Sciences du Numérique et Technologie
Notions de Python

I. Les variables

Définition

Une **variable** est un emplacement mémoire. Dans cet emplacement est stockée la **valeur** de la variable.

En python, on affecte une valeur à une variable avec le symbole `=` :

```
>>> a = 5
>>> a
 5
```

II. Les types de données

II.1. Les entiers

Définition

Les nombres entiers sont de type `int` (integer).

```
>>> type(5)
<class 'int'>
>>> a = 7
>>> type(a)
<class 'int'>
```

II.2. Les flottants

Définition

Les nombres qui ne sont pas entiers sont des flottants. Il sont de type `float` .

```
>>> type(5.2)
<class 'float'>
>>> a = 3.0
>>> type(a)
<class 'float'>
```

II.3. Les tableaux

Définition

Un tableau est une collection d'éléments, souvent de même type. En Python, un tableau est composé de plusieurs éléments séparés par des virgules, le tout étant entre crochets. Un tableau est de type `list`.

Propriété

Un tableau est ordonné (les éléments qui la compose sont dans un certain ordre) et mutable (on peut changer la valeur de ses éléments). On peut ainsi accéder à un élément en connaissant son emplacement, appelé **indice**. On dit que l'élément du tableau est **indexé** par son indice.
Les indices commencent toujours à 0. Ainsi le premier élément d'un tableau est l'élément d'indice 0.

```
>>> type([5, 3, 2])
<class 'list'>
>>> mon_tableau = [5, 3, 2]
>>> mon_tableau[1]
3
>>> mon_tableau[0]
5
>>> mon_tableau[1] = 8
>>> mon_tableau
[5, 8, 2]
>>> len(mon_tableau)
3
```

II.4. Les chaînes de caractères

Définition

Une chaîne de caractères est une collection de caractères entourés de guillemets (`' '` ou `" "`). Elle est de type `str` (string).

Propriété

Une chaîne de caractères est ordonnée (les éléments qui la compose sont dans un certain ordre) et immuable (on ne peut pas changer la valeur de ses éléments). On peut ainsi accéder à un élément en connaissant son emplacement, appelé **indice**. On dit que l'élément de la chaîne de caractères est **indexé** par son indice.
Les indices commencent toujours à 0. Ainsi le premier élément d'une chaîne de caractères est l'élément d'indice 0.

```
>>> type('Bonjour')
<class 'str'>
>>> ma_chaine = "Bonjour"
>>> ma_chaine[2]
'n'
>>> ma_chaine[0]
'B'
>>> ma_chaine[3] = 'p'
Traceback (most recent call last):
...
TypeError: 'str' object does not support item assignment
>>> len(ma_chaine)
7
```

III. Opérations élémentaires

III.1. addition

Définition

En Python, on utilise l'opérateur binaire `+` pour additionner.

```
>>>15 + 3
 18
```

III.2. soustraction

Définition

En Python, on utilise l'opérateur binaire `-` pour soustraire.

```
>>>15 - 3
 12
```

III.3. multiplication

Définition

En Python, on utilise l'opérateur binaire `*` pour multiplier.

```
>>>15 * 3
 45
```

III.4. diviser

Définition

En Python, on utilise l'opérateur binaire `/` pour effectuer la division décimale.

Définition

En Python, on utilise l'opérateur binaire `//` pour effectuer la division euclidienne.

Définition

En Python, on utilise l'opérateur binaire `%` pour récupérer le reste de la division euclidienne.

```
>>>15 / 2
 7.5
>>> 15 / 3
 5.0
>>> 15 // 3
 5
>>> 15 // 2
 7
>>> 15 % 2
 1
```

Remarque

La division décimale renvoie toujours un flottant alors que la division euclidienne renvoie toujours un entier.

III.5. puissance

Définition

En Python, on utilise l'opérateur binaire `**` pour effectuer une puissance.

```
>>>2 ** 5
32
```

IV. Les fonctions

Définition

Une **fonction** est une variable dont la valeur est une suite d'instructions, ainsi que d'éventuels arguments. Pour implémenter une fonction, on utilise l'instruction `def` suivi du nom de la fonction, de parenthèses dans lesquels on indique les éventuels arguments séparés par des virgules et enfin de `:`.

```
def quotient(a, b):
    q = a // b
    return q
```

On peut alors appeler la fonction en la nommant et indiquant les paramètres.

```
>>> quotient(7, 2):
3
```

V. Les tests conditionnels

V.1. Les conditions

Définition

Une **condition** est une instruction qui renvoie un booléen.

Les comparaisons

Les opérateurs de comparaison renvoient un booléen.

