PRÉPARATION A.T.S GÉNIE CIVIL
LYCÉE CANTAU D'ANGLET

Génie Civil

Débuter avec python - Algorithmique

Philippe Lawrence

LYCEE CANTAU

9 janvier 2021

Débuter avec python - Algorithmique

Philippe Lawrence

9 janvier 2021

Table des matières

Un programme informatique sert à automatiser des tâches répétitives et complexes comme par exemple une résolution mathématique, du traitement d'images, de video, etc ...Différents langages de programmation comme python, C, C++, ...permettent de créer des programmes. Nous allons utiliser le premier cité : le langage appelé "Python". C'est un logiciel libre multiplateforme. Ce langage sait quasiment tout faire dans des domaines variés comme le calcul, le logiciel, les applications web.

1 Fonctionnement du programme

Sous windows, le langage python est inclu dans une interface graphique qui s'appelle "idle". Cette interface se présente sous la forme de plusieurs fenêtres. La fenêtre principale porte le nom d'interpréteur. Un prompt (>>>) donne la main à l'utilisateur pour saisir du code python. Il existe des raccourcis clavier pour pouvoir relancer une commande déjà utilisée, ce qui est fort pratique.
Dès qu'on veut exécuter plusieurs lignes de langage, il est indispensable d'ouvrir en plus l'éditeur de fichier en allant dans le menu fichier "file/new"' ou "file/open"' de la fenêtre principale. Le programme est écrit directement dans l'éditeur, puis il est exécuté en allant dans le menu "run/run module" ou raccourci F5 (de la fenêtre éditeur).

2 Affectation

Comme en mathématique, nous avons besoin de déclarer par exemple que x vaut 0 ($x = 0$). On parle d'affectation, et on doit comprendre qu'on a crée une variable nommé x et prenant la valeur 0. Sur l'algorigramme [1], la représentation d'une affectation est la suivante :
Représentation : $x \leftarrow 0$
En python, une affectation se fait en écrivant :

```
1  x = 0
2  print(x) # on peut afficher la valeur de x
```

La fonction `print()` est une fonction standard intégrée au langage qui sert à produire un affichage sur la sortie standard (interpréteur). Nous verrons plus loin les fonctions crées par l'utilisateur.

3 Opérations élémentaires

3.1 Addition

Représentation : $x \leftarrow 0$
$x \leftarrow x+5$

```
1  x = 0 # initialisation obligatoire
2  x = x + 5 # impossible en math!!
3  print(x)
```

L'addition peut aussi être écrite `x += 5`.

3.2 Soustraction

Représentation : $x \leftarrow x-4$

```
1  x = 0
2  x = x - 4
3  print(x)
```

1. Un algorigramme est une représentation graphique d'une partie de programme. Les algorigrammes ne permettent la représentation que de programmes simples et ne sont pas adaptés aux structures rencontrées dans les langages de haut niveau comme python.

3.3 Multiplication

Représentation : $x \leftarrow x * 10$

```
x = 0
x = x * 10
print(x)
```

3.4 Division

Représentation : $x \leftarrow x/4$

```
x = 10
x = x / 4
print(x)
```

3.5 Puissance

La puissance x^y est obtenue à partir de deux symboles multiplier.

```
x = 10
print(x**2)
```

4 Type de variables

Dans le début de ce document, nous avons déjà manipulé deux types de valeurs : les nombres entiers et les nombres décimaux. Il y en a beaucoup d'autres comme les listes, les chaînes de caractères, les dictionnaires ... Ces types de valeurs appartiennent à des classes.

La fonction **type** permet de connaitre la classe d'une valeur. Par exemple, la classe des entiers est **int** (integer) et la classe des nombres décimaux ou flottant est **float** [2].

```
x = 10
print(type(x)) # affiche <type 'int'> : entier
x = 10.
print(type(x)) # affiche <type 'float'>
print(type(10+10.)) # <type 'float'>
```

Les classes possèdent des fonctions spéciales qu'on appelle des *méthodes*.

