

# REMISE À NIVEAU PYTHON

*D'après "Python au lycée"  
d'Arnaud Bodin*

HAI108I2

## 1 Mode interactif

### 1.1 Expressions mathématiques

Lancez **python3** dans un terminal. Tapez ensuite les instructions suivantes :

#### Code 1 : Premières instructions

```
5+7
6*7
3*(12+5)
3*1.5
3**2
10**(-3)
14/4
1/3
14//4
14%4
abs(-6)
(1+2j)**2
```

Pour des opérations moins élémentaires, nous pouvons utiliser la librairie **math** :

#### Code 2 : Instructions mathématiques

```
from math import *
pi
ceil(pi)
sin(pi/2)
```

#### Leçon 1 : À retenir

- différence entre représentation entière et représentation réelle,
- différence entre division entière et division réelle,
- les opérateurs de puissance **\*\*** et de modulo **%**, la valeur absolue **abs**,
- la notation des complexes avec un **j**,
- **from <lib> import \*** permet d'importer et d'utiliser toutes les fonctions de la librairie **<lib>**.

#### Exercice 1 : Premiers calculs avec *Python*

1. Combien y-a-t-il de secondes en un siècle ? (sans tenir compte des années bissextiles)
2. Quels sont les trois derniers chiffres de

$$123456789 \times 123456789 \times 123456789 \times 123456789 \times 123456789 \times 123456789$$

## 1.2 Expressions booléennes

Tapez les instructions suivantes et s'en servir pour répondre à l'exercice 2.

### Code 3 : Booléens

```
False or False
False or True
True or False
True or True
False and False
False and True
True and False
True and True
not True
not False
```

### Exercice 2

Donnez les tables de vérités de **et**, **ou**, **non**.

### Exercice 3 : Expressions booléennes

Essayer de prédire la valeur des expressions suivantes et vérifier qu'ils sont justes.

```
(True and False) or True
not (True and (False or True))
True and (not False)
(True and not(True)) or (not(False) and True)
```

## 1.3 Chaînes de caractères

### Code 4 : Chaînes de caractères

```
"in girum imus nocte et consumimur igni"
'fugit irreparable tempus'
'un autre : "dura lex : sed lex"'
len('cinq') # len : renvoie la longueur de la chaîne en parametre)
len("trois")
"" #(chaîne vide)
"" #(chaîne vide)
len("")
len("")
"memento" + ' audere ' + "semper" # + : operateur de concatenation
str(5*8) # conversion en chaîne de caractère
```

### Leçon 2 : Chaîne de caractères

**len(ch)** : renvoie le nombre de caractères la chaîne.

**+** : opérateur de concaténation.

**"** : chaîne vide.

**''** : chaîne vide.

**str(exp)** renvoie la valeur de l'expression convertie en chaîne de caractères.

## 1.4 Type

La fonction `type` retourne le type de la valeur passée en paramètre.

### Code 5 : Commande type

```
type(2)
type(2.2)
type("")
type("citius altius fortius")
type(True)
```

## 1.5 Affichage

Il est possible de faire de l’affichage avec la fonction `print`.

### Code 6 : Commande print

```
print(2 + 5)
print("2*(5+8)=", 2*(5+8))
print("2*(5+8)=" + str(2*(5+8)))
```

### Remarque 1 : f-strings

Depuis *Python 3.6*, les *f-strings* permettent d’insérer des expressions dans des chaînes de caractères en utilisant une syntaxe minimale.

Pour utiliser les *f-strings* il suffit de mettre un "f" devant la chaîne de caractères et pour insérer la valeur d’une expression dans la chaîne il suffit de mettre l’expression entre accolade. Si il n’y a pas d’expression à substituer il n’est pas nécessaire de mettre le "f" devant.

### Code 7 : f-strings

```
nom="Paul"
age=31
print(f"Votre nom est {nom} et vous avez {age} ans")

print(f"abs(-6)={abs(-6)}")
```

Certains caractères ne peuvent pas être affichés tel quel. Il est donc nécessaire de les échapper. Pour les accolades `{}` il faut en mettre deux à la suite par exemple. Les *f-strings* permettent également de contrôler l’affichage en précisant le nombre de caractère à afficher, le format d’affichage, la base....

### Code 8 : Format d’affichage

```
print(f"{2021:012b}") # affichage en binaire, sur 12 chiffres, en complétant par
→ des 0 si besoins
print(f"{2:+.4f}") # affichage sous forme réelle, avec 4 chiffres après la
→ virgule et en forçant l’affichage du signe
print(f"{2**40:_}") # séparation des chiffres par blocs de 3
print(f"{2**40:,}")
print(f"{'test':>10}") # on complète à gauche par un caractère
print(f"{'test':<10}") # à droite
print(f"{'test':^10}") # des deux côtés
```

## 1.6 Variables

### Leçon 3 : Variable

Une **variable** permet de stocker une **valeur**. Son type (réelle, entier, chaîne de caractères,...) est dynamique et dépend de la valeur affectée. Le symbole `=` désigne l'**affectation**. L'affectation a pour but d'**affecter** une valeur à une variable. La variable est placée à gauche du symbole et la valeur est placée à droite. Lorsque qu'une expression est placée à droite, celle-ci est **évaluée** (la valeur associée à l'expression est calculée). La valeur est ensuite affectée à la variable. Aussi. Lorsqu'une nouvelle valeur est affectée à une variable, l'ancienne valeur de la variable est écrasée (perdue). Il n'est pas nécessaire de prédéfinir (déclarer) une variable ; il suffit de la nommer au moment où on en a besoin.

### Code 9 : Variables

```
base = 8
hauteur = 3
aire = base * hauteur / 2
print(aire)
print(Aire) # Erreur
```

Attention ! Python distingue les majuscules et les minuscules.

### Remarque 2 : Variable et types

En python, typage est dynamique. Le type d'une variable, déterminé par Python, peut donc varier au cours du temps. Tester le code suivant et observer bien.

```
a=8
type(a)
a=False
type(a)
a="bonjour"
type(a)
```

### Exercice 4 : Équation du second degré

Nous souhaitons écrire une suite d'instructions permettant de résoudre une équation du second degré de la forme  $aX^2 + bX + c = 0$ . Pour cela, nous utiliserons trois variables  $a$ ,  $b$  et  $c$  ; vous pourrez également définir d'autres variables. Par exemple, pour l'équation  $5X^2 + 4X - 1 = 0$ , nous posons  $a = 5$ ,  $b = 4$  et  $c = -1$ . Nous aimerions avoir l'affichage final sous la forme suivante :

l'equation  $5 X^{**2} + 4 X - 1 = 0$  admet les solutions -1.0 et 0.2

### Exercice 5 : Épargne

Tu places la somme de 100 euros sur un compte d'épargne. Chaque année, les intérêts rapportent 10%. Donne le code qui permet de calculer et d'afficher le capital pour les trois premières années.

La procédure d'échange est une opération qui revient très souvent en informatique.

### Exercice 6 : Échange

Je définis deux variables par  $a=9$  et  $b=11$ . Quelle séquence d'instructions permet l'échange des valeurs de sorte qu'à la fin  $a$  vaut 11 et  $b$  vaut 9 ?

$a=b$	$c=b$	$c=a$	$c=a$
$b=a$	$a=b$	$a=b$	$a=c$
	$b=c$	$b=c$	$c=b$
			$b=c$

### Remarque 3

En *Python*, il est aussi possible d'écrire `a,b=b,a` pour faire l'échange !

## 2 Éditeur

À partir de maintenant, nous allons travailler avec l'éditeur **Visual Studio Code**. Pour avoir un environnement plus agréable et pratique pour l'enseignement, nous allons ajouter quelques extensions :

- Python,
- indent-rainbow.

### Leçon 4 : Input

La méthode `input` permet la saisie par l'utilisateur d'une valeur. Par exemple :

```
a=input()
```

permet la saisie d'une valeur et affecte cette valeur (une chaîne de caractères) à la variable `a`. Si on veut un entier, il faut convertir avec la fonction `int(chaine)`. On peut inclure également un message :

```
a=input("Donnez une valeur pour a :")
```

### Exercice 7 : Retour sur les équations du second degré

Écrire un programme *Python* qui demande à l'utilisateur les valeurs de `a`, `b`, `c` de l'équation  $aX^2 + bX + c = 0$  et affiche les solutions. (On supposera que les valeurs de `a`, `b`, `c` impliquent l'existence de deux solutions réelles distinctes.) Pour cela, ouvrir un fichier `exo6.py` dans *spyder*. Après l'avoir édité et sauvé, vous lancerez l'interprétation du programme. Voici un exemple d'affichage suite à l'interprétation :

```
Entrer les valeurs de a, b, c de l'equation aX**2+bX+C=0
Valeur de a : 5
Valeur de b : 4
Valeur de c : -1
Resolution de (5)X**2 + (4)X + (-1) = 0
Valeur du discriminant : 36
Les solutions sont -1.0 et 0.2
```

### Leçon 5 : Cycle de développement

Le cycle de développement est la séquence d'opérations **éditer**, **sauvegarder**, **compiler**, **exécuter** et on recommence : il est important de réaliser le cycle de développement sans lever les mains du clavier (gain d'efficacité). Pour cela, apprenez petit à petit les raccourcis clavier. Ainsi on réalise le cycle de développement de la manière suivante :

1. on édite dans la partie éditeur,
2. on sauve C-s
3. on exécute par F5
4. on bascule vers le terminal avec C-'
5. on revient à l'éditeur avec C-shift-E et on recommence.

### Exercice 8 : Calculatrice

Écrire un programme qui demande deux valeurs (à stocker dans deux variables) et une opération (parmi `+`, `-`, `*`, `/`) et qui effectue le calcul. On pourra utiliser la méthode `eval`.

#### Remarque 4 : Eval

`eval(expr)`, où `expr` est une chaîne de caractères, interprète l'expression comme une expression python et l'évalue. Ainsi, `eval("2+5")` renverra 7.

## 3 La boucle "pour"

La boucle "pour" est la façon la plus simple de répéter des instructions.

### Leçon 6 : Boucle Pour

Diagram illustrating the syntax of a `for` loop:

- `for` and `in` are reserved words (mots réservés "for" et "in").
- `i` is a variable (une variable).
- `range(n)` is the list to iterate over, here  $0, 1, 2, \dots, n-1$  (liste à parcourir, ici  $0, 1, 2, \dots, n-1$ ).
- `:` is the colon (deux points).
- The indented block of instructions (instructions indentées) will be repeated  $n$  times, for  $i = 0, i = 1, \dots$ , up to  $i = n-1$  (bloc d'instructions indentées, sera répété  $n$  fois, pour  $i = 0, i = 1, \dots$ , jusqu'à  $i = n-1$ ).
- The instructions following the loop (instructions suivantes) continue the program (suite du programme).

