‘Data Analysis’
Computer Lab Session
General Introduction to Data Analysis with Python

Master 1 MLDM / CPS^2 / COSI / 3DMT
Saint-Étienne, France

Ievgen Redko

1 Outline

1. **Introduction to Python (1/2)**
2. Introduction to Python (2/2)
3. Probability, Random Variables and Probability Distributions
4. Linear Algebra (1/2)
5. Linear Algebra (2/2)
6. Principal Component Analysis
7. Linear Regression (1/2)
8. Linear Regression (2/2)
9. Clustering (1/2)
10. Clustering (2/2)

2 Working with Variables

We can use Python in different ways: in prompt, in IDE environments or in notebooks. The most basic case is to launch Python in the terminal. In this case, Python’s prompt can be used to do some simple operations.

```
>>> # We can use it as a calculator
... 1 + 2 + 3
6
>>> # or to manipulate variables
... a = 1 + 2 + 3
>>> b = 2.0 + 1
>>> # ... that can be used afterwards
```

```
... a
6
>>> b
3.0
>>> a * b # Multiplication
18.0
>>>
>>> a ** 2 # Power of 2
36
>>> a / 4 # Division
1
>>> a // 4 # Integer division
1
>>> a % 4 # Modulo
2
```

2.1 Variable Assignment

We assign values to variables with the assignment operator '='. This latter can be used together with different arithmetic operations .

```
a = 6
a += 2 # Means a = a + 2

b = 2.0
a /= b # Means a = a / b

a *= b + 1.0 # Means ... up to you to guess
```

2.2 Variable Printing

Simply putting the name of the variable in the code will print its value. As before, this can be combined with various arithmetic operations.

```
>>> a = 6
>>> b = 2.0
>>> a
6
>>>
>>> b
2.0
>>> a + b
8.0
>>> a - b
4.0
>>> a + b
8.0
>>> a - b;
4.0
>>> result = a - b
```

```
>>> result
4.0
```

2.3 Error messages

If something goes wrong, you will see an error message like in cases shown below.

```
>>> a / 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: float division by zero
# Aie aie aie who taught you to divide by 0?
>>>
>>> z
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'z' is not defined
# I do not know this one, bro.
>>>
>>> a +
  File "<stdin>", line 1
    a +
      ^
SyntaxError: invalid syntax
# What does that mean?
```

3 Basic Data Types

There are several basic Python data types that are of frequent occurrence in routine Python calculations.

3.1 Integer and real (float) types

```
>>> type(a) # Function to known the type of object
<type 'float'>
>>>
>>> type(b)
<type 'float'>
>>>
>>> type(a - b)
<type 'float'>
>>>
>>> b_prime = int(b) # Conversion of a real to an integer
>>> print(b, '==>', b_prime)
(2.0, '==>', 2)
>>> type(b_prime)
<type 'int'>
>>>
>>> float(a)
12.0
```

3.1.1 Logical

A **logical** value is often created via comparison between variables.

```
>>> v = True
>>> f = False
>>>
>>> type(v)
<type 'bool'>
>>>
>>> v and f
False
>>>
>>> v or f
True
>>>
>>> v and not f
True
>>>
>>> a == 6
False
>>>
>>> a < 6
False
>>>
>>> a >= 6
True
>>>
>>> (a == b or b < 3) and (a != -1 or False)
True
```

Standard logical operations are "and", "or", and "not".

3.1.2 Strings

```
>>> "salut"
'salut'
>>>
>>> one_word = "salut"
>>> type(one_word)
<type 'str'>
>>>
>>> len(one_word) # length of a string (ie number of characters)
5
>>>
>>> str(a) # Conversion to string
'12.0'
>>>
>>> print(str(a) + str(b))
12.02.0
>>> print(str(a), str(b))
('12.0', '2.0')
```

```
>>>
>>> float(str(a) + str(b)) # Conversion from string to real
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for float(): 12.02.0
>>>
>>> one_word[3] # 4th elements of a string (index starts at 0!)
'u'
>>>
>>> one_word[2:4] # Access to a substring
'lu'
>>>
>>> two_words = one_word + ' toi!'
>>> two_words
'salut toi!'
>>>
>>> len(two_words)
10
>>>
>>> two_words[0:len(one_word)] == one_word
True
```

