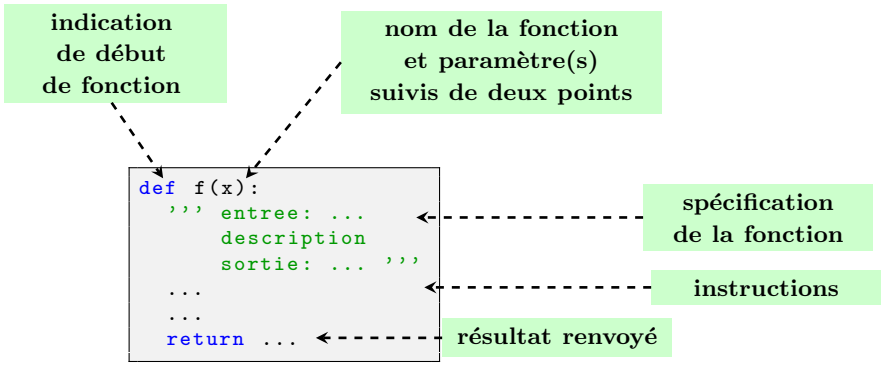
Rappels de python : fonctions, tests, boucles, listes

L'enseignement d'informatique en filière ECG se place dans la continuité de celui du secondaire. Néanmoins, nous rappelons quelques éléments de syntaxe dans ce premier document et les premières séances serviront à réviser ces notions.

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*).

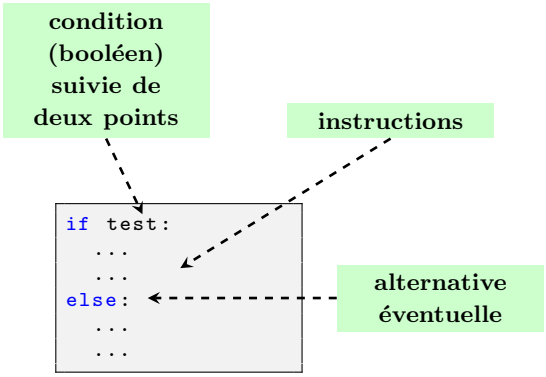
Exercice 1 (écriture de fonctions simples)

1. Écrire une fonction `moyenne` calculant la moyenne de trois nombres.
2. Écrire une fonction `somme_produit` renvoyant le couple formé par la somme et le produit de deux nombres.
3. On rappelle que la température t en degrés Fahrenheit correspond à la température $(t-32)\times\frac{5}{9}$ en degrés Celsius. Écrire deux fonctions `F_vers_C` et `C_vers_F` effectuant les conversions d'un système dans un autre.
4. La formule donnant le volume d'un tétraèdre dont les cotés ont des longueurs a, b, c, d, e et f est :

$$\mathrm{V} = \frac{1}{12}\sqrt{p-q+r}$$

où $p = 4a^2b^2c^2$, $q = a^2y^2 + b^2z^2 + c^2x^2$, $r = xyz$, $x = a^2+b^2-d^2$, $y = b^2+c^2-e^2$ et $z = a^2+c^2-f^2$. Écrire une fonction `volume` qui renvoie le volume d'un tétraèdre de côtés donnés.

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



Exercice 2

1. Écrire une fonction `f` correspondant à la fonction mathématique f donnée par :

$$f(x) = \begin{cases} 1 & \text{si } 0 \leqslant x \leqslant 1 \\ 0 & \text{sinon} \end{cases}.$$

2. Écrire une fonction `trinome` qui prend comme arguments trois nombres a, b et c et qui renvoie le nombre de solutions de l'équation $ax^2 + bx + c$.
3. Écrire une fonction `voyelle` qui prend en argument une chaîne de caractères et qui indique par un booléen si cette chaîne commence par une voyelle.

