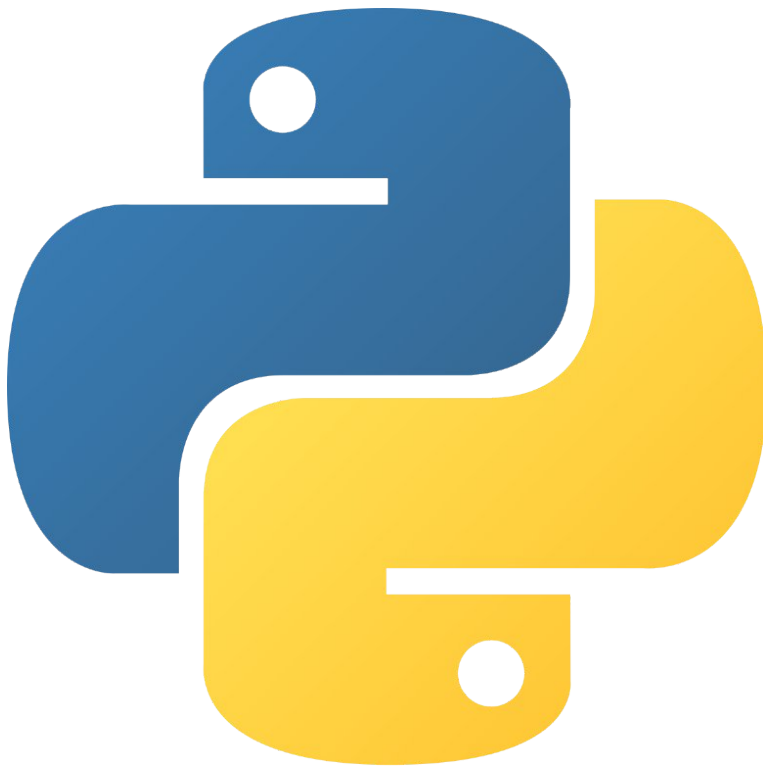


Mémento

Python

Version 3.0 (créé le 10/12/2022, modifié le 30/09/2025)



Python est un langage de programmation de haut niveau qui vous permet de travailler plus rapidement et d'intégrer plus efficacement vos systèmes. Python est puissant... et rapide, fonctionne partout, est convivial et facile à apprendre.



Loric Informatique

Table des matières

1. Prise en main.....	6
1.1. Outils nécessaires.....	6
1.2. Exécuter un programme Python (fichier.py).....	6
1.3. Premier code « Hello World! ».....	6
2. Bases.....	6
2.1. Syntaxe.....	6
2.2. Les variables.....	7
2.2.1. Types de variables.....	7
2.2.2. Opérations sur les variables.....	8
2.3. Commentaires.....	8
2.4. Les listes.....	9
2.5. Les dictionnaires.....	10
2.6. Conditions.....	10
2.6.1. Opérateurs de comparaison.....	10
2.6.2. Tests de conditions.....	11
2.7. Boucles.....	11
3. Les fonctions.....	12
3.1. Créer une fonction.....	12
3.2. Retourner une plusieurs valeurs de la fonction (non obligatoire).....	12
3.3. Faire un appel à la fonction.....	12
4. Les classes.....	13
4.1. Visibilités.....	13
4.2. Les classes de base.....	13
4.2.1. Créer une classe.....	13
4.2.2. Définir la classe dans un objet.....	13

4.2.3. Ajouter un retour de la classe lors de la conversion en chaîne de caractères.....	14
4.2.4. Créer une méthode (dans une classe)	14
4.2.5. Faire un appel d'une méthode autre que <code>__init__</code> et <code>__str__</code> ou d'une variable dans une classe.....	14
4.2.6. Faire un appel d'une méthode autre que <code>__init__</code> et <code>__str__</code> ou d'une variable en dehors d'une classe.....	14
4.2.7. Faire un appel d'une variable statique dans une classe et en dehors.....	14
4.3. L'héritage.....	14
4.3.1. Créer une classe héritée d'une autre classe.....	15
4.3.2. Définir la classe dans un objet.....	15
5. Les instructions.....	15
5.1. Instructions de bases.....	15
5.2. Les fichiers.....	16
6. Les bibliothèques.....	17
6.1. Importer une bibliothèque.....	17
6.2. Lister les bibliothèques requises.....	18
6.2.1. Exporter la liste des bibliothèques requises pour ce projet.....	18
6.2.2. Installer toutes les bibliothèques requises.....	18
6.3. Créer un environnement virtuel.....	18
6.3.1. Sur Linux.....	18
6.3.2. Sur Windows.....	18
6.4. Random.....	19
6.5. Turtle.....	19
6.6. Numpy.....	22
6.7. Math.....	22
6.8. Time.....	24
7. L'interface graphique.....	25

