super().__init__(parent)
6         self.pack()
7         self.createWidgets()
8
9     def show_values(self):
10        print("show values")
11        for v in self.vars:

```

```
12         print(v.get())
13
14     def createWidgets(self):
15         self.QUIT = tk.Button(self)
16         self.QUIT["text"] = "QUIT"
17         self.QUIT["fg"] = "red"
18         self.QUIT["command"] = self.quit
19         self.QUIT.pack({"side": "left"})
20
21
22         self.vars = []
23         self.cbs = []
24         self.vars.append(tk.IntVar())
25         cb = tk.Checkbutton(text="Blue",
variable=self.vars[-1])
26         cb.pack({"side": "left"})
27         self.cbs.append(cb)
28
29         self.vars.append(tk.IntVar())
30         cb = tk.Checkbutton(text="Yellow",
variable=self.vars[-1])
31         cb.pack({"side": "left"})
32         self.cbs.append(cb)
33
34         self.show = tk.Button(self)
35         self.show["text"] = "Show",
36         self.show["command"] = self.show_values
37         self.show.pack({"side": "left"})
38
39     def main():
40         root = tk.Tk()
41         app = Example(parent=root)
42
43         root.lift()
44         root.call('wm', 'attributes', '.', '-topmost',
True)
45         root.after_idle(root.call, 'wm', 'attributes',
'.', '-topmost', False)
46
47         app.mainloop()
48
49 main()
```

Tk: Runner

```
1 import tkinter as tk
2 import time
3
4 # TODO: async or threading to run long-running other
5 # processes
6
7 class RunnerApp(tk.Frame):
8     def __init__(self, parent=None):
9         super().__init__(parent)
10        self.pack()
11
12        # Capture event when someone closes the window
13        # with the X on the top-right corner of the window
14        parent.protocol("WM_DELETE_WINDOW",
15        self.close_app)
16
17        self.QUIT = tk.Button(self)
18        self.QUIT["text"] = "QUIT"
19        self.QUIT["fg"] = "red"
20        self.QUIT["command"] = self.close_app
21        self.QUIT.pack({"side": "left"})
22
23        self.start_button = tk.Button(self)
24        self.start_button["text"] = "Start"
25        self.start_button["command"] = self.start
26        self.start_button.pack({"side": "left"})
27
28        self.stop_button = tk.Button(self)
29        self.stop_button["text"] = "Stop"
30        self.stop_button["command"] = self.stop
31        self.stop_button.pack({"side": "left"})
32
33        self.text = tk.Text(self, state='disabled')
34        self.text.pack({"side": "bottom"})
35
36        self.stop_process = False
37
38    def close_app(self):
39        print("close")
40        self.stop_process = True
```

```
40         self.quit()
41
42     def stop(self):
43         print("stop")
44         self.stop_process = True
45         self.add_line('stop')
46
47     def start(self):
48         self.stop_process = False
49         for i in range(100):
50             if self.stop_process:
51                 break
52             self.add_line(str(i))
53             time.sleep(0.1)
54
55     def add_line(self, line):
56         self.text['state'] = 'normal' # allow editing
57         self.text.insert('end', line + "\n")
58         self.text['state'] = 'disabled' # disable
59         self.text.see('end') # scroll to the end as
60         we make progress
61         self.update() # update the content and allow
62         other events (e.g. from stop a\
63         nd quit buttons) to take place
64
65     def main():
66         tk_root = tk.Tk()
67         app = RunnerApp(parent=tk_root)
68
69         tk_root.lift()
70         tk_root.call('wm', 'attributes', '.', '-topmost',
71         True)
72         tk_root.after_idle(tk_root.call, 'wm',
73         'attributes', '.', '-topmost', False)
74
75     main()
```

Tk: Runner with threads

```
1 import tkinter as tk
2 import time
3 import threading
4 import queue
5 import ctypes
6
7 class MyStopButton(Exception):
8     pass
9
10 class ThreadedJob(threading.Thread):
11     def __init__(self, que):
12         self.que = que
13         threading.Thread.__init__(self)
14     def run(self):
15         thread = threading.current_thread()
16         print("Start thread {}".format(thread.name))
17         try:
18             for i in range(10):
19                 print(i)
20                 self.que.put(str(i))
21                 time.sleep(1)
22         except Exception as err:
23             print(f"Exception in {thread.name}: {err}")
24             {err.__class__.__name__})
25
26     def raise_exception(self):
27         thread = threading.current_thread()
28         print(f"Raise exception in {thread.name}")
29         thread_id = self.native_id
30         res =
31         ctypes.pythonapi.PyThreadState_SetAsyncExc(thread_id,
32             ctypes.py_object\
33             (MyStopButton))
34         if res > 1:
35             ctypes.pythonapi.PyThreadState_SetAsyncExc(thread_id, 0)
36             print('Exception raise failure')
37             print("DONE")
```

```
38 class RunnerApp(tk.Frame):
39     def __init__(self, parent=None):
40         super().__init__(parent)
41         self.pack()
42
43         # Capture event when someone closes the window
44         # with the X on the top-right corner of the window
45         parent.protocol("WM_DELETE_WINDOW",
46         self.close_app)
47
48         self.QUIT = tk.Button(self)
49         self.QUIT["text"] = "QUIT"
50         self.QUIT["fg"] = "red"
51         self.QUIT["command"] = self.close_app
52         self.QUIT.pack({"side": "left"})
53
54         self.start_button = tk.Button(self)
55         self.start_button["text"] = "Start"
56         self.start_button["command"] = self.start
57         self.start_button.pack({"side": "left"})
58
59         self.stop_button = tk.Button(self)
60         self.stop_button["text"] = "Stop"
61         self.stop_button["command"] = self.stop
62         self.stop_button.pack({"side": "left"})
63
64         self.text = tk.Text(self, state='disabled')
65         self.text.pack({"side": "bottom"})
66
67         self.stop_process = False
68
69     def close_app(self):
70         print("close")
71         self.stop_process = True
72         self.quit()
73
74     def stop(self):
75         print("stop")
76         print(self.job.name)
77         self.job.raise_exception()
78         #self.stop_process = True
79         self.add_line('stop')
80
```

```

81     def start(self):
82         self.stop_process = False
83         self.start_button['state'] = 'disabled'
84         self.que = queue.Queue()
85         self.job = ThreadedJob(self.que)
86         self.job.start()
87         self.master.after(100, self.process_queue)
88
89     def process_queue(self):
90         print("process " + str(time.time()))
91         if not self.job.is_alive():
92             self.job.join()
93             self.job = None
94             self.stop_process = True
95             self.start_button['state'] = 'normal'
96             print("finished")
97             return
98
99     try:
100         msg = self.que.get(0)
101         self.add_line(msg)
102     except queue.Empty:
103         pass
104     finally:
105         if not self.stop_process:
106             self.master.after(100,
self.process_queue)
107
108     def add_line(self, line):
109         self.text['state'] = 'normal' # allow editing
of the Text widget
110         self.text.insert('end', line + "\n")
111         self.text['state'] = 'disabled' # disable
editing
112         self.text.see('end') # scroll to the end as
we make progress
113         self.update() # update the content and allow
other events (e.g. from stop a\nd quit buttons) to take place
114
115
116
117 def main():
118     tk_root = tk.Tk()
119     app = RunnerApp(parent=tk_root)

```

```

120
121     tk_root.lift()
122     tk_root.call('wm', 'attributes', '.', '-topmost',
123     True)
123     tk_root.after_idle(tk_root.call, 'wm',
124     'attributes', '.', '-topmost', False)
124
125     app.mainloop()
126
127
128 main()

```

Getting started with Tk

```

1 import tkinter as tk
2
3 class Example(tk.Frame):
4     def __init__(self, parent=None):
5         super().__init__(parent)
6         self.pack()
7         self.createWidgets()
8
9     def say_hi(self):
10        print("hi there, everyone! ")
11        print("Name: {}".format(self.name.get()))
12        print("Password:")
13        print("{}".format(self.password.get()))
14        print("count: {}".format(self.count.get()))
15        self.password.delete(0, 'end')
16
17     def createWidgets(self):
18         self.QUIT = tk.Button(self)
19         self.QUIT["text"] = "QUIT"
20         self.QUIT["fg"] = "red"
21         self.QUIT["command"] = self.quit
22         self.QUIT.pack({"side": "left"})
23
24         # Simple Label widget:
25         self.name_title = tk.Label(self, text="Name:")
26         self.name_title.pack({"side": "left"})
27
28         # Simple Entry widget:

```

```

29         self.name = tk.Entry(self)
30         self.name.pack({"side": "left"})
31         self.name.insert(0, "Your name")
32
33     # Simple Label widget:
34     self.password_title = tk.Label(self,
35                                     text="Password:")
36     self.password_title.pack({"side": "left"})
37
38     self.count = tk.IntVar()
39     self.count.set(2)
40     self.my_radio = []
41     radio = [(1, "One"), (2, "Two"), (3, "Three")]
42     for ix in range(len(radio)):
43         self.my_radio.append(tk.Radiobutton(self,
44                                           text=radio[ix][1], variable=self.
45                                           count, value=radio[ix][0]))
46         self.my_radio[ix].pack({"side": "bottom"})
47
48     # In order to hide the text as it is typed
49     # (e.g. for Passwords)
50     # set the "show" parameter:
51     self.password = tk.Entry(self)
52     self.password["show"] = "*"
53     self.password.pack({"side": "left"})
54
55     self.hi_there = tk.Button(self)
56     self.hi_there["text"] = "Hello",
57     self.hi_there["command"] = self.say_hi
58
59     self.hi_there.pack({"side": "left"})
60
61 def main():
62     root = tk.Tk()
63     app = Example(parent=root)
64
65     root.lift()
66     root.call('wm', 'attributes', '.', '-topmost',
67 True)
68     root.after_idle(root.call, 'wm', 'attributes',
69 '.', '-topmost', False)
70
71     app.mainloop()
72
73 main()

```

Exercise: Tk - Calculator one line

Write a Tk application that behaves like a one-line calculator. It has an entry box where one can enter an expression like “2 + 3” and a button.

When the button is pressed the expression is calculated.

There is another button called “Quit” that will close the application.

Exercise: Tk Shopping list

Create a Tk application that allows you to create a [shopping list](#).

Exercise: Tk TODO list

- Create a Tk application to handle your TODO items.
- A Menu to be able to exit the application
- A List of current tasks.
- A way to add a new task. For a start each task has a title and a status. The status can be “todo” or “done”. (default is “todo”)
- A way to edit a task. (Primarily to change its title).
- A way to mark an item as “done” or mark it as “todo”.
- A way to move items up and down in the list.
- The application should automatically save the items in their most up-to-date state in a “database”. The database can be a JSON file or and SQLite database or anything else you feel fit.

Exercise: Tk Notepad

- Create a Notepad like text editor.
- It needs to have a menu called File with item:
New/Open/Save/Save As/Exit
- It needs to have an area where it can show the content of a file. Let you edit it.
- Create a menu called About that displays an about box containing the names of the authors of the app.
- Menu item to Search for text.

Exercise: Tk Copy files

An application that allows you to type in, or select an existing file and another filename
for which the file does not exists.
Then copy the old file to the new name.

Exercise: Tk

- Application that accepts a “title” - line of text, a file selected, a new filename (that probably does not exist) and then runs.

Solution: Tk - Calculator one line

