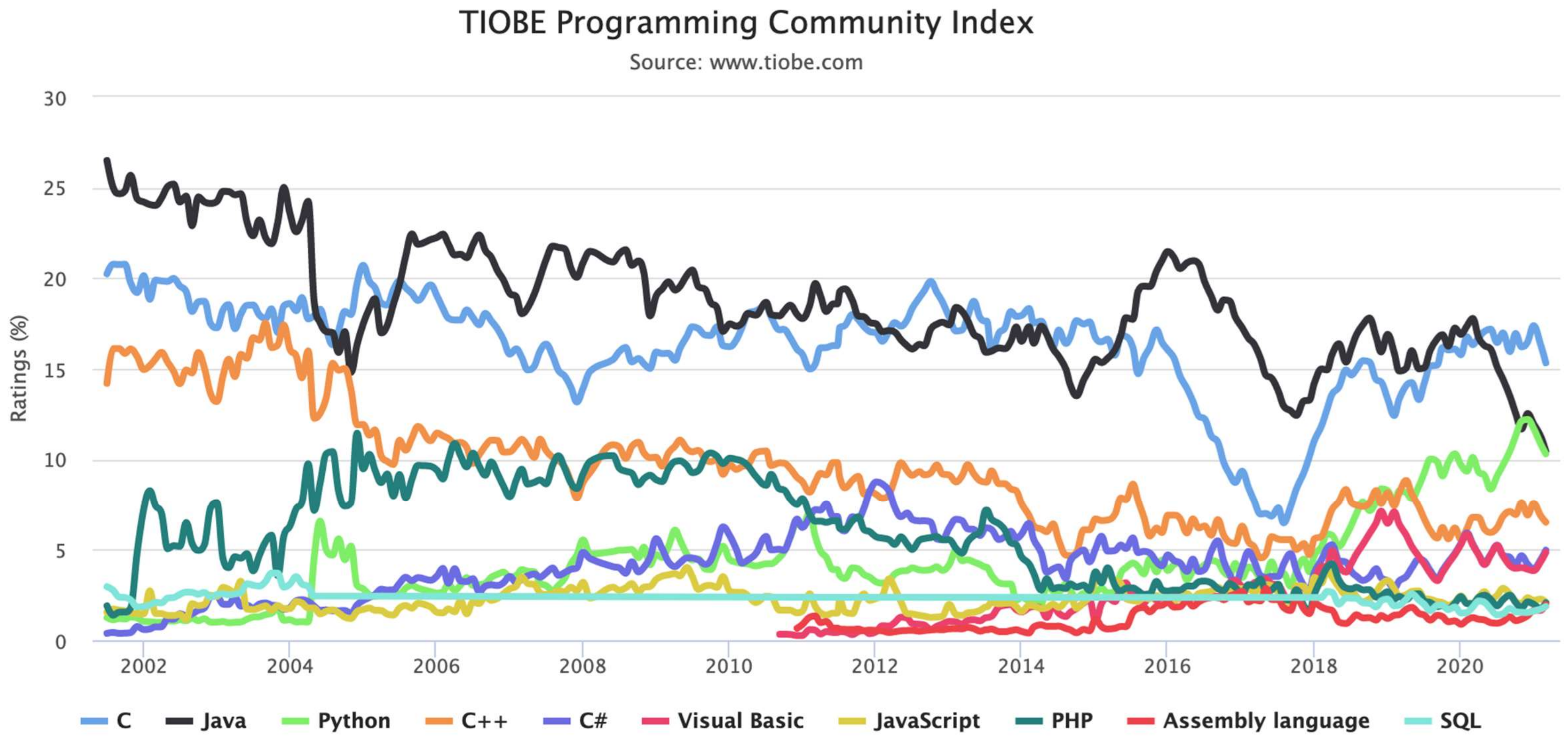
# Python Fundamentals

https://github.com/TesselHaagen/PythonTraining-11-13-dec

# Popularity


TIOBE Programming Community Index
Source: www.tiobe.com
Ratings (%)
30
25
20
15
10
5
0
2002
2004
2006
2008
2010
2012
2014
2016
2018
2020
C
Java
Python
C++
C#
Visual Basic
JavaScript
PHP
Assembly language
SQL