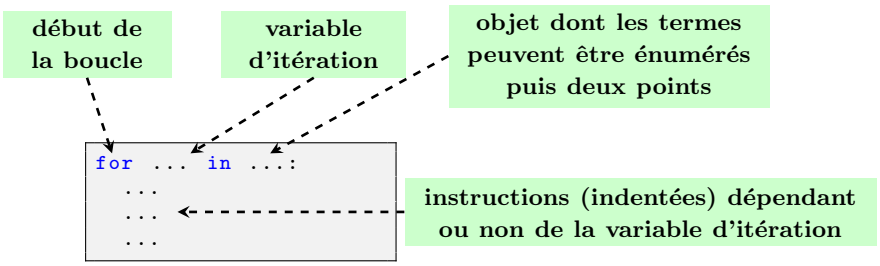
Exercice 3 (conception d'un jeu de test)

Considérons la fonction suivante :

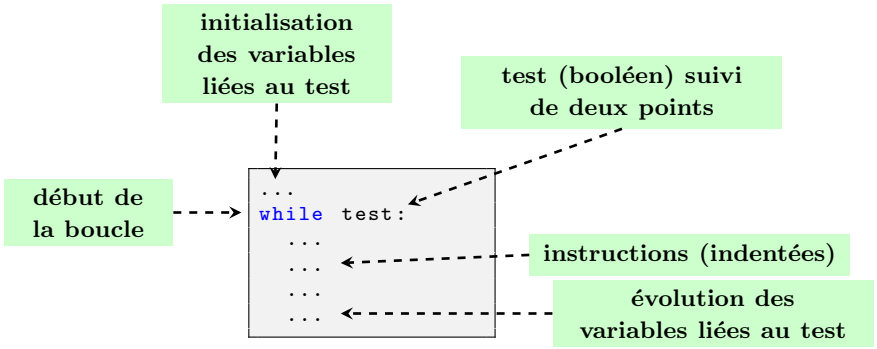
```
def f(n1,n2,n3):
    ''' entrée: trois nombres (float) n1, n2 et n3
        précondition: n1 != n2 and n2 != n3 and n3 != n1
        renvoie un numéro de cas parmi 6 selon les valeurs de n1, n2 et n3
        sortie: chaîne de caractère '''
    if n1 < n2 and n2 < n3:
        return 'cas 1'
    elif n1 < n3 and n3 < n2:
        return 'cas 2'
    elif n2 < n1 and n1 < n3:
        return 'cas 3'
    elif n2 < n3 and n3 < n1:
        return 'cas 4'
    elif n3 < n1 and n1 < n2:
        return 'cas 5'
    else:
        return 'cas 6'
```

1. Concevoir un *jeu de tests* pour cette fonction c'est-à-dire une suite d'exemples permettant de vérifier tous les cas possibles.
2. Écrire une autre version de cette fonction sans les opérateurs `and` et `or` et la vérifier avec le même jeu de tests.

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'itération 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* (la boucle est-elle «infinie» ou non ?) sera étudiée dans la suite de l'année.

Exercice 4 (manipulation de boucles simples)

1. Écrire une fonction d'argument un entier `n` et renvoyant la somme des `n` premiers nombres impairs.
2. Écrire une fonction d'argument un entier `n` et renvoyant la somme des nombres impairs inférieurs ou égaux à `n`.
3. Écrire une fonction d'argument un entier `n` et renvoyant la valeur de factorielle de `n`.
4. Écrire une fonction d'argument un entier `n` et renvoyant la plus grande puissance de 2 inférieure ou égale à `n`.
5. Écrire une fonction d'argument un entier `n` supérieur ou égal à 2 et qui indique par un booléen si ce nombre est nombre premier.
6. Écrire une fonction d'argument une chaîne de caractères et renvoyant la longueur de cette chaîne de caractères (sans utiliser `len`).
7. Écrire une fonction d'argument une chaîne de caractères et renvoyant le nombre de voyelles (non accentuées) de cette chaîne de caractères.
8. Écrire une fonction d'argument une chaîne de caractères et renvoyant un booléen indiquant s'il s'agit ou non d'un palindrome.

