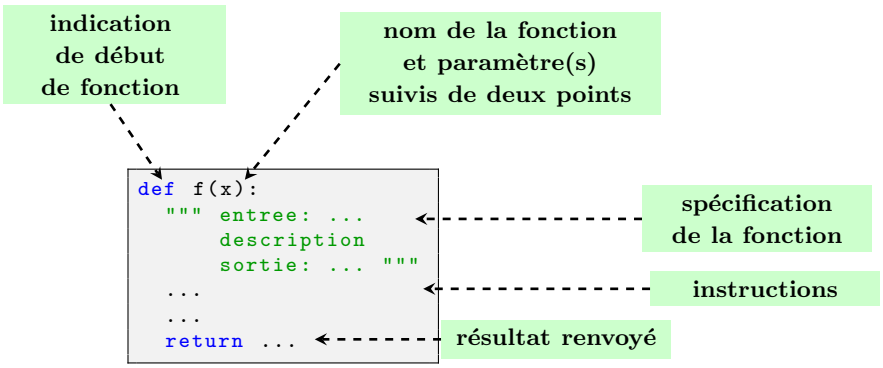
SÉANCE 1

QUELQUES RAPPELS DE PYTHON

I - Syntaxe des instructions de base

On rappelle que l'on définit une ***fonction*** de la façon suivante :



Le mot clé `return` peut apparaître plusieurs fois mais dès qu'il est «rencontré» une fois, cela met un terme à l'exécution des instructions qui suivent. Une fonction peut également agir sur les paramètres (on parle d'*effet de bord*) voire à ne faire que cela et ne rien renvoyer (on dit alors souvent que cette fonction est une *procédure*).

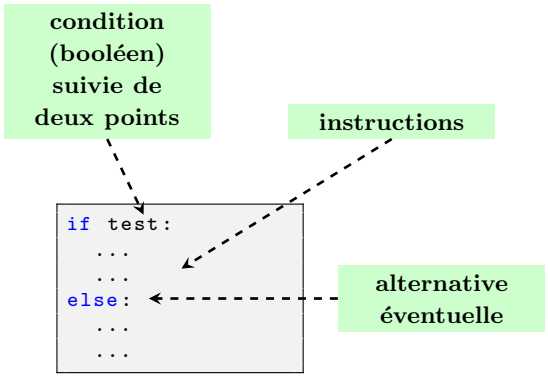
Exemple

Définissons en Python les fonctions $f : \mathbb{R} \to \mathbb{R}, x \mapsto x^2 + x - 1$ et $g : \mathbb{R} \to \mathbb{R}, x \mapsto e^{x+1}$.

Exemple

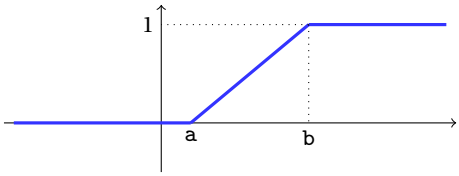
Définissons en Python une fonction qui prend en argument une chaîne de caractères et qui renvoie la valeur du dernier caractère.

Il convient de maîtriser les structures conditionnelles à l'aide de la syntaxe suivante :

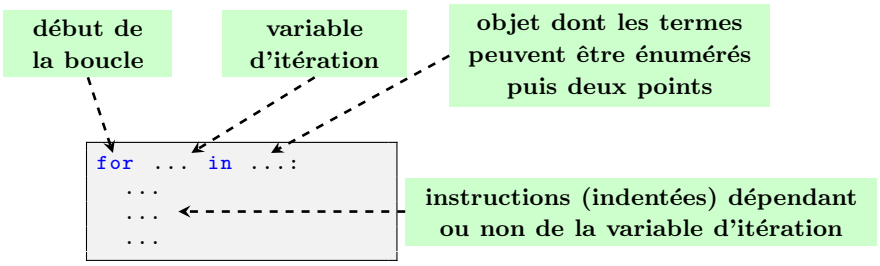


Exemple

Définissons la fonction `f` d'arguments `a`, `b` (avec `a` < `b`) et `x` et qui renvoie l'image de `x` par la fonction représentée ci-dessous :

