Cours Introduction à la Programmation Python II

(IP1 Python)

Arnaud Sangnier
sangnier@irif.fr

Mercredi 20 Septembre 2017
MIASHS et MATHS

Dans l'épisode précédent - I

- Les programmes s'écrivent dans des langages
- Cette année : **Python 3**
- Pour écrire un programme, on utilise un éditeur
- Le programme est interprété en lançant l'interpréteur, par exemple

```
python3 monprogramme.py
```

- Un programme est composé de différentes instructions
- Pour qu'un programme affiche quelque chose, il faut qu'on lui dise, par exemple :

```
print ("Hello World !")
```

Dans l'épisode précédent - II

- Un programme manipule des données auxquelles sont associées un type
- Nous avons vu deux types : **int** et **str**
- Il faut toujours savoir quel est le type de la donnée manipulée
- On peut convertir une chaîne de caractère en entier :
  - int("22") donne l'entier 22
- Et un entier en chaîne de caractères
  - str(22) donne la chaîne "22"

Dans l'épisode précédent - III

- Présentation des variables
- Elle servent à stocker des données
- Un nom de variable commence par une lettre
- Trois opérations possibles
  - **Affectation**, **lecture** et **modification**
- Le signe = sert à affecter ou à modifier et n'est pas comme l'égal mathématique
- Pour interpréter x = y + 5 , le programme calcule d'abord y + 5 et mais ensuite le résultat dans x

Plusieurs valeurs possibles mais un seul programme

- Une clef essentielle de la programmation est d'écrire des programmes qui vont marcher pour différentes valeurs possibles
- Par exemple :
  - Un programme qui calcule n! (factorielle de n)
  - Un programme qui calcule la somme de deux entiers a et b
  - Un programme qui calcule le pgcd de deux entiers a et b
- Ici les valeurs de n, a et b ne sont pas connus à l'avance, on sait juste qu'il s'agit d'entier

Les fonctions

- Une fonction d'un programme est une liste d'instructions
- Elle peut être appelée plusieurs fois
- Elle peut prendre des valeurs en entrée
  - Il s'agit des **arguments**
- Elle peut calculer une valeur et la renvoyer
  - Il s'agit de la valeur de retour
- On lui donne un nom (le nom de fonctions)
- Exemple : la fonction print( ) qui ne retourne pas de valeur mais qui affiche à l'écran

Exemple de fonctions

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

a = f(3)
b = f(5)

print (a)
print(b)
```

Définition de la fonction f

Appel de la fonction f

- Affiche

```
6
10
```

- **Si on enlève les print, le programme n'affiche rien**

Que fait la machine ?

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

- Si par exemple on a une ligne z = f (4)

  1) Elle remplace la valeur x de f par 4

  2) Elle calcule 2*4

  3) Elle renvoie la valeur 8

  4)Elle stocke cette valeur dans z

Exemple

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

z = 10
z = f(z)

print (z)
```

- Affiche

```
20
```

Exemple

```
def f (x) :
      x = x + 1
      return (2 * x)

x = 10
z = f(x)

print (x)
print (z)
```

ATTENTION :
Les deux x ne sont pas la même variable

- Affiche

```
10
22
```

Exemple

```
def f (x) :
      x = x + 1
      return (2 * x)

x = 10
z = f(x)

print (x)
print (z)
```

- Plus sûr en écrivant :

```
def f (y) :
      y = y + 1
      return (2 * y)

x = 10
z = f(x)

print (x)
print (z)
```

On évite ainsi les confusions possibles

Les fonctions sans return

- Dans certains cas, on peut vouloir qu'une fonction ne retourne aucune valeur
- Il n'y a pas de return dans le code
- Pourquoi faire ?
  - Par exemple une fonction d'affichage
  - On peut vouloir afficher un message 'paramétré' par un argument
  - Par exemple, afficher le message Hello nom où nom est remplacé par un nom donné en argument
- Si dans une fonction f, il n'y a pas de return, il ne faut pas faire x = f () → **cela n'a aucun sens** !

