UNIVERSITY OF TWENTE.

PYTHON **Review**

Presented by:
Mahdi Farnaghi
Assistant Prof.
Department of Geo-Information Processing

FACULTY OF GEO-INFORMATION SCIENCE AND EARTH OBSERVATION

NOTIONS TO BE GRASPED

High-level languages

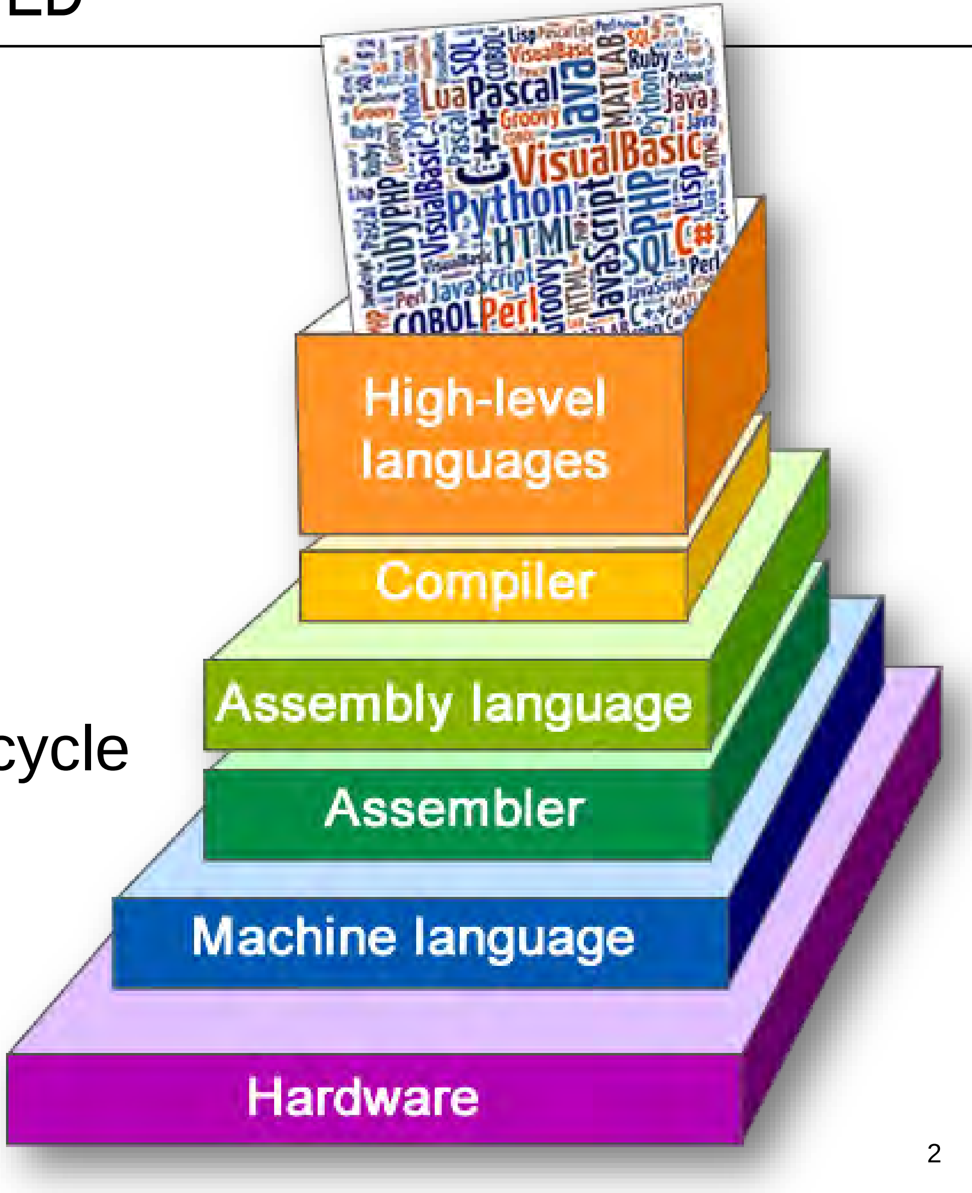
- Python
- JavaScript
- Html
- Css

High-level, low-level

Compile

Assemble

Program development cycle