Notez bien que ce qui délimite le bloc d'instruction, c'est l'**indentation**, c'est-à-dire les espaces placées en début de ligne qui décalent les lignes vers la droite. Toutes les lignes d'un bloc doivent avoir exactement la même indentation. Note également la présence de ":" en fin de ligne de la déclaration du `for` !

### Code 10 : Boucle Pour

```
for i in range(10):  
    print(i*i)
```

### Code 11 : Somme

```
somme=0  
for i in range(20):  
    somme=somme+i  
print(somme)
```

### Leçon 7 : range

`range(n)` : énumère les entiers de 0 à  $n-1$ ,  
`range(a,b)` : énumère les entiers de  $a$  à  $b-1$ ,  
`range(d,f,p)` : énumère les entiers en partant de  $d$ , de  $p$  en  $p$  sans dépasser  $f-1$ .

### Exercice 9 : Construire des boucles simples

1. Affiche les cubes des entiers de 0 à 100.
2. Affiche les puissances quatrièmes des entiers de 10 à 20.
3. Affiche les racines carrées des entiers de 0, 5, 10,  $\dots$ , 100.
4. Affiche les puissances de 2, de  $2^0$  à  $2^{10}$ .

### Exercice 10 : Des boucles plus compliquées

Demande une valeur  $n$  et calcule la somme :

$$1^2 + 2^2 + 3^2 + \dots + n^2.$$

### Exercice 11 : Des boucles plus compliquées

Demande une valeur  $n$  impaire et calcule le produit :

$$1 \times 3 \times 5 \times \cdots \times n.$$

On peut également imbriquer deux ou plusieurs boucles.

### Exercice 12 : Des boucles imbriquées

Affiche les tables de multiplication entre 1 et 10. Voici un exemple de ligne à afficher :

$$6 \times 9 = 54$$

### Exercice 13 : Des boucles imbriquées (2)

Écrire un programme qui demande un nombre impair à l'utilisateur et affiche ensuite une pyramide comme celle-ci :

```
*  
***  
*****  
*****
```

## 4 Tortue

Le module *turtle* permet facilement de dessiner en *Python*. Il s'agit de commander une tortue en lui donnant des ordres simples comme "avancer", "tourner"...

### Code 12 : La tortue Python

```
from turtle import *  
  
forward(100) # on avance  
left(90) # 90 degrés à gauche  
forward(50)  
width(5) # épaisseur du trait  
forward(100)  
color('red')  
right(90)  
forward(200)  
exitonclick()
```

### Leçon 8 : Principales commandes de turtle

- `forward(l)` avance de  $l$ ,
- `backward(l)` recule de  $l$ ,
- `right(a)` tourne vers la droite d'un angle  $a$  sans avancer,
- `left(a)` tourne vers la gauche d'un angle  $a$  sans avancer,
- `setheading(d)` s'oriente dans une direction  $d$  (0=droite, 90=haut...),
- `goto(x,y)` se déplace au point  $(x,y)$ ,
- `down()` abaisse le stylo,
- `up()` lève le stylo,
- `color(c)` change la couleur "red", "green", "blue"...,
- `position()` renvoie la position de la tortue,
- `heading()` renvoie la direction (en degré) vers laquelle pointe la tortue,

`towards(x,y)` renvoie l'angle entre l'horizontale et la droite passant par la tortue et le point  $(x,y)$ ,  
`exitonclick()` termine le programme dès qu'on clique.

Les coordonnées de l'écran vont par défaut de  $-475$  à  $+475$  pour les  $x$  et de  $-400$  à  $+400$  pour les  $y$ .

#### Exercice 14 : Pentagone

Trace un pentagone bleu de côté 100 en répétant 5 fois un même bloc d'instructions.

#### Exercice 15 : Polygone régulier

Demande une longueur et un nombre de côté et trace un polygone régulier à  $n$  cotés dont le périmètre est la longueur demandée. Il est possible d'utiliser `int` pour convertir une chaîne en entier.

#### Exercice 16 : Spirale

Trace une spirale en vert en tournant toujours dans le même sens mais en avançant d'une longueur qui augmente à chaque étape.

#### Exercice 17 : Graphe d'une fonction

Trace le graphe de la fonction carré. Pour cela, répète pour  $x$  de  $-200$  à  $+200$  :

- poser  $y = \frac{1}{100}x^2$
- aller à  $(x,y)$

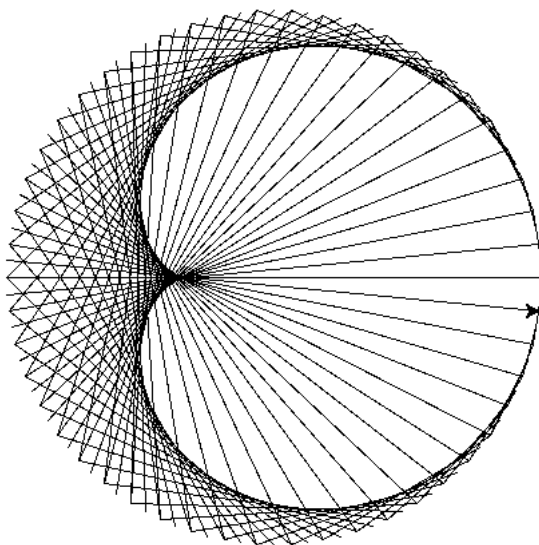
#### Exercice 18 : Coeur des tables de multiplication

On place sur un cercle  $n$  points numérotés de 0 à  $n-1$ . Pour chaque  $k \in \{0, \dots, n-1\}$ , on relie le point  $k$  et le point numéro  $2 \times k \bmod n$  par un segment.

Les coordonnées  $(x_i, y_i)$  peut être calculées par la formule :

$$x_i = r \times \cos\left(\frac{2 \times i \times \pi}{n}\right) \text{ et } y_i = r \times \sin\left(\frac{2 \times i \times \pi}{n}\right)$$

Avec  $n = 100$ , on obtient :



On peut définir plusieurs tortues qui se déplacent de façon indépendante chacune de leur côté.



### Code 13 : Plusieurs Tortues

```
tortue1 = Turtle()
tortue2 = Turtle()

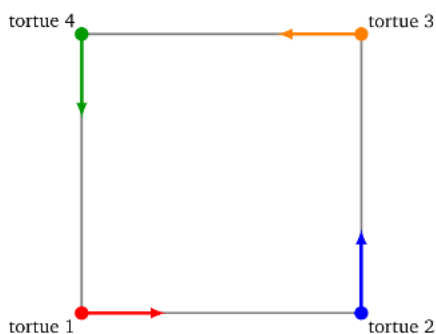
tortue1.color('red')
tortue2.color('blue')

tortue1.forward(100)
tortue2.left(90)
tortue2.forward(100)
```

Nous allons organiser une course entre 4 tortues où chacune d'elles essaient de rattraper celle qui est devant.

### Exercice 19 : Course des tortues

On considère quatre tortues initialement placées comme sur la figure suivante :



- la tortue 1 court après la tortue 2,
- la tortue 2 court après la tortue 3,
- la tortue 3 court après la tortue 4,
- la tortue 4 court après la tortue 1.

À chaque étape :

- tu récupères la position la tortue 1 par `pos1=tortue1.position()` (et idem pour les autres tortues)
- tu calcules l'angle entre la tortue 1 et la tortue 2 par `angle1=tortue1.towards(pos2)`
- tu orientes la tortue 1 selon cet angle : `tortue1.setheading(angle1)`
- tu avances la tortue 1 de 10.

Remarque : on peut accélérer une tortue avec `tortue1.speed(0)`.

## 5 Si alors sinon

On peut avoir besoin de n'effectuer des opérations que si une condition est vérifiée et ne rien faire si elle est fausse (mais passer à la suite tout de même).

### Leçon 9 : Si ... alors ...

```
if condition:
    instruction_1
    instruction_2
    .....
instructions suivantes
```

Annotations :

- mot réservé "if"
- une condition
- deux points
- bloc d'instructions indenté, sera exécuté seulement si la condition est vraie
- suite du programme

On peut avoir besoin d'avoir un traitement différent selon la condition : effectuer certaines opérations si la condition est vraie et d'autres si elle est fausse.

## Leçon 10 : Si ... alors ... sinon ...

```
if condition:
    → instruction
    → instruction
    → .....
else:
    → instruction
    → .....
instructions suivantes
```

bloc exécuté si la condition est vraie

bloc exécuté si la condition est fausse

## Exercice 20 : Retour(2) sur les équations du second degré

Modifier votre programme de résolution pour tenir compte des cas où il n'y a pas de solutions. Votre programme doit afficher le nombre de solutions et les solutions si il en existe.

## Remarque 5 : Module random

Le module `random` génère des nombres au hasard. Pour s'en servir, il faut ajouter au début de votre programme :

```
from random import *
```

- la commande `randint(a,b)` renvoie un entier au hasard compris (au sens large) entre  $a$  et  $b$ .
- `random()`, sans argument, renvoie un nombre réel dans l'intervalle  $[0, 1[$ .

## Exercice 21 : Quiz multiplications

Écris un programme qui réalise les actions suivantes :

1. initialise deux variables  $a$  et  $b$  au hasard entre 1 et 12,
2. affiche à l'écran le message "combien vaut le produit  $axb$ ?" (en remplaçant  $a$  et  $b$  par leurs valeurs)
3. récupère la réponse de l'utilisateur,
4. si la réponse est correcte, affiche "Bravo", sinon "Perdu! la réponse était ..."

## Leçon 11 : Comparaisons

- $a==b$  teste l'égalité entre deux valeurs,
- $a<b$  teste si  $a$  est strictement inférieur à  $b$ ,
- $a<=b$  teste si  $a$  est inférieur au sens large à  $b$ ,
- $a!=b$  teste si  $a$  est différent de  $b$ .

Le programme suivant affiche tous les entiers de 0 à 99. Comprends ce programme. Que représentent les variables  $u$  et  $d$ ?

## Code 14 : Chiffres

```
for d in range(10):
    for u in range(10):
        n=10*d+u
        print(n)
```

## Exercice 22 : Propriétés de nombre

Trouve tous les entiers compris entre 0 et 999 qui vérifient toutes les propriétés suivantes :

1. le chiffre des dizaines est pair,
2. la somme des chiffres est supérieure ou égale à 15,

3. la somme des centaines et des unités est divisible par 7

Modifie ton programme pour compter le nombre d'entiers vérifiant les propriétés.

### Exercice 23 : Propriétés des triangles

Demande à l'utilisateur trois valeurs entières  $a$ ,  $b$  et  $c$  avec  $a \leq b \leq c$ . Le programme doit vérifier et afficher les propriétés suivantes :

**Ordre** tester si les longueurs vérifient bien  $a \leq b \leq c$ ;

**Existence** Il existe un triangle correspondant à ces longueurs si et seulement si :  $a + b > c$ .  
Tester si c'est le cas.

**Rectangle** tester si le triangle est un triangle rectangle (si les valeurs  $a$ ,  $b$  et  $c$  forment un triangle).