```

1 import tkinter as tk
2
3 app = tk.Tk()
4 app.title('Calculator')
5
6 entry = tk.Entry(app)
7 entry.pack()
8
9 def calc():
10     print("clicked")
11     inp = entry.get()

```

```
12 print(inp)
13 out = eval(inp)
14 entry.delete(0, tk.END)
15 entry.insert(0, out)
16
17 def close():
18     app.destroy()
19
20 calc_btn = tk.Button(app, text='Calculate', width=25,
command=calc)
21 calc_btn.pack()
22
23
24 close_btn = tk.Button(app, text='Close', width=25,
command=close)
25 close_btn.pack()
26
27 app.mainloop()
```

```
1 import tkinter as tk
2
3 # This solutions is not ready yet
4
5 app = tk.Tk()
6 app.title('Calculator')
7
8 entry = tk.Entry(app)
9 entry.pack()
10
11 def calc():
12     print("clicked")
13     inp = entry.get()
14     print(inp)
15     out = eval(inp)
16     entry.delete(0, tk.END)
17     entry.insert(0, out)
18
19 def close():
20     app.destroy()
21     exit()
22
23 def enter(num):
24     entry.insert(tk.END, num)
25
```

```
26 def add_button(num, frame):
27     btn = tk.Button(frame, text=num, width=25,
28                     command=lambda : enter(num))
29     btn.pack(side="left")
30     buttons[num] = btn
31
32 numbers_frame = tk.Frame(app)
33 numbers_frame.pack()
34 numbers_row = { }
35 numbers_row[1] = tk.Frame(numbers_frame)
36 numbers_row[1].pack(side="top")
37 numbers_row[2] = tk.Frame(numbers_frame)
38 numbers_row[2].pack(side="top")
39 numbers_row[3] = tk.Frame(numbers_frame)
40 numbers_row[3].pack(side="top")
41 ops_row = tk.Frame(numbers_frame)
42 ops_row.pack(side="top")
43
44 buttons = { }
45
46 add_button(1, numbers_row[1])
47 add_button(2, numbers_row[1])
48 add_button(3, numbers_row[1])
49 add_button(4, numbers_row[2])
50 add_button(5, numbers_row[2])
51 add_button(6, numbers_row[2])
52 add_button(7, numbers_row[3])
53 add_button(8, numbers_row[3])
54 add_button(9, numbers_row[3])
55
56 for op in ['+', '-', '*', '/']:
57     add_button(op, ops_row)
58
59
60 calc_btn = tk.Button(app, text='Calculate', width=25,
61                     command=calc)
62 calc_btn.pack()
63
64 close_btn = tk.Button(app, text='Close', width=25,
65                     command=close)
66 close_btn.pack()
67 app.mainloop()
```

Solution: Tk

```
1 import tkinter as tk
2 from tkinter import filedialog
3
4 def run_process():
5     print("---- Start processing ----")
6     title = title_entry.get()
7     print(title)
8     filename = input_file.get()
9     print(filename)
10
11     app.destroy()
12
13 def select_input_file():
14     file_path = filedialog.askopenfilename()
15     filedialog.asksaveasfile()
16     print(file_path)
17     input_file.set(file_path)
18
19 app = tk.Tk()
20 app.title('Convert file')
21
22 input_file = tk.StringVar()
23
24 title_label = tk.Label(app, text='Title')
25 title_label.pack()
26 title_entry = tk.Entry(app)
27 title_entry.pack()
28
29 input_button = tk.Button(app, text='Input file',
command=select_input_file)
30 input_button.pack()
31 input_label = tk.Label(app, textvariable=input_file)
32 input_label.pack()
33
34
35 button = tk.Button(app, text='Process', width=25,
command=run_process)
36 button.pack()
37
38 app.mainloop()
```

Solution: Tk Notepad

```
1 import tkinter as tk
2 from tkinter import filedialog, simpledialog,
messagebox
3 import os
4
5 file_path = None
6
7 app = tk.Tk()
8 app.title('Menu')
9
10 def run_new():
11     global file_path
12     file_path = None
13     text.delete(1.0, tk.END)
14
15 def run_open():
16     global file_path
17     file_path = filedialog.askopenfilename(filetypes=
((("Any file", "*"),)))
18     if file_path and os.path.isfile(file_path):
19         with open(file_path) as fh:
20             content = fh.read()
21             text.delete(1.0, tk.END)
22             text.insert('end', content)
23
24 def run_save():
25     global file_path
26     if file_path is None:
27         file_path =
filedialog.asksaveasfilename(filetypes=(("Any file",
"*"),))
28         if not file_path:
29             file_path = None
30             return
31         #print(f'''{file_path}''')
32         content = text.get(1.0, tk.END)
33         with open(file_path, 'w') as fh:
34             fh.write(content)
35
36 def run_exit():
37     print("exit")
```

```
38     app.destroy()
39
40 def run_about():
41     #print(dir(simpledialog))
42     #answer = simpledialog.Dialog(app, "The title")
43     messagebox.showinfo(title = "About", message =
44 "This simple text editor was crea\
45 ted as a solution for the exercise.\n\nCopyright:
Gabor Szabo")
46
47 menubar = tk.Menu(app)
48
49 menu1 = tk.Menu(menubar, tearoff=0)
50 menu1.add_command(label="New", underline=0,
command=run_new)
51 menu1.add_command(label="Open", underline=0,
command=run_open)
52 menu1.add_command(label="Save", underline=0,
command=run_save)
53 menu1.add_separator()
54 menu1.add_command(label="Exit", underline=1,
command=run_exit)
55 menubar.add_cascade(label="File", underline=0,
menu=menu1)
56
57 menubar.add_command(label="About", underline=0,
command=run_about)
58
59 app.config(menu=menubar)
60
61 text = tk.Text(app)
62 text.pack({"side": "bottom"})
63
64 app.mainloop()
65
66 # TODO: Show the name of the file somewhere? Maybe at
the bottom in a status bar?
67 # TODO: Indicate if the file has been changed since
the last save?
68 # TODO: Ask before exiting or before replacing the
content if the file has not been \
69 saved yet.
70 # TODO: Undo/Redo?
71 # TODO: Search?
72 # TODO: Search and Replace?
```

Simple file dialog

```
1 from tkinter import filedialog  
2  
3 input_file_path = filedialog.askopenfilename(filetypes=([("Excel files", "*.xlsx"), ("CSV files", "*.csv"), ("Any file", "*"))))  
4 print(input_file_path)  
5  
6 input("Press ENTER to end the script...")
```

Python Pitfalls

Reuse of existing module name

```
1 import random
```

```
2
```

```
3 print(random.random())
```

```
1 $ python examples/pitfalls/random.py
```

```
1 Traceback (most recent call last):
2   File "examples/pitfalls/random.py", line 1, in <module>
3     import random
4   File ".../examples/pitfalls/random.py", line 3, in <module>
5     print(random.random())
6 TypeError: 'module' object is not callable
```

- Write an example to use random number and call your example **number.py**
- Same with any other module name.
- Lack of multi-level namespaces
- Solution: user longer names. Maybe with project specific names.

Use the same name more than once

```
1 class Corp(object):
2     people = []
3     def add(self, name, salary):
4         Corp.people.append({ 'name': name, 'salary' :
```

```
salary})  
5  
6     def total(self):  
7         self.total = 0  
8         for n in Corp.people:  
9             self.total += n['salary']  
10        return self.total  
11  
12 c = Corp()  
13 c.add("Foo", 19)  
14 print(c.total())  
15  
16 c.add("Bar", 23)  
17 print(c.total())
```

```
1 $ python examples/pitfalls/corp.py
```

```
1 19  
2 Traceback (most recent call last):  
3   File "examples/pitfalls/corp.py", line 19, in  
<module>  
4     print(c.total())  
5 TypeError: 'int' object is not callable
```

Compare string and number

```
1 x = 2  
2 y = "2"  
3  
4 print(x > y)  
5 print(x < y)
```

Python 2 - compares them based on the type of values (wat?)

```
1 $ python examples/pitfalls/compare.py
```

```
1 False  
2 True
```

Python 3 - throws exception as expected.

