Leçon d'informatique : Memento Python

S. Benlhajlahsen - PCSI$_1$

Sommaire

Introduction : Cette annexe liste limitativement les éléments du langage Python (version 3 ou supérieure) dont la **connaissance est exigible des étudiants**. Aucun concept sous-jacent n'est exigible au titre de la présente annexe. Aucune connaissance sur un module particulier n'est exigible des étudiants. Toute utilisation d'autres éléments du langage que ceux que liste cette annexe, ou d'une fonction d'un module, doit obligatoirement être accompagnée de la documentation utile, sans que puisse être attendue une quelconque maîtrise par les étudiants de ces éléments.

I Traits généraux

I.A Typage dynamique

L'interpréteur détermine le type à la volée lors de l'exécution du code.

Explication : Pas besoin de préciser le type d'un objet avant de lui donner une valeur. L'affectation de la valeur et du type se fait en une seule étape (voir figure 1).

En langage python :

```
1 a =  2
2 # python comprends que a
3 # est un entier et vaut 2
```

Exemple dans le langage C :

```
1 int a ;
2 a = 2
3 # En C, on déclare d'abord
4 # le type puis la valeur.
```

FIGURE 1

I.B Principe d'indentation

L'indentation est à la base du langage Python. Elle correspond à un décalage en début de ligne (voir figure 2).

```
1     #cette phrase est indentée
2 #cette phrase n'est pas indentée
3 for i in range(2): # ligne sans indentation
4     print(i)       # ligne avec indentation
```

FIGURE 2

Remarque : Dans la plupart des éditeurs, l'indentation est obtenue par la touche de tabulation et correspond une longueurs de 2 à 4 espaces.

I.C Portée lexicale

Lorsqu'une expression fait référence à une variable à l'intérieur d'une fonction, Python cherche la valeur définie à l'intérieur de la fonction et à défaut la valeur dans l'espace global du module.

Explication : C'est la distinction entre variable globale et variable locale. Une variable locales est une variable qui n'existe pas que à l'intérieur de la fonction. À l'inverse, une variable définie dans le « programme principal »a une portée sur toute la suite du programme.

Dans l'exemple 1 de la figure 3, `a` est une variable globale et `x` est une variable locale. En ligne 5, on fait un appel de fonction `f(3)`. Python cherche la valeur locale de x qui est 3. Ensuite, il cherche la valeur de `a` qui est un paramètre global valant 2. La fonction renvoie alors `5`.
Dans l'exemple 2, il y « trois `x` ». On définit d'abord `x=2` en ligne ce qui en fait une variable globale. En ligne 11, on demande `f(3)`. Python attribue alors 3 au `x` de la ligne 8. `x` est alors modifié en ligne 9 et on renvoie en ligne 10 la dernière valeur connue soit 4. Dans cet exemple, f renverra toujours 4.

```
1  # Exemple 1 : a est global
2  a = 2
3  def f(x):
4      return x+a
5  print(f(3)) # affiche 5
6  # Exemple 2 :
7  x = 2
8  def f(x):
9      x = 4
10     return x
11 print(f(3)) # affiche 4
```

FIGURE 3 – Portée lexicale.

I.D Appel de fonction par valeur

L'exécution de $f(x)$ évalue d'abord x puis exécute f avec la valeur calculée. Dans l'exemple de la figure 4, la fonction f est définie comme $f : x \mapsto x^2$. À l'appel de la ligne 4, la fonction commence par évaluer `a+2` puis en prend le carré. Cela donne bien $(2+3)^2 = 25$.

```
1 a=3
2 def f(x):
3     return x**2
4 print(f(2+a)) # affiche 25
```

FIGURE 4 – Appel par valeur.

II Type de base

II.A Opérations sur les entiers

Le type `int` correspond aux nombres entiers [1]. Les opérations sur les entiers sont alors (voir figure 5) :

- `+` et `-` pour l'addition et la soustraction ;
- `*` pour la multiplication ;

1. *integer* en anglais.