**Équilatéral** Teste si le triangle est équilatéral (si les valeurs  $a$ ,  $b$  et  $c$  forment un triangle).

**Isocèle** Teste si le triangle est isocèle (si les valeurs  $a$ ,  $b$  et  $c$  forment un triangle).

**Angles aigus** Teste si tous les angles sont aigus (il faut que les cos de tous les angles soit positifs. On peut les calculer avec la formule  $\cos(\alpha) = \frac{-a^2 + b^2 + c^2}{2bc}$  ou  $a$  est la longueur opposé à l'angle  $\alpha$ ).

### Exercice 24 : Nombre mystère

- L'ordinateur choisit un nombre au hasard entre 0 et 99,
- le joueur propose une réponse,
- l'ordinateur répond "le nombre à trouver est plus grand" ou "le nombre à trouver est plus petit",
- le joueur a sept tentatives pour trouver la réponse.

*Indications* : Pour quitter une boucle **for**, tu peux utiliser la commande **break**. Utilise ceci lorsque le joueur trouve la bonne réponse.

### Exercice 25 : L'ordinateur triche

Reprends le programme précédent. Mais maintenant l'ordinateur triche. À chaque tour, il change un peu le nombre mystère en ajoutant à chaque fois un nombre au hasard pris entre  $-3$  et  $3$  (sans sortir de  $[0, 99]$ ).

## 6 Fonctions

### 6.1 Définitions

Une fonction en informatique est une portion réalisant un tâche bien précise et qui pourra être utilisée une ou plusieurs fois dans la suite du programme. Pour définir une fonction avec Python, c'est très simple.

## Leçon 12 : Définition et appel de fonction

*# définition de la fonction*

```
def mafonction(param) :  
    instruction_1  
    instruction_2  
    ...  
    instruction_n  
    return resultat
```

*# appel de la fonction*

```
x = 7  
val = mafonction(x)
```

mot réservé "def"  
nom de la fonction  
liste des paramètres  
évalue le résultat, termine la fonction et renvoie le résultat  
résultat renvoyé et récupéré  
appel de la fonction  
argument de la fonction

Les avantages de la programmation utilisant des fonctions sont les suivants :

- on écrit le code d'une fonction une seule fois, mais on peut appeler la fonction plusieurs fois ;
- en divisant notre programme en petits blocs ayant chacun leur utilité propre, le programme est plus facile à écrire, à lire, à corriger et à modifier ;
- on peut utiliser une fonction (écrite par quelqu'un d'autre, comme par exemple la fonction `sqrt()`) sans connaître tous les détails internes de sa programmation.

### Code 15 : Calcul du cube

```
def cube(a) :  
    c = a*a*a  
    return c  
  
x=3  
y=4  
z=cube(x)+cube(y)  
print(z)
```

### Exercice 26 : Fonction sans paramètre ni sortie

Programmez une fonction appelée `affiche_table_de_7()` qui affiche la table de multiplication par 7 :  $1 \times 7 = 7$ ,  $2 \times 7 = 14$ ...

### Exercice 27 : Fonction sans paramètre ni sortie (suite)

Programmez une fonction appelée `affiche_bonjour()` qui demande à l'utilisateur son prénom et affiche ensuite "Bonjour" suivi du prénom de l'utilisateur.

### Exercice 28 : Fonction avec paramètre sans sortie

Programmez une fonction appelée `affiche_une_table(n)` qui dépend d'un paramètre  $n$  et qui affiche la table de multiplication par l'entier  $n$ . Par exemple, la commande `affiche_une_table(5)` doit afficher :  $1 \times 5 = 5$ ,  $2 \times 5 = 10$ ...

### Exercice 29 : Fonction avec paramètre sans sortie

Programmez une fonction appelée `affiche_salutation(formule)` qui dépend d'un paramètre `formule`. Cette fonction demande le prénom de l'utilisateur et affiche une formule de salutation suivi du prénom. Par exemple `affiche_salutation("Coucou")` afficherait "Coucou" suivi du prénom donné par l'utilisateur.

### Exercice 30 : Fonction sans paramètre et avec sortie

Programme une fonction appelée `demande_prenom_nom()` qui demande d'abord le prénom de l'utilisateur, puis son nom et renvoie comme résultat l'identité complète avec le nom en majuscule. Par exemple, si l'utilisateur saisi « Dark » puis « Vador », la fonction renvoie la chaîne "Dark VADOR" (la fonction n'affiche rien).

Indication : Si `chaine` est une chaîne de caractères, alors `chaine.upper()` est la chaîne transformée avec les caractères en majuscules. Exemple : si `chaine = "Vador"` alors `chaine.upper()` renvoie "VADOR".

### Code 16 : Fonctions : suite

```
def somme_produit(a,b):  
    """Calcule la somme et le produit de deux nombres"""  
    s = a + b  
    p = a * b  
    return s, p  
  
som, pro = somme_produit(6,7)
```

### Exercice 31 : Solutions d'une équation de second degré

Soit  $a$ ,  $b$  et  $c$  trois entiers. Définir une fonction qui renvoie les solutions de l'équation  $ax^2 + bx + c = 0$ .  
**Remarque :** les complexes existent en python :  $a=1-3j$ . Ils se notent avec un  $j$  au lieu du  $i$ .

### Exercice 32 : Multiplication

Définir une fonction qui, à partir de deux entiers  $a$  et  $b$  positifs, calcule le produit  $a \times b$  en n'utilisant que des additions.

### Exercice 33 : Factorielle

Définir une fonction qui, à partir d'un entier  $n$  positif, calcule la factorielle de  $n$ .

### Exercice 34 : Primalité

Définir une fonction qui, à partir d'un entier  $n$  positif strictement, indique par un booléen si  $n$  est premier ou non.

### Exercice 35 : Nombre premier (2)

Améliorez la fonction précédente en `est_premier2(n)` qui teste les entiers jusqu'à  $\sqrt{n}$ .

**Question :** Pourquoi peut-on s'arrêter à  $\sqrt{n}$  ?

### Exercice 36 : Nombre premier (3)

Améliorez la fonction précédente en `est_premier3(n)` qui à partir de 3 ne teste plus que les impairs.

On voudrait vérifier expérimentalement que nous avons gagné en efficacité. Voilà comment il est possible de le faire.

### Code 17 : Comparaison de temps

```
import timeit  
print(timeit.timeit("est_premier(1013)",setup="from __main__ import  
    ↪ est_premier",number=100))
```

```
print(timeit.timeit("est_premier2(1013)",setup="from __main__ import
↳ est_premier3",number=100))
print(timeit.timeit("est_premier3(1013)",setup="from __main__ import
↳ est_premier3",number=100))
```

La fonction `timeit` demande 3 arguments :

- une chaîne de caractères contenant l'expression à mesurer,
- un argument qui indique où se trouve la fonction,
- le nombre de fois qu'il faut recommencer (ici 100 fois).

### Exercice 37 : Primalité (suite)

Définir une fonction qui affiche tous les nombres premiers inférieurs à une limite passée en paramètre. Votre fonction devra utiliser la précédente.

### Exercice 38 : Polygones

Définir une fonction qui trace un polygone régulier à  $n$  côtés de longueur  $a$ . Si  $n$  est impair, tous les côtés seront verts, sinon, les côtés seront en alternance rouges et bleus.

### Remarque 6 : Retenons donc

- Il peut y avoir plusieurs paramètres en entrée.
- Il peut y avoir plusieurs résultats en sortie.
- **Très important !** Il ne faut pas confondre afficher et renvoyer une valeur. L'affichage (par la commande `print()`) affiche juste quelque chose à l'écran. La plupart des fonctions n'affichent rien, mais renvoient une valeur (ou plusieurs). C'est beaucoup plus utile car cette valeur peut être utilisée ailleurs dans le programme.
- Dès que le programme rencontre l'instruction `return`, la fonction s'arrête et renvoie le résultat. Il peut y avoir plusieurs fois l'instruction `return` dans une fonction mais une seule sera exécutée. On peut aussi ne pas mettre d'instruction `return` si la fonction ne renvoie rien.
- Dans les instructions d'une fonction, on peut bien sûr faire appel à d'autres fonctions !
- Il est important de bien commenter tes programmes. Pour documenter une fonction, tu peux décrire ce qu'elle fait en commençant par un *docstring*, c'est-à-dire une description (en français) entourée par trois guillemets :

```
""" Ma fonction fait ceci et cela. """
```

à placer juste après l'entête.

- Lorsque l'on définit une fonction, les variables qui apparaissent entre les parenthèses sont appelées les **paramètres** ; par contre, lorsque l'on appelle la fonction, les valeurs entre les parenthèses sont appelées les **arguments**. Il y a bien sûr une correspondance entre les deux.

## 6.2 Portée des variables

La portée d'une variable désigne les parties du code source où elle est utilisable. Ce peut être par exemple, le projet entier (plusieurs fichiers source), le fichier source dans lequel la variable est créée, ou simplement une partie spécifique de ce fichier.

En Python, la portée d'une variable peut être :

- restreinte à une fonction : dans ce cas nous dirons que la variable est locale.
- le programme principal et ainsi que l'ensemble des fonctions défini par le programmeur. Dans ce cas nous dirons que la variable est globale.

Dans un fichier source ne contenant pas de fonction toutes les variables sont donc globales. Lorsqu'il contient des fonctions, elles peuvent être locales ou globales.

La règle permettant de savoir si une variable est locale ou globale est implicite : toutes les variables subissant une affectation dans le corps d'une fonction sont par défaut locales à cette fonction et ne peuvent donc être utilisées en dehors de cette fonction. Par contre, les variables apparaissant dans le membre droit

d'une affectation sont par défaut globales (à moins qu'elles n'aient auparavant subi une affectation dans la même fonction).

### Exercice 39 : Essayer de prédire l'affichage de ces codes

```
a=8
def affiche():
    print(a)

affiche()
```

```
a=3
def affiche():
    a=7
    print(a)

affiche()
print(a)
```

```
a=2
def affiche():
    print(a)
    a=9

affiche()
```

## 6.3 Paramètres

Les paramètres formels d'une fonction peuvent être considérés comme des variables locales. Donc tout ce que nous avons dit sur les variables locales est également valable pour les paramètres. En particulier :

- Deux paramètres de fonctions distinctes peuvent avoir le même nom.
- La durée de vie d'un paramètre est celle de l'exécution de la fonction qui l'utilise.

L'exécution d'un appel de fonction provoque les opérations suivantes :

- Les paramètres effectifs sont tout d'abord évalués au moment de l'appel.
- De la mémoire est allouée pour chaque paramètre formel.
- La **valeur** de chaque paramètre effectif est affectée au paramètre formel correspondant, ou plus précisément de même position : la valeur du premier paramètre effectif est affecté au premier paramètre formel, la valeur du second paramètre effectif est affecté au second paramètre formel, ... etc.
- Le code de la fonction est exécuté avec ces valeurs de paramètres.

