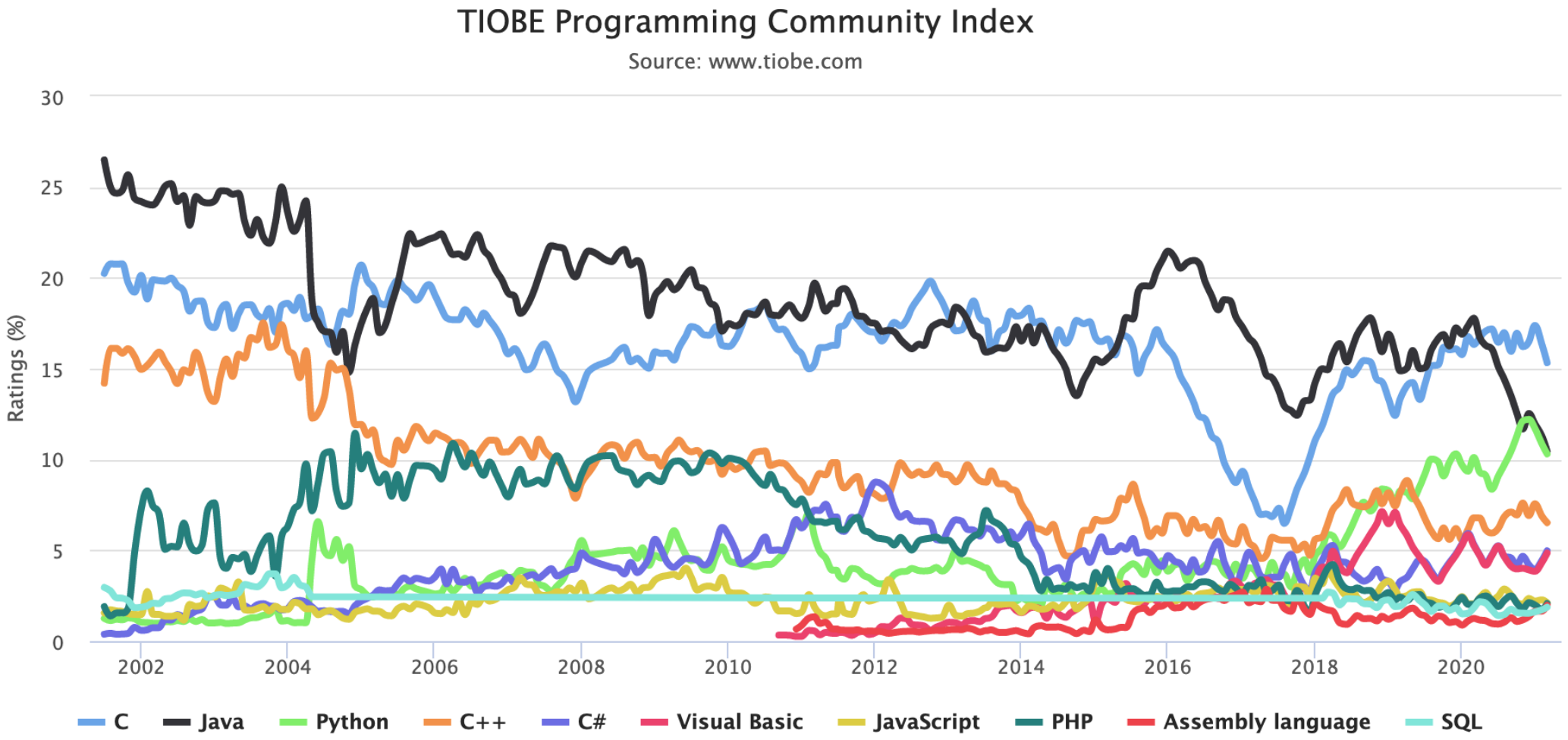
computrain

Python Fundamentals

Popularity



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

Program

Introduction
Variables en operators
Strings
Program Flow

Datastructures
Functions

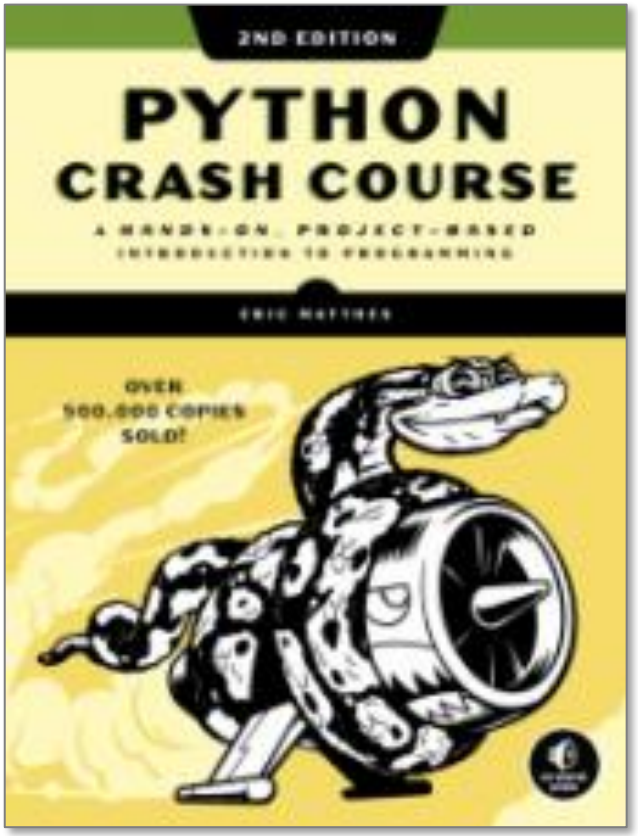
Book

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
- Python 2 End of Life in 2020
- The Zen of Python (import this)
- Pythonic, Pythonista, Idiomatic Python