Exercice 5 (suite de Syracuse)

Pour tout entier naturel a, on peut considérer la suite $(u_n)_{n\in\mathbb{N}}$ donnée par $u_0 = a$ et pour tout $n \in \mathbb{N}$:

$$u_{n+1} = \begin{cases} \frac{1}{2}u_n & \text{si } u_n \text{ est pair} \\ 3u_n + 1 & \text{si } u_n \text{ est impair} \end{cases}.$$

1. Écrire une fonction `suite` d'arguments deux entiers naturels a et n et qui renvoie le terme u_n de la suite correspondante.
2. Écrire une fonction `attente` d'argument un entier naturel a et qui renvoie le plus petit entier n tel que $u_n = 1$ (pour la suite correspondante) ou qui renvoie -1 si l'on n'a toujours pas rencontré la valeur 1 au bout de 1000 termes de la suite.

Dans la suite de cette séance, nous allons manipuler des tableaux. La structure que nous considérons pour l'instant est celle de type `list` que l'on désigne en parlant de *liste*.
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
>>> list(range(5))   # définition à partir d'un objet itérable
[0, 1, 2, 3, 4]
>>> list(range(4))+[1,5,42]   # par concaténation de listes
[0, 1, 2, 3, 1, 5, 42]
>>> [0]*5   # par concaténation de copies d'une même liste
[0, 0, 0, 0, 0]
>>> [2*k+1 for k in range(5)]   # en compréhension
[1, 3, 5, 7, 9]
```

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] # de l'indice 1 à l'indice 10 non compris avec un pas de 2
[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]
```

Pour le moment, nous n'aurons besoin que de ce qui suit :

```
>>> L=[1,6,9,23,21,0,3]
>>> len(L)  # longueur de la liste
7
>>> L.append(5); L.append(8); L   # ajout d'une valeur
[1, 6, 9, 23, 21, 0, 3, 5, 8]
```

Exercice 6

Écrire une fonction :

1. d'argument `n` et qui renvoie la liste formée par `n` zéros ;
2. d'argument `n` et qui renvoie la liste des `n` premiers carrés ;
3. d'argument `n` et qui renvoie la liste des `n` premiers carrés pairs ;
4. d'argument `n` et qui renvoie la liste des couples (i, j) avec $0 \leqslant i < n$ et $0 \leqslant j < n$;
5. d'argument `n` et qui renvoie la liste des couples (i, j) avec $0 \leqslant i < j < n$;
6. d'argument une liste d'entiers `L` et qui renvoie la liste formée par les termes d'indices pairs de `L` ;
7. d'argument une liste d'entiers `L` et qui renvoie la liste formée par les valeurs positives de `L` ;
8. d'argument une chaîne de caractères `c` et qui renvoie la liste formée par les indices pour lesquels il y a un `e` dans `c` ;
9. d'argument une chaîne de caractères `c` et qui renvoie la liste formée du nombre de consonnes, du nombre de voyelles et du nombre d'autres caractères de `c` (on pourra supposer qu'il n'y a pas de lettre accentuée) ;
10. d'argument une chaîne de caractères `c` et qui renvoie la liste formée par tous les suffixes de `c`.

Exercice 7

Écrire une fonction `suite` d'arguments une fonction f, un nombre a et un entier n, et qui renvoie la liste des termes $u_0, \ldots, u_n$ de la suite u donnée par :

$$u_0 = a \text{ et } \forall n \in \mathbb{N},\ u_{n+1} = f(u_n).$$

Pour tester cette fonction, on peut importer le module `math` à l'aide de l'instruction :

```
>>> import math
```

Exercice 8

Écrire une fonction `distance` prenant en argument une liste de nombres (de type `float`) qui calcule la somme des écarts entre les éléments de la liste et leur moyenne, c'est-à-dire que si la liste contient les nombres $x_1, \ldots, x_k$ alors la fonction doit renvoyant la valeur de :

$$\frac{1}{k}\sum_{j=1}^{k} \left|x_j - \overline{x}\right| \quad \text{où } \overline{x} \text{ est la moyenne de } x_1, \ldots, x_k.$$

Exercice 9

Écrire une fonction `separe` prenant en argument une liste de nombres entiers et qui renvoie un couple de listes : la première formée par les nombres pairs de la liste initiale et la seconde formée par les nombres impairs.
Par exemple, on doit obtenir :

```
>>> separe([2,3,5,8,7,9,9,4,11,-2,-1])
([2, 8, 4, -2], [3, 5, 7, 9, 9, 11, -1])
>>> separe([3,1,3,5,7])
([], [3, 1, 3, 5, 7])
```