### Exercice 40 : Qu'affiche ce code ?

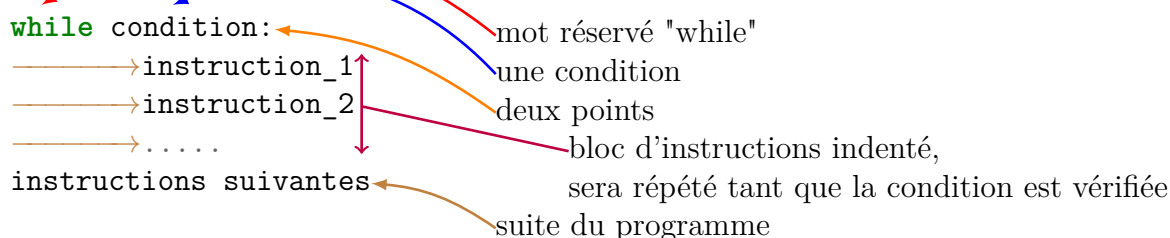
```
def inc(x):
    x=x+1
    print(x)

x=5
print(x)
inc(x)
print(x)
```

## 7 Boucle Tant que

La boucle "*tant que*" exécute des instructions tant qu'une condition est vraie. Dès que la condition devient fausse, elle passe aux instructions suivantes.

### Leçon 13 : Boucle Pour



```
while condition:
    instruction_1
    instruction_2
    .....
instructions suivantes
```

mot réservé "while"

une condition

deux points

bloc d'instructions indenté, sera répété tant que la condition est vérifiée

suite du programme

### Code 18 : Compte à rebours

```
n=10
while n>=0:
    print(n)
    n=n-1
```

Ce programme affiche un compte à rebours 10, 9, 8,  $\dots$  3, 2, 1, 0. Tant que la condition  $n \geq 0$  est vraie, on diminue  $n$  de 1. La dernière valeur affichée est  $n = 0$ , car ensuite  $n = -1$  et la condition  $n \geq 0$  devient fausse donc la boucle s'arrête.

### Code 19 : Puissance de 2 immédiatement supérieure

```
n=100
p=1
while p<n:
    p=p*2
print(p)
```

### Exercice 41 : pgcd

On peut calculer le *pgcd* de deux nombres  $a$  et  $b$  en utilisant l'algorithme suivant :

```
Tant que a!=b faire
si a>b alors
a=a-b
sinon
b=b-a
fin si
le résultat est a
```

Programmez une fonction *pgcd* qui calcule le *pgcd* de deux nombres passés en paramètre.

### Exercice 42 : Calcul d'une moyenne

Créer une fonction qui demande à un utilisateur de saisir des nombres positifs et en calcule la moyenne jusqu'à la saisie d'un nombre négatif.

### Exercice 43 : Plus petit diviseur

Programmez une fonction *plus\_petit\_diviseur*( $n$ ) qui renvoie le plus petit diviseur  $d \geq 2$  d'un entier  $n \geq 2$ .

Voici les grandes lignes :

- on commence avec  $d = 2$ ,
- Tant que  $d$  ne divise pas  $n$  alors on passe au candidat suivant ( $d$  devient  $d + 1$ ).
- À la fin  $d$  est le plus petit diviseur de  $n$  (dans le pire des cas  $d = n$ ).

### Exercice 44 : Nombre premier

Modifiez la fonction *plus\_petit\_diviseur*( $n$ ) pour obtenir une fonction *est\_premier*( $n$ ) qui indique par un booléen si  $n$  est premier.

### Exercice 45 : Nombre premier suivant

Déterminez une fonction *nombre\_premier\_apres*( $n$ ) qui renvoie le premier nombre premier  $p$  supérieur ou égal à  $n$ .

Quel est le premier nombre premier supérieur à 100000 ?



### Exercice 46 : $N^{\text{ème}}$ premier

Déterminez une fonction `neme_premier(n)` qui renvoie le premier  $n^{\text{ème}}$  nombre premier. Par exemple, le 6<sup>ème</sup> nombre premier est 13 (après 2, 3, 5, 7, 11).

## 8 Listes

Une **liste** est une suite d'éléments. Cela peut être une liste d'entiers, par exemple `[3, -8, 12, 9]`, ou bien une liste de chaînes de caractères, par exemple `["lundi", "samedi", "dimanche"]` ou bien les objets peuvent être de différents types `[3.14, "pi", 10e-3, "x", True]`.

### 8.1 Manipulation de listes

#### Code 20 : Création de listes

```
liste1 = [5,3,4,1,2] # une liste de 5 entiers
liste2 = ["samedi", "dimanche"] # une liste de chaînes de caractères.
liste3 = [] # une liste vide
liste4 = [1]*10 # une liste de 10 éléments égaux à 1
```

Pour obtenir un élément de la liste, il suffit d'écrire `liste[i]` où  $i$  est le rang de l'élément souhaité. **Attention**, les listes sont numérotées à partir du rang 0.

#### Code 21 : Accéder à un élément

```
liste = [5,3,4,1,2]
print(liste[3])
print(liste[0])
print(liste[-1])
print(liste[-2])
```

Pour ajouter un élément à la fin de la liste, il suffit d'utiliser la commande `liste.append(element)`<sup>1</sup>. Par exemple si `premiers=[2,3,5,7]` alors `premiers.append(11)` rajoute 11 à la liste, si ensuite on exécute la commande `premiers.append(13)` alors maintenant la liste `premiers` vaut `[2,3,5,7,11,13]`.

#### Code 22 : Création de listes

```
liste_carres = [] # On part d'une liste vide
for i in range(10):
    liste_carres.append(i**2) # On ajoute le carré de i à la liste

print(liste_carres)
```

### Exercice 47 : Liste donnée par utilisateur

Définissez une fonction qui renvoie une liste remplie par des valeurs positives saisies par un utilisateur. La fonction s'arrête quand une valeur négative est saisie.

### Exercice 48 : Liste des premiers

Étant donné un entier  $n$  créer la liste des  $n$  premiers nombres premiers.

Il est possible de modifier un élément d'une liste par une affectation. `L[i]` peut être vu comme une variable.

---

1. La compréhension de la notation `l.append(e)` nécessite d'aborder des concepts de programmation objet que nous n'aborderons pas dans ce cours.

### Code 23 : Modification de liste

```
ma_liste=[5,4,2,2,1]
ma_liste[2]=3
print(ma_liste)
```

La longueur d'une liste (son nombre d'élément) est donnée par `len(liste)`.

### Code 24 : Longueur de liste

```
liste=[5,3,4,2,1]
print(len(liste))
print(len([]))
```

## 8.2 Parcours de liste

Avec la fonction `len(liste)` nous disposons d'un moyen de parcourir une liste

### Code 25 : Parcours de liste

```
liste=[5,4,2,3,1]
for i in range(len(liste)):
    print(i, liste[i])
```

Mais il existe un parcours beaucoup plus simple (si nous n'avons pas besoin de connaître la position de l'élément dans la liste) avec une nouvelle forme de l'itération `for`.

### Code 26 : Parcours de liste (2)

```
liste=[5,4,2,3,1]
for e in liste: # pour chaque élément e de la liste
    print(e) # affiche e
```

### Exercice 49 : Somme d'une liste

Définissez deux fonctions qui calculent la somme des éléments d'une liste.

### Exercice 50 : Existence d'un élément dans une liste

Définissez deux fonctions qui déterminent si un élément  $e$  est présent dans une liste.

### Remarque 7 : Appartenance

Il est très important de savoir écrire la fonction précédente. Il existe cependant la possibilité d'écrire :

```
if e in liste:...
```

### Exercice 51 : Nombre d'occurrences d'un élément dans une liste

Définissez une fonction qui détermine combien de fois un élément  $e$  est présent dans une liste.

### Exercice 52 : Extraire

Donner une fonction qui renvoie la liste des positions où apparaît une valeur  $x$  dans la liste  $L$ .

### Remarque 8 : Strings

Les chaînes de caractères peuvent se manipuler comme étant des listes de caractères :

```
message="Bonjour à tous"
for c in message:
```

```
print(c)
```

### Exercice 53 : Sans voyelle

Écrire une fonction qui retourne un texte donné en paramètre sans ses voyelles.

### Exercice 54 : Maximum

Donner une fonction qui calcule le maximum d'une liste donnée en paramètre.

### Exercice 55 : Croissant

Donner une fonction qui détermine si une liste est triée croissante.

## 8.3 Slice

Le terme anglais de *slice* est associé à l'idée de découpage (une part de gâteau ou de pizza). En programmation, et en Python en particulier, un slice permet le découpage de structures de données séquentielles, typiquement les chaînes de caractères ou les listes.

### Leçon 14 : Slice

Si  $i$  et  $j$  sont des indices positifs, la syntaxe `L[i:j]` désigne la liste formée des éléments  $L[k]$  où  $k$  vérifie  $i \leq k < j$ . Noter l'intervalle entier semi-fermé : le terme d'indice de droite n'est jamais inclus dans le slice obtenu.

### Code 27 : slice

```
L = [65, 31, 9, 32, 81, 82, 46, 12]
print(L[2:6])
```

### Leçon 15 : Indice négatif

Un indice strictement négatif dans une liste permet d'accéder à la liste en se référant à la fin de la liste. Par exemple,  $L[-5]$  désigne le 5<sup>e</sup> élément de  $L$  à partir de la fin.

### Code 28 : Indice négatif

```
L=[3,4,7,9,2,6]
print(L[-1])
print(L[-4])
```

### Code 29 : Slice et indice négatif

```
L=[3,4,7,9,2,6]
print(L[2:-2])
print(L[-5:-1])
```

### Remarque 9 : Omission des paramètres

Les indices du slice sont optionnels :

- `L[:b]` est équivalent à `L[0:b]`
- `L[a:]` est équivalent à `L[a:len(L)]`
- `L[:]` est équivalent à `L[0:len(L)]`

### Code 30 : Omission

```
L=[3,4,7,9,2,6]
print(L[:-2])
print(L[-5:])
print(L[:])
```

### Remarque 10 : Pas dans un slice

La syntaxe des slices admet une extension autorisant un 3<sup>e</sup> entier entre les crochets : `L[i:j:k]`. Cet entier  $k$  désigne un pas, comme dans `range`.

### Code 31 : Pas à pas

```
L=[1,2,3,4,5,6,7,8,9]
print(L[:9:2])
print(L[1::3])
print(L[::-1])
```

### Exercice 56 : Sous-listes

Soit  $T$  un texte (une chaîne de caractères) et  $m$  un motif (une autre chaîne de caractère). Donner une fonction qui détermine si  $m$  apparaît dans  $T$ .

