CONSEIL ET EXPERTISE TECHNIQUE

Infrastructure ▪ Cybersecurity ▪ Cloud ▪ Data

Python 02

Matthieu DESTOMBES
Mail : matthieu.destombes@ynov.com

01 Mars 2021
Toulouse

Correction

La correction du TP se trouve dans le package

- « final_01.zip »
- Explication du corrigé

Théorie

- Les bases de Python (Avancé)

Les bases de Python (Part II)

- Les listes, les dictionnaires et les tuples
- La porté des variables
- Les interactions console
- L’arithmétique
- Les fonctions avancées
- La doc-chaîne
- Les modules et imports

Les bases de Python (Part II) => Les listes, les dictionnaires et les tuples

- Rappel
- Plusieurs types
  - Déjà connus
  - Les listes
  - Les dictionnaires
  - Les tuples

Les bases de Python (Part II)
=> Les listes

```
test_range_1 = range(3)
test_range_2 = range(10, 12)

print("List range 1: {0}".format(list(test_range_1)))
print("List range 2: {0}".format(list(test_range_2)))
```

```
List range 1: [0, 1, 2]
List range 2: [10, 11]
```

- Composition
- Création/Complétion
- Utilisation

```
test_list = ["One", "Two", "Three"]

test_list.append("Four")
test_list.append(["Five", "Six"])
```

```
print("List: {0}".format(test_list))
```

```
List: ['One', 'Two', 'Three', 'Four', ['Five', 'Six']]
```

```
print("Element 2 in List: {0}".format(test_list[1]))
```

```
Element 2 in List: Two
```

```
if 0 in test_range_1:
    print("Yes, it is!")
else:
    print("No, it is not!")
```

```
Yes, it is!
```

```
for element in test_list:
    print("Current element: {0}".format(element))
```

```
Current element: One
Current element: Two
Current element: Three
Current element: Four
Current element: ['Five', 'Six']
```

Les bases de Python (Part II)
=> Les dictionnaires

- Composition
- Création/Complétion
- Utilisation

```
dictionary_1 = {}
dictionary_2 = {'a': 10, 'b': 20}
dictionary_3 = {'A': 100, 'B': 200}

print("Dic 1:{0}".format(dictionary_1))
print("Dic 2:{0}".format(dictionary_2))
print("Dic 3:{0}".format(dictionary_3))
```

```
Dic 1:{}
Dic 2:{'a': 10, 'b': 20}
Dic 3:{'A': 100, 'B': 200}
```

```
dictionary_2['c'] = [0, 1, 2]
dictionary_2['d'] = dictionary_3

print("Dic 2:{0}".format(dictionary_2))
```

```
Dic 2:{'a': 10, 'b': 20, 'c': [0, 1, 2], 'd': {'A': 100, 'B': 200}}
```

```
print("Function 'items':{0}".format(dictionary_3.items()))
print("Function 'keys' :{0}".format(dictionary_3.keys()))
print("Function 'values':{0}".format(dictionary_3.values()))
```

```
Function 'items':dict_items([('A', 100), ('B', 200)])
Function 'keys' :dict_keys(['A', 'B'])
Function 'values':dict_values([100, 200])
```

Les bases de Python (Part II)
=> Les tuples

- Composition

```
my_tuple = ("Lower", 2, 10)

print("My tuple is: {0}".format(my_tuple))
print("My first value in tuple is: {0}".format(my_tuple[0]))
```

```
My tuple is: ('Lower', 2, 10)
My first value in tuple is: Lower
```

- Création/Complétion

```
for value in my_tuple:
    print("Value is: {0}".format(value))
```

```
Value is: Lower
Value is: 2
Value is: 10
```

- Utilisation

```
print(
    "I can say with no doubt => '{0}' is '{1}' than '{2}'!".format(
        my_tuple[1],
        my_tuple[0],
        my_tuple[2]
    )
)
```

```
I can say with no doubt => '2' is 'Lower' than '10'!
```


Les bases de Python (Part II)
=> La porté des variables

- Utilisable dans le bloc
- Possibilité d'utiliser une variable exté
  - global [nom_variable]

```python
my_outside_var = 0


def first_step():
    global my_outside_var

    my_outside_var += 1


def second_step():
    global my_outside_var

    my_outside_var = 10


print('\n')

print("Original                 : {0}".format(my_outside_var))

first_step()
print("After running first step : {0}".format(my_outside_var))

second_step()
print("After running second step: {0}".format(my_outside_var))
```

```
Original                 : 0
After running first step : 1
After running second step: 10
```