- `a**b` renvoie l'exponentiation a^b ;
- `a//b` renvoie le quotient entier de la division euclidienne soit $\left\lfloor \frac{a}{b} \right\rfloor$;
- `a%b` renvoie le reste de la division euclidienne. Dans l'exemple, on retrouve bien que $25 = 2 \times 11 + 3$.

```
a = 5
print(type(a)) # affiche <class 'int'>
print(type(a) == int) # affiche True
# addition
print(1+2) # affiche 3
#soustraction
print(1-10) # affiche -9
#multiplication
print(3*2) # affiche 6
#dividende de la division euclidienne
print(25//11) # affiche 2
#reste de la division euclidienne
print(25%11) # affiche 3
#puissance 2^3
print(2**3) # affiche 8
```

FIGURE 5 – Opérations sur les entiers.

II.B Opérations sur les flottants

Le type `float` correspond aux nombres à virgule[2]. Les opérations sur les flottants sont alors (voir figure 5) :

- `+` et `-` pour l'addition et la soustraction ;
- `*` pour la multiplication ;
- `a**b` renvoie l'exponentiation a^b ;
- `a/b` renvoie le quotient.

Remarque : La division `a/b` de deux entiers fournit aussi un flottant.

```
a = 3.1415
print(type(a)) # renvoie <class 'float'>
print(type(a) == float) # renvoie True
# addition
print(1.2+2.3) # affiche 3.5
#soustraction
print(3.5-1.0) # affiche 2.5
#multiplication
print(1.95*2) # affiche 3.9
#division
print(25/11) # affiche 2.272727272727273
#puissance 2.5^(0.5).
print(2.5**0.5) # affiche 1.5811388300841898
# Cet exemple correspond à la racine carré.
```

FIGURE 6 – Opérations sur les flottants.

2. *float* en anglais.

II.C Opérations sur les booléens

Le type `bool` est réservé aux variables pouvant prendre deux valeurs : `True` ou `False`, c'est-à-dire vrai ou faux.

- Une erreur classique est d'oublier la majuscule initiale.
- On dispose des opérateurs logiques usuels : `and`, `or` et `not`.

On pourra se reporter à la figure 7 et les exemples de la figure 8.

a	b	a and b	a or b	not b
True	True	True	True	False
True	False	False	True	True
False	True	False	True	
False	False	False	False	

FIGURE 7

```
1 a = True
2 b = False
3 print(type(a)) # affiche <class 'bool'>
4 print(a and b) # affiche False
5 print(a and a) # affiche True
6 print(a or b) # affiche True
7 print(b or b) # affiche False
8 print(not a) # affiche False
9 print(not(b)) # affiche  True
```

FIGURE 8 – Opérations sur les booléens.

II.D Comparaisons

Pour comparer deux valeurs, on peut effectuer l'opération :

- `a==b` qui est un test d'égalité. Ainsi, `2==3` renvoie `False` alors `2==2` renvoie `True`;

  Attention : Il ne faut pas confondre `a==2` et `a=2`. Pour `a==2`, on teste si `a` vaut 2 et cela renvoie `True` ou `False`. En revanche, l'instruction `a=2` est une opération d'affectation. On affecte la valeur 2 à la variable `a`. À ce propos, on remarque que le test d'égalité est commutatif, c'est-à-dire que `a==2` et `2==a` sont deux instructions équivalentes alors que l'instruction `2=a` **n'a pas de sens** et produira une erreur dans le programme.

- `a!=b` qui est un test d'inégalité. Ainsi, `2!=3` renvoie `True`.

  Remarque : `a!=b` donne le même résultat que l'instruction `not (a==b)`.

- Les opérateurs `>`, `>=`, `<` et `<=` correspondent respectivement aux tests *strictement supérieur à*, *supérieur ou égal à*, *strictement inférieur à* et *inférieur ou égal à*. Ainsi, `2>2` renvoie `False` et `2>=2` renvoie `True`.

III Type structurés

Remarque : Les objets de types `int`, `float` ou `bool` sont des objets « simples »car il n'y a qu'une seule valeur stockées à l'intérieur. Nous introduisons, à présent, des objets plus complexes.

III.A Structures indicées immuables

III.A.1 Chaînes de caractères

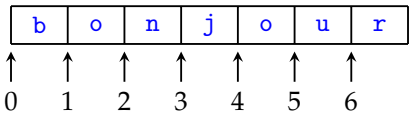
Un chaîne de caractères est constitué d'un ensemble de caractère. C'est un objet de type `str`[3] Par exemple, `a= "bon"` est constitué des tois caractères `"b"`, `"o"` et `"n"`. On notera que :

3. chaîne de caractère se dit *string* en anglais.

