Programming with Python - Refresher course

Sonia Martinot, PhD, AI Researcher at CEA

Master in Data Science & Business Analytics
CentraleSupelec & ESSEC

September 6, 2024

Contents

Current Section

Who is teaching ?

- MSc in Applied Mathematics & Data Science from CentraleSupelec
- PhD in Applied Mathematics from UniversitÃ© Paris-Saclay
- Researcher at the Commissariat Ã l'Ã©nergie atomique

Content of course

8 courses of 3 hours:

- 06/09: 16h30-19h30 Basics
- 09/09: 16h30-19h30 Basics
- 11/09: 9h-12h Numpy
- 12/09: 16h30-19h30 Matplotlib
- 17/09: 13h15-16h15 & 16h30-19h30 Image processing & Create a Game in Python
- 30/09: 9h-12h & 13h15-16h15 Numerical methods & OOP

Hands-on course: always bring your computer and write code !

Courses will be held in English.

Questions

- Who has never coded in Python ?
- Who has never programmed anything ever ?

Important: Don’t stay stuck, ask questions !

Current Section

Python

What is Python ?

- It is Free !
- Relatively easy
- Created in 1991
- Mainly used for data analysis and machine learning and deep learning (cf. Pytorch)
- Object Oriented Programming language (vs functional programming)

How do you compile the code ?

- Create a python script `program.py` and from console, type: `python program.py`.
- Use Jupyter.

Python

IDE : Integrated Development Environment, is a software designed to help you write code. Many IDEs exist:

- Visual Studio Code
- PyCharm
- Sublime Text
- Vim
- Spyder

You can use the one you prefer during the courses. We will work with Google Colab / Jupyter Notebook.

Syntax

- Very light syntax
- In Python no ;
- Only tabulation and :
- Every block (functions, loop) uses one tabulation and one :

Example with a `for` loop:

```
for i in range(10):
    do something
other block doing things outside the for loop
```

Current Section

Data Type

- **Integer**: 1, 2, 3, 10, 11 ...
- **Float**: 1.1, 3.0 ...
- **Boolean**: only **True** or **False.**
- **String**: use quotation marks - ’something’

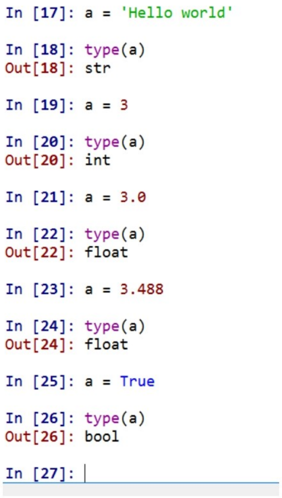
Data Type

- To create a variable `a`, use the instruction `a = expression`
- 5 and 5.0 are different.
- To change the data type, i.e. **cast** a variable to another type, use the functions `float()` and `int()`:

```
# Declare variable
a = 5.0              # Variable a is a float
# Cast it to integer
integer_a = int(a)  # Then a = 5
```

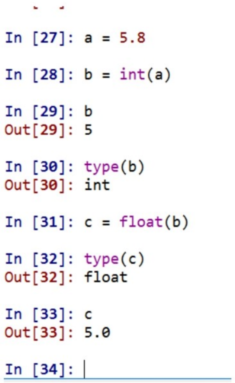
- To know the datatype of a variable use the function `type()`

Data Types



```
In [17]: a = 'Hello world'

In [18]: type(a)
Out[18]: str

In [19]: a = 3

In [20]: type(a)
Out[20]: int

In [21]: a = 3.0

In [22]: type(a)
Out[22]: float

In [23]: a = 3.488

In [24]: type(a)
Out[24]: float

In [25]: a = True

In [26]: type(a)
Out[26]: bool

In [27]: |
```

FIGURE – Variable type



```
In [27]: a = 5.8

In [28]: b = int(a)

In [29]: b
Out[29]: 5

In [30]: type(b)
Out[30]: int

In [31]: c = float(b)

In [32]: type(c)
Out[32]: float

In [33]: c
Out[33]: 5.0

In [34]: |
```

FIGURE – Type conversion

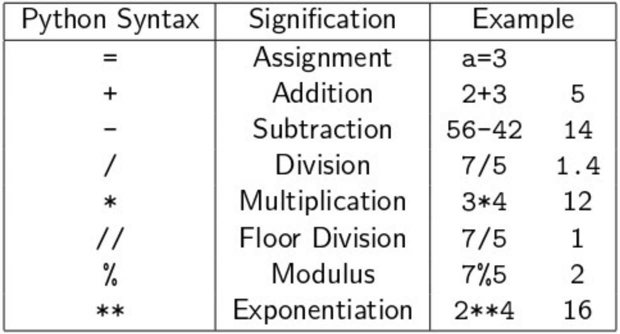
Variable names

You can give the name you want to a variable, but some words are reserved : the **keywords** and the **built-in functions.** Some examples:

- `is`
- `for`
- `list()`
- `range()`
- `dict()`
- `in`
- `del`
- `len()`
- `max()`
- `min()`
- `while`
- `input()`
- `pow()`
- `str()`

You can create a variable called as a **built-in function** but this is not recommended and often leads to bugs.
For **keywords** it is simply not possible.