Exemple

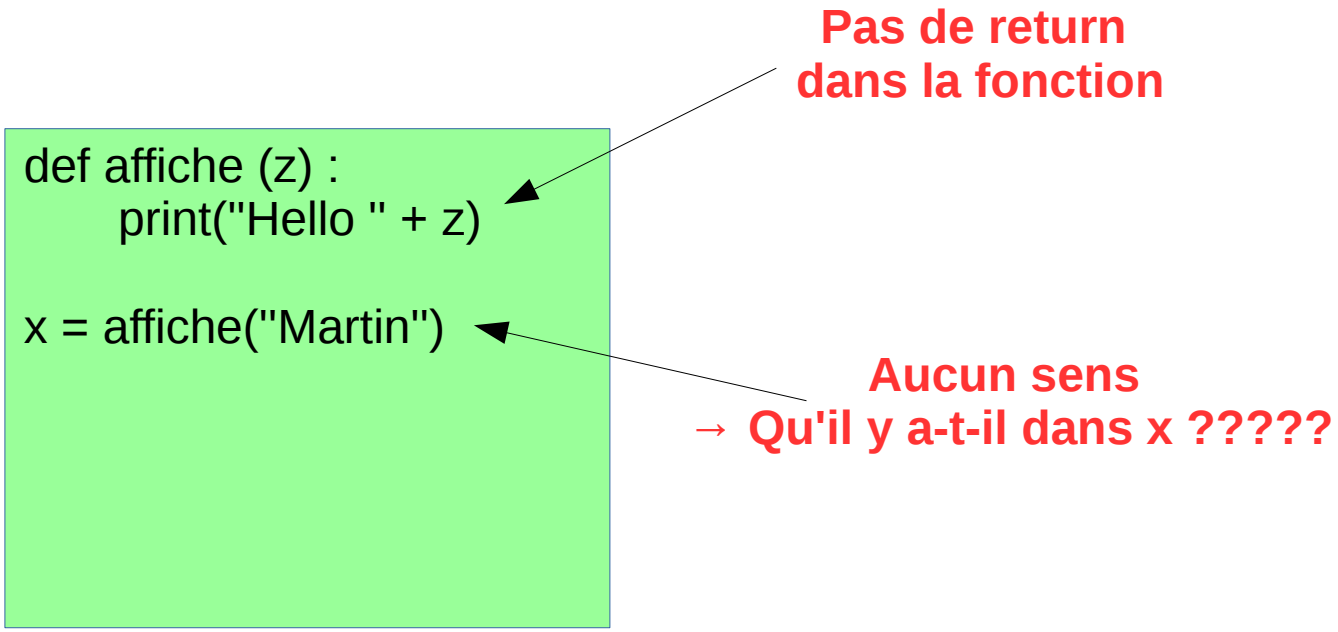
```
def affiche (z) :
        print("Hello " + z)

affiche("Bob")
affiche("Alice")
```

- Affiche

```
Hello Bob
Hello Alice
```

Exemple



Il faut toujours savoir si une fonction renvoie une valeur et aussi quel est le type de la valeur retournée

Les tests

- Comme un programme doit être paramètrable selon les valeurs qu'on lui donne, il est important de pouvoir tester ces valeurs
- Pour cela on a l'instruction if … else …
- Par exemple, pour dire si la valeur dans x est positive ou nulle fait quelque chose sinon fait autre chose
- Pour tester ou comparer deux entiers on a les opérateurs : <, >, >=,<=, ==,!=
- Pour tester ou comparer deux chaînes de caractères, on a les opérateur : == et !=

Exemple

```
if (x == 5) :
        print ("AH")
else :
        print("OH")
```

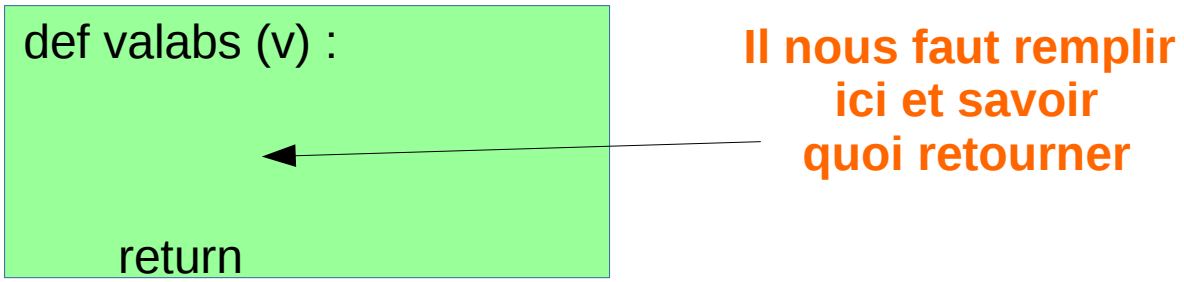
Affiche AH si x vaut 5 et OH sinon

```
if (v == "Hi") :
        print("Salut")
else :
        print("Que veux-tu"?)
```

Affiche Salut si v vaut "Hi" et sinon affiche Que veux-tu ?

Une première fonction

- Énoncé : faire une fonction qui renvoie la valeur absolue d'un entier
- D'abord déterminer les informations suivantes :
  - Combien de paramètres : un seul qui prend des valeurs de type int
  - Retourne-t-elle une valeur : oui de type int
  - Comment va-t-on l'appeler : par exemple valabs



Une première fonction

- Énoncé : faire une fonction qui renvoie la valeur absolue d'un entier
- Comment fonctionne la valeur absolue :
  - Si la la valeur de v est positive ou nulle, c'est la valeur de v
  - Si la valeur de v est négative, c'est – la valeur de v

```
def valabs (v) :
        x = 0
        if (v >=0) :
                x = v
        else :
                x = -v
        return x
```

Exemple

```
def valabs (v) :
        x = 0
        if (v >=0) :
                x = v
        else :
                x = -v
        return x

y = valabs(-10)
z = valabs(12)
print(y)
print(z)
```