- Une chaîne de caractère est définie par les guillemets `"..."` ou par des apostrophes `"a"` et `'a'` correspondent à la même chaîne de caractère.

  > **Remarque :** L'objet `a="1"` correspond à la chaîne de caractère `"1"` et pas à l'entier 1. Autrement dit, `type("1")` renvoie `str` et pas `int`.

- `""` et `''` correspondent à des chaînes vides, `"a"` est une chaîne de longueur un ;
- `len` est une fonction qui renvoie la « longueur »de la chaîne de caractère, c'est-à-dire le nombre de caractère. Ainsi, `len("bonjour")` renvoie 7.
- Pour une chaîne de caractère `a` et `n=len(a)`, les caractères sont indexés de 0 à `n-1`. Dans l'exemple de la chaîne de caractères `"bonjour"`, la lettre `"b"` a l'indice 0, la lettre `"o"` a l'indice 1 et la dernière lettre `"r"` correspond à l'indice $6 = 7 - 1$.



- On accède à un caractère de la chaîne `a` par son indice `i` en faisant `a[i]`. Ainsi, `"bonjour"[2]` renvoie le troisième caractère, c'est-à-dire `"n"`.
- L'opérateur `+` entre deux chaînes les **concatène**[4]. Ainsi, `"aba"+"jour"` renvoie `"abajour"`.

  > **Remarque :** Cet opérateur n'a aucun rapport avec l'addition. En effet, l'addition entre deux entiers est commutative ($3 + 2 = 2 + 3$). Ici, `"jour"+"aba"` renvoie `"jouraba"` et pas `"abajour"`.

- L'opérateur `*` permet la répétition d'une chaîne de caractère. Ainsi, `3*"bon"` renvoie `"bonbonbon"`.
- On peut découper une partie du mot grâce au *slicing*[5]. Ainsi, pour une chaîne `a` :
  - `a[i:j]` renvoie la partie de la chaîne comprise entre les indices `i` et les indices `j` **exclu**. Par exemple, `"bonjour"[1:3]` renvoie `"on"`.

    > **Remarque** On peut aussi omettre l'un deux indices :
    > - `"bonjour"[:3]` renverra les premiers caractères jusqu'à l'indice 3 exclu soit `"bon"` ;
    > - `"bonjour"[1:]` renverra tous les caractères à partir du deuxième soit `"onjour"` ;
    > - `"bonjour"[:]` renverra une copie soit `"bonjour"` ;

III.A.2 Tuples

Un *tuple* est un ensemble indexé délimité par des parenthèses. Les élements du `tuple` peuvent être de type différents. Par exemple, `a = (10,20,30,1.52,40,'mama', False)` est un *tuple* à 7 éléments composés d'entiers, de flottants et de booléen.

- `()` est le *tuple* vide et `(3,)` est un *tuple* à un élément[6] ;
- `len( (1,3,5) )` renvoie la « longueur »du tuple, c'est-à-dire son nombre d'éléments. Ainsi, `len((1,3,5))` renvoie 3 et, pour `a = (10,20,30,1.52,40,'mama', False)`, `len(a)` renvoie 7.
- On accède aux élements par leurs indices. Ainsi, `(1,3,5)[1]` renvoie `3` et `a[5]` renvoie `'mama'`.
- l'opérateur `+` permet la concaténation. Ainsi, `(1,2)+(10,20)` renvoie `(1,2,10,20)` ;
- L'opérateur `*` permet la répétition. Ainsi, `3*(10,20)` renvoie `(10, 20, 10, 20, 10, 20)`.
- On peut hacher/slicer un tuple. Ainsi, `(10,20,30,1.52,40,'mama', False)[1:4]` renvoie les éléments compris entre les indices 1 et 4 exclus soit encore `(20, 30, 1.52)`.

III.B Listes

Une liste est un ensemble indexé délimité par des crochets. Les élements de la liste[7] peuvent être de type différents. Par exemple, `L = [10,20,30,1.52,40,'mama', False]` est une liste à 7 éléments composés d'entiers, de flottants et de booléen.

4. la concaténation correspond à une mise « bout à bout ».
5. hachage ou tranche en français.
6. Attention, `(3)` est considéré comme l'entier 3.
7. de type `list`.

