TP04 BasesPython

TP 04 : Les bases en Python	Thème 4 : Langages et Programmation
	COURS

Lien Capytale

Ce document recense les informations essentielles (donc à connaître absolument) pour démarrer en Python.

Votre objectif est de suivre ce notebook et de faire les exercices du notebook d'exercices qui sont indiqués au fur et à mesure afin de mettre en pratique.

1. I. Variables et affectations en Python

Dans le langage Python, le symbole `=` correspondant à l'affectation (écrite ← en pseudo-code).

Pour affecter la valeur 2 à une variable `a` on écrit simplement `a = 2`.

Ainsi, l'algorithme

Algo

```
a ← 3
b ← 2 * a
```

s'écrit en Python de la façon suivante :

Script Python

```
a = 3
b = 2 * a
```

✏ Faire les exercices 1, 2, 3.

2. II. Instructions conditionnelles : `if`, `elif` et `else`

En Python, les instructions conditionnelles se codent en utilisant les instructions `if`, `elif` (= contraction de *else if*), `else` qui sont les traductions respectives de *si*, *sinon si* et *sinon*.

Script Python

```
if condition1:
    bloc_instructions_1
```

```
elif condition2:
    bloc_instructions_2
else:
    bloc_instructions_3
```

Remarques : - ne pas oublier les deux points à la fin des lignes avec `if`, `elif` et `else` qui permettent d'ouvrir le bloc d'instructions à effectuer dans chaque cas ; - les instructions à effectuer sont indentées d'une tabulation par rapport aux `if`, `elif` et `else` : c'est la syntaxe Python, elle doit absolument être respectée ; - les instructions `elif` et `else` sont optionnelles, c'est-à-dire que l'on peut avoir seulement un `if` comme ceci

Algo

````
```python
if condition:
    bloc_instructions

```

ou un `if` suivi d'un `else` comme cela

```python
if condition:
    bloc_instructions_1
else:
    bloc_instructions_2

```
````

- au contraire du pseudo-code, on n'écrit pas en Python de `fin si` car celui-ci est matérialisé par la fin des indentations.

Exemple 1 (rôle des indentations) : Dans le programme suivant, le dernier message s'affiche à chaque fois, en revanche les deux messages précédents ne s'affichent que si on entre dans le bloc `if`, autrement dit si la condition `a >= 10` est vraie.

Script Python

```
a = 15
if a >= 10: # condition vraie
    print("Vous avez la moyenne")
    print("Message dans le bloc d'instructions du if")
print("Message en dehors du bloc d'instruction du if") # en dehors du bloc d'instructions du if
```

Algo

```
Vous avez la moyenne
Message dans le bloc d'instructions du if
Message en dehors du bloc d'instruction du if
```

Script Python

```
a = 8
if a >= 10: # condition fausse
    print("Vous avez la moyenne")
```

```
    print("Message dans le bloc d'instructions du if")
print("Message en dehors du bloc d'instruction du if") # en dehors du bloc d'instructions du if
```

Algo

```
Message en dehors du bloc d'instruction du if
```

Exemple 2 : L'algorithme

Algo

```
si scoreA > scoreB alors
    vainqueur ← "équipe A"
sinon si scoreA < scoreB alors
    vainqueur ← "équipe B"
sinon
    vainqueur ← "Match nul"
```

se traduit en Python de la manière suivante :

Script Python

```
if scoreA > scoreB:
    vainqueur = "équipe A"
elif scoreA < scoreB:
    vainqueur = "équipe B"
else:
    vainqueur = "Match nul"
```

> ✏ Faire les exercices 4, 5

3. III. Boucles bornées : boucles `for`

En Python, les boucles Pour se codent en utilisant l'instruction `for` :

Script Python

```
for element in sequence:
    bloc_instructions
```

Remarques : - `element` est une variable créée par le `for`, ce n'est pas à vous de l'instancier. Elle prend successicement chacune des valeurs figurant dans la `sequence` parcourue ; - ne pas oublier les deux points à la fin de la ligne avec `for` qui permettent d'ouvrir le bloc d'instructions à exécuter (à répéter) ; - les instructions à effectuer sont indentées d'une tabulation par rapport au `for` ; - au contraire du pseudo-code, on n'écrit pas en Python de `fin pour` car celui-ci est matérialisé la fin des indentations.

