Python Advanced

Program

Day 1

Object-oriented programming
Classes, methods and attributes
Magic methods
Inheritance

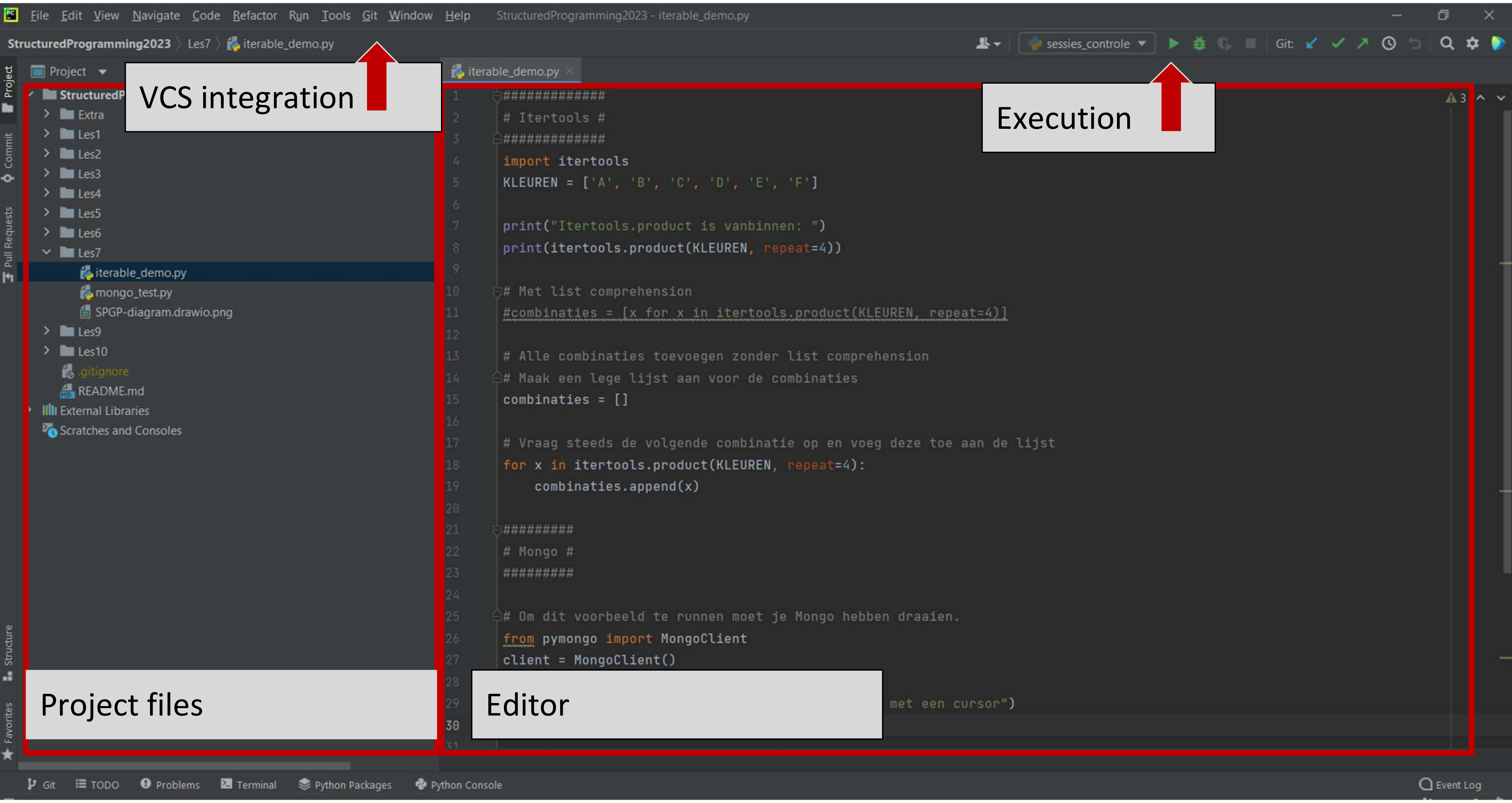
Day 2

Python Standard Library
Python Package Index

Recap

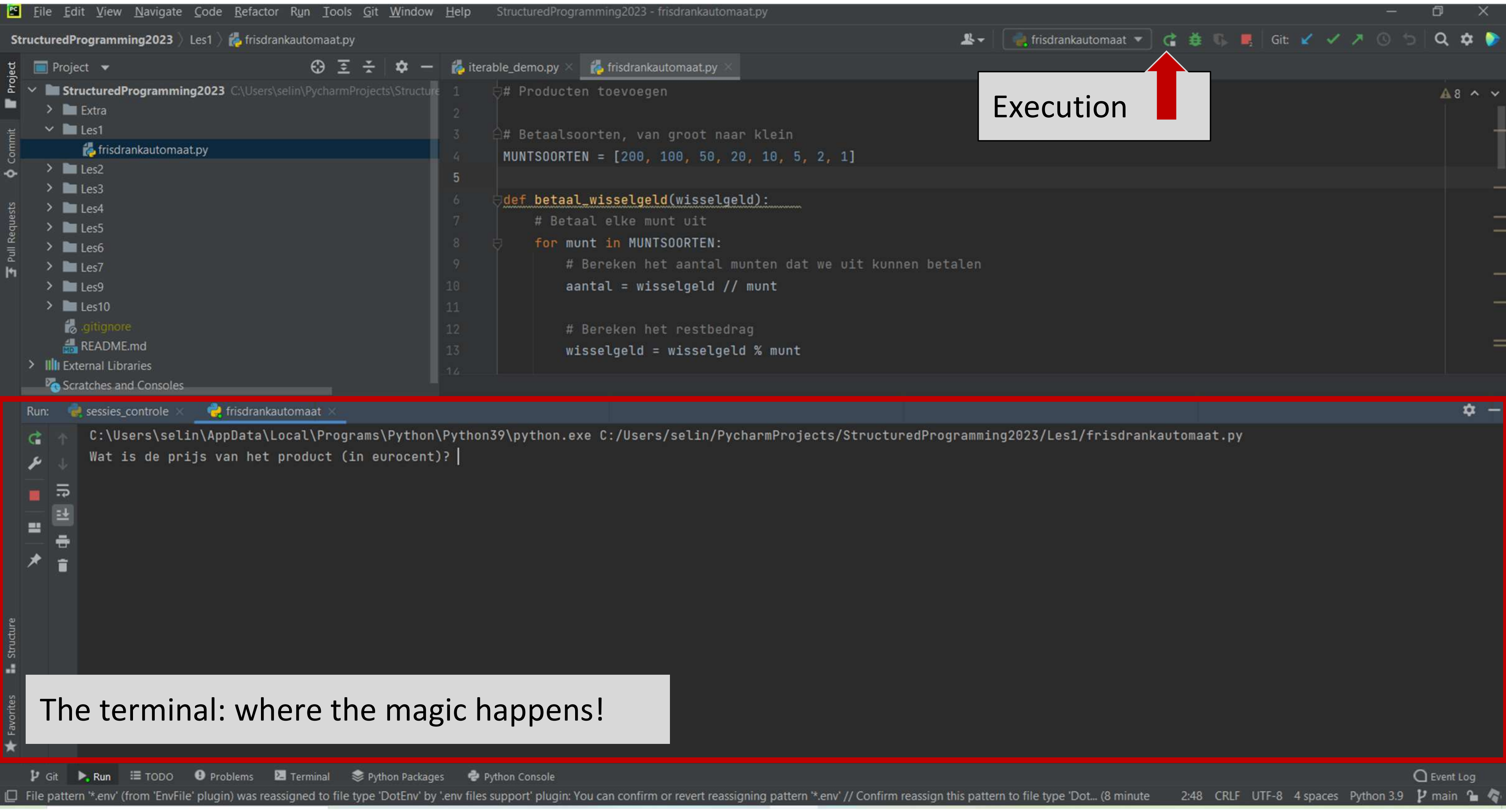
- Python prompt
- Executing a Python module
- Variables
- Operators
- Built-in functions
- Strings
- Conditional statements
- Loop statements
- Data structures – Lists, Sets, Dicts
- Comprehension
- Functions – definition, arguments, return
- Generators
- Reading and writing files
- Exceptions

