
computrain
IT & Digital Skills
Python Expert
python™
www.computrain.nl


computrain
IT & Digital Skills
Introduction
www.computrain.nl

Program

Python Advanced Concepts

Higher Order Functions - Decorators

Advanced Classes

Project Setup

Unittesting

File Formats: Pickle, XML, JSON

Database Access

ORM, SqlAlchemy

GUI: tkinter

Web Frameworks: Django, Flask

RESTful API: FastAPI

Webscraping: Requests, BeautifulSoup

Concurrency

Topics

- pythonic / idiomatic python
- OO - associations
- virtual environments
- type hinting
- logging
- zipfile
- pylint
- docstrings
- pydoc
- unit testing
- File I/O
- concurrency
- GUI
- configparser
- git & github
- functools
- itertools
- pathlib
- shutil
- dateutil , calendar , arrow
- database access
- web applications
- executables

Python Basics

- Built-in numeric types: bool, int, float, complex
- Operators: arthimetic, comparative, boolean
- Strings: concatenation, string methods, formating, unicode
- Flow statements: while, for, if-elif-else
- Functions : arguments, default values, variadic arguments, keyword arguments, return value, lambda
- Built-in functions
- Reading from and writing to files, context manager
- Lists: creation, indexing, slicing, modification, unpacking, comprehesion
- Dicts : creation, modification, indexing, iterating
- Sets: creation, modification, set methods and operators
- Object Orientation : classes, objects, methods and attributes

Advanced Python

www.computrain.nl

Pythonic / Idiomatic Python

- PEP 8 – Style Guide for Python Code https://peps.python.org/pep -0008/

Pylint

- Pylint is a **static code analyser** for Python

- Pylint analyses your code without actually running it. It checks for errors, enforces a coding standard, looks for code smells, and can make suggestions about how the code could be refactored.

- It can also be integrated in most editors or IDEs.

- pylint my_script.py

Pip

```
Usage:
  pip <command> [options]

Commands:
  install                     Install packages.
  download                    Download packages.
  uninstall                   Uninstall packages.
  freeze                      Output installed packages in requirements format.
  inspect                     Inspect the python environment.
  list                        List installed packages.
  show                        Show information about installed packages.
  check                       Verify installed packages have compatible dependencies.
  config                      Manage local and global configuration.
  search                      Search PyPI for packages.
  cache                       Inspect and manage pip's wheel cache.
  index                       Inspect information available from package indexes.
  wheel                       Build wheels from your requirements.
  hash                        Compute hashes of package archives.
  completion                  A helper command used for command completion.
  debug                       Show information useful for debugging.
  help                        Show help for commands.
```

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

Virtual Environments

Python virtual environments aim to provide a lightweight , isolated Python environment

- venv

```
$ python -m venv venv

$ venv\Scripts\activate            # windows
$ source venv/bin/activate         # macos

$ deactivate
```

- virtualenv
  - The package is a superset of `venv`, which allows you to do everything that you can do using `venv`, and more

Conda Environments

- Conda Package and Environment Manager

- If you're working in the data science space and with Python alongside other data science projects , then conda is an excellent choice that works across platforms and languages .

- conda

```
$ conda create –n new_environment
$ conda activate new_environment
$ conda deactivate
```

Poetry

- PYTHON PACKAGING AND DEPENDENCY MANAGEMENT
- Poetry is a tool for dependency management and packaging in Python. It allows you to declare the libraries your project depends on and it will manage (install /update) them for you . Poetry offers a lockfile to ensure repeatable installs , and can build your project for distribution .

- poetry commands

```
$ poetry new <my-package>
$ poetry init
$ poetry install
$ poetry update
$ poetry add
$ poetry remove
$ poetry show
$ poetry build
$ builld publish
```

Project structure

- https://docs.python -guide.org/writing/structure/

```python
"""Script to ..."""

import sys

def main():
  """Main entry point for the script."""
  pass

if __name__ == '__main__':
  sys.exit(main())
```

```
helloworld/
│
├── bin/
│
├── docs/
│   ├── hello.md
│   └── world.md
│
├── helloworld/
│   ├── __init__.py
│   ├── runner.py
│   ├── hello/
│   │   ├── __init__.py
│   │   ├── hello.py
│   │   └── helpers.py
│   │
│   └── world/
│       ├── __init__.py
│       ├── helpers.py
│       └── world.py
│
├── data/
│   └── input.csv
│
├── tests/
.   ├── hello
│   │   ├── helpers_tests.py
│   │   └── hello_tests.py
│   │
│   └── world/
│       ├── helpers_tests.py
│       └── world_tests.py
│
├── .gitignore
├── LICENSE
├── README.md
├── requirements.txt
├── setup.py
└── venv/
```

Logging

- import **logging**
- 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!!!!!')
```

Docstrings

- Best practices
  - All modules, classes, methods , and functions , including the __init__ constructor in packages should have docstrings .
  - Descriptions are capitalized and have end-of-sentence punctuation .
  - Always use """Triple double quotes.""" around docstrings .

- Conventions for multi-line docstrings
  - Sphinx style
  - Google style
  - Numpy style

```
"""[Summary]

:param [ParamName]: [ParamDescription], defaults to [DefaultVal]
:raises [ErrorType]: [ErrorDescription]
:return: [ReturnDescription]
"""
```

pydoc

- Documentation generator and online help system

- The built-in function help() invokes the online help system in the interactive interpreter, which uses pydoc to generate its documentation as text on the console.

- In order to find objects and their documentation, pydoc imports the module(s) to be documented. Therefore, any code on module level will be executed on that occasion. Use an `if __name__ == '__main__':` guard to only execute code when a file is invoked as a script and not just imported.