Les bases de Python (Part II)
=> Les interactions console

```python
input_value_1 = input()
input_value_2 = input("Enter something: ")

print("Value 1: '{0}'".format(input_value_1))
print("Value 2: '{0}'".format(input_value_2))
```

- Fonction interne
  - input()
  - input("Something to say")

```
56
Enter something: 54


Value 1: '56'
Value 2: '54'
```


Les bases de Python (Part II) => L'arithmétique 1

```
print("{0} Plus {1} : {2}".format(
    my_values[0],
    my_values[3],
    my_values[0] + my_values[3]))
print("{0} Plus {1} : {2}".format(
    my_values[1],
    my_values[3],
    my_values[1] + my_values[3]))
```

```
0 Plus 10 : 10
0.0 Plus 10 : 10.0
```

```
my_values = [0, 0.0, 5, 10, 2, 2.0, 3]

print("All of my values: {0}".format(my_values))
```

```
All of my values: [0, 0.0, 5, 10, 2, 2.0, 3]
```

- Addition (+)
- Soustraction (-)

```
print("{0} Minus {1} : {2}".format(
    my_values[2],
    my_values[0],
    my_values[2] - my_values[0]))
print("{0} Minus {1} : {2}".format(
    my_values[2],
    my_values[1],
    my_values[2] - my_values[1]))
```

```
5 Minus 0 : 5
5 Minus 0.0 : 5.0
```

```
print("{0} Multiplied by {1} : {2}".format(
    my_values[2],
    my_values[4],
    my_values[2] * my_values[4]))
print("{0} Multiplied by {1} : {2}".format(
    my_values[2],
    my_values[5],
    my_values[2] * my_values[5]))
```

```
5 Multiplied by 2 : 10
5 Multiplied by 2.0 : 10.0
```

- Multiplication (*)
- Incrémentation (+=)

```
temp_value = my_values[6]
temp_value += my_values[4]
print("{0} Incremented by {1}: {2}".format(
    my_values[6],
    my_values[4],
    temp_value))
```

```
3 Incremented by 2: 5
```


Les bases de Python (Part II) => L'arithmétique 2

```
print("{0} Divided by {1} : {2}".format(
    my_values[3],
    my_values[4],
    my_values[3] / my_values[4]))
print("{0} Divided by {1} : {2}".format(
    my_values[2],
    my_values[4],
    my_values[2] / my_values[4]))
```

```
10 Divided by 2 : 5.0
5 Divided by 2 : 2.5
```

```
my_values = [0, 0.0, 5, 10, 2, 2.0, 3]

print("All of my values: {0}".format(my_values))
```

```
All of my values: [0, 0.0, 5, 10, 2, 2.0, 3]
```

- Division (/)
- Modulo (%)

```
print("{0} Modulo {1} : {2}".format(
    my_values[3],
    my_values[4],
    my_values[3] % my_values[4]))
print("{0} Modulo {1} : {2}".format(
    my_values[2],
    my_values[4],
    my_values[2] % my_values[4]))
```

```
10 Modulo 2 : 0
5 Modulo 2 : 1
```

- Puissance (**)
- Décrémentation (-=)

```
print("{0} To the power of {1} : {2}".format(
    my_values[3],
    my_values[4],
    my_values[3] ** my_values[4]))
print("{0} To the power of {1} : {2}".format(
    my_values[2],
    my_values[4],
    my_values[2] ** my_values[4]))
```

```
10 To the power of 2 : 100
5 To the power of 2 : 25
```

```
temp_value = my_values[6]
temp_value -= my_values[4]
print("{0} Decremented by {1}: {2}".format(
    my_values[6],
    my_values[4],
    temp_value))
```

```
3 Decremented by 2: 1
```


Les bases de Python (Part II)
=> Les fonctions avancées

```python
def my_function(
    mandatory_args,
    optional_args=None
):

    print("Mandatory arguments is: {0}".format(mandatory_args))

    if optional_args is not None:
        print("Optional arguments is: {0}".format(optional_args))
        process_is = "set"

    else:
        print(
            "Optional arguments is set to default value as: {0}".format(
                optional_args
            )
        )
        process_is = "default"

    return process_is, [mandatory_args, optional_args]
```

- Gestion des arguments
- Gestion des retours

```python
print("Function call 1 with {0} and {1}:\n{2}".format(
    10,
    20,
    my_function(10, 20)
))
```

```
Mandatory arguments is: 10
Optional arguments is: 20
Function call 1 with 10 and 20:
('set', [10, 20])
```

```python
print("Function call 1 with {0}:\n{1}".format(
    10,
    my_function(10)
))
```

```
Mandatory arguments is: 10
Optional arguments is set to default value as: None
Function call 1 with 10:
('default', [10, None])
```