PyCharm



```python
#############
# Itertools #
#############
import itertools
KLEUREN = ['A', 'B', 'C', 'D', 'E', 'F']

print("Itertools.product is vanbinnen: ")
print(itertools.product(KLEUREN, repeat=4))

# Met list comprehension
#combinaties = [x for x in itertools.product(KLEUREN, repeat=4)]

# Alle combinaties toevoegen zonder list comprehension
# Maak een lege lijst aan voor de combinaties
combinaties = []

# Vraag steeds de volgende combinatie op en voeg deze toe aan de lijst
for x in itertools.product(KLEUREN, repeat=4):
    combinaties.append(x)

#########
# Mongo #
#########

# Om dit voorbeeld te runnen moet je Mongo hebben draaien.
from pymongo import MongoClient
client = MongoClient()

                                                met een cursor")
```

PyCharm



```python
# Producten toevoegen

# Betaalsoorten, van groot naar klein
MUNTSOORTEN = [200, 100, 50, 20, 10, 5, 2, 1]

def betaal_wisselgeld(wisselgeld):
    # Betaal elke munt uit
    for munt in MUNTSOORTEN:
        # Bereken het aantal munten dat we uit kunnen betalen
        aantal = wisselgeld // munt

        # Bereken het restbedrag
        wisselgeld = wisselgeld % munt
```

```
C:\Users\selin\AppData\Local\Programs\Python\Python39\python.exe C:/Users/selin/PycharmProjects/StructuredProgramming2023/Les1/frisdrankautomaat.py
Wat is de prijs van het product (in eurocent)?
```

File pattern '*.env' (from 'EnvFile' plugin) was reassigned to file type 'DotEnv' by '.env files support' plugin: You can confirm or revert reassigning pattern '*.env' // Confirm reassign this pattern to file type 'Dot... (8 minute 2:48 CRLF UTF-8 4 spaces Python 3.9 main

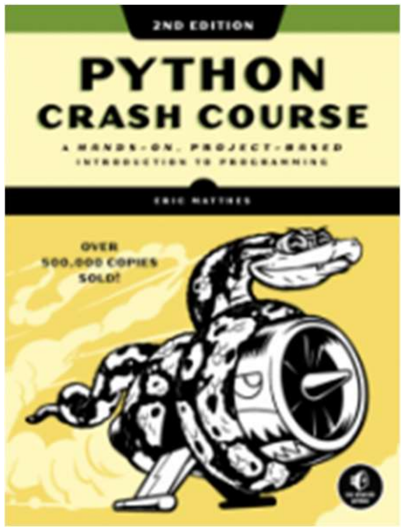
Book

Part I: Basics

1. Getting started
2. Variables and Simple Data Types
3. Introducing Lists
4. Working with Lists
5. If Statements
6. Dictionaries
7. User input and While Loops
8. Functions
9. Classes
10. Files and Exceptions
11. Testing Your Code

Part II: Projects

12. Project 1: Alien Invasion
13. Project 2: Data Visualization
14. Project 3: Web Applications

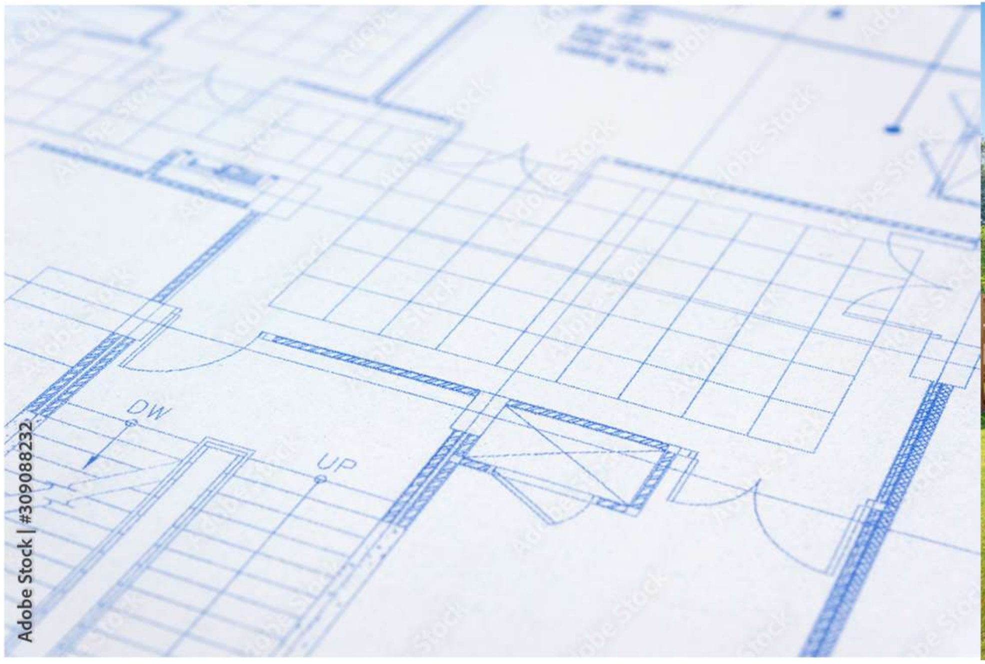

2ND EDITION
PYTHON
CRASH COURSE
A HANDS-ON, PROJECT-BASED INTRODUCTION TO PROGRAMMING
ERIC MATTHES
OVER 500,000 COPIES SOLD!

Resources:
https://ehmatthes.github.io/pcc_2e/regular_index/

Object-Oriented Programming

Class
?