computrain

## Program

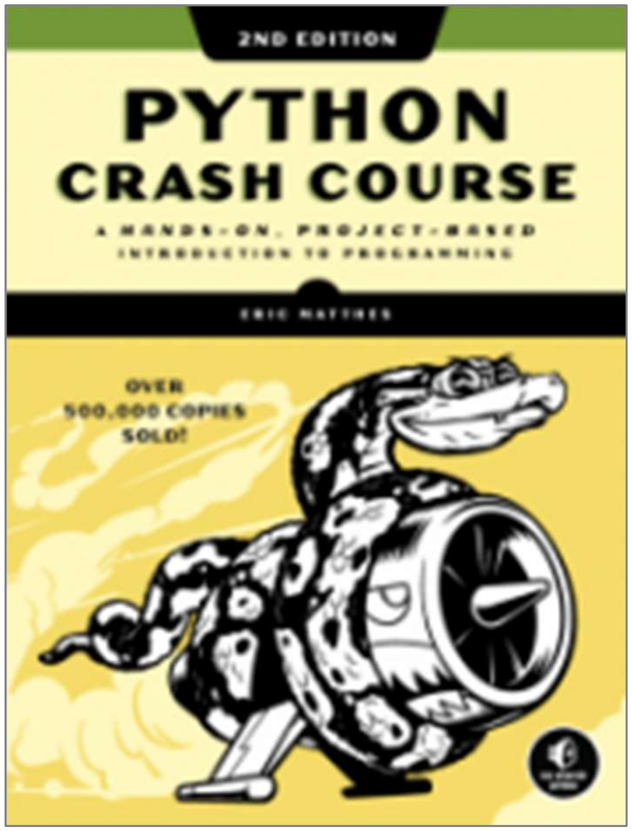
**Day 1**

Introduction
Variables en operators
Strings
Program Flow

**Day 2**

Datastructures
Functions

# Book


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