Les bases de Python (Part II)
=> La doc-chaîne

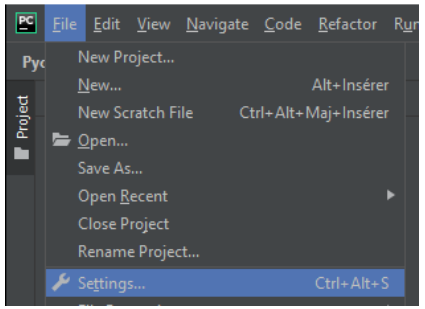
- Utilité
- Composition

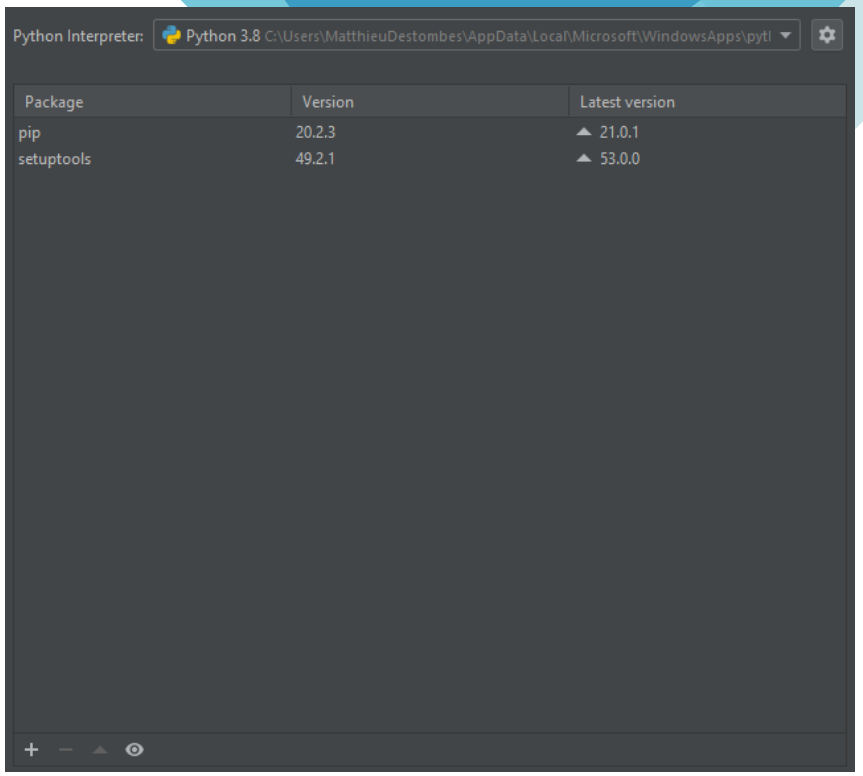
```python
def my_function(
    mandatory_args,
    optional_args=None
):
    """
    This is my first example function. There is different behavior depending on
      the assignment of the second factual argument.

    :param mandatory_args: Integer - This a mandatory argument.
    :param optional_args: Integer - This a optional argument.

    :return: Tuple - The process, The mandatory_args value,
      The optional_args value.

    """

    print("Mandatory arguments is: {0}".format(mandatory_args))
```

```python
print("Function call 1 with {0}:\n{1}".format(
    10,
    my_function(10)
))
```

```
test.for_diapo
def my_function(mandatory_args: Any,
                optional_args: Any = None) -> Tuple[str, List[Optional[Any]]]

This is my first example function. There is different behavior depending on
      the assignment of the second factual argument.

Params: mandatory_args – Integer - This a mandatory argument.
        optional_args – Integer - This a optional argument.

Returns: Tuple - The process, The mandatory_args value, The optional_args value.
```


Les bases de Python (Part II)
=> Les modules et imports

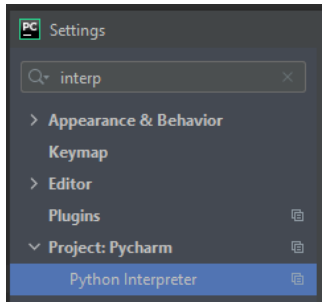




- Contenu
- De nouveau modules

```
$> pip3 install [le_module]
```

- Utilisation




```python
import random

for current_run in range(5):
    my_random_value = random.randint(0, 100)
    print("My random value for run '{0}' is: {1}".format(
        current_run + 1,
        my_random_value
    ))
```

```
My random value for run '1' is: 50
My random value for run '2' is: 3
My random value for run '3' is: 14
My random value for run '4' is: 66
My random value for run '5' is: 44
```

