Introduction à python

1 Variables

1.1 Création de variable

Pour créer une variable en pythton, il suffit d'utiliser le symbole =

```
a = 568
```

La variable a contient maintenant la valeur 568. On lui a affecté cette valeur.

```
a
```

```
568
```

On peut évidemment changer la valeur d'une variable :

```
a = 125; a
```

```
125
```

Remarquer comment on a enchainé les commandes grâce à un ";".

On peut également affecter la valeurs de plusieurs variables en même temps.

```
a,b = 0,1
print(a)
print(b)
```

```
0
1
```

Pour exécuter plusieurs instructions, on peut également passer une ligne entre chaque. L'instruction print affiche simplement la variable.

1.2 Types des variables

Il existe plusieurs types de variables simples en python :

les booléens : True et False

les entiers (int)

les flottants (float)

les chaînes de caractères qui contienent du text (str). On peut accéder séparément à chaque caractère ou lettre. Ils sont numérotés à partir de 0

```
type(a)
```

```
builtins.int
```

```
ch = "Hello, world";type(ch)
```

```
builtins.str
```

```
ch[0]
```

```
'H'
```

```
booleen = True; type(booleen)
```

```
bool
```

```
type(3.14159)
```

```
float
```

Opérations Les opérations sur les nombres ont été vues en TD. On peut également effectuer quelques opérations sur les chaînes.

```
"a" + "bcd"
```

```
'abcd'
```

```
"abc" * 5
```

```
'abcabcabcabcabc'
```

Pour connaître la longueur d'une chaîne, on utilise la fonction len

```
len(("a" + "2")*5)
```

```
10
```

1.3 Listes

Pour construire une liste, on utilise des crochets, et on note entre ces crochets les valeurs souhaitées, séparées par des virgules. Dans une liste les valeurs sont numérotées à partir de 0. L'accès à un élément donné se fait par numéro (ou indice).

```
l = [5,9,7,2]
l[0],l[3]
```

```
(5, 2)
```

```
l[5]
```

```
␣
↪---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)

    <ipython-input-144-437134752604> in <module>()
----> 1 l[5]


    IndexError: list index out of range
```

On peut très bien mélanger les types de valeurs dans une même liste.

```
l2 = ["a", 1, 0.5, True]
l2[2]
```

```
0.5
```

```
len(l2)
```

```
4
```

1.3.1 Accès à plusieurs éléments.

Dans une chaîne de caractère comme dans une liste, on peut accéder par numéro aux éléments qui la composent. il st également possible de sélectionner plusieurs éléments en une seule fois via un slice

```
l2[1:3]
```

```
[1, 0.5]
```

```
ch[0:7]
```

```
'Hello, '
```

Remarquer que le deuxième indice est exclu de l'ensemble retourné. Si l'un des deux indices est omis, python considère que l'on veut partir du début ou aller jusqu'au bout.

```
print(ch[1:])
print(ch[:5])
```

```
ello, world
Hello
```

```
len(l2[1:])
```

```
3
```

1.4 Intervalles

Python possède une commande simple pour créer des invervalles d'entier : range

```
range(5) #cré l'ensemble d'entier {0,1,2,3,4}
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

```
range(1,5) #cré l'ensemble d'entier {1,2,3,4}
```

```
range(1, 5)
```

Comme pour les selections multiples, le deuxième indice est exclu de l'intervalle créé, et si le premier indice est omis, il vaut 0 par défaut.

Les intervalles se comportent comme des listes que l'on ne peut pas modifier.

1.5 Modifications des types composés

Les chaînes de caractères ne sont pas modifiables.

```
ch[1] = "a"
```

```
␣
↪---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)

    <ipython-input-156-8d6c53499986> in <module>()
----> 1 ch[1] = "a"


    TypeError: 'str' object does not support item assignment
```

Par contre les listes sont modifiables à volonté

```
l[0] = 27; l[1] = -12
l
```

```
[27, -12, 7, 2]
```

```
l[0:2] = [8,6]
l
```

```
[8, 6, 7, 2]
```

Il y a cependant un piège avec la modification des listes. Python ne stocke pas la valeurs de la liste en lien avec le nom de celle ci, mais seulement l'adresse mémoire où la trouver. Exemple :

```
l3 = [1, l]
l[0] = 0
print(l3)
print(l)
```

```
[1, [0, 6, 7, 2]]
[0, 6, 7, 2]
```

```
l3[3] = -7 #on ne peut pas étendre une liste en ajoutant directement par␣
↪indice un élément à la fin.
```

```
␣
↪---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)

    <ipython-input-160-1697fa6d3145> in <module>()
----> 1 l3[3] = -7 #on ne peut pas étendre une liste en ajoutant directement␣
↪par indice un élément à la fin.


IndexError: list assignment index out of range
```

Pour éviter de modifier un object composé, on peut utiliser les méthodes de copie.