Exercice 10 (recherche d'un maximum)

Considérons la fonction suivante :

```
1 def recherche(L):
2     ''' entrée : L liste (non vide) de nombres
3         sortie : valeur maximale de L (nombre) '''
4     a = L[0]
5     for k in range(len(L)):
6         if L[k] > a:
7             a = L[k]
8     return a
```

1. Expliquer la ligne 4. Peut-on la remplacer par `a=0` ?
2. Que se passe-t-il si, à la ligne 6, on remplace `L[k]>a` par `L[k]>=a` ?
3. Proposer des modifications de la fonction ci-dessus pour obtenir une fonction renvoyant :
   a. le couple formé par le minimum et le maximum des valeurs de `L` ;
   b. le couple formé par le maximum de `L` et le nombre d'occurrences de ce maximum ;
   c. le couple formé par le maximum de `L` et la liste des indices de `L` en lesquels il y a cette valeur maximale ;
   d. la deuxième plus grande valeur de `L` (ou la plus grande si cette dernière est présente au moins deux fois).

Exercice 11 (présence d'une valeur dans un tableau)

1. Écrire une fonction `presence` prenant en argument un tableau d'entiers et un entier et renvoyant un booléen indiquant si cet entier apparaît dans le tableau (sans utiliser `in`).
   Par exemple, on doit obtenir :

```
>>> presence([1,2,3,4,5],3)
True
>>> presence([1,2,3,4,5],6)
False
```

2. Optimiser la fonction précédente dans le cas où le tableau d'entiers est trié dans l'ordre croissant.
3. Écrire une fonction `occurrences` prenant en argument un tableau d'entiers et un entier et renvoyant le nombre d'occurrences de cet entier dans le tableau.
   Par exemple, on doit obtenir :

```
>>> occurrences([1,2,3,4,5],1)
1
>>> occurrences([1,2,3,1,1],1)
3
>>> occurrences([1,2,3,1,5],6)
0
```

Exercice 12 (listes d'intervalles)

Dans cet exercice, un segment $[a, b]$ (avec $a \leqslant b$) est représenté par la liste : `[a,b]`.

1. Deux segments sont disjoints si leur intersection est vide. Écrire une fonction `disjoints` de deux arguments `i1` et `i2` qui teste si les segments `i1` et `i2` sont disjoints. La fonction renvoie le booléen `True` s'ils sont disjoints, `False` sinon.
2. La fusion de deux segments a comme minimum le plus petit des minimums des deux segments, et comme maximum le plus grand des maximums des deux segments. Écrire une fonction `fusion` de deux arguments `i1` et `i2` et qui renvoie le segment correspondant à la fusion de `i1` et `i2`.

3. Une « *liste bien formée* » est une liste de segments qui vérifie les propriétés suivantes :

- les segments sont deux à deux disjoints ;
- les segments de la liste sont classés par ordre croissant, en considérant qu'un segment `i1` est strictement plus petit qu'un segment `i2` si et seulement si le maximum de `i1` est strictement inférieur au minimum de `i2`.

a. Les listes suivantes sont-elles bien formées ?

- `L1=[[0,1],[2,5],[3,6]]`
- `L2=[[2,5],[0,1],[3,6]]`
- `L3=[[0,1],[2,3],[4,6]]`

b. Écrire une fonction `verifie` qui teste si une liste de segments est bien formée.

4. Écrire une fonction `appartient` de deux arguments `x` et `L` qui teste si la valeur `x` est élément d'un des segments de la liste `L` bien formée.