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

## 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

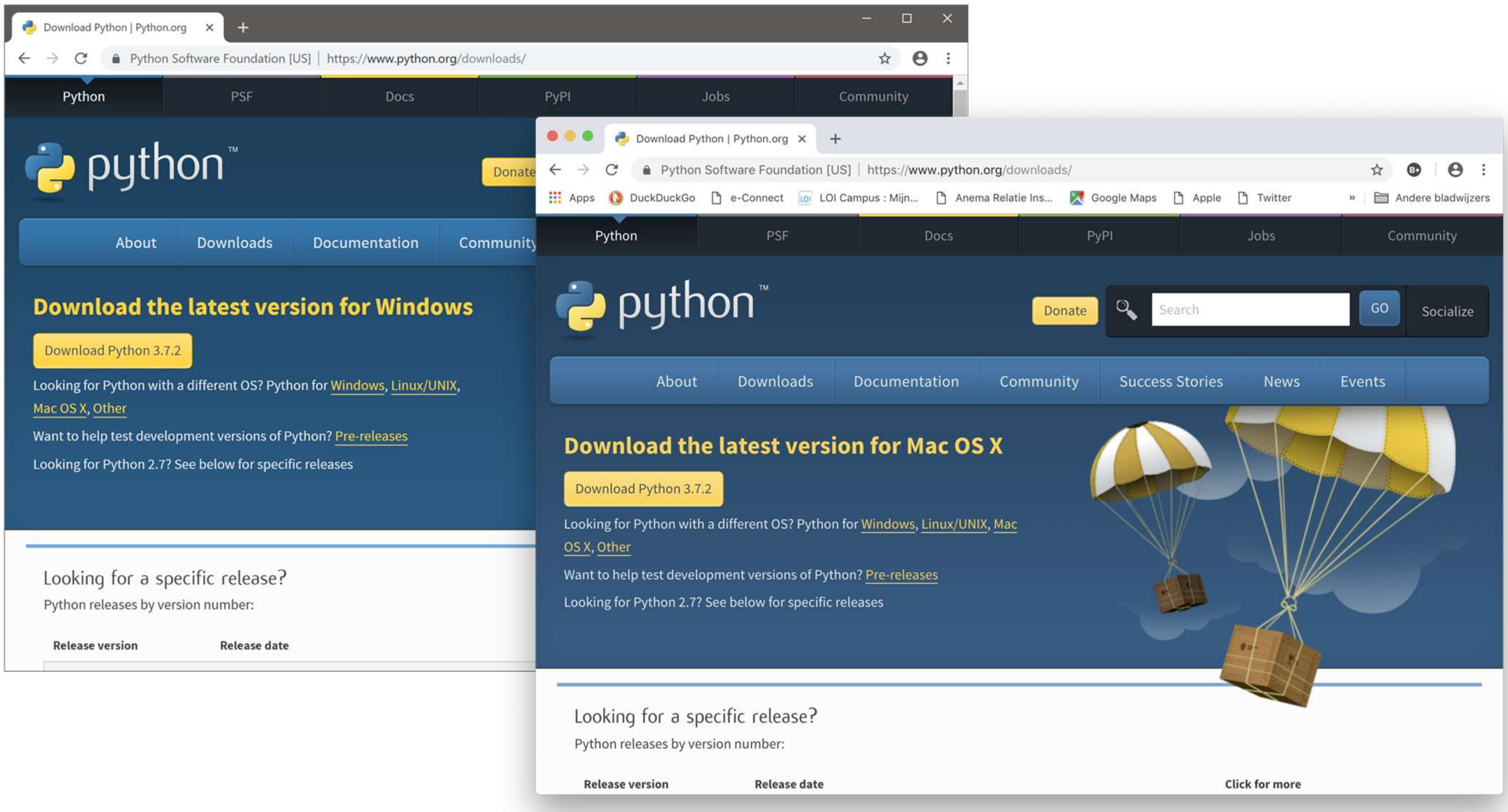
- General Purpose
- Scripting
- Data processing
- Scientific
- Desktop GUI
- Web

