Python

Les fondamentaux du langage Python

Olivier SNOECK

Olivier
SNOECK

Les lignes d'instructions

- Une ligne de code = une ligne d'instruction :

```
print("Hello world!")
```

Olivier
SNOECK

Les lignes d'instructions

- Une ligne de code = une ligne d'instruction :

```
print("Hello world!")
```

- Utiliser une variable dans l'impression :

```
variable = 4
print("Hello world!", variable)
```

Olivier
SNOECK

Les lignes d'instructions

- Une ligne de code = une ligne d'instruction :

```
print("Hello world!")
```

- Utiliser une variable dans l'impression :

```
variable = 4
print("Hello world!", variable)
```

- Une ligne de code = deux lignes d'instructions :

```
variable = 4 ; print("Hello world!", variable)
```

Olivier
SNOECK

Les lignes d'instructions

- Une ligne de code = une ligne d'instruction :
```
print("Hello world!")
```
- Utiliser une variable dans l'impression :
```
variable = 4
print("Hello world!", variable)
```
- Une ligne de code = deux lignes d'instructions :
```
variable = 4 ; print("Hello world!", variable)
```
- Une ligne de code = plusieurs lignes d'instructions :
```
variable = 4 ; variable*=5; print("Hello world!", variable)
```

Olivier
SNOECK

Les commentaires

```
1  import numpy as np
2  compteur,notesaisie,sommenote=0,0,0
3  ''' Utilisation d'une liste
4  -> une liste permet de stocker plusieurs types de données dans une seule variable
5  Une liste est une des 4 façons de stocker des données: Tuple, Set et les dictionnaires
6  Les "cases" dans les listes sont indexées. La 1ère a l'index [0], le 2nd a l'index [1] '''
7  note=np.array([])
8
9  print("Bonjour dans ce calculateur de moyenne de vos notes")
10
11 while 1:
12     notesaisie=input("Saisissez votre note: ")
13     if ((notesaisie != "P") and (notesaisie != "p")):
14         if (int(notesaisie) < 0) or (int(notesaisie) > 20):
15             print("Erreur de saisi")
16         else:
17             note=np.append(note,int(notesaisie)); print(note); print(len(note))
18     else:
19         break
20
21 for i in range(len(note)):
22     sommenote=sommenote+note[i]
23
24 if len(note) !=0: #gestion s'il n'y a pas de note saisie
25     print("Votre moyenne est: ",format (sommenote/len(note), '.2f'))
26 else:
27     print("Vous n'avez pas saisi de note")
28
29 #autre solution
30 try:
31     print("Votre moyenne est: ",format (sommenote/len(note), '.2f'))
32 except ZeroDivisionError:
33     print("Vous n'avez pas saisi de note")
```

Olivier
SNOECK

Affichage sur la moniteur

Pour afficher une variable : `print(nomdelavariable)`.
Pour afficher une chaîne de caractères (solution 1) :
`print("le texte à afficher").`
Pour afficher une chaîne de caractères (solution 2) :
`variable = "texte à afficher"; print(variable)`

Il existe plusieurs méthodes (sous programmes) que l'on peut utiliser avec print, par exemple :

```
import os ; print('Bonjour, ' + os.getlogin() + '!
Comment vas-tu aujourd'hui?')
```

Quelques mots clefs : end="" ; sep=""

On peut aussi écrire dans un fichier....

Olivier
SNOECK

Les structures de contrôle : if - elif - else

La structure if est une structure conditionnelle en informatique permettant de prendre en compte des conditions.

Les structures de contrôle : if - elif - else

La structure if est une structure conditionnelle en informatique permettant de prendre en compte des conditions.
Si la condition est **vraie**, alors la suite de la structure comprise après les **:** (et indentée) sera exécutée.

Les structures de contrôle : if - elif - else

La structure if est une structure conditionnelle en informatique permettant de prendre en compte des conditions.
Si la condition est **vraie**, alors la suite de la structure comprise après les **:** (et indentée) sera exécutée.
Si la condition **n'est pas vraie**, alors la structure comprise après les : (et indentée) ne sera pas exécutée. La structure elif (ou else) qui suit la structure if le sera.