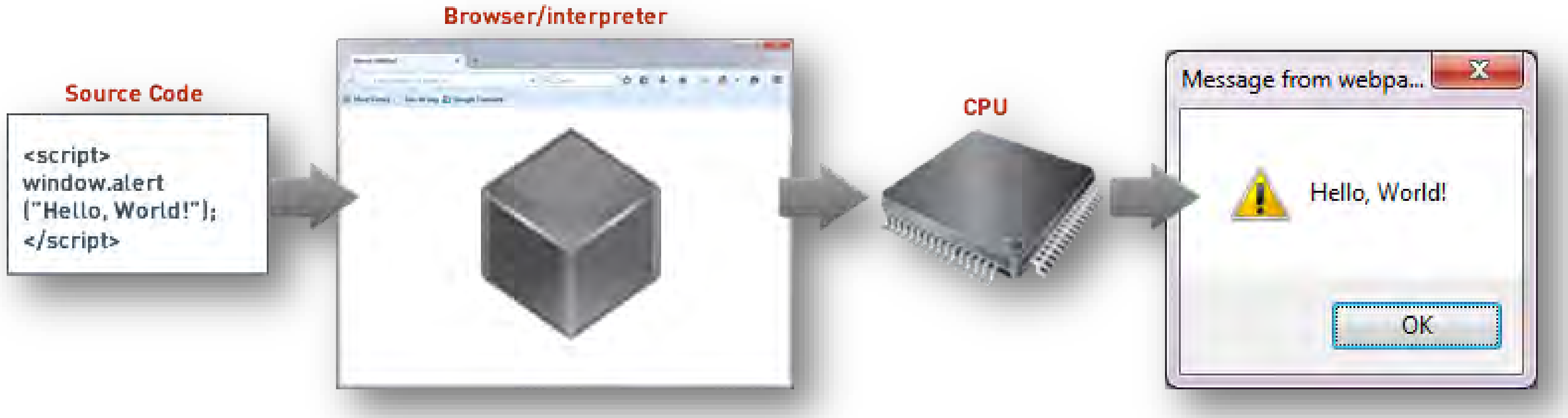


PROGRAM INTERPRETER

An interpreter is a “line-by-line compiler”

It compiles and executes the program instruction-by-instruction

Scripts like Python, HTML and Javascript are run in this way



```
<script>
window.alert
("Hello, World!");
</script>
```

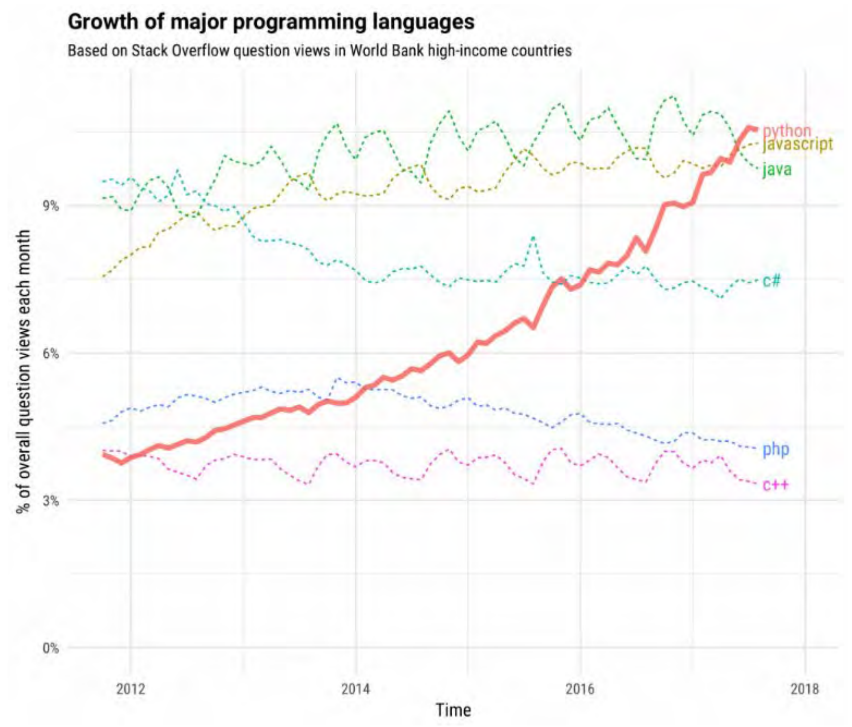
UNIVERSITY OF TWENTE.