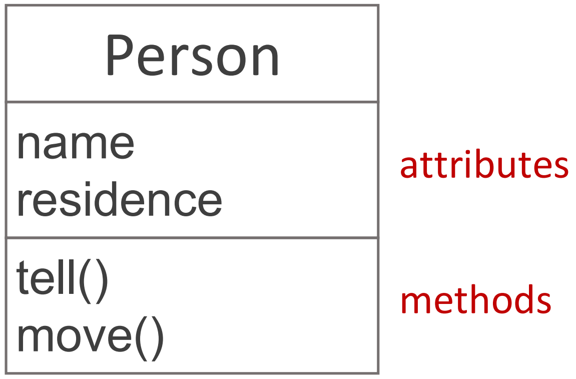
Objects
?



Object-Oriented Programming

Class

Objects



Object-Oriented Programming

instantiation

Class
blueprint

Person
name residence

tell()
move()

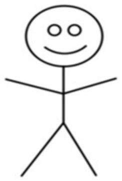
attributes

methods

Objects
instances

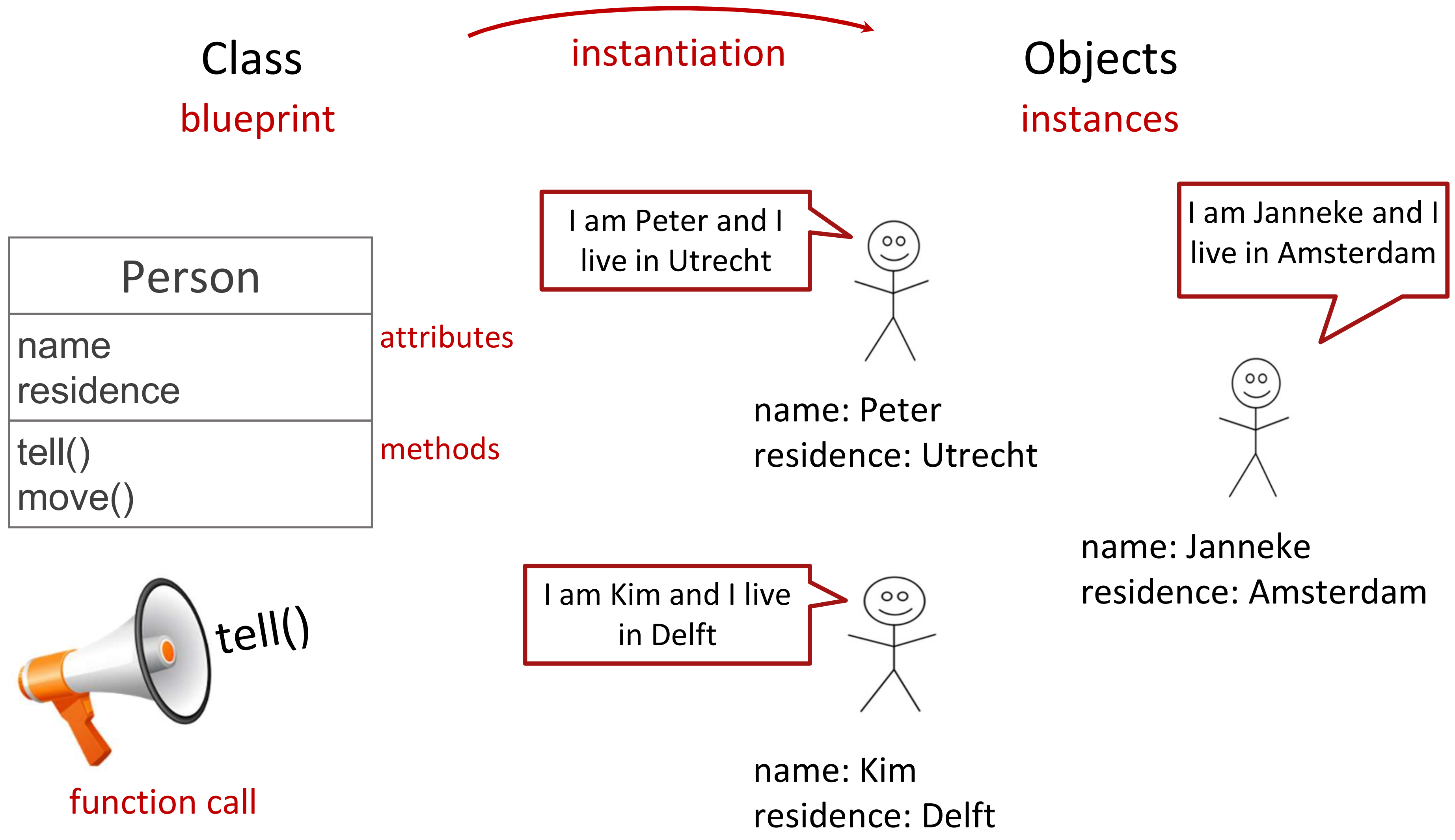
name: Peter
residence: Utrecht

name: Janneke
residence: Amsterdam



name: Kim
residence: Delft

Object-Oriented Programming



Object-Oriented Programming

Class → Objects

blueprint — instantiation → instances

str

upper()
lower()
split()
strip()
join()
title()
capitalize()
replace()

attributes

methods

'something' → 'SOMETHING'

'something else' → 'SOMETHING ELSE'

'hello world' → 'HELLO WORLD'

function call

Object-Oriented Programming

Class

blueprint

instantiation

Objects

instances

list

append()
insert()
extend()
pop()
sort()

attributes

methods

[2, 1, 3] → [1, 2, 3]

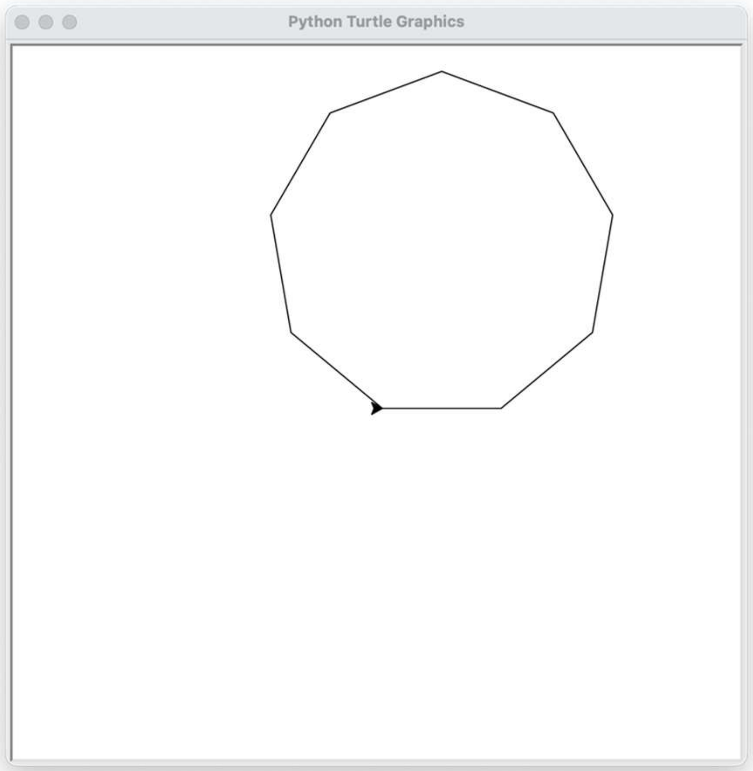
[‘a’, ‘c’, ‘b’] → [‘a’, ‘b’, ‘c’]

function call

Exercise: Turtle

Draw a polygon with turtle.

