Initiation à Python

Séance 2

Valentin Bahier

2020-2021

Plan

Boucles

- Boucle while
- Boucle for

Fonctions

- Des fonctions prédéfinies
- Créer une fonction

Techniques et méthodes

- pour les nombres
- pour les chaînes de caractères
- pour les listes
- pour les dictionnaires

Plan

Boucles
- Boucle while
- Boucle for

Fonctions
- Des fonctions prédéfinies
- Créer une fonction

Techniques et méthodes
- pour les nombres
- pour les chaînes de caractères
- pour les listes
- pour les dictionnaires

Plan

Boucles
- Boucle while
- Boucle for

Fonctions
- Des fonctions prédéfinies
- Créer une fonction

Techniques et méthodes
- pour les nombres
- pour les chaînes de caractères
- pour les listes
- pour les dictionnaires

Boucle while

Une **boucle** est un procédé permettant de répéter des mêmes instructions d'un programme sans avoir à les récrire.

Une **boucle while** répète des instructions tant qu'une condition est satisfaite.

```
while condition:
    bloc d'instructions
```

⚠ Le bloc d'instructions est indenté.
⚠ Risque de boucle infinie si la condition est toujours satisfaite.

Exercice 8

Exécuter le code suivant et observer le résultat

```
1 k = 1
2 while k < 10:
3     print("La valeur de k est", k)
4     k = k * 2
5 print("Fin")
```

Exercice 8

Exécuter le code suivant et observer le résultat

```
1 k = 1
2 while k < 10:
3     print("La valeur de k est", k)
4     k = k * 2
5 print("Fin")
```

Réponse : Tant que k est inférieur à 10, on affiche la valeur de k puis on multiplie k par 2.

$\Rightarrow$ Pas besoin de savoir à l'avance combien de fois le bloc d'instructions va être exécuté.

Exercice 9

Créer un programme qui affiche tous les carrés parfaits (1, 4, 9, 16, etc...) jusqu'à 1000.

Exercice 9

Créer un programme qui affiche tous les carrés parfaits (1, 4, 9, 16, etc...) jusqu'à 1000.

Réponse :

```
k = 1
while k**2 < 1000:
    print("Le carré de", k, "vaut", k**2)
    k = k + 1
print("Fin")
```

Plan

Boucles
- Boucle while
- Boucle for

Fonctions
- Des fonctions prédéfinies
- Créer une fonction

Techniques et méthodes
- pour les nombres
- pour les chaînes de caractères
- pour les listes
- pour les dictionnaires

Boucle for

Comme la boucle while, la **boucle for** répète des instructions, mais cette fois-ci de manière bornée, c'est-à-dire qu'on connaît à l'avance le nombre de répétitions.

```
for élément in ensemble:
    bloc d'instructions
```

⚠ Le bloc d'instructions est indenté.

Exercice 10

Exécuter le code suivant et comprendre ce qu'il fait

```
for i in range(5):
  print(i)
for mot in ["je","vais","très","bien"]:
  print(mot)
for j in range(7,10):
  print(j)
for i in "fantastique":
  print(i)
```

Exercice 10

Exécuter le code suivant et comprendre ce qu'il fait

```
1 for i in range(5):
2   print(i)
3 for mot in ["je","vais","très","bien"]:
4   print(mot)
5 for j in range(7,10):
6   print(j)
7 for i in "fantastique":
8   print(i)
```

Réponse : `range(n)` est l'ensemble des entiers de 0 à $n-1$.
`range(m,n)` est l'ensemble des entiers de m à $n-1$.

$\Rightarrow$ L'élément situé juste après le `for` joue le rôle d'*indice muet*, on peut lui donner le nom que l'on veut (lettres majuscules et minuscules, sans espace) et n'est pas stocké en mémoire.