```
1 $ python3 examples/pitfalls/compare.py
```

```
1 Traceback (most recent call last):
2   File "examples/pitfalls/compare.py", line 4, in
<module>
3     print(x > y)
4 TypeError: unorderable types: int() > str()
```

Compare different types

```
1 x = 2
2 y = "2"
3
4 print(x == y)
5
6 with open(__file__) as fh:
7     print(fh == x)
```

In both Python 2 and Pyhton 3 these return **False**

```
1 import sys
2
3 hidden = 42      # would be random
4
5 if sys.version_info.major < 3:
6     guess = raw_input('Your guess: ')
7 else:
8     guess = input('Your guess: ')
9
10 if hidden == guess:
11     print("Match!")
```

Will never match. Even if user types in 42. - Hard to debug and understand as there is no error.

Sort mixed data

```
1 from __future__ import print_function
2
3 mixed = [10, '1 foo', 42, '4 bar']
4 print(mixed)    # [100, 'foo', 42, 'bar']
5 mixed.sort()
6 print(mixed)    # [42, 100, 'bar', 'foo']
```

In Python 2 it “works” is some strange way.

```
1 $ python examples/pitfalls/sort.py
```

```
1 [10, '1 foo', 42, '4 bar']
2 [10, 42, '1 foo', '4 bar']
```

In Python 3 it **correctly** throws an exception.

```
1 air:python gabor$ python3 examples/pitfalls/sort.py
```

```
1 [10, '1 foo', 42, '4 bar']
2 Traceback (most recent call last):
3   File "examples/pitfalls/sort.py", line 5, in <module>
4     mixed.sort()
5 TypeError: unorderable types: str() < int()
```

Linters

Static Code Analyzis - Linters

- PEP8
- Flake8
- Pylint

PEP8

```
1 pip install pep8
```

- [pep8](#)
- [pep8](#)

F811 - redefinition of unused

```
1 import subprocess
2 import datetime
3 import sys
4 from datetime import datetime
```

```
1 $ flake8 importer.py
2 importer.py:4:1: F811 redefinition of unused 'datetime'
from line 2
```

Warn when Redefining functions

```
1 sum = 42
2
```

```
3 def len(thing):
4     print(f"Use {thing}.__len__() instead!")
5
6 len("abc")
```

```
1 **** Module redef
2 redef.py:1:0: C0111: Missing module docstring
(missing-docstring)
3 redef.py:2:0: W0622: Redefining built-in 'sum'
(redefined-builtin)
4 redef.py:4:0: W0622: Redefining built-in 'len'
(redefined-builtin)
5 redef.py:2:0: C0103: Constant name "sum" doesn't
conform to UPPER_CASE naming style \
6 (invalid-name)
7 redef.py:4:0: C0111: Missing function docstring
(missing-docstring)
8
9 -----
-----
```

10 Your code has been rated at -2.50/10 (previous run:
-2.50/10, +0.00)

Python .NET

IronPython

Python running on the [DLR](#)
that is on top of the [CLR](#) of Microsoft.

- [https://ironpython.net/]
- [GitHub](#)
- Only supports Python 2
- [Iron Python 3](#)
- Not ready for production

Use .NET libraries from Python

- [pythonnet](#)
- [pythonnet source code](#)

¹ pip install pythonnet

The latest Visual Studio is supposed to include [Nuget](#), but if you don't have it, you can download it from [Nuget downloads](#)

Make sure `nuget.exe` is somewhere in your PATH:

For example I've created `C:\Bin`, put the `nuget.exe` in this directory and added `C:\Bin` to the PATH.

Then install the compilers using nuget install Microsoft.Net.Compilers as suggested on [Roslyn](#)

This created the Microsoft.Net.Compilers.3.4.0 directory in my home directory

Make sure csc.exe is somewhere in your PATH or use the full path to it:

```
"UsersGabor SzaboMicrosoft.Net.Compilers.3.4.0\tools\csc.exe"  
/t:library MyMath.cs
```

Python and .NET console

```
1 import clr  
2 from System import Console  
3  
4 Console.WriteLine("Hello My World!")
```

```
1 python net_console.py
```

Python and .NET examples

```
1 namespace MyMath  
2 {  
3     public static class MyMathClass  
4     {  
5         public static int addInts(int a, int b)  
6         {  
7             return a+b;  
8         }  
9  
10        public static double addDouble(double a,  
double b)  
11        {  
12            return a+b;  
13        }  
14
```

```

15         public static string addString(string a,
string b)
16         {
17             return a+" "+b;
18         }
19
20         public static bool andBool(bool a, bool b)
21         {
22             return a && b;
23         }
24
25         public static string str_by_index(string[] a,
int b)
26         {
27             return a[b];
28         }
29         public static int int_by_index(int[] a, int b)
30         {
31             return a[b];
32         }
33
34     }
35 }
```

```

1 import clr
2 dll = clr.FindAssembly('MyMath')    # returns path to
dll
3 assembly = clr.AddReference('MyMath')
4 #print(type(assembly)) # <class
'System.Reflection.RuntimeAssembly'>
5 #print(dir(assembly))
6 from MyMath import MyMathClass
7 from MyMath import MyMathClass as My
8
9
10 assert My.addInts(2, 3) == 5
11 assert My.addInts(2.7, 7.8) == 9
12 assert My.addDouble(11.2, 23.3) == 34.5
13 assert My.addString("hello", "world") == "hello world"
14
15 assert My.andBool(1, 1) is True
16 assert My.andBool(1, 0) is False
17 assert My.andBool(True, True) is True
```

```
18 assert My.andBool(False, True) is False
19
20 assert My.str_by_index(["apple", "banana", "peach"],
0) == "apple"
21 assert My.str_by_index(["apple", "banana", "peach"],
1) == "banana"
22 assert My.int_by_index([17, 19, 42], 1) == 19
23 # Mixed list cannot be passed
24
25 # tuple can be passed
26 assert My.int_by_index((17, 21, 42), 2) == 42
27
28 # TODO: string, char, float
29 # TODO strings, lists, dicts,
30 # TODO complex data structures in C#
31 # TODO Async
```

```
1 csc /t:library MyMath.cs
2 python myapp.py
```

C:\Windows\Microsoft.NET\Framework\v4.0.30319\
C:\Program Files\dotnet\

Exercise Python and .NET

- Take a .NET class that you would like to use, try that.

Python and Java

Jython

- [Jython](#)
- See separate chapter

Calling Java from Python

- [Pyjnius/Jnius](#) - [GitHub](#)
- [JCC](#)
- [javabridge](#)
- [Jpype](#) - [GitHub](#)
- [Py4j](#)

Jython - Python running on the JVM

Jython Installation

- [Jython](#)
- java -jar jython-installer-2.7.0.jar
- ~/jython2.7.0/

Jython Installation

```
1 java -jar ~/jython2.7.0/jython.jar
2
3 java -jar ~/jython2.7.0/jython.jar some.py
```

Jython load Java class

```
1 cd examples/mymath/
2 java -jar ~/jython2.7.0/jython.jar
3 Jython 2.7.0 (default:9987c746f838, Apr 29 2015,
02:25:11)
4 [Java HotSpot(TM) 64-Bit Server VM (Oracle
Corporation)] on java1.8.0_60
5 Type "help", "copyright", "credits" or "license" for
more information.
6 >>> import Calculator
7 >>> Calculator.add(2, 3)
8 5
9 >>> Calculator.add(10, 3)
10 10
11 >>>
```

Jython load Java class in code

```
1 public class Calculator {  
2     public static Integer add(Integer a, Integer b) {  
3         if (a == 10) {  
4             return 10;  
5         }  
6         return a+b;  
7     }  
8  
9 }
```

```
1 # use only with Jython  
2  
3 import Calculator  
4 print(Calculator.add(4, 8))  
5 print(Calculator.add(10, 8))
```

```
1 cd examples/jython/mymath/  
2 java -jar ~/jython2.7.0/jython.jar calc.py
```

Jython test Java class

```
1 import unittest  
2 import Calculator  
3  
4 class TestAdd(unittest.TestCase):  
5  
6     def test_add(self):  
7         self.assertEqual(Calculator.add(4, 8), 12)  
8         self.assertEqual(Calculator.add(10, 8), 18)  
9         self.assertEqual(Calculator.add(-1, 1), 0)  
10  
11 if __name__ == '__main__':  
12     unittest.main()  
13
```

```
1 java -jar ~/jython2.7.0/jython.jar calc.py  
2 java -jar ~/jython2.7.0/jython.jar -m unittest discover
```

PIL - Pillow

Install Pillow

- [Pillow](#)
- [Pillow on PyPI](#)
- [GitHub](#)

```
1 pip install pillow
```

Create First Image

```
1 from PIL import Image
2
3 img = Image.new('RGB', size=(100, 60), color='#eb8634')
4 img.save('first.png')
5 img.show()      # Using ImageMagic on Linux
```

- Color can be one of the well-known names e.g. “red”
- Color can be RGB in decimal or hex. (RGB=Red Green Blue)

Write Text on Image

```
1 from PIL import Image, ImageDraw
2
3 img = Image.new('RGB', size=(100, 60),
color='#eb8634')
4
5 draw = ImageDraw.Draw(img)
6 draw.text(
7     text="Some text",
8     xy=(10, 20),
9 )
```

```
10  
11 img.save('first.png')  
12 img.show()
```

Select font for Text on Image

```
1 from PIL import Image, ImageDraw, ImageFont  
2  
3 img = Image.new(mode='RGB', size=(300, 60),  
color='#eb8634')  
4 font =  
ImageFont.truetype('Pillow/Tests/fonts/FreeMono.ttf', 20)  
5 #font =  
ImageFont.truetype(f'c:\Windows\Fonts\Candara.ttf', 30)  
6 #font =  
ImageFont.truetype(f'c:\Windows\Fonts\Candarab.ttf', 30)  
7 #font =  
ImageFont.truetype(f'c:\Windows\Fonts\david.ttf', 30)  
8  
9  
10 draw = ImageDraw.Draw(img)  
11 draw.text(  
12     text="Some text",  
13     xy=(10, 20),  
14     font=font,  
15 )  
16  
17 img.save('first.png')  
18 img.show()
```

Font directories

```
1 Linux: /usr/share/fonts/  
2 Max OS: /Library/Fonts/  
3 Windows: C:\Windows\fonts
```

Get size of an Image