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

Python features

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

Python applications

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

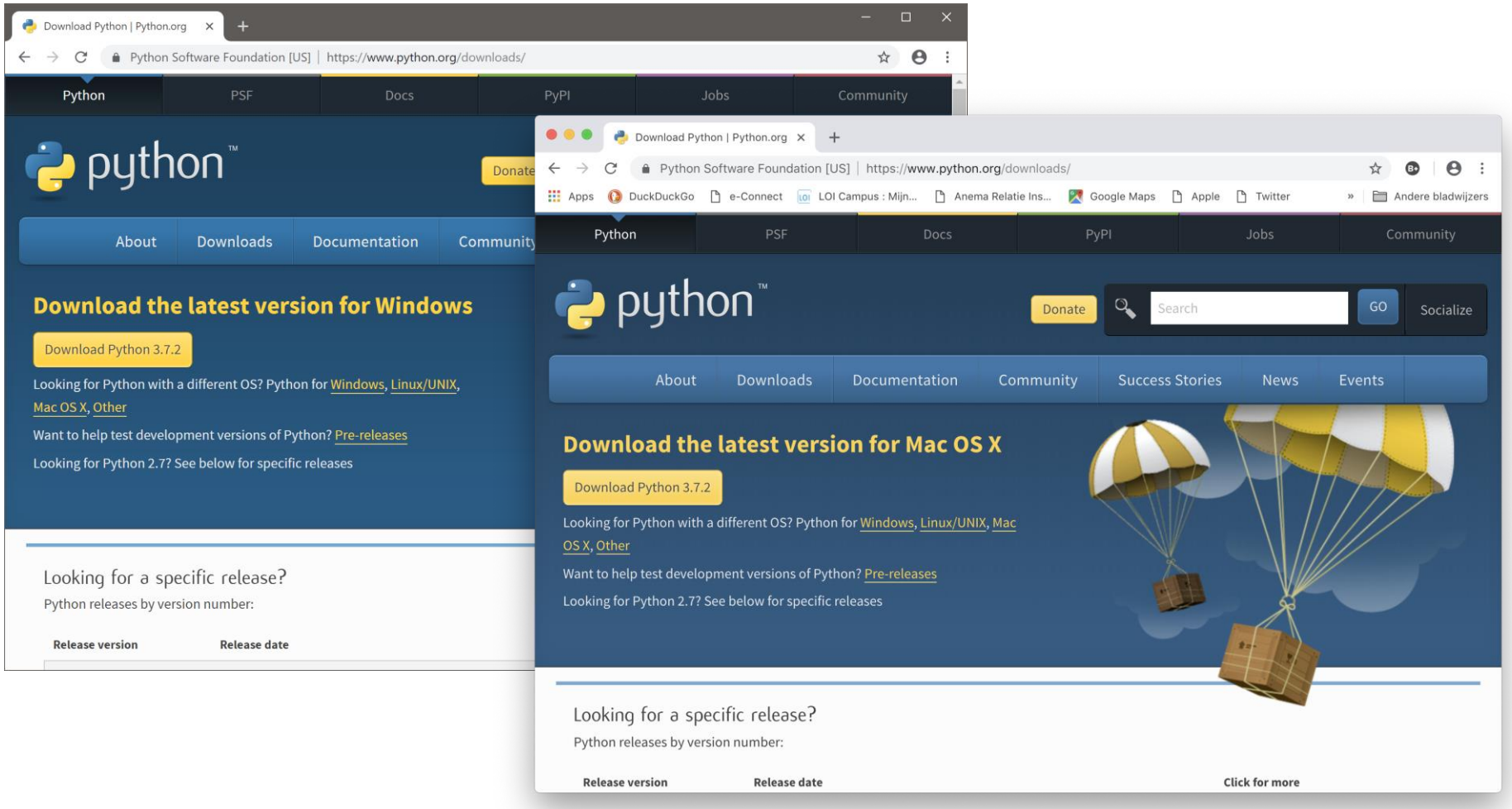
computrain

Links

- https://www.python.org/
- https://nl.wikipedia.org/wiki/Python_(programmeertaal)
- https://www.w3schools.com/python/python_reference.asp
- http://www.pythontutor.com/visualize.html#mode=edit