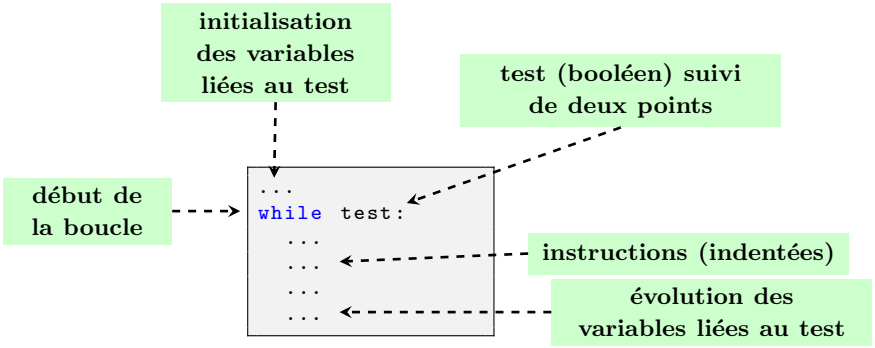


Il existe en Python deux types de structures itératives. Un premier type de ***boucle*** concerne le cas où le nombre d'itérations est déterminé et dépend de l'ensemble qui est décrit par la variable d'itération. La syntaxe générale est de la forme suivante :



L'objet dont les éléments sont énumérables peut être de la forme `range(n)` (ce qui correspond aux entiers de `0` à `n-1`), une chaîne de caractères, une liste, etc.

L'autre type de boucle a un nombre d'itérations qui n'est pas déterminé à l'avance ; la poursuite d'une telle boucle est liée à la réalisation d'une condition. La syntaxe générale est de la forme suivante :



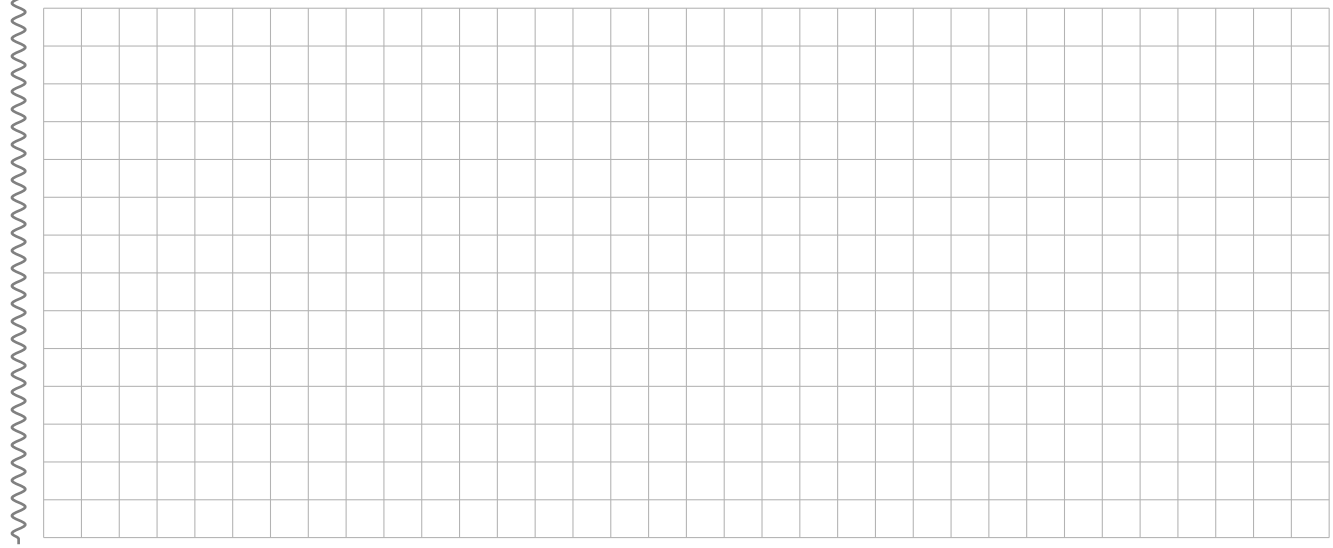
La question de la *terminaison* se pose alors (la boucle est-elle «infinie» ou non ?).

Exemple

Écrivons une fonction d'argument un entier naturel `n` et qui renvoie la somme des `n` premiers nombres impairs.

Exemple

Écrivons une fonction d'argument un entier naturel `n` et qui renvoie la plus grande puissance de 2 inférieure ou égale à `n`.