```
1 from PIL import Image
2 import sys
3 if len(sys.argv) !=2:
4     exit(f"Usage: {sys.argv[0]} {FILENAME}")
5
6 in_file = sys.argv[1]
7
8 img = Image.open(in_file)
9 print(img.size)      # a tuple
10 print(img.size[0])   # width
11 print(img.size[1])   # height
```

Get size of text

```
1 font = ImageFont.truetype(
2         'path/to/font.ttf', size
3     )
4 size = font.getsize(text)
```

Resize an existing Image

```
1 from PIL import Image
2
3 in_file = 'in.png'
4 out_file = 'new.png'
5
6 img = Image.open(in_file)
7
8 size = (img.size[0] / 2, img.size[1] / 2)
9 img.thumbnail(size)
10
11 img.save(out_file)
```

Crop an existing Image

```
1 from PIL import Image
2
3 in_file = 'in.png'
4 out_file = 'out.png'
```

```
5
6 img = Image.open(in_file)
7 width, height = img.size
8 width, height = img.size
9
10 # crop
11 # 10 pixels from the left
12 # 20 pixels from the top
13 # 30 pixels from the right
14 # 40 pixels from the bottom
15
16 cropped = img.crop((10, 20, width - 30, height - 40))
17 cropped.save(out_file)
18 cropped.show()
```

Combine two images

- Load one image from file
- Create a plain background
- Put the loaded image on the background
- Save the combined image

Rotated text

```
1 from PIL import Image, ImageDraw, ImageFont, ImageOps
2
3 img = Image.new(mode='RGB', size=(400, 200),
color='#eb8634')
4
5 font =
ImageFont.truetype('Pillow/Tests/fonts/FreeSansBold.ttf',
30)
6
7 text_layer = Image.new('L', (330, 50))
8 draw = ImageDraw.Draw(text_layer)
9 draw.text( (30, 0), "Text slightly rotated",
font=font, fill=255)
10
11 rotated_text_layer = text_layer.rotate(10.0, expand=1)
12 img.paste( ImageOps.colorize(rotated_text_layer,
```

```
(0,0,0), (10, 10,10)), (42,60), ro\
13 tated_text_layer)
14 img.show()
```

Rotated text in top-right corner

TODO: fix this

```
1 from PIL import Image, ImageDraw, ImageFont, ImageOps
2
3 width = 400
4 height = 200
5 start = 100
6 end = 50
7
8 img = Image.new(mode='RGB', size=(width, height),
color='#FAFAFA')
9
10 stripe_color = "#eb8634"
11 draw = ImageDraw.Draw(img)
12 draw.polygon([(width-start, 0), (width-end, 0),
(width, end), (width, start)], fill\
13 =stripe_color)
14
15
16 font =
ImageFont.truetype('Pillow/Tests/fonts/FreeSansBold.ttf',
30)
17 text_layer = Image.new('RGB', size=(100, 100),
color=stripe_color)
18
19 draw = ImageDraw.Draw(text_layer)
20 text = "Free"
21 size = draw.textsize(text=text, font=font)
22 # print(size)
23 draw.text(xy=(20, 0), text=text, font=font, fill=1)
24 #
25 rotated_text_layer = text_layer.rotate(-45.0,
expand=0)
26 rotated_text_layer.show()
27 #img.paste( ImageOps.colorize(rotated_text_layer,
(0,0,0), (10, 10,10)), (42,60), r\
28 otated_text_layer)
```

```
29 #img.paste(im = rotated_text_layer, box=(300, 0))
30 #img.paste(im = text_layer, box=(300, 0))
31 #img.show()
```

Embed image (put one image on another one)

```
1 from PIL import Image
2
3 in_file = 'python.png'
4
5 width = 600
6 height = 300
7 background = Image.new(mode='RGB', size=(width,
height), color='#AAFAFA')
8
9 img = Image.open(in_file)
10 (emb_width, emb_height) = img.size
11 print(emb_width)
12 print(emb_height)
13
14 # slightly off the lower right corner of the
background image
15 # using the image as the mask makes its background
transparent
16 background.paste(im = img, box=(width-emb_width-10,
height-emb_height-10), mask=img)
17
18 background.show()
```

Draw a triangle

```
1 from PIL import Image, ImageDraw
2
3 img = Image.new(mode='RGB', size=(800, 450),
color='#eb8634')
4
5 draw = ImageDraw.Draw(img)
6 draw.polygon([(800, 275), (800, 450), (300, 450)])
7
```

```
8 img.save('first.png')
9 img.show()
```

Draw a triangle and write text in it

```
1 from PIL import Image, ImageDraw, ImageFont
2
3 img = Image.new(mode='RGB', size=(800, 450),
color='#eb8634')
4
5 draw = ImageDraw.Draw(img)
6 draw.polygon([(800, 275), (800, 450), (300, 450)], fill = (255, 255, 255))
7
8 font =
ImageFont.truetype('Pillow/Tests/fonts/FreeSansBold.ttf',
30)
9
10 draw.text((500, 400), 'Hello from Python', (0, 0, 0),
font=font)
11
12
13 img.save('first.png')
14 img.show()
```

Draw a triangle and write rotated text in it

```
1 from PIL import Image, ImageDraw, ImageFont, ImageOps
2
3 img = Image.new(mode='RGB', size=(400, 200),
color='#eb8634')
4
5 # draw = ImageDraw.Draw(img)
6 # draw.polygon([(800, 275), (800, 450), (300, 450)], fill = (255, 255, 255))
7 #
8 #
9 #font = ImageFont.load_default()
10 font =
ImageFont.truetype('Pillow/Tests/fonts/FreeSansBold.ttf',
30)
```

```
11 # txt = Image.new('L', (500, 500))
12 # d = ImageDraw.Draw(txt)
13 # d.text((300, 400), 'Hello from Python', font=font,
color="white")
14 # w=txt.rotate(17.5, expand=1)
15 #
16 # img.paste(txt)
17 # img.paste( ImageOps.colorize(w, (0,0,0),
(255,255,84)), (242,60), w)
18 # # img.save('first.png')
19 # img.show()
20 #
21
22 text_layer = Image.new('L', (300, 50))
23 draw = ImageDraw.Draw(text_layer)
24 draw.text( (30, 0), "Text slightly rotated",
font=font, fill=255)
25
26 rotated_text_layer = text_layer.rotate(10.0, expand=1)
27 img.paste( ImageOps.colorize(rotated_text_layer,
(0,0,0), (10, 10,10)), (42,60), ro\
28 tated_text_layer)
29 img.show()
```

Draw a rectangular

```
1 from PIL import Image, ImageDraw
2
3 img = Image.new(mode='RGB', size=(800, 450),
color='#eb8634')
4
5 draw = ImageDraw.Draw(img)
6 draw.polygon([(400, 200), (400, 300), (200, 300), (200,
200) ])
7
8 img.save('first.png')
9 img.show()
```

Draw a rectangle

```
1 from PIL import Image, ImageDraw
2
3 img = Image.new('RGB', size=(100, 100))
4
5 draw = ImageDraw.Draw(img)
6 draw.rectangle((10, 10, 90, 90), fill="yellow",
outline="red")
7 img.show()
```

Draw circle

```
1 from PIL import Image, ImageDraw
2
3 img = Image.new('RGB', (200, 200))
4
5 draw = ImageDraw.Draw(img)
6 draw.ellipse((50, 50, 150, 150), fill="#F00F4F")
7 img.show()
```

Draw heart

```
1 from PIL import Image, ImageDraw
2
3 def heart(size, fill):
4     width, height = size
5     img = Image.new('RGB', size, (0, 0, 0, 0))
6     draw = ImageDraw.Draw(img)
7     polygon = [
8         (width / 10, height / 3),
9         (width / 10, 81 * height / 120),
10        (width / 2, height),
11        (width - width / 10, 81 * height / 120),
12        (width - width / 10, height / 3),
13    ]
14    draw.polygon(polygon, fill=fill)
15    #img.show()
16
17    draw.ellipse((0, 0, width / 2, 3 * height / 4),
fill=fill)
18    draw.ellipse((width / 2, 0, width, 3 * height /
4), fill=fill)
```

```
19     return img
20
21 img = heart((50, 40), "red")
22 img.show()
```

Some samples, including this one, originally by [Nadia Alramli](#)

Rectangle with rounded corners

```
1 from PIL import Image, ImageDraw
2
3
4 def round_corner(radius, fill):
5     """Draw a round corner"""
6     corner = Image.new('RGB', (radius, radius), (0, 0,
0, 0))
7     draw = ImageDraw.Draw(corner)
8     draw.pieslice((0, 0, radius * 2, radius * 2), 180,
270, fill=fill)
9     return corner
10
11
12 def round_rectangle(size, radius, fill):
13     """Draw a rounded rectangle"""
14     width, height = size
15     rectangle = Image.new('RGB', size, fill)
16     corner = round_corner(radius, fill)
17     rectangle.paste(corner, (0, 0))
18     rectangle.paste(corner.rotate(90), (0, height -
radius)) # Rotate the corner an\
19 d paste it
20     rectangle.paste(corner.rotate(180), (width -
radius, height - radius))
21     rectangle.paste(corner.rotate(270), (width -
radius, 0))
22     return rectangle
23
24
25 img = round_rectangle((50, 50), 10, "yellow")
26
27 img.show()
```

Some samples, including this one, originally by [Nadia Alramli](#)

TODO

<http://web.archive.org/web/20130115175340/http://nadiana.com/pil-tutorial-basic-advanced-drawing>

- Make the background color change from top to bottom
- Add straight lines to existing images
- Blur image
- Add rectangular to area on existing image
- Draw other simple images

FAQ

How not to name example scripts?

Don't - by mistake - call one of your files the same as a module you will be loading.

For example `random.py` is a bad idea if you will `import random`. Your code will try to locate `random.py` to load, but will find itself and not the one that comes with Python.

Python will also create a `random.pyc` file - a compiled file - and it will take time till you recall this and delete that too.

Till then the whole thing will seem to be broken.

Platform independent code

In general Python is platform independent, but still needs some care to make sure

you don't step on some aspects of Operating System or the file system that works differently on other OS-es.

- Filenames are case sensitive on some OS-es (e.g. Windows). They used to be restricted to 8.3. Make sure you are within the restriction of every OS you might want to use.
- Directory path: (slash or backslash or something else?) use the `os.path` methods.
- `os.path.expanduser('~')` works on both Linux and Windows, but the root of a Linux/Unix file system starts with a slash (/)

and on Windows it is c:\ and d:\ etc.

- On Linux/Unix you have user ‘root’ and on Windows ‘Administrator’
- File permissions are different on Linux and Windows.
- Stay away from OS specific calls, but as a last resort use os.name or sys.platform to figure out which os is this.
os.name is ‘posix’ on Linux and ‘nt’ on Windows.
- For GUI use wxWindows that has a native look on Windows and Gnome look on Linux.
- Pay attention to any 32/64 bit issues. Big/Little Endian issues.
- Some modules might be OS specific. Check the documentation.
- Pay attention to the use of os.system and subsystem modules.

How to profile a python code to find causes of slowness?

Use one of these modules:

- cProfile is in C. It is faster and preferable.
- profile

pdb = Python Debugger

Include the following code in your script at any point, and run the script as you’d do normally.

It will stop at the given point and enter the debugger.