- Check out the different methods that you can use with turtle!
  - in particular: **forward** and **left**
- Import the turtle library
- Draw a polygon
  - Calculate the angle of each corner
  - Hint: https://en.wikipedia.org/wiki/Regular_polygon
- Draw a square
- End the program with turtle.done()



Classes

- First define a class with the keyword **class**
- Instantiate an object with the class
- Set the state of the object by assigning values to the attributes
- Call the methods of the object
- Use the object operator **.** (a dot) to access attributes and methods

```
class Person:
    pass

# -------------------------------

p = Person()
p.name = 'Albert'
p.residence = 'Amsterdam'
```

Methods

- Methods are functions within a class. Methods can have arguments and a return statement just like normal functions.
- The first argument is automatically set to the 'target' object. This is typically called **self** and refers to the object itself.
- You can access the methods of a class using the object operator, which is the dot.

```
class Person:
    def tell(self):
        return f'I am f{self.name}'

# ------------------------------------------------------------

p = Person()
p.name = 'Albert'
print(p.tell())
```

Object initialization

- When an object in created (instantiated) from a class the **__init__** method is automatically called
- **__init__** is called a **magic method**. They are also called **dunder** methods for their double underscores.
- There are so many more magic methods!

```
class Person:
    def __init__(self, name):
        self.name = name
    def tell(self):
        return(f'I am {self.name}')

# ------------------------------------------------------------

p = Person('Albert')
print( p.tell() )
```

Public or not

- An attribute can be indicated as **non-public** by adding _ as a prefix to the name of the attribute.
- This is more of a a guideline "We are all adults and know what we're doing."
- You can add a double underscore __ to obstify de name of the attribute outside of the class. This is to prevent naming collissons when inheriting classes.

```
class Person:
    def __init__(self, name):
        self._name = name
    def tell(self):
        return(f'I am {self._name}')

# ------------------------------------------------------------

p = Person('Albert')
print( p.tell() )
```

Attributes

- Attributes are typically initialized in the **__init__** method
- Attributes are dynamic and can be assigned a value anywhere
- But this is not always the intention! (We’ll get back to that)

```
class Person:
    def __init__(self, name, residence = 'unknown'):
        self.__name = name
        self.__residence = residence

    def tell(self):
        return(f'I am {self.__name} from {self.__residence}')

# ------------------------------------------------------------

p = Person('Albert', 'Amsterdam')
print(p.tell())
```

Exercise: Bank account

Create a BankAccount class, then create several BankAccount instantiations and demonstrate that you can deposit and withdraw an amount to the account.

Tips:

- Create a class BankAccount first
- Add attributes in the **__init__** method. Attributes should be **balance** and **holder**.
- Add the methods: **deposit** and **withdraw** that take an amount (in euros) argument and a third method **info** that returns information about the account.
- Instantiate several BankAccount objects and write some code to demonstrate the capabilities of the class!

Exercise: Class Car

- Create a class named **Car**
- Add the __init__ method and set several attributes like **make**, **type** and **color**
- Set the **mileage** attribute to 0
- Create a method **info** that describes the car and the mileage
- Create a method **drive** that takes an amount of kilometres and adds that to the mileage.
- Test your class by instantiating a car and calling the methods
- What happens when you drive a negative distance?

Class-wide attributes

- Class wide attributes are attributes that are related to the class instead of to an object of that class.
- Class wide attributes can be accessed by all objects of the class.
- They do not use underscores.

```
class Mathematics:
    pi = 3.14159
    e = 2.71828

print( Mathematics.pi )
print( Mathematics.e )
```

Class-wide methods

- Class-wide methods are methods related to the class.
- A method can be indicated as a class-wide method with a decorator **@classmethod** or **@staticmethod**.

```
class Mathematics:
    @staticmethod
    def power1(x, n):
        result = 1
        for _ in range(n):
            result *= x
        return result

    @classmethod
    def power2(cls, x, n):
        return cls.power1(x, n)

print(Mathematics.power1(2, 4))
print(Mathematics.power2(2, 4))
```

Example

```
class Person:

     __slots__ = ('__name', '__residence')

    def __init__(self, name, residence = 'unknown'):
        self.__name = name
        self.__residence = residence

    def tell(self):
        return('I am {} and I live in {}'\
               .format(self.__name, self.__residence))

    def move(self, new_residence):
        self.residence = new_residence


p = Person('Albert', 'Amsterdam')
print(p.tell())
p.move('Eindhoven')
print(p.tell())
```

Magic Methods/Dunders

- A class can have many different special methods.
- Also called special methods.
- A magic method is called by Python in all kind of situations, typically when operators & casting are used

__init__	Objects	__eq__	Operators
__del__		__ne__	
		__lt__	
__str__	Casting	__le__	
__repr__		__gt__	
__int__		__ge__	
__float__		__add__	
		__sub__	

Exercise: Vector class

Create a 2D-Vector class. Also add operator overloading for the + sign to add two vectors together and to create a string representation.

Tips:

- Build a class called Vector
- Add two attributes: x and y
- Implement the **__init__** method that takes two arguments: x and y
- Implement the **__str__** method.
- Implement the **__add__** method the define the adding of two vectors.
- Test your class by creating two vectors and adding these together.

Inheritance

- Classes and functionality can be reused by using **inheritance**
- The original class is called the parent class, the superclass or the base class
- The new class is called the child class, the subclass or the derived class
- Enclose the parent in parentheses after the new class name
- All the attributes and methods of the parent class are available in the child class
- In the **__init__** method of the child class we always call the **__init__** method of the parent class with the **super** method

```
class Vector(object):
```

```
class ChildClass(ParentClass):
    def __init__(self, name):
        super().__init__(name)
```

Exercise: Shapes

Create 3 classes, a parent class **shape** and two child classes **circle** and **square**

Tips:

- Implement the **__init__** method that takes the argument w and initialize attributes **perimeter** and **surface**
- Implement the **__str__** method.
- Implement the method **calc_perimeter** and **calc_surface** that calculates those.
- Implement the **__eq__** and **__lt__** methods to compare two shapes

Python Standard Library

- The Python Standard Library consists of more than 200 modules and packages
- The Python has "batteries included"

os	csv	json	subprocess
os.path	collections	xml	socket
sys	array	sqlite3	asyncio
string	decimal	zipfile	urllib
re	fractions	time	http
math	statistics	argparse	tkinter
random	pathlib	logging	doctest
datetime	pickle	threading	unittest
calendar	shelve	multiprocessing	timeit

sys - System-specific

- System-specific parameters and functions

version
version_info
path
argv
exit
stdin / stdout / stderr

```
import sys

# get Python version
print(sys.version)

# add directory to sys.path
sys.path.append(r'c:\pythondev')
```

sys

- Get the current version of Python
- Return the message you are currently running Python version …

os - Operating system interfaces

- The **os.path** module provides a portable way of using operating system dependent functionality.

rename
remove
mkdir
makedirs
chdir
getcwd
rmdir
listdir

```
import os

# set current working directory
os.chdir(r'c:\pythondev')
print(os.getcwd())
```

exec

pathlib - Object-oriented filesystem paths

This module offers classes representing filesystem paths with semantics appropriate for different operating systems.