- `[]` est la liste vide ;
- `len( [1,3,5] )` renvoie la « longueur »de la liste, c'est-à-dire son nombre d'éléments. Ainsi, `len([1,3,5])` renvoie 3 et, pour `L = [10,20,30,1.52,40,'mama', False]`, `len(a)` renvoie 7.
- On accède aux élements par leurs indices. Ainsi, `[1,3,5][1]` renvoie `3` et `L[4]` renvoie `40`.
- l'opérateur `+` permet la concaténation. Ainsi, `[1,2]+[10,20]` renvoie `[1,2,10,20]` ;
- L'opérateur `*` permet la répétition. Ainsi, `3*[10,20]` renvoie `[10, 20, 10, 20, 10, 20]`.
- On peut hacher/slicer une liste. Ainsi, `[10,20,30,1.52,40,'mama', False][2:4]` renvoie les éléments compris entre les indices 2 et 4 exclus soit encore `[30, 1.52]`.

> **Remarque :** Il semble ne pas y avoir de différences notables entre listes et tuples. Cependant, les tuples ou les chaînes de caractères sont des objets immuables, c'est-à-dire qu'on ne peut pas modifier un des ses éléments sans créer un nouvel objet. À l'inverse, on peut modifier un ou plusieurs éléments d'une liste. Dans l'exemple de la figure 9, on cherche d'abord à modifier le deuxième élément du *tuple* `a`. Cela produit une erreur. De même, en tentant de modifier la première lettre de `mot="bonjour"`, on obtient aussi une erreur. Par contre, on peut remplacer le deuxième éléments de la liste `L` par 100 et cela renvoie `[1,100,3]`.

```
1 a = (10,20,30)
2 a[1] = 100 # renvoie une erreur
3 mot = 'bonjour'
4 mot[0] = 't' # renvoie une erreur
5 L = [1,2,3]
6 L[1]=100
7 print(L)
```

FIGURE 9 – Les `tuple` ou `str` sont immuables alors que les `list` sont mutables.

- On peut construire des suites **par compréhension** :
  - `[i for i in range(5)]` renvoie `[0, 1, 2, 3, 4]` ;
  - `[x for x in "bonjour"]` renvoie la liste des caractères : `['b','o','n','j','o','u','r']` ;
  - `[e for i in s]` donne la liste des éléments de la structure `s`[8].
- Si on souhaite rajouter un élément à la fin d'une liste `append`, on peut faire une concaténation de liste (voir figure 10) ou utiliser la méthode `append`.

> **Attention :** Comme on le voit, la méthode `append` ne renvoie rien. Autrement dit, la méthode `append` modifie directement la liste et il ne faut pas chercher à en stocker le résultat[a].

a. qui n'existe pas ! La méthode renvoie `None` qui est un objet non-typé.

- Enfin, si on souhaite enlever le dernier élément d'une liste, on peut utiliser la méthode `pop` (voir figure 11).

III.C Dictionnaires

> **Remarque** Les tuples, listes et chaînes partagent une même structure d'indexation : leurs éléments sont indexés entre 0 et `n-1` si n est la longueur de l'objet. À présent, on va introduire les dictionnaires dont les éléments ne sont pas des entiers.

Un dictionnaire est un ensemble d'éléments, délimité par des accolades. Les éléments d'indexation peuvent être alors des entiers, des flottants, des chaînes…

- `{}` est le dictionnaire vide ;
- On considère l'exemple de la figure 12. Supposons qu'on ait un panier de fruits composés de 5 pommes, 20 bananes et 10 fraises. On construit un dictionnaire qui associe le fruit à la quantité. Dans le programme, le dictionnaire peut être créer en une seule étape ou progressivement. On dit que `"pomme"`, `"banane"` et `"fraise"` constituent les **clés** du dictionnaire[9] et `5`, `20` et `10` constituent les **valeurs** du dictionnaire[10].

8. liste, tuple, chaîne, dictionnaire…
9. *keys* en anglais.
10. *values* en anglais.

```python
L = [1,2,3]
# methode 1 :
L2 = L+[4]
print(L2) # affiche [1, 2, 3, 4]
# methode 2 :
L.append(4)
print(L) # affiche [1, 2, 3, 4]
# a éviter !!!
L= L.append(4)
print(L) # affiche None
```

FIGURE 10 – Utilisation de `append`.

```python
# méthode 1 :
L = [1,2,3]
L.pop()
print(L) # affiche [1, 2]
# méthode 2 :
L = [1,2,3]
L2 = L[:len(L)-1]
print(L2) # affiche [1, 2]
# méthode 2 :
L = [1,2,3]
L3 = L[:-1]
print(L3) # affiche [1, 2]
```

FIGURE 11 – Utilisation de `pop`.