II - Rappels sur les listes

Une liste est un ensemble ordonné (chaque élément à un numéro d'ordre) d'éléments (de types hétérogènes). Elles sont définies entre crochets et les éléments sont séparés par une virgule. Voici, au travers d'exemples, quelques façons de définir des listes :

```
>>> L = [1, 2, 3] # définition directe
>>> L = list(range(5)); print(L) # définition à partir d'un objet itérable
[0, 1, 2, 3, 4]
>>> L = list(range(4)) + [1, 5, 42]; print(L) # par concaténation de listes
[0, 1, 2, 3, 1, 5, 42]
>>> L = [0]*5; print(L) # par concaténation de copies d'une même liste
[0, 0, 0, 0, 0]
>>> L1 = [2*k+1 for k in range(5)]; print(L1) # en compréhension
[1, 3, 5, 7, 9]
>>> L2 = [k**2 for k in L1 if k>4]; print(L2) # en compréhension avec condition
[25, 49, 81]
```

On peut directement accéder à tous les éléments de la liste et les modifier :

```
>>> L=[2, 5, 42, 6, 3, -2, 4, 6, 8, 9, 12]
>>> L[2]; L[-3]
42
8
>>> L[1:5] # sous-liste de l'élément n°1 à l'élément n°5 non compris
[5, 42, 6, 3]
>>> L[1:10:2] # idem avec un pas
[5, 6, -2, 6, 9]
>>> L[2:] # de l'indice 2 jusqu'à la fin
[42, 6, 3, -2, 4, 6, 8, 9, 12]
>>> L[2::3] # idem avec un pas
[42, -2, 8]
>>> L[:8] # du début jusqu'à l'indice 8 non compris
[2, 5, 42, 6, 3, -2, 4, 6]
>>> L[:8:2] # idem avec un pas
[2, 42, 3, 4]
>>> L[:-2:3] # fonctionne également avec des n° négatifs
[2, 6, 4]
>>> L[0] = 76; L # remplacement d'une valeur
[76, 5, 42, 6, 3, -2, 4, 6, 8, 9, 12]
>>> L[1:6] = [0]; L # remplacement d'une tranche
[76, 0, 4, 6, 8, 9, 12]
```

Rappelons quelques opérations et méthodes sur les listes :

```
>>> L=[1, 6, 9, 23, 21, 0, 3]
>>> len(L) # longueur de la liste
7
>>> L.append(5); L # ajout d'une valeur en tête de liste
[1, 6, 9, 23, 21, 0, 3, 5]
>>> L.append(8); L
[1, 6, 9, 23, 21, 0, 3, 5, 8]
>>> L.insert(2, 8); L # insertion d'une valeur à un emplacement spécifié
[1, 6, 8, 9, 23, 21, 0, 3, 5, 8]
>>> L.pop() # retourne et supprime la tête de liste
8
>>> L
[1, 6, 8, 9, 23, 21, 0, 3, 5]
>>> L.pop(5) # idem avec un emplacement spécifié
21
>>> L
[1, 6, 8, 9, 23, 0, 3, 5]
```

Enfin, rappelons le «danger» (et la source d'erreurs !) résidant dans la copie d'une liste :

```
>>> from copy import copy
>>> a=[1, 2, 3, 4, 5]
>>> b = a; c = copy(a)
>>> a[0] = 42; a; b; c
[42, 2, 3, 4, 5]
[42, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
```

III - Quelques situations classiques

1. Écrire une fonction `appartient(a, L)` renvoyant un booléen indiquant si l'élément `a` appartient à la liste `L`.

2. Modifier la fonction précédente pour traiter le cas où `L` est une liste de nombres triés dans l'ordre croissant.

3. Écrire une fonction `seuil(L, x)` renvoyant la liste formée par les éléments de la liste `L` supérieurs ou égaux à `x`.

4. Écrire une fonction `extremes(L)` renvoyant le couple formé du minimum et du maximum d'une liste `L` de nombres.

5. Écrire une fonction `indmax(L)` renvoyant l'indice de la dernière occurence du maximum d'une liste `L` de nombres.

6. Écrire une fonction `nombre(lis, x, n)`, d'argument une liste `lis`, un objet `x` et un entier `n`, qui renvoie `True` si l'objet `x` apparaît exactement `n` fois dans `lis` (et qui renvoie `False` sinon).

7. Écrire une fonction `deuxieme(lis)`, d'argument une liste d'entiers `lis` ayant au moins 2 éléments et dont tous les «éléments» sont différents, qui renvoie le deuxième plus grand entier de la liste.

8. Écrire une fonction `distmin(lis)`, d'argument une liste de flottants `lis`, qui renvoie le plus petit écart entre deux éléments distincts de la liste.