PYTHON

a **widely used** general-purpose, high-level programming language

an *interpreted, interactive, object-oriented* programming language
a scripting language

a portable language: it runs on UNIX, Windows, OS/2, Mac, and many other platforms



UNIVERSITY OF TWENTE.

INSTALLING PYTHON INTERPRETER

Two python branches exist:

- Version 2
- **Version 3 (we will use this version)**

Python 2.x is legacy, Python 3.x is the present and future of the language
(wiki.python.org)

www.python.org

WHAT ABOUT ANACONDA?

www.anaconda.com/download/

OFFICIAL PYTHON DOCUMENTATION

docs.python.org

A python tutorial from the official docs (at least until chapter 7):

docs.python.org/3/tutorial

PYTHON IS MODULAR

A module or library in Python is a container with definitions, statements and objects.

```
from osgeo import gdal
import os

dataDirectory=r'C:\gdal\data\tmax'

# initialize dataset variable
raster = None
#change to the data directory
os.chdir(dataDirectory)
# open dataset
raster = gdal.Open("2014.tif")
print("file opened!")
if raster is not None:
    raster = None
    print("file closed!")
```

Import a module!

core
jupyter
pandas
numpy
matplotlib
gdal
osgeo
requests
scikit-learn
psycopg2

python

UNIVERSITY OF TWENTE.

INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)

For code development we will make use of an IDE

- **VSCode**
- Komodo
- Spyder
- Eclipse
- Notepad++
- Emacs
- Vim
- **PyCharm**

Another option:

- Jupyter Notebooks

“The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.”

VARIABLES and TYPES

VARIABLES

A variable is like a label of a little box that can store "things"

A variable always has:

identifier	a
value	1
type	integer

UNIVERSITY OF TWENTE.

VARIABLES AND TYPES

Objects always have a type

```
>>> a = 1
>>> print ( type(a))
<class 'int'>

>>> a = "Hello"
>>> print ( type(a))
<class 'str'>

>>> print ( type(1.0) )
<class 'float'>
```

INVALID NAMES

Not all names are valid!

```
>>> 76trombones = 'big party'
SyntaxError: invalid syntax

>>> more@ = 1000
SyntaxError: invalid syntax

>>> class = 'Spatial analysis'
SyntaxError: invalid syntax
```

NAMING PROBLEMS

```
bad name = 5
SyntaxError: invalid syntax
```

(names cannot contain spaces!)

```
Bob = 23
year = bob
NameError: name 'bob' is not defined
```

(names are case sensitive!)

<u>Python language is case sensitive</u>

REMEMBER!

UNIVERSITY OF TWENTE.

NAMING PROBLEMS

Be careful with too obvious names!

```
>>> print ( type(2) )
<class 'int'>

>>> type = 23
>>> print ( type(2) )
TypeError: 'int' object is not callable
```

Whoops! Existing things can be destroyed!

Do not name your variable with already existing function names!

STATEMENTS AND EXPRESSIONS

STATEMENTS

A statement is an instruction that the Python interpreter can execute

A statement can:

- o change the data environment (the part of the memory that holds the data)
- o alter the flow of execution

```
>>> x = 2          assignment statement
>>> print (x)      print statement
2                  result of the print
```

A script/program is just a sequence of statements

EXPRESSIONS

A statement may contain expressions

- An expression is a combination of operands (values, variables) and operators
- Operands are input to operators
- Expressions are evaluated by the interpreter
- Expressions create and process objects

UNIVERSITY OF TWENTE.