Les structures de contrôle : if - elif - else

La structure if est une structure conditionnelle en informatique permettant de prendre en compte des conditions.
Si la condition est **vraie**, alors la suite de la structure comprise après les **:** (et indentée) sera exécutée.
Si la condition **n'est pas vraie**, alors la structure comprise après les : (et indentée) ne sera pas exécutée. La structure elif (ou else) qui suit la structure if le sera.
Les structures elif (et else) sont des structures facultatives qui, lorsqu'elles sont présentes, sont exécutées si et seulement si la condition est fausse.

Olivier
SNOECK

Les structures de contrôle : if - elif - else

```
1  if test1 == True:
2          # action1 à exécuter si le test1 est vrai
3          # ...
4          # action7 à exécuter si le test1 est vrai
5  elif test2 == False:
6          '''
7          action8 à exécuter si le test2 est faux
8          ...
9          action15 à exécuter si le test2 est faux
10         '''
11 #...
12 elif test3:
13         # ...
14 else:
15         # action75 à exécuter si aucun test est vérifié
16         # ...
17         # action77 à exécuter si aucun test est vérifié
```

Note : Les structures elif (et else) sont des structures facultatives..

Olivier
SNOECK

Les structures de contrôle : if - elif - else

! 1 seule ligne avec de multiple "else" sur la même ligne !

```
1 a = 330
2 b = 330
3 print("A") if a > b else print("=") if a == b else print("B")
```

afficher "A" si a $>$ b sinon afficher "=" si a$==$b sinon afficher "B"

Olivier
SNOECK

Les structures de contrôle : if - elif - else

! 1 seule ligne avec de multiple "else" sur la même ligne !

```
1 a = 330
2 b = 330
3 print("A") if a > b else print("=") if a == b else print("B")
```

afficher "A" si a > b sinon afficher "=" si a==b sinon afficher "B"

Un test "if" avec 1 condition logique **and**

```
1 a = 200
2 b = 33
3 c = 500
4 if a > b and c > a:
5   print("Les 2 conditions sont vraies")
```

Olivier
SNOECK

Les structures de contrôle : if - elif - else

! 1 seule ligne avec de multiple "else" sur la même ligne !

```
1 a = 330
2 b = 330
3 print("A") if a > b else print("=") if a == b else print("B")
```

afficher "A" si a > b sinon afficher "=" si a==b sinon afficher "B"

Un test "if" avec 1 condition logique **and**

```
1 a = 200
2 b = 33
3 c = 500
4 if a > b and c > a:
5   print("Les 2 conditions sont vraies")
```

Un test "if" avec 1 condition logique **or**

```
1 a = 200
2 b = 33
3 c = 500
4 if a > b or a > c:
5   print("Au moins une des 2 conditions est vraie")
```

Exercice if elif else

Écrire un programme qui prend en entrée une température t et qui renvoie l'état de l'eau à cette température c'est à dire "SOLIDE", "LIQUIDE" ou "GAZEUX".
On prendra comme conditions les suivantes :

- Si la température est strictement négative alors l'eau est à l'état solide.
- Si la température est entre 0 et 100 (compris) l'eau est à l'état liquide.
- Si la température est strictement supérieure à 100.

Entrée : Une température t.
Sortie : L'état de l'eau à cette température parmi les trois possibilités : "SOLIDE", "LIQUIDE" ou "GAZEUX".
On utilisera la fonction **return** pour renvoyer les résultats.
Le programme doit boucler sans arrêt (sauf avec ctrl+c).

Olivier
SNOECK

L'instruction while

L'instruction **while** sert à recommencer une série d'instructions **tant que** la condition est vraie.

```
1 a = 10
2 while a > 0:
3         print("a = ", a)
4         a-=1
5 print("On sort de la boucle car a = 0")
```

et le résultat est :

a = 10
a = 9
a = 8
a = 7
a = 6
a = 5
a = 4
a = 3
a = 2
a = 1
On sort de la boucle car a = 0

Olivier
SNOECK

L'instruction while

L'instruction **while** sert à recommencer une série d'instructions **tant que** la condition est vraie.

```
1 a = 10
2 while a > 0:
3         print("a = ", a)
4         a-=1
5 print("On sort de la boucle car a = 0")
```

On peut facilement créer une boucle infinie avec **while True :**

```
1 a = 10
2 while True:
3     if a > 0:
4         print("a = ", a)
5         a-=1
6     else:
7         break
8 print("On sort de la boucle car a = 0")
```

L'instruction **break** permet de terminer l'itération immédiatement, quelque soit le nombre d'itérations effectuées ou restant à faire.