3.2 Lists

```
>>> my_list = [2, 4, 6, 8] # List creation
>>>
>>> my_list
[2, 4, 6, 8]
>>>
>>> type(my_list)
<type 'list'>
>>>
>>> # Another way to initialize a list
... list_of_fives = [5] * 3
>>> list_of_fives
[5, 5, 5]
>>>
>>> my_list[0] # first element of a list
2
>>> my_list[-1] # Last elements of a list
8
>>>
>>> my_list[4] # Out of bounds index raises an error
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>>
>>> my_list[1:3] # Sublist [start:end]. WARNING: last element is not included
[4, 6]
>>>
>>> my_list[1:-1]
```

```
[4, 6]
>>>
>>> my_list[1:]
[4, 6, 8]
>>>
>>> my_list[0:3]
[2, 4, 6]
>>>
>>> my_list[:3]
[2, 4, 6]
>>>
>>> my_list[:-1]
[2, 4, 6]
>>>
>>> my_list[1:1]
[]
>>>
>>> my_list[3] = 10 # Modification of an elements
>>> my_list
[2, 4, 6, 10]
>>>
>>> 3 in my_list # Test of belonging to a list
False
>>>
>>> 4 in my_list
True
>>>
>>> my_phrase = ['A', 'list', 'can', 'contain', 'words']
>>> my_phrase
['A', 'list', 'can', 'contain', 'words']
>>>
>>> len(my_phrase) # Length of a list
9
>>>
>>> len(my_list)
4
>>>
>>> sum(my_list) # Sum of the elements
22
>>>
>>> empty_list = list()    # Empty list
>>> another_empty_list = [] # Another way to create an empty list
```

List's methods An object of type list has its own methods. To know them, put a dot after the name of the list and press TAB. Try it! It also works by simply writing list. and by pressing on TAB. As an example, try methods reverse, append et sort.

```
>>> my_list.reverse() # Inversion of a list
>>> my_list
[10, 6, 4, 2]
>>>
```

```
>>> my_list.append(5) # Adding an elements to a list
>>> my_list
[10, 6, 4, 2, 5]
>>>
>>> my_list.sort() # Sorting the list
>>> my_list
[2, 4, 5, 6, 10]
>>>
>>> my_phrase.sort() # Also works with words
>>> my_phrase.sort(reverse=True) # Sort in the decreasing order
>>> my_phrase
['words', 'list', 'contain', 'can', 'A']
>>>
>>> my_phrase.sort(key=len) # Sorting using the words length.
... my_phrase
['A', 'can', 'list', 'words', 'contain']
>>>
>>> my_phrase.sort(key=len, reverse=True)
>>> my_phrase
['contain', 'words', 'list', 'can', 'A']
>>>
# Pointer to my_list (WARNING: This is not a copy of a list!)
>>> some_list = my_list
>>> print(some_list == my_list)

>>> some_list[0] = 0
>>> print(some_list == my_list)

# Copy the content of the list
>>> a_copy = list(my_list) # you can also use my_list[:]
>>> print(a_copy == my_list)

>>> a_copy[0] = 3
>>> print(a_copy == my_list)

# Operator "is" allows to know if two lists point to the same object
>>> print('some_list␣is␣my_list␣:', some_list is my_list)
>>> print('a_copy␣is␣my_list␣:', a_copy is my_list)
```

3.3 Dictionaries

```
>>> my_dic
{'vingt-quatre': 24, 'trois': 3, 'quarante-deux': 42}
>>>
>>> my_dic['quatre'] = my_dic['trois'] + 1
>>> my_dic
{'quatre': 4, 'vingt-quatre': 24, 'trois': 3, 'quarante-deux': 42}
>>>
>>> print('*␣dictionary␣keys:␣', list(my_dic.keys()))
>>> print('*␣values␣of␣the␣keys:␣␣␣␣␣␣', list(my_dic.values()))
```

```
>>> print('* tuples (key, value): ', list(my_dic.items()))
>>>
>>> 'soixante-six' in my_dic
False
>>>
>>> 'quatre' in my_dic
True
>>>
>>> 4 in my_dic
False
```

4 Conditional structures

Indentation is **IMPORTANT** in Python.

