python-1

September 15, 2020

0.1 # Introduction to Programming – Python

0.2 Course 1 – first steps

0.2.1 ESSEC Business School

gael.guibon@gmail.com

gael.guibon@telecom-paris.fr

Original content inspired by Clement Plancq's IM courses

0.3 # A program

- A program is a **set of instructions**
  - Orders for the machine to execute later on
- A program will be compilated for the machine to understand it
- A compilated program is only binary code

0.4 # Programming languages

- Languages are divided by level :
  1. High level: **Python**, Java, etc.
  2. Low-Level: COBOL, FORTRAN, C, etc.
  3. Assembly: some CPUs
  4. Machine code
  5. Micro-code: CPUs

0.5 # What you need

- Logic:
  - Propositional logic
  - First-order logic
  - Predicate logic
- Rigor:

  - Verify everything, every character, every content
  - Order is important!
- Patience:
  - Verify/understand error logs
  - Proceed step by step

0.6 # What you will learn

1. To **understand** and use basic types
2. To **implement** simple logic
3. To **write** simple code

1 Python

Python is a high level programming language under free licence. It was created by Guido Van Rossum but since 2001 the development is lead by the Python Software Foundation (Guido was the *"benevolent dictator for life"* until july 2018).

Language evolutions are detailed in Python Enhancement Proposals (PEP).

1.1 Python functions

To know what does each native Python functions, you need to access the documentation. * type : `help(print)`
`q` to exit the doc * in your browser : `https://docs.python.org/3/library/functions.html` * or : DevDocs

Famous PEPs: * PEP 20 "Zen of Python"1 (`import this` in python console) * PEP 8 "Style Guide for Python Code"

1 "There should be one– and preferably only one –obvious way to do it."

[ ]:
```
# show the code guidelines for Python
import this
```

2 A basic instruction

We can print anything by using the `print()` function.

`print( SOMETHING )`

[ ]:
```
# what does the print() function ?
help(print)
```

[ ]:
```
print('Hello world')
```

- Principle: 1 line = 1 instruction
- Everything after # is a comment

```
# this is a comment !!!
```

[ ]:
```
print('hey') #first instruction
print('you') #second instruction (order is important)
```

3 Variables

- Variables are data containers.
- = is used to put value into a variable
- Variable's content can be put into another variable

```
variable1 = variable2
```

[ ]:
```
name = 'Gaël'
print(name)
```

[ ]:
```
# variables values
lastname = 'Guibon'
firstname = 'Gaël'
print(lastname)
print(firstname)
lastname = firstname
print(lastname, firstname) # multiple arguments can be passed into print␣
↪function.
# They will automatically be separated by a space.
```

4 Variables names

Variables can have any names **EXCEPT the already declared keywords**. To show all the python keywords do:

[ ]:
```
import keyword # import libraries. this one is a standard library
print(keyword.kwlist)
```

- Variable names are alphanumeric values (upper or lowercase), or/and numbers, or/and underscores. Nothing more.
- Variable names need to be clear and understandable. For instance:

```
birthdayDate = '01012005' # good
date = '01012005' # bad
xyhzi = '010120005' # very bad
```

- It is recommended to follow the official Python advices PEP 8.

[ ]:
```
# Python does not accept this kind of variable name
my-super-username = 'Robert' # known as KebabCase
```

[ ]:
```
1_does_not_work = 'bad name'
```

[ ]:
```
# Use this instead
mySuperUsername = 'Robert' # known as CamelCase
# or this
my_super_username = 'Robert' # known as SnakeCase
# print them to see that it works
```

5 Arithmetic Operators

`+` addition

[ ]:
```
2 + 5
```

`-` substraction

[ ]:
```
3 - 2
```

`*` multiplication

[ ]:
```
2 * 3
```

`/` division

[ ]:
```
36 / 6
```

`%` modulus (division remainder)

[ ]:
```
36 % 8
```

`//` Floor division (cut the result after the period)

[ ]:
```
3 // 2
```

`**` exponent

[ ]:
```
10 ** 2 * 4
```

- Operation order and priorities are the same as in mathematics
- You can use parenthesis to define priorities

[ ]:
```
36 / (2*3)
```

- Of course, it works with variables

[ ]:
```
a = 3
b = 2
print(a*b)
```

6 Variable types

- Data can have different forms, same goes for variables
- Each variable has a **Type**
- Here are the most commons types in Python:
  - **String**: str() : string of characters (text)
  - **Integer**: int() : integer numbers
  - **Float**: float() : numbers with floating decimal points
  - **Boolean**: bool() : True or False
  - Long (*not in Python*)
  - Char (*not in Python*)
- Python is a strong dynamic typed language
  - dynamic = variable type is determined by the interpretor (compilor)
  - strong = no implicit conversion of types
- Try this :

[ ]:
```
"Hello" + 1
```

It does not work as we try to concatenate a String (str) with an Integer (int).