Plan

Boucles
- Boucle while
- Boucle for

Fonctions
- Des fonctions prédéfinies
- Créer une fonction

Techniques et méthodes
- pour les nombres
- pour les chaînes de caractères
- pour les listes
- pour les dictionnaires

Plan

Boucles
- Boucle while
- Boucle for

Fonctions
- Des fonctions prédéfinies
- Créer une fonction

Techniques et méthodes
- pour les nombres
- pour les chaînes de caractères
- pour les listes
- pour les dictionnaires

Fonctions standards

Nous avons déjà eu l'occasion de voir pendant la première séance quelques fonctions natives dans le langage Python :
`print` , `type` , `len` , `not`
Il y en a beaucoup d'autres (voir `https://docs.python.org/fr/3.9/library/functions.html`).

Une fonction utile en pratique est la fonction `help` appliquée à une fonction qu'on ne sait pas utiliser, permettant d'afficher la documentation de la fonction en question.

Syntaxiquement, on met les arguments des fonctions entre parenthèses après les fonctions, même quand il n'y en a pas :
`une_fonction_sans_argument()`

Quelques fonctions utiles

`str`	transforme en chaîne de caractères
`int`	transforme en nombre entier
`float`	transforme en nombre décimal
`input`	demande une valeur à l'utilisateur

Exercice 11

Entrer les instructions suivantes dans la console

```
1  str(26)
2  float("26")
3  int("26")
4  int(26.7)
5  int(-26.7)
6  a = input("donner un nombre entre 1 et 10 : ")
7  8
8  a
9  type(a)
10 a = int(a)
```

Exercice 11

Entrer les instructions suivantes dans la console

```
1  str(26)
2  float("26")
3  int("26")
4  int(26.7)
5  int(-26.7)
6  a = input("donner un nombre entre 1 et 10 : ")
7  8
8  a
9  type(a)
10 a = int(a)
```

Réponse : À la fin, a contient l'entier 8.

$\Rightarrow$ La fonction `input` affiche la chaîne de caractères mise en argument, et renvoie une chaîne de caractères.

Plan

Boucles
- Boucle while
- Boucle for

Fonctions
- Des fonctions prédéfinies
- Créer une fonction

Techniques et méthodes
- pour les nombres
- pour les chaînes de caractères
- pour les listes
- pour les dictionnaires

Définir une fonction

Pour créer une fonction, la syntaxe est la suivante

```
def ma_fonction( arguments éventuels ):
  bloc d'instructions
```

Les arguments (s'il y en a plusieurs) sont séparés par des virgules.

⚠ Le bloc d'instructions est indenté.
⚠ Les parenthèses sont nécessaires, même si la fonction ne prend pas d'argument.

Exercice 12

- Exécuter le programme suivant

```
1 def fonction_test():
2     a = 1
3 a = fonction_test()
4 print(a)
```

Que se passe t-il ?

- Réessayer après avoir inséré `return a` à la fin de la fonction, comme ceci

```
1 def fonction_test():
2     a = 1
3     return a
4 a = fonction_test()
5 print(a)
```

Exercice 12

- Exécuter le programme suivant

```
1 def fonction_test():
2     a = 1
3 a = fonction_test()
4 print(a)
```

Que se passe t-il ?

- Réessayer après avoir inséré `return a` à la fin de la fonction, comme ceci

```
1 def fonction_test():
2     a = 1
3     return a
4 a = fonction_test()
5 print(a)
```

Réponse : a devient connue en ajoutant la commande `return` .
$\Rightarrow$ Dans le bloc d'instructions de la fonction que l'on définit, on décrit simplement le mécanisme de la fonction, mais aucune variable n'est stockée. On dit que les variables utilisées dans la définition de la fonction sont **locales** (par opposition à **globales**).

Exercice 13

Créer une fonction `perimetre` qui prend en arguments les longueurs des côtés d'un triangle, et qui renvoie le périmètre de ce triangle.

Exercice 13

Créer une fonction `perimetre` qui prend en arguments les longueurs des côtés d'un triangle, et qui renvoie le périmètre de ce triangle.

Réponse :

```
1 def perimetre(a,b,c):
2     return a + b + c
```

Dans la console on peut tester la fonction que l'on vient de créer, en tapant par exemple

```
1 perimetre(5,6,9) #cela devrait afficher 20
2 perimetre(3,1,3.2) #cela devrait afficher 7.2
```

$\Rightarrow$ Les fonctions créées fonctionnent avec autant de types de paramètres auxquels il est possible de les appliquer. (Ici on pourrait même prendre trois chaînes de caractères pour `a`, `b` et `c`).