5 fonctions et méthodes

5.1 Les fonctions

Une fonction est un outil servant à effectuer des opérations sur les variables. Il existe des fonctions appartenant au langage (dite standard comme **int**, **float**, **type** ...) et des fonctions crées par l'utilisateur. Il existe de nombreuses fonctions disponibles. Les fonctions standards peuvent s'appliquer à plusieurs types de variables.

5.2 Les méthodes

Une méthode produit un travail similaire à celui d'une fonction. La différence provient du fait qu'une méthode est attachée à une classe et ne peut être utilisée en dehors de ce contexte. La syntaxe (la façon d'appeler une méthode) est par ailleurs complètement différente de celle d'une fonction.

Par exemple, si x est une variable, une méthode de x s'utilise de la façon suivante :

2. **int()** et **float()** sont par ailleurs aussi des fonctions pour convertir en entier ou en nombre décimal

```
    x.une_methode() # sans argument
    x.une_methode(arg1, arg2)  # avec arguments
```

Une méthode est en fait une fonction qui n'existe que pour une classe. Une fonction est plus générale. Par exemple, la fonction **print** affiche aussi bien un entier, un décimal ou une liste. Pour connaître la liste des méthodes existantes, il existe la fonction **dir**.

```
    print(dir(10))
```

```
>>>['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__', '__delattr__']
```

6 Chaînes de caractères

Appelée *string* en anglais, une chaîne de caractère est une valeur constituée d'une succession ordonnée de plusieurs caractères. Typiquement, une variable qui contient un mot ou un texte est une chaine de caractère. Afin de différentier une chaîne d'une variable, les caractères sont placés à l'intérieur d'apostrophe (simple ou double).

```
    s = "une chaine de caractères"
    s = 'une autre chaine avec des 1 2 ? - ! *'
    s = '1' # ici 1 n'est pas un entier mais une chaine!!
```

Chaque caractère de *s* possède une position dans *s*, on parle d'*index*. On commence à compter à l'index 0.

6.1 Découpage des chaînes de caractères

Découper une chaîne de caractères sert à en extraire une partie.
Pour extraire un caractère, on utilise pour cela l'index du caractère placé entre crochet.

```
    s = "Bonjour"
    s[0] # retourne 'B'
    s[1] # retourne 'o'
    s[-1] # retourne 'r', on compte à partir de la fin
```

L'index -1 correspond au dernier caractère.
Pour extraire plusieurs caractères, on procède comme suit :

```
    s[0:3] # retourne 'une'
    s[4:10] # retourne 'chaine'
    s[4:] # retourne à partir de 4 jusqu'à la fin
```

6.2 Méthode des chaînes de caractères

La liste des méthodes pour une chaîne est obtenue avec l'instruction **dir**(s).

a) Méthode strip

```
    s = s.strip(" ") # supprime les espaces
    # au debut ou a la fin de s
    s = s.strip("\n") # supprime les retours a la ligne
    # au debut ou en fin de chaine
```

Remarque : '\n' est un caractère spécial correspondant à un retour à la ligne.

b) Méthode split

Cette méthode découpe une chaîne autour du motif indiqué comme argument et crée une liste.

```
    s = "une chaine de caracteres"
    li = s.split(" ")
    print(li)
```

Produit :

```
>>>['une', 'chaine', 'de', 'caracteres']
```

c) Recherche de la position d'un élément dans une chaîne

Il est parfois utile de pouvoir trouver l'index (c'est à dire la position) d'un caractère dans une chaîne de caractères. Pour cela, nous utilisons la méthode `index` ou la méthode `find`.

```
1    x = "Bonjour bonjour"
2    print(x.index("j")) # affiche 3
3    print(x.find("j"))  # affiche 3
4    print(x.index("q")) # déclenche une erreur
5    print(x.find("q"))  # affiche -1  (caractère non trouvé)
```

Remarque 1 : la méthode `find` retourne -1 si le caractère n'est pas présent dans la chaîne, alors que la méthode `index` génère une erreur. On préférera donc la méthode `find`.
Remarque 2 : si le caractère est présent plusieurs fois, les deux méthodes retournent l'index de la première occurrence.