- Affiche

```
10
12
```

Une deuxième fonction

- Énoncé : faire une fonction qui prend en arguments deux chaines de caractères et teste si elles ont la même longueur et affiche 'Meme' si c'est le cas et 'Different' sinon
- On dispose de la fonction len() qui prend en arguments une chaîne de caractères et renvoie sa longueur
- Déterminer les informations suivantes :
  - Combien de paramètres : deux qui prennent des valeurs de type str
  - Retourne-t-elle une valeur : non
  - Comment va-t-on l'appeler : par exemple complen

Exemple

```
def complen (s1,s2) :
        if (len(s1) == len(s2)) :
                print("Meme")
        else :
                print("Different")

complen("Bob", "Lol")
complen("a","ab")
```

- Affiche

```
Meme
Different
```

Les boucles

- Une autre chose intéressante sont les boucles
- Un programme répète souvent les mêmes instructions en boucle
- Il existe une instruction pour dire au programme de répéter une instruction un certain nombre de fois
  - Par exemple for i in range (0, 10, 1)
  - En français, il faut lire pour i allant de 0 (inclus) à 10 (exclus) en augmentant de 1 à chaque étape
- Par exemple pour dire à un programme d'afficher 1000 fois Bonjour
  - On peut copier-coller 1000 fois print("Bonjour") → 
  - On peut utiliser une boucle

Exemple

```
for i in range (0, 1000, 1) :
    print("Bonjour")
```

- Affiche

```
Bonjour
Bonjour
Bonjour
Bonjour
Bonjour
.
.
Bonjour
```

Piège avec les boucles

- Il faut être attentif à combien de fois on fait la boucle
  - Par exemple for i in range (1,9,1) → on fait la boucle 8 fois
  - for i in range (1,10,2) → on fait la boucle 5 fois (valeur de i : 1,3,5,7,9)
- La variable i dans for i in range (1,10,1) :
  - Est une variable normale, dont la valeur change à chaque tour de boucle
  - **Il ne faut pas la modifier dans la boucle for**

Exemple

```
for i in range (0, 4, 1) :
        print(i)
        print(2*i)
```

- Affiche

```
0
0
1
2
2
4
3
6
```

Utilité des boucles

- Est-ce-que les boucles ne servent qu'à afficher ? NON
- Elles sont aussi utiles
  - pour faire des calculs mathématiques (par exemple somme des n premiers entiers, factorielle, etc)
  - pour parcourir des listes (on verra ça plus tard)
- Le comportement répétitif est vraiment la base de la plupart des programmes
  - Pensez à votre ordinateur ou tablette, le programme sous-jacent fait tout le temps la même chose, il attend que vous lanciez d'autres programmes ou que vous fassiez une action

Encore une fonction

- Énoncé : faire une fonction qui prend en arguments un entier n et renvoie la somme des entiers de 1 à n
- Déterminer les informations suivantes :
  - Combien de paramètres : un qui prend une valeur de type int
  - Retourne-t-elle une valeur : oui une valeur de type int
  - Comment va-t-on l'appeler : par exemple sumN
- Comment la programmer
  - Il faut ajouter 1 puis 2 puis 3 puis 4 etc jusqu'à n
  - On a l'impression qu'il y a une boucle
  - À chaque fois on ajoute i
  - Où stocker le résultat de ce calcul → Utilisons une variable

Exemple

Observez bien les bornes

```
def sumN (n) :
        x = 0
        for i in range(1,n+1,1) :
                        x = x + i
        return x

y = sumN (10)
print (y)
```

- Affiche

```
55
```

Une fonction classique

- Énoncé : faire une fonction qui prend en arguments un entier n et renvoie factorielle de
- Déterminer les informations suivantes :
  - Combien de paramètres : un qui prend une valeur de type int
  - Retourne-t-elle une valeur : oui une valeur de type int
  - Comment va-t-on l'appeler : par exemple fact
- Comment la programmer → Très similaire à la fonction précédente
  - On multiplie 1 par 2 puis par 3 puis par 4 etc jusqu'à n
  - On a l'impression qu'il y a une boucle
  - À chaque fois on ajoute i
  - Où stocker le résultat de ce calcul → Utilisons une variable

Exemple

```
def fact (n) :
        x = 0
        for i in range(1,n+1,1) :
                        x = x * i
        return x

y = fact (4)
print (y)
```

**Remplace + par *
Et le tour est joué**

- Affiche

```
24
```


FAUX

Exemple

```
def fact (n) :
        x = 0
        for i in range(1,n+1,1) :
                x = x * i
        return x

y = fact (4)
print (y)
```

La variable est mal initialisée
Le programmera affichera 0

- Affiche

```
0
```

Exemple

```
def fact (n) :
        x = 1
        for i in range(1,n+1,1) :
                        x = x * i
        return x

y = fact (4)
print (y)
```

- Affiche

```
24
```

Combiner les différents types d'instructions

- Comme on l'a vu on peut utiliser des boucles et des tests au sein d'une fonction
- On peut globalement faire toutes les combinations possibles
  - Utilisation de tests dans une boucle
  - De boucle au milieu d'un test
- On peut aussi utiliser une fonction à l'intérieur d'une fonction qui elle même utilise une fonction etc