### Leçon 16 : Affectation de slice

- On peut écrire `L[a:b]=L2`. Dans ce cas là, les éléments de `L[a : b]` sont supprimés et `L2` est inséré à la position  $a$ .

```
L=[1,2,3,4,5,6,7,8,9]
L[3:6]=[12,13,14,15,16]
print(L)
```

- L'affectation d'une liste dans un slice **avec pas** ne peut se faire que si les deux sont de même longueur :

```
L=[1,2,3,4,5,6,7,8,9]
L[::3]=[0, 0, 0] # ok
L[::2]=[6,6,6] # pas ok
```

### Exercice 57 : Eratosthène

Pour déterminer l'ensemble des nombres premiers inférieurs à  $n$ , on peut créer un tableau de longueur  $n+1$  (pour ne pas être embêté avec les décalages d'indices) contenant la valeur `True` à tous les indices (tant qu'on n'a pas prouvé que  $n$  est composé, il est supposé premier).

- on indique que 0 et 1 ne sont pas premiers (en passant à `False` leurs cases.
- puis en "criblant" :
  - tous les multiples (stricts) de 2 sont composés : on les passe à `False` dans la liste ;
  - tous les multiples (stricts) de 3 sont composés : on les passe à `False` dans la liste ;
  - pour 4, qui est composé (puisque `t[4] == False`), on laisse tomber (ses multiples sont déjà rayés) ;
  - tous les multiples (stricts) de 5 sont composés : ...
  - pour 6, qui est composé...
  - etc.
- On continue tant que le nombre par lequel on crible n'a pas dépassé  $\sqrt{n}$ .

Donner une fonction qui renvoie la liste des booléens indiquant quels entiers sont premiers (pour une certaine limite).

Donner une fonction qui renvoie la liste des premiers inférieurs à  $n$ .

## 8.4 Copie de liste

### Code 32 : Affectation et copie de listes

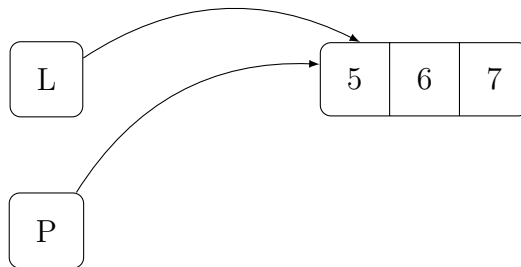
```
L=[1,2,3]
P=L
P.append(4)
print(P)
print(L)
```

### Remarque 11 : Copie de liste

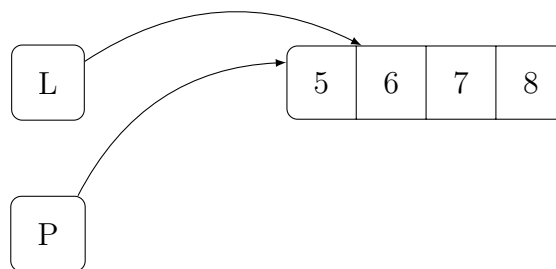
Quand on a une liste  $L=[5,6,7]$ , nous stockons consécutivement en mémoire les valeurs 5, 6 et 7. L'objet  $L$  indique où ils sont stockés :



Lors de l'affectation  $P=L$ , la **valeur** de  $L$  est copiée dans le paramètre  $P$ . Cette valeur en question est l'emplacement mémoire du contenu. Nous avons donc la situation suivante :



Lorsque nous modifions  $P$ , nous ajoutons en mémoire la valeur 8 après le 7.



Mais comme  $L$  continue de désigner la même zone mémoire,  $L$  est également modifié et contient  $[5,6,7,8]$ .

### Code 33 : Affectation et copie de listes

```
L=[1,2,3]
P=L.copy()
P.append(4)
print(P)
print(L)
```

Attention, une telle copie ne fonctionne que si les éléments de la liste sont de type simple. Si la liste contient d'autres listes, il faut alors une copie profonde.

## 8.5 Concaténation de listes

### Code 34 : Que donne ce programme ?

```
def modifie(l):  
    l.append(8)  
L=[5,6,7]  
modifie(L)  
print(L)
```

Il se passe la même chose que pour l'affectation. Lors de l'appel, la valeur de  $L$  (l'emplacement mémoire) est copié dans  $l$ , mais les deux listes indiquent la même zone mémoire.

### Exercice 58 : Fusion de deux listes

Créer une fonction `fusion(l1,l2)` qui prend deux listes en paramètre  $l_1$  et  $l_2$  et qui renvoie une liste contenant les éléments de  $l_1$  suivis des éléments de  $l_2$  sans avoir modifié  $l_1$ .

### Exercice 59 : Fusion de deux listes (2)

Créer une fonction `fusion2(l1,l2)` qui prend deux listes en paramètre  $l_1$  et  $l_2$  et qui modifie  $l_1$  pour lui ajouter les éléments de  $l_2$ . Cette fonction ne renvoie rien.

### Remarque 12 : Concaténation

On peut également concaténer deux listes grâce à l'opérateur "+": `[3,5,6] + [7,9]` donne la liste `[3,5,6,7,9]`.

## 8.6 Manipulation de listes

### Exercice 60 : Rotation de liste

Programme une fonction `rotation(liste)` qui décale d'un rang tous les éléments d'une liste (le dernier élément devenant le premier). La fonction renvoie une nouvelle liste. Par exemple `rotation([1,2,3,4])` renvoie la liste `[4,1,2,3]`.

### Exercice 61 : Inversion de liste

Programme une fonction `inverser(liste)` qui inverse l'ordre des éléments d'une liste. Par exemple `inverser([1,2,3,4])` renvoie la liste `[4,3,2,1]`.

### Exercice 62 : Suppression de case

Programme une fonction `supprimer_rang(liste,rang)` qui renvoie une liste formée de tous les éléments, sauf celui au rang donné. Par exemple `supprimer_rang([8,7,6,5,4],2)` renvoie la liste `[8,7,5,4]` (l'élément 6 qui était au rang 2 est supprimé).

### Exercice 63 : Suppression de valeur

Programme une fonction `supprimer_element(liste,element)` renvoyant une liste qui contient tous les éléments sauf ceux égaux à l'élément spécifié. Par exemple `supprimer_element([8,7,4,6,5,4],4)` renvoie la liste `[8,7,6,5]` (tous les éléments égaux à 4 ont été supprimés).

## 8.7 Tri à bulles

Le tri à bulles est une façon simple de trier une liste du plus petit au plus grand élément. Le principe est le suivant :

- On parcourt la liste en partant du début. Dès que l'on rencontre deux éléments consécutifs dans le mauvais ordre, on les échange.
- À la fin du premier passage, le plus grand élément est à la fin et il ne bougera plus.
- On recommence du début (jusqu'à l'avant-dernier élément), cette fois les deux derniers éléments sont bien placés.
- On recommence. Il y a en tout  $n - 1$  passages si la liste est de taille  $n$ .

#### Exercice 64 : Tri à bulles

Programme le tri à bulles. L'algorithme est donc le suivant :

```
Pour i allant de n-1 à 0 faire
  Pour j allant de 0 à i-1 faire
    Si L[j+1] < L[j] alors échanger L[j] et L[j+1]
```

## 8.8 Visualisation de listes

Avec le module *matplotlib* il est très facile de visualiser des données :

#### Code 35 : Points $(i, L[i])$

```
import matplotlib.pyplot as plt
liste1 = [3,5,9,8,0,3]
liste2 = [4,7,7,2,8,9]
plt.plot(liste1,color="red")
plt.plot(liste2,color="blue")
plt.show()
```

Pour afficher des points  $(x_i, y_i)$ , il faut fournir la liste des coordonnées  $x$ , puis celle des coordonnées  $y$  :

#### Code 36 : courbe $(x, y)$

```
import matplotlib.pyplot as plt
liste_x = [2, 3, 5, 7, 9]
liste_y = [4, 9, 25, 49, 81]
plt.plot(liste_x,liste_y,color="red")
plt.show()
```

#### Exercice 65 : Boulet de canon

On tire un boulet de canon depuis l'origine  $(0, 0)$ . L'équation de la trajectoire est donnée par :

$$y(x) = -\frac{1}{2}g \frac{1}{v^2 \cos^2(\alpha)} x^2 + \tan(\alpha)x,$$

- $\alpha$  est l'angle de tir,
- $v$  est la vitesse initiale,
- $g$  est la constante de gravitation :  $g=9.81$ .

Donnez une fonction `tir(x,v,alpha)` qui renvoie la valeur  $y(x)$ .

Donnez une fonction `trajectoire(xmax, n, v alpha)` qui calcule la liste des ordonnées  $y$  des  $n+1$  points régulièrement espacés entre 0 et  $x_{max}$  ( $x_i = i \frac{x_{max}}{n}$ ).

Pour  $v = 50$ ,  $x_{max} = 270$  et  $n = 100$ , affiche différentes trajectoires selon les valeurs de l'angle  $\alpha$ .

## 9 Activité : recherche d'éléments

Dans tout ce qui suit, nous nous intéressons à la question suivante : Soit  $L$  une liste d'entiers et  $x$  un entier, est-ce que  $x$  est dans  $L$  ?

### 9.1 Recherche linéaire

Le plus simple est de tester tous les  $e \in L$  et de vérifier si on tombe sur  $x$ .

#### Exercice 66 : Recherche linéaire

Écrivez une fonction `recherche(L,x)` qui, sur ce principe, renvoie un booléen indiquant la présence de  $x$  dans  $L$ .

#### Exercice 67 : Efficacité

Combien d'étapes au pire faut-il pour avoir la réponse ? Et dans quel cas le pire arrive-t-il ?

### 9.2 Recherche dichotomique

Sans plus d'information sur  $L$ , il n'est pas possible de faire mieux. Nous allons voir qu'avec des hypothèses supplémentaires sur  $L$  nous pouvons être beaucoup plus efficace. Dans la suite, nous supposons que la liste  $L$  est *triée croissante*.

#### Exercice 68 : Recherche séquentielle améliorée

Suis-je toujours obligé d'aller au bout de la liste si  $x \notin L$  ? Améliorer le programme précédent pour conclure dès que c'est possible.

#### Exercice 69 : Efficacité 2

Quel est la pire situation maintenant ? Combien d'étapes me faut-il alors ?

Nous allons améliorer notre méthode :

Supposons que  $L = \begin{bmatrix} 2 & 5 & 7 & 12 & 21 & 26 & 33 \end{bmatrix}$  et que je cherche à savoir si 28 est présent.

1. Je vais regarder d'abord la valeur de l'élément central :  $\begin{bmatrix} 2 & 5 & 7 & 12 & 21 & 26 & 33 \end{bmatrix}$ .

2. Comme  $28 > 12$ , je vais continuer ma recherche sur  $\begin{bmatrix} 2 & 5 & 7 & 12 & 21 & 26 & 33 \end{bmatrix}$ .