6.3 Formatage des chaines de caractères

Cette opération consiste à insérer des valeurs dans des chaines de caractères, généralement pour produire un affichage.

```
1    x = 365*24*3600
2    message = "Le nombre de secondes dans une année est de %d" % x
3    print(message)
```

ou bien :

```
1    print("Le nombre de secondes dans une année est de %d" % x)
```

L'option de formatage `%d` permet la conversion de l'entier x en chaine de caractère. Afin d'insérer une chaine, l'option de formatage serait `%s`. L'option de formatage `%f` considère la valeur comme un nombre décimal et l'affiche avec six chiffres après la virgule. Pour le formatage `%.2f`, le modificateurs ".2" de l'option tronque la valeur à deux chiffres après la virgule.

7 Listes

Une liste est une structure que nous pouvons décrire comme un tableau à une seule ligne servant à ranger différents éléments. Chaque élément dans la liste est repéré par sa position. Une liste se définit en utilisant les crochets :

```
1 li = [1, 2.2, 8, 4]  # liste contenant 1, 2.2, 8, 4
2 print(li)
```

7.1 Nombre d'éléments d'une liste

On parle de longueur de la liste et est obtenue à l'aide de la fonction **`len`**[3].

```
>>> len(li)
4
```

3. Cette fonction sert aussi à déterminer la longueur d'une chaine de caractères, d'un dictionnaire ...

7.2 Découpage d'une liste

On utilise la position de l'élément dans la liste afin de récupérer cet élément, en commençant à compter à 0, de la même façon que pour les chaînes. C'est à dire que l'index du premier élément est 0, celui du dernier est $n-1$ si la liste a n éléments. La syntaxe est la suivante :

```
1 li = [1, 2.2, 8, 'chaine']
2 x = li[0] # x = 1
3 y = li[3] # y = 'chaine'
4 print(x, y)
```

Outre des nombres et les chaînes, une liste peut elle-même contenir d'autres listes, on parle alors de sous-listes. Exemple :

```
1 li = [[0, 0], [9, 7]]
2 x = li[0] # x = [0, 0]
3 y = li[1] # y = [9, 7]
4 print(y[1]) # affiche 7
5 print(li[1][1]) # affiche 7 aussi
```

7.3 Méthodes des listes

a) Ajout d'un élément dans une liste

On utilise la méthode `append` qui va ajouter l'élément souhaité à la fin de la liste. Il faut respecter la syntaxe suivante :

```
1 li = [1, 2.2, 8, 4]
2 li.append(10) # on ajoute 10 en fin de liste
3 print(li)
```

Il n'est pas possible d'ajouter un élément à une liste si celle-ci n'existe pas. Il faut donc initialiser la liste en écrivant `li = []`, ce qui correspond à une liste vide.

b) Suppression d'un élément dans une liste

On utilise la méthode `remove` en spécifiant la valeur que l'on veut supprimer :

```
1 li = [1, 2.2, 8, 4]
2 li.remove(2.2) # suppression du second élement
3 print(li)
```

c) Recherche de la position d'un élément dans une liste

Il est parfois utile de pouvoir trouver l'index (c'est à dire la position) d'une valeur dans une liste. Pour cela, nous utilisons la méthode `index`.

```
1 li = [1, 2.2, 8, 4]
2 print(li.index(2.2)) # affiche 1
```

8 Les dictionnaires

Un dictionnaire en python est une structure assez proche d'une liste mais au lieu d'utiliser des index numériques (0, 1, 2 . . .), on utilise des clés. Contrairement à une liste, un dictionnaire n'est donc pas ordonné et parler du premier élément du dictionnaire n'a aucun sens.