4.1 if

```
>>> a = 2
>>> if a == 3:
...     a = 2*a
...     print(a)
```

```
>>> if a != 3: # Level 0 of indentation
...     print('Multiply by 2.') # Level 1 of indentation
...   a = 2*a   # ERROR
```

4.2 if, elif, else

```
if a < 4:
    print('a < 4')
elif a >= 6:
    if a == 6:
        print("a = 6")
    else:
        print("a > 6")
elif a - 5 == 0:
    print('a = 5')
else:
    print('a = 4')
```

5 Loops

5.1 Loop « for »

```
>>> for i in range(10):
...     print(i, 'squared is equal', i**2)

>>> for i in range(10, 14):
...     print(i, 'squared is equal ', i**2)
```

```
>>> my_list = [8, 2, 4, 10, 8]
>>> for x in my_list:
...     print(x, 'squared␣is␣equal␣', x**2)

>>> for i in range(len(my_list)):
...     x = my_list[i]
...     print('Index', i, ':␣', x, 'squared␣is␣equal␣', x**2)

>>> for i, x in enumerate(my_list):
...     print('Index', i, ':␣', x, 'squared␣is␣equal␣', x**2)

>>> word2num = dict()
>>> tuple_of_words = ('zero', 'one', 'two')
>>> for i, mot in enumerate(tuple_of_words):
...     word2num[mot] = i
...
>>> for key in word2num.keys():
...     value = word2num[key]
...     print(key, '==>', value)

>>> for key, value in word2num.items():
...     print(key, '==>', value)
```

5.2 Loop « while »

```
>>> fib = [1, 1]
>>> while fib[-1] < 100:
...     element = fib[-2] + fib[-1]
...     fib.append(element)
...
>>> print(fib)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144]
```

Exercises

Exercise 1

We use a list to represent a vector $\boldsymbol{v} \in \mathbb{R}^5$. Calculate the Euclidean norm of v defined as follows:

$$\|\boldsymbol{v}\| = \sqrt{\sum_{i=1}^{5} v_i^2}.$$

Exercise 2

We create another vector $\boldsymbol{u} \in \mathbb{R}^5$ and calculate its scalar product with $\boldsymbol{v}$ as follows:

$$\boldsymbol{u} \cdot \boldsymbol{v} = \sum_{i=1}^{5} u_i v_i.$$

Implement it using the following **for** loop with **zip** function.

```
>>> for x, y in zip(vector_u, vector_v):
...     print(x, ';', y)
```

Exercise 3

We represent a matrix $\boldsymbol{M} \in \mathbb{R}^{3\times5}$ as a list of lists. Create a matrix $\boldsymbol{M}^T \in \mathbb{R}^{3\times5}$ representing the transpose of M.

```
matrix_M = [[1, 3, 3, 2, 8], [2, 5, 3, 5, 1] , [1, 2, 3, 2, 1]]

print('Matrix M:')
for line in matrix_M:
    print(line)

matrix_M_T = []
nb_lines = len(matrix_M)
nb_columns = len(matrix_M[0])

# To complete

print('-----------------------------------')
print('Transpose of the matrix M:')
for line in matrix_M_T:
    print(line)
```

If everything went well, you can validate your solution using the following code:

```
nb_lines = len(matrix_M)
nb_columns = len(matrix_M[0])

assert len(matrix_M_T) == nb_columns, "Wrong number of rows"

for i in range(nb_lines):
```

```
    assert len(matrix_M_T[i]) == nb_lines, "Wrong number of columns"
    for j in range(nb_columns):
        assert matrix_M[i][j] == matrix_M_T[j][i], "Value inconsistency"

print('**Test passed!**')
```