```
1 import pdb; pdb.set_trace()
```

[pdb](#)

Avoid Redefining functions

Can I tell python to stop compilation when someone is redefining a function?

Or at least give me a warning?

Use `pylint` for that

Appendix

print_function

```
1 from __future__ import print_function  
2  
3 print(23)
```

Dividers (no break or continue)

We will see how break and continue work, but first let's see a loop to find all the dividers on a number n.

```
1 i = 2  
2 n = 3*5*7  
3 while i < n:  
4     if (n / i) * i == n:  
5         print('{:2} divides {}'.format(i, n))  
6     i = i + 1
```

```
1 3 divides 105  
2 5 divides 105  
3 7 divides 105  
4 15 divides 105  
5 21 divides 105  
6 35 divides 105
```

Lambdas

```
1 a = lambda x: True  
2 b = lambda x: False  
3 c = lambda x: x  
4 #c = lambda x: return  
5 #c = lambda x: pass
```

```
6 d = lambda x: c(x)+c(x)
7
8 print(a(1))
9 print(b(1))
10 print(c(42))
11 print(d(21))
```

Abstract Class

```
1 import abc
2
3 class Port():
4     __metaclass__ = abc.ABCMeta
5
6     @abc.abstractmethod
7     def num(self):
8         pass
9
10 class HTTPPort(Port):
11     def num(self):
12         return 80
13
14 class FTPPort(Port):
15     def num(self):
16         return 21
17
18 class ZorgPort(Port):
19     def nonum(self):
20         return 'zorg'
21
22 f = FTPPort()
23 print(f.num())
24 h = HTTPPort()
25 print(h.num())
26 z = ZorgPort()
27 # Traceback (most recent call last):
28 #   File "abstract.py", line 26, in <module>
29 #       z = ZorgPort()
30 # TypeError: Can't instantiate abstract class ZorgPort
31 # with abstract methods num
```

32

```
33 print(z.num())
```

Remove file

[os.remove](#) or
[os.unlink](#)

Modules: more

- sys.modules to list loaded modules
- imp.reload to reload module (Just reload before 3.3)

```
1 import __builtin__
2
3 def xx(name):
4     print("hello")
5 __builtin__.__import__ = xx;
6
7 print('body')
8 def f():
9     print("in f")
```

```
1 import sys
2
3 print('mod' in sys.modules) # False
4
5 import mod
6 print('mod' in sys.modules) # True
7 print(sys.modules['mod'])
8     # <module 'mod' from
9     '/stuff/python/examples/modules/mod.py'>
10 print(sys.modules["sys"])      # <module 'sys' (built-
    in)>
```

import hooks

Python resources

- [Central Python site](#)
- [Python documentation](#)
- [Learning Python the Hard way](#)
- [Python Weekly](#)
- [PyCoder's Weekly](#)

Progress bar

```
1 # http://stackoverflow.com/questions/3173320/text-
progress-bar-in-the-console
2 import time, sys
3
4 for i in range(10):
5     sys.stdout.write('\r' + '=' * i)
6     sys.stdout.flush()
7     time.sleep(1)
```

from future

```
1 from __future__ import print_function
2 from __future__ import division
```

or

```
1 from __future__ import print_function, division
```

See also [future](#)

We cannot import everything that is in **future**, because we don't know what will be in **future** in the future....
and we don't want to blindly change the behaviour of Python.

Variable scope

- There are two scopes: outside of all functions and inside of a function.
- The first assignment to a variable defines it.
- Variables that were declared outside all functions can be seen inside, but cannot be changed.
- One can connect the outside name to an inside name using the ‘global’ keyword.
- if and for blocks don’t provide scoping.

```
1 a = 23
2
3 def main():
4     global b
5     b = 17
6     c = 42
7     print('a:', a)      # a: 23
8     print('b:', b)      # b: 17
9     print('c:', c)      # c: 42
10
11    if True:
12        print('a:', a)      # a: 23
13        print('b:', b)      # b: 17
14        b = 99
15        print('b:', b)      # b: 99
16        print('c:', c)      # c: 42
17
18    print('a:', a)      # a: 23
19    print('b:', b)      # b: 99
20    print('c:', c)      # c: 42
21
22
23 main()
24
25 print('a:', a)  # a: 23
26 print('b:', b)  # b: 99
```

```
27 print('c:', c)  # c:  
28 # Traceback (most recent call last):  
29 #   File "examples\basics\scope.py", line 27, in  
<module>  
30 #     print 'c:', c # c:  
31 # NameError: name 'c' is not defined
```

global scope

scope

```
1 # x is global  
2  
3 x = 1  
4 print(x, "- before sub")  
5  
6 def f():  
7     #print(x, "- inside before declaration")  #  
UnboundLocalError  
8     x = 2  
9     print(x, "- inside sub")  
10  
11 print(x, "- after sub declaration")  
12  
13 f()  
14  
15 print(x, "- after calling sub")  
16  
17 # 1 - before sub  
18 # 1 - after sub declaration  
19 # 2 - inside sub  
20 # 1 - after calling sub
```

```
1 # x is global  
2  
3 def f():  
4     #print(x, "- inside before declaration")  #  
UnboundLocalError
```

```
5     x = 2
6     print(x, "- inside sub")
7
8 x = 1
9 print(x, "- before calling sub")
10
11 print(x, "- after sub declaration")
12
13 f()
14
15 print(x, "- after calling sub")
16
17 # 1 - before calling sub
18 # 1 - after sub declaration
19 # 2 - inside sub
20 # 1 - after calling sub
```

If we declare a variable outside of all the subroutines, it does not matter if we do it before the sub declaration, or after it. In neither case has the global variable any presence inside the sub.

```
1 def f():
2     x = 2
3     print(x, "- inside sub")
4
5 # print(x, " - after sub declaration")    # NameError
6
7 f()
8
9 # print(x, " - after calling sub")      # NameError
10
11 # 2 - inside sub
```

A name declared inside a subroutine is not visible outside.

```
1 def f():
2     global x
3     # print(x)    # NameError
4     x = 2
5     print(x, "- inside sub")
6
7 # print(x, " - after sub declaration")    # NameError
8
9 f()
10
11 print(x, "- after calling sub")
12
13 # 2 - inside sub
14 # 2 - after calling sub
```

Unless it was marked using the `global` word.

type

```
1 x = 2
2 y = '2'
3 z = [2, '2']
4 d = {}
5
6 def f():
7     pass
8 l = lambda q: q
9
10 class Cold():
11     pass
12 cold = Cold()
13
14 class Cnew(object):
15     pass
16 cnew = Cnew()
17
18 # r = xrange(10)    # Python 3 does not have xrange
```

```
19
20 print(type(x))    # <type 'int'>
21 print(type(y))    # <type 'str'>
22 print(type(z))    # <type 'list'>
23 print(type(d))    # <type 'dict'>
24 print(type(f))    # <type 'function'>
25 print(type(l))    # <type 'function'>
26 print(type(Cold)) # <type 'classobj'>
27 print(type(cold)) # <type 'instance'>
28 print(type(Cnew)) # <type 'type'>
29 print(type(cnew)) # <class '__main__.Cnew'>
30 #print(type(r))  # <type 'xrange'>
31
32 print(type(x).__name__) # int
33 print(type(y).__name__) # str
34 print(type(z).__name__) # list
```

Look deeper in a list

```
1 x = ['abcd', 'efgh']
2 print(x)          # ['abcd', 'efgh']
3
4 print(x[0:1])    # ['abcd']
5 print(x[0])       # 'abcd'
6
7 print(x[0][0])   # a
8 print(x[0][1])   # b
9 print(x[0][0:2]) # ab
```

Exercise: iterators - count

- Reimplement the count functions of itertools using iterator class.

(We have this as one of the example)

Simple function (before generators)

TODO: probably not that interesting

```
1 def number():
2     return 42
3
4 print(number())      # 42
5 print(number())      # 42
6 print(number())      # 42
```

```
1 def number():
2     return 42
3     return 19
4     return 23
5
6 print(number()) # 42
7 print(number()) # 42
8 print(number()) # 42
```

Other slides

Other slides

Some slides that used to be part of the material and they might return to be there, but for now they were parked here.

Atom for Python

Some details about the Atom editor. You can freely skip this part. Personally I don't use it now.

- [Atom](#)

Autocomplete

- `apm install autocomplete-python`

Autocomplete

- `easy_install jedi`
- `apm install autocomplete-plus-python-jedi`

Linter

- `easy_install flake8`
- `easy_install flake8-docstrings`
- `apm install linter`

- `apm install linter-flake8`

[source](#)

IDLE - Integrated Development Environment

- Python shell
- Better editing
- Limited debugger
- `c:\Python27\Lib\idlelib\idle.bat`
- `C:\Users\Gabor\AppData\Local\Programs\Python\Python35\Lib\idlelib\idle.bat`

sh-bang - executable on Linux/Apple

```
1 #!/usr/bin/env python
2
3 print("Hello World")
```

- The first line starting with `#` is needed if you want to have a file that can be executed without explicitly typing in `python` as well.
- Make your file executable: **`chmod u+x hello_ex.py`**
- Run like: **`./hello_ex.py`**
- In order to run it as **`hello_ex.py`** it needs to be located in one of the directories listed in the **`PATH`** environment variable.

Strings as Comments

marks single line comments.

There are no real multi-line comments in Python, but we can use triple-quotes to create multi-line strings and if they are not part of another statement, they will be disregarded by the Python interpreter. Effectively creating multi-line comments.