8.1 Comment créer un dictionnaire ?

Pour initialiser un dictionnaire , on utile la syntaxe suivante :

```
a = {}
```

8.2 Comment ajouter des valeurs dans un dictionnaire ?

Pour ajouter des valeurs à un dictionnaire il faut indiquer une clé ainsi qu'une valeur :

```
>>> a = {}
>>> a["nom"] = "engel"
>>> a["prenom"] = "olivier"
>>> a
  {'nom': 'engel', 'prenom': 'olivier'}
```

Vous pouvez utiliser des clés numériques comme dans la logique des listes .

8.3 Récupérer une valeur dans un dictionnaire

La méthode get vous permet de récupérer une valeur dans un dictionnaire et si la clé est introuvable, vous pouvez donner une valeur à retourner par défaut :

```
>>> di = {"name": "Olivier", "age": 30}
>>> di.get("name")
'Olivier'
>>> print(di.get("adresse"))
None
>>> print(di.get("adresse", "Adresse inconnue")) # valeur par défaut
 'Adresse inconnue'
```

8.4 Vérifier la présence d'une clé dans un dictionnaire

Vous pouvez utiliser le mot clé **in** pour vérifier la présence d'une clé que vous cherchez :

```
>>>di = {'a':1, 'b':5}
>>> "a" in di
True
>>> "c" in di
False
```

8.5 Supprimer une entrée de dictionnaire

Il est possible de supprimer une entrée en indiquant sa clé, comme pour les listes :

```
>>> del di["a"]
>>> di
{'b': 5}
```

8.6 Récupérer les clés par une boucle

Pour récupérer les clés on utilise la méthode `keys()`.

```
>>> fiche = {"nom":"engel","prenom":"olivier"}
>>> for key in fiche.keys():
...     print(key)
...
```

```
nom
prenom
```

8.7 Récupérer les valeurs par une boucle

Pour cela on utilise la méthode values

```
>>> fiche = {"nom":"engel","prenom":"olivier"}
>>> for value in fiche.values():
...     print(value)
...
engel
olivier
```

9 Les fonctions

Ecrire un programme nécessite souvent d'écrire des centaines voire des milliers de lignes de code. Le code devient vite dense et illisible sans l'utilisation de fonction. Une fonction est un morceau de code, séparé du reste du programme, effectuant une opération bien définie. Un autre intérêt des fonctions est de pouvoir être utilisées autant de fois que nous le souhaitons sans être obligé d'écrire du code en double. Une fonction est un outil.

9.1 Syntaxe d'une fonction

Les fonctions sont crées avec le mot clé **def** et doivent être placées en début de programme. Les lignes dans la fonction sont indentées d'une tabulation.

```
def ma_fonction():
    faire_quelque_chose
    faire_autre_chose

x = 0 # on n'est plus dans la fonction
ma_fonction() # appel de la fonction
```

9.2 Indentation du code

Plutôt que d'utiliser des accolades pour indiquer où commence et où se termine la fonction, le langage python a choisi un système d'intentation des lignes, c'est à dire qu'on décale le code vers la droite d'une tabulation.

9.3 Valeur en retour d'une fonction

Une fonction peut soit effectuer des choses comme un affichage à l'écran, soit renvoyer une valeur, par exemple le résultat d'un calcul. Dans ce dernier cas, il faut utiliser le mot clé **return** suivi de la valeur à retourner. Exemple :

```
def ma_fonction():
    x = 10
    y = 2
    return x*y

# appel de la fonction
a = ma_fonction()
print(a)
```

Remarque : sans le mot clé **return**, une fonction retournera `None` (élément vide).

9.4 Arguments d'une fonction

Il est souvent utile de passer des valeurs du programme vers la fonction. Pour cela, un système appelé "arguments de la fonction" a été crée. Il consiste à passer les variables en les plaçant à l'intérieur des parenthèses. Il peut y avoir 0 ou plusieurs arguments (2 arguments dans l'exemple ci-dessous).

```
1 def ma_fonction(a, b):
2     print(a)
3     print(b)
4
5 # premier appel
6 ma_fonction(10, 11)
7 # second appel
8 ma_fonction(20, 21)
```

9.5 Exemple

On souhaite créer une fonction puissance qui calcule x^y, x et y étant des arguments passés à la fonction.

```
1 def puissance(x, y):
2     z = x**y
3     return z
4
5 x = puissance(10, 2)
6 print(x)
```

La figure 1 illustre la fonction "puissance" sous forme d'algorigramme.