Symbole	Signification
==	est égal à
!=	n'est pas égal à (ou est différent de)
<	est inférieur à
>	est supérieur à
<=	est inférieur ou égal à
>=	est supérieur ou égal à

V.2. L'appartenance

Propriété

L'opérateur d'appartenance `in` renvoie un booléen.

```
>>> chaine = "abcde"
>>> "b" in chaine
True
>>> f in chaine
False
>>> f not in chaine
True
```

V.3. Instructions conditionnelles

Définition

Une **instruction conditionnelle** est une instruction qui s'effectue en fonction de l'évaluation d'une condition booléenne.

Algorithmique	En python
Si *condition* alors *instructions*	`if condition:` `    instructions`

Si *condition*
alors *instructions1*
sinon *instructions2*
	`if condition:`
	`    instructions1`
	`else:`
	`    instructions2`

Si *condition1*
alors *instructions*
et si *condition2*
alors *instrucitons2*
	`if condition1:`
	`    instructions1`
	`elif conditions2:`
	`    instructions2`

Si *condition1*
alors *instructions*
et si *condition2*
alors *instrucitons2*
⋮
sinon *instructions3*
	`if condition1:`
	`    instructions1`
	`elif conditions2:`
	`    instructions2`
	`...`
	`else:`
	`    instructions3`

Exemple

```
nb = ...
if nb % 2 == 0:
    resultat = nb // 2
else:
    resultat = 3 * nb + 1
```

Par exemple :

- Si la variable `nb` a pour valeur 15, alors la variable `resultat` aura pour valeur 46.
- Si la variable `nb` a pour valeur 16, alors la variable `resultat` aura pour valeur 8.

```
nb = ...
if nb % 2 == 0:
    resultat = nb // 2
elif nb % 3 == 0:
    resultat = nb + 2
else:
    resultat = 3 * nb + 1
```

Par exemple :

- Si la variable `nb` a pour valeur 15, alors la variable `resultat` aura pour valeur 17.
- Si la variable `nb` a pour valeur 16, alors la variable `resultat` aura pour valeur 8.
- Si la variable `nb` a pour valeur 17, alors la variable `resultat` aura pour valeur 52.

VI. Les boucles

VI.1. Les boucles non bornées

Lorsqu'on veut répéter une série d'instructions mais qu'on ne sait pas quand/si cela peut s'arrêter, on utilise une boucle non bornée, ou boucle Tant que, ou boucle while.

Définition

Une boucle Tant que est constituée d'une condition et d'une séquence d'instructions.
Lors de l'exécution de la boucle, la condition est évaluée. Si cette condition est vraie, la séquence d'instruction est alors évaluée.
La boucle est répétée jusqu'à ce que la condition soit fausse.

En Python, on écrit :

```
1 while condition:
2     instructions
```

On peut l'écrire en français :
Tant que la condition est vérifiée, on exécute les instructions.

Exemple

```
1 i = 5
2 while i <= 12:
3     print(i)
4     i = i + 1
```

Ce script affichera les nombres de 5 à 12 inclus, les uns en-dessous des autres.

Remarque

Si la condition porte sur une varaible, il faut absolument qu'elle soit initialisée avant la boucle.
De plus, la variable en question doit s'approcher du cas d'arrêt à chaque entrée dans la boucle, sous peine de créer une boucle qui se répète à l'infini.

VI.2. Les boucles bornées

Définition

Lorsqu'on veut répéter un certain nombre de fois une séquence d'instructions, on utilise une boucle bornée (ou boucle Pour).
On utilise pour cela un **compteur** qui s'incrémente à chaque passage dans la boucle. Dès que le compteur dépasse une valeur de fin donnée, il n'y a plus de passage dans la boucle.

Remarque

Le compteur peut prendre la forme d'un nombre qui s'incrémente mais aussi de n'importe quel type d'élément appartenant à une collection ordonnée.

En Python, on écrit :

```
1 for element in collection:
2     instructions
```

Exemple

```
>>> res = ''
>>> for caractere in 'abcdef':
        res = caractere + res
>>> res
 'fedcba'
```

Ici, le compteur sous-entendu est l'indice du caractère `caractere` qui se trouve dans la chaîne `'abcdef'`.

VI.3. la collection range

Définition

La collection `range` est une collection de nombres entiers.
Il y a trois façons de l'utiliser :

- `range(fin)` : collection des nombres entiers de `0` (inclus) à `fin` exclu.
- `range(debut, fin)` : collection des nombres entiers de `debut` (inclus) à `fin` exclu.
- `range(debut, fin, pas)` : collection des nombres entiers de `debut` (inclus) à `fin` exclu incrémentés de `pas`.

Exemple

`range(7)` correspond aux nombres 0, 1, 2, 3, 4, 5 et 6.
`range(2, 7)` correspond aux nombres 2, 3, 4, 5 et 6.
`range(2, 7, 3)` correspond aux nombres 2 et 5.
On peut utiliser la collection range dans les boucles `for` :

```
>>> chaine = 'abcdef'
>>> res = ''
>>> for indice in range(len(chaine)):
        res = chaine[indice] + res
>>> res
 'fedcba'
```