Exercice while

1er exercice :

L'utilisateur donne un entier positif et le programme annonce combien de fois de suite cet entier est divisible par 2.

- avec une boucle **if else**
- avec une boucle **while**

2nd exercice :

En utilisant une boucle **while**, entrez un prix HT (entrez « 0 » pour terminer) et affichez sa valeur TTC.

Les listes

- Python connaît différents types de données combinés, utilisés pour regrouper plusieurs valeurs. Le plus souple est la liste, qui peut être écrit comme une suite, placée entre crochets, de valeurs (éléments) séparées par des virgules. Les éléments d'une liste ne sont pas obligatoirement tous du même type, bien qu'à l'usage ce soit souvent le cas.
  - Exemple : `ages = [22, 25, 21, 26, 25, 52]`

Les listes

- Python connaît différents types de données combinés, utilisés pour regrouper plusieurs valeurs. Le plus souple est la liste, qui peut être écrit comme une suite, placée entre crochets, de valeurs (éléments) séparées par des virgules. Les éléments d'une liste ne sont pas obligatoirement tous du même type, bien qu'à l'usage ce soit souvent le cas.
  - Exemple : `ages = [22, 25, 21, 26, 25, 52]`

Les listes

- Python connaît différents types de données combinés, utilisés pour regrouper plusieurs valeurs. Le plus souple est la liste, qui peut être écrit comme une suite, placée entre crochets, de valeurs (éléments) séparées par des virgules. Les éléments d'une liste ne sont pas obligatoirement tous du même type, bien qu'à l'usage ce soit souvent le cas.
  - Exemple : `ages = [22, 25, 21, 26, 25, 52]`
- Les listes ont un indice qui commence par l'indice 0 :
  - Exemple : `ages [0] = 22`

Les listes

- Python connaît différents types de données combinés, utilisés pour regrouper plusieurs valeurs. Le plus souple est la liste, qui peut être écrit comme une suite, placée entre crochets, de valeurs (éléments) séparées par des virgules. Les éléments d'une liste ne sont pas obligatoirement tous du même type, bien qu'à l'usage ce soit souvent le cas.
  - Exemple : `ages = [22, 25, 21, 26, 25, 52]`
- Les listes ont un indice qui commence par l'indice 0 :
  - Exemple : `ages [0] = 22`

Les listes

- Python connaît différents types de données combinés, utilisés pour regrouper plusieurs valeurs. Le plus souple est la liste, qui peut être écrit comme une suite, placée entre crochets, de valeurs (éléments) séparées par des virgules. Les éléments d'une liste ne sont pas obligatoirement tous du même type, bien qu'à l'usage ce soit souvent le cas.
  - Exemple : `ages = [22, 25, 21, 26, 25, 52]`
- Les listes ont un indice qui commence par l'indice 0 :
  - Exemple : `ages [0] = 22`
- On peut afficher la liste de 2 manières :
  - Exemple : `print(ages)` ou `print(ages[:])`

Les listes

- Python connaît différents types de données combinés, utilisés pour regrouper plusieurs valeurs. Le plus souple est la liste, qui peut être écrit comme une suite, placée entre crochets, de valeurs (éléments) séparées par des virgules. Les éléments d'une liste ne sont pas obligatoirement tous du même type, bien qu'à l'usage ce soit souvent le cas.
  - Exemple : `ages = [22, 25, 21, 26, 25, 52]`
- Les listes ont un indice qui commence par l'indice 0 :
  - Exemple : `ages [0] = 22`
- On peut afficher la liste de 2 manières :
  - Exemple : `print(ages)` ou `print(ages[:])`

Les listes

- Python connaît différents types de données combinés, utilisés pour regrouper plusieurs valeurs. Le plus souple est la liste, qui peut être écrit comme une suite, placée entre crochets, de valeurs (éléments) séparées par des virgules. Les éléments d'une liste ne sont pas obligatoirement tous du même type, bien qu'à l'usage ce soit souvent le cas.
  - Exemple : `ages = [22, 25, 21, 26, 25, 52]`
- Les listes ont un indice qui commence par l'indice 0 :
  - Exemple : `ages [0] = 22`
- On peut afficher la liste de 2 manières :
  - Exemple : `print(ages)` ou `print(ages[:])`
- On peut "trancher" la liste :
  - Exemple : `print(ages[-3:])` et on obtient `[26, 25, 52]`