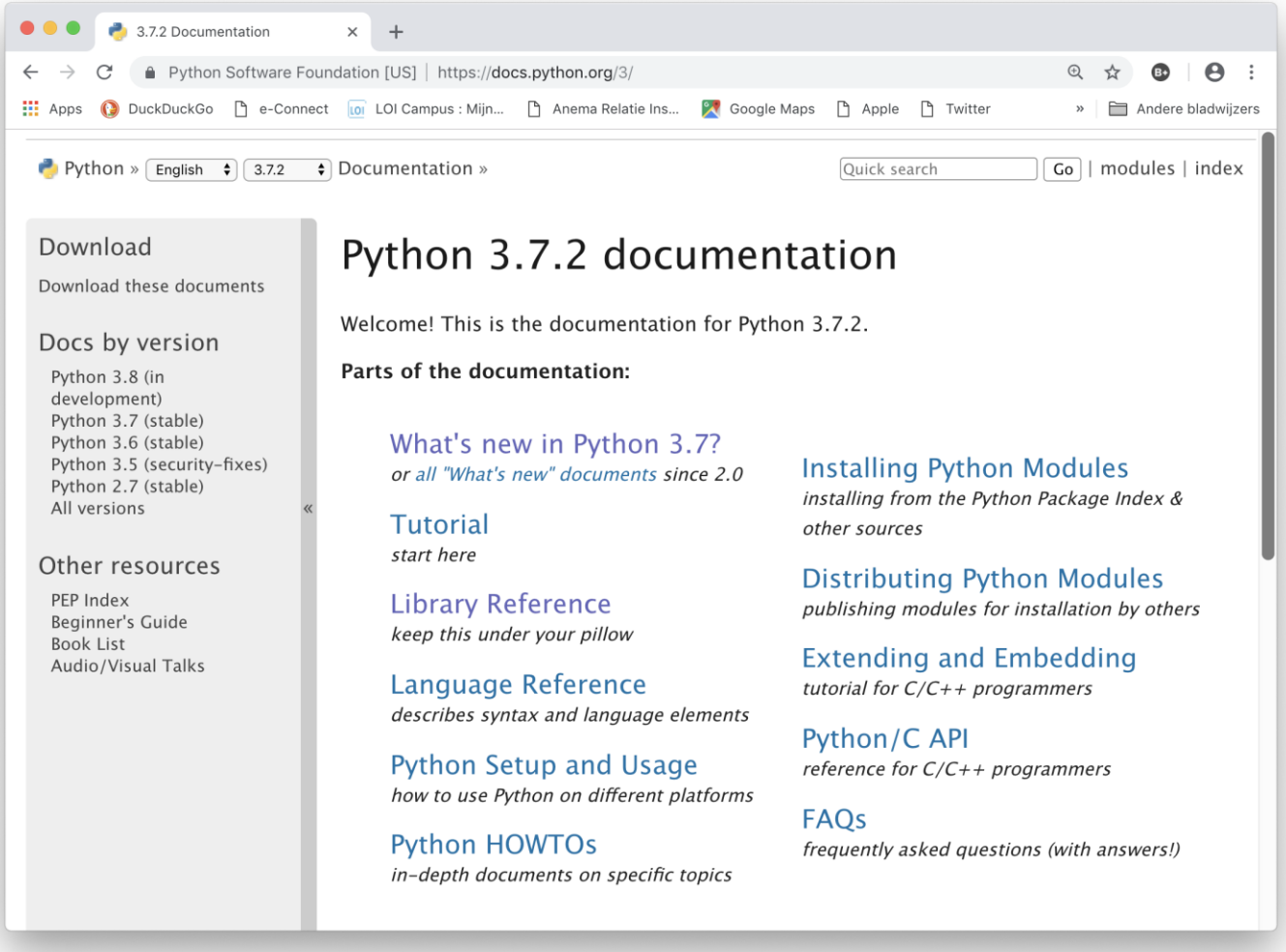
Installation Python

- Python 3.x

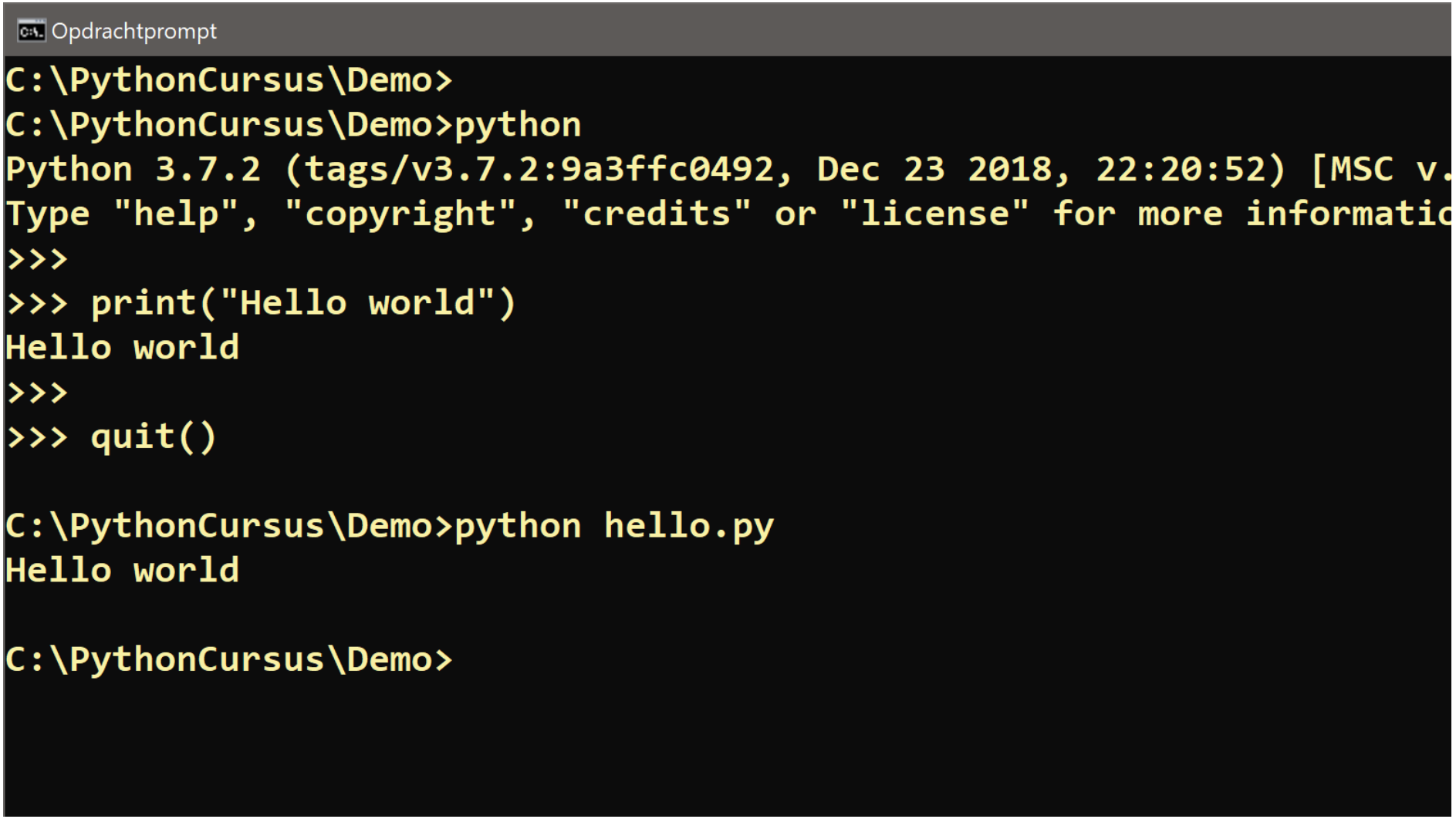


Documentation

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



Python Prompt



IDLE

- Interactive Python Prompt
- Python editor

Python 3.7.0 Shell

```
Python 3.7.0 (default, Jun 28 2018, 07:39:16)
[Clang 4.0.1 (tags/RELEASE_401/final)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello")
Hello
>>>
```

hello.py - /Users/peter/Computrain/Python NEW/sandbox/hello.py (3.7.0)

```
# Commentaar
print("Hello")
```

Other Editors & IDE's

Code Editors

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

Integrated Development Environments

- PyCharm
- VS Code with Python extension
- Spyder
- JupyterLab

Print function

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

The **print()** function prints tekst 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 then 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 on 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

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

https://www.w3schools.com/python/python_ref_keywords.asp

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()	

https://docs.python.org/3/library/functions.html

Python Standard Library

Bundled with core Python distribution

- 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/module/package 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 )

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 searched in the current directory first.