EXAMPLES

Comments can span entire lines:

```
# compute the area
area = length * width
```

Or, comments can be at the end of lines:

```
area = length * width    # compute area
```

multiline comments start and end with 3 quotes (single or double)

```
‘‘‘
this
is a
multiline comment
’’’
```

Multiline comments are used at the beginning of modules, functions, classes, and methods to insert descriptive text

FUNCTIONS
DEFINITION AND USAGE

FUNCTIONS

- A function is a portion of code, which performs a specific task

- There are 2 types of functions in Python:

1. **Built-in** to the language
   - available directly
   - available within modules e.g. *Math* module

2. **Custom** functions
   - customized routines created by the user

WHY FUNCTIONS?

Functions are useful!

Functions:

1. group statements
2. eliminate repetitive code
3. cut large programs into smaller bits
4. allow re-use of code

EXAMPLE OF FUNCTIONS

```
def print_squared(x):
    print (x**2)
```

This is a void function,

```
def squared(x):
    return x**2
```

This is a fruitful function,
it returns a value (different from *None*)

```
def squared_positive(x):
    if x<0:
        print ("Number is negative")
        return
    return x**2
```

This is a hybrid function

THINGS YOU CAN'T DO…

Assign a value to a variable and use it ***outside*** your function

function variables are *local*

```
def example_function(part1, part2):
        var1 = part1 + part2
        print (var1)

>>> example_function(1,2)
3
>>> print ( var1)
NameError: name 'var1' is not defined
```

…this is what the ***return*** statement is for

Exit or quit function!

The exit() or quit() function is useful in debug mode. This function stops the execution of the code. No more code is interpreted!

For example:

```
def function(x):
        print('Hello world')
        exit('Stop here')
        print('Hello world 2')
```

You can use exit() or quit() They are the same!

UNIVERSITY OF TWENTE.

CONDITIONALS

SYNTAX

```
if boolean expression :
    statements1
else:
    statements2
```

- If the expression is True, then the first body of statements is executed
- If the expression is False, then the second body of statements is executed
- Normal processing proceeds afterwards

UNIVERSITY OF TWENTE.

NESTED IF-STATEMENTS

Is x zero?

```
>>> if x > 0:
            print( 'x is positive' )
       else:
            if x < 0:
                   print( 'x is negative' )
            else:
                   print( 'x is zero' )
```

INDENTATION

Rule of thumb:

1. A block (or body) starts after a colon :
2. Everything to the lower-right belongs to the same block
3. All the statements within the same block *must* have the same indentation!

COMMON MISTAKES

```
if 1 == 1:
dothis()
dothat()
```

IndentationError: expected an indented block

```
if 1 + 1 == 2:
dothis()
    dothat()
```

IndentationError: expected an indented block

```
if 1 + 1 == 2
    dothis()
```

SyntaxError: invalid syntax

Lists and tuples

Built-in Python data structures

Python knows a number of built-in *compound* data types (containers or collections), used to group together other objects

Sequences

- Types: strings, lists, tuples
- Operations: Indexing, slicing, adding, multiplying, iteration & membership

Dictionaries

- Map keys to values through index
- Suitable for unstructured data

Sets

- Unordered and do not map keys to values

(Each category is called container)

Lists

he most versatile container is the *list*, which can be written as a list of comma-
eparated values (items) between square brackets

st items can be of different types:

```
> list_num = [1, 2, 4, 8, 16, 32.0, 64.0, 128, 256, 512.0]

> list_str = [“dear”, “students”, “this”, “is”, “a”, “list”, “of”, “str”]

> list_mixed = [“dear”, 32.0, “this”, “is”, “a”, 512.0, “!”, 2016]
```

Operations: Indexing

```
                  0           1              2        3          4
> fruits = [   "orange", "watermelon", "lemon", "coconut", "pineapple",
               "banana", "pomegranate", "kiwi", "grapes", "apricot" ]
                  5           6              7        8          9
```

```
                 -10         -9             -8       -7         -6
> fruits = [   "orange", "watermelon", "lemon", "coconut", "pineapple",
               "banana", "pomegranate", "kiwi", "grapes", "apricot" ]
                 -5          -4             -3       -2         -1
```

We can refer to elements in the list **forward** and **backwards**

Operations: Slicing

Slicing: Introducing **“:”** operator

A slice [n:m] is a segment of the sequence.