Numeric Operators



Python Syntax	Signification	Example	
=	Assignment	a=3	
+	Addition	2+3	5
-	Subtraction	56-42	14
/	Division	7/5	1.4
*	Multiplication	3*4	12
//	Floor Division	7/5	1
%	Modulus	7%5	2
**	Exponentiation	2**4	16

TABLE – Numerical operators

Boolean Operators

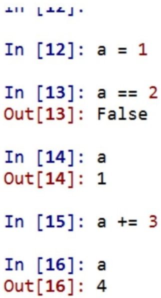
Python Syntax	Signification	Example	
`==`	Equal to	`2==3`	False
`!=`	Not equal to	`2!=3`	True
`>`	Greater than	`5>7`	False
`>=`	Greater than or equal to	`5 >= 5`	True
`<`	Less than	`4 < 8`	True
`<=`	Less than or equal to	`6 < 4`	False
`and`	Boolean and	`True and False`	False
`or`	Boolean or	`True or False`	True
`not`	Boolean not	`not True`	False
`in`	In sequence	`5 in [1,2,3]`	False
`not in`	Not in sequence	`5 not in [1,2,3]`	True

TABLE – Boolean operators

Rules for operators

- = and == are different !
- Assignment can be combined with operator. Ex : c += 1
- Priority rules :
  - Parenthesis
  - Exponentiation
  - Multiplication/Division/Modulo/Float Division
  - Addition/Subtraction
  - Equality/Inequality
  - Boolean and/or
  - In
- If you are not sure on the priority : use parenthesis !

Operators


```
In [12]: a = 1

In [13]: a == 2
Out[13]: False

In [14]: a
Out[14]: 1

In [15]: a += 3

In [16]: a
Out[16]: 4
```

Current Section

Data Structures

- **List** : a list (called array in other languages) is used to store multiple variables a the same time. They have a length and you can access any member of the list by its position. Can be changed (mutable).
- **Dictionnary** : a dictionnary is an unordered collection where you can access element with a key.
- **Tuple** : a tuple is similar to a list but you cannot change it (unmutable).
- **Set** : a set is a collection without indexing and with only one time each element.

List

Creation of a list:

- Empty list :a = []
- Non-empty List : a = [1, 2, 3]

Methods of a list:

- Add an element to a list.
- Remove an element from the list.
- Access an element from the list.
- Find the lenght of the list.

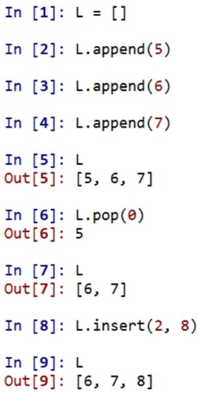
Important : The 1st element of a list has the index 0. The last one has the index n1 if the list has a length n.

Lists Methods

Python Syntax	Signification
`L = []`	Create an empty list
`L.append(element)`	Add an element to the end of the list
`L.pop(index)`	Remove and return an element at the given index
`L.insert(index, element)`	Insert an element at the defined index
`L.index(element)`	Returns the index of the first matched item
`L1 + L2`	Concatenate two list

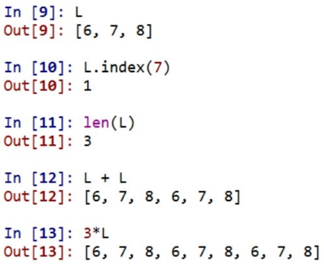
`len(L)`
`i * L`
	Return the length of a list
	Copy the list *i* times

Lists Examples



```
In [1]: L = []

In [2]: L.append(5)

In [3]: L.append(6)

In [4]: L.append(7)

In [5]: L
Out[5]: [5, 6, 7]

In [6]: L.pop(0)
Out[6]: 5

In [7]: L
Out[7]: [6, 7]

In [8]: L.insert(2, 8)

In [9]: L
Out[9]: [6, 7, 8]
```