7.1. Syntaxe.....	25
7.2. Les différents paramètres de la fenêtre.....	26
7.3. Gestion de la barre de menu.....	26
7.4. Gestion du panel (zone graphique).....	27
7.4.1. Les différents layouts (méthodes de positionnement).....	27
7.4.2. Les différents paramètres.....	27
7.5. Gestion des widgets et des conteneurs.....	28
7.5.1. Instructions en commun pour tous les widgets et conteneurs.....	28
7.5.2. Paramètres en commun pour tous les widgets et conteneurs.....	28
7.5.3. Les différents widgets.....	28
7.5.4. Les différents conteneurs.....	29
7.6. Les événements.....	31
7.6.1. Les différents écouteurs d'événements.....	31
7.6.2. Informations sur les événements.....	32
7.7. Les boîtes de dialogues.....	32
7.7.1. MessageBox (pour afficher des messages avec un bouton OK).....	32
7.7.2. Simpledialog (pour demander une saisie de l'utilisateur).....	34
7.7.3. Filedialog (nouvelle méthode pour choisir un fichier).....	35
7.7.4. Colorchooser (pour choisir une couleur).....	37
7.8. Les fenêtres modales.....	38
7.8.1. Syntaxe.....	38
7.8.2. Les différents paramètres de la fenêtre modale.....	38
8. Les tests.....	39
8.1. Les tests unitaires avec pytest.....	39
8.1.1. Exécuter les tests unitaires.....	39
8.1.2. Test unitaire simple.....	39
8.1.3. Tests unitaires regroupés.....	40
8.1.4. Vérification qu'une erreur s'est bien produite.....	40

9. Les bonnes habitudes à avoir	40
9.1. Les conventions de nommage.....	40
9.2. Les tests unitaires.....	41
9.2.1. Le pattern Given-When-Then	41

1. Prise en main

1.1. Outils nécessaires

- Python ou Miniconda (version 3.13 ou supérieure)
- IDE (ex : Pyzo ou Visual Studio Code)

Si l'IDE n'arrive pas à démarrer Python parce qu'il manque « distutils », exécuter dans le terminal la commande : `pip install setuptools`

1.2. Exécuter un programme Python (fichier.py)

```
python fichier.py
```

ou

```
python3 fichier.py
```

1.3. Premier code « Hello World! »

```
print("Hello World!")
```

2. Bases

2.1. Syntaxe

```
instruction1  
instruction2  
instruction3
```

2.2. Les variables

2.2.1. Types de variables

Fonction pour connaître le type : `type(variable)`

2.2.1.1. Types de base

Type	Description	Fonction pour le convertir
int	Nombre entier	<code>int(variable)</code>
float	Nombre décimal (ex : 0.1)	<code>float(variable)</code>
str	Chaîne de caractères entre ' ' ou " " (caractère <code>\n</code> : retour à la ligne ; <code>\t</code> : tabulation ; <code>\b</code> : retour en arrière ; <code>\f</code> : nouvelle page)	<code>str(variable)</code>

2.2.1.2. Autres types

Type	Description
bool	Valeur pouvant être True ou False
list	Liste de valeurs sous forme de <code>[val1, val2]</code>
dict	Dictionnaire sous forme de <code>{cle1 : val1, cle2 : val2}</code>
NoneType	Variable qui a pour valeur None (qui est vide)
tuple	Tuplet sous forme de <code>(val1, val2)</code>
set	Tuplet sous forme de <code>{val1, val2}</code> sans ordre d'importance

2.2.2. Opérations sur les variables

Instruction	Description
<code>1 + 2</code>	Renvoie 3
<code>3 - 1</code>	Renvoie 2
<code>6 * 4</code>	Renvoie 24
<code>5 / 2</code>	Renvoie 2.5
<code>5 // 2</code>	Renvoie 2 (le quotient sans décimal)
<code>5 % 2</code>	Renvoie 1 (le reste de la division)
<code>5 ** 2</code>	Renvoie 25 (5 à la puissance 2)
<code>a = 5</code>	Affecte 5 à une variable (pas de déclaration de variable nécessaire)
<code>a, b = 0.5, 60</code>	Affecte des valeurs à de multiples variables
<code>a = a + 3</code>	Ajoute 3 à une variable
<code>a += 3</code>	Ajoute 3 à une variable
<code>texte = 'chaine';</code>	Affecte une chaîne de caractères à une variable (les chaînes peuvent s'additionner avec l'opérateur +)
<code>texte = "chaine";</code>	Affecte une chaîne de caractères à une variable (les chaînes peuvent s'additionner avec l'opérateur +)