Plan

Boucles
- Boucle while
- Boucle for

Fonctions
- Des fonctions prédéfinies
- Créer une fonction

Techniques et méthodes
- pour les nombres
- pour les chaînes de caractères
- pour les listes
- pour les dictionnaires

Techniques ? Méthodes ?

Nous enrichissons maintenant nos techniques pour travailler avec des données. Souvent, les techniques passent par des **méthodes**, c'est-à-dire des fonctions qui sont propres à une certaine **classe d'objets**. Dans ce cours nous n'irons pas plus loin que de considérer les *méthodes* comme des fonctions particulières, s'écrivant après les objets, comme ceci

objet`.une_methode(`arguments éventuels `)`

Plan

Boucles
- Boucle while
- Boucle for

Fonctions
- Des fonctions prédéfinies
- Créer une fonction

Techniques et méthodes
- pour les nombres
- pour les chaînes de caractères
- pour les listes
- pour les dictionnaires

Compteurs

Un **compteur** est une variable numérique (habituellement de type `int`) qu'on utilise pour compter par exemple le nombre de fois qu'une instruction a été répétée dans une boucle.

`a += 1`	**incrémente** a de 1
`a -= 1`	**décrémente** a de 1

De manière plus générale on a les raccourcis d'écriture

`a += x`	`a = a + x`
`a -= x`	`a = a - x`
`a *= x`	`a = a * x`
`a /= x`	`a = a / x`

Plan

Boucles
- Boucle while
- Boucle for

Fonctions
- Des fonctions prédéfinies
- Créer une fonction

Techniques et méthodes
- pour les nombres
- pour les chaînes de caractères
- pour les listes
- pour les dictionnaires

Concaténation et sous-chaînes

Nous avons déjà vu que la concaténation de deux chaînes de caractères s'effectue avec le symbole +. Je vous laisse deviner ce que fait le symbole * entre une chaîne de caractères et un nombre entier...

À la manière des listes on peut aussi cibler une partie des caractères d'une chaîne

```
"Bonjour"[1:3]
```
on

```
"Bonjour"[3:]
```
jour

Si c'est le bleu c'est pour tes yeux

Les chaînes de caractères n'ont pas d'attribut de style (couleur, police, gras, etc..) donc on ne peut pas modifier leur style. En revanche, on peut passer des lettres en majuscules/minuscules. Je vous laisse compléter le tableau ci-dessous vous-même (on applique les méthodes à une chaîne de caractères `s`) :

`s.upper()`	
`s.lower()`	
`s.capitalize()`	

La méthode `.format`

Exercice 14

Entrer les instructions suivantes dans la console

```
"{0} {2}-{1} ?".format("comment", "tu", "vas")
'{0}{1}{0}{0} chapeau poin{0}'.format('tu', 'rlu')
"Le chameau a {chameau} bosses, mais le dromadaire
    seulement {dromadaire}".format(dromadaire="1",
    chameau="2")
```

Plan

Boucles
- Boucle while
- Boucle for

Fonctions
- Des fonctions prédéfinies
- Créer une fonction

Techniques et méthodes
- pour les nombres
- pour les chaînes de caractères
- pour les listes
- pour les dictionnaires

Ajouter ou supprimer des éléments d'une liste

Le symbole + permet aussi de concaténer des listes.

Les principales méthodes sur une liste `L` sont les suivantes :

`L.append(x)`	ajoute l'élément `x` à la fin de la liste
`L.insert(i,x)`	ajoute l'élément `x` en indice `i`
`L.pop()`	supprime le dernier élément
`L.remove(x)`	supprime la première occurrence de `x`
`L.index(x)`	donne l'indice du premier élément valant `x`

Exercice 15

Créer une fonction `parfaits` qui prend en argument un nombre entier `n` et renvoie une liste contenant tous les carrés parfaits inférieurs à `n`.

Exercice 15

Créer une fonction `parfaits` qui prend en argument un nombre entier `n` et renvoie une liste contenant tous les carrés parfaits inférieurs à `n`.

