Introduction to Python Programming (2/3)

Institute of Medical Informatics

University of Lübeck

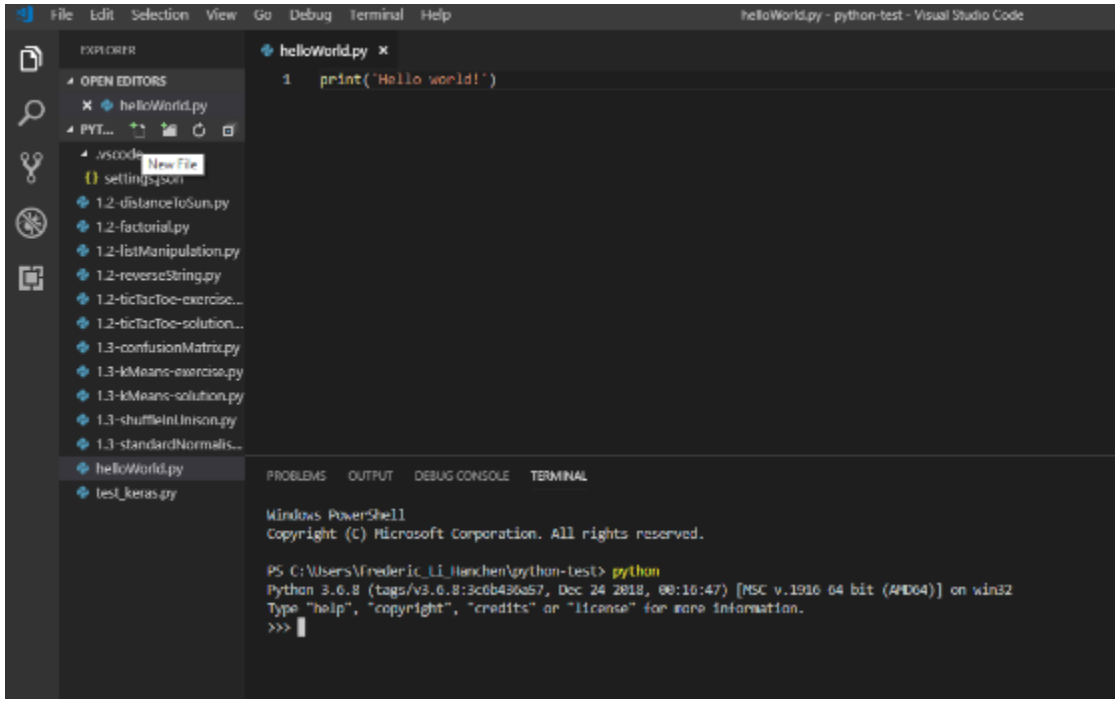
Contents of the session

- Introduction to Python (https://docs.python.org/3/tutorial/)
  - How to run Python code?
  - Identifiers/ Name convention / Keywords
  - Syntax (indentation, comments)
  - Basic operators (+,*,/,-,==,**)
  - Main variable types (int, float, string, bool)
  - Lists, Tuples, dictionaries
  - If statements
  - Loops
  - Functions
  - Classes
- Python Exercises

How to run Python code?

- Two main ways to execute Python code:
  1. Start the **Python interpreter** by typing > python in command line

  2. Write a **Python script** (.py file) then execute it by typing > python name_of_the_script.py in command line



Python Identifiers

- A **Python identifier** is a name used to identify a variable, function, class, module or other object:
  - Starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).
  - Case sensitive.
- When using a Python identifier, avoid:
  - Using a Python keyword.
  - Starting or ending the identifier with underscore (_) unless you have a specific reason to do so.
- Python is a **dynamically typed** language:
  - No need to declare any variable type.
  - The type of a variable can be changed in the same script.
  - E.g.:

```
aVariable = 0
…
aVariable = “Hello world!”
```

Python keywords

- Reserved words (cannot use them as constant or variable)