Path
PurePath
WindowsPath
PureWindowsPath

operator /

```
from pathlib import Path

p = Path('.')

list(p.glob('**/*.py'))
```

shutil

High-level file operations

- **copy**
- **copytree**
- **rmtree**
- **move**
- **disk_usage**
- **chown**

glob

- The glob module finds all the pathnames matching a specified pattern according to the rules used by the Unix shell.

```
import glob
glob.glob('./file[0-9].*')
```

subprocess

- The subprocess module allows you to spawn new processes, connect to their input/output/error pipes, and obtain their return codes.

run

```
import subprocess

subprocess.run(["ls", "-l"])

subprocess.run(["ls", "-l", "/dev/null"], capture_output=True)
```

tempfile

This module generates temporary files and directories.

It works on all supported platforms.

- **TemporaryFile**
- **NamedTemporaryFile**
- **TemporaryDirectory**

```
import tempfile

with tempfile.TemporaryFile() as fp:
    fp.write(b'Hello world!')
    fp.seek(0)
    fp.read()
```

OS

- Use the os library to get the contents of a directory in a list.

datetime - Basic date and time types

- The datetime module supplies classes for manipulating dates and times.

class datetime.date
class datetime.time
class datetime.datetime
class datetime.timedelta
class datetime.tzinfo
class datetime.timezone

strftime
strptime

```
from datetime import datetime, date
datetime.strptime(s)
d = date(2020, 2, 28)

print(d.strftime('%Y-%m-%d'))
```

datetime

- input a date and print the date in another format

string - Common string operations

- String methods

count	isnumeric	split
find	join	strip
format	lower	title
index	replace	

```
# remove vowels
s = 'an example text'
vowels = "aeiou"
trans = str.maketrans("", "", vowels)
result = s.translate(trans)
```

re - Regular expression operations

- Online: https://www.regex101.com/#python

```
search(pattern, string, flags)
match(pattern, string, flags)
findall(pattern, string, flags)
sub(pattern, repl, string, max=0, flags)
compile(pattern)
```

```
import re
match = re.search(r'@([\w\.]+)\b', 'albert@gmail.com', re.I)
if match:
   for group in match.groups():
      print('Domain: ', group)
```

re

- Go to the website https://rubular.com
- Build and test a regular expression to match an e-mail address
- Use the same regular expression in python with the search method in the re library

math - Mathematical functions

pi	cosh	gcd	log2
e	degrees	hypot	modf
	erf	inf	nan
acos	erfc	isclose	pow
acosh	exp	isfinite	radians
asin	expm1	isinf	remainder
asinh	fabs	isnan	sin
atan	factorial	ldexp	sinh
atan2	floor	lgamma	sqrt
atanh	fmod	log	tan
ceil	frexp	log10	tanh
copysign	fsum	log1p	tau
cos	gamma		trunc

random - Pseudo-random numbers

- Generate pseudo-random numbers

```
random.seed()
random.randrange(start, stop)
random.randint(a, b)
random.choice(sequence)
random.choices(sequence, k=1)
random.shuffle(sequence)
random.sample(sequence, n)
random.random()
```

```
import random
items = 'abcdefghiklmnopqrstuwvxyz0123456789'
sample = random.sample(items, 3)
```

json - JavaScript Object Notation

- JSON encoder and decoder

```
json.dump(object, file)
json.dumps(object)
json.load(file)
json.loads(string)
```

```
import json

s = json.dumps([1,2,3,{'4': 5, '6': 7}])

with open('bestand.json', 'w') as f:
json.dump([1,2,3,{'4': 5, '6': 7}], f)

json.loads('[1,2,3,{"4":5,"6":7}]')
```

pickle - Python object serialization

- A **shelf** is a persistent, dictionary-like object that stores any arbitrary Python that can be pickled.

pickle.dump(object, file)

pickle.dumps(object)

pickle.load(file)

pickle.loads(string)

```
import pickle

class User:
    def saveToPickle(self):
        with open('user.pickle','wb') as f:
            pickle.dump(self, f)
    def loadFromPickle(self):
        with open('user.pickle','rb') as f:
            self.__dict__.update(pickle.load(f).__dict__)
    @classmethod
    def createFromPickle(cls):
        with open('user.pickle','rb') as f:
            return pickle.load(f)
```

pickle

- Create a datastructure and store this in a pickle file. Create a second python script that reads pickle file and restores the data in the data structure.

xml

- The **xml.etree.ElementTree** module implements a simple and efficient API for parsing and creating XML data.
- This module provides limited support for **XPath** expressions for locating elements in a tree. https://www.w3schools.com/xml/xpath_intro.asp
- ElementTree provides a simple way to build XML documents and write them to files.

```python
import xml.etree.ElementTree as ET

tree = ET.parse('data.xml')
root = tree.getroot()

print(root.attrib)

for element in root.findall('//name'):
    print(element.text)
```

xml

- Read the Macbeth xml file and generate several overviews.
    - The name of the play
    - The names of all the personas
    - The names of all the scenes

```python
import xml.etree.ElementTree as ET

tree = ET.parse('data.xml')
root = tree.getroot()

print(root.attrib)

for element in root.findall('//name'):
    print(element.text)
```

statistics

- This module provides functions for calculating mathematical statistics of numeric (Real-valued) data.

bisect_left	median	pstdev
bisect_right	median_grouped	pvariance
collections	median_high	stdev
groupby	median_low	variance
harmonic_mean	mode	
mean	numbers	

```
import statistics
numbers = [23, 64, 86, 23, 54, 76, 98, 21]
print('Median:', statistics.median(numbers))
print('Mean:', statistics.mean(numbers))
print('St.Dev.:', statistics.stdev(numbers))
```

Statistics

Create a function that calculates and returns the mean, median and mode of a list of numbers.