Olivier
SNOECK

Les listes

```
1  ages = [22, 25, 21, 26, 25, 52] #ages est une liste
2  print("ages = ",ages) # affichage de la liste ages
3  ages [0] = 52
4  print("ages[0] = ",ages[0]) # affichage de la 1ère valeur de la liste ages
5  print("ages = ",ages) # affichage de la liste ages
6  print("len(ages) = ",len(ages))
7
8  ''' La tranche permet de récupérer une partie d'une liste en utilisant un indiçage
   ↪  construit sur le modèle [m:n+1] pour récupérer tous les éléments, du émième au énième
   ↪  (de l'élément m inclus à l'élément n+1 exclu)'''
9  print(ages[0:2])
10
11 ''' Les indices négatifs reviennent à compter à partir de la fin.
12 Leur principal avantage est que vous pouvez accéder au dernier élément d'une liste à
   ↪  l'aide de l'indice -1'''
13 print(ages[-3:]) # on affiche les 3 dernières valeurs (indiçage négatif (commence par
   ↪  -1))
14
15 # Quelques opérations sur les listes:
16 ages = ages + [25 , 12] # j'insère une nouvelle liste
17 print("ages = ",ages)
18
19 ages.append(77) #j'insère une nouvelle valeur
20 print("ages = ",ages)
21 print("len(ages) = ",len(ages))
22
23 listemixte = [52 , "professeur" , 27, "Billel"]
24 print("listemixte = " , listemixte)
25
26 ''' L'instruction range() est une fonction spéciale en Python qui génère des nombres
   ↪  entiers compris dans un intervalle. Lorsqu'elle est utilisée en combinaison avec la
   ↪  fonction list(), on obtient une liste d'entiers.'''
27 print("list(range(10)) = ",list(range(10)))
```

Olivier
SNOECK

Les listes

Et le résultat est :
ages = [22, 25, 21, 26, 25, 52]
ages[0] = 52
ages = [52, 25, 21, 26, 25, 52]
len(ages) = 6
[52, 25]
[26, 25, 52]
ages = [52, 25, 21, 26, 25, 52, 25, 12]
ages = [52, 25, 21, 26, 25, 52, 25, 12, 77]
len(ages) = 9
listemixte = [52, 'professeur', 27, 'Billel']
list(range(10)) = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Les listes

Voici quelques méthodes usuelles :

Méthodes	Descriptions
list(s)	Retourne le contenu de la liste s
s.append(x)	Ajoute l'élément x à la fin de la liste s
s.extend(x)	Ajoute la liste x à la liste s
s.count(x)	Compte le nombre d'occurrence de x dans s
s.pop(i)	Retourne la valeur de l'élément i et l'efface de s
s.remove(x)	Efface x dans s
...	....

Olivier
SNOECK

L'instruction for

Le mot clef **in** vérifie l'appartenance d'un élément dans une séquence (une liste par exemple).
Le mot clef **for**, associé au mot clef **in**, permet de réaliser une itération sur l'ensemble des éléments d'une séquence

```
1 for a in (2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73,
↪  79, 83, 89, 97):
2     print (a, "est premier")
```

et le résultat est :

2 est premier
3 est premier
5 est premier
7 est premier
11 est premier
13 est premier
17 est premier
19 est premier
23 est premier
29 est premier
31 est premier
.....

Exercice sur les listes

1er exercice :

- Créer un programme qui permet de sauvegarder les noms de 5 stagiaires saisi par l'utilisateur.
- Modifier ce programme pour qu'il range les noms par ordre alphabétique et qui les affiche.
- Modifier ce programme que le éléments de la listes aient la 1ère lettre en majuscule et les autres lettres en minuscule.

2nd exercice :

- Créer un programme qui demande les notes à un étudiant, et ensuite, affiche l'ensemble de ses notes ainsi que sa moyenne.
- Lorsque l'étudiant appuie sur une lettre (de votre choix), la moyenne, l'écart-type sont affichés ainsi qu'une courbe représentant la dispersion des notes. Pour cela, utilisez la librairie numpy.