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

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
```

Democodes

Variables:

- DemoNaamgevingVariabelen.pyt
- DemoVariabelen.py
- DemoVariabelen2.py
- DemoVariabelen3.py
- DemoVariabelenMetAdres.py
- DemoVariabelenMetAdressen.py

Modules:

- DemoImporteerEigenModules.py
- DemoimporteerEigenModules2.py
- DemoImportMath.py
- DemoMath.py
- DemoModule1.py
- DemoModule2.py

Comments:

- DemoComments.py

Print and input functions:

- DemoEersteScript.py
- DemoFirst.py
- DemoInput.py
- DemoInvoerInput.py
- DemoPrint.py
- DemoPrintAndInput.py
- first.py
- first2.py
- MijnEerstePythonFile.py
- MijnEersteScript.py
- MijnTweedeScript.py
- VraagNaarLeeftijd.py

Exercise 1.1

Python prompt

Experiment with the python prompt.

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

Exercise 1.2

Hello

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 the file as first.py
6. Run the file.
7. Change the file to first ask for your name and store the result in a variable name using the input() function.
8. Use the print function to print "Hello [name]"
   (if Albert is your name, Albert is shown.)
9. Save and run the file.

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
- A value is evaluated to **False** if the value is:
  - 0
  - 0.0
  - 0j
  - ''
  - ()
  - []
  - {}
  - False
  - None
- Otherwise **True**
- bool()

```
end_of_loop = True
is_even = number % 2 == 0
is_even = bool(number % 2)
```

String

- A string can be specified with single quotes '
- or with double quotes "
- or even with triple quotes """

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

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

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

Numeric operators

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

- Use brackets ( ) to specify precedence

```
result = (56 + 4) * 821 – 10 ** 2 // 10

result          # => 49250
```

Math library

- import **math**

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

Random library

- import **random**

seed()	randint()	random()	shuffle()
	randrange()	uniform()	choice()
		normalvariate()	choices()
		gauss()	sample()

```
import random

random.seed(999)

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

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 and 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

Democodes

Boolean types and if statements:

- DemoBool.py
- DemoBool2.py
- DemoComparisonOperators.py
- DemoIfs.py

Libraries in Random and Math:

- DemoMath.py
- DemoRandom.py
- DemoStdLibMath.py

Strings and numeric types:

- DemoNumericTypes.py
- DemoStringFormatting.py
- DemoStringInt.py
- DemoStrings.py
- DemoStrings2.py
- DemoStrings3.py
- DemoStrings4.py
- DemoStringsConcat.py
- DemoStringSlicing.py
- DemoStringUnicode.py

Exercise 1.3

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 result
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.

Exercise 1.4

Dimensions of 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**() and assign to a variable **r**.
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.

Exercise 1.5

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 **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'....' since Python 3.6

```
name = 'Guido'
age = 62

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

computrain

String methods

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

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

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( 'Pati\u00EBnt' )
print( '\u2665' )
print( '\u20AC' )    # Euro sign €
```

Democodes

Boolean types and if statements:

- DemoBool.py
- DemoBool2.py
- DemoComparisonOperators.py
- DemoIfs.py

Libraries in Random and Math:

- DemoMath.py
- DemoRandom1.py
- DemoRandom2.py
- DemoStdLibMath.py

Strings and numeric types:

- DemoNumericTypes.py
- DemoStringFormatting.py
- DemoStringInt.py
- DemoStrings.py
- DemoStrings2.py
- DemoStrings3.py
- DemoStrings4.py
- DemoStringsConcat.py
- DemoStringSlicing.py
- DemoStringUnicode.py

Exercise 1.6

Working with strings

Experiment with strings.

1. Create a new python file.
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. Print the first three characters by using slicing.
6. Check if the text ends with a question mark.
7. Print the text in lowercase with all spaces replaced by an underscore by using the method **replace()**. This is called **snake_case.**

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'
```

computrain

Looping with while

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

- Semi-colon :
- Indenting
  - 4 spaces

- Some way or another the condition has to be set to **False** to stop the loop

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

Looping with for

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

- Semi-colon :
- Indenting
  - 4 spaces

```
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

Function 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 and 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)
```

Democodes

Break and Continue

- DemoBreakContinue.py
- DemoBreakContinues.py
- DemoContinue.py

For and While loops:

- DemoForLoop.py
- DemoForLoops.py
- DemoForLoops2.py
- DemoForLoops3.py
- DemoLoops.py
- DemoLoops2.py
- DemoOneindigeLoopPythonic.py
- DemoRanges.py
- DemoWhile.py
- DemoWhileLoop.py
- DemoWhileLoopMetLetter.py
- DemoWhileTrue.py
- DemoWhileTextZolang.py

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"

Exercise 1.8

Count vowels

- Get some text from input and put this in a variable
- Loop through the vowels ['a', 'e', 'i', 'o', 'u', 'y']
- Count the number of occurances of each vowel 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

```
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 complete text contains 929 characters.
The text contains 291 vowels.
```

Datastructures

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

computrain

List

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

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

computrain

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

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

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

Function range()

- range(stop)
- range(start, stop)
- range(start, stop, step)

- generator function => just in time

```
range(10)          # 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
range(3, 9)        # 3, 4, 5, 6, 7, 8
range(3, 9, 2)     # 3, 5, 7
```

computrain

Tuple

- An inmutable list of elements
- Similar to a list
- Round brackets ()
- Or no brackets at all!
- 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,) # an element
tuple6 = 'NL','B','L'
```

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!")
```

index and slicing

- index [index]
- slicing [start:stop] of [start:stop:step]
  stop is not included!
- 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']
```

Sequence type operations

- concatenation **+**
- **index()** method
- **count()** method
- built-in functions, e.g. len(), min() en max()

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

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

print( len(list3) ) # 6
print( min(list3) ) # 'a'
print( max(list3) ) # 'z'
```

Unpacking

- Lists and tuples can be '**unpacked**' into multiple 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 # swapping contents of v1 and v2
```

split and join

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

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

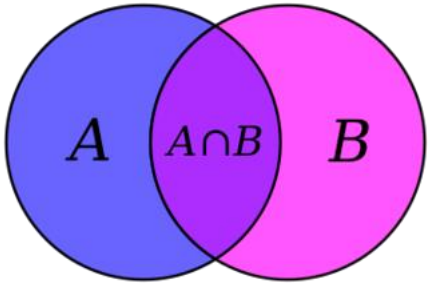
```
sentence = "number of cars"

words = sentence.split()
print(words)

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

Set

- A set of unique elements without any order.
- Function **set()** is used to make a set from other collections
- Curly brackets **{}**




```
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}
```

computrain

Dict

- A collection of **key** – **value** pairs
- The **dict()** function creates a dictionary

```
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'
```

Comprehension

Create a datastructure from an other datastructure

List comprehension	[i*i for i in range(10)] [i*i for i in range(10) if i > 5] [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}

```
[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 ['guido', 'tom', 'albert']
```

Other datastructures

- array
- namedtuple
- deque

```
import namedtuple

Point = namedtuple('Point', ['x', 'y'])
p1 = Point(11, 22)
p2 = Point(x=11, y=22)

p1.x # => 11
p1.y # => 22
```

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

Exercise 2.1

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)**

Exercise 2.2

Occurance of words

- Get an arbitrary piece of text from internet
- Create a python script that reads the complete tekst with **s = input()**
- Convert to lowercase and remove dots and commas
  - use **s.lower().replace('.', '').replace(',', '')**
  - or **s.lower().translate(str.maketrans('', '', '.,!?()[]'))**
  - or with a regular expresson **re.sub('[^a-z\s]', '', s.lower())**
- Split the text into words with **text.split()**
- Create a set of unique words
- For each unique word count the number of occurances
- Store the results in a dictionary: **d[word] = n**
- Print the results: **for word, n in d.items()**

Exercise 2.3

Password generator

Generate a password of at least 6 characters
with at least 1 capital, 1 lowercase, 1 number and 1 special character.

Tips:

- Start with 4 string with character families.
  - E.g. capitals = 'ABCDEF..', numbers = '0123456789'
- Use random library to select a sample from these strings. The results are lists.
  - import random
  - part1 = random.choices(capitals, k=3)
- Concatenate the lists together.
  - characters = part1 + part2 + part3 + part4
- Shuffle the order of the elements with random.shuffle(characters).
- Turn the list of characters into a string with join():
  - password = ''.join(characters)
- print the generated password.

Exercise 2.4

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 = [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)]

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

computrain

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 within a function have a local scope. These are only valid withn the function.
- Arguments of a function also have local scope.

computrain

Democodes

Built-in functions

- DemoBuiltinFunctions.py

Datastructures and more:

- DemoDict.py
- DemoDict2.py
- DemoComprehension.py
- DemoIndexSlicing.py
- DemoLists.py
- DemoLists2.py
- DemoLists3.py
- DemoLists4.py
- DemoLists5.py
- DemoLists5.py
- DemoLists6.py

Datastructures and more:

- DemoListsAppendAndExtend.py
- DemoMatrix.py
- DemoRanges.py
- DemoScopesVariables.py
- DemoSets.py
- DemoSlicing.py
- DemoSplitStringLst.py
- DemoSplitJoin.py
- DemoTuple.py
- DemoTuples.py

Exercise 2.5

Banner

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

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

Tips:

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

Exercise 2.6

Range of floats

The range function can only generate integers. Create a generator function that kan generate a sequence of floats 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**

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
```

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)
```

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))
```

Democodes

Built-in functions

- DemoBuiltinFunctions.py

Functions:

- DemoFunctions.py
- DemoFunctions1.py
- DemoFunctions2.py
- DemoFunctions3.py
- DemoFunctions4.py
- DemoFunctions5.py
- DemoFunctions10.py
- DemoFunctionsScope.py
- DemoGeneratorFunctions.py
- DemoGeneratorFunctionsLes.py
- DemoGeneratorFunctionsRandom.py
- DemoMapFunction.py

Exercise 2.7

Sort a list

- Enter a piece of tekst and split into words
- Use the **sorted** function to sort these words
- Create a function called number_of_vowels to count the number of vowels
  - tip: **sum([word.count(v) for v in 'aeiou'])**
- Use this function to sort the list on number of vowels
  - tip: **sorted(words, key=number_of_vowels)**

Exercise 2.8

Fibonacci Generator

- Write a Python generator function, called fibonacci_generator(…), that generates Fibonacci numbers up to a specified limit.
- The generator must produce consecutive Fibonacci numbers until the next number in the sequence exceeds the limit.

Exercise 2.9

Prime Generator

- Write a Python generator function, called prime_generator(…), that generates prime numbers up to a specified limit.

- Tips:
  - Make function is_prime() to determine, if the number is a prime number or not.

Structure of a Python script

- Most Python scripts have the following structure from top to bottom:

- Global variables
- Functions
- Main function to start the script

Example will be shown in PythonFileStructure.py.

Python Fundamentals

Thank you for your attention!

Goodbye!