Tips:

- Define a function as **def central_measures(numbers)**
- Calculate the measures:
  - The **mean** is the sum of the values divided by the number of values
  - The **median** is middle value of the sorted list of values
  - The **mode** is the most frequently occuring value
- Return the measures as a tuple with **return mean, median, mode**
- Call the function with a list of arbitrary numbers
- Print the result

doctest

- The doctest module searches for pieces of text that look like interactive Python sessions, and then executes those sessions to verify that they work exactly as shown.

```
def square(n):
    """Calculate the square of n.

    >>> [square(n) for n in range(6)]
    [0, 1, 4, 9, 16, 25]
    """
    return n ** 2


if __name__ == "__main__":
    import doctest
    doctest.testmod(verbose=True)
```

doctest

- Create a function and add a docstring with doctests for the function.

unittest - Unit testing framework

- There is also an **assert** statement

```
import unittest


class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())


if __name__ == '__main__':
    unittest.main()
```

assertEqual(a, b)
assertNotEqual(a, b)
assertTrue(x)
assertFalse(x)
assertIs(a, b)
assertIsNot(a, b)
assertIsNone(x)
assertIsNotNone(x)
assertIn(a, b)
assertNotIn(a, b)
assertIsInstance(a, b)
assertNotIsInstance(a, b)

csv – Comma Seperated Values

- CSV File Reading and Writing

reader
writer
DictReader
DictWriter

```
import csv

filename = 'data.csv'

with open(filename) as f:
    reader = csv.DictReader(f, delimiter=';')
    for row in reader:
        print(row['first_name'], row['last_name'])
```

decimal

- The decimal module provides support for fast correctly-rounded decimal floating point arithmetic.

```python
from decimal import Decimal

d1 = Decimal('0.1')
d2 = Decimal('0.2')

result = float(d1 + d2)
```

fractions

- The fractions module provides support for rational number arithmetic.

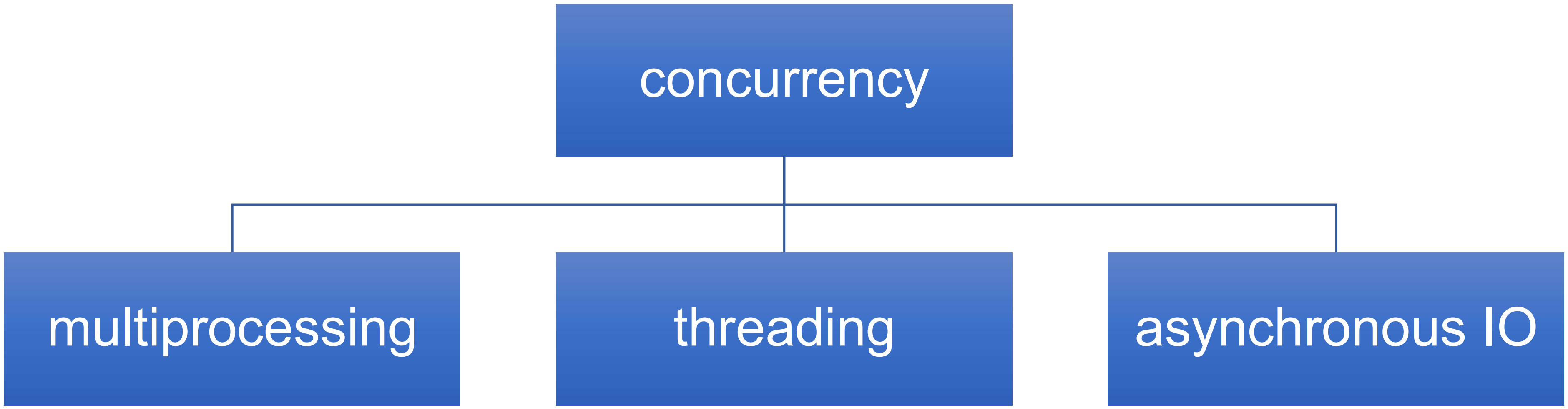
```
from fractions import Fraction

d1 = Fraction(1, 3) # => 1/3
d2 = Fraction(1, 2) # => 1/2

result = d1 + d2 # => 5/6
```

sqlite3

- DB-API 2.0 interface for SQLite databases
- PEP 249 - Database API Specification 2.0

```
import sqlite3

conn = sqlite3.connect('example.db')
c = conn.cursor()

c.execute("""CREATE TABLE stocks
        (date text, trans text, symbol text, qty real, price real)""")

c.execute("""INSERT INTO stocks
        VALUES ('2006-01-05’, 'BUY’, 'RHAT’, 100, 35.14)""")

conn.commit()

for row in c.execute('SELECT * FROM stocks ORDER BY price'):
    print(row)

conn.close()
```

Concurrency



Python Standard Library:

- **multiprocessing** package
- **threading** package
- **asyncio** package and **async/await** keywords (introduced in Python 3.4)

Comparison

	Multiprocessing	Threading	Asynchronous IO
Package	multiprocessing	threading	asyncio
Class	Proces	Thread	Coroutine
Python	Class Proces	Class Thread	Keywords async, await
Data sharing	Message	Shared data	
Usage	CPU intensive	IO intensive	IO intensive

Proces versus Thread

- True parallelism in Python is achieved by creating multiple processes, each having a Python interpreter with its own separate GIL.

Process	**Thread**
processes run in separate memory (process isolation)	threads share memory
uses more memory	uses less memory
children can become zombies	no zombies possible
more overhead	less overhead
slower to create and destroy	faster to create and destroy
easier to code and debug	can become harder to code and debug

Python GIL

- A global interpreter lock (GIL) is a mechanism used in Python interpreter to synchronize the execution of threads so that only one native thread can execute at a time, even if run on a multi-core processor.
- The C extensions, such as numpy, can manually release the GIL to speed up computations. Also, the GIL released before potentially blocking I/O operations.
- Note that both Jython and IronPython do not have the GIL.

threading - Thread-based parallelism

- **Thread**
  - start
  - run
  - join
  - name
- **active_count**
- **current_thread**
- **main_thread**

```
import time
from threading import Thread

def myfunc(i):
    print "sleeping 5 sec from thread %d" % i
    time.sleep(5)
    print "finished sleeping from thread %d" % i

for i in range(10):
    t = Thread(target=myfunc, args=(i,))
    t.start()
```

asyncio

- At the heart of async IO are **coroutines**. A coroutine is a specialized version of a Python generator function.

```
import asyncio

async def count():
    print("One")
    await asyncio.sleep(1)
    print("Two")

async def main():
    await asyncio.gather(count(), count(), count())

if __name__ == "__main__":
    import time
    s = time.perf_counter()
    asyncio.run(main())
    elapsed = time.perf_counter() - s
    print(f"{__file__} executed in {elapsed:0.2f} seconds.")
```

multiprocessing

- The multiprocessing library is based on spawning Processes.
- A process starts a fresh Python interpreter thereby side-stepping the Global Interpreter Lock
- The multiprocessing module allows the programmer to fully leverage multiple processors on a given machine.

```
from multiprocessing import Process

def f(name):
    print('hello', name)

if __name__ == '__main__':
    p = Process(target=f, args=('bob',))
    p.start()
    p.join()
```