3. Je regarde la valeur de l'élément central :  $\begin{bmatrix} 2 & 5 & 7 & 12 & 21 & 26 & 33 \end{bmatrix}$ .

4. Comme  $28 > 26$ , je continue ma recherche sur  $\begin{bmatrix} 2 & 5 & 7 & 12 & 21 & 26 & 33 \end{bmatrix}$ .

5. Plus qu'une valeur, je teste : la réponse est non, 28 n'est pas dans le tableau.

Pour écrire cette fonction, nous avons besoin de deux variables supplémentaires :

**un entier**  $g$  qui donne la borne gauche de l'intervalle de recherche

**un entier**  $d$  qui donne la borne droite de l'intervalle de recherche

Évidemment, au départ, il faudra commencer avec  $g=0$ , et  $d=\text{len}(L)-1$ .

L'algorithme est alors le suivant :

```
dichotomie(L,x)
    g=0
    d=len(L)-1
    Tant que g<=d faire // J'ai un intervalle de valeur
        si x<L[g] ou x>L[d] alors
            // x est plus petit que le premier élément ou plus grand que le dernier
            renvoyer Faux
```



```

sinon
    // Je calcule le milieu
    m=(g+d)//2
    si L[m]==x alors
        renvoyer vrai
    sinon si L[m]<x alors
        g=m+1 // je dois chercher dans la partie droite
    sinon
        d=m-1 // je dois chercher dans la partie gauche
// J'ai vidé l'intervalle et je ne l'ai pas trouvé
Renvoyer Faux

```

### Exercice 70 : Recherche dichotomique

À partir de ce squelette de programme, essayer d'écrire `dichotomie(L,x)`.

### Exercice 71 : Efficacité 3

Combien d'étapes sont nécessaires pour donner la réponse ?

*indication* : À chaque étape, l'intervalle est divisé par deux.

## 9.3 Recherche par interpolation

Nous pouvons faire mieux. Vous faites mieux quand vous chercher un mot dans un dictionnaire !

Imaginez chercher le mot *banane* dans un dictionnaire. Vous ne faites pas une recherche linéaire en prenant tous les mots dans l'ordre ! Vous n'ouvrez pas votre dictionnaire au milieu pour faire une recherche dichotomique. Vous savez que *banane* commence pas un **b** et se trouve donc plutôt au début. Intuitivement, vous interpolez vers où peut se situer ce mot.

Nous allons faire pareil. Reprenons l'exemple  $L = \begin{array}{|c|c|c|c|c|c|c|} \hline 2 & 5 & 7 & 12 & 21 & 26 & 33 \\ \hline \end{array}$  où nous cherchions 28. Le premier élément (au rang 0) vaut 2, le dernier (au rang 6) vaut 33. Si les éléments sont bien répartis, je devrais trouver 28 à la place  $\frac{6 \cdot (28-2)}{(33-2)} \approx 5$ . Plutôt que de regarder 12 (à la case 3) je regarde 26 (à la case 5), je me suis rapproché beaucoup plus vite.

La fonction `interpolation(L,x)` est la même que `dichotomie(L,x)`, seul le calcul de  $m$  change.

### Exercice 72 : Calcul de $m$

Quel est la place estimée de  $x$  en fonction de  $g, d, x$  et  $L$  ? Donner le code de la fonction `interpolation(L,x)`

*Combien d'étapes dans le pire des cas ?* Question très difficile, c'est normal de ne pouvoir répondre. On peut montrer que si la distribution est uniforme il faut  $\log_2(\log_2(|L|))$  étapes.

## 9.4 Validation expérimentale

Nous allons modifier légèrement les programmes pour compter les nombres d'étapes de calcul de chacune des méthodes. Pour ce faire, nous ajoutons une variable `etape=0` au début de chaque fonction. À chaque itération, nous l'augmentons de 1 et dans chaque `return` nous renvoyons également la valeur de cette variable.

### Exercice 73 : Validation

Modifiez les fonctions précédentes, tester sur des listes de taille 100000 et comparer aux valeurs théoriques (attention, il faut que la liste au départ soit triée).

Pour une exécution où la valeur recherchée n'était pas présente, j'ai obtenu sur mon ordinateur :

```
(False, 100000)
(False, 17) 16.609640474436812
(False, 5) 4.053948940531981
```

Pour une exécution où la valeur recherchée était présente, j'ai obtenu sur mon ordinateur :

```
(True, 45762)
(True, 13) 16.609640474436812
(True, 5) 4.053948940531981
```

### Exercice 74 : Limites

Quelles sont les tailles de tableau pour lesquels je peux répondre (théoriquement) à coup sur en 5 étapes ?

## 10 Récursivité

Un algorithme (ou fonction) **récuratif** est un algorithme qui résout un problème en calculant des solutions d'instances plus petites du même problème. L'approche récursive est un des concepts de base en informatique.

Les premiers langages de programmation qui ont autorisé l'emploi de la récursivité sont *LISP* et *Algol 60*. Depuis, tous les langages de programmation généraux réalisent une implémentation de la récursivité.

On oppose généralement les algorithmes récuratifs aux algorithmes dits itératifs qui s'exécutent sans appeler explicitement l'algorithme lui-même.

### 10.1 Définition

Prenons l'exemple de la factorielle. Celle-ci se définit pour des entiers naturels de la fonction suivante :

$$n! = 1 \times 2 \times 3 \times \cdots \times (n-1) \times n$$

autrement dit :

$$n! = (n-1)! \times n$$

Donc pour définir la fonction qui calcule la factorielle de  $n$ , il suffit d'appeler cette même fonction mais en lui demandant de calculer la factorielle de  $(n-1)$ , et multiplier le résultat par  $n$ . La factorielle de  $(n-1)$  sera calculée en calculant la factorielle de  $(n-2)$  et ainsi de suite. Il suffit juste de définir que la factorielle de 1 est 1 pour "arrêter la récursion" et ne plus appeler la fonction qui calcule la factorielle.

On voit avec cet exemple deux points majeurs de la récursion :

- L'algorithme s'appelle lui-même,  $n!$  a besoin de  $(n-1)!$
- Il est nécessaire d'"arrêter la récursion", en définissant un (ou des) cas ("cas de base") où l'algorithme n'a pas besoin de s'appeler lui-même, ceci afin que l'algorithme ne s'invoque pas lui-même indéfiniment.

L'idée de la récursivité est d'utiliser une définition équivalente, à savoir une définition par récurrence sur la valeur de l'argument :

$$n! = \begin{cases} 1 & \text{si } n = 0 \\ n \times (n-1)! & \text{sinon} \end{cases}$$

Autrement dit, la factorielle d'un nombre qui n'est pas 0 est obtenue en multipliant par  $n$  la factorielle de  $n-1$ . Cette définition de la factorielle peut se traduire par le programme suivant :

### Code 37 : factorielle

```
def factorielle(n):
    if n==0:
```

```

        return 1
    else:
        return n*factorielle(n-1)

```

## 10.2 Exercices

### Exercice 75 : Suite définie par récurrence

Soit  $(u_n)$  la suite définie par :

$$\begin{cases} u_0 = -2 \\ u_n = -\frac{1}{3}u_{n-1} + 1 \quad \text{si } n > 0 \end{cases}$$

Donner une fonction qui calcule le  $n^{\text{ème}}$  de la suite  $(u_n)$ .

### Exercice 76 : Somme des carrés

Donner une fonction récursive qui calcule

$$1^2 + 2^2 + 3^2 + \dots + n^2$$

### Exercice 77 : Puissance

Donner une définition récursive de *puissance*( $x, n$ ) qui calcule  $x^n$  pour  $n \geq 0$ . Comment traiter le cas  $n < 0$  ?

### Exercice 78 : Exponentiation Rapide

On peut définir *puissance*( $x, n$ ) également comme ceci :

$$puissance(x, n) = \begin{cases} puissance(x^2, n/2) & \text{si } n \text{ est pair} \\ x * puissance(x^2, (n-1)/2) & \text{si } n \text{ est impair} \end{cases}$$

### Exercice 79 : Rapidité

Comparer les temps d'exécution des deux fonctions précédentes.

### Exercice 80 : Nombre de chiffres

Écrire une fonction qui renvoie le nombre de chiffres dans l'écriture décimale de  $n$ .

## 10.3 Récursivité et listes

### Exercice 81 : Somme de liste

Donner une fonction récursive qui calcule la somme des éléments d'une liste.

### Exercice 82 : Palindrome

Écrire une fonction qui prend en paramètre une chaîne de caractères (qui se manipule comme une liste de caractères) et qui détermine si c'est un palindrome (mot qui est identique dans les deux sens de lecture).

## 10.4 Pour aller plus loin

### Exercice 83 : Partitions

On appelle **partition** d'un entier une somme d'entiers naturels décroissants. On note  $d(p, q)$  le nombre de partitions différentes de  $p$  en au maximum  $q$  parties. Ainsi, les partitions de 5 en au plus 3 parties sont 5, 4 + 1, 3 + 2, 3 + 1 + 1, 2 + 2 + 1. On a donc  $d(5, 3) = 5$ .

Expliquer les cas d'arrêts suivants :

$$\begin{cases} d(0, q) = 1 \\ d(p + 1, 0) = 0 \\ d(p, q) = d(p, p) \quad \text{pour } p < q \end{cases}$$

Expliquer pourquoi nous avons la relation

$$d(p, q) = d(p - q, q) + d(p, q - 1)$$

Programmer la fonction partition.

## 11 Listes 2

Python offre un certain nombre de fonction permettant d'appliquer des traitements sur tous les éléments d'une liste afin de définir des filtres ou bien de réaliser des calculs..

### Leçon 17 : Filtre

La fonction `filter(f, L)` applique la fonction passée en premier argument sur chacun des éléments de la liste passée en second argument et retourne une nouvelle liste qui contient tous les éléments pour lesquels la fonction a retourné une valeur vrai.

### Code 38 : Filtre

```
def positif(n):  
    return n > 0  
print(filter(positif, [3, -6, 4, -2, -4, 0, 1]))
```

### Exercice 84 : Liste des premiers

En réutilisant la fonction `estPremier3`, donner en une instruction la liste des nombres premiers inférieurs à 1000.

### Leçon 18 : Application

La fonction `map(f, L)` retourne une nouvelle liste qui contient l'image par  $f$  de tous les éléments de  $L$ .

Dans le cas où plusieurs séquences sont passées en paramètre, la fonction doit prendre autant de paramètres qu'il y a de séquences. `map` retourne une liste contenant le résultat de chacun des calculs.

### Code 39 : Application

```
def triple(n):  
    return 3 * n  
print(map(triple, [3, -6, 4, -2, -4, 0, 1]))  
def sum(x, y):  
    return x + y  
print(map(sum, [1, 2, 3], [4, 5, 6]))
```

## Leçon 19 : Liste en compréhension

Comme en mathématiques, il est possible de définir une liste en compréhension, c'est-à-dire en donnant une définition de ses éléments.

### Code 40 : Compréhension

```
L=[2*i+1 for i in range(30)] # définit les 30 premiers impairs
Q=[e**2 for e in L] # les carrés des précédents
Z=[e for e in L if e <= 200]
```

Il est enfin à noter que l'utilisation de ces constructions est en général plus performante que l'utilisation d'une boucle **for**. En effet, leur mise en oeuvre est faite au coeur de l'interprète au lieu d'être interprétée.