## 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



**Download the latest version for Windows**

Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

Want to help test development versions of Python? Pre-releases

Looking for Python 2.7? See below for specific releases

Looking for a specific release?

Python releases by version number:

**Download the latest version for Mac OS X**

Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

Want to help test development versions of Python? Pre-releases

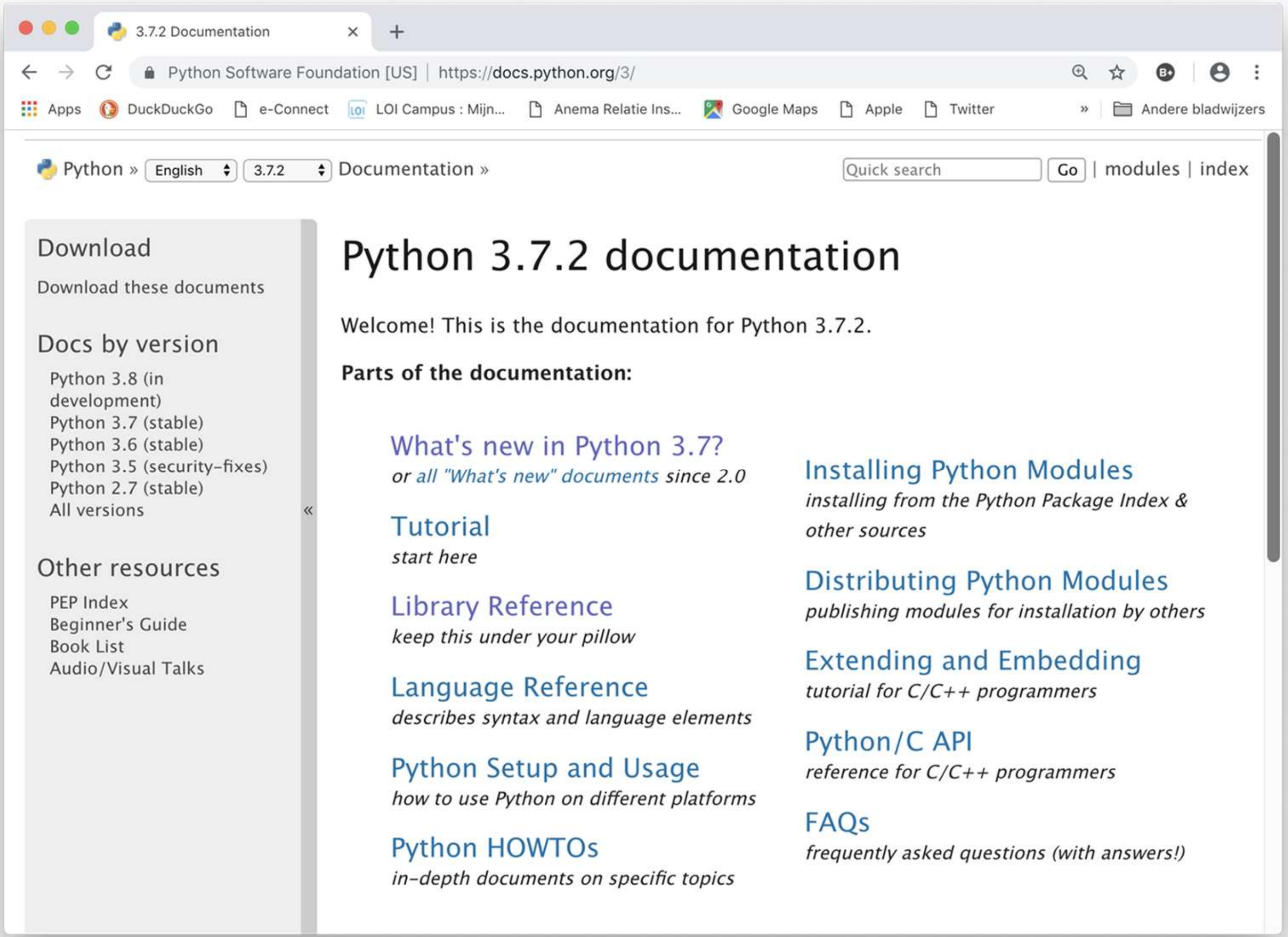
Looking for Python 2.7? See below for specific releases