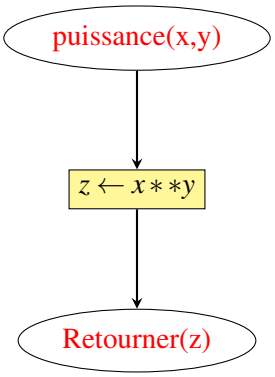

FIGURE 1 – Algorigramme de la fonction "puissance"

10 Les valeurs booléennes

La plus part des grandeurs informatiques en python ont une valeur booléenne associée. On appelle valeur booléenne un valeur du type vrai ou faux : en python : `True` ou `False`. Il existe la fonction **`bool`** qui permet de tester la valeur booléenne d'une valeur. Par exemple :

```
>>> bool(0) #valeur booléenne de l'entier zéro
>>> False
>>> bool(1) #valeur booléenne de l'entier 1
>>> True
>>> bool([]) #valeur booléenne d'une liste vide
>>> False
>>> bool([0]) #valeur booléenne d'une liste contenant un zéro
```

```
>>> True
```

On comprendra mieux dans la suite la force des grandeurs booléennes. Nous venons de voir que toutes valeurs possèdent sa correspondance en valeur booléenne. La comparaison de deux valeurs va également retourner une valeur booléenne.

10.1 Les comparaisons

Les comparaisons fonctionnent de pair avec les tests présentés au paragraphe suivant. En comparant des éléments, il devient possible d'orienter le programme dans telle ou telle direction.

La valeur renvoyée par une comparaison est une grandeur booléenne, c'est à dire qu'elle ne peut prendre que deux valeurs vrai ou faux. Par exemple, considérons a et b deux variables contenant des nombres. Il est primordial de pouvoir comparer a et b, en vue d'effectuer telle ou telle tache en fonction du résultat .

Les principales comparaisons sont l'égalité "==", la différence "! =", la supériorité ">" ou ">=", l'infériorité "<" ou "<=" ...

```
>>> a = 1
>>> b = 2
>>> a == b
False
>>> a != b
True
>>> a > b
False
>>> type(a==b)
<type 'bool'>
```

10.2 Les tests

Un test permet d'effectuer des opérations différentes selon qu'une condition est vraie ou fausse. Un test utilise obligatoirement l'instruction **if** (si) ainsi qu'une comparaison. Il est possible d'ajouter une autre condition, par exemple un **else** (sinon). Les lignes qui se trouvent dans le test doivent être indentées d'une tabulation supplémentaire comme dans les fonctions.

```
1 x = 5
2 y = 6
3 if x == y:
4     print("Egal")
5 else:
6     print("Différent")
```

La figure 2 illustre le test "si / sinon" sous forme d'algorigramme.

10.3 Le bon usage des tests en python

Tester si une liste est vide : plutôt que de calculer et tester la longueur d'une liste **if len(li) == 0:**, préférer :

```
1 li = []
2 if not li:
3   print("Liste vide")
```

Le mot clé **not** permet d'opposer le résultat de la comparaison.
Tester si une valeur est présente dans une liste :

```
1 li = [1, 3, 2]
2 x = 4
3 if x in li:
4   print("Présent dans la liste")
```