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

Naming conventions

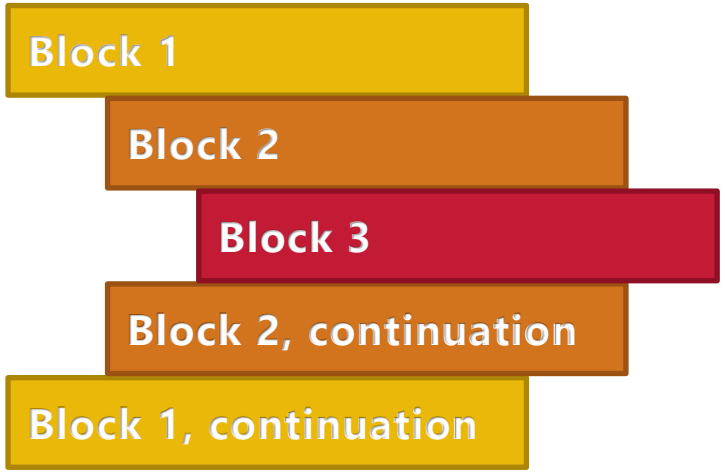
https://visualgit.readthedocs.io/en/latest/pages/naming_convention.html

- Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
- Starting an identifier with a single leading underscore indicates that the identifier is private.
- Starting an identifier with two leading underscores indicates a strongly private identifier.
- Identifiers which start and end with two trailing underscores are language-defined special names. The most useful examples are:
  - `__init__` which refer to the class constructor
  - `__name__` and `__main__` which are used to define a main file:

```
if __name__ == “___main__”:
    ……
```

Indentation

- In Python, no braces to indicate blocks of code
- Denoted by **line indentation**
- Number of spaces in the indentation up to you, but all statements within the block must be indented the same amount
  - Commonly used indentations: **tabs** or **4 spaces.**
- Notes:
  - mixing spaces and tabs together → error!
  - no need to use any character to denote an end of line (e.g. ";")




```
for idx in range(n):
    if B:
        Task inside if
    Task inside for
Task outside for
```

Multi-Line Statements

- Statements in Python typically end with a new line
- Use of the line continuation character (\)

```
total = item_one + \
        item_two + \
        item_three
```

- Statements contained within the [], {}, or () brackets do not need to use the line continuation character

```
days = ['Monday', 'Tuesday', 'Wednesday',
        'Thursday', 'Friday']
```

Quotation

- Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals
- Start and end same type
- Triple quotes are used to span the string across multiple lines

```
word = 'word'
sentence = "This is a sentence."
paragraph = """This is a paragraph. It is
made up of multiple lines and
sentences."""
```

Comments

- A **hash sign (#)** that is not inside a string literal begins a comment -> whole line is commented (ignored by the interpreter)
- Can be used to comment multiple lines in a row
- **Triple-quoted string** is also ignored by Python interpreter and can be used as a multiline comments

```
#This is a comment
print("Hello World")

'''
This is
a
multirow comment
'''
```

Using libraries

- Each library needs to be (installed and) **imported** before use
- Three common alternatives:
  - Import the full library:
    ```
    import numpy
    ```
  - Import selected functions from the library:
    ```
    from numpy import array, sin, cos
    ```
  - Import all functions from the library:
    ```
    from numpy import *
    ```
- Note regarding solution #3: different modules may contain functions with the same name → importing all functions from modules can cause problems!
- All methods support **aliases**, e.g.:
  ```
  import numpy as np # numpy is be referred to as np
  from numpy import zeros as z # numpy.zeros can be called
  using z(…)
  ```

Basic operations

- The operators +, -, * and / work just like in most other languages
- The standard comparison operators are written the same as in C:
  - < (less than)
  - > (greater than)
  - == (equal to)
  - <= (less than or equal to)
  - >= (greater than or equal to)
  - != (not equal to)
- Modulo operation: %
- Floor division: a // b

Assigning values to variables

- Variables do not need explicit declaration to reserve memory space
- No type needed
- Equal sign (=)
- delete entire variables: *del *variable name**

```
# An integer assignment
counter = 100
# A floating point
miles = 1000.0
# A string
name = "John"
# one line
counter, miles, name = 100, 1000.0, "John"
# Assign single value to several variables
a = b = c = 1
# Delete a variable
del a
```

Standard data types

- Python has six standard data types
  - Numbers
  - Boolean
  - String
  - List
  - Tuple
  - Dictionary
- For numbers: integer, float, complex
- For Booleans: `True, False`
  - Note: in Python 3.x, Booleans inherit from integers, i.e. `True == 1` and `False == 0` will both return `True`.