- pour un dictionnaire `d`, on accède à ses valeurs par `d.values()` ;
- pour un dictionnaire `d`, on accède à ses clés par `d.keys()` ;

  Remarque : Si on veut la liste des clés, on peut utiliser la synthaxe `list(d.keys())`. De même, le *tuple* des clés sera obtenu par `tuple(d.keys())`.

```python
# on construit le dico en une seule étape
d = {"pomme":5, "banane":20, "fraise":10}
type(d)           # affiche <class 'dict'>
d.keys()          # affiche les clés
d.values()        # affiche les valeurs
d['pomme']        # affiche 5
len(d)            # affiche 3
```

```python
d = {}              # dictionnaire vide
#       on le remplit progressivement
d["pomme"] = 5
d["banane"] = 20
d["fraise"] = 10
print(d) # affiche
# {'pomme':5,'banane':20,'fraise':10}
```

FIGURE 12 – Exemple de création de dictionnaire.

- `len(d)` permet d'accéder la longueur du dictionnaire.

  Remarque : Il ne peut pas y avoir deux clés identiques. Par contre, deux valeurs peuvent être identiques[a]. Ainsi, `len(d)` renvoie le nombre de clés du dictionnaire.

  a. Par exemple, `{"a":1,"b":1}` est un dictionnaire pour lequel les deux clés sont associés à la même valeur.

IV Structure de contrôle

IV.A Instruction d'affectation

L'opérateur `=` permet d'affecter[11] à une variable à gauche le résultat à droite. Dans la synthaxe a=3+2, Python calcule d'abord le résultat à droite et ensuite l'affecte à la variable à gauche[12].

Empaquetage et dépaquetage pour les *tuple* : On prend l'exemple de la figure 13. La création du *tuple* `a` revient à empaqueter[a] des valeurs à l'intérieur de cet objet. On rappelle que le *tuple* est immuable. Pour **dépaqueter**[b] les valeurs, on peut utiliser la synthaxe de la ligne 2 où on attribue à `x` et à `y` les deux valeurs empaquetées dans le *tuple*.

a. *packing* en anglais.
b. *unpacking* en anglais.

```
1 a = (10,20)
2 x, y = a
3 print(x) # affiche 10
4 print(y) # affiche 20
```

FIGURE 13 – Dépaquetage pour le *tuple*.

Double affectation ou affectation en parallèle : Une conséquence immédiate est l'affectation en parallèle de plusieurs variables. L'instruction `a,b=1,2` affecte simultanément la valeur 1 à la variable `a` et la valeur 2 à la variable `b`.

Permutation de valeurs : Une autre conséquence immédiate est la permutations de valeurs. de plusieurs variables. L'instruction `a,b=b,a` permutte les valeurs de `a` et de `b`.

```
1 a = 10
2 b = 20
3 a,b = b,a
4 print(a) # affiche 20
5 print(b) # affiche 10
```

FIGURE 14 – Permutation de valeurs.

IV.B Instruction conditionnelle

Une instruction conditionnelle simple est introduite avec la mot `if`[13] (voir figure 15). L'instruction ne sera exécutée que si la clause est vérifiée[14]. Dans le cas où la clause n'est pas vérifiée, l'instruction est ignorée et on passe à la suite du programme.

Remarque : On notera la présence des deux points `:` et de l'indentation pour le bloc d'instruction.

Si l'on veut que le programme fasse une autre instruction lorsque la clause n'est pas vérifiée, on peut utiliser `else`[15] (voir figure 16).
Enfin, on peut utiliser plusieurs clauses à l'aide de `elif`[16] (voir figure 17).

IV.C Boucle `while`

Une boucle[17] a pour objectif de répéter une instruction un certain nombre de fois. La boucle `while`, aussi appelle boucle conditionnelle, répétera une instruction tant qu'une condition est vérifiée.