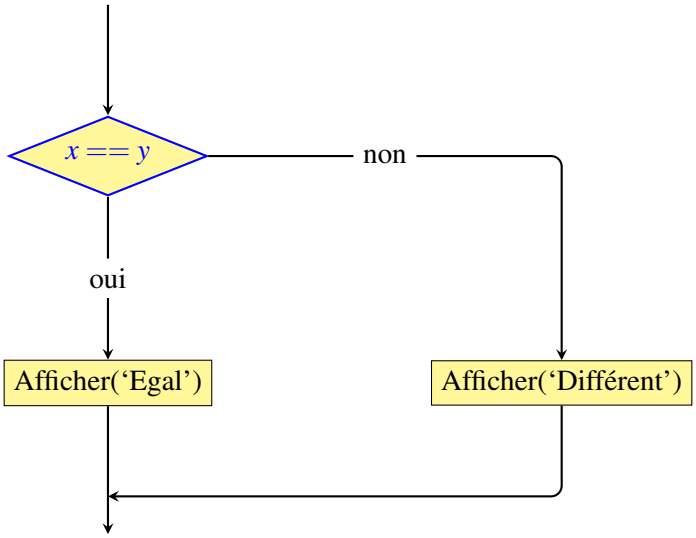

FIGURE 2 – Algorigramme : représentation d'un test

Le mot clé **in** permet de tester la présence d'un élément dans un autre élément de type liste ou dictionnaire.

11 Les boucles

Une boucle est un dispositif permettant de répéter une séquence d'opérations un nombre contrôlé de fois. Chaque répétition porte le nom d'itération. Les lignes qui se trouvent dans une boucle doivent être indentées d'une tabulation supplémentaire comme dans les fonctions.

11.1 La boucle "pour"

La boucle **for** est utilisée pour répéter une opération *n* fois, *n* étant le nombre d'éléments présents dans la liste donnée. Les éléments de la liste sont pris un par un, du premier au dernier, puis le processus s'arrête quand il n'y a plus d'éléments. Cette structure **for** est typique de python et n'existe pas dans tous les langages. Elle est extrêmement efficace car elle s'arrête sans avoir besoin de prévoir un test de fin.
Un exemple d'algorigramme de la structure **for** est illustré sur la figure 3.
Code python correspondant à l'algorigramme de la figure 3 : en python, cet algorigramme s'écrit en 3 lignes seulement !

```
1 li = [1, 8, 78, 109]
2 for n in li:
3     print(n)
```

Remarque : on pourra utiliser un retardateur (module `time`) pour ralentir l'exécution de la boucle et ainsi mieux en saisir le principe.

```
1 import time
2 li = [1, 8, 78, 109]
3 for n in li:
4     print(n)
5     time.sleep(1) # attendre 1 seconde
```

On observe clairement que la forme python est beaucoup plus simple que l'algorigramme équivalent. C'est une des forces de ce langage de programmation.

Générer une liste d'entiers

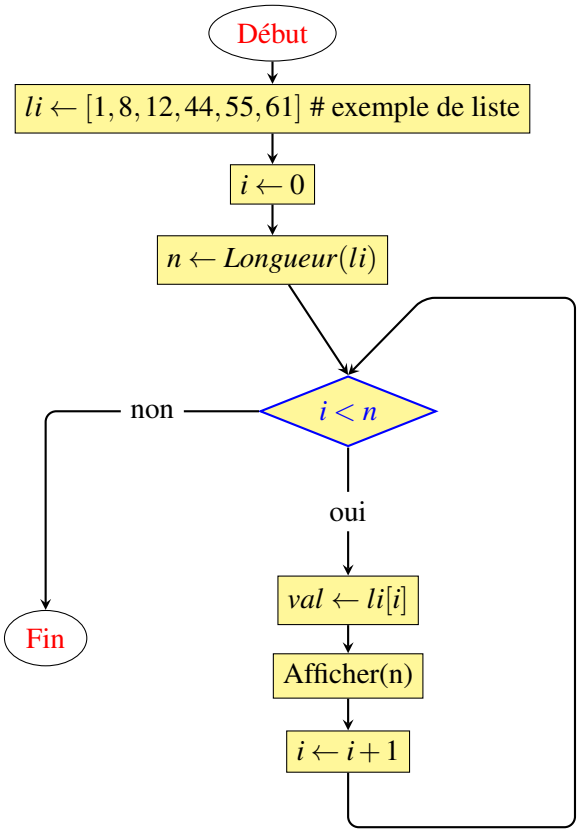

FIGURE 3 – Algorigramme de la boucle ”for”. Affichage des nombres contenus dans une liste.