Looking for a specific release?

Python releases by version number:

## Documentation

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


3.7.2 Documentation

Python Software Foundation [US] | https://docs.python.org/3/

Apps DuckDuckGo e-Connect LOI Campus : Mijn... Anema Relatie Ins... Google Maps Apple Twitter » Andere bladwijzers

Python » English 3.7.2 Documentation » Quick search Go | modules | index

Download

Download these documents

Docs by version

Python 3.8 (in development)
Python 3.7 (stable)
Python 3.6 (stable)
Python 3.5 (security-fixes)
Python 2.7 (stable)
All versions

Other resources

PEP Index
Beginner's Guide
Book List
Audio/Visual Talks

Python 3.7.2 documentation

Welcome! This is the documentation for Python 3.7.2.

**Parts of the documentation:**

What's new in Python 3.7?
*or all "What's new" documents since 2.0*

Tutorial
*start here*

Library Reference
*keep this under your pillow*

Language Reference
*describes syntax and language elements*

Python Setup and Usage
*how to use Python on different platforms*

Python HOWTOs
*in-depth documents on specific topics*

Installing Python Modules
*installing from the Python Package Index & other sources*

Distributing Python Modules
*publishing modules for installation by others*

Extending and Embedding
*tutorial for C/C++ programmers*

Python/C API
*reference for C/C++ programmers*

FAQs
*frequently asked questions (with answers!)*

# 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 |

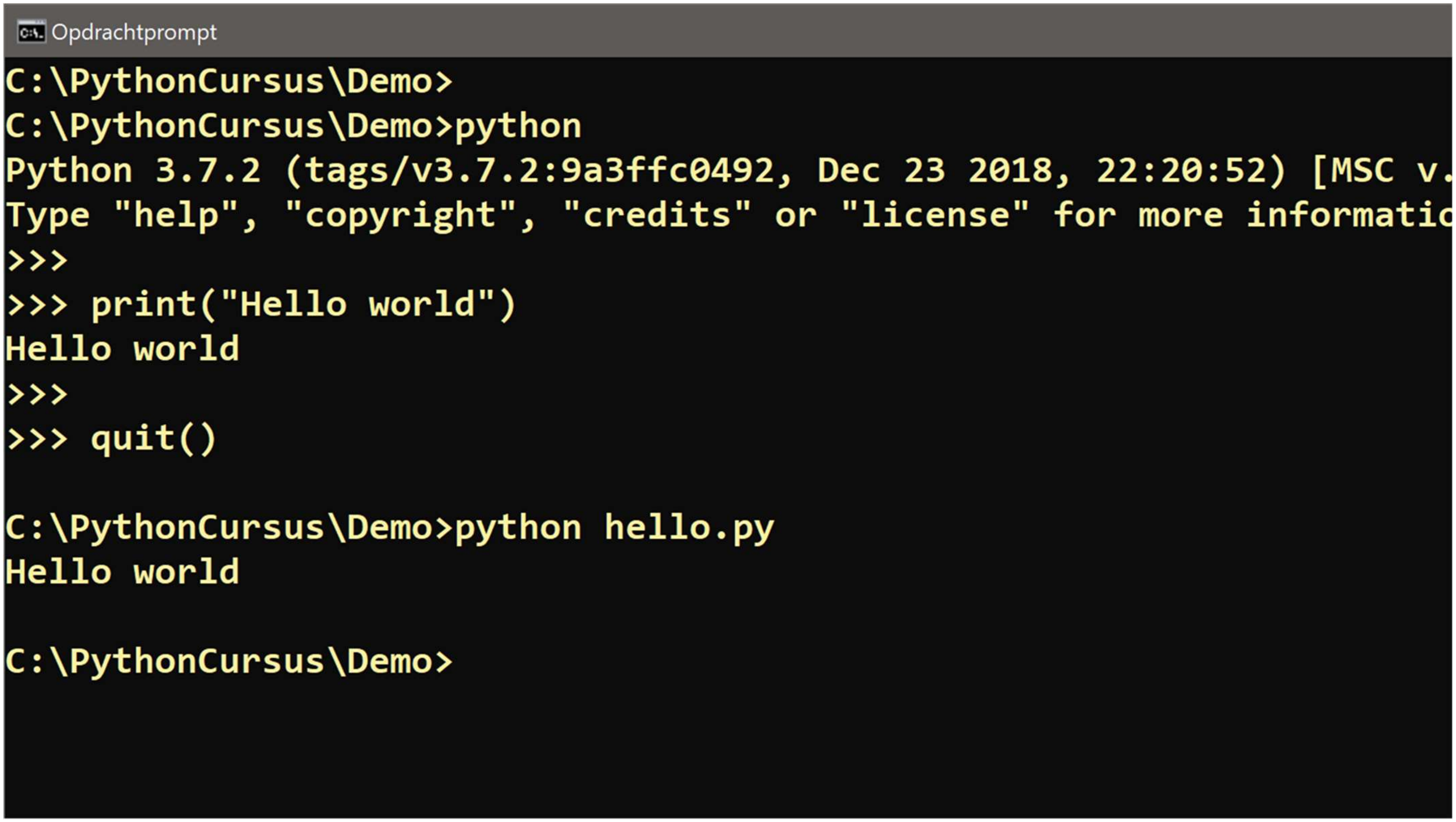
https://docs.python.org/3.0/whatsnew/3.0.html

