# Les structures de boucles

Python dispose de deux structures permettant de faire des boucles : une lorsque le nombre d'itérations est connu à l'avance et une autre lorsque la boucle doit s'arrêter grâce à une condition.

La structure *boucle avec compteur* s'utilise avec la syntaxe suivante :

```
for compteur in iterateur:
    # Instructions
```

et la structure *Tant que* selon la syntaxe :

```
while condition:
    # Instructions
```

A nouveau, l'indentation est syntaxique. Nous allons à présent donner des exemples d'utilisation de ces deux structures. On peut en particulier noter que la fin d'une structure de boucle se fait en supprimant une indentation. Il n'y a pas besoin de symbole particulier (parenthèse, accolade ou autres). C'est l'indentation qui indique le début et la fin de la boucle.

*NB* : n'oubliez pas le `:` qui est indispensable à la fin de la ligne débutant la boucle...

# Boucle for

Voici la synthaxe de la boucle *pour* :

```
for ma_variable in objet_iterable:
    # Instructions
```

- l'indentation est toujours synthaxique,
- il ne faut pas oublier les `:` ,
- les `objets_iterables` classiques que nous utiliserons le plus souvent sont :
  - une liste,
  - l'itérateur `range` (ex : `for i in range(10):` ),
  - un tableau `numpy` (ex : `for i in np.linspace(0,10):` ), que nous verrons plus tard dans le cours sur le module `numpy` .

**Remarque :** l'intérêt des itérateurs est qu'il n'y a pas de stockage de tous les éléments de la liste mais seulement de l'élément courant (gros gain de place mémoire).

## parcours des éléments d'une liste

Voici un premier exemple de parcours des éléments d'une liste.

In [ ]:
```
# Boucle sur les éléments d'une liste
jours = ['lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi', 'samedi
', 'dimanche']
```

In [ ]:
```
for jour in jours :
    print(f"{jour} est un des jours de la semaine !")
```

Si l'on souhaite également avoir un compteur pour afficher le numéro du jour de la semaine, il y a plusieurs façons de faire.

In [ ]:
```
numero = 0
for jour in jours:
    numero += 1
    print(f"{jour} est le jour numéro {numero} dans la semaine !")
```

In [ ]:
```
jours = ['lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi', 'samedi
', 'dimanche']
print(list(enumerate(jours)))
for numero, jour in enumerate(jours, start=1):
    print(f"{jour} est le jour numéro {numero} dans la semaine !")
```

In [ ]:
```
help(enumerate)
```

## boucle à l'aide d'un itérateur

L'itérateur classique que nous utiliserons toujours est `range`

Il s'utilise de plusieurs façons selon les arguments qui lui sont passés :

```
# boucle de 0 à entier_fin - 1
for k in range(entier_fin):
    # Instructions

# boucle de entier_debut à entier_fin - 1
for k in range(entier_debut, entier_fin):
    # Instructions

# boucle de entier_debut à entier_fin - 1 par pas de entier_pas
for k in range(entier_debut, entier_fin, pas):
    # Instructions
```

In [ ]:
```
help(range)
```

In [ ]:
```
entier_debut, entier_fin, entier_pas = -10, 10, 2
```

In [ ]:
```
print(f"boucle sur range({entier_fin}) : ")
for k in range(entier_fin):
    print(f"{k:2d} ", end='')
```

In [ ]:
```
print(f"boucle sur range({entier_debut}, {entier_fin}) : ")
for k in range(entier_debut, entier_fin):
    print(f"{k:2d} ", end='')
```

In [ ]:
```
print(f"boucle sur range({entier_debut}, {entier_fin}, {entier_pas}
) : ")
for k in range(entier_debut, entier_fin, entier_pas):
    print(f"{k:2d} ", end='')
```

**Exercice**

1. Calculez à l'aide d'une boucle la somme de tous les entiers entre $0$ et $100$.
2. Calculez le produit de tous les nombres impairs compris entre $0$ et $100$.

In [ ]:
```
S, n = 0, 100
for k in range(n + 1):
    S += k
print(f"La somme des entiers compris entre 0 et {n} vaut {S}")
```

In [ ]:
```
P, n = 1, 100
for k in range(1, n + 1, 2):
    P *= k
print(f"Le produit des entiers impairs compris entre 0 et {n} vaut
{P}")
```

# Boucle while

Une boucle `while` est utilisée lorsque le nombre d'itérations n'est pas connu à l'avance : on arrête la boucle quand une certaine condition est satisfaite. La syntaxe est la suivante

```
while condition:
    # itérations
```

Dans ce cas, la boucle est exécutée tant que la condition est vérifiée (retourne le booléen `True`).

*Attention* : il est très facile de fabriquer une boucle infinie avec la commande `while`. On ajoute donc souvent un compteur de sécurité pour éviter ce problème.

Voici un exemple qui permet de calculer la plus grande puissance de $2$ (négative) notée $2^{-k}$ telle que $1 + 2^{-k} = 1$. Ce résultat n'est pas possible dans le monde des mathématiques mais évident dans le monde numérique...