```
1 print("hello")
2
3 'A string which is disregarded'
4
5 print(42)
6
7 '''
8     Using three single-quotes on both ends (a triple-
9     quoted string)
10    can be used as a multi-line comment.
11
12 print("world")
```

pydoc

If you really want it, you can also read some of the documentation on the command line, but unless you are locked up some place without Internet connection, I don't recommend this.

Type `pydoc`. On Windows, you might need to create the following file and put it in a directory in your PATH. (see `echo`

%PATH%)

```
1 @python c:\Python27\Lib\pydoc.py %*
```

How can I check if a string can be converted to a number?

There is no `is_int`, we just need to try to convert and catch the exception, if there is one.

```
1 def is_float(val):
2     try:
3         num = float(val)
4     except ValueError:
5         return False
6     return True
7
8 def is_int(val):
9     try:
10        num = int(val)
11    except ValueError:
12        return False
13    return True
14
15 print( is_float("23") )      # True
16 print( is_float("23.2") )    # True
17 print( is_float("23x") )    # False
18 print( '-----' )          # -----
19 print( is_int("23") )       # True
20 print( is_int("23.2") )     # False
21 print( is_int("23x") )     # False
```

Spyder Intro

- iPython console (bottom right)
- Spyder-Py2 / Preferences / Console / Advanced Settings
- Save the file (Ctrl-S / Command-S)
- Run/Run (F5)

- F9 - execute selected text (e.g. we can execute a function definition after we've changed it)
- TAB for autocomplete names of already existing variables.

```
1 print("abc")
2 "abc".           shows the available methods.
3 "abc".center     Command-I will explain what is "center"
```

Interactive Debugging

```
1 def f(a, b):
2     c = a + b
3     d = a * b
4     return c+d
5
6 def run():
7     print(f(2, 3))
8
9     import code
10    code.interact(local=locals())
11
12    print(f(19, 23))
13
14 run()
```

Parameter passing

```
1 def hello(name):
2     msg = name + '!!!!'
3     print('Hello ' + msg)
4
5 hello('Foo')
6 hello('Bar')
```

```
1 Hello Foo!!!!
```

Command line arguments and main

```
1 import sys
2
3 def hello(name):
4     msg = name + '!!!!'
5     print('Hello ' + msg)
6
7 def main():
8     hello(sys.argv[1])
9
10 main()
```

Run as **python argv.py Foo**

Later we'll see the `argparse` module that can handle command line arguments in a better way.

Infinite loop

```
1 i = 0
2 while True:
3     i += 1
4     print(i)
5
6 print("done")
```

break

```
1 i = 0
2 while True:
3     print(i)
4     i += 1
5     if i >= 7:
6         break
7
8 print("done")
```

```
1 0
2 1
```

```
3 2
4 3
5 4
6 5
7 6
8 done
```

continue

```
1 i = 0
2 while True:
3     i += 1
4
5     if i > 3 and i < 8:
6         continue
7
8     if i > 10:
9         break
10    print(i)
```

```
1 1
2 2
3 3
4 8
5 9
6 10
```

While with many conditions

```
1 while (not found_error) and (not found_warning) and
(not found_exit):
2     do_the_real_stuff()
3
4 while True:
5     line = get_next_line()
6
7     if found_error:
8         break
9
10    if found_warning:
```

```
11         break
12
13     if found_exit:
14         break
15
16     do_the_real_stuff()
```

while loop with many conditions

```
1 while True:
2     line = get_next_line()
3
4     if last_line:
5         break
6
7     if line is empty:
8         continue
9
10    if line_has_a_hash: # at the beginning:
11        continue
12
13    if line_has_two_slashes: // at the beginning:
14        continue
15
16    do_the_real_stuff()
```

Format with conversion (stringification with str or repr)

Adding !s or !r in the place-holder we tell it to call the str or repr method of the object, respectively.

- **repr (repr)** Its goal is to be unambiguous
- **str (str)** Its goal is to be readable
- The default implementation of both are useless
- Suggestion
- [Difference between str and repr](#)

```
1 class Point:
2     def __init__(self, a, b):
3         self.x = a
4         self.y = b
5
6 p = Point(2, 3)
7 print(p)                      # <__main__.Point object at
0x10369d750>
8 print("{}".format(p))        # <__main__.Point object at
0x10369d750>
9 print("{}!".format(p))       # <__main__.Point object at
0x10369d750>
10 print("{}!".format(p))      # <__main__.Point object at
0x10369d750>
```

```
1 class Point:
2     def __init__(self, a, b):
3         self.x = a
4         self.y = b
5     def __format__(self, spec):
6         #print(spec) // empty string
7         return "{{'x':{}, 'y':{}}}".format(self.x,
self.y)
8     def __str__(self):
9         return "({},{})".format(self.x, self.y)
10    def __repr__(self):
11        return "Point({}, {})".format(self.x, self.y)
12
13 p = Point(2, 3)
14 print(p)                      # (2,3)
15 print("{}".format(p))        # {'x':2, 'y':3}
16 print("{}!".format(p))       # (2,3)
17 print("{}!".format(p))      # Point(2, 3)
```

Name of the current function in Python

```
1 import inspect
2
3 def first():
4     print(inspect.currentframe().f_code.co_name)
5     print(inspect.stack()[0][3])
```

```
6     second()
7
8
9 def second():
10    print(inspect.currentframe().f_code.co_name)
11    print(inspect.stack() [0] [3])
12
13 def main():
14    first()
15
16 main()
```

Name of the caller function in Python

```
1 import inspect
2
3 def first():
4     print("in first")
5     print("Called by", inspect.stack() [1] [3])
6     second()
7
8 def second():
9     print("in second")
10    print("Called by", inspect.stack() [1] [3])
11
12 def main():
13    first()
14
15 main()
```

Stack trace in Python using inspect

```
1 import inspect
2
3 def first():
4     second()
5
6
7 def second():
8     for info in inspect.stack():
9         #print(info)
```

```
10         #FrameInfo(
11             #     frame=<frame at 0x1c18b18, file
12             #'stack_trace.py', line 9, code second>,
13             #     filename='stack_trace.py',
14             #     lineno=8,
15             #     function='second',
16             #     code_context=[''    for level in
17             inspect.stack():\n'],
18             #     index=0)
19
20
21     #print(info.frame)
22     print(info.filename)
23     print(info.lineno)
24     print(info.function)
25     print(info.code_context)
26     print('')
27
28
29 if __name__ == '__main__':
30     main()
```

```
1 stack_trace.py
2 8
3 second
4 ['    for info in inspect.stack():\n']
5
6 stack_trace.py
7 4
8 first
9 ['    second()\n']
10
11 stack_trace.py
12 26
13 main
14 ['    first()\n']
15
16 stack_trace.py
17 30
18 <module>
19 ['    main()\n']
```

Module Fibonacci

```
1 def fibonacci_number(n):
2     if n==1:
3         return 1
4     if n==2:
5         return 1
6     if n==3:
7         return 5
8
9     return 'unimplemented'
10
11 def fibonacci_list(n):
12     if n == 1:
13         return [1]
14     if n == 2:
15         return [1, 1]
16     if n == 3:
17         return [1, 1, 5]
18     raise Exception('unimplemented')
```

PyTest - assertion

```
1 import mymath
2
3 def test_fibonacci():
4     assert mymath.fibonacci(1) == 1
```

```
1 $ py.test test_fibonacci_ok.py
2 ===== test session starts =====
3 platform darwin -- Python 2.7.5 -- py-1.4.20 -- pytest-
2.5.2
4 collected 1 items
5
6 test_fibonacci_ok.py .
7
8 ===== 1 passed in 0.01 seconds =====
```

PyTest - failure

```
1 import mymath
2
3 def test_fibonacci():
4     assert mymath.fibonacci(1) == 1
5     assert mymath.fibonacci(2) == 1
6     assert mymath.fibonacci(3) == 2
7
8 ===== FAILURES =====
9
10
11    def test_fibonacci():
12        assert mymath.fibonacci(1) == 1
13        assert mymath.fibonacci(2) == 1
14    >       assert mymath.fibonacci(3) == 2
15 E       assert 5 == 2
16 E           +  where 5 = <function fibonacci at
17 E                           0x10a024500>(3)
18 E           +  where <function fibonacci at
19 E                           0x10a024500> = mymath.fibonacci
20
21 test_fibonacci.py:6: AssertionError
22 ===== 1 failed in 0.02 seconds =====
```

PyTest - list

```
1 import fibo
2
```

```
3 def test_fibonacci():
4     assert fibo.fibonacci_number(1) == 1
5     assert fibo.fibonacci_number(2) == 1
6     assert fibo.fibonacci_number(3) == 2
7     assert fibo.fibonacci_number(4) == 2
8
9 def test_fibo():
10    assert fibo.fibonacci_list(1) == [1]
11    assert fibo.fibonacci_list(2) == [1, 1]
12    assert fibo.fibonacci_list(3) == [1, 1, 2]
```