```
$ pydoc sys
```

Project

- Build a project structure for the project.
- Move the implemented class to a module in de src/ models directory.
- Make sure to add the if __name__ == '__main __:' line.

Decorator

- Build a "debug" decorator
- The decorator should print the name of the function , the provided arguments , resulting return value and the time it took to execute the function .
- Optionally store the results in a log file with the logging library

TkInter calculator

- Build a simple calculator with TkInter
- The calculator should have
  - two input fields (Entry widgets)
  - a third entry field to display the result
  - a button the triggers a function that adds the values entered in the input fields and displays the result in the result field
  - optionally add more buttons to als subtract or multiple the values
  - optionally add a message field (Label widget) to display information during the calculation

Flask List

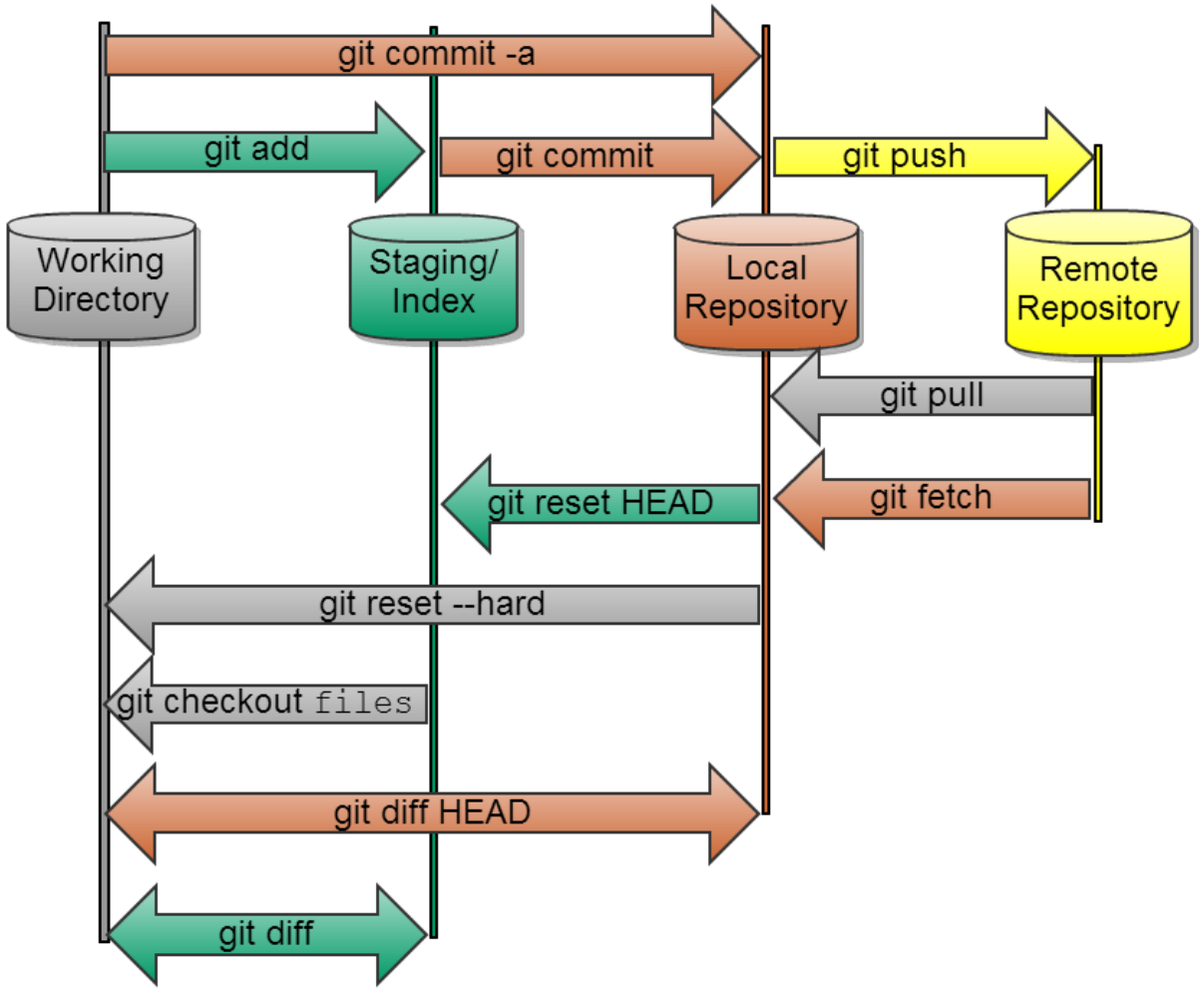
- Build a simple flask application that can display the contents of a dictionary as a html table .
- First create a flask application
- Then create a dictionary containing information.
- Create a HTML base template
- Create a HTML template to display a table based on dictionary key value pairs
- Create an endpoint mapping function to display your page
- In this function render the table template and pass your dictionary as data to the template.

GIT

- Git is a free and open source distributed version control system

```
$ git config --global user.name "My Name"
$ git config --global user.email "me@gmail.com"

$ git init demo
$ git clone URL

$ git add  --all

$ git commit -m "Commit Message"

$ git status

$ git push
$ git pull
```

GIT Workflow & Commands



Configparser

- This module provides the ConfigParser class which implements a basic configuration language which provides a structure similar to what's found in Microsoft Windows INI files.