Lists

- Items separated by commas and enclosed within square brackets ([])
  - Similar to arrays in C.
- **Indexing starts at 0**
- Items can be of different data type
- Items can deleted: *del*

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']

print(list) # Prints complete list
print(list[0]) # Prints first element of the list
print(list[1:3]) # Prints elements starting from 2nd till 3rd
print(list[2:]) # Prints elements starting from 3rd element
print(tinylist * 2) # Prints list two times
print(list + tinylist) # Prints concatenated lists
```

Tuple

- Items separated by commas and enclosed within parentheses
- Indexing starts at 0
- **Cannot be updated**, i.e. read-only lists

```
# Creates a tuple
tuple = (1,2,3)
# Prints tuple
print(tuple)
# Prints first element of tuple
print(tuple[0])
# Not allowed:
tuple[0] = 1
```

Dictionary

- Kind of hash table type
- Consist of **key-value pairs**
  - Keys can be almost any Python type
  - Values can be any arbitrary Python object
- Dictionaries are enclosed by curly braces ({ })

```
python_dict = {'name': 'john','code':6734, 'dept': 'sales'}

# Prints complete dictionary
print(python_dict)
# Prints all the keys
print(python_dict.keys())
# Prints all the values
print(python_dict.values())
# Prints value for given key
print(python_dict['code'])
```

IF statement

```
x = 1
if x < 0:
         x = 0
         print('Negative changed to zero')
elif x == 0:
         print('Zero')
elif x == 1:
         print('Single')
else:
         print('More')
```

- Form
  if *condition* :
  code block
- Elif/else are optional

Loops - For

- Iterates over the items of any sequence
- Iterate over a slice copy of the entire list with *list[:]*

```
list = ['element1', 'element2', 'element3']
for element in list:
        print(element)
```

- Iterate over a sequence of numbers: *range* - function

```
for i in range(5): *
        print(i)
```

```
0
1
2
3
4
```

* *range()* returns an iterator, not a list!

Loops - While

- The while statement is used for repeated execution as long as an expression is true
- Optional else clause: will be executed the first time the expression is false and the loop terminates

```
i = 0
while i<5:
        print(i)
        i += 1
else:
        print("Finished")
```

- The *break* statement, like in C, breaks out of the innermost enclosing *for* or *while* loop.

Functions

- The keyword *def* introduces a function definition
- Followed by:
  - function name
  - parenthesized list of parameters
- Function body must be intended

```
def print_square (number):
        print(number * number)

print_square(2)
```

Example procedure (there is no return value)
In fact Python returns *None*

Functions – Return value

- *Return* statement to return a value of the function

```
def square (number):
    return number * number
a = square(2)
```

- Python does not support *overloading* a function

Functions - Argument

- Default argument values

```
def func(mandatory_var, default_arg_var=2):
        print(mandatory_var, default_arg_var)

func(5,5)
func(5)
```

- Keyword arguments

```
# allowed:
func(mandatory_var = 5, default_arg_var = 5)
func(default_arg_var = 5, mandatory_var = 5)
func(mandatory_var = 5)
func(5, default_arg_var = 5)
# not allowed:
func(default_arg_var = 5)
func(5, mandatory_var = 5)
```

Class

- Class definition with key word *class*
- The instantiation operation creates an empty object and automatically invokes function *__init__(self)*