### Exercice 85 : Compréhension

Donnez en une seule instruction les listes suivantes :

- tous les nombres pairs compris entre 45 et 65 (inclus).
- tous les multiples de 7 inférieurs à 70.
- la liste des nombres de 1 à 50, sans les vingtaines.
- le produit cartésien de [1,2,3,4] avec ['a', 'b', 'c']<sup>a</sup>

---

a. Il peut y avoir plusieurs **for** dans une liste en compréhension.

### Exercice 86 : Filter et Map

Définissez des fonctions similaires à **filter** et **map** en utilisant des listes en compréhension.

### Exercice 87 : Pairs-Impairs

Soit  $L$  une liste d'entiers, donnez une fonction qui renvoie une liste contenant les mêmes éléments que  $L$  mais avec les pairs au début et les impairs à la fin.

### Exercice 88 : Split

Soit  $L$  une liste d'entiers, donnez une fonction qui renvoie une liste contenant les mêmes éléments que  $L$  mais avec les éléments plus petits que  $L[0]$  au début, les éléments plus grands que  $L[0]$  à la fin (et le bon nombre de  $L[0]$  au milieu).

### Exercice 89 : Tri Rapide

Modifiez le programme précédent pour obtenir un tri rapide (voir wikipedia).

Nous aurions pu aborder ici beaucoup d'autres primitives *python* comme **enumerate** ou **zip**. Nous laisserons le lecteur curieux les découvrir par lui même.

## 12 Dictionnaires

Comme on l'a vu avec les listes, il existe des types non scalaires, c'est-à-dire qui contiennent plusieurs valeurs et qu'on appelle des types construits. Un nouvel exemple de type construit est le **dictionnaire** ou table d'association. Les éléments d'une liste sont ordonnés et on accède à un élément grâce à sa position en utilisant l'indice de l'élément. Un dictionnaire en Python va aussi permettre de rassembler des éléments mais ceux-ci seront identifiés par une **clé**. On peut faire l'analogie avec un dictionnaire de français où on accède à une définition avec un mot.

Contrairement aux listes, un dictionnaire utilise des accolades pour le définir.

### Code 41 : Dictionnaire

```
roues={"voiture" : 4, "tricycle" : 3, "vélo" : 2}  
print(roues["vélo"])
```

### Remarque 13 : Choix des clés

Une clé peut être de différents types : chaînes, entiers, réels, complexes. Mais elle doit être de type constant donc ne peut être une liste par exemple.

Pour ajouter un élément à un dictionnaire, il suffit d'affecter une valeur à une nouvelle clé.

### Code 42 : Ajout dans un dictionnaire

```
roues["monocycle"]=1  
print(roues)
```

### Remarque 14 : Dictionnaire vide

Un dictionnaire vide se note {}.

### Leçon 20 : Parcours d'un dictionnaire

Il existe plusieurs façons de parcourir un dictionnaire à choisir selon les besoins :

- Parcours de toutes les associations avec la méthode **items()** :

```
for assoc in roues.items():  
    print(assoc)
```

- Parcours de toutes les clés avec la méthode **keys()**

```
for k in roues.keys():  
    print(k, roues[k])
```

- Parcours des clés et des valeurs avec la méthode **items()**

```
for cle, val in roues.items():  
    print(cle, val)
```

### Remarque 15

On ne peut pas parcourir simplement les valeurs d'un dictionnaire.

### Remarque 16

La méthode **keys()** est utile également pour savoir si une clé est déjà présente.

### Exercice 90 : Vendeurs

On dispose d'un dictionnaire associant à des noms de commerciaux d'une société le nombre de ventes qu'ils ont réalisées. Par exemple :

```
ventes={"Vincent":14, "Mickael":19, "Michel":15, "Annie":21}
```

1. Écrivez une fonction qui prend en entrée un tel dictionnaire et renvoie le nombre total de ventes dans la société.
2. Écrivez une fonction qui prend en entrée un tel dictionnaire et renvoie le nom du vendeur ayant réalisé le plus de ventes. Si plusieurs vendeurs sont ex-aequo sur ce critère, la fonction devra retourner le nom de l'un d'entre eux.

### Exercice 91 : Fréquence de lettres

Donner une fonction qui prend en entrée une chaîne de caractères et renvoie un dictionnaire indiquant pour chaque lettre sa fréquence dans la chaîne.

### Exercice 92 : Notes

Écrivez une fonction qui prend en entrée un dictionnaire associant à un nom une liste de notes et qui retourne la liste des noms des personnes qui ont la moyenne la plus élevée (s'il y a des ex-aequo, cette liste contiendra plusieurs éléments, sinon, elle n'en contiendra qu'un) et la moyenne correspondante. On pourra utiliser le dictionnaire suivant pour tester la fonction ainsi écrite :

```
notes = {"Tom": [8, 10, 12], "Milan": [10, 9], "Alex": [], "Lina": [12, 10, 8]}
```

### Exercice 93 : Fusion

Écrivez une fonction qui prend en entrée 2 dictionnaires et retourne un dictionnaire fusionnant les informations de ces dictionnaires de la manière suivante : pour chaque clef présente dans au moins un dictionnaire, la valeur associée sera la liste des valeurs associées à cette clef dans les dictionnaires passés en argument à la fonction.

Exemple : pour

```
dict1 = {"a": 1, "d": 4, "c": 7}
dict2 = {"a": 2, "c": 3, "h": 9}
```

La fusion devra retourner :

```
{"a": [1,2], "c": [7,3], "d" : [4], "h": [9]}
```

### Exercice 94 : Dictionnaire de dictionnaires

On dispose d'un dictionnaire de personnes. Chaque personne est une entrée dans un dictionnaire avec comme clé un identifiant. La valeur associée est un autre dictionnaire avec comme clé le prénom, le nom et l'âge de la personne.

Par exemple :

```
personnes = {"Président" : {"prenom" : "Emmanuel", "nom" : "Macron", "age" : 43},
             ↪ "premier" : {"prenom" : "Jean", "nom" : "Castex", "age" : 56}}
```

Ecrire une fonction qui affiche "Prenom NOM a xx ans" pour chacune des personnes du dictionnaire.

## 13 Fichiers

- fichier général, - erreur : try except ? assert - fichier structuré : csv ? à balise ? - application : traitement de données...

Jusqu'à présent, nous avons vu comment écrire sur la sortie standard (fonction **print**) et comment lire depuis l'entrée standard (fonction **input**). Ces échanges de flux de données peuvent aussi s'appliquer aux fichiers stockés en mémoire sur l'ordinateur.

Avant de pouvoir lire et écrire dans un fichier, vous devez créer l'objet *Python* qui servira de lien avec le fichier résidant sur votre machine. Pour cela, on utilise la fonction *Python* **open**, qui prend en argument la localisation (relative ou absolue) du fichier ainsi que le mode d'ouverture. La fonction **open** renvoie alors un objet de type fichier. Une fois que créé l'objet fichier, vous pourrez appeler ses méthodes pour lire ou écrire. Finalement, la fonction **close** (obligatoire !) permet de terminer la connexion avec le fichier.

### Code 43 : Ouverture d'un fichier

```
myFile = open('results.txt', 'w')  
# On réalisera plus tard des opérations sur le fichier  
myFile.close()
```

### Leçon 21 : Mode d'ouverture

Le deuxième paramètre de la fonction `open` indique le mode de traitement du fichier. Les trois principaux modes sont 'r' pour read et 'w' pour write, et 'a' pour append. Il est également possible d'utiliser des modes plus complexes (binaires et lecture+écriture), décrits dans le tableau ci-dessous.

Mode	Description
"r"	Ouvre un fichier en lecture uniquement. Le pointeur est placé au début du fichier. C'est la valeur par défaut.
"w"	Ouvre un fichier en écriture uniquement. Écrase le contenu du fichier s'il existe déjà, crée un nouveau fichier sinon. Le pointeur est placé au début du fichier.
"a"	Ouvre un fichier en écriture uniquement. Le pointeur est placé à la fin du fichier s'il existe, sinon un nouveau fichier est créé.
"rb" ou "wb" ou "ab"	Ouvre le fichier en mode binaire (lecture des octets au lieu des caractères).
"r+" ou "w+" ou "a+"	Ouvre le fichier en lecture et en écriture.
"rb+" ou "wb+" ou "ab+"	Combinaison des deux précédents.

### Leçon 22 : Lecture dans un fichier

Après avoir ouvert un fichier en mode lecture, plusieurs fonctions vous permettent d'en stocker le contenu.

**read()** Copie l'intégralité du fichier dans une chaîne de caractères

**readlines()** Copie l'intégralité du fichier dans une liste (un élément = une ligne)

**readline()** Copie uniquement la ligne suivante du fichier dans une chaîne de caractères

**read(n)** Copie un nombre donné de caractères du fichier dans une chaîne de caractères

### Exercice 95 : Lecture fichier

Donnez une fonction qui prend en paramètre le nom d'un fichier (existant) et qui affiche son contenu en numérotant les lignes.

### Remarque 17 : For sur un fichier

Il est également possible de boucler directement sur l'objet fichier avec un **for** (ou un **while**)

```
myFile=open("fichier.txt", "r")  
for line in myFile:  
    print(line)
```

### Exercice 96 : Lecture avec un for

Refaites l'exercice précédent avec un **for**.



## 14 Tkinter

Pour une documentation plus détaillée mais sans aller trop loin tout de même, il est bien de regarder ce site.

### 14.1 Une application simple

Nous allons progressivement construire une application graphique toute simple. Essayer le code suivant :

#### Code 44 : Étape 1

```
from tkinter import *  
window=Tk()
```

Ce court programme ouvre une fenêtre graphique.

Personnalisons un peu la fenêtre en ajoutant les instructions suivantes :

#### Code 45 : Étape 2

```
window.title("Ma première application")  
window.geometry("600x400")
```

Nous avons défini la taille de la fenêtre ainsi qu'un titre. Vérifiez qu'il est possible de modifier la géométrie de la fenêtre à la souris.

#### Code 46 : Étape 3

```
window.minsize(400,300)  
window.config(bg="#00CC00")
```

Remarquez qu'à partir de là une taille minimale est imposée et que nous avons fixé un fond coloré.

#### Code 47 : Étape 4

```
message=Label(window, text="Lanceur de site web")  
message.pack()
```

Nous venons de créer un texte à afficher dans notre application. La fonction `Label` prend en paramètre le conteneur du message (ici la fenêtre) et le texte. On parle de *widget* ou de *composant graphique*. Il en existe bien d'autres (radios, champs de saisie, boutons...). Une fois le label créé, la méthode `pack` le place physiquement dans la fenêtre. C'est un *gestionnaire de positionnement*. Nous en verrons un plus évolué dans un second temps.