```
import configparser
config = configparser.ConfigParser()
config['DEFAULT'] = {'ServerAliveInterval': '45',
                     'Compression': 'yes',
                     'CompressionLevel': '9'}
config['bitbucket.org'] = {}
config['bitbucket.org']['User'] = 'hg'
config['topsecret.server.com'] = {}
topsecret = config['topsecret.server.com']
topsecret['Port'] = '50022' # mutates the parser
topsecret['ForwardX11'] = 'no' # same here
config['DEFAULT']['ForwardX11'] = 'yes'
with open('example.ini', 'w') as configfile:
    config.write(configfile)
```

```
[DEFAULT]
ServerAliveInterval = 45
Compression = yes
CompressionLevel = 9
ForwardX11 = yes
[bitbucket.org]
User = hg
[topsecret.server.com]
Port = 50022
ForwardX11 = no
```

Import

- import in a script absolute imports
- vs
- import in a package relative imports

```
try:
    from .encryption import Encryption

except ImportError:
    from encryption import Encryption
```

https://realpython.com/python-import

Type Hinting

- Type hinting is a formal solution to statically indicate the type of a value within your Python code. It was specified in **PEP 484** and introduced in **Python 3.5** .

- Python's type hints provide you with optional static typing to leverage the best of both static and dynamic typing.

```
def headline(text: str, align: bool = True) -> str:
    if align:
        return f"{text.title()}\n{'-' * len(text)}"
    else:
        return f" {text.title()} ".center(50, "o")
```

Variadic arguments

- * args Unspecified positional arguments are collected in a list
- ** kwargs Unspecified keyword arguments are collected in a dict

- Use a \ to specified that all previous arguments must be positional arguments
- Use a * to specified that all following arguments must be keyword arguments

- This is frequently used to pass arguments on to a function that is called from within the function .

Functions are First Class - Citizens

- Functions are first -class citizens :
  - functions can be passed around and used as arguments just like any other object.
  - functions can be defined inside other functions . Such functions are called inner functions .
  - functions can be returned as return values .

Advanced Functions

- functional programming : map, filter, sorted , reduce
- higher order functions
- closure
- partial
- decorators

Functional Programming

- Pure functions

- No side -effects
  - no printing
  - no globals

- Built -in functions
  - map
  - filter
  - sorted
  - reduce (in functools)

- Lambda

Closures

- A closure is a function defined within an other function .
- The inner function retains the variables during definition .

```
def make_counter():
    count = 0 # Enclosing variable
    def counter():
        nonlocal count
        count += 1
        return count
    return counter

# Using the closure
counter = make_counter()

print(counter()) # Output: 1
print(counter()) # Output: 2
print(counter()) # Output: 3
```

Partial

- A partial function is a new function created by fixing some portion of the arguments of an existing function
- functools.partial

```
from functools import partial

# Define a function that takes three arguments
def multiply(x, y, z):
    return x * y * z

# Create a partial function with y = 2 and z = 3
partial_multiply = partial(multiply, y=2, z=3)

# Now we only need to pass one argument (x)
result = partial_multiply(4) # equivalent to multiply(4, 2, 3)
```

Decorators

- A decorator is a function that takes another function and extends the behavior of the latter function without explicitly modifying it.
- Use @ to wrap a function in a decorator

```
def my_decorator(f):
    def wrapper(*args, **kwargs):
        # do something before running the function
        return_value = f(*args, **kwargs)
        # do something after running the function
        return return_value
    return wrapper

@my_decorator
def another_function():
    print('Hello')

another_function()
```

Generator expression

- Generator expression (x**2 for x in range(100))

```
# list comprehension
doubles = [2 * n for n in range(50)]

# generator expression
doubles = (2 * n for n in range(50))

# same as the list comprehension above
doubles = list(2 * n for n in range(50))
```

Generator functions

- The keyword **yield** specifies a generator function
- When the yield keyword is hit the function returns a result
- The next time the function is called the function continues where it left off

```
import random

def random_order1(numbers):
    random.shuffle(numbers)
    for number in numbers:
        yield number

def random_order2(numbers):
    random.shuffle(numbers)
        yield from numbers
```


computrain
IT & Digital Skills
Object Oriented Programming
www.computrain.nl

Object Oriented Programming

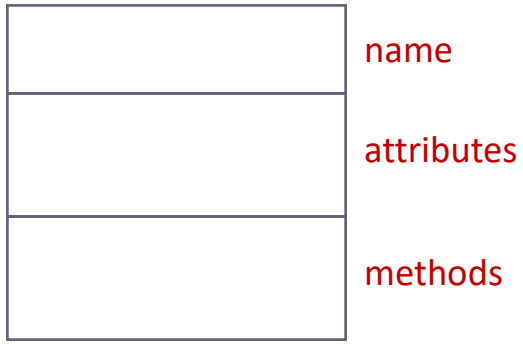
Class

Objects

Object Oriented Programming

Class

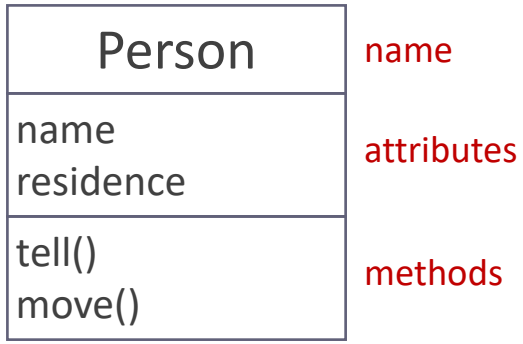
Objects



Object Oriented Programming

Class

Objects



Object Oriented Programming

Class

Objects

Person	name
name residence	attributes
tell() move()	methods

name: Peter
residence: Utrecht

name: Janneke
residence: Amsterdam

naam: Kim
residence: Delft

instances

Object Oriented Programming

Class

instantiation

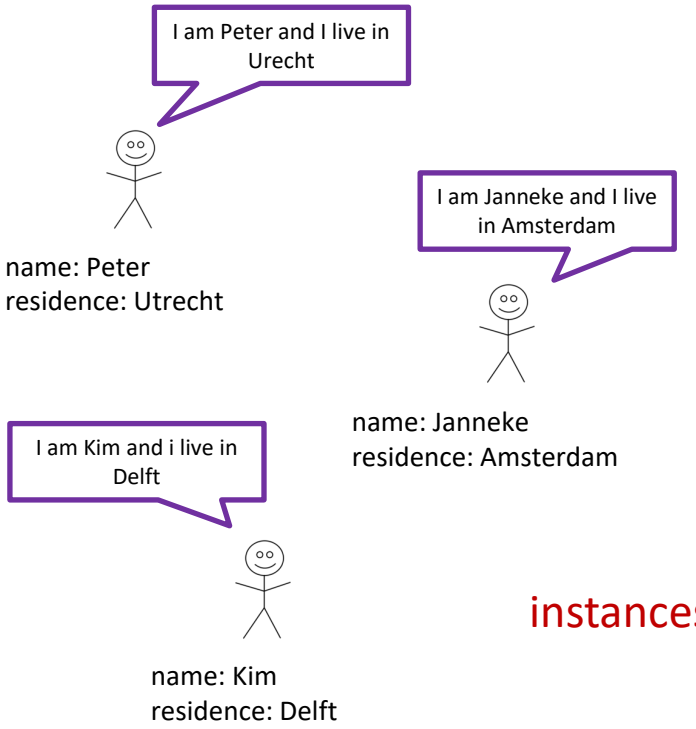
Objects

Person
name residence
tell() move()

name

attributes

methods



instances

Object Oriented Programming

Class

instantiation

Objects

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

name

attributes

methods

'something'

'something else'

'hello world'

instances

Object Oriented Programming

Class

Objects

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

name

attributes

methods

[1, 2, 3]

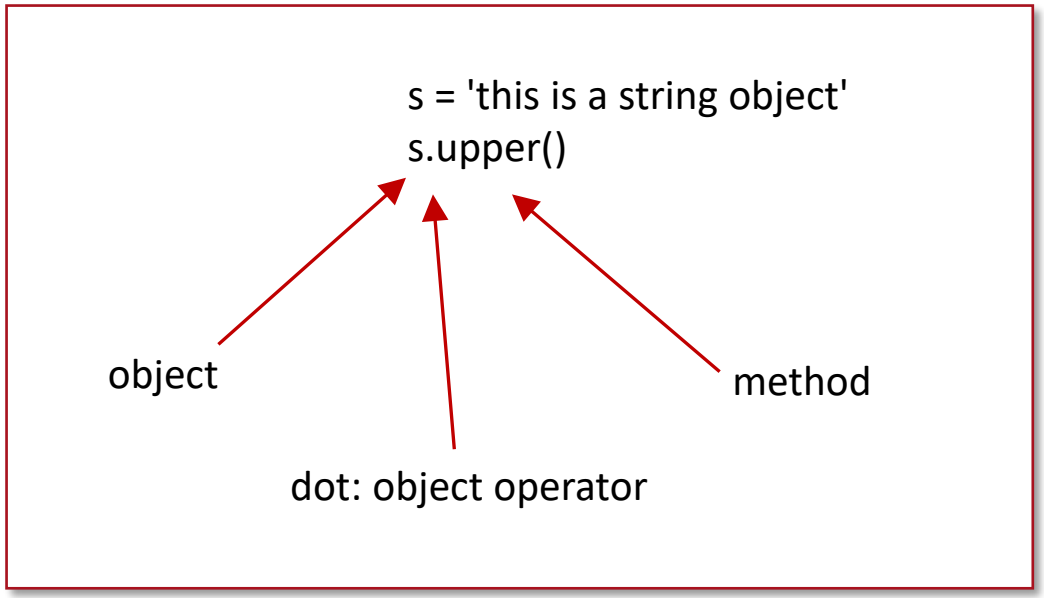
['something', 'something else']

instances

Object Oriëntation

- Classes
  - Attributes
  - Methods

- Object Instantiation



Example

```
class Person:

                __slots__ = ('_name', '_residence')

    def __init__(self, name, residence = 'unknown'):
        self._name = name
        self._residence = residence

    def tell(self):
        return(f'I am {self._name} and I live in {self._residence})

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

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

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

Fractions

- Create a Fraction class to work with fractions.
- What are the attributes?
- Which methods are relevant?
- Implement the class and use magic methods to implement operations like addition and multiplication.

Project

During this course we will build an application and iteratively add functionality.
The project is to build a shopping cart. Articles can be added to the cart and the total price can be calculated. We store data in a database, manage the database with a graphical user interface en provide a web interface for users to add items to the shopping cart.

- The first step is to implement a class (model) for the shopping cart.
- Implement several appropriate attributes and useful methods.

Special Methods

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

__init__	__eq__	__gt__
__del__	__ne__	__ge__
__str__	__lt__	__add__
__repr__	__le__	__sub__

Classes

- Decorators
  - @classmethod
  - @staticmethod
  - @property
  - @<property>.setter
  - @dataclass

- Inheritance
  - super()
  - multiple inheritance , MRO

Class-wide methods

- Class-wide methods are methods related to the class and not to an instance.
- A method can be indicated as a class-wide methode 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))
```

DataClass

- The **dataclass()** decorator examines the class to find fields. A field is defined as a class variable that has a type annotation.
- The **dataclass()** decorator will add various “dunder” methods such as __init__() and __repr__() to the class.
- New in version 3.7.

```
from dataclasses import dataclass

@dataclass
class Item:
   """Class for keeping track of an item in inventory."""
   name: str
   unit_price: float
   quantity_on_hand: int = 0

   def total_cost(self) -> float:
      return self.unit_price * self.quantity_on_hand
```

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)
```

Project

- Create a number of unittests for the class implemented earlier .

File I/O

- Context Manager **with**

read	readlines	seek
readline	write	close

```
keyword = 'xxx'
filename = 'data.txt'

with open(filename) as f:
    for line in f:
        line = line.strip()
        if keyword in line:
            print(line)
```

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)
```

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.

```
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)
```

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**

- **Dash**
- **Jupyter Widgets**

tkinter

- The tkinter package (“ Tk interface”) is the standard Python interface to the Tk GUI toolkit .


```
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()
```

tkinter

- Widgets
  - Frame
  - Button
  - Label
  - Entry
  - Text
  - Radio Button
  - Check Button
  - Combobox
  - Menu
  - Menu Button
  - Scale
  - Scrollbar
  - List Box
  - Canvas
  - …

- Layout Managers
  - pack
  - grid
  - place

- Dialogs
  - askfloat ()
  - askinteger ()
  - askstring ()
  - askopenfile ()
  - askopenfiles ()
  - asksaveasfile ()
  - askopenfilename ()
  - askopenfilenames ()
  - asksaveasfilename ()
  - askdirectory ()

Project

- Create a GUI for the class implemented earlier .

itertools

- The following functions all construct and return iterators.

count	pairwise
cycle	starmap
repeat	takewhile
	tee
accumulate	zip_longest
batched	
chain	product
compress	permutations
dropwhile	combinations
filterfalse	combinations_with_replacement
groupby	
islice	

Functools

- Higher-order functions and operations on callable objects

cache	singledispatch
cached_property	singledispatchmethod
cmp_to_key	update_wrapper
lru_cache	wraps
total_ordering	
partial	
partialmethod	
reduce	

Pathlib

- Object-oriented filesystem paths

```
from pathlib import Path

p = Path('.')

[x for x in p.iterdir() if x.is_dir()]

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

Shutil

- High -level file operations

copy	move
copytree	disk_usage
rmtree	chown

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
```

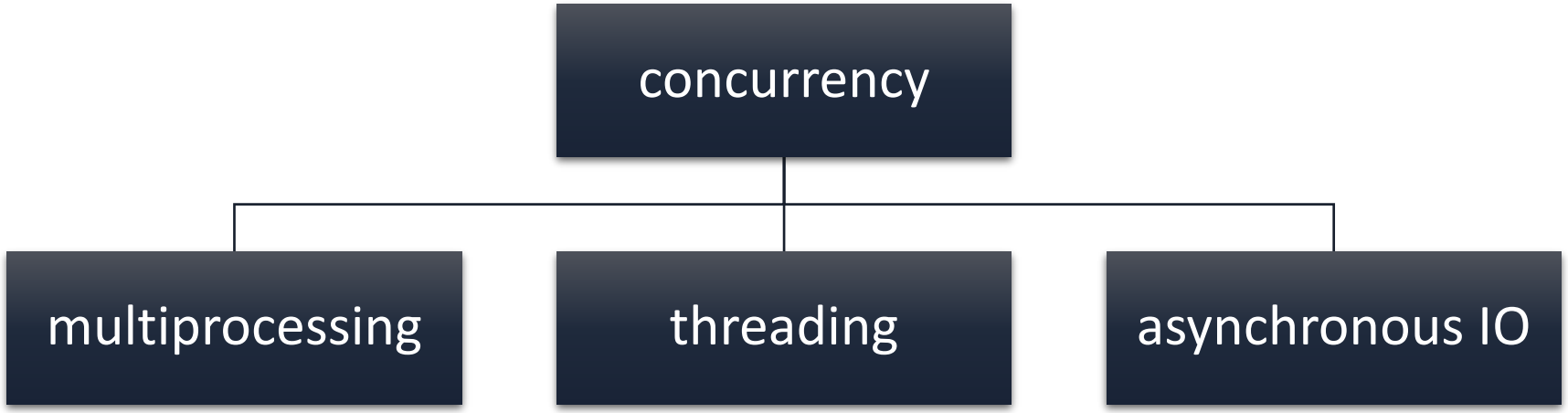
```
from datetime import date

d = date(2020, 2, 28)

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

Dateutil , calendar , arrow

- The *dateutil* module provides powerful extensions to the standard *datetime* module, available in Python.

```
from datetime import datetime
from dateutil.relativedelta import relativedelta

user_input = input('What is your date of birth? [dd-mm-yyyy] : ')

try:
   date_of_birth = datetime.strptime(user_input, '%d-%m-%Y')

except ValueError:
   print('Invalid date.')

else:
   delta = relativedelta(datetime.today(), date_of_birth)
   age = delta.years
   print(f'You are {age} years old.')
```

Concurrency



Python Standard Library:

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

Comparison

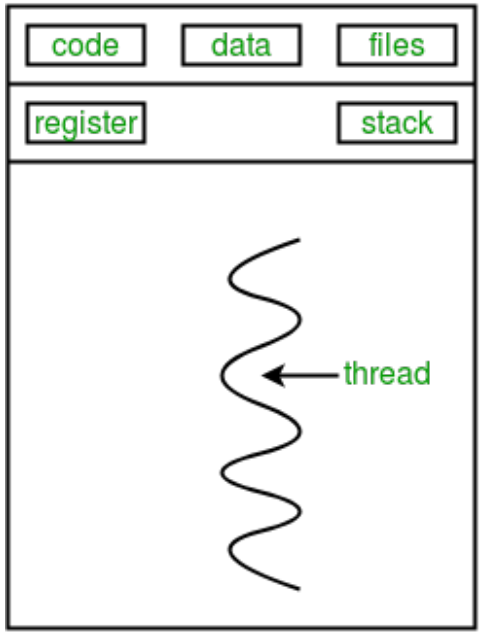
	Multiprocessing	Threading	Asynchronous IO
Package	multiprocessing	threading	asyncio
Class	Proces	Thread	Coroutine
Python	Class Proces	Class Thread	Keywords async, await
Data sharing	Message	Shared data	
Usage	CPU intensive	IO intensive	IO intensive

Proces versus Thread

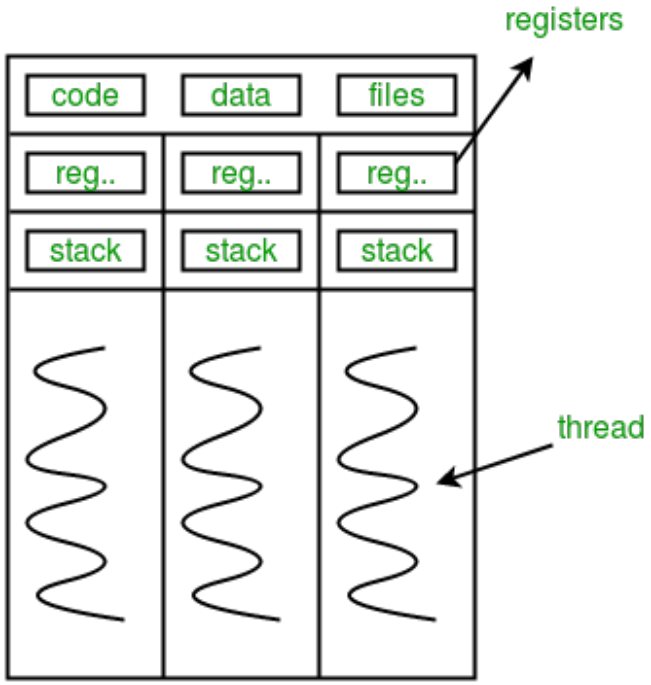
- True parallelism in Python is achieved by creating multiple processes, each having a Python interpreter with its own separate GIL.

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

Threading



single-threaded process



multithreaded process

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 potentionally blocking I/O operations.
- Note that both Jython and IronPython do not have the GIL.

Threading

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

```
import time
from threading import Thread

def myfunc(i):
    print(f'thread {i} sleeping for 5 secconds')
    time.sleep(5)
    print(f'thread {i} finished sleeping')

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(i):
    print(f'async coroutine {i} sleeping for 5 secconds')
    await asyncio.sleep(5)
    print(f'async coroutine {i} finished sleeping')

async def main():
    await asyncio.gather(count(1), count(2), count(3))

if __name__ == "__main__":
    asyncio.run(main())
```

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.

```
import time
from multiprocessing import Process

def f(i):
    print(f'process {i} sleeping for 5 secconds')
    time.sleep(5)
    print(f'process {i} finished sleeping')

if __name__ == '__main__':
    for i in range(5):
        p = Process(target=f, args=(i,))
        p.start()
```

Database access

PEP 249 - Database API Specification 2.0 https://peps.python.org/pep-0249/

- **Connection objects**
  - connect(…)
  - .close()
  - .commit()
  - .rollback()
  - .cursor()
- **Cursor objects**
  - .callproc()
  - .close()
  - .execute()
  - .executemany()
  - .fetchone()
  - .fetchmany()
  - .fetchall
  - .nextset
  - .arraysize

RDBMS

• SQLite	sqlite3
• Microsoft SQL Server	pyodbc
• Oracle	
• PostgreSQL	psycopg2
• MySQL	mysql -connector -python

Sqlite3

- DB-API 2.0 interface for SQLite databases

```
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()
```

PostgreSQL

SQL

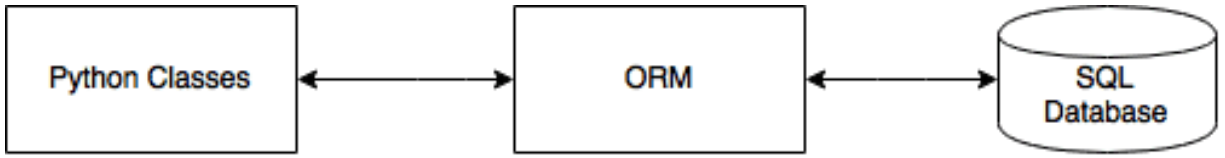
- CREATE TABLE
- ALTER TABLE
- DROP TABLE
- INSERT
- SELECT
- UPDATE
- DELETE

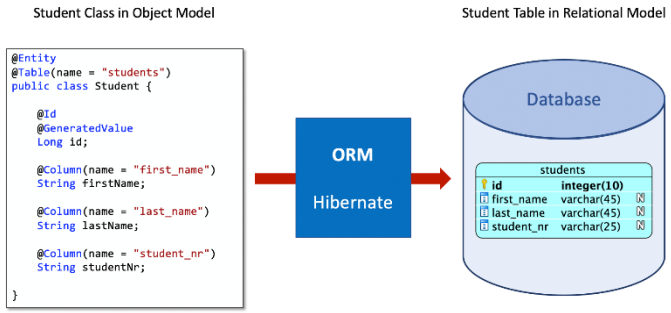
- CRUD
  - Create
  - Read
  - Update
  - Delete

- LIMIT clause
- FROM clause
- JOIN clause
- WHERE clause
- ORDER BY clause
- GROUP BY clause

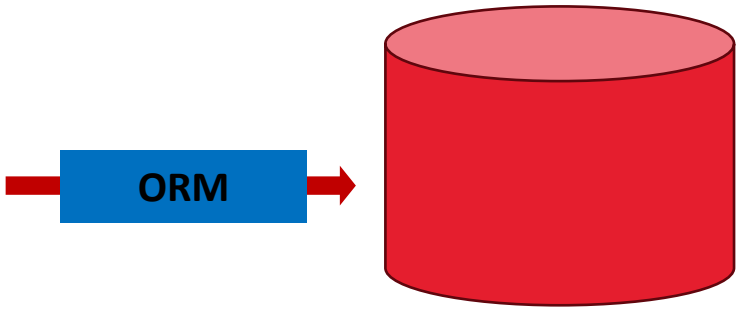
SqlAlchemy

- The Python SQL Toolkit and Object Relational Mapper






```python
from sqlalchemy.orm import *

class User(Base):
    __tablename__ = "users"
    id: Mapped[int] = mapped_column(primary_key=True)
    first_name: Mapped[str] = mapped_column(String(45))
    last_name: Mapped[str] = mapped_column(String(45))
    student_nr: Mapped[str] = mapped_column(String(25))
```



NoSQL

NoSQL databases are more flexible than relational databases. In these types of databases, the data storage structure is designed and optimized for specific requirements . There are four main types for NoSQL libraries :

- Document -oriented
- Key-value pair
- Column -oriented
- Graph

Python NoSQL:

- MongoDB
- Redis
- Cassandra
- Neo4j

Links

- https://pythonspot.com/python -database/
- https://www.geeksforgeeks.org/python -database -tutorial/
- https://www.tutorialspoint.com/postgresql/index.htm

Web Frameworks

- Django
- Flask
- Pyramid
- Bottle
- Sierra
- FastAPI

- Streamlit
- Dash

Web Framework

- Routing
- Database ORM
- Templating

Front-end development
HTML
CSS
Javascript
Media

Back-end development
Webserver
Programming Language
Web Framework
Database

Django

MVC Framework

Models

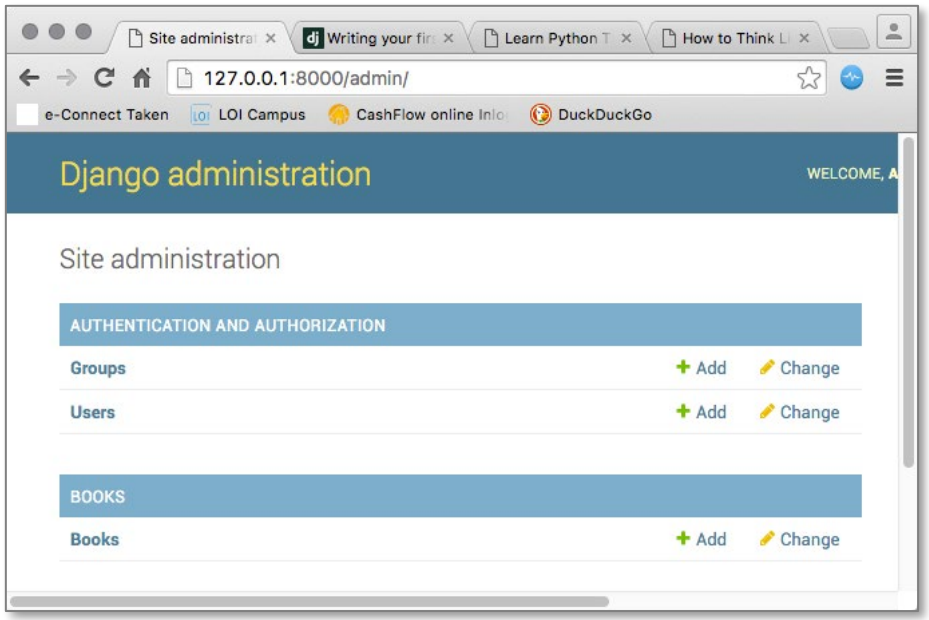
Object -Relational Mapping

URL Mapping

Views

HTML Templates

Command line bootstrap:



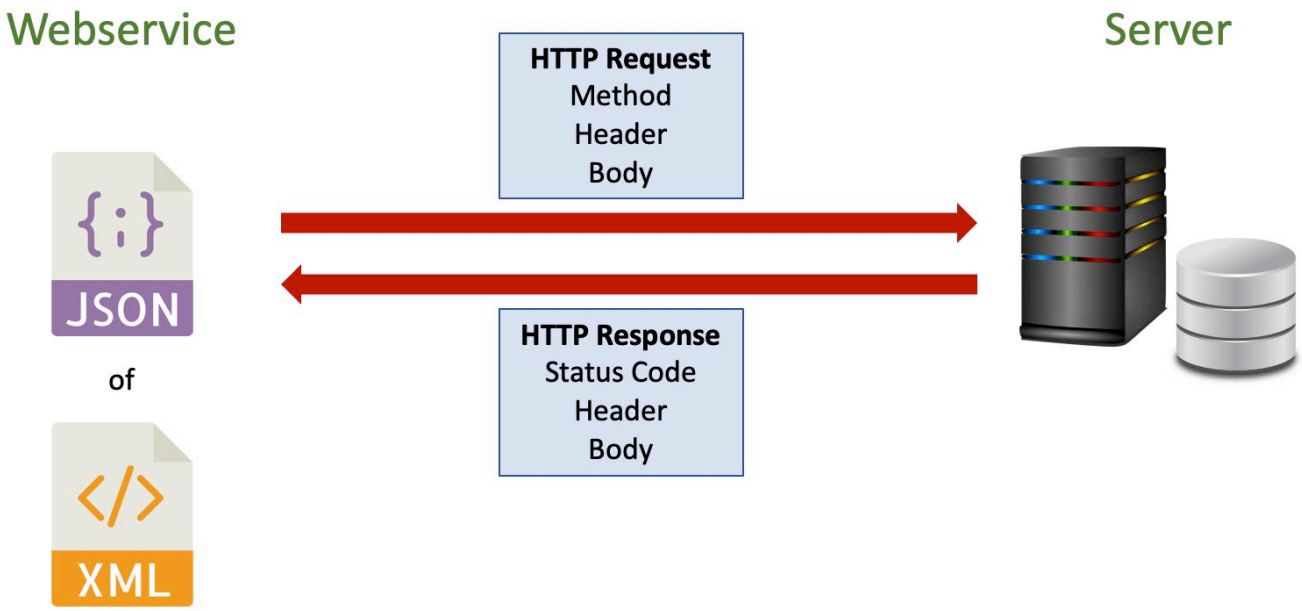

```
$ mkdir django-demo
$ cd django-demo
$ virtualenv -p python3.5 venv
$ . venv/bin/activate
$ pip install django
$ django-admin
$ django-admin startproject demo
$ python manage.py createsuperuser
$ python manage.py startapp books
$ python manage.py makemigrations
$ python manage.py migrate
$ python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

Flask

- **Flask** is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries.[2] It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools.

RESTful

- Client – Server
- Http Verbs – Get, Post, Put, Delete => CRUD
- Payload – Data - JSON



FastAPI

Webscrapping

- Requests
- Beautifulsoup
- Scrapy

- Selenium
- Robot Framework

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))
```

BeautifulSoup

Executable

- windows py2exe
- macos py2app
- linux / unix shebang #!/usr/bin/env python3

Alternative 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)