```
class Bag:
    def __init__(self):
        self.data = []

    def add(self, x):
        self.data.append(x)

    def addtwice(self, x):
        self.add(x)
        self.add(x)
```

More information …

- Basic “informal” introduction
- More information can be found under https://docs.python.org/3/tutorial/
- Open topics:
  - Inheritance
  - Private variables
  - Dates/Times
  - Random
  - Multi-Threading
  - …

Exercises

Note: exercises inspiration coming from https://www.practicepython.org/

1. Function definition
2. List manipulation
3. String manipulation
4. Dictionary manipulation
5. Object-oriented programming

Exercise 1

Write an implementation of the factorial function.

Tips:

- The factorial function can be recursively defined as follows:

$$\forall n \in \mathbb{N}, \quad factorial(n) = \begin{cases} 1 \quad if \quad n = 0 \\ n \times factorial(n-1) \quad otherwise \end{cases}$$

- Write your function in a script, then execute it in a terminal (with VS Code: right click on script => "Run Python file in terminal")

Exercise 2

- Write a function `printNegative` which prints all negative elements of an integer or float input list.

- Write a function `filterOdd` which takes a list of integers as input and returns a sub-list of the input containing only its odd elements (tip: the Python modulo operator is %; e.g. 7%2 returns 1).

- Write a function `removeDuplicate` which takes a list of any type as input, and returns a list without any duplicate elements as output.

Exercise 3

Write a function `printReverseString` which takes a sentence under the string format as input (e.g. `"Hello world!"`) and prints the sentence with words in the reverse order (e.g. `"world! Hello"`).

Tips:

- A sentence is an ensemble of words separated by spaces
- The Python string methods `split` and `join` should be useful for this exercise

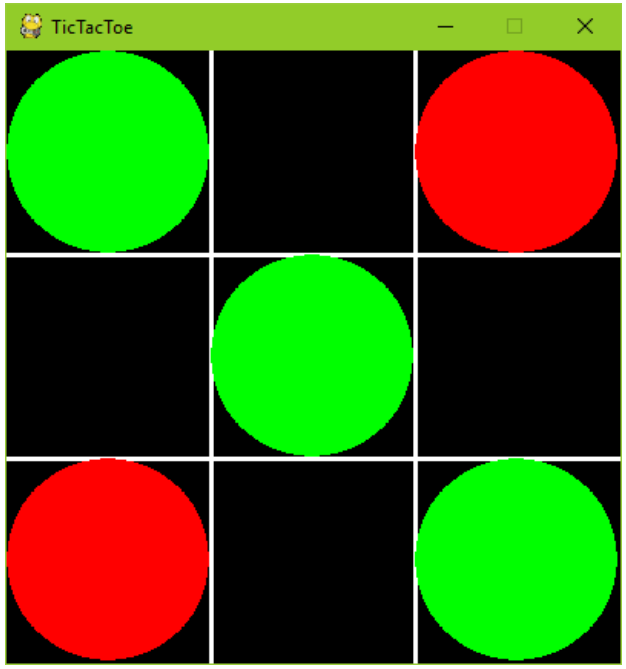
Exercise 4

Write a function `distanceToSun` which asks the user to input the name of a planet of our solar system in command line, and prints the distance between the planet and the Sun in kilometres.

Tips:

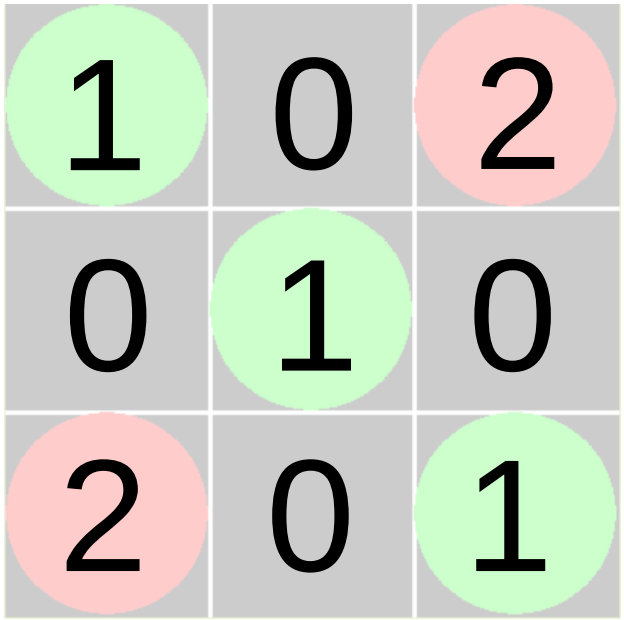
- Pick the online source of your choice to find distances between planets and the Sun.
- Asking for an user input in terminal can be done using the `userInput = input("text to display")` syntax.
- Remember to treat the case where the user input is invalid.
- Stopping a function can be done using `return`
- Reminder: Python dictionary keys are case-sensitive.

Exercise 5: TicTacToe (1/3)



- Implement a simple TicTacToe game
- Code file to fill out provided on Moodle
- Implementation of a very rudimentary AI: random moves
- Possibility to work on more elaborated AIs if still time

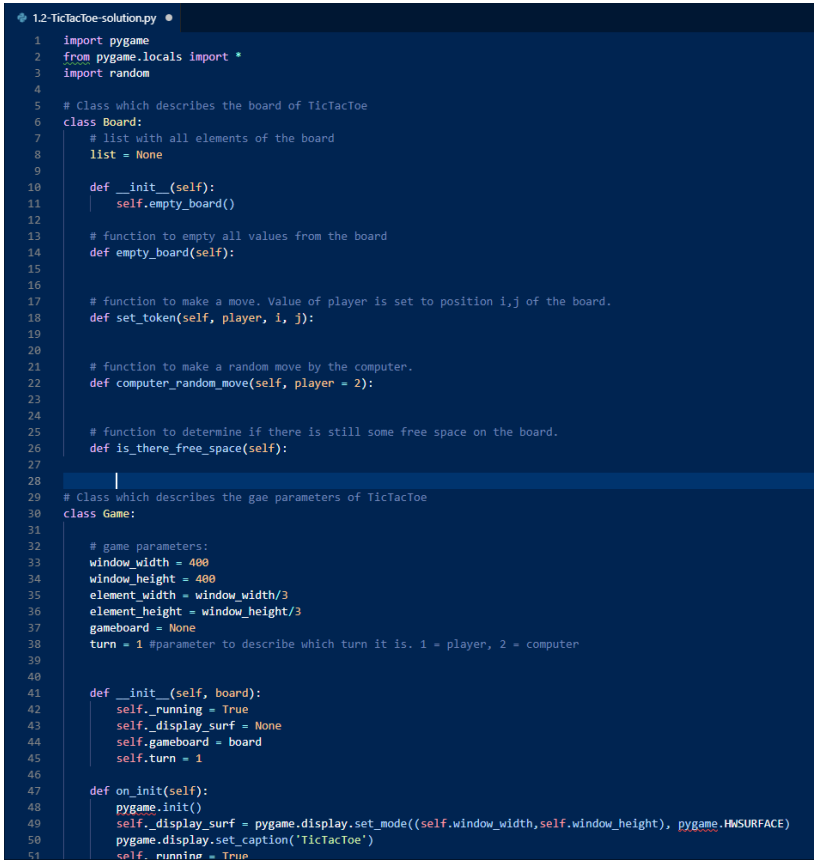
Exercise 5: TicTacToe (2/3)



- Board represented as a matrix
- Python implementation: numpy array or list of lists
- Elements of the matrix ϵ {0,1,2} for {empty, player1, player2} respectively

Exercise 5: TicTacToe (3/3)

- Object-oriented implementation provided
- Two classes
  - `Board`: board representation and manipulation
  - `Game`: game management and display
- TODO: write the methods of the class `Board`
- If finished with random AI, try:
  - Rule-based AI
  - "Real" AI based on the Minimax algorithm



Conclusion

- Basic informal introduction to Python
- First basic Python exercises
- Next week: Python exercises in relation to machine learning