You can find the variable's type with `type()` function.

[ ]:
```
print( type('hello world') )
print( type(23) )
print( type(False) )
print( type(23.45) )
```

6.1 Type conversion

- `str()`, `int()`, `float()` convert the passed argument as the desired type
- `bool()` returns `True` if the argument is true, `False` otherwise.

[ ]:
```
int(3.14159265359) # transforms a float into an integer
```

[ ]:
```
"Hello" + str(1) # transforms an integer into a String and concatenate both␣
↪Strings
```

7 Strings Operations

- Concaténate Strings: `+`

[ ]:
```
"Hello" + "world"
```

[ ]:
```
"Hello" + " " + "world"
```

- Strings can be surrounded by simple quotes `'hello'` or double quotes `"hello"`
- If your work contains a simple quote use double quotes:

```
negation = "don't do that!"
```

[ ]:
```
'it doesn't work'
```

[ ]:
```
"It doesn't work? Oh yes it does!"
```

- You can always escape special characters with \ to indicate to not parse them

[ ]:
```
'It doesn\'t work? Oh yes it does!'
```

[ ]:
```
''' this is a multiline String
I can put a huge text between 3 simple or double quotes'''
```

[ ]:
```
""" Works with tripple double quotes"""
```

8 Conditions

```
if condition:
    #begin of block
    instruction1
    instruction2
    #end of block
```

Indentations are important in Python

8.1 Comparison Operators

`<` lesser than / `<=` equal or lesser than `>` greater than / `>=` equal or greater than
`==` equal / `!=` not equal `is` identify as / `is not` not identified as

[ ]:
```
x = 4
if x > 3:
    print("x equals ", x, "and is it greater than 4?", x>3)
```

[ ]:
```
if x > 3:
    print("yes x is greater than 3")
else:
    print("nope x is not greater than 3")
```

8.2 Logical Operators

`not` logical NOT (reverse a condition: "is he human?" is the inverse of "is he not human?") `and` logical AND (if and only if both conditions are True, returns True. Else, returns False.) `or` disjonction (if one condition is True, returns True. Else, returns False.)

[ ]:
```
if x > 3 and x <= 5:
    print("x is between 4 and 5" )
elif x > 5:
    print("x is lesser than 5")
else:
    print("x is neither")
```

8.3 Membership Operators

`in` : verify is something is in a string or list `not in` : reverse version

[ ]:
```
sentence = 'I want to eat something!'
print( 'eat' in sentence )
print( 'eat' not in sentence )
print( 'hello' not in sentence )
```

9 Basic Role Playing Game (1)

- Objective: construct a textual role playing game. The game is only text with choices, conditions to verify the choices and player status.
- To do so you need one additional function (see below)

9.1 Rules

- Player advance from a room to another by textually selecting one of the rooms.
- Player starts with 200 hp (health points)
- A bad decision cost the player to lose 25 hp.

9.2 Input() Function

- Retrieve the player input

[ ]:
```
# Examples
# Get the input function and display a greeting message
print("Hello Player One, what's your name?")
playerName = input()
print("Welcome", playerName)
```

```
# Example room pattern
print('''Your are in a tiny room. Humidity fills the air but your stomach␣
 ↪reminds you that you are very hungry.

You are in front of two doors. Behind the first one you can hear muffled voices.

Behind the second one you can smell something intriguing.''')
print('Which door do you choose? Type "first" for the first room and "second"␣
 ↪for the second room.')
selection = input()
if selection == 'first':
    print('Game Over! Try again', playerName)
elif selection == 'second':
    print('Upon opening the door, you can see a huge fest with exquisite meals␣
 ↪everywhere.')
```

- Complete the Role Playing Game skeleton by adding choices, player HP, etc.
- *Tip: this kind of game is a decision tree of choices.*

10 Install local python environnement

At home you may want to use Python locally. Here are the steps: 1. Download the latest Python (Python3 not 2) from here: https://www.python.org/downloads/ 2. Install it (if you are using Ubuntu 16.04 you already have Python installed)

Start using it: - Open terminal(unix/mac) or CommandLine(windows), type `python` to start an interactive python environnement - Create a file named `my_super_program.py` and type `print('hello')` inside. Execute this file by typing `python3 my_super_program.py`.