```
l4 = l2.copy()
print(l4)
print(l2)
```

```
['a', 'b', 0.5, True]
['a', 'b', 0.5, True]
```

```
l4[0:4] = [1,2,3,4] #dernier indice exclu !
print(l4)
print(l2)
```

```
[1, 2, 3, 4]
['a', 'b', 0.5, True]
```

2 Structures de contrôle

2.1 Tests

Toute expression qui possède une valeur booléenne peut être testée pour décider de la marche à suivre ensuite.

```
if a > 5:
    print("a est grand")
else:
    print("a est petit")
a
```

```
a est petit
```

```
0
```

Plusieurs chose à remarquer :

Les mot clé if (si), then (alors) et else (sinon)

La condition (qui est vraie ou fausse suivant la valeur de a)

La première ligne se termine par ":"

Les instruction après ":" sont décalées vers la droite : on a indenté le code en ajoutant une tabulation (ou 4 espaces, c'est la convention admise)

La condition doit être une expression dont la valeur est un booléen. Les opérations logiques ont bien évidemment une traducton python :

L'égalité s'écrit == (le symbole = est déjà réservé à l'affectation de variable)

Les inégalités s'écrivent <, >, <=, >=

Le ou s'écrit or et le et s'écrit and

la négation de condition est not (condition)

2.2 Boucles

On a souvent besoin de répéter une certaine instruction un nombre donné de fois. La construction python correspondante est :

```
for i in liste:
    instructions
```

La variable i va prendre toutes les valeurs présentes dans la liste (dans l'ordre...) et pour chaque valeur de i les instructions sont exécutées.

```
a = 0
for i in range(10):
    a = a + i
a
```

```
45
```

A remarquer :

Les mots clés for (pour) et in (dans)

La fin de l'instruction for est marquée par ":"

Les instructions à excécuter sont indentées par rapport à l'instruction de boucle.

On utilise une liste, mais tout object iterable convient, c'est à dire tout object qui contient des éléments accéssible séquentiellement (un par un, dans un certain ordre). Par exemple les range, liste, chaînes de caractères sont des bons candidats.

```
for c in "Hello, world":
    print(c)
```

```
H
e
l
l
o
,

w
o
r
l
d
```

Application aux listes On peut utiliser la structure for in pour construire des listes. La liste est alors dite définie par compréhension

```
l = [x + 1 for x in range(5)]
l
```

```
[1, 2, 3, 4, 5]
```

2.3 Fonctions

Pour définir une nouvelle fonction dans python, rien de plus simple :

```
def ajouteUn(n):
    return n+1
```

Plusieurs chose à remarquer :

Le mot clé def qui indique la déclaration d'une fonction

Le nom de la nouvelle fonction. Ici "ajouteUn".

La liste des arguments (ou paramètres) est notée entre parenthèse

La première ligne se termine par ":"

Le corps de la fonction est indenté

Le mot clé return qui indique le résultat que doit rendre la fonction.

```
ajouteUn(6), ajouteUn(3.14159)
```

```
(7, 4.14159)
```

Toute variable (attention, pas de paramètre) qui apparaît dans la fonction est traitée comme une variable locale, c'est à dire que sa durée de vie est égale à la durée du calcul du résultat de la fonction. Elle disparaît ensuite.

```
def rien(n):
    y = 2
    return n

rien(5)
```

```
5
```

```
y
```

```
␣
↪---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)

    <ipython-input-176-009520053b00> in <module>()
----> 1 y


    NameError: name 'y' is not defined
```

Plus subtil :

```
a = 5
def rien2(n):
    a = -1
    return n

rien2(4)
```

```
4
```

```
a
```

```
5
```

La variable a est une variable locale dans la fonction, elle cache la définition précédente de a le temps du calcul puis disparaît, rendant ainsi visible l'ancienne valeur de a. Par contre, toute expression à l'intérieur du corp de la fonction (ie. indentée) a accès à la valeur de la variable a.

```
def quelquechose(n):
    a = -1
    def g():
        return a + n
    return g()
```

```
quelquechose(2)
```

```
1
```

2.3.1 Interraction avec les variables simples et composées.

```
def foisDeux(n):
    n = n*2
```

```
a = 3; foisDeux(a); a
```

```
3
```

La variable a est de type simple, sa valeur est passée directement à la fonction. Le code précédent est équivalent à :

```
a = 3; foisDeux(3); a
```

```
3
```

De même pour les chaîne, flottant et booléens :

```
f = 0.2; foisDeux(f); f
```

```
0.2
```

```
foisDeux(ch); ch
```

```
'Hello, world'
```

```
ch*2
```

```
'Hello, worldHello, world'
```

Par contre pour les listes et dictionnaires, comme on l'a vu précédement, on peut modifier les éléments :

```
def annule(l):
    l[0] = 0
```

```
print(l2)
annule(l2)
print(l2)
```

```
['a', 'b', 0.5, True]
[0, 'b', 0.5, True]
```