FIGURE – Example 1



```
In [9]: L
Out[9]: [6, 7, 8]

In [10]: L.index(7)
Out[10]: 1

In [11]: len(L)
Out[11]: 3

In [12]: L + L
Out[12]: [6, 7, 8, 6, 7, 8]

In [13]: 3*L
Out[13]: [6, 7, 8, 6, 7, 8, 6, 7, 8]
```

FIGURE – Example 2

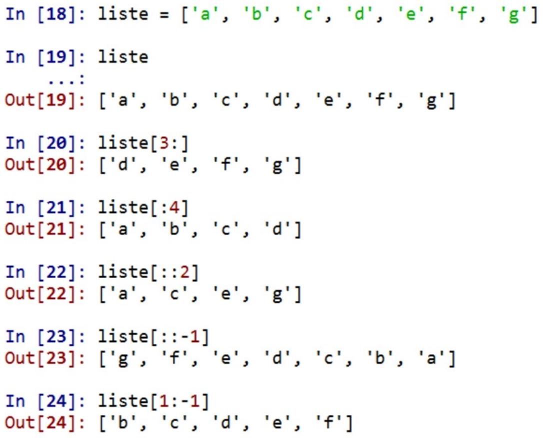
Lists Slicing

- **Slicing** : Create a shorter list from an existing list.
- Syntax : `new_list = my_list[start : end : step]`

Example:

```
# Starts at 2nd element and ends at 5th element with step 1
my_list[2:5]

# Starts at 1st element and ends at last element with step -1
my_list[::-1]
```

Lists Slicing



```
In [18]: liste = ['a', 'b', 'c', 'd', 'e', 'f', 'g']

In [19]: liste
    ...:
Out[19]: ['a', 'b', 'c', 'd', 'e', 'f', 'g']

In [20]: liste[3:]
Out[20]: ['d', 'e', 'f', 'g']

In [21]: liste[:4]
Out[21]: ['a', 'b', 'c', 'd']

In [22]: liste[::2]
Out[22]: ['a', 'c', 'e', 'g']

In [23]: liste[::-1]
Out[23]: ['g', 'f', 'e', 'd', 'c', 'b', 'a']

In [24]: liste[1:-1]
Out[24]: ['b', 'c', 'd', 'e', 'f']
```

FIGURE – Slicing Example

Lists Comprehension

- An easy and very Pythonic way to create a list

```
my_list = = [function(i) for i in range(n)]
```

- Comprehension list can be combined with:
  - `if` instructions
  - Other lists

If instructions

- Example - list of odd numbers inferior to 10:

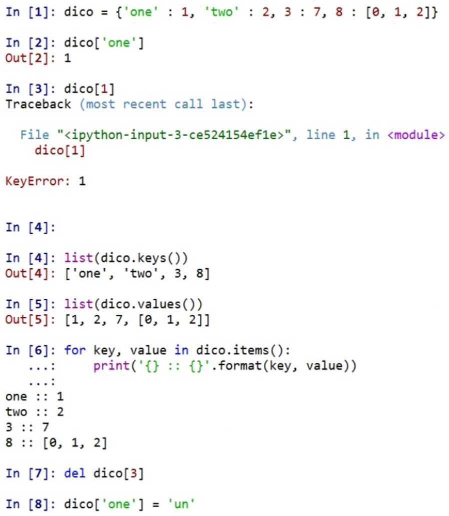
```
my_list = [2*i + 1 for i in range(5)]
my_list = [i for i in range(10) if i%2 == 1]
```

Dictionary Methods

Python Syntax	Signification
`D = {}`	Create an empty dictionnary
`D = {'key' : value}`	Create a dictionnary
`D[key] = value`	Add or modify a (key, value) in a dictionnary
`D['key']`	Get a value
`D.pop['key']`	Remove a value
`D.keys()`	Get a list of the keys
`D.values()`	Get a list of the values
`D.items()`	Get a list of the (key, value)
`key in D`	Check if key exists