3.1. La fonction `range()` pour créer des séquences de nombres

La fonction `range()` permet de créer des séquences de nombres entiers :

- `range(n)` crée une séquence des `n` entiers, de 0 inclus à `n` **exclu**, c'est-à-dire : 0, 1, 2, ..., n-1 (⚠ le premier est 0, donc le n-ième est n-1).
- `range(n, m)` crée la séquence d'entiers de `n` inclus à `m` **exclu**, c'est-à-dire : n, n+1, ..., m-1.
- `range(n, m, p)` crée la séquence de nombres de `n` inclus à `m` **exclu** avec un pas égal à `p`, c'est-à-dire : n, n+p, n+2p, ...

Exemples :

- `range(5)` : crée la séquence de nombres de 0 inclus à 5 **exclu** : c'est-à-dire les nombres 0, 1, 2, 3, 4.
- `range(2, 8)` : crée la séquence de nombres de 2 inclus à 8 **exclu** : c'est-à-dire les nombres 2, 3, 4, 5, 6, 7.
- `range(2, 11, 3)` : créee la séquence de nombres de 2 inclus à 11 **exclu**, **par pas de 3** : c'est-à-dire les nombres 2, 5, 8.

On peut visualiser cette séquence en affichant leurs valeurs successives :

Script Python

```
for element in range(5):
    print(element)
```

Algo

```
0
1
2
3
4
```

ou en utilisant la fonction `list()` :

Script Python

```
list(range(5))
```

Algo

```
[0, 1, 2, 3, 4]
```

> N'hésitez pas à modifier les deux cellulles précédentes pour voir d'autres séquences.

Ainsi, pour répéter un bloc d'instructions 36 fois, il suffit d'écrire :

Script Python

```
for i in range(36):
    bloc_instructions
```

En effet, la variable `i` créée par la boucle va prendre successivement les valeurs : 0, 1, 2, 3, ..., 35. Elle prend donc 36 valeurs et le bloc `bloc_instructions` sera donc répété 36 fois.

On aurait aussi pu écrire cela :

Script Python

```
for i in range(0, 36):
    bloc_instructions
```

ou encore :

```python for i in range(1,37): bloc_instructions
```

Algo

```
ou encore :

```python
for valeur in range(36):
    bloc_instructions
```

car dans chaque cas, la variable créée (`i` ou `valeur`) parcoure 36 valeurs différentes.

> ✏ Faire les exercices 6, 7, 8, 9.

3.2. Parcourir des chaînes de caractères

La boucle `for` permet de parcourir tous les éléments d'une *séquence*. En utilisant `range()` on a vu que l'on peut parcourir une séquence de nombres.

Il se trouve que les chaînes de caractères sont également des *séquences* ... de caractères. On peut donc aussi les parcourir très simplement avec une boucle `for`, de deux manières différentes.

Par ses caractères

On peut parcourir une chaîne directement par ses caractères :

Script Python

```
chaine = "Bonjour les élèves !"
for caractere in chaine :
    print(caractere)
```

Algo

```
B
o
n
j
o
u
r

l
e
```
```

```
s

é
l
è
v
e
s

!
```

La variable `caractere` créée par le `for` prend successivement les valeurs de la séquence `"Bonjour les élèves !"`, c'est-à-dire qu'elle prend la valeur `'B'` puis la valeur `'o'`, puis la valeur `'n'`, etc. La boucle s'arrête lorsque toute la séquence a été parcourue.

Par l'indice de ses caractères

On peut aussi utiliser la fonction `range()` pour parcourir les caractères par leurs indices.

Script Python

```
chaine = "Bonjour les élèves !"
for indice in range(len(chaine)) :
    print(chaine[indice])
```

Algo

```
B
o
n
j
o
u
r

l
e
s

é
l
è
v
e
s

!
```

L'instruction `len(chaine)` renvoie la longueur de la chaîne de caractères, soit 20 dans notre exemple. Donc `range(len(chaine))` est évaluée à `range(20)` qui crée la séquence d'entiers 0, 1, 2, ..., 19 qui vont être affectés successivement à la variable `indice`. On affiche ensuite, à chaque tour de boucle, le caractère en position `indice` dans `chaine` c'est-à-dire `chaine[0]`, puis `chaine[1]`, ... et enfin `chaine[19]`.