11. On parle aussi d'assignation ou de déclaration.
12. Cela explique que la synthaxe `3+2=a` est fausse !
13. « si »en français.
14. Elle vaut `True`.
15. « sinon »en français.
16. contraction de *else* et de *if*.
17. *loop* en anglais.

```
if clause:
    instruction
# Exemple
if 3>2:
    print("3 est supérieur à 2")
```

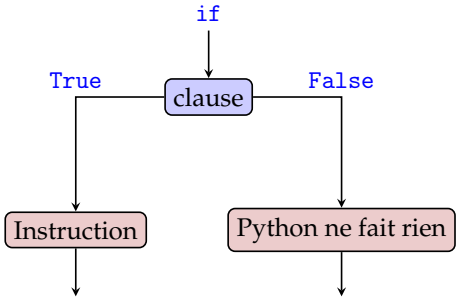


FIGURE 15 – Structure conditionnelle simple.

```
if clause:
    instruction1
else:
    instruction2
#Exemple :
if x>2:
    print("x est supérieur à  2")
else:
    print("x n'est pas supérieur à 2")
```

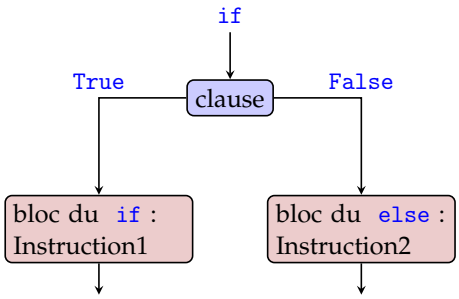


FIGURE 16 – Structure conditionnelle avec `else`.

Remarque : On notera la présence des deux points `:` puis de l'indentation.

Dans l'exemple 1 de la figure 18, `i` vaut initialement 0, on peut rentrer dans la boucle si `i<4`, ce qui est le cas initialement. À chaque tour de boucle, on commence par afficher la valeur de `i` puis on l'augmente de 1. Bien évidemment, au bout de plusieurs tours de boucle la condition `i<4` n'est plus vérifiée et on sort de la boucle. Cet exemple va afficher successivement 0, 1, 2 puis 3.

Remarque : Il faudra veiller à ce que la boucle ne s'arrête jamais. Dans l'exemple 2 de la figure 18, comme `i` reste constamment égal à 0, la clause de la boucle est toujours vérifiée et le programme affichera indéfiniment 0 : c'est la boucle infinie ! Sauf cas exceptionnel, c'est à éviter.

IV.D Boucle `for`

Une boucle `for`, aussi appelée boucle inconditionnelle, va répéter une instruction un nombre de fois prédéterminé. On utilise pour cela une variable que va prendre les valeurs appartenant à un ensemble (liste, *tuple*,...). Ainsi, dans les exemples 1 à 4 de la figure 19, la variable de la boucle `for` va prendre successivement les valeurs appartement à la liste, *tuple*, chaîne.

Utilisation de `range` : Dans les exemples de 5 à 8 de la figure 19, on utilise `range` qui renvoie des valeurs entières :

- `range(10)` renvoie les valeurs entières comprises entre 0 et 9 ;
- `range(3, 10)` renvoie les valeurs entières comprises entre 3 et 9 ;
- `range(3, 10,2)` renvoie les valeurs entières comprises entre 3 et 9 par incrément[a] de 2, c'est-à-dire 3, 5, 7 et 9 ;
- `range(10,2,-1)` renvoie les valeurs entières comprises entre 10 et 3 par incrément de -1, c'est-à-dire 10, 9, 8, 7, 6, 5, 4 et 3 ;

On retiendra que la synthaxe de range est :

```
range(start, stop, step)
```

```python
if clause1:
    instruction1
elif clause2:
    instruction2
else:
    instruction3
# exemple 3 :
if x>2:
    print("x est supérieur à  2")
elif x<10 :
    print("x est aussi inférieur à 10")
else:
    print("x n'est pas dans ]2;10[")
```

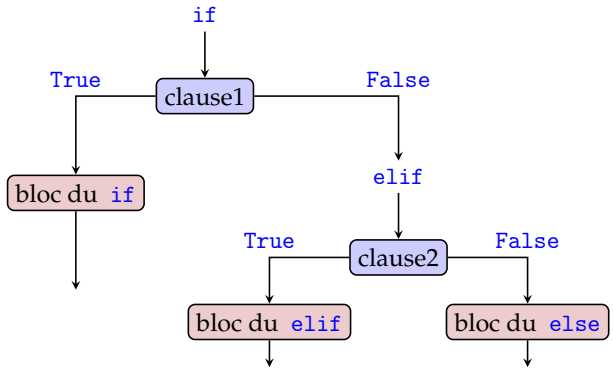


FIGURE 17 – Structure conditionnelle avec `elif`.

```python
while condition:
    instruction
# Exemple 1
i = 0
while i<4:
    print("i")
    i = i + 1
# Exemple 2
# piège à éviter : infinite loop
i=0
while i<4:
    print(i)
```