TABLE – Dictionnary methods

Dictionary Example



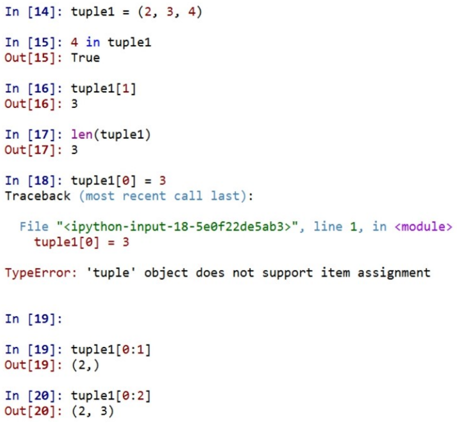
```
In [1]: dico = {'one' : 1, 'two' : 2, 3 : 7, 8 : [0, 1, 2]}

In [2]: dico['one']
Out[2]: 1

In [3]: dico[1]
Traceback (most recent call last):

  File "<ipython-input-3-ce524154ef1e>", line 1, in <module>
    dico[1]

KeyError: 1


In [4]:

In [4]: list(dico.keys())
Out[4]: ['one', 'two', 3, 8]

In [5]: list(dico.values())
Out[5]: [1, 2, 7, [0, 1, 2]]

In [6]: for key, value in dico.items():
   ...:     print('{} :: {}'.format(key, value))
   ...:
one :: 1
two :: 2
3 :: 7
8 :: [0, 1, 2]

In [7]: del dico[3]

In [8]: dico['one'] = 'un'
```

Tuple

Definition: A tuple is an ordered collection of elements. You cannot add or remove element of the tuple.

- To create a tuple use **parentheses**: `t = (5, 3)`
- To access an element use **brackets** like with a list: `t[0]`
- Slicing is possible like for a list: `t[:-1]`
- Changing or adding a value is not possible.
- A tuple has a length: `len(t)`
- You can check if an item is in a tuple.
- Outputs of functions can be a tuple.

Tuple Example



```
In [14]: tuple1 = (2, 3, 4)

In [15]: 4 in tuple1
Out[15]: True

In [16]: tuple1[1]
Out[16]: 3

In [17]: len(tuple1)
Out[17]: 3

In [18]: tuple1[0] = 3
Traceback (most recent call last):

  File "<ipython-input-18-5e0f22de5ab3>", line 1, in <module>
    tuple1[0] = 3

TypeError: 'tuple' object does not support item assignment


In [19]:

In [19]: tuple1[0:1]
Out[19]: (2,)

In [20]: tuple1[0:2]
Out[20]: (2, 3)
```

Tuple

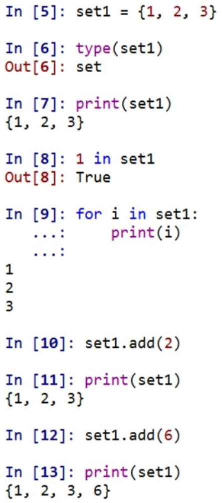
Definition: A set is an unordered collection of elements. There are no duplicates in a set.

- To create a set use **braces**: `t = { 1, 2 }`
- You cannot access an element using indexing.
- You can access all elements with a for loop.
- Removing or adding a value is possible:

```
# Remove inplace element
my_set.remove(element)
# Add an element to the set inplace
my_set.add(element)
```

- A set has a length: `len(t)`
- You can check if an item is in a set with `in`: `1 in t`

Set Example



```
In [5]: set1 = {1, 2, 3}

In [6]: type(set1)
Out[6]: set

In [7]: print(set1)
{1, 2, 3}

In [8]: 1 in set1
Out[8]: True

In [9]: for i in set1:
   ...:     print(i)
   ...:
1
2
3

In [10]: set1.add(2)

In [11]: print(set1)
{1, 2, 3}

In [12]: set1.add(6)

In [13]: print(set1)
{1, 2, 3, 6}
```

Recap

List	Tuple
Ordered Changeable Duplicate members `L = [1, 2, 3]`	Ordered Unchangeable Duplicate members `T = (1, 2, 3)`
Set	**Dictionnary**
Unordered Changeable No duplicate members `S = {1, 2, 3}`	Unordered Changeable No duplicate keys `D = {'key' : value}`

Current Section

Loops and statements

In Python you have 3 basic structures to know:

- If statements
- For loops
- While loops

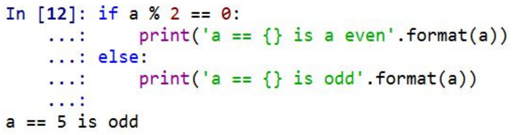
If statements

The syntax for `if` statements is as follows:

```
if condition1:
    intruction1
elif condition2:
    instruction2:
else:
instruction3
```