> ✏ Faire l'exercice 10.

4. IV. Boucles non bornées : boucles `while`

En Python, les boucles "Tant que" se codent en utilisant l'instruction `while` :

Script Python

```
while condition:
    bloc_instructions
```

Remarques : - `condition` est une variable booléenne qui est soit vraie (`True`) soit fausse (`False`) ; - Tant que `condition` vaut `True` les instructions du bloc sont répétées ; - On passe à la suite du programme dès que `condition` vaut `False`. Cela signifie que si `condition` reste vraie tout le temps, la boucle `while` boucle à l'infini, ce qui pourra causer des problèmes plus ou moins importants ; - ne pas oublier les deux points à la fin de la ligne avec `while` qui permettent d'ouvrir le bloc d'instructions à exécuter (à répéter) ; - les instructions à effectuer sont indentées d'une tabulation par rapport au `while` ; - au contraire du pseudo-code, on n'écrit pas en Python de `fin tant que` car celui-ci est matérialisé la fin des indentations.

Exemple :

Script Python

```
a = 3
while a < 14:
    a = a + 2
    print(a) # pour voir l'évolution de la variable a
```

Algo

```
5
7
9
11
13
15
```

✏ Faire les exercices 11, 12, 13.

5. V. Les fonctions

5.1. Définition et syntaxe en Python

Dans un langage de programmation, on utilise ce qu'on appelle des **fonctions**. Une fonction est un ensemble d'instructions qui peut recevoir des **arguments** et qui peut renvoyer un résultat qui est souvent le contenu d'une ou plusieurs variables.

En Python, on définit une fonction en utilisant l'instruction `def` (de l'anglais *define*, qui veut dire "définir") :

Script Python

```
def nom_de_la_fonction(parametre1, parametre2, ..., parametreN):
    corps_de_la_fonction
```

- l'instruction `def` est suivie du **nom** de la fonction ;
- les **paramètres** de la fonction sont ensuite écrits entre parenthèses et séparés par des virgules ;
- il existe des fonctions sans paramètre, les parenthèses sont néanmoins obligatoires et restent vides ;
- il ne faut pas oublier les **deux points** après les parenthèses de la première ligne ;
- le corps de la fonction est un bloc d'instructions qui contient toutes les lignes qui doivent être exécutées lorsque la fonction est appelée. Le corps de la fonction doit nécessairement être **indenté**, c'est-à-dire qu'il doit être décalé d'une tabulation par rapport à l'instruction `def`.

Très souvent, le corps de la fonction se terminera par l'instruction `return` suivie de la ou des valeurs que la fonction doit renvoyer. Si la fonction doit renvoyer plusieurs valeurs, celles-ci sont à séparer par des virgules.

Ainsi, le schéma général d'une fonction `Python` est :

Script Python

```
def nom_de_la_fonction(parametre1, parametre2, ..., parametreN):
    instructions                          # sur plusieurs lignes éventuellement
    return valeur1, valeur2, valeur3, etc.  # souvent une fonction ne renvoie qu'une valeur
```

Exemple : Voici une fonction

Script Python

```
def addition(a, b):
    s = a + b
    return s
```

que l'on peut aussi écrire plus simplement :

Script Python

```
def addition(a, b):
    return a + b
```

> ? Identifiez dans ces deux fonctions écrites différemment : leur nom, leur(s) paramètre(s), leur corps, leur(s) valeur(s) renvoyée(s).

Remarque : il existe des fonctions qui ne renvoient aucune valeur, l'instruction `return` n'est donc pas utilisée dans le corps de ces fonctions.

5.2. Appel à une fonction

⚠ Lorsque l'on exécute le code qui définit une fonction, aucune valeur n'est renvoyée ! Cela a seulement pour objectif d'enregistrer la fonction en mémoire.

Script Python

```
# à exécuter : il ne se passera rien (visuellement)

def addition(a, b):
    return a + b
```

Pour utiliser une fonction il faut l'**appeler**. On appelle une fonction par son nom en donnant des arguments (des valeurs) à ses paramètres. Dans ce cas, la fonction va renvoyer la ou les valeurs attendues.