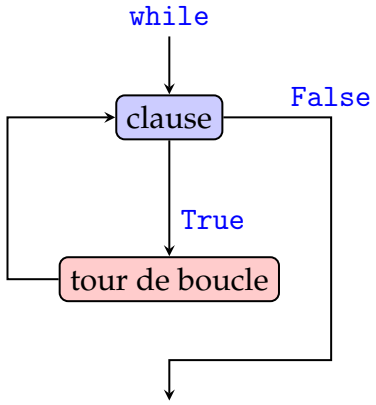


FIGURE 18 – Boucle `while`.

c'est-à-direqu'on renvoie les valeurs dans ⟦`start`, `stop-1`⟧ par incrément `step`.

a. ou pas

IV.E Définition d'une fonction `def`

La définition d'une fonction commence par `def`. Dans l'exemple 1 de la figure 20, on définit la fonction $f_1 : x \mapsto x+2$. On notera la présence des deux points « `:` », l'indentation à la ligne suivante et la présence du `return`. Ensuite, l'exemple 2 (fonction f_2), la valeur retournée par la fonction dépendra du résultat du test conditionnel. Puis, pour l'exemple 3, on a placé deux `return` mais seul le premier est pris en compte.

Remarque : Au premier `return` rencontré, on « sort »de la fonction et on exécute la suite du programme.

Différence entre `return` et `print` : On pourrait croire que `return` joue le même rôle que `print`. Cependant, dans l'exemple 4, on construit deux fonctions qui l'une **renvoie** `x` et **affiche** `x`. On tente ensuite de multiplier $f_4(2)$, ce qui renvoie 4, et $f_5(2)$ par 2 qui renvoie un message d'erreur. En effet, comme la fonction $f_5(2)$ ne renvoie rien, on ne peut pas le multiplier par 2. On dit alors que $f_5(2)$ est **non-typé**.

V Divers

```python
# Exemple 1 : avec une liste
L = [1,3,5]
for i in L:
    print(i)
# Exemple 2 : avec un tuple
t = (1,3,5)
for j  in t:
    print(i)
# Exemple 3 : avec une chaine
mot = "bonjour"
for lettre in mot:
    print(lettre)
# Exemple 4 : avec un dictionnaire
dico = {"pomme":3,"poire":10}
for cle in dico:
    print(cle)
for cle in dico.keys():
    print(cle)
for valeur in dico.values():
    print(valeur)
```

```python
# Exemple 5 :
for i in range(10):
    print(i)
# Exemple 6 :
for i in range(3,10):
    print(i)
# Exemple 7 :
for i in range(3,10,2):
    print(i)
# Exemple 8 :
for i in range(10,2,-1):
    print(i)
```

FIGURE 19 – Boucle `for`.

V.A Introduction d'un commentaire avec `#`

Une ligne de code commençant par le croisillon `#` ne sera pas exécuté. On dit que c'est un commentaire. Celui-ci permet d'aider. une personne, autre que le concepteur, de comprendre ce que fait le programme.

> **Remarque :** Une autre façon de commenter un programme sur plusieurs est d'utiliser des triples guillemets `"""..."""` ou des triples apostrophes `'''...'''` (voir figure 21).

V.B Utilisation simple de print

La fonction `print` permet de faire afficher un résultat sur l'interpréteur. Ainsi, `print("a")` affichera la lettre `"a"` et `print(1+2, "toto")` affichera 3 puis `"toto"`.

> **Remarque hors-programme :** La fonction `print` possède deux paramètres intéressants. On peut préciser la séparation entre deux affichages et la fin de de l'affichage. Par exemple, `print("bon","jour", sep="++", end="&&")` affichera `bon++jour&&`.

V.C Importation d'un module

On parle aussi de *package* ou bibliothèque.