In [ ]:
```
k, kmax = 0, 10
pk = 1
while 1+pk != 1 and k < kmax:
    k += 1
    pk /= 2
if k == kmax:
    print("Critère d'arrêt atteint !")
print(f"1+2^({-k}) = {1+pk}")
```

In [ ]:
```
def mon_pgcd(a, b):
    """Calcule le pgcd de a et b via l'algorithme d'Euclide"""
    if type(a) != int or type(b) != int:
        print("les arguments ne sont pas des entiers")
    while b != 0:
        a, b = b, a % b
    return a

print(mon_pgcd(10, 5))
```

### Quand faut-il utilisez une boucle `while` ?

- pour une récurrence complexe,
- ou lorsque le nombre d'itérations n'est pas connu à l'avance.

Généralement, une fonction récursive peut être remplacée par une boucle `while` et vis-versa. Dans la plupart des cas, la fonction récursive sera plus élégante mais généralement plus lente. *Remarque : dans certains langages compilés (C++ entre autre), la différence de vitesse d'exécution tant à disparaître.*

**Exercice : le nombre d'or**

Le nombre d'or $\phi$ vérifie :

$$\phi = 1 + \cfrac{1}{1 + \cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1+\dots}}}}}$$

Comme $1 > 1/(1+x) > 1/2$ pour tout $x \in ]0;1[$, on en déduit l'encadrement :

$$\begin{cases} u_n &= 1 + \underbrace{\left(1+\left(1+\left(1+\dots 1\right)^{-1}\dots\right)^{-1}\right)^{-1}}_{n \text{ fois}} \\ v_n &= 1 + \underbrace{\left(1+\left(1+\left(1+\dots 2\right)^{-1}\dots\right)^{-1}\right)^{-1}}_{n \text{ fois}} \\ & \min(u_n, v_n) < \phi < max(u_n, v_n) \end{cases}$$

On a donc les relations de récurrences $u_{n+1} = 1 + 1/u_n$ et $v_{n+1} = 1 + 1/v_n$ avec $u_0 = 1$ et $v_0 = 2$.

> Codez une fonction `approx_nb_or` qui prend un entier $n$ et renvoie une approximation du nombre d'or avec $n$ chiffres significatifs derrière la virgule.

In [10]:
```
from math import sqrt
n = 10
epsilon = 10**(-n)
u, v = 1, 2
n, nmax = 0, 100
while abs(u-v) > 2*epsilon and n < nmax:
    u = 1 + 1/u
    v = 1 + 1/v
    n += 1
phi = .5*(1+sqrt(5))  # solution de X^2-X-1=0
phi_num = .5*(u+v)
print(f"Estimation du nombre d'or à {epsilon:10.3E} près : {phi_num
:10.7f} (erreur = {phi-phi_num:10.3E})")
if n == nmax:
    print(f"Attention : convergence peut-être pas atteinte !")
```

```
Estimation du nombre d'or à  1.000E-10 près :  1.6180340 (erreur =
2.455E-11)
```

**Exercice : deviner un nombre**

> Générer un code qui choisi un entier aléatoire entre 0 et 100 puis demande à l'utilisateur de trouver ce nombre. Pour cela l'utilisateur entre un nombre, s'il est bon, le code termine, s'il est mauvais, le code indique s'il est plus petit ou plus grand.

In [11]:
```
from random import randint
help(randint)
```

```
Help on method randint in module random:

randint(a, b) method of random.Random instance
    Return random integer in range [a, b], including both end poin
ts.
```

In [15]:
```
n = randint(0, 100)  # nombre aléatoire généré par la machine
p = -1
while p != n:
    p = int(input("Proposez un nombre entre 0 et 100 : "))
    if p > n:
        print(f"Le nombre {p} est trop grand !")
    if p < n:
        print(f"Le nombre {p} est trop petit !")
    if p == n:
        print(f"Gagné {p} == {n}")
```

```
41
Proposez un nombre entre 0 et 100 : 0
Le nombre 0 est trop petit !
Proposez un nombre entre 0 et 100 : 42
Le nombre 42 est trop grand !
Proposez un nombre entre 0 et 100 : 60
Le nombre 60 est trop grand !
Proposez un nombre entre 0 et 100 : 41
Gagné 41 == 41
```

# Correction

**ATTENTION** ne lisez la partie plus bas que si vous n'arrivez pas à répondre aux questions

*Notre correction n'est pas la meilleur, la votre vaut beaucoup plus si vous l'avez comprise !!!*

In [ ]:
```
val_or = 1.6180339887498948482045868343656381177 2
n = 9
u, v, e = 1, 2, 10**(-n)
while (v-u) > e:
    v, u = 1 + 1/u, 1 + 1/v
val_app = .5*(u+v)
print(f"Nombre d'or avec {n} chiffres corrects après la virgule : {
val_app} (erreur = {val_app-val_or:10.3e})")
```

In [ ]:
```
from random import randint
help(randint)
```

In [ ]:
```
N = randint(0, 100)  # nombre aléatoire entre 0 et 100
print(f"Le nombre mystère est ??")
while True:
    n = int(input("Proposez un nombre entre 0 et 100 :"))
    if n < N:
        print(f"{n} est trop petit")
    if n > N:
        print(f"{n} est trop grand")
    if n == N:
        print(f"Bravo le nombre mystère est bien {N} !")
```

In [ ]:
```

```