```
1 $ py.test test_fibo.py
2 ===== test session starts
=====
3 platform darwin -- Python 2.7.5 -- py-1.4.20 --
pytest-2.5.2
4 collected 1 items
5
6 test_fibo.py F
7
8 ===== FAILURES
=====
9 _____ test_fibo
=====
10
11     def test_fibo():
12         assert mymath.fibo(1) == [1]
13         assert mymath.fibo(2) == [1, 1]
14 >     assert mymath.fibo(3) == [1, 1, 2]
15 E         assert [1, 1, 5] == [1, 1, 2]
16 E             At index 2 diff: 5 != 2
17
18 test_fibo.py:6: AssertionError
19 ===== 1 failed in 0.01 seconds
=====
```

SAX with coroutine

```
1 import xml.sax
2
3 file = 'examples/xml/data.xml'
4
```

```

5 class EventHandler(xml.sax.ContentHandler):
6     def __init__(self,target):
7         self.target = target
8     def startElement(self,name,attrs):
9         self.target.send(('start',
10 (name,attrs._attrs)))
11     def characters(self,text):
12         self.target.send(('text',text))
13     def endElement(self,name):
14         self.target.send(('end',name))
15
16 def printer():
17     def start(*args,**kwargs):
18         cr = func(*args,**kwargs)
19         cr.next()
20         return cr
21
22     return start
23
24 # example use
25 if __name__ == '__main__':
26     @coroutine
27     def printer():
28         while True:
29             event = (yield)
30             print(event)
31
32     xml.sax.parse(file, EventHandler(printer()))

```

copied from [Stack Overflow](#)
based on [coroutines](#)

```

1 import xml.sax
2
3 file = 'examples/xml/data.xml'
4
5 class EventHandler(xml.sax.ContentHandler):
6     def __init__(self,target):
7         self.target = target
8     def startElement(self,name,attrs):
9         self.target.send(('start',
10 (name,attrs._attrs)))
11     def characters(self,text):

```

```

11         self.target.send(('text', text))
12     def endElement(self, name):
13         self.target.send(('end', name))
14
15     def coroutine(func):
16         def start(*args, **kwargs):
17             cr = func(*args, **kwargs)
18             cr.next()
19             return cr
20         return start
21
22 # example use
23 if __name__ == '__main__':
24     @coroutine
25     def printer():
26         while True:
27             event = (yield)
28             print(event)
29
30     xml.sax.parse(file, EventHandler(printer()))

```

Getting the class name of an object

How to find out which class an object (instance) belongs to?

```

1 import re
2
3 a = 2
4 b = "3"
5 c = 2.3
6
7 m = re.search(r'\d', str(c))
8
9 print(a.__class__)      # <type 'int'>
10 print(b.__class__)     # <type 'str'>
11 print(c.__class__)     # <type 'float'>
12
13 print(type(a))        # <type 'int'>
14 print(type(b))        # <type 'str'>
15 print(type(c))        # <type 'float'>
16
17

```

```
18 print(a.__class__.__name__)      # int
19 print(b.__class__.__name__)      # str
20 print(c.__class__.__name__)      # float
21
22 print(re.__class__.__name__)     # module
23 print(m.__class__.__name__)      # SRE_Match or Match
```

Inheritance - super

We can also call super() passing a different class name

```
1 class Point():
2     def __init__(self, x, y):
3         print('__init__ of point')
4         self.x = x
5         self.y = y
6
7 class Circle(Point):
8     def __init__(self, x, y, r):
9         print('__init__ of circle')
10        super().__init__(x, y)
11        self.r = r
12
13 class Ball(Circle):
14     def __init__(self, x, y, r, z):
15         print('__init__ of ball')
16         #super(Circle, self).__init__(x, y) # r
17         Point.__init__(self, x, y) # r
18         self.z = z
19
20
21 b = Ball(2, 3, 10, 7)
22 print(b)
23
24 # __init__ of ball
25 # __init__ of point
26 # <__main__.Ball object at 0x10a26f190>
```

Inheritance - super - other class

We cannot pass any class name to super()

```
1 class Point:
2     def __init__(self, x, y):
3         print('__init__ of point')
4         self.x = x
5         self.y = y
6
7 class Circle(Point):
8     def __init__(self, x, y, r):
9         print('__init__ of circle')
10        super(Circle, self).__init__(x, y)
11        self.r = r
12
13 class Ball(Circle):
14     def __init__(self, x, y, r, z):
15         print('__init__ of ball')
16         super(Zero, self).__init__(x, y)
17         self.z = z
18
19 class Zero:
20     def __init__(self, x, y):
21         print('really?')
22     pass
23
24
25 b = Ball(2, 3, 10, 7)
26 print(b)
27
28 # __init__ of circle
29 # Traceback (most recent call last):
30 #   File "bad_shapes.py", line 25, in <module>
31 #     b = Ball(2, 3, 10, 7)
32 #   File "bad_shapes.py", line 16, in __init__
33 #     super(Zero, self).__init__(x, y)
34 # TypeError: super(type, obj): obj must be an instance or subtype of type
```

iterator - pairwise

```
1 def pairwise(iterable):
2     "s -> (s0,s1), (s2,s3), (s4, s5), ..."
3     i = 0
4     while i+1 < len(iterable):
5         t = (iterable[i], iterable[i+1])
6         i += 2
7         yield t
8
9 l = [1, 2, 3, 4, 5, 6]
10 for x, y in pairwise(l):
11     print(f"{x} + {y} = {x + y}")
```

iterator - grouped

```
1 def grouped(iterable, n):
2     """s -> (s0,s1,s2,...sn-1),
3             (sn,sn+1,sn+2,...s2n-1),
4             (s2n,s2n+1,s2n+2,...s3n-1), ..."""
5
6     i = 0
7     while i+n-1 < len(iterable):
8         t = tuple(iterable[i:i+n])
9         i += n
10        yield t
11
12 l = [1, 2, 3, 4, 5, 6, 7, 8, 9]
13 for x, y, z in grouped(l, 3):
14     print("{} + {} + {} = {}".format(x, y, z, x + y +
z))
```

```
1 1 + 2 + 3 = 6
2 4 + 5 + 6 = 15
3 7 + 8 + 9 = 24
```

itertools - groupby

Group elements

```
1 from itertools import groupby
2
3 def groupby_even_odd(items):
4     f = lambda x: 'even' if x % 2 == 0 else 'odd'
5     gb = groupby(items, f)
6     print(gb)
7     for k, items in gb:
8         print('{}: {}'.format(k, ', '.join(map(str,
9 items))))
10 groupby_even_odd([1, 3, 4, 5, 6, 8, 9, 11])
```

Circular references

circular references are cleaned up by the garbage collector but maybe not all the memory is given back to the OS, and it can take some time to clean them up.

```
1 import time
2
3
4 def create_pair():
5     a = {'name' : 'Foo'}
6     b = {'name' : 'Bar'}
7     a['pair'] = b
8     b['pair'] = a
9     #print(a)
10
11
12 for i in range(1, 3000000):
13     create_pair()
14
15 print("let's sleep now a bit")
16 time.sleep(20)
```

but weakref might expedite the cleanup. See also the gc module and if I can show it

<http://stackoverflow.com/questions/2428301/should-i-worry-about-circular-references-in-python>

Context managers: with (file) experiments

```
1 with open('out.txt', 'w') as h:  
2     h.write("hello\n")  
3  
4 h = open('out.txt')  
5 print(h.read())
```

```
1 f = open('out.txt', 'w')  
2 f.write("hello\n")  
3 f.close()  
4  
5 # for line in open("myfile.txt"):  
6 #     print line,  
7 # the file is closed only when script ends
```

itertools - izip

Python 3 does not need this any more as the built-in zip is already an iterator.

Combine two unbounded lists

```
1 from itertools import izip, count  
2  
3 for t in izip(count(start=1, step=1), count(start=10,  
step=-1)):  
4     print("{} + {} = {}".format(t[0], t[1],  
t[0]+t[1]))  
5     if t[0] > 20:  
6         break  
7 #   1 + 10 = 11  
8 #   2 + 9 = 11  
9 #   3 + 8 = 11  
10 #  4 + 7 = 11  
11 # ...
```

```
12 # 20 + -9 = 11
13 # 21 + -10 = 11
```

mixing iterators

Combine three unbounded lists

```
1 from itertools import izip, count
2 from my_iterators import fibo, alter
3
4 mixer = izip(count(), fibo(), alter())
5
6 for mix in mixer:
7     print "{:3} {:3} {:3}".format(*mix)
8     if mix[0] >= 8: break
9
10    # 0      1      1
11    # 1      1      -2
12    # 2      2      3
13    # 3      3      -4
14    # 4      5      5
15    # 5      8      -6
16    # 6     13      7
17    # 7     21     -8
18    # 8     34      9
```

mixing iterators

```
1 def fibo():
2     a, b = 1, 1
3     while True:
4         yield a
5         a, b = b, a+b
6
7 def alter():
8     n = 1
9     while True:
10        yield n
11        if n < 0:
12            n -= 1
13        else:
```

```
14             n += 1
15         n *= -1
```

itertools - pairwise

```
1 from itertools import izip
2
3 def pairwise(iterable):
4     "s -> (s0,s1), (s2,s3), (s4, s5), ... "
5     a = iter(iterable)
6     return izip(a, a)
7
8 l = [1, 2, 3, 4, 5, 6, 7]
9 for x, y in pairwise(l):
10    print("{} + {} = {}".format(x, y, x + y))
11
12 # 1 + 2 = 3
13 # 3 + 4 = 7
14 # 5 + 6 = 11
```

Every 2 element from a list. We are using the exact same iterator object in both places of the izip() call, so every time izip() wants to return a tuple, it will fetch two elements from the same iterator.

Iterating over every two elements in a list

itertools - grouped

Every N element from a list

```
1 from itertools import izip
2
3 def grouped(iterable, n):
4     '''s -> (s0,s1,s2,...sn-1),
5            (sn,sn+1,sn+2,...s2n-1),
6            (s2n,s2n+1,s2n+2,...s3n-1), ... '''
7     a = iter(iterable)
```