logging - Logging facility for Python

- Setup with **basicConfig**
- Logging Levels: DEBUG, INFO, WARNING, ERROR, CRITICAL

```
import logging

logging.basicConfig(
    filename = None, # or to a file 'example.log',
    level = logging.ERROR,
    format = '%(asctime)s.%(msecs)03d - %(message)s',
    datefmt = '%Y-%m-%dT%H:%M:%S')

logging.debug('This message should go to the log file')
logging.info('So should this')
logging.warning('And this, too')
logging.error('Watch out!')
logging.critical('ERROR!!!!!')
```

timeit

- Measure execution time of small code snippets.

```python
from timeit import timeit

timeit('"-".join(str(n) for n in range(100))', number=10000)
timeit(lambda: "-".join(map(str, range(100))), number=10000)
```

zipfile

- The ZIP file format is a common archive and compression standard. This module provides tools to create, read, write, append, and list a ZIP file.

```
import zipfile
import pandas as pd

with zipfile.ZipFile("FinalExam.zip") as z:
    with z.open("AdvWorksCusts.csv") as f:
        df_Customers = pd.read_csv(f)

    with z.open("AW_AveMonthSpend.csv") as f:
        df_AveMonthSpend = pd.read_csv(f)

    with z.open("AW_BikeBuyer.csv") as f:
        df_BikeBuyer = pd.read_csv(f)
```

tarfile

- The tarfile module makes it possible to read and write tar archives, including those using gzip, bz2 and lzma compression.

```
import tarfile

t = tarfile.open('example.tar.gz', 'r')
print("Files in TAR file:")
print(t.getnames())
```

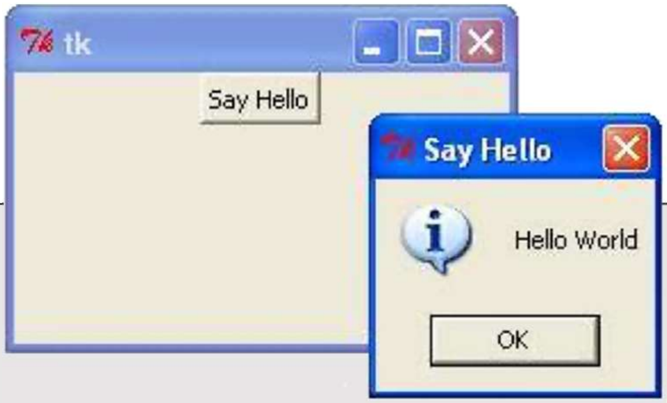
GUI Frameworks

- **TkInter** - The traditional Python user interface toolkit.

- **PyQt** - Bindings for the cross-platform Qt framework.
- **PySide** - PySide is a newer binding to the Qt toolkit
- **wxPython** - a cross-platform GUI toolkit that is built around wxWidgets
- **Win32Api** - native window dialogs
- **PyMsgBox**

tkinter

- The tkinter package (“Tk interface”) is the standard Python interface to the Tk GUI toolkit.
- There are also Standard Dialogs



```
import tkinter as tk
import tkMessageBox

top = tk.Tk()

def hello():
   tkMessageBox.showinfo("Say Hello", "Hello World")

btn1 = tk.Button(top, text = "Say Hello", command = hello)
btn1.pack()

top.mainloop()
```

The Python Package Index - PyPI

- The official third-party software repository for the Python programming language
- The Python Package Index is a repository of software for the Python programming language. There are currently > **300000** packages.
- Install packages with the **pip** command.

```
pip list
pip search

pip install numpy
pip install scipy
pip install matplotlib
pip install pandas
pip install requests
pip install pyodbc
```

Virtual Environment

- Seperated enviroments

```
$ virtualenv -p python3.5 venv
$ . venv/bin/activate
```

- Requirements file

```
$ pip list > requirements.txt
$ pip install -r requirements.txt
```

pyodbc - Accessing ODBC databases

```
import pyodbc

conn = pyodbc.connect(
    'DRIVER={SQL Server};'
    'SERVER=localhost\SQLEXPRESS;'
    'DATABASE=mijndatabase;'
    'UID=username; PWD=pa55w0rd')

sql = 'SELECT customers.* FROM customers'

cursor = conn.cursor()
for row in cursor.execute(sql):
    print("{}, {}".format(row.name, row.residence)

cursor.close()
conn.close()
```

PEP 249 -- Python Database API Specification v2.0

numpy

- NumPy is the fundamental package for scientific computing with Python.
- NumPy's main object is the homogeneous multidimensional array.
- Vectorized operations

```
import numpy as np

a = np.array([1,2,3,4])
b = np.array( [ (1.5,2,3), (4,5,6) ] )
c = np.narray( [ [1,2], [3,4] ], dtype=complex )

np.zeros( (3,4) )
np.arange( 0, 2, 0.4 )   # array([ 0., 0.4, 0.8, 1.2, 1.6, 2.0])
np.linspace( 0, 2*pi, 100 ).  # 100 numbers from 0 to 2*pi
```

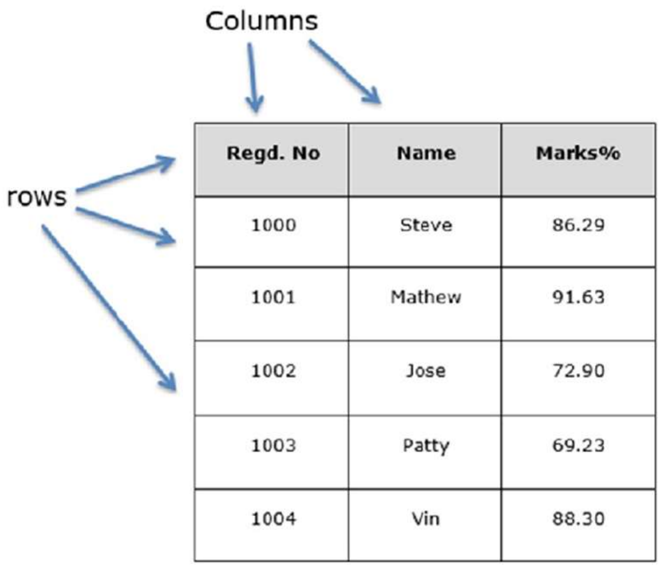
scipy

- It provides many user-friendly and efficient numerical routines, such as routines for numerical integration, interpolation, optimization, linear algebra, and statistics.

Clustering
Constants
Discrete Fourier transforms
Integration
Interpolation
Input and output
Linear algebra
Miscellaneous routines
Multi-dimensional image processing
Orthogonal distance regression
Optimization and Root Finding
Signal processing
Sparse matrices
Sparse linear algebra
Compressed Sparse Graph Routines
Spatial algorithms and data structures
Special functions
Statistical functions
Statistical functions for masked arrays
Low-level callback functions

pandas

- Pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool.
- **Series** is a one-dimensional labeled array capable of holding any data type
- **DataFrame** is a 2-dimensional labeled data structure with columns of potentially different types.