2.3. Commentaires

`#Commentaire tenant sur une ligne`

`##Commentaire souligné selon le logiciel tenant sur une ligne`

`"""`

`Commentaire pouvant être sur une ou plusieurs lignes`

`"""`

2.4. Les listes

Instruction	Description
<code>liste = []</code>	Crée une liste vide
<code>liste = ["Loric", "Informatique", 69]</code>	Crée une liste avec des valeurs (le type de valeurs n'a pas d'importance)
<code>liste[i]</code>	Renvoie la valeur de la liste à la position <i>i</i> (l'indice de la première valeur est 0), et permet aussi l'écriture d'une autre valeur
<code>liste[a:b]</code>	Renvoie les valeurs de la position <i>a</i> comprise jusqu'à la position <i>b</i> non comprise
<code>liste[a:]</code>	Renvoie les valeurs de la position <i>a</i> comprise jusqu'à la fin
<code>liste[:b]</code>	Renvoie les valeurs du début jusqu'à la position <i>b</i> non comprise
<code>liste.append(valeur)</code>	Ajoute une valeur dans la liste
<code>liste.pop(i)</code>	Enlève et renvoie la valeur de la liste à la position <i>i</i>
<code>liste.pop()</code>	Enlève et renvoie la dernière valeur de la liste
<code>len(liste)</code>	Renvoie la longueur de la liste
<code>liste.find(valeur, d, f)</code>	Récupérer la position de la première valeur (de l'indice <i>d</i> à <i>f</i> si spécifiés)
<code>liste = chaine.split(separateur)</code>	Couper une chaîne de caractères en précisant le séparateur (valeur) et renvoie une liste
<code>liste = [i for i in range(n)]</code>	Crée une liste de 0 à <i>n</i> -1
<code>nouvelle_liste = liste.copy()</code>	Crée une copie de la liste (<i>nouvelle_liste = liste</i> affecte l'adresse de la liste à une nouvelle variable pointant sur cette même liste)

Les fonctions de recherche marchent également pour les chaînes de caractères.

2.5. Les dictionnaires

Instruction	Description
<code>dict = {}</code>	Crée un dictionnaire vide
<code>dict = {"a": "Loric", "b": 69}</code>	Crée un dictionnaire avec des valeurs (le type de valeurs n'a pas d'importance)
<code>dict["a"]</code>	Récupère la valeur contenue dans une clé
<code>dict["c"] = 3</code>	Ajoute 3 au dictionnaire

2.6. Conditions

Une condition renvoie True si elle est respectée et False sinon

2.6.1. Opérateurs de comparaison

Condition	Description de ce que vérifie la condition
<code>a == b</code>	<code>a</code> égal à <code>b</code>
<code>a < b</code>	<code>a</code> strictement inférieur à <code>b</code>
<code>a > b</code>	<code>a</code> strictement supérieur à <code>b</code>
<code>a <= b</code>	<code>a</code> supérieur ou égal à <code>b</code>
<code>a != b</code>	<code>a</code> n'est pas égal à <code>b</code>
<code>a in b</code>	<code>a</code> est présent dans <code>b</code> (qui peut être une liste)
<code>a is None</code>	Tester si une variable est nulle
<code>or</code>	À mettre entre deux conditions, permet d'avoir une des deux conditions qui doit être vraie
<code>and</code>	À mettre entre deux conditions,

	permet d'avoir deux conditions qui doivent être vraie
<code>not condition</code>	Ne doit pas respecter la condition

2.6.2. Tests de conditions

Instruction	Description
<code>if condition1: instruction1</code>	Si <i>condition1</i> est vraie, alors on exécute <i>instruction1</i>
<code>if condition1: instruction1 else: instruction2</code>	Si <i>condition1</i> est vraie, alors on exécute <i>instruction1</i> , sinon, on exécute <i>instruction2</i>
<code>if condition1: instruction1 elif condition2: instruction2 else: instruction3</code>	Si <i>condition1</i> est vraie, alors on exécute <i>instruction1</i> , sinon, si <i>condition2</i> est vraie, on exécute <i>instruction2</i> , sinon, on exécute <i>instruction3</i>
<code>try: instruction1 except: instruction2</code>	Si <i>instruction1</i> provoque une erreur, on exécute <i>instruction2</i>

2.7. Boucles

Instruction	Description
<code>for i in range(n): instruction1</code>	On répète <i>n</i> fois l'instruction pour <i>i</i> allant de 0 compris à <i>n</i> non compris
<code>for i in range(d, f): instruction1</code>	On répète <i>f-d</i> fois l'instruction pour <i>i</i> allant de <i>d</i> compris à <i>f</i> non compris
<code>for i in range(d, f, p): instruction1</code>	On répète $(f-d)/p$ fois l'instruction pour <i>i</i> allant de <i>d</i> compris à <i>f</i> non compris avec pour pas égal à <i>p</i>
<code>for elt in liste: instruction1</code>	On parcourt la liste (ou une chaîne de caractères) pour <i>elt</i> prenant toutes les valeurs de la liste
<code>while condition:</code>	On répète jusqu'à ce que la

<i>instruction1</i>	condition soit fausse (peut ne pas être répété)
<code>break</code>	Permet de sