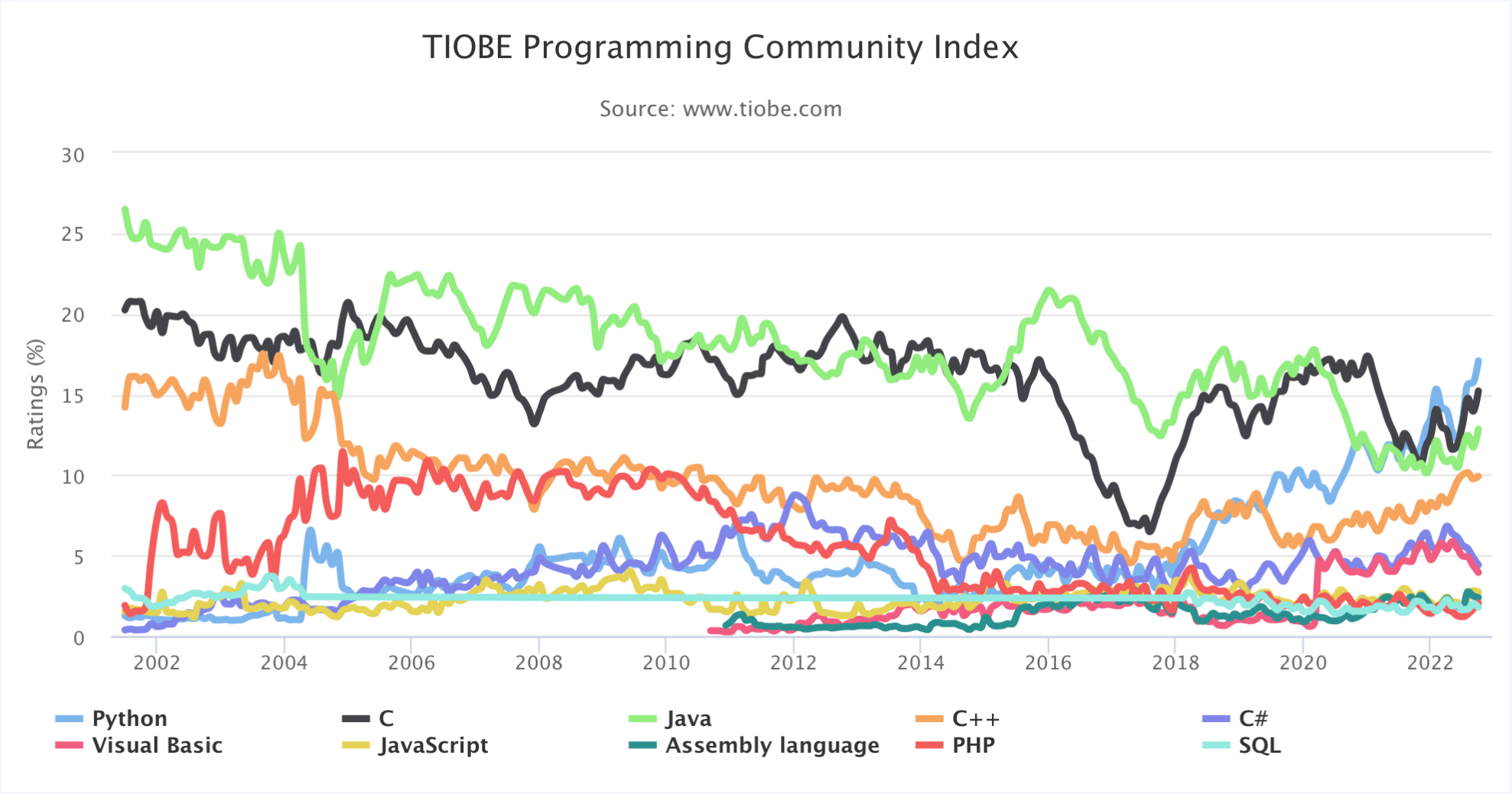
computrain

Python Programming

Popularity



https://www.tiobe.com/tiobe-index/

computrain

Program

Introduction
Variables en operators
Strings
Program Flow

Datastructures
Functions

Object oriented programming
Classes, methodes and attributes
Magic methodes
Inheritance

Python Standard Library
Python Package Index

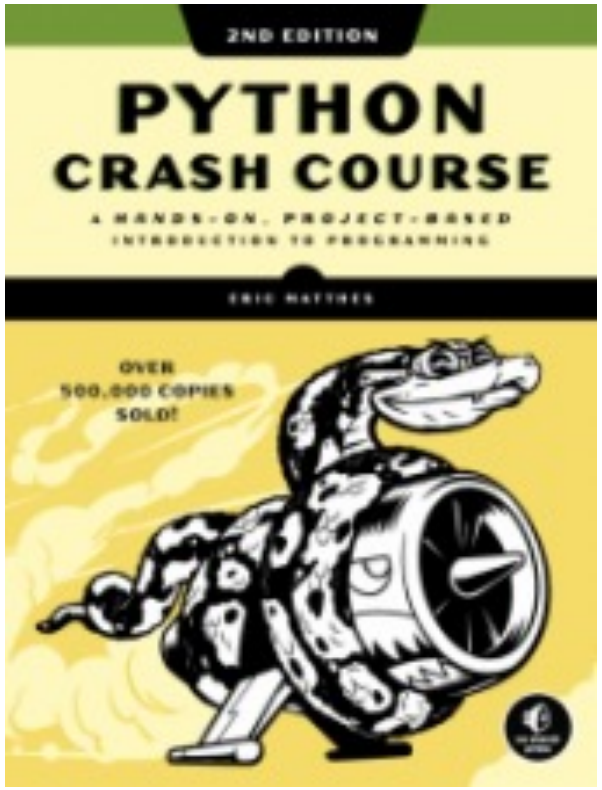
Book

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



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

Python background

- Since 1991
- Guido van Rossum
- Monty Python's Flying Circus
- Python 3 since 2008
- End of Life Python 2 in 2020
- The Zen of Python (import this)
- Pythonic, Pythonista, Idiomatic Python

Python features

- Intrepreted
- Multi-platform (Windows, Mac OS X, Linux, ...)
- Dynamic types
- Object oriented
- Datastructures
- Integration with C and C++
- Many Libraries

Python applications

- General Purpose
- Scripting
- Data processing
- Scientific
- Desktop GUI
- Web

Links

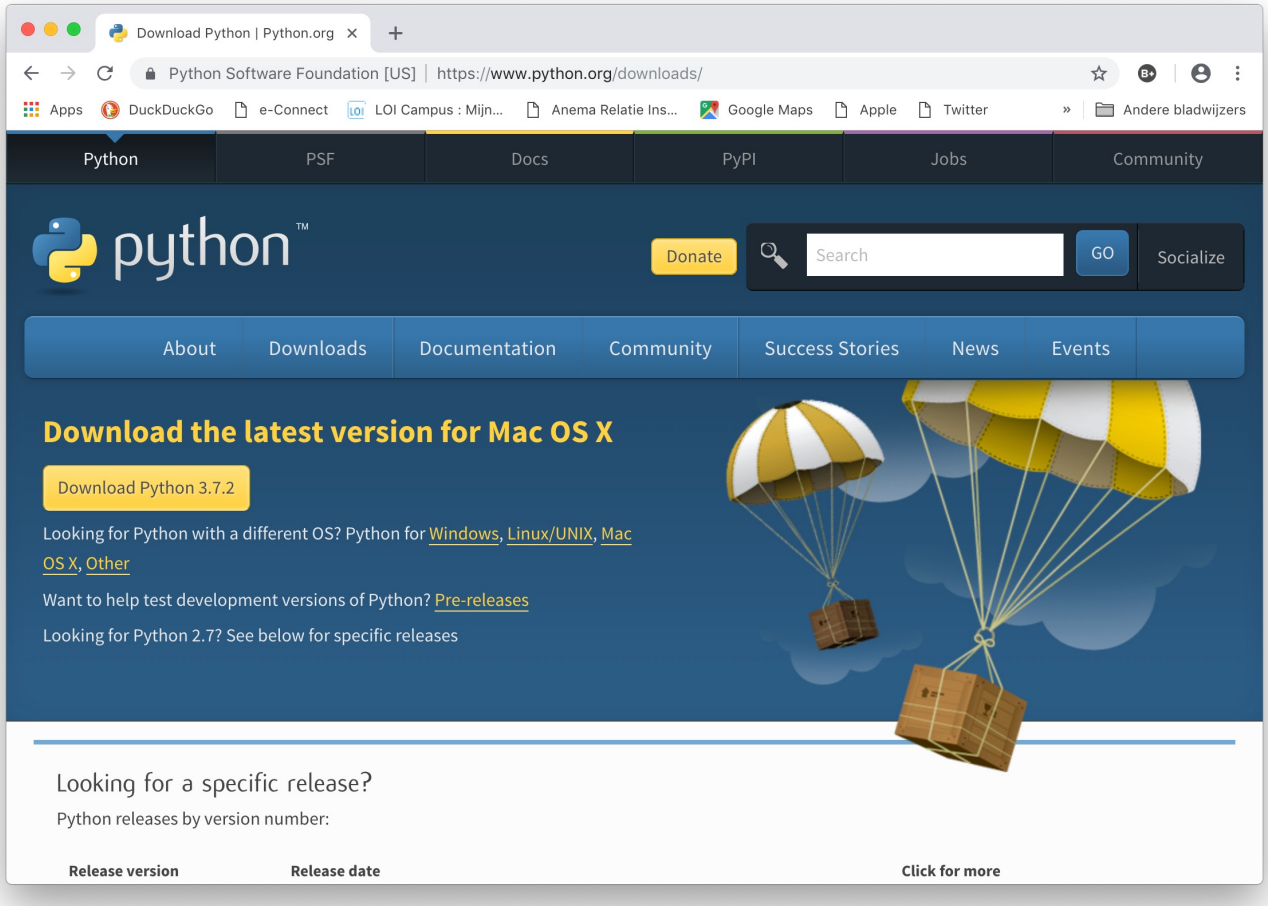
- https://www.python.org/

- https://www.w3schools.com/python/python_reference.asp
- http://www.pythontutor.com/visualize.html#mode=edit

- https://github.com/get-to-work-together/Python-Belastingdienst

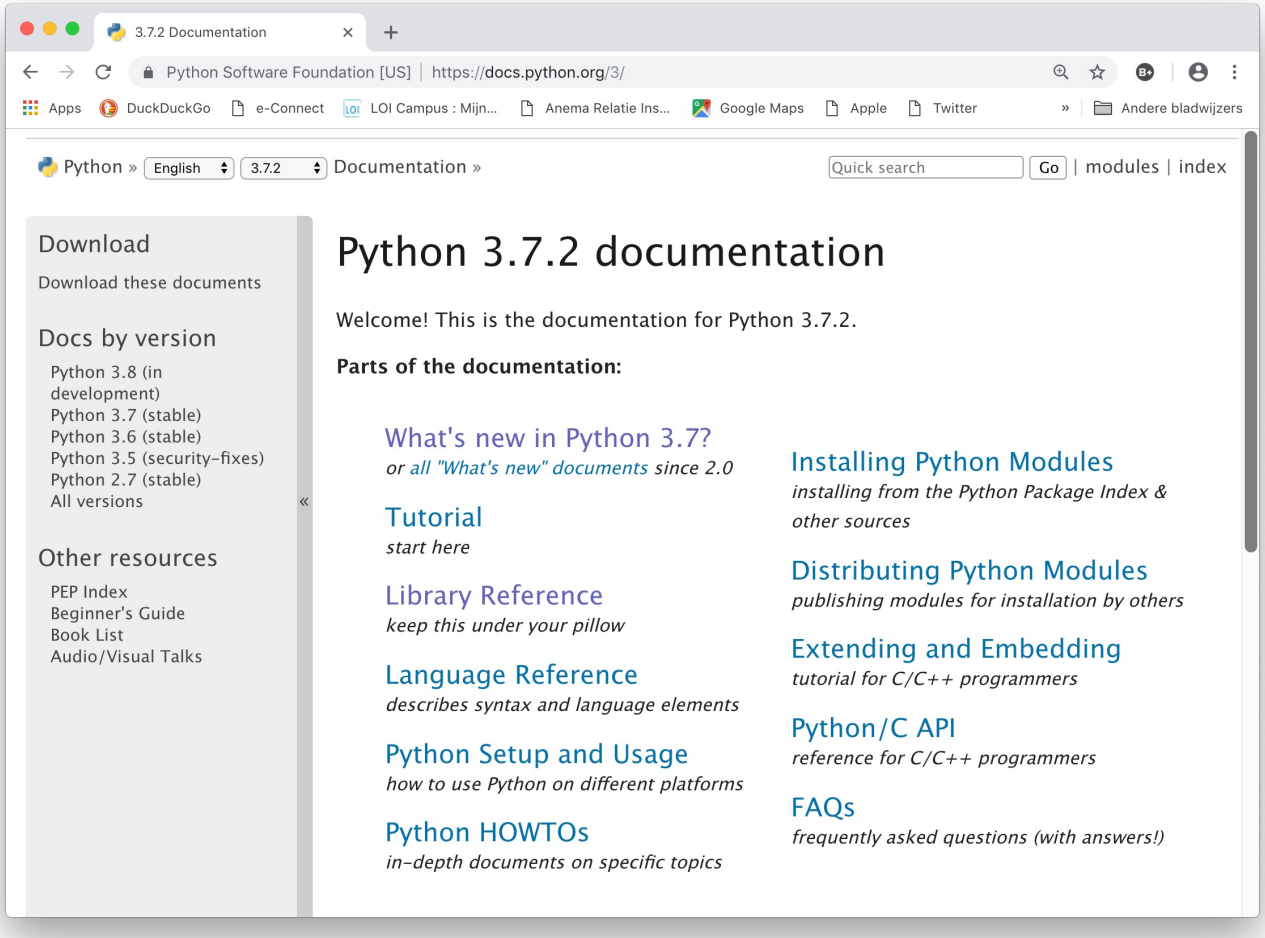
Installation Python

- https://python.org



Documentation

- https://docs.python.org/3/



computrain

Python 2 versus Python 3

- Not compatible !!!
- Python 3 since 2008
- End of Life Python 2.7 in 2020

Python 2.7	Python 3
print "hello"	print("hello")
raw_input("Geef tekst:")	input("Geef tekst")
5/2 => 2	5/2 => 2.5
byte strings en u'\u2660'	unicode strings
xrange()	range()
old-style and new-style objects	new-style objects

https://docs.python.org/3.0/whatsnew/3.0.html

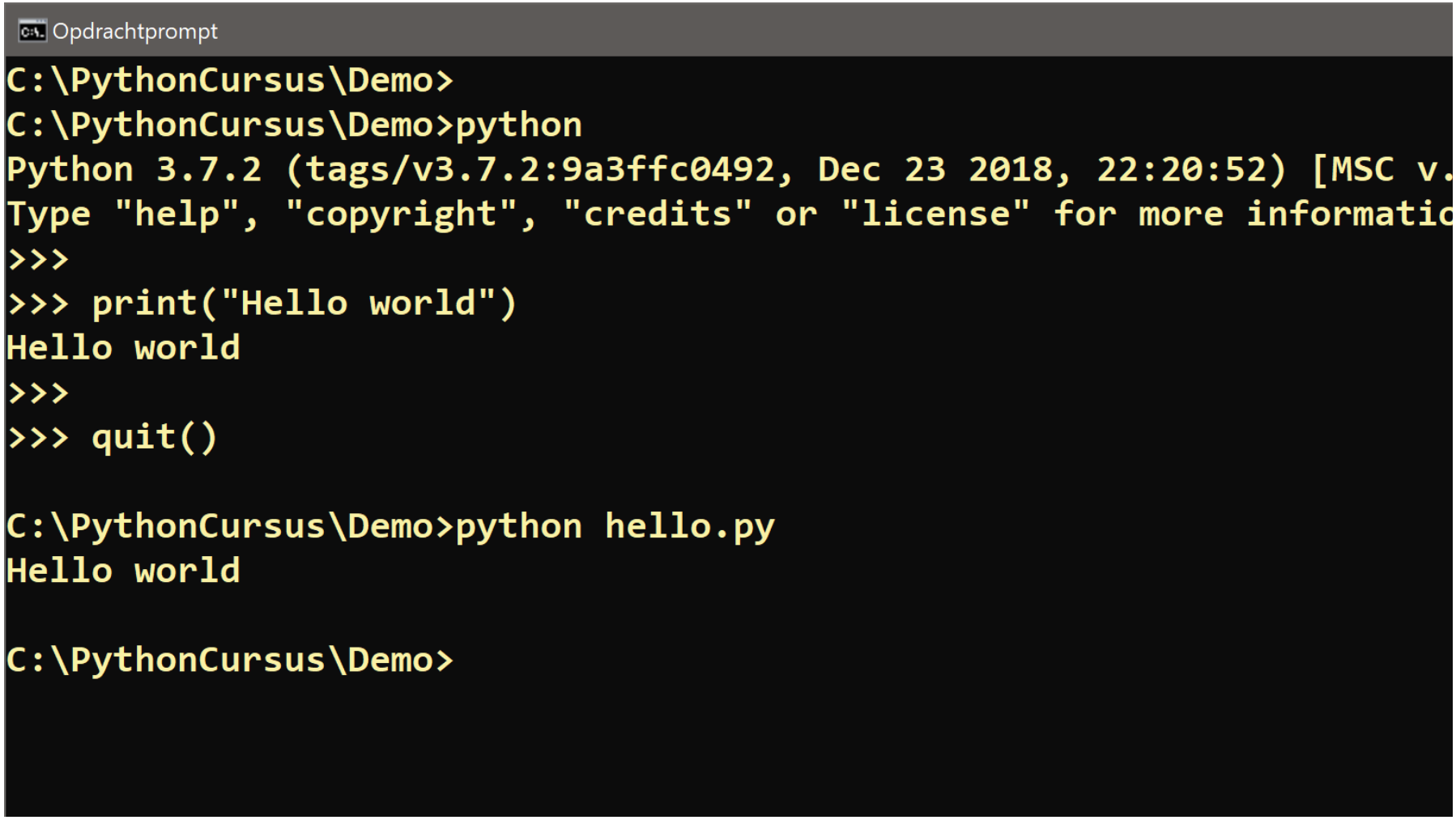
Python 2 and Python 3

- 2to3.py
- __future__ library
- six library

```
from __future__ import print_function
from __future__ import division
from __future__ import unicode_literals
from __future__ import absolute_import
```

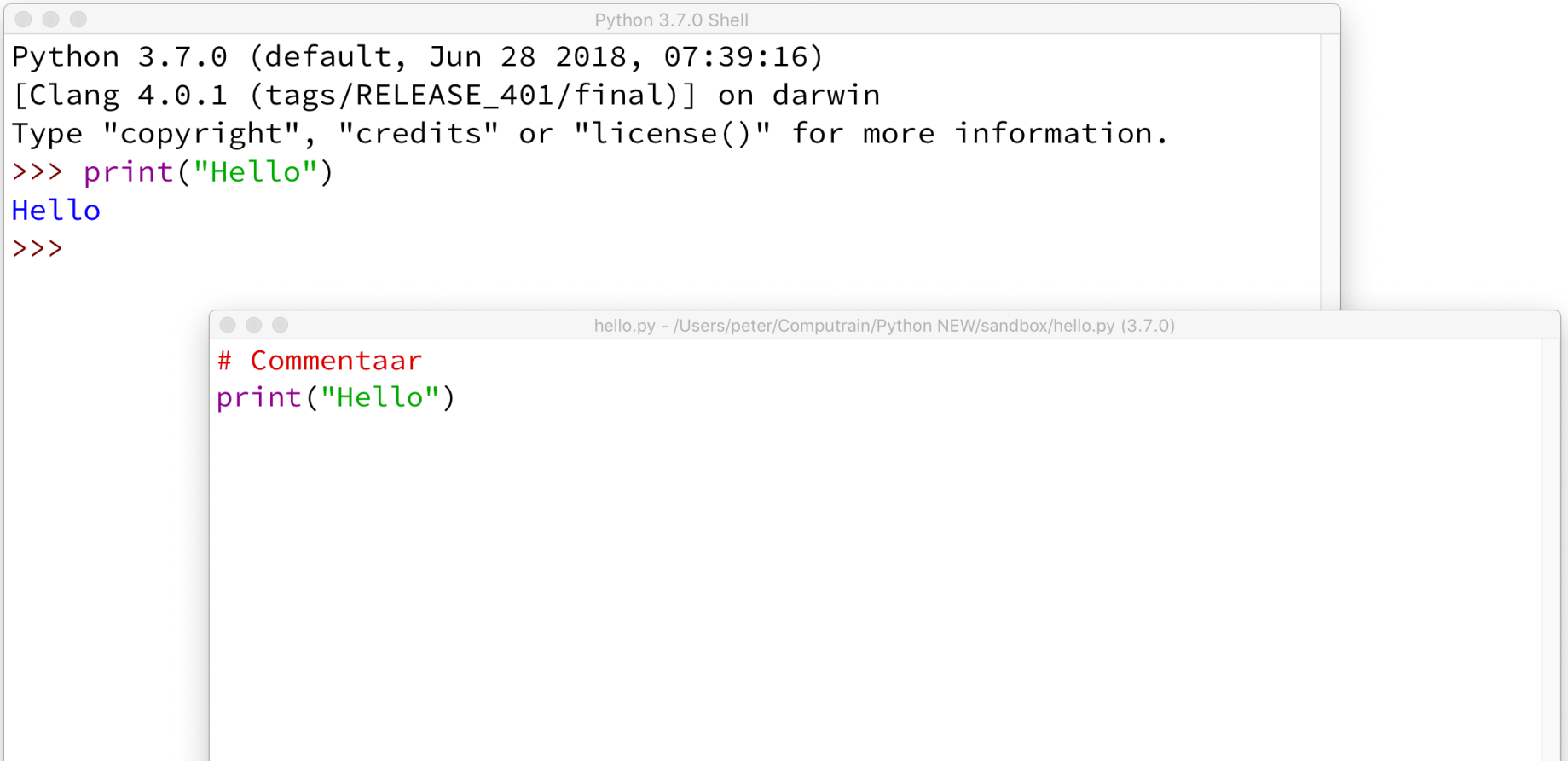
computrain

Python Prompt



IDLE

- Interactive Python Prompt
- Python editor



Other Editors & IDE's

Code Editors

- Notepad++
- Sublime
- Atom
- … and many more

Integrated Development Environments

- PyCharm
- VS Code with Python extension
- Spyder

Python prompt

Experiment with the python prompt

1. Open the python prompt.
    1. by opening IDLE
    2. by typing python in the command window
    3. in any other IDE. For example PyCharm.
2. Execute several numeric calculations.
3. Use the print function to print Hello World

Python editor

Create and execute a Python module

1. Open IDLE or an other IDE of your choice
2. Create a new Python file called first.py
3. Save this file in a newly created directory for this course
4. Use the print function to **print** Hello World
5. Save and run the file
6. Change the file to first ask for your name and store the result in a variable **name** using the **input()** function
7. Use the print function to print "Hello Albert" (if Albert is your name). You can use the + operator to concatenate both strings.
8. Save and run the file

computrain

Print function

- The **print()** prints text to the console.

```
>>> print("Hello world")
Hello world
>>> print(10)
10
```

Variables

- A variable does not need to be declared
- Variables have a name:
  - Upper and lower case letters, digits and underscore _
  - May not start with a digit
- Python is case sensitive!

```
>>> name = "Guido van Rossum"
>>> n = 10
>>> pi = 3.14159
>>>
>>> print(name)
Guido van Rossum
>>> print(n)
10
>>> print(pi)
3.14159
```

Input function

built-in function: **input()**

The input() function prints the prompt on the console and waits for the user to enter some text. The entered text is then returned as the return value of the function.

```
name = input('What is your name? : ')

print(name)
```

Comments

Everything a line following a # character is comment.

```
# demo.py
#
# This is a Python module.
#
# Always add comments.
#
# Comments clarify what's happening in the code.

name = "Guido" # Comments can also follow a statement
```

computrain

Keywords

and	del	if	pass
as	elif	import	raise
assert	else	in	return
async	except	is	True
await	False	lambda	try
break	finally	None	while
class	for	nonlocal	with
continue	from	not	yield
def	global	or	

computrain

Built-in functions

abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

computrain

Python Standard Library

- import to specify that a library is going to be used

- dir() to get the contents of a library
- help() to get help on the library

```
import math

dir(math)

help(math)
```

Import

A library must always be imported first.

- **import** math
- **from** math **import** pi, cos
- **import** math **as** m

```
import math # import module
print( math.pi )

from math import pi  # import variable / function from module
print( pi )

from math import *  # import all from module
print( pi )

import math as m  # import module as an alias
print( m.pi )
```

Modules

A Python module:

- a file with python code
- **.py** extension

When importing a module a file with that name is search first in the current directory.

sys.path has list of directory paths that will be searched when importing a library.

computrain

Packages

A Python package:

- consists of modules
- and subpackages
- grouped together in a directory
- with an **__init__.py** file

- import

```
package/
    __init__.py
    subpackage1/
        __init__.py
        module1.py
        module2.py
    subpackage2/
        __init__.py
        module3.py
        module4.py
```

```
import package.subpackage1.module1 as m1
from package.subpackage2 import module3
```

computrain

Numeric Types

- whole numbers — int()
- decimal numbers — float()
- complex numbers — complex()

```
i1 = 8
i2 = 73492734987239874239874
i3 = int('15')

f1 = 1.5
f2 = 7e-10
f3 = float('3.14')

c1 = 1j # square root of -1
c2 = complex(2)
```

Boolean

- A boolean variable can be **True** or **False**
- These are also keywords
- bool()

- A value is evaluated to False if:
  - 0
  - 0.0
  - 0j
  - ''
  - ()
  - []
  - {}
  - False

- Otherwise True

String

- A string can be specified with single quote's '
- or with double quote's ".
- or even with triple quote's """

- The function **str()** converts values to a string.

- Concatenation:
  - Strings are concatenated with a **+**

```
firstname = 'Albert'
lastname = 'Einstein'
name = firstname + ' ' + lastname
print(name)
```

computrain

Numeric operators

- addition +
- subtraction -
- multiplication *
- division /
- floored division //
- modulo %
- power **

computrain

Math library

- import **math**

- methods in the math library

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

computrain

Random library

- import **random**

- methods in the random library:
  - seed()
  - random()
  - randint()

```
import random

random.seed(999)

random_number = random.randint(1, 100)
```

computrain

Comparison operators

• is larger	>
• is larger or equal	>=
• is smaller	<
• is smaller of equal	<=
• is equal	==
• is not equal	!=
• is identical	is
• is not identical	is not
• is element in list	in

Conditional operators

- and
- or
- not

A	B	A and B	A or B
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

```
number = 7
is_valid_number = number >= 1  number <= 10
```

Bitwise operators

- and &
- or |
- xor ^
- not !

A	B	A & B	A | B	A ^ B
0111	1111	0111	1111	1000
0111	0001	0001	0111	0110
0111	1010	0010	1111	1101

Leapyear

Write a program that determines if a year is a leapyear.

1. Create a new Python module with a name like leapyear.py
2. Then ask the user to **input** a year.
3. Change the input to a number using **int()**
4. Calculate if the year is a leapyear
    1. a year is a leapyear if the **year can be divided by 4**
    2. but (and) the **year can not be divided by 100**
    3. except (or) if the **year can be divided by 400**
5. **Print** the results
6. Test your program for different years

Tip: Use the module operator to compare the remainder of a division with 0 to determine if a number can be divided by another number. E.g.: 2021 % 4 == 0.

Calculate a circle

Write a program that calculates the area and circumference of a circle.

1. Create a new Python module with a name like circle.py
2. First **import** the math library.
3. Then ask the user to **input** the radius.
4. Change the input to a number using **float()**
5. Calculate the area with **area = πr^2**
6. Calculate the circumference with **circumference = $2\pi r$**
7. Print the results

Tip: The math library has a value for **π** in math.pi.

Random dice

Write a program that simulates throwing 5 dice.

1. Create a new Python module with a name like dice.py
2. Import the random library
3. Generate a random number between 1 and 6 with random.randint(1, 6) and store the number in a variable like dice1.
4. Repeat this 4 more times creating variables dice2 up to dice5.
5. Print the values of the dice
6. Also print the total sum of the values

String formatting

- Concatenation +
- Format operator %
- Format method .format()
- F-strings f'....'
- Print function with arguments print(.., .., .., ..)

```
name = 'Guido'
age = 62

print( name + ' is ' + str(age) + ' years old' )
print( '%s is %d years old ' % (name, age) )
print( '{} is {} years old'.format(name, age) )
print( f'{name} is {age} years old' )
print( name, 'is', age, 'years old' )
```

computrain

String methods

capitalize	index	isspace	rfind	swapcase
casefold	isalnum	istitle	rindex	title
center	isalpha	isupper	rjust	translate
count	isascii	join	rpartition	upper
encode	isdecimal	ljust	rsplit	zfill
endswith	isdigit	lower	rstrip	
expandtabs	isidentifier	lstrip	split	
find	islower	maketrans	splitlines	
format	isnumeric	partition	startswith	
format_map	isprintable	replace	strip	

```
name = 'Guido'
print( name.upper() )
print( name.lower() )
print( name.isnumeric() )
```

computrain

Index and slicing

- A string behaves as a list of characters that can be selected with an index:
  - s[0] => first character
  - s[1] => second character
  - s[-1] => last character

- A string can be sliced:
  - s[0:4] => the first four characters
  - s[:4] => the first four characters also
  - s[-3:] => the last three characters

Unicode

- In Python 3 all strings are unicode strings.

- List of Unicode characters at
  https://en.wikipedia.org/wiki/List_of_Unicode_characters

eg.:

\u2660	\u2665	\u2666	\u2663
♠	♥	♦	♣
\u2664	\u2661	\u2662	\u2667
♤	♡	♢	♧

```
print( 'Patiënt' )
print( 'Pati00EBnt' )
print( '\u2665' )
print( '\u20AC' )    # Euro sign €
```

Working with strings

Experiment with strings.

1. Create a new python file. E.g. strings.py
2. Ask the user to input some text and store the response in a variable.
3. Print the text in all uppercase and also in all lowercase characters.
4. Use the capitalize() and title() methods and print the results
5. Check if the tekst ends with a question mark.
6. Print the tekst in lowercase with all spaces replaced by an underscore. This is called **snake_case**

Count vowels

- Get some tekst from input and put this in a variable
- Loop through the vowels 'a', 'e', 'i', 'o', 'u', 'y'
- Count the number of occurances in the text
- Print a message for each vowel indicating the number of occurances
- After looping through the vowels
- … print a message indicating the total length of the text
- … and the total number of vowels

```
The complete text contains 929
characters.

Found the vowel 'a' 58 times
Found the vowel 'e' 97 times
Found the vowel 'i' 66 times
Found the vowel 'o' 39 times
Found the vowel 'u' 23 times
Found the vowel 'y' 8 times

The text contains 291 vowels.
```

Conditional statement

- **if**
- **if … else**
- **if … elif … else**

- semi-colon **:**

- Indenting!
  - 4 spaces

```
gender = ...
age = ...

if gender == 'm':
    if age < 21:
        print('boy’)
    else:
        print('man’)
elif gender == 'v’:
    if age < 21:
        print('girl’)
    else:
        print('woman')
else:
    print('other')
```

Conditional operator

- One-liner conditional expression if only a value if required

- outcome1 **if** condition **else** outcome2

```
salution = 'sir' if gender.lower() == 'm' else 'madam'
```

Match

- NEW! Since Python 3.10

- match
- case

- wildcards
- conditional
- patterns
- guards

```
gender = ...

match gender:
case 'm':
    print('man')
case 'f' | 'v':
    print('woman')
case _:
    print('other')
```

https://www.python.org/dev/peps/pep-0622/

computrain

Looping with while

- **while** keyword
  - repeat as long as the condition is **True**

- Someway or another the condition has to be set to **False** to stop the loop.

```
counter = 0
while counter < 10:
    print(counter)
    counter += 1
```

computrain

Looping with for

- keywords **for** ... **in**
  - Repeat the statements for each element in the list

```
for number in [0,1,2,3,4,5,6,7,8,9]:
    print(number)

for letter in ['A','B','C']:
    print(letter)

for word in ['Python','is','beautiful']:
    print(word)
```

computrain

range()

- **range(stop)**
  - Generates numbers from 0 to stop. Stop is not included!
- **range(start, stop)**
  - Generates numbers from start to stop. Stop is not included.
- **range(start, stop, step)**
  - Generates numbers from start to stop with is step size. Stop is not included.

```
for number in range(10):
    print(number)

for getal in range(1, 11):
    print(number)

for getal in range(1, 11, 2):
    print(number)
```

Break en Continue

- **break**
  - Stops looping and steps out of the loop
- **continue**
  - Stops with the current loop and continue with the next element

```
magicnumber = 11

for i in range(1, 21):
    if i == magicnumber:
        break
    print(i)

for i in range(1, 21):
    if i == magicnumber:
        continue
    print(i)
```

Pythonic

- **while True**
- Python Does not have a **do ... while** statement. The condition is always evaluated before the loop. It is possible that the statements in loop are never executed.
- A do ... while statement can be simulated with a while True statement combined with a break condition.

```
while True:
    number = int(input('Enter an number between 1 and 10: '))

    if 1 <= number <= 10:
        break

print('The number is %d' % number)
```

Exercise 1.7

Life stage

Print the stage of life depending on the age entered by the user.

Age	Life stage
0 - 2	Baby
2 - 4	Toddler
4 - 13	Kid
13 - 20	Teenager
20 - 65	Adult
65 or older	Elder

Tips:

- Create a new python module
- Use **input()** to ask for the age
- Assign the integer value to a variable. Use **int().**
- Use a serie of **if** and **elif** statements to determine which message to print depending on the age entered. The upper bound is exclusive.

Exercise 5.6 (page 85) from the Book "Python Crash Course"

Guessing game

Guess a number between 1 and 100
What is your next guess? 50
lower ...
What is your next guess? 25
lower ...
What is your next guess? 12
higher ...
What is your next guess? 19
higher ...
What is your next guess? 22
lower ...
What is your next guess? 21
YEAAAH! You guessed it in 6 guesses

```
import random
secret_number = random.randint(1, 100)
```

Which order?

1. print('lower ...')
2. print('YEAAAH!!!! You guessed it in %d guesses' % number_of_guesses)
3. while True:
4. if guess > magic_number:
5. number_of_guesses += 1
6. magic_number = random.randint(1, 100)
7. break
8. print('Guess a number between 1 and 100')
9. print('higher ...')
10. else:
11. guess = int(input('What is your next guess? '))
12. import random
13. elif guess < magic_number:
14. number_of_guesses = 0

computrain

Datastructures

- Sequence types
  - list
  - tuple
- Set types:
  - set
- Dictionary types
  - dict

computrain

List

- A mutable list of elements.
- There is an order.
- Square brackets **[]**
- Built-in function **list()**

```
list1 = []    # empty
list2 = [9,8,7,6,5,4,3,2,1]
list3 = ['Amsterdam','New York','Parijs']
list4 = list(range(10))
```

List modification methods

- append()
- extend()
- insert()
- pop()
- remove()
- sort()
- reverse()
- del

```
codes = ['NL','B','L']

codes.append('F')# ['NL','B','L','F']
codes.extend(['D', 'I'])  # ['NL','B','L','F','D','I']
codes.insert(1, 'ES')  # ['NL','ES',B','L','F','D','I']
code = codes.pop()  # ['NL','ES',B','L','F','D']
codes.remove('L')# ['NL','ES',B','F','D']
del codes[1]  # ['NL',B','F','D']
codes.sort()  # ['B','D','F','NL']
```

Built-in functions and lists

len()	map()	all()
max()	filter()	any()
min()	sorted()	
sum()		

```
l1 = [1, 4, 7, 9, 2]

len(l1)              # 5
max(l1)              # 9
min(l1)              # 1
sorted(l1)           # [1, 2, 4, 7, 9]
```

computrain

index and slicing

- index [index]
- slicing [start:stop] of [start:stop:step]
- Built-in function **slice()**

```
import string
letters = list(string.ascii_uppercase)

print( letters[0] )  # A
print( letters[10] ) # K
print( letters[-1] ) # Z
print( letters[0:3] )# ['A','B','C']
print( letters[:3] ) # ['A','B','C']
print( letters[10:13] ) # ['K','L','M']
print( letters[-3:] )# ['X','Y','Z']
klm = slice(10,13)
print( letters[klm] )# ['K','L','M']
```

in

- **in** operator

- Evaluates to **True** if the element is in the list (or tuple or set).

```
cities_visited = ['Amsterdam','New York','Parijs']

destination = 'Amsterdam'

if destination in cities_visited:
    print("Been there!")
```

computrain

Sequence type operations

- concatenation **+**

- **index()** method
- **count()** method

```
list1 = ['a','b','c']
list2 = ['x','y','z']
list3 = list1 + list2          # concatenation

print( list3.index('b') )      # 1
print( list3.count('b') )      # 1
```

split and join

- **split()** method
  - returns a list of parts

- **join()** method
  - Returns a string with all elements concatenated

```
sentence = "number of cars"

words = sentence.split()
print(words)

print('_'.join(words))
```

List comprehension

- Create a list from another list
- [x ** 2 **for** x **in** range(10)]
- [x ** 2 **for** x **in** range(100) **if** x % 5 == 0]

```
[x**2 for x in range(10)]          # [0,1,2,9,16,25,36,49,64,91]
[x**2 for x in range(100) if x%5 == 0]     # [0,25,100,225,...]

[name[0].upper() for name in ['peter', 'tom', 'Harshini']
```

Print list of entered names

Enter a number of names. If no name is entered (return) continue with the rest of the program and print the entered names. Sorted if possible.

Tips:

- Start with an empty list **names = []**
- Use a **while** loop to ask for a name with **name = input(...)**
- Add the entered name to the list with **names.append(name)**
- If no name has been entered **break** out of the loop
- **Print** the entered names in a for loop.
- Sort the list with **sorted(names)**

Create a random password

Write a program that creates a new password that meets the requirements:

- at least 1 uppercase character
- at least 1 lowercase character
- at least 1 digit
- at least 1 special character

1. Import the random library
2. Create lists for each character group and use choices to randomly select a number of characters from each group
3. **Concatenate** the lists to one list
4. **Shuffle** the resulting list
5. **Join** all the characters to a string and print the result

Tip: Use the functions **choices** and **shuffle** from the random library.

Playing cards

Select 5 random cards from a deck of playing cards.

Tips:

- Define the 4 suits in a **list**
  - suits = ['clubs', 'diamonds', 'hearts', 'spades']
- Define the 13 ranks in a **list**
  - ranks = '2 3 4 5 6 7 8 9 10 J Q K A'.split()
- Combine these lists in a new **list** with all combinations using a double list comprehension:
  - cards = [f'{r} {s}' for r in ranks for s in suits]
- Shuffle the list with **random.shuffle(cards)**
- Select 5 cards with **cards.pop()**
  - hand = [cards.pop() for _ in range(5)]

computrain

Tuple

- An inmutable list of elements
- Similar to a list

- Round brackets ()
- Or no brackets at all!

- Built-in function **tuple()**

```
tuple1 = ()                          # empty
tuple2 = (9,8,7,6,5,4,3,2,1)
tuple3 = ('Amsterdam','New York','Parijs')
tuple4 = tuple(range(10))
tuple5 = (1,)                       # one element
tuple6 = 'NL','B','L'
```

computrain

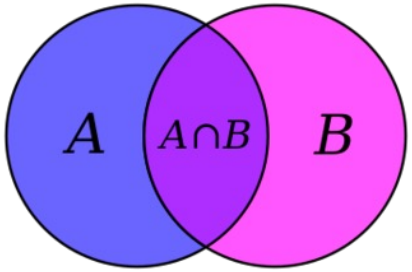
Unpacking

- Lists and tuples can be '**unpacked**' into variables
- Instead of assigning one value at a time

```
list1 = ['a','b','c','d','e']
v1, v2, v3, v4, v5 = list1
v1, v2, *rest = list1
v1, v2, *_ = list1

v1, v2 = v2, v1                # swap the value of v1 and v2
```

Set

- A set of unique elements without any order.
- Curly brackets **{}**
- Built-in function **set()**




```
s1 = set()                  # empty set
s1.add(5)                   # {5}
s1.update({1,7,9})          # {1, 5, 9, 7}
s1.remove(9)                # {1, 5, 7}
s1.discard(9)               # {1, 5, 7}

9 in s1                     # False
```

computrain

Set methodes

add	intersection	remove
clear	intersection_update	symmetric_difference
copy	isdisjoint	symmetric_difference_up
difference	issubset	date
difference_update	issuperset	union
discard	pop	update

```
s1 = {1, 4, 7, 9, 2}
s2 = {2, 4, 6, 9}

s1.union(s2)                    # {1, 2, 4, 6, 7, 9}
s1.intersection(s2)             # {9, 2, 4}
s1.difference(s2)               # {1, 7}

s1 | s2                         # {1, 2, 4, 6, 7, 9}
s1 & s2                         # {9, 2, 4}
s1 - s2                         # {1, 7}
```

Dict

- A collection of **key** – **value** pairs

- Built-in function **dict()**

```
d = {'nl':'+31', 'b':'+32', 'uk':'+44'}

d['nl'] # '+31'
d['f'] = '+33'

d.get('d')
d.get('d', '???')

d.keys()
d.values()
d.items()

d.update({'d':'+49', 'es':'+34'})
```

zip function

- The zip function combines multiple lists of the same length to one list of tuples

```
keys = ['Amsterdam','Eindhoven','Utrecht','Delft']
values = ['020','040','030','015']

d = dict(zip(keys, values))

d['Amsterdam'] # => '020'
```

Occurance of words

- Copy an arbitrary piece of text from internet
- Create a python script that reads the text
  - use: s = input()
- Change to lower case and remove punctuation characters (dots and commas)
  - use: s = s.lower().replace('.', '') .replace(',', '')
  - or: s = s.lower().translate(str.maketrans('','', string.punctuation))
- Split the text into words
  - use: words = s.split()
- Create a set of unique words with the set() function
- For each unique word count the number of occurances in the list of words
- Store the results in a dictionary
  - d[word] = n
- Print the results
  - for word, n in d.items()

computrain

Working with datastructures

List comprehension	[i**2 for i in range(10)] [i**2 for i in range(10) if i%2==0] [i*j for i in range(5) for j in range(5)]
Set comprehension	{e for e in s}
Dict comprehension	{k: v for k, v in d.items())}
Generator	(e for e in range(10))
Sorting	sorted(list, key = lambda item: len(item))
Map	map(lambda x: x**2, range(10))
Filter	filter(lambda x: x%2==0, range(10))

computrain

Other datastructures

- array
- namedtuple
- deque
- enum

```
from array import array

array('l')
array('u', 'hello \u2641')
array('l', [1, 2, 3, 4, 5])
array('d', [1.0, 2.0, 3.14])
```

```
from collections import namedtuple

Point = namedtuple('Point', ['x', 'y'])
p1 = Point(11, 22)
p2 = Point(x=11, y=22)
p1.x # => 11
p1.y # => 22
```

```
from enum import Enum

class Color(Enum):
    RED = 1
    GREEN = 2
    BLUE = 3

print(Color.RED)
```

```
from collections import deque

numbers = deque([4,5,2,7,9,2])
numbers.append(9)
numbers.appendleft(0)
numbers.pop()
numbers.popleft()
```

Functions

- Statements grouped together to preform a certain task

- Always consists of two steps:
  1. defining a function with the **def** keyword
  2. calling the function using parentheses

```
def print_goodmorning():
    print('Goodmorning')
    print('How are you today?')
    print('Have a great day!')


print_goodmorning()
```

Arguments

- Arguments can be passed to functions

```
def print_goodmorning(name):
    print('Goodmorning %s' % name)
    print('How are you today?')
    print('Have a great day!')


print_goodmorning('Albert')
print_goodmorning('Peter')
```

Arguments with default values

- Arguments can have default values
- If the argument is not passed to the function de default value is used.

```
def book_flight(fromairport, toairport, numadults=1, numchildren=0):
    print('\nFlight booked from %s to %s' % (fromairport, toairport))
    print('Number of adults: %d' % numadults)
    print('Number of children: %d' % numchildren)

# Usage (i.e. client code)
book_flight('BRS', 'VER', 2, 2)
book_flight('LHR', 'VIE', 4)
book_flight('LHR', 'OSL')
```

computrain

Keyword arguments

- Arguments can be specified by the name of the argument.
- Keyword arguments can be specified in any order

```
def book_flight(fromairport, toairport, numadults=1, numchildren=0):
    print('\nFlight booked from %s to %s' % (fromairport, toairport))
    print('Number of adults: %d' % numadults)
    print('Number of children: %d' % numchildren)

# Usage (i.e. client code)
book_flight(fromairport='BRS', toairport='VER', numadults=2, numchildren=2)
book_flight('LHR', 'CDG', numchildren=2)
book_flight(numchildren=3, fromairport='LGW', toairport='NCE')
```

Return value

- A result can be returned with the **return** keyword

```
def calculate_bmi(weight, height):
    bmi = weight / height ** 2
    return bmi

print(calculate_bmi(90, 1.80))
```

Local variables

- The scope of a variable is defined as the region in the code where the variable is valid

- Variables defined within a function have a **local scope**.
- Otherwise a variable has a **global scope.**
- These are only valid withn the function.

- To use and modify a global variable inside of a function use the **global** keyword

```
def print_goodmorning(name):
    print('Goodmorning %s' % name)

name = 'Albert'
print_goodmorning(name)
```

Putting functions in a module

- Functions can be grouped together in a module
- The module can be imported whenever you want to use one of the functions
- The **sys** module has a **path** variable specifying the directories to look for the module.

functions.py

```
def do_something():
    pass

def do_something_else():
    pass
```

```
import functions
functions.do_something()

import functions as fu
fu.do_something()

from functions import do_something_else
do_something_else()
```

First class citizens

- Functions are first-class citizens in Python. This means that functions can be passed round just as other objects and values.

```
def print_goodmorning(name):
    print('How are you today?')

f = print_goodmorning

f('Peter')
```

Lambda

- The **lambda** keyword is used to specify an anonymous function

```
is_even = lambda number: number % 2 == 0
```

Sort words on number of vowels

Tips

- Enter a piece of tekst
- Split into words
- Use the sorted function to sort these words
- Add a **key** argument using lambda to define a function
  - E.g. lambda word: sum([word.count(v) for v in 'aeiouy'])

computrain

Variadic arguments

- Variadic arguments can take any number of arguments
- Use a * character
- The arguments are collected in a list

```
def maximum(*numbers):
    highest = numbers[0]
    for number in numbers:
        if number > highest:
            highest = number
    return highest


maximum(2, 5)
maximum(2, 5, 7, 3, 4)
```

Leap year

Create a function that determines if a year is a leapyear.

Tips:

- Define a function with the year as an argument:
  - E.g. **def is_leapyear(year):**
- Do the calculation:
  - **b = (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)**
- Return the result
  - Return b
- Test the function for different years

Banner

Create a function that prints text surrounded by stars. Like a banner.

```
**********
*  Peter  *
**********
```

Tips:

- Define the function called **print_banner**
- Define one argument called text
- Print out the lines

min_max function

Create a function that calculates and returns the minumum and maximum of a list of numbers by looping through the numbers only once.

Tips

- Define a function min_max that takes a varying number of arguments
- Start with local variables minimum and maximum with the value of the first number.
- Loop through the numbers and update the variables if necessary
- Return the minimum and maximum as a tuple.

remove_all

Create a function that will remove all instances of a specified values from a list.

Tip:

- Define a function remove_all with argument list and the value to remove
- In a while loop recurrently call the remove method of the list

computrain

Functional programming

- Pure functions
- No side-effects
  - no globals
  - no printing

computrain

Built-in functions

- sorted()
- filter()
- map()

```
l1 = ['one','two','three','four']
l1_sorted = sorted(l1, key = len)

l2 = [23, 45, 56, 38, 59, 82, 75]
l2_filtered = filter(lambda x: x%5 == 0, l2)

l3_mapped = map(lambda x: x**3, range(10))
```

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
```

Generator expression

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

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

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

drange

Create a generator function that kan generate a sequence of decimals similar to the bulit-in function range.

Tip:

- Define a function drange with arguments start, stop, step and endpoint. The endpoint arguments specifies if the endpoit is included or not.
- Give default values 1 for the step and False for endpoint.
  - E.g. **def drange(start, stop, step=1.0, endpoint=False)**
- Create a loop that calculates the numbers from start to end with an increment of step.
  - E.g. **number += step**
- If endpoint is set to true also include the endpoint also.
- You can use standard floats to achieve this but using Decimal will improve the precision. E.g. **from decimal import Decimal**

computrain

Read from a file

- The **open()** built-in function is used to access files
- A file can be opened in different modes: read, write or append
- The keyword **with** specifies a context manager

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)
```

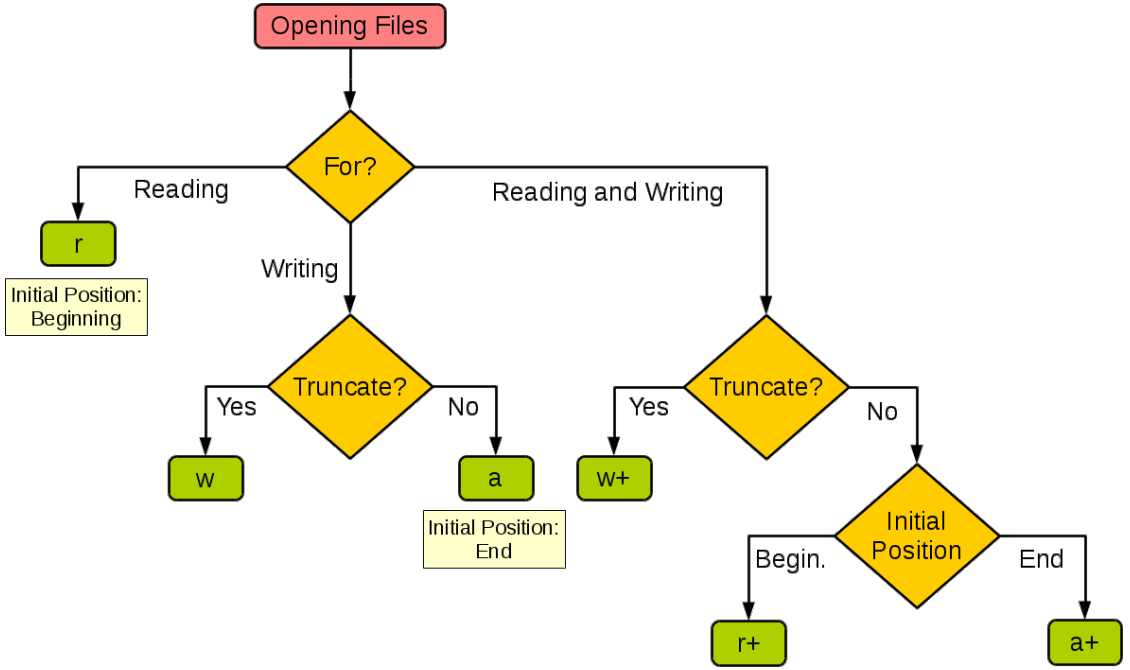
computrain

Reading a CSV file

```
filename = 'ca-500.csv'

with open(filename) as f:

    headers = f.readline().rstrip('\n').split(';')

    for line in f:
        columns = line.rstrip('\n').split(';')

        d = dict(zip(headers, columns))

        if d['city'] in ('Montreal', 'Vancouver'):

            print('{:10} {:15} {:20} {:30}'.format(
                  d['first_name'],
                  d['last_name'],
                  d['city'],
                  d['email']))
```

Write to a file

- Modes: r, w, a, r+, w+, a+, b




```
filename = 'data.txt'

with open(filename, mode = 'w') as f:
    f.write('ID, A, B\n')
    f.write('line 1, 2.0, 10.0\n')
    f.write('line 2, 2.1, 10.0\n')
    f.write('line 3, 2.1, 10.0\n')
```

Read a text file

Create a plain text file with several lines. Then open the file with Python and only print the lines with an e-mail adres.

Tips:

- Create a plain text file with an editor like Notepad.
- Add several lines. Add an e-mail address to some of the lines.
- Save the file in the same directory as your Python files.
- Create a new Python program.
- Specify the filename and location in a variable. E.g. **filename = 'textfile.txt'**
- Open the file with the context manager. E.g. **with open(filename) as f:**
- Read the line of the file on a for loop like **for line in f:**
- Strip the line of trailing newline characters with **line = line.strip()**
- Use an if statement to only print the lines containing an e-mail address
- Optional: Write these lines to an other new file.

computrain

Three types of errors

- Syntax errors — The IDE is your friend
- Runtime errors — Catch the error with try and except
- Functional errors — Use the debugger to find the error

Exceptions

- Run-time errors cause the execution of the code to stop.
- Run-time errors are called **Exceptions**

```
Traceback (most recent call last):
  File "<input>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero
```

- Where are many different types of Exceptions:

StopIteration
SystemExit
StandardError
ArithmeticError
OverflowError
FloatingPointError
ZeroDivisionError
AssertionError
AttributeError
EOFError
ImportError
KeyboardInterrupt
LookupError
IndexError
KeyError
NameError
UnboundLocalError
EnvironmentError
IOError
FileNotFoundError
OSError
SyntaxError
IndentationError
SystemError
SystemExit
TypeError
ValueError
RuntimeError
NotImplementedError

Catching Exceptions

- Exceptions can be caught by using the **try** and **except** keywords.
- Different exceptions can be caught by multiple except blocks
- A **finally** and an **else** block can optionally also be added.

```
try:
    d = 1/0
    d = int("one")

except ZeroDivisionError:
    print("Cannot divide by zero")

except ValueError:
    print("Wrong type")

except:
    print("Another type of error occured")
```

computrain

Throwing an Exception

- An exception can also be thrown from the code with the **raise** keyword.
- An built-in Exception can be thrown or a custom user-defined Exception

```
def calculate_area(width, height):
    if width < 0 or height < 0:
        raise Exception('Invalid argument')
    return width * height

try:
    area = calculate_area(10, -2)

except Exception as err:
    print(err)
```

Foolproof input

Create a function that asks to enter a number between two bounds given as arguments. The function should gracefully handle numbers outside of the bounds and also wrong types of input.

Tips:

- Define a function foolproof_input with argument lower and upper
- In a while loop use input() to get a response from the user
- Turn the input into a number with int()
- Catch the error if the input cannot be converted to a number and give a message.
- Check if the number is between the given bounds. If not give a message.
- Break out of the loop if a correct number was entered.
- Return the number
- Test the function

computrain

Object Oriented Programming

Class

Objects

computrain

Object Oriented Programming

Class

Objects

name

properties

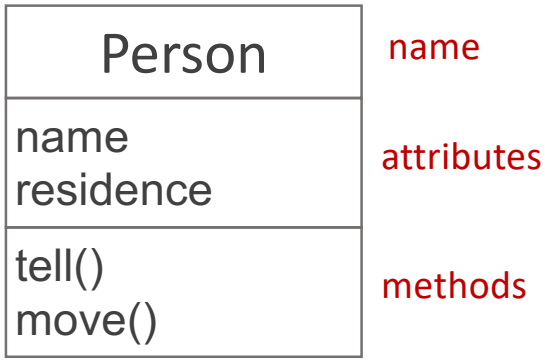
behaviour

computrain

Object Oriented Programming

Class

Objects



computrain

Object Oriented Programming

Class

Objects

Person
name residence
tell() move()

name

attributes

methods

name: Peter
residence: Utrecht

name: Janneke
residence: Amsterdam

naam: Kim
residence: Delft

instances

Object Oriented Programming

Class

instantiation

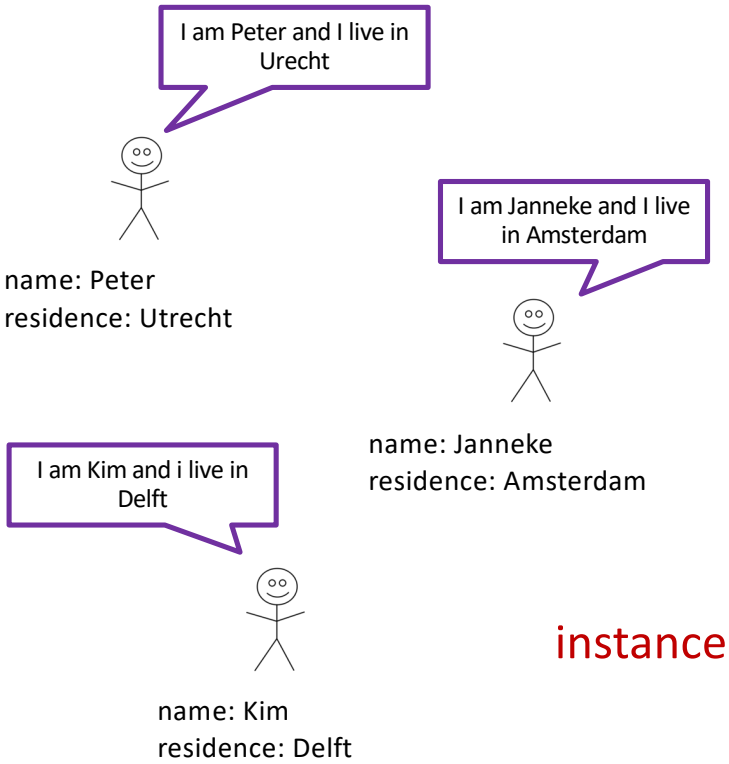
Objects

Person
name residence
tell() move()

name

attributes

methods



instances

computrain

Object Oriented Programming

Class

instantiation

Objects

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

name

attributes

methods

'something'

'something else'

'hello world'

instances

computrain

Object Oriented Programming

Class

Objects

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

name

attributes

methods

[1, 2, 3]

['something', 'something else']

instances

computrain

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'
```

computrain

Methods

- Methods are just like functions with in a class. Methodes 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**.
- Also access methods met the object operator .

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

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

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

computrain

Object initialisation

- When an object in created (instantiated) from a class de **__init__** method is automatically called
- __init__ is called a **magic method**. Where are many 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() )
```

Attributes

- Attributes are typically initialized in the __init__ method.
- Attributes are dynamic and can be assigned a value anywhere.

```
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() )
```

Bankaccount

Create a Bankaccount class, create serveral Bankaccount objects and demonstrate that you can deposit and withdraw amount to the account.

Tips:

- Create a class **Bankaccount**
- Add attributes in the **__init__** method. Attributes should be **holder**, **accountnr** and **balance**.
- Add the methods: **deposit** and **withdraw** that take an amount as argument
- Also add a third method **info** that returns information about the account.
- Instantiate several bankaccount objects and demonstrate the working of the class.

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() )
```

computrain

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

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

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

computrain

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))
```

computrain

Example

```
class Person:

    __slots__ = ('__name', '__residence')

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

    def tell(self):
        return('I am %s and I live in %s' %
               (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() )
```

Class Car

- Create a class named **Car**
- Set several attributes like **make**, **model** 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 kilometers and adds that to the mileage.
- Test your class by instantiating a car and calling the methods

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
```

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__

Vector class

Create a 2d-Vector class. Also add operator overloading for the + sign to add two vectors together.

Tips:

- Build a class called Vector
- Add an __init__ method that takes two arguments: x and y
- Add an __add__ method the define the adding of two vectors.

computrain

Inheritance

- Classes can be reused by 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 avaiable in the child class.
- In the __init__ method of the child class always first call de __init__ method of the parent class with the **super** method

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

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

computrain

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	argv
version_info	exit
path	stdin / stdout / stderr

```
import sys

# get Python version
print(sys.version)

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

os - Operating system interfaces

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

rename	listdir	getenv
remove	path	getpid
mkdir	pipe	getppid
makedirs	scandir	kill
chdir	walk	wait
getcwd	fork	nice
rmdir	system	os.path

```
import os

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

os.path - Pathname manipulations

- **os.path** implements common pathname manipulations

abspath	extsep	isfile	sep
basename	genericpath	islink	split
curdir	getatime	ismount	splitdrive
defpath	getctime	join	splitext
dirname	getmtime	os	stat
exists	getsize	pathsep	sys
expanduser	isabs	realpath	
expandvars	isdir	relpath	

```
import os.path

# set current working directory
os.path.isfile(r'c:\python.exe')
```

computrain

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)
```