La fonction **range** permet de générer une liste immuable [4] d'entiers. Cette fonction est particulièrement utile pour les boucles de type **for**.

```
li = range(5)
print(li)
```

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

```
for i in range(5):
    print(i) # affiche les entiers de 0 a 4
```

La boucle **for** est utilisée chaque fois que le nombre d'itérations est connu à l'avance.

11.2 La boucle “tant que”

La boucle **while** est utilisée pour répéter une opération plusieurs fois mais sans connaître au départ le nombre de fois (c'est à dire que la boucle sera stoppée en fonction d'un critère qui dépend des opérations effectuées mais qui n'est pas connu a priori).
Les lignes qui suivent illustrent comment utiliser une boucle **while** pour afficher les entiers compris entre 0 et 10 (fig 4).

```
i = 0
while i <= 10:
    print(i)
    i = i+1
```

4. signifie que cette liste ne peut pas être modifiée. Si la liste générée doit être modifiée, il faut la transformer en liste en utilisant `list(range(1000))`

Il est à noté que la variable *i* doit être augmentée de 1 à chaque itération, sans quoi le programme ne pourrait pas quitter la boucle **while** (entrainant un bug).

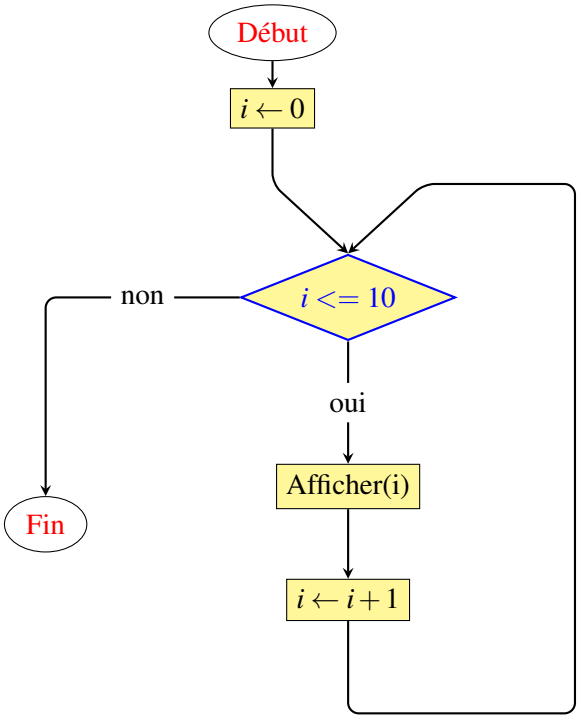

FIGURE 4 – Algorigramme de la boucle “while”. Il est à noter que les boucles “for” et “while” ont la même représentation.

11.3 Les déclarations break et continue

Le mot clé **break** permet de stopper définitivement le processus itératif.

```
1 i = 0
2 while True: # ne s'arrete jamais !!
3     print(i)
4     if i >=10:
5         break
6     i = i+1
```

L’exemple ci-dessus est équivalent au précédent.
Le mot clé **continue** permet de passer à l’itération suivante en stoppant l’itération en cours.

```
1 li = [1, 6, 4, 0, 8]
2 produit = 1
3 for i in li:
4   if i == 0:
5     continue # évite de multiplier par 0
6   produit = produit*i
7 print(produit)
```

12 Lecture d’un fichier de données

Il peut s’agir d’ouvrir une ressource de type fichier csv ou txt.

```
1 with open('file.txt', "r") as f:
2     data = f.read() # data contient l'ensemble du fichier
3     print(data)
```

L'instruction de la ligne 2 lit le fichier et place son contenu dans la variable `data` sous la forme d'une seule chaine de caractères (contenant donc l'ensemble des lignes). Il est souvent préférable d'utiliser la méthodes `readlines()` qui place chaque ligne du fichier comme chaine de caractères dans une liste.

```
1 with open('file.txt', "r") as f:
2     li = f.readlines() # li est une liste
3     print(li)
```