Regd. No	Name	Marks%
1000	Steve	86.29
1001	Mathew	91.63
1002	Jose	72.90
1003	Patty	69.23
1004	Vin	88.30

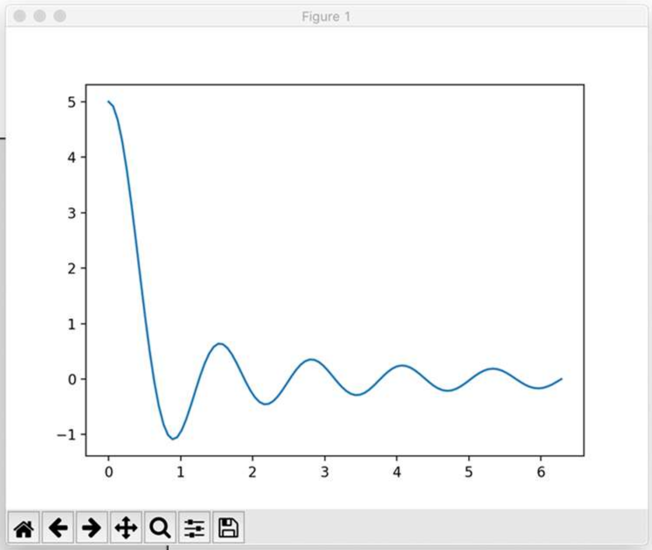
https://pandas.pydata.org/docs/getting_started/10min.html#min

matplotlib

- Matplotlib is a Python 2D plotting library which produces publication quality figures

Line plot	**3D plot**	**Polar plot**
Histogram	**Image plot**	
Scatter plot	**Contour plot**	

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace( 0.000001, 2*np.pi, 100 )
y = 1/x * np.sin(5*x)

plt.plot(x, y)
plt.show()
```



Matplotlib

- Write a Python program to draw a line with suitable label in the x axis, y axis and a title.
- Write a Python program to display the grid and draw line charts of the closing value of Alphabet Inc. between October 3, 2016 to October 7, 2016. Customized the grid lines with linestyle -, width .5. and color blue:

```
Date,Close
03-10-16,772.559998
04-10-16,776.429993
05-10-16,776.469971
06-10-16,776.859985
07-10-16,775.080017
```

Get the weather in New York

Use requests to query openweathermap.org for the weather in a specified city.

Tips:

- import **requests**
- build the url (see https://openweathermap.org/current)
  use: **appid=d1526a9039658a6f76950cff21823aff**
- use the following code to get the response:
  response = requests.get(url)
- use json to decode the response into a Python dictionary
  response.json()
- get and print the temperature

requests – HTTP for Humans

- **Requests** is an elegant and simple HTTP library for Python, built for human beings.

```
import requests

url = "http://api.openweathermap.org/data/2.5/weather"
url += "?appid=d1526a9039658a6f76950cff21823aff"
url += "&units=metric"
url += "&mode=json"
url += "&q=New York"

response = requests.get(url)
if (response.status_code == 200):
    body = response.text
    decoded = response.json()
    temperature = decoded['main']['temp']
else:
    print("Error for city %s" % (city))
```

django

MVC Framework

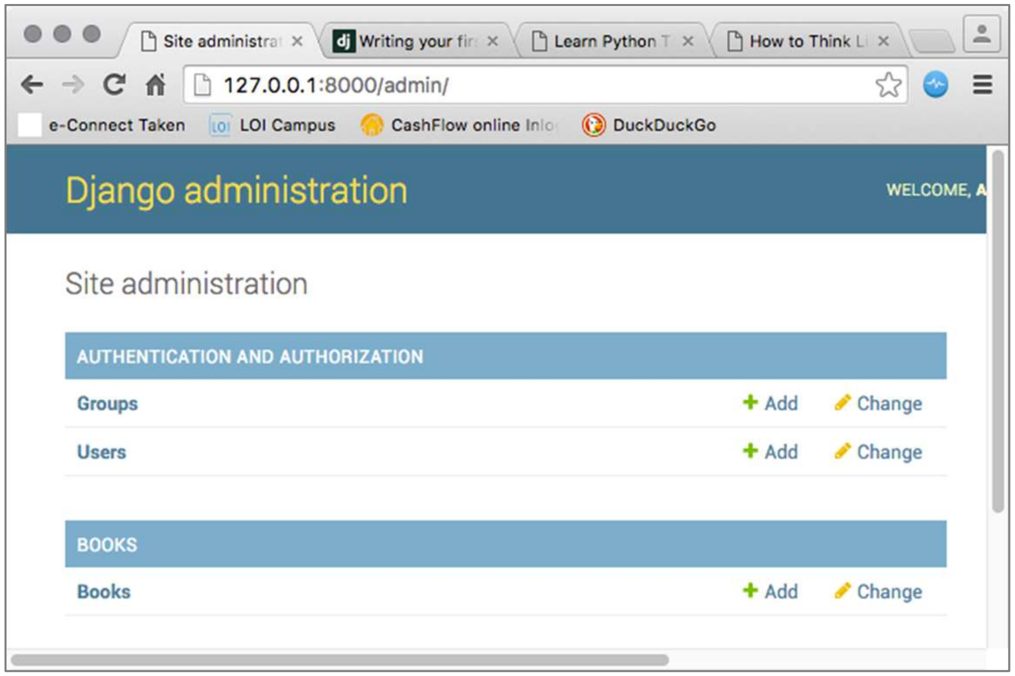
Models

Object-Relational Mapping

URL Mapping

Views

HTML Templates



Django administration WELCOME, A

Site administration

AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	Change
Users	+ Add	Change

BOOKS		
Books	+ Add	Change

Command line bootstrap:

```
$ mkdir django-demo
$ cd django-demo
$ virtualenv -p python3.5 venv
$ . venv/bin/activate
$ pip install django
$ django-admin
$ django-admin startproject demo
$ python manage.py createsuperuser
$ python manage.py startapp books
$ python manage.py makemigrations
$ python manage.py migrate
$ python manage.py runserver
```

Python Distributions

- Anaconda
- Active Python
- Python (X,Y)
- IPython
- Enthought Canopy
- Sage
- PyPy
- Pocket Python
- Portable Python

Implementations

- CPython reference implementation
- IronPython (Python running on .NET)
- Jython (Python running on the Java Virtual Machine)
- PyPy (A fast python implementation with a JIT compiler)
- Stackless Python (Branch of CPython supporting microthreads)
- MicroPython (Python running on micro controllers)

Style Guide for Python Code

- PEP 8 - Style Guide for Python Code

Ducktyping

- “If it looks like a duck and quacks like a duck, it must be a duck.”

- A programming style which does not look at an object’s type to determine if it has the right interface; instead, the method or attribute is simply called or used

EAFP versus LBYL

- EAFP: "it's easier to ask for forgiveness than permission
- LBYL: "look before you leap"