Exercice 13 (fusion de deux listes)

Considérons la fonction suivante :

```
def fusion(L1,L2):
    ''' entrée : L1 et L2 listes d'entiers triées par ordre croissant
        ----> il se passe quelque chose à expliciter !
        sortie : une liste L d'entiers '''
    L = []
    i,j = 0,0
    while i < len(L1) and j < len(L2):
        if L1[i] < L2[j]:
            L.append(L1[i])
            i = i+1
        else:
            L.append(L2[j])
            j = j+1
    return L
```

1. Pourquoi ce nom `fusion` ?

2. Déterminer (sur papier ou de tête !) ce que donnent les instructions suivantes (avant de vérifier !) :

```
>>> fusion([1,5,9],[3,4,8,12,15,21,42])
>>> fusion([1,5,9],[-2])
>>> fusion([1,5,9],[1])
>>> fusion([1],[1,5,9])
>>> fusion([1,5,9],[])
```

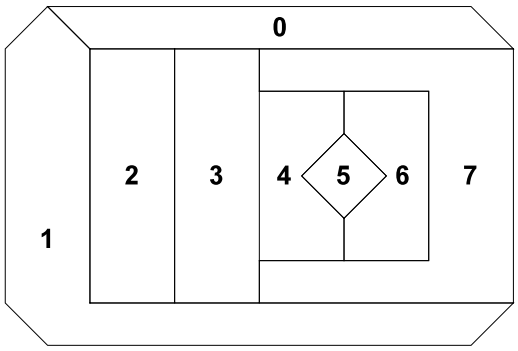
3. Modifier la fonction précédente afin qu'elle renvoie la liste triée constituée par la réunion de tous les éléments présents dans les deux listes `L1` et `L2` en arguments.

Par exemple, on souhaite obtenir (en appelant `fusionbis` cette nouvelle fonction) :

```
>>> fusionbis([1,5,9],[3,4,8,12,15,21,42])
[1, 3, 4, 5, 8, 9, 12, 15, 21, 42]
>>> fusionbis([1],[1,5,9])
[1, 1, 5, 9]
>>> fusionbis([1,5,9],[1])
[1, 1, 5, 9]
>>> fusionbis([1,5,9],[])
[1, 5, 9]
```

Exercice 14 (coloriage de graphe)

On cherche à colorer la carte suivante, comportant 8 pays numérotés de 0 à 7, avec comme seul contrainte que deux pays ayant une frontière commune ne peuvent être de la même couleur. Comme les pays, les couleurs sont numérotées à partir de zéro.



1. Définir la liste `carte` de taille 7 telle que l'élément `carte[i]` soit lui-même une liste : la liste des numéros supérieurs à i des pays ayant une frontière commune avec le pays numéro i. Par exemple `carte[3]` est la liste `[4,7]` (0, 1 et 2 n'apparaissent pas puisque inférieurs à 3). Noter que la liste `carte` contient un élément de moins que le nombre de pays puisque le pays de numéro le plus grand ne peut pas avoir de voisin de numéro supérieur.

2. Écrire puis tester une fonction `voisins` de trois arguments, deux nombres entiers distincts i et j et une liste `carte` caractérisant les frontières communes d'un ensemble de pays, qui renvoie `True` si les pays numérotés i et j ont une frontière commune, et `False` sinon.

3. L'attribution des couleurs à chaque pays est caractérisée par une liste `colors` : `colors[`p`]` est la couleur attribuée au pays p ; `colors[`p`]` vaut -1 si la couleur n'est pas encore attribuée. La liste `colors` ne contient que des -1 au départ et ses valeurs sont modifiées progressivement au fur et à mesure que les couleurs sont attribuées.

Définir une fonction `colorePays` de trois arguments, la liste `colors` des couleurs attribuées, le numéro p du pays à colorer et la liste `carte` caractérisant les frontières. Cette fonction ne renvoie rien mais modifie la liste `colors` en donnant à `colors[`p`]` la plus petite couleur possible, en fonction des couleurs des pays voisins qui sont déjà colorés.

4. Écrire une fonction `colorer1`, d'argument une liste `carte` caractérisant les frontières communes d'un ensemble de pays, renvoyant la liste `colors` des couleurs attribuées, en colorant les pays un par un par ordre croissant de leurs numéros.

Tester cette fonction sur l'exemple et faire afficher les couleurs utilisées ainsi que leur nombre.

5. Écrire une fonction `colorer2`, analogue à `colorer1` et prenant un argument supplémentaire : une liste `ordre` fixant l'ordre de coloration des pays. Cet ordre semble-t-il avoir une influence sur le nombre de couleurs utilisées ? Pour le cas de figure traité ici, quel est le nombre minimum de couleurs utilisées ?