```
8     iterators = [a] * n
9     return izip(*iterators)
10
11 l = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
12 for x, y, z in grouped(l, 3):
13     print("{} + {} + {} = {}".format(x, y, z, x + y +
z))
14
15 # 1 + 2 + 3 = 6
16 # 4 + 5 + 6 = 15
17 # 7 + 8 + 9 = 24
```

range vs xrange in Python

```
1 from __future__ import print_function
2 import sys
3
4 r = range(1000)
5 x = xrange(1000)
6
7 for v in r:    # 0..999
8     pass
9 for v in x:    # 0..999
10    pass
11
12 print(sys.getsizeof(r))    # 8072
13 print(sys.getsizeof(x))    # 40
```

In Python 2 `range` creates a list of values `range(from, to, step)` and `xrange` creates and iterator.

In Python 3 `range` creates the iterator and if really necessary then `list(range())` can create the list.

range vs. xrange in Python

profile (with hotshot) slow code

It was experimental and dropped from Python 3

- [](<https://docs.python.org/2/library/hotshot.html>)

```
1 import slow
2 import os
3 import hotshot, hotshot.stats
4
5 prof = hotshot.Profile("slow.prof")
6 prof.runcall(slow.main, 1000)
7 prof.close()
8 stats = hotshot.stats.load("slow.prof")
9 stats.strip_dirs()
10 stats.sort_stats('time', 'calls')
11 stats.print_stats(20)
12
13 os.remove("slow.prof")
```

```
1             501501 function calls in 0.337 seconds
2
3     Ordered by: internal time, call count
4
5      ncalls  tottime  percall  cumtime  percall
filename:lineno(function)
6      498501    0.192    0.000    0.192    0.000
slow.py:37 (swap)
7          1    0.136    0.136    0.335    0.335
slow.py:21 (sort)
8      999    0.006    0.000    0.006    0.000
slow.py:4 (f)
9      999    0.002    0.000    0.002    0.000
random.py:173 (randrange)
10     1    0.001    0.001    0.003    0.003
slow.py:31 (get_str)
11     999    0.000    0.000    0.000    0.000
slow.py:10 (g)
12     1    0.000    0.000    0.337    0.337
slow.py:14 (main)
13     0    0.000            0.000
profile:0 (profiler)
```

Abstract Base Class without abc

Only works in Python 2?

```
1 import inspect
2
3 class Base():
4     def __init__(self, *args, **kwargs):
5         if self.__class__.__name__ == 'Base':
6             raise Exception('You are required to
7 subclass the {} class'
8                         .format('Base'))
9
10    methods = set([ x[0] for x in
11                    inspect.getmembers(self.__class__,
12 predicate=inspect.ismethod)])
13    required = set(['foo', 'bar'])
14    if not required.issubset( methods ):
15        missing = required - methods
16        raise Exception("Required method '{}' is
17 not implemented in '{}'"
18                         .format(', '.join(missing),
19 self.__class__.__name__))
20
21
22 class Real(Base):
23     def foo(self):
24         print('foo in Real')
25     def bar(self):
26         print('bar in Real')
27     def other(self):
28         pass
29
30
31 class Fake(Base):
32     # user can hide the __init__ method of the parent
33     # class:
34     #     def __init__(self):
35     #         pass
36     def foo(self):
37         print('foo in Fake')
38
39
40 r = Real()
41 #b = Base()  # You are required to subclass the Base
42 class
```

```
35 #f = Fake()  # Required method 'bar' is not
 implemented in class 'Fake'
```

Abstract Base Class with abc Python 2 ?

```
1 from abc import ABCMeta, abstractmethod
2
3 #class Base(metaclass = ABCMet):
4 class Base():
5     __metaclass__ = ABCMeta
6
7     @abstractmethod
8     def foo(self):
9         pass
10
11    @abstractmethod
12    def bar(self):
13        pass
14
15
16 class Real(Base):
17     def foo(self):
18         print('foo in Real')
19     def bar(self):
20         print('bar in Real')
21     def other(self):
22         pass
23
24 class Fake(Base):
25     def foo(self):
26         print('foo in Fake')
27
28 r = Real()
29 f = Fake()
30     # TypeError: Can't instantiate abstract class Fake
 with abstract methods bar
```

- [Abstract Base Classes in Python](#)
- [abc](#)

Abstract Base Class with metaclass

```

34
35         class_object.__init__ = my_init
36
37
38 class Base(object):
39     __metaclass__ = MyABC
40     __required_methods__ = ['foo', 'bar']
41
42 # b = Base() # Exception: You are required to subclass
the 'Base' class
43
44 class Real(Base):
45     def foo():
46         pass
47     def bar():
48         pass
49
50 r = Real()
51
52 class Fake(Base):
53     def foo():
54         pass
55
56 #f = Fake() # Exception: Required method 'bar' is not
implemented in class 'Fake'
57
58 class UnFake(Fake):
59     def bar():
60         pass
61
62 uf = UnFake()

```

Create class with metaclass

```

1 class M(type):
2     pass
3
4 class A(object):
5     pass
6
7 class B(object):
8     __metaclass__ = M

```

```

9
10 a = A()
11 print(type(a))
12 b = B()
13 print(type(b))
14
15
16
17 class Meta(type):
18     def __init__(self, *args, **kwargs):
19         print('Meta.__init__')
20         print(self) # <class '__main__.C'>
21         print(args) # ('C', (<type 'object'>,),
22                # {'__module__': '__main__',
23                # '__metaclass__': <class
24                '__main__.Meta'>})
25         print(kwargs) # {}
26
27
28
29 c = C()
30 print(type(c))
31
32 class MyABC(type):
33     def __init__(self, *args):
34         print('Meta.__init__')
35         print(args) # ('C', (<type 'object'>,),
36                # {'__module__': '__main__',
37                # '__metaclass__': <class
38                '__main__.Meta'>})
39
40     __metaclass__ = MyABC

```

```

1 # http://stackoverflow.com/questions/100003/what-is-a-
2 # metaclass-in-python
3
4 # Create a new-style class
5 class A(object):
6     pass
7 print(type(A))          # <type 'type'>
8 a = A()

```

```
8 print(type(a))          # <class '__main__.A'>
9
10 B = type('B', (), {})
11 print(type(B))          # <type 'type'>
12 b = B()
13 print(type(b))          # <class '__main__.B'>
14
15 # old style
16 class C():
17     pass
18 print(type(C))          # <type 'classobj'>
19 c = C()
20 print(type(c))          # <type 'instance'>
21
22 # Have attributes in the class
23 class AA(object):
24     name = 'Foo'
25 print(AA.name)          # Foo
26 aa = AA()
27 print(aa.name)          # Foo
28
29
30 BB = type('BB', (), {'name' : 'Bar'})
31 print(BB.name)          # Bar
32 bb = BB()
33 print(bb.name)          # Bar
34
35
36 # Inherit from a class
37 class AAA(AA):
38     pass
39 print(AAA.name)          # Foo
40 aaa = AAA()
41 print(aaa.name)          # Foo
42
43 BBB = type('BBB', (BB,), {})
44 print(BB.name)          # Bar
45 bbb = BBB()
46 print(bbb.name)          # Bar
47
48
49 def f(self):
50     print(self.name)
51
52 class AAAA(object):
```

```
53     name = 'AAAA-Foo'
54     def show(self):
55         print(self.name)
56
57 aaaa = AAAA()
58 aaaa.show() # AAAA-Foo
59
60 BBBB = type('BBBB', (), {'name': 'BBBB-Bar', 'show' :
f})
61 bbbb = BBBB()
62 bbbb.show() # BBBB-Bar
```

- [what is a metaclass](#)

Python Descriptors

A more manual way to implement the `property()` functionality we have just seen.

Use cases:

- Implement type-checking and/or value checking for attribute setters ()
- [Descriptors](#)
- [Descriptor HowTo Guide](#)

alter iterator

Is this interesting at all ?

```
1 from my_iterators import alter
2
3 for a in alter():
4     print(a)
5     if a >= 6:
6         break
7
8 # 1
```

```
9 # -2
10 # 3
11 # -4
12 # 5
13 # -6
14 # 7
```

Create a counter queue

```
1 import threading
2 import Queue
3
4 class ThreadedCount(threading.Thread):
5     def __init__(self, name, start, stop):
6         threading.Thread.__init__(self)
7         self.name = name
8         self.counter = start
9         self.limit = stop
10    def run(self):
11        while self.counter < self.limit:
12            self.counter += 1
13            print(self.name, self.counter)
14
15        print(self.name, "finished")
16        return
17
18 queue = Queue()
19 foo = ThreadedCount("Foo", 1, 10)
20 bar = ThreadedCount("Bar", 1, 10)
21 foo.start()
22 bar.start()
23 print("main - running")
24
25 foo.join()
26 bar.join()
27 print("main - thread is done")
```

A Queue of tasks

```
1 from queue import Queue
2 from threading import Thread
```

```

3
4 def source():
5     """Returning the list of tasks"""
6     return range(1, 10)
7
8 def do_work(item):
9     print("Working on item " + str(item) + "\n",
end="")

10 # print("Working on item ", str(item))
11 # would show the output intermingled as the separate
12 # items of the print statement
13 # (even the trailing newline) might be printed only
14 # after context switch
15
16
17 def worker():
18     while True:
19         item = q.get()
20         do_work(item)
21         q.task_done()

22 def main():
23     for i in range(num_worker_threads):
24         t = Thread(target=worker)
25         t.daemon = True
26         t.start()

27     for item in source():
28         q.put(item)

29     q.join()          # block until all tasks are done
30
31
32 num_worker_threads = 3
33 q = Queue()
34 main()

```

Filtered Fibonacci with ifilter

```

1 from series import fibonacci
2 from itertools import ifilter
3
4 even = ifilter(lambda f: f % 2 == 0, fibonacci())
5 for e in even:

```

```
6     print(e)
7     if e > 200:
8         break
```

Python from .NET

TODO and add to dotnet

TODO: example with async call in .NET getting back to python