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'))
```

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)
```

computrain

math - Mathematical functions

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

computrain

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=5)
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 __del__(self):
        with open('user.pickle','wb') as f:
            pickle.dump(self, f)

    def __init__(self):
        with open('user.pickle','rb') as f:
            self.__dict__.update(pickle.load(f).__dict__)
```

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.
- 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)
```

computrain

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

computrain

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)
```

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)

computrain

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 d in reader:
        print(d['first_name'], d['last_name'])
```

computrain

Reading a CSV file with csv

- **csv** module from Python Standard Library

```
import csv

filename = "ca-500.csv"
colnames = ('first_name', 'last_name', 'city', 'email')

with open(filename) as f:
    reader = csv.DictReader(f, delimiter=';', quotechar='"')

    for d in reader:
        if d['city'] in 'Montreal':

            print('{:10}{:20}{:30}{:30}'.format(
                  *[d[fieldname] for fieldname in colnames]))
```

decimal

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

```
from decimal import Decimal

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

result = float(d1 + d2)
```

computrain

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()
```

computrain

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)
```

computrain

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].*')
```

tempfile

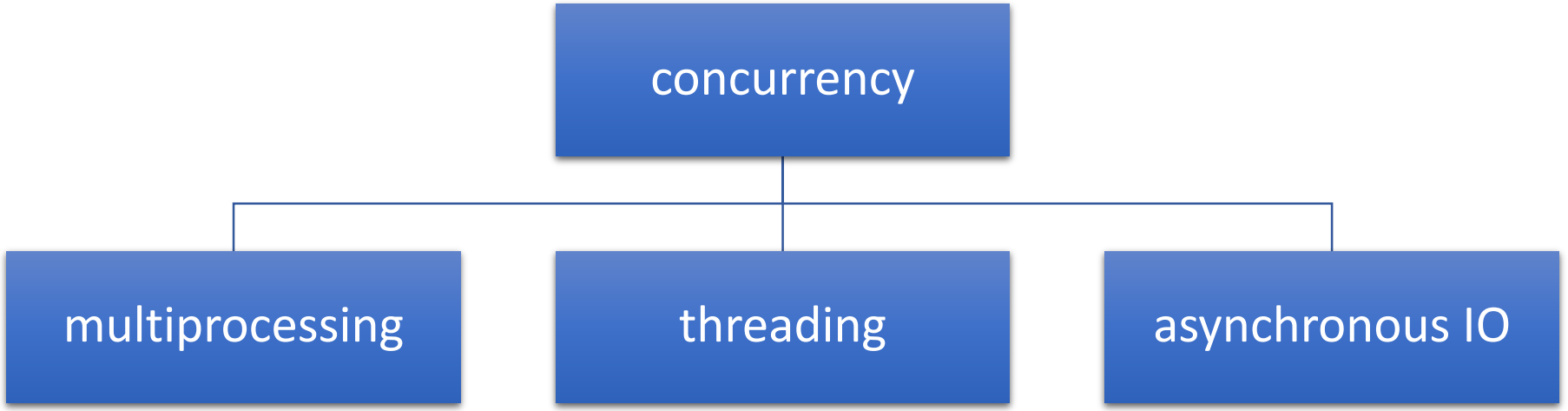
This module generates temporary files and directories.

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()
```

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 potentionally 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.")
```

computrain

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!!!!!')
```

time

- Time-related functions

- time()
- time_ns()
- sleep(secs)
- strftime(format)
- strptime(string, format)

```
import time

t0 = time.time()
time.sleep(2)
t1 = time.time()
d = t1 – t0
print(d)
```

timeit

- Measure execution time of small code snippets.

```
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 > **400000** 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 openpyxl
pip install pyodbc
```

openpyxl – Accessing Excel Workbooks

```
import openpyxl

wb = openpyxl.Workbook()
ws = wb.active

for col_idx in range(1,40):
    col = openpyxl.cell.get_column_letter(col_idx)

for row in range(1,600):
    ws.cell('%s%s'%(col,row)).value = '%s%s'%(col,row)

ws = wb.create_sheet()
ws.title = 'openpyxl'
ws['F5'] = 'use it'
wb.save(filename='using_openpyxl.xlsx')

wb = openpyxl.load_workbook(filename ='myfile.xlsx')

print wb.get_sheet_names()
```

pyodbc - Accessing ODBC databases

PEP 249 -- Python Database API Specification v2.0

```
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()
```

computrain

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.array( [ [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
```

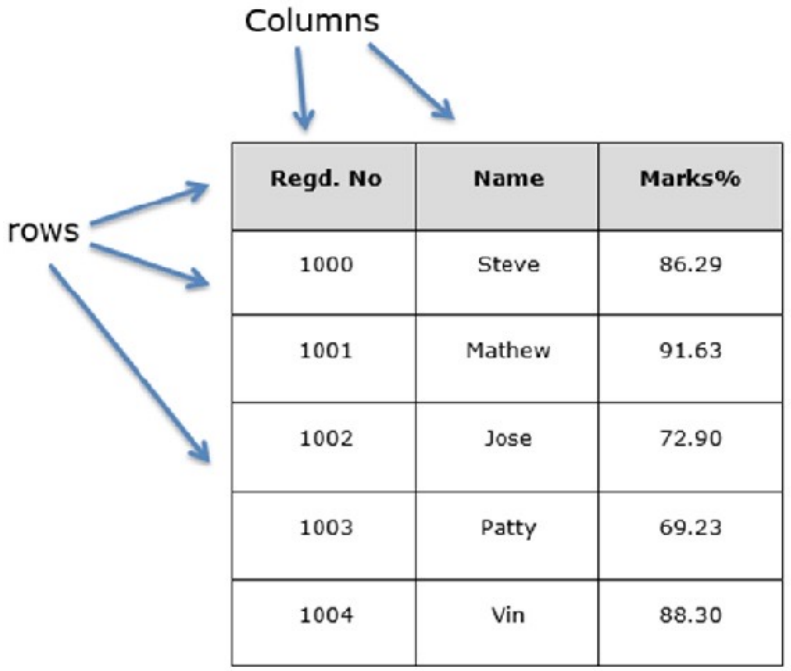
scipy

- 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.



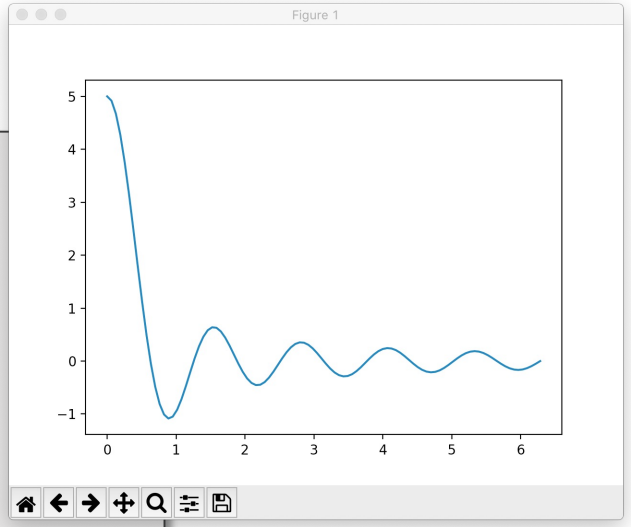
https://pandas.pydata.org/docs/getting_started/10min.html#mi

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()
```



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))
```

Get the weather in New York

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

Tips:

- import **urllib.request** and **json**
- build the url (see https://openweathermap.org/current)
  use: **appid=d1526a9039658a6f76950cff21823aff**
- use the following code to get the response:

```
site = urllib.request.urlopen(url)
response = str(site.read(), encoding='UTF-8')
```

- use json to decode the response into a Python dictionary
- get and print the temperature

django

MVC Framework

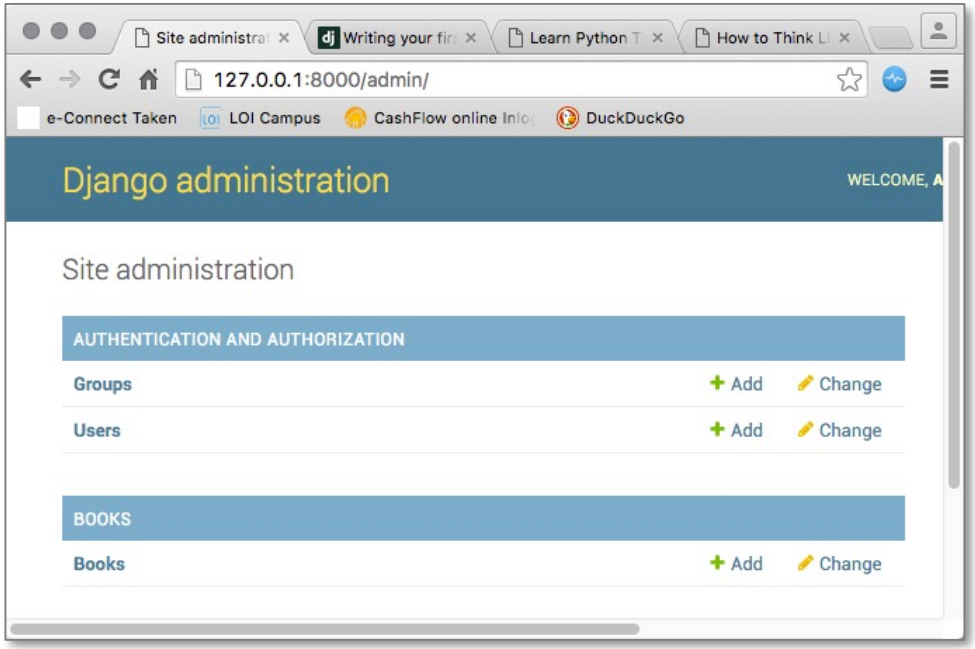
Models

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
```

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

Virtual Environment

- Seperated enviroments

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

- Requirements file

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

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

computrain

EAFP versus LBYL

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