> **Exemple du module `math` :** Le module `math` permet l'utilisation de fonctions mathématiques comme `cos` ou `exp`. En figure 22, on propose trois façons d'utiliser ce module :
>
> - on importe d'abord le module par `import math`. Pour utiliser la fonction sinus de ce module, on exécute `math.sin(x)`. Dans ce cas, tous les objets de ce module commenceront par `math.` ;
> - Pour éviter de recopier le nom de la bibliothèque à chaque fois, on peut donner un *alias*[a] à cette bibliothèque. Par exemple avec `import math as m`, les fonctions seront appelés comme `m.sin(x)` ou `m.exp(x)` ;
> - Enfin, on peut importer tous les objets d'une bibliothèque sans *alias* par `from math import *`. Dans ce cas, on peut utiliser « directement »les fonctions sans préciser qu'elles viennent de cette bibliothèque.

a. c'est-à-dire un surnom.

```python
# Exemple 1
def f1(x):
    return x+2
# Exemple 2 :
def f2(x):
    if x>0:
        return 1
    else:
        return -1
# Exemple 3 :
def f3(x):
    return 3*x
    return 5*x
# Exemple 4 :
def f4(x):
    return(x)
print(2*f4(2)) # renvoie 4
def f5(x):
    print(x)
print(2*f5(2)) # renvoie :
# TypeError: unsupported operand type(s) for *: 'int' and 'NoneType'
```

FIGURE 20 – Déclaration d'une fonction avec `def`.

```python
# un commentaire
# un autre commentaire
# un troisième commentaire

def f(x):
    return x**2

"""
des commentaires
sur plusieurs
lignes...
"""
```

FIGURE 21 – Commentaires dans un programme.

V.D Manipulation de fichiers

Les données d'un ordinateur sont stockées dans des fichiers, eux-mêmes disposés dans des dossiers ou repertoires. Un fichier se caractérise par :

- son nom et une extension [18] qui indique le type de fichier. C'est une chaîne de caractère de la forme :
  - `"mon_repertoire.txt"` sera un fichier texte ;
  - `"la_marseilleise.mp3"` sera un fichier audio ;
  - `"lecon.pdf"` sera un fichier pdf qui est un format de document portable assez universel pour l'impression ;
- un chemin d'accès qui indique le repertoire de stockage :
  - pour les OS [19] Linux ou Mac, le chemin est de la forme : `"/Users/jeandupont/Documents/"` ;
  - pour les OS Windows, le chemin est de la forme : `"C:\Users\jeandupont\Documents\"`.

  On fera la différence entre le *slash* `/` et le *baskslash* `\`.

La manipulation des fichiers-texte en python se ref-LE0001-85 fait au moyen des commandes :

18. ou suffixe
19. *operating system*

```
import math
print(math.pi)
print(math.sin(10))
def f(x):
    return math.exp(x)
```

```
import math as m
print(m.pi)
print(m.sin(10))
def f(x):
    return m.exp(x)
```

```
from math import *
print(pi)
print(sin(10))
def f(x):
    return exp(x)
```

FIGURE 22 – Trois utilisations du module `math`.

- `open("chemin/nom_fichier", "r")` si on veut lire le fichier. On indiquera `"w"` si on souhaite écrire dessus ;
- `write("chaine")` permet d'écire une chaîne sur le fichier ;
- `close` qui permet de terminer le processus d'écriture. Deux exemples sont donnés en figure 23.

```
# lecture d'un fichier
f=open("/Users/Documents/texte.txt", "r")
L = f.readlines()
f.close()

print(L) # affiche la liste des lignes du
    texte
```

```
# écriture d'un poème japonais
f=open("/Users/Documents/haiku.txt", "w")
f.write("Un vieil étang.\n")
f.write("Une grenouille qui plonge,\n")
f.write("Le bruit de l'eau.\n")
f.close()
```

FIGURE 23 – Lecture et écriture d'un fichiers texte.

V.E Assertion : `assert`

L'instruction assert aide à détecter les problèmes au début de votre programme, où la cause est claire, plutôt que plus tard comme effet secondaire d'une autre opération.

```
def f1(x):
    return x**0.5
print(f1(-4))
# renvoie un complexe
# (1.2246467991473532e-16+2j)
def f2(x):
    assert x>0
    return x**0.5
print(f2(-4))
# renvoie un message d'erreur
```

FIGURE 24 – utilisation de la méthode `assert`.