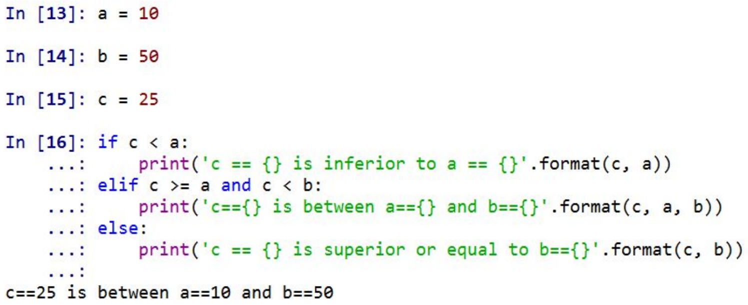
If statements: examples

Example - Print if a number is even or odd



```
In [12]: if a % 2 == 0:
    ...:     print('a == {} is a even'.format(a))
    ...: else:
    ...:     print('a == {} is odd'.format(a))
    ...:
a == 5 is odd
```

Example - Checking an inequality



```
In [13]: a = 10

In [14]: b = 50

In [15]: c = 25

In [16]: if c < a:
    ...:     print('c == {} is inferior to a == {}'.format(c, a))
    ...: elif c >= a and c < b:
    ...:     print('c=={} is between a=={} and b=={}'.format(c, a, b))
    ...: else:
    ...:     print('c == {} is superior or equal to b=={}'.format(c, b))
    ...:
c==25 is between a==10 and b==50
```

For loops

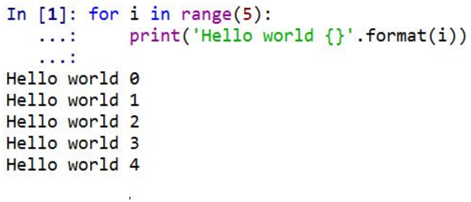
Goal: Do the same thing multiple times.

The syntax for `for` loops is as follows:

```
for i in range(n, m):
    instructions
```

For loops: example 1

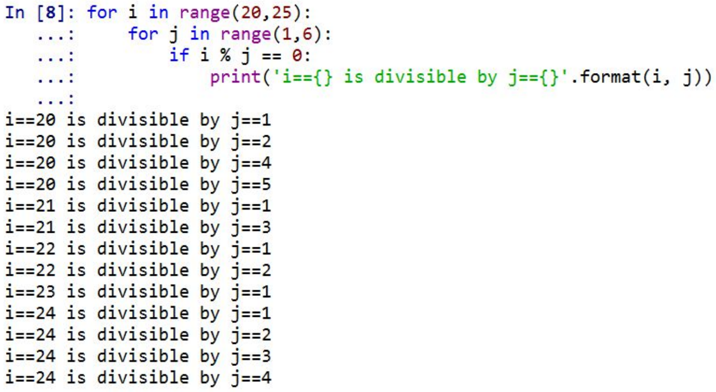
Example - Print "Hello world" 5 times



```
In [1]: for i in range(5):
   ...:     print('Hello world {}'.format(i))
   ...:
Hello world 0
Hello world 1
Hello world 2
Hello world 3
Hello world 4
```

For loops: example 2

Example - Check if numbers between 20 and 24 are divisible by numbers between 1 and 5



```
In [8]: for i in range(20,25):
   ...:     for j in range(1,6):
   ...:         if i % j == 0:
   ...:             print('i=={} is divisible by j=={}'.format(i, j))
   ...:
i==20 is divisible by j==1
i==20 is divisible by j==2
i==20 is divisible by j==4
i==20 is divisible by j==5
i==21 is divisible by j==1
i==21 is divisible by j==3
i==22 is divisible by j==1
i==22 is divisible by j==2
i==23 is divisible by j==1
i==24 is divisible by j==1
i==24 is divisible by j==2
i==24 is divisible by j==3
i==24 is divisible by j==4
```

While loops

The syntax for `while` loops is as follows:

```
while condition:
    instructions
```

- A while loop is useful when you do not know how many iterations you need to do.
- The while loop keeps running **as long as the** `condition` **is a boolean equal to True.**
- The boolean condition can be a function that returns a boolean.

Important: **Always check the while loop terminates** or suffer the consequences: your program will run forever and eventually crash your computer.

While loops: example

Example - Print "Hello world" 5 times

```
i = 0
while i < 5:
    print ("Hello world")
    i += 1

# Calculating a sum until a condition is reached i, sum = 0, 0
while sum < 100:
    i += 1
    sum += i
print ("i : {} sum : {}". format (i, sum))
```

Current Section

Sources

- https://www.lri.fr/~hivert/COURS/Methodo/python.pdf
- https://perso.limsi.fr/pointal/_media/python:cours:courspython3.pdf
- http://cs231n.github.io/python-numpy-tutorial/
- https://www.courspython.com/apprendre-numpy.html
- http://perso.numericable.fr/jules.svartz/prepa/
- http://alain.troesch.free.fr/

Thank you for your attention ! Let’s practice !