Les Tuples

- A partir des types de base (int, float, etc.), il est possible d'en élaborer de nouveaux. On les appelle des types construits.
- Un exemple de type construit est le tuple. Il permet de créer une collection ordonnée de plusieurs éléments..
- Les tuples ressemblent aux listes, mais on ne peut pas les modifier une fois qu'ils ont été créés.
- On dit qu'un tuple n'est pas **mutable** : ils sont non modifiables.
- On le définit avec des parenthèses.
- Exemple : `t = 12345, 54321, 'hello!'`

Olivier
SNOECK

Les Tuples

```
1  t = 12345, 54321, 'hello!' #t est un tuple
2  u = t, (1, 2, 3, 4, 5) #u est un tuple
3  print("t[0] = ",t[0]) # affichage de la 1ère valeur du tuple t
4
5  print("t = ",t) # affichage du tuple t
6  print("len(t) = ",len(t))
7
8  print("u = ", u) # affichage du tuple u
9  print("len(u) = ",len(u))
10
11 #t[0]=5 #va provoquer une erreur d'assignation
12 v = ([1, 2, 3], [3, 2, 1]) # création d'un tuple à 2 dimensions
13 print("v = ", v)
14 print("len(v) = ",len(v))
15
```

Et le résultat est :

```
t[0] = 12345
t = (12345, 54321, 'hello!')
len(t) = 3
u = ((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
len(u) = 2
v = ([1, 2, 3], [3, 2, 1])
len(v) = 2
```

Olivier
SNOECK

Les Dictionnaires

Un dictionnaire est une collection non ordonnée de relations entre clés et valeurs. Une clé d'un dictionnaire doit-être hashable. Il ne s'agit pas d'objets séquentiels comme les listes ou chaînes de caractères, mais plutôt d'objets dits de correspondance (mapping objects en anglais) ou tableaux associatifs. En effet, on accède aux valeurs d'un dictionnaire par des clés. Ceci semble un peu confus ? Regardez l'exemple suivant :

```
1 ani = {}  # Création du dictionnaire vide
2 ani["nom"] = "girafe" # ici, on remplit le dictionnaire avec la clef nom
3 ani["taille"] = 5.0 # ici, on remplit le dictionnaire avec la clef taille
4 ani["poids"] = 1100 # ici, on remplit le dictionnaire avec la clef poids
5
6 print(ani) # affichage du dictionnaire dans l'ordre de son remplissage
```

ce qui nous affiche dans la console :
'nom' : 'girafe', 'taille' : 5.0, 'poids' : 1100

Les Dictionnaires

Voici quelques méthodes usuelles :

Méthodes	Descriptions
.keys()	Renvoie la (ou les) clef (s)
.values()	Renvoie la (ou les) valeur (s)
.items()	Créé un nouveau dictionnaire
.get()	= .key() sans erreurs, même si elle n'existe pas
sorted()	Tri alphabétique le dictionnaire par ses clefs
sorted(dictio,key=dictio.get)	Tri par la clef
...	....

Les dictionnaires feront parties d'un autre cours...

Olivier
SNOECK

Les fonctions

Une fonction est une suite d'instructions (des lignes de programme) que l'on appellera par le nom de la fonction, comme un sous-programme.

```
1 def carre(x):         # Ici, lors de l'appel de la fonction carre, je passe l'argument x
2         return x**2         # return signifie que le résultat de carre(x)=x**2
3
4 def hello():         #Ici, je n'ai pas besoin de passer d'argument
5         print("Bonjour à tous")
6
7 hello()
8 print ("Le carré de 4 est: ", carre(4))
```

ce qui nous affiche dans la console :

```
Bonjour à tous
Le carré de 4 est: 16
```

Le mot-clef pour la définition d'une fonction est **def**.
N'oubliez pas l'indentation après les **:**

Exercice

1. Écrire une fonction qui, recevant une taille n en paramètre, affiche un tapis de n+1 lignes sur n+1 colonnes, traversé par une diagonale.
   Le tableau est remplit du caractère "cœur".
   La diagonale est vide.
2. Modifier la fonction pour qu'elle prenne en argument la position de la diagonale (diagonale vide).
3. Réaliser un programme qui permette d'afficher le tapis avec la diagonale qui se déplace dans la console.