Réponse :

```
def parfaits(n):
  L = []  # initialisation de la liste
  k = 1  # compteur pour la boucle while
  while k**2 < n:
    L.append(k**2)  # ajoute le carré dans la liste
    k += 1  # incrémentation
  return L
```

Exercice 16

Créer une fonction `rotation` qui prend en entrée une liste `L` et qui renvoie la liste obtenue en déplaçant le dernier élément de `L` en première position. (Exemple : on veut que `["a","b","c"]` devienne `["c","a","b"]`)

Exercice 16

Créer une fonction `rotation` qui prend en entrée une liste `L` et qui renvoie la liste obtenue en déplaçant le dernier élément de `L` en première position. (Exemple : on veut que `["a","b","c"]` devienne `["c","a","b"]`)

Réponse : Une première possibilité :

```
1 def rotation(L):
2     M = [L[-1]]  # liste contenant le dernier élément
    de L
3     for i in range(len(L)-1):
4         M.append(L[i])  # ajoute le i-ème élément de
    la liste L à M
5     return M
```

Une deuxième possibilité :

```
1 def rotation(L):
2     M = [L[-1]] + L  # concaténation
3     M.pop()  # suppression du dernier élément
4     return M
```

Plan

Boucles
- Boucle while
- Boucle for

Fonctions
- Des fonctions prédéfinies
- Créer une fonction

Techniques et méthodes
- pour les nombres
- pour les chaînes de caractères
- pour les listes
- pour les dictionnaires

Modifier un dictionnaire

Pour ajouter un élément (une paire clé: valeur) dans un dictionnaire `D`, rien de plus simple :

`D[`clé`]` = valeur

Pour supprimer l'élément clé: valeur de `D` :

`D.pop(`clé`)`

Pour mettre à jour plusieurs valeurs en même temps :

`D.update(`clé: nouvelle valeur, clé2: nouvelle valeur2, ...`)`

Exercice 17

Créer une fonction `ajout_contact` qui ajoute à un annuaire un nom associé à un numéro de téléphone. (La fonction prendra donc trois arguments : un dictionnaire et deux chaînes de caractères).

Exercice 17

Créer une fonction `ajout_contact` qui ajoute à un annuaire un nom associé à un numéro de téléphone. (La fonction prendra donc trois arguments : un dictionnaire et deux chaînes de caractères).

Réponse : ⚠ Le piège à éviter est d'écrire :

```
1 def ajout_contact(D,nom,num):
2     D[nom] = num
```

En effet, en faisant cela on altère définitivement le dictionnaire auquel on applique la fonction, ce qui n'est pas ce qu'on cherche. Une bonne réponse :

```
1 def ajout_contact(D,nom,num):
2     E = D.copy()  # même "E = D" ne serait pas bon !
3     E[nom] = num
4     return E
```

⇒ La méthode `.copy()` évite la redoutable liaison de variables.

Parcourir un dictionnaire

Exercice 18

Exécuter les instructions suivantes et comprendre ce qu'elles font

```
1 D = {"forme": "hexagone", "couleur": "jaune"}
2 for i in D:
3     print(i)
4 for i in D.values():
5     print(i)
6 for i,j in D.items():
7     print(i,j)
```

Exercice 18

Exécuter les instructions suivantes et comprendre ce qu'elles font

```
1 D = {"forme": "hexagone", "couleur": "jaune"}
2 for i in D:
3     print(i)
4 for i in D.values():
5     print(i)
6 for i,j in D.items():
7     print(i,j)
```

Réponse : La première boucle affiche les clés, la deuxième les valeurs, la troisième les deux à la fois.

$\Rightarrow$ La méthode `.items()` permet d'accéder aux paires clé: valeur d'un dictionnaire.

Conclusion

Dans cette séance nous avons pris en main

- les boucles while et for
- les fonctions

et amélioré nos techniques pour manipuler des nombres, des chaînes de caractères, des listes, et des dictionnaires.

Dans la prochaine séance nous parlerons de modules et librairies, et découvrirons plus particulièrement

- le module `numpy`
- le module `random`