#### Code 48 : Étape 5

```
message=Label(window, text="Lanceur de site web", fg="white", bg="#00CC00", font=  
    ↳ ('Helvetica', '16', 'bold italic'))  
message.pack(side=LEFT)
```

La méthode `pack` empile les éléments dans le conteneur. Nous pouvons lui préciser comment empiler sur le dernier élément. Ici nous empilons en bas. Les valeurs possibles sont TOP (par défaut), RIGHT, LEFT ou BOTTOM. Nous la laisserons à TOP pour la suite.

#### Code 49 : Étape 6

```
entre=Entry(window, bg="white", fg="#00CC00", font= ('Helvetica', '16'))  
entre.pack(fill=X)
```

Nous avons ajouté un champ de saisie. Remarquez que lors d'un agrandissement de la fenêtre, ce champ continue d'occuper tout l'espace en largeur. C'est ce que signifie le `fill=X` lors du pack.

### Code 50 : Étape 7

```
lance=Button(window, text="Go !!!!", bg="white", fg="#00CC00", relief=FLAT)
lance.pack(pady=20)
```

Maintenant un bouton ! Notez le relief choisi. Essayez avec RAISED, SUNKEN, GROOVE ou RIDGE. À quoi correspond *pady* ?

Il ne nous reste presque plus qu'à associer une action à notre bouton.

### Code 51 : Étape 8

```
def go():
    print("bonjour")
lance['command']=go

window.mainloop()
```

Nous avons défini une simple fonction que nous avons associé à au bouton *lance*. Il faut noter que nous ne pouvons associer que **des fonctions sans argument**. Noter également que nous ne donnons que le nom de la fonction **sans parenthèse**. Nous aurions pu faire l'association directement lors de la création du bouton en spécifiant

```
Button(window, text="Go !!!!", bg="white", fg="#00CC00", relief=FLAT, command=go)
```

La méthode finale `mainloop` doit être appelée (généralement après avoir créé tous les widgets statiques) afin de démarrer le traitement des événements. Vous pouvez arrêter ce traitement en boucle en utilisant la méthode `quit()`.

Pour finir notre application, voilà la dernière étape :

- ajouter `from webbrowser import open`
- modifier la fonction `go` :

```
def go():
    adresse=entre.get()
    open(adresse)
```

Dernière amélioration : pour éviter que tous nos widgets soient entassés en haut, nous allons les placer dans une **frame** et c'est cette dernière que nous allons 'packer' dans la fenêtre en demandant à l'étendre au maximum, ce qui aura pour effet de la centrer. Nous obtenons le code final suivant :

### Code 52 : Code final de l'application

```
from tkinter import *
from webbrowser import open

def go():
    adresse=entre.get()
    open(adresse)

window=Tk()
window.title("Ma première application")
window.geometry("400x300")
window.minsize(300,200)
window.config(bg="#00CC00")

interieur=Frame(bg="#00CC00")
```

```

message=Label(interieur, text="Entrez une URL", fg="white", bg="#00CC00", font=
↳ ('Helvetica', '20', 'bold italic'))
message.pack(side=TOP, pady=20)

entre=Entry(interieur, bg="white", fg="#00CC00", font= ('Helvetica', '16'))
entre.pack(fill=X, pady=20)

lance=Button(interieur, text="Go !!!!!", bg="white", fg="#00CC00", relief=FLAT)
lance.pack(pady=20)

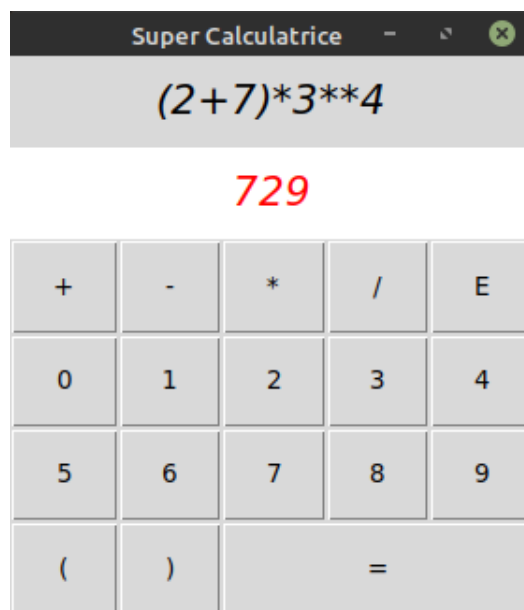
lance['command']=go
interieur.pack(expand=YES, fill=X)

window.mainloop()

```

## 14.2 La calculatrice : grid

L'objectif de cette partie est de réaliser une application de calculatrice simple, grâce aux notions de *grid* et *bind* qui ressemblera à celle-ci :



transformer ce qui suit en activité

```

from tkinter import *
from tkinter.font import Font

def clique(event):
    v = event.widget['text']
    if v == "E":
        expression["text"] = ""
        resultat["text"] = "0"
    else:
        expression["text"] += v

def calcule():

```

```

try:
    v = eval(expression["text"])
except (SyntaxError, ZeroDivisionError):
    v = "Erreur"
resultat["text"] = str(v)

window = Tk()
window.title("Super Calculatrice")
window.geometry("280x280")
window.minsize(280, 300)

police = Font(family='Baskerville', size=16, weight='normal', slant='italic')

expression = Label(window, text="", fg="black", bd=1, font=police)
expression.grid(row=0, column=0, columnspan=5, sticky=NSEW)
resultat = Label(window, text="0", bg="white", fg="red", bd=1, font=police)
resultat.grid(row=1, column=0, columnspan=5, sticky=NSEW)

symboles = "+-*/E0123456789()"
bouton = []
r, c = 2, 0
for i, e in enumerate(symboles):
    bouton.append(Button(window, text=e))
    bouton[-1].bind("<Button-1>", clique)
    # bouton[-1].config(height=2, width=2)
    bouton[-1].grid(row=r, column=c, sticky=NSEW)
    c = c + 1
    if c == 5:
        c = 0
        r = r + 1
egal = Button(window, text="=", command=calcule)
egal.grid(row=5, column=2, columnspan=3, sticky=NSEW)
for i in range(5):
    Grid.columnconfigure(window, index=i, weight=1)
for i in range(6):
    Grid.rowconfigure(window, index=i, weight=1)

window.mainloop()

```

## 14.3 Le morpion : du dessin

Grâce aux *canvas*, il est également possible de dessiner.

transformer ce qui suit en activité

```

##-----Importation des Modules-----##
from tkinter import *
##----- Définition des Variables globales -----##
cases=[ [0, 0, 0],
        [0, 0, 0],
        [0, 0, 0]]
drapeau = True # True pour les croix, False pour les
               ↪ ronds
n = 1 # Numéro du tour de jeu

```

```

##----- Définition des Fonctions -----##
def afficher(event) :
    """ Entrées : Un événement de la souris
        Sortie : Affiche en temps réel les coordonnées de la case du clic de
        → souris """
    global drapeau, cases, n
    l = (event.y-2)//100 # Ligne du clic
    c = (event.x-2)//100 # Colonne du clic
    if (n < 10) and (cases[l][c] == 0):
        if drapeau: # drapeau == True
            dessin.create_line(100*c+8, 100*l+8, 100*c+96, 100*l+96, width = 5, fill =
            → 'blue')
            dessin.create_line(100*c+8, 100*l+96, 100*c+96, 100*l+8, width = 5, fill =
            → 'blue')
            cases[l][c] = 1
            message.configure(text='Aux ronds de jouer')
        else:
            dessin.create_oval(100*c+8, 100*l+8, 100*c+96, 100*l+96, width = 5, outline
            → = 'red')
            cases[l][c] = -1
            message.configure(text='Aux croix de jouer')
        drapeau = not(drapeau)
        if (n >= 5) and (n <= 9):
            somme = verif(cases)
            if somme == 1 or somme == -1:
                n = gagner(somme)
            elif n == 9:
                n = gagner(0)
        n += 1
def verif(tableau):
    """ Entrées : un tableau "carré"
        Sorties : Calcule les sommes de chaque ligne/colonne/diagonale
        et vérifie l'alignement. """
    sommes = [0,0,0,0,0,0,0,0] # Il y a 8 sommes à vérifier
    # Les lignes :
    sommes[0] = sum(tableau[0])
    sommes[1] = sum(tableau[1])
    sommes[2] = sum(tableau[2])
    # Les colonnes
    sommes[3] = tableau[0][0]+tableau[1][0]+tableau[2][0]
    sommes[4] = tableau[0][1]+tableau[1][1]+tableau[2][1]
    sommes[5] = tableau[0][2]+tableau[1][2]+tableau[2][2]
    # Les diagonales
    sommes[6] = tableau[0][0]+tableau[1][1]+tableau[2][2]
    sommes[7] = tableau[0][2]+tableau[1][1]+tableau[2][0]
    for i in range(8): # Parcours des sommes
        if sommes[i] == 3:
            return 1
        elif sommes[i] == -3:
            return -1
    return 0
def gagner(a):
    """ Cette fonction indique le gagnant en modifiant le message et en
        renvoyant la valeur 9. """

```

```

if a == 1:
    message.configure(text = 'Les croix ont gagné !')
elif a == -1:
    message.configure(text = 'Les ronds ont gagné !')
elif a == 0:
    message.configure(text = 'Match nul !')
return 9
def reinit():
    """Cette fonction ré-initialise les variables globales."""
    global drapeau, cases, n
    cases = [[0, 0, 0],
              [0, 0, 0],
              [0, 0, 0]]
    drapeau = True           # True pour les croix, False pour les ronds
    n = 1
    message.configure(text='Aux croix de jouer')
    dessin.delete(ALL)       # Efface toutes les figures
    lignes = []
    for i in range(4):
        lignes.append(dessin.create_line(0, 100*i+2, 303, 100*i+2, width=3))
        lignes.append(dessin.create_line(100*i+2, 0, 100*i+2, 303, width=3))
##-----Création de la fenêtre-----##
    fen = Tk()
    fen.title('Morpion')
##-----Création des zones de texte-----##
    message=Label(fen, text='Aux croix de jouer')
    message.grid(row = 0, column = 0, columnspan=2, padx=3, pady=3, sticky = W+E)
##-----Création des boutons-----##
    bouton_quitter = Button(fen, text='Quitter', command=fen.destroy)
    bouton_quitter.grid(row = 2, column = 1, padx=3, pady=3, sticky = S+W+E)
    bouton_reload = Button(fen, text='Recommencer', command=reinit)
    bouton_reload.grid(row = 2, column = 0, padx=3, pady=3, sticky = S+W+E)
##-----Création du canevas-----##
    dessin=Canvas(fen, bg="white", width=301, height=301)
    dessin.grid(row = 1, column = 0, columnspan = 2, padx=5, pady=5)
##-----La grille-----##
    lignes = []
##-----Evenements-----##
    dessin.bind('<Button-1>', afficher)
##-----Programme principal-----##
    reinit()
    fen.mainloop()           # Boucle d'attente des événements

```