- From the *n-th* item
- Until the *m-th* item (but excluding the *m-th* item)
- See it as **[start:stop]**

```
>>> fruits = [   “orange”, “watermelon”, “lemon”, “coconut”, “pineapple”,
                 “banana”, “pomegranate”, “kiwi”, “grapes”, “apricot” ]

>>> print(fruits[4:8])
>>> [“pineapple”, “banana”, “pomegranate”, “kiwi”]

>>> print(fruits[-3:])
>>> [“kiwi”, “grapes”, “apricot”]

>>> print(fruits[-3:-1])
>>> [“kiwi”, “grapes”]

>>> print(fruits[:4])
>>> [“orange”, “watermelon”, “lemon”, “coconut”]
```

Tuples

A tuple is like a list, but **immutable**!

- Once it is defined, it cannot be changed
- It is also a sequence data type
- You can define a tuple in two ways:
    - Using parenthesis ( )
    - Using the container **tuple()**

```
>>> fruits = ("orange", "watermelon", "lemon", "coconut")
>>> print(fruits[1])
>>> "watermelon"

>>> fruits = tuple("orange", "watermelon", "lemon", "coconut")
>>> print(fruits[1])
>>> "watermelon"
```

UNIVERSITY OF TWENTE.

Iterations

Iterations

The repetition of a task is called **iteration**
Iterations are performed **in statements called loops**

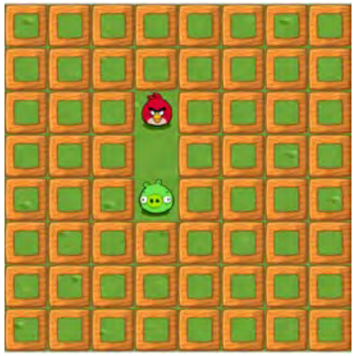
In Python, you can use two types of loops:
- The `for` loop
- The `while` loop

The `while` loop

A simple while loop counting until 10 ...

... like in angry birds example!

```
i = 1
while i <= 10:
      print(i)
      i = i + 1
```



This structure is very typical in while loops:

```
i = 1                   ➔ initialization
while i <= 10:          ➔ condition
      print(i)          ➔ statement
      i = i + 1         ➔ update condition
```

Variable "`i`" controls the loop, it is called counter or loop variable

UNIVERSITY OF TWENTE.

The **while** loop

We can iterate lists

```
fruits = ["orange", "watermelon", "lemon",
"coconut"]
i = 0
while i < len(fruits):
   print(fruits[i], end="")
   i = i + 1
```

Output: “orange”, “watermelon”, “lemon”, “coconut”

The `for` loop

A for loop starts with the keyword **`for`**

This keyword is always followed by:

1) A **variable +** keyword **`“in”`** **+ sequence**
2) A **colon** character
3) A **body** of statements
4) This statements are **indented**!

The **`for`** loop

The skeleton of a for loop is:

```
for variable in sequence:
        statement #1
        statement #2
        ...
        statement #N
```

Iterates over the items of any sequence (i.e. lists, tuple, string)
in the order the appear in the sequence

`break` `and` **`continue`** `keywords work for` **`for`** `too`

The `for` loop

You can iterate a list as you wish:

```
fruits = [ "orange", "watermelon", "lemon", "coconut",
"pineapple", "banana","pomegranate", "kiwi","grapes",
"apricot" ]

for fruit in fruits: # pair numbers
    print(fruit)
```

Can you tell me the output of this?

Nested loops

How to iterate 2-dimensional sequences? With nested loops!

```
m = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

the_sum = 0

for i in m:
  for j in i:
    the_sum += j
    print(j)

print("The sum is: ", the_sum)
```

```
1
2
3
4
5
6
7
8
9
The sum is:  45
```

UNIVERSITY OF TWENTE.