Script Python

```
# appel à la fonction : qui renvoie alors ce qu'il faut !

addition(2, 5)
```

Algo

```
7
```

Remarque : au premier `return` rencontré l'exécution de la fonction est stoppée : si on veut renvoyer plusieurs valeurs on ne peut pas utiliser plusieurs `return` ; il faut séparer les valeurs à renvoyer par des virgules ;

Par exemple, si on veut une fonction qui renvoie la somme **et** le produit de deux nombres, on ne peut pas écrire

Script Python

```
def somme_et_produit(a, b):
    return a + b  # fonction stoppée après cette ligne (premier return rencontré)
    return a * b  # ne sera pas exécuté !
```

mais il faudrait écrire cela :

Script Python

```
def somme_et_produit(a, b):
    return a + b, a * b  # les deux valeurs seront bien renvoyées
```

On peut facilement vérifier :

Script Python

```
def somme_et_produit(a, b):
    return a + b  # fonction stoppée après cette ligne (premier return rencontré)
    return a * b  # ne sera pas exécuté !
```

Script Python

```
somme_et_produit(2, 5)
```

Algo

```
7
```

Script Python

```python
def somme_et_produit(a, b):
    return a + b, a * b  # les deux valeurs seront bien renvoyées
```

Script Python

```python
somme_et_produit(2, 5)
```

Algo

```
(7, 10)
```

> ✏ Faire les exercices 14, 15, 16.

6. VI. Les entrées en Python

Pour que l'utilisateur puisse saisir des valeurs au clavier, on utilise la fonction `input`. Elle déclenche l'ouverture d'une boîte de dialogue dans laquelle on peut saisir la valeur souhaitée.

Script Python

```python
nom_utilisateur = input("Veuillez saisir votre nom : ")
age_utilisateur = input("Veuillez saisir votre âge : ")
print(f"Vous vous appelez {nom_utilisateur} et vous avez {age_utilisateur} ans.")
print(type(age_utilisateur)) # OBSERVEZ BIEN LE TYPE !!
```

Algo

```
Veuillez saisir votre nom :  Test
Veuillez saisir votre âge :  18


Vous vous appelez Test et vous avez 18 ans.
<class 'str'>
```

> **ATTENTION :** Par défaut, en Python, les variables saisies au clavier et récupérées par la fonction `input` sont des chaînes de caractères.

Cela peut créer des erreurs :

Script Python

```python
age_utilisateur = input("Veuillez saisir votre âge : ")
age_utilisateur = age_utilisateur + 2
```

Algo

```
Veuillez saisir votre âge : 65



---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-15-8ae97b43fbf4> in <module>
      1 age_utilisateur = input("Veuillez saisir votre âge : ")
----> 2 age_utilisateur = age_utilisateur + 2


TypeError: can only concatenate str (not "int") to str
```

L'erreur se situe ligne 2 comme l'indique la flèche verte. Le message d'erreur `TypeError: can only concatenate str (not "int") to str` indique que Python ne peut concaténer une chaîne de caractères qu'avec une autre chaîne de caractères (mais pas avec un entier).

En effet, la variable `age_utilisateur` étant de type `str`, le symbole `+` qui suit est le symbole de concaténation, qui ne peut pas s'appliquer à l'entier 2 situé après car celui-ci est de type `int`.

RÈGLE D'USAGE IMPORTANTE : pour éviter ces problèmes, il suffit de préciser le type de variable que l'utilisateur doit saisir avant la fonction `input` comme indiqué ci-dessous :

- pour saisir un entier : `nom_variable = int(input("message à afficher"))`
- pour saisir un flottant : `nom_variable = float(input("message à afficher"))`
- pour saisir un booléen : `nom_variable = bool(input("message à afficher"))`
- pour saisir un entier long : `nom_variable = long(input("message à afficher"))`

Ainsi, pour régler le problème précédent, il suffit d'écrire :

Script Python

```
age_utilisateur = int(input("Veuillez saisir votre âge :"))
print(f"Dans deux ans, vous aurez {age_utilisateur + 2} ans")
```

Algo

```
Veuillez saisir votre âge : 15


Dans deux ans, vous aurez 17 ans
```

> Nous n'utiliserons les instructions d'entrée en Python que dans les cas où cela s'avère nécessaire, par exemple si une interaction est nécessaire avec l'utilisateur. Dans les autres cas, on privilégiera les appels de fonctions en modifiant leurs arguments.