## 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
```

# Python Prompt



Opdrachtprompt

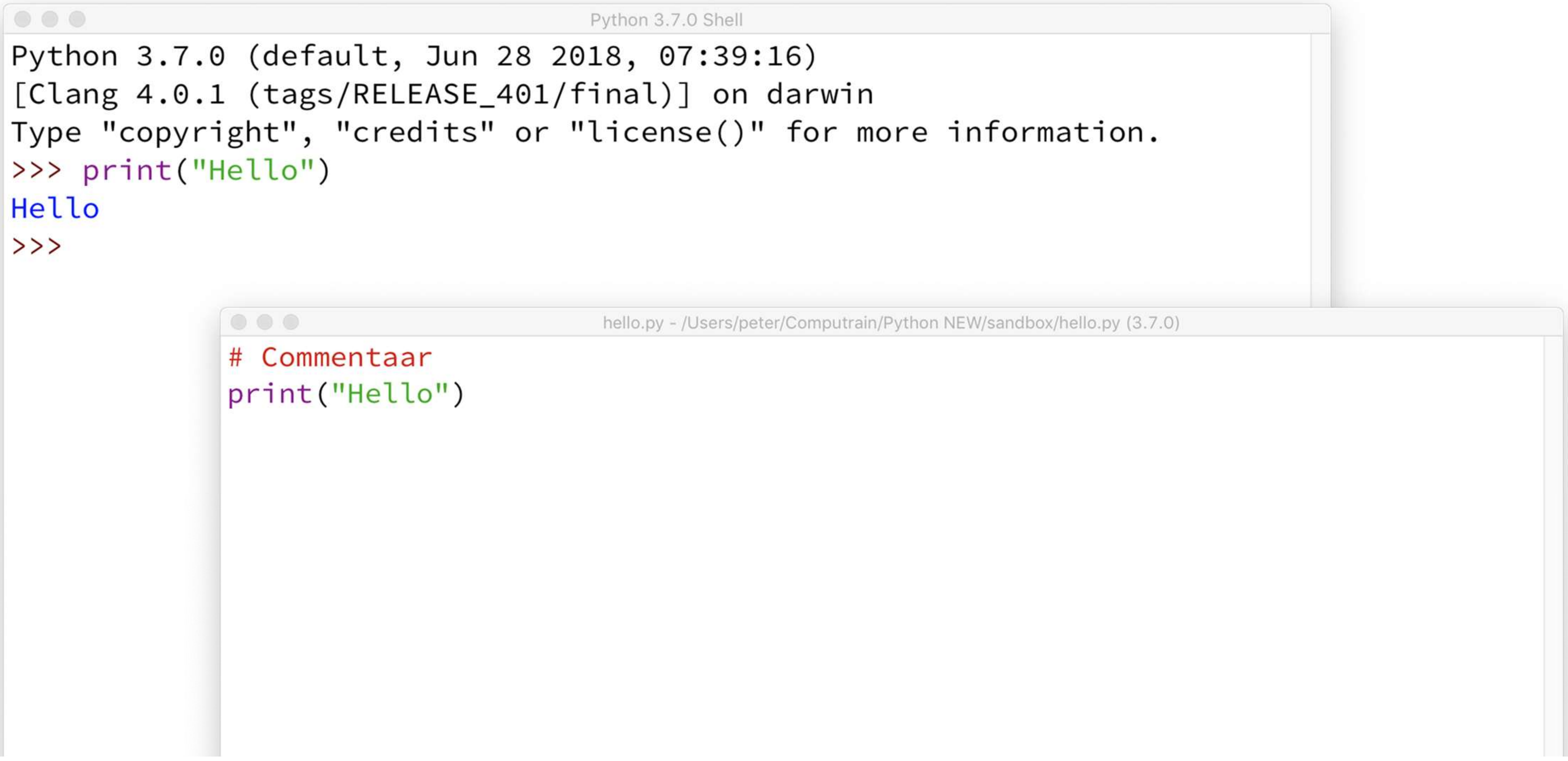
```
C:\PythonCursus\Demo>
C:\PythonCursus\Demo>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.
Type "help", "copyright", "credits" or "license" for more informatio
>>>
>>> print("Hello world")
Hello world
>>>
>>> quit()

C:\PythonCursus\Demo>python hello.py
Hello world

C:\PythonCursus\Demo>
```

## IDLE

- Interactive Python Prompt
- Python editor



```
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

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 Albert" (if Albert is your name).
9. Save and run the file

## 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
```

## Keywords

False

None

True

and

as

assert

async

await

break

class

continue

def

del

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

# 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
```

# Numeric Types

|  |  |
|---|---|
| • whole numbers | int() |
| • decimal numbers | float() |
| • complex numbers | complex() |

```
i1 = 8
i2 = 734927349872398742398 74
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:
- Otherwise **True**
- bool()

- 0
- 0.0
- 0j
- ''
- ()
- []
- {}
- False
- None

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

## 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)
```

# 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
acosh
asin
asinh
atan
atan2
atanh
ceil
copysign
cos
cosh
degrees
```

## Random library

- import **random**

seed()

randint()

randrange()

```
import random

random.seed(999)

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

random()

# 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 |

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 =** $\pi r^2$
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' )
```

## String methods

capitalize
casefold
center
count
encode
endswith
expandtabs
find
format

format_map

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

isalpha

# 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 €
```

Exercise 1.6

## Working with strings

Experiment with strings.

1. Create a new python file. E.g. strings.py
2. Ask the user to input some tekst and store the response in a variable t
3. Print the tekst 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 tekst ends with a question mark
7. Print the tekst 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'
```

## 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)
```

## 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 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 1.8

# 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 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.
```

Exercise 1.9

## 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
```

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

## Datastructures

- Sequence types
  - list
  - tuple

- Set types:
  - set

- Dictionary types
  - dict

## 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))
```

# List modification methods

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

```
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()
min()
max()
sum()
sorted()
map()
filter()

all()
any()

```
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
```

## 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

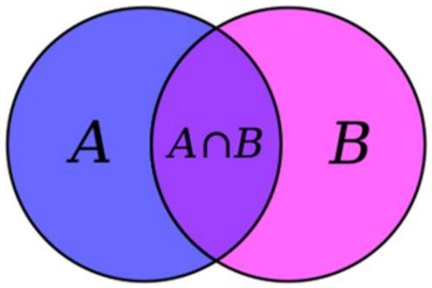
```
sentence = "number of cars"

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
```

# Set methodes

add
clear
copy
difference
difference_update
discard
intersection
intersection_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
- 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')
```

## 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")
```

## 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.

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
```

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
- Use this function to sort the list on number of vowels
    - use ‘key’ argument of sorted

## Variadic arguments

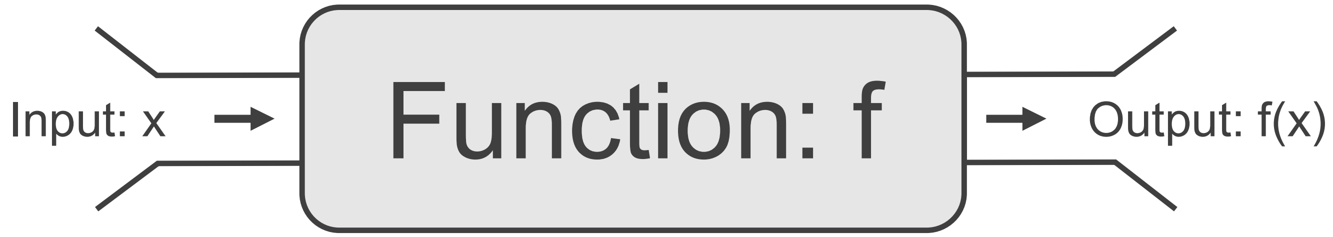
- Variadic arguments can take any number of arguments
- Use a * character
- The arguments are collected in a list

```
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)
```

# Functional programming

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



# 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))
```

# 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
readline
readlines
write
seek
close

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

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

# Write to a file

- Modes: r, w, a, b

```
filename = 'data.txt'

with open(filename, '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')
```

Exercise 2.8

## Writing to and reading from a file

- First create a file with open and write mode
- Write a header line to the file. E.g. 'ID,var1,var2,var3'
- Write a couple of lines to the file. E.g. '1001, 5, 1.23, "Y"'

- Then create a new program
- Open the file in read mode
- Read the header (first line)  and split into a list of headers
- For each line split the line into values
- Create a dictionary with the header and the values using zip()

- Filter on one off the fields and print the lines

Exercise 2.9

# Read a CSV file

Filter lines from a CSV file

Tips:

- Get the ca-500.csv file
- Open the file within a context manager with the keyword **with**
- Read the first line. The header.
- In a for loop through all lines.
- For each line strip the newline character from the end with **strip**
- Split the line into a list of values with **split**
- Only select lines with city 'Montreal'
- Print firstname, lastname, city and email

# Solution Read 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']))
```

# 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]))
```

# 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
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")
```

## EAFP versus LBYL

**EAFP**

**Easier to ask for forgiveness than permission**. This common Python coding style assumes the existence of valid keys or attributes and catches exceptions if the assumption proves false. This clean and fast style is characterized by the presence of many try and except statements. The technique contrasts with the LBYL style common to many other languages such as C.

**LBYL**

**Look before you leap**. This coding style explicitly tests for pre-conditions before making calls or lookups. This style contrasts with the EAFP approach and is characterized by the presence of many if statements.

In a multi-threaded environment, the LBYL approach can risk introducing a race condition between "the looking" and "the leaping". For example, the code, if key in mapping: return mapping[key] can fail if another thread removes *key* from *mapping* after the test, but before the lookup. This issue can be solved with locks or by using the EAFP approach.

## 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)
```

Exercise 2.10

## Custom error message

Try to open a (non-existing) file for reading and return a custom error message if the file does not exist.

Tips:

- Assign a **filename** to a variable. E.g. filename = 'a_file_that_does_not_exist.txt'
- Add a **try** statement
- Open the file within the try block using the context manager **with**
- If succesfull **read** the complete file and print the contents.
- Add an **except** statement for an IOError exception
- Within the except block **print** a custom error message "Cannot open the file"

Exercise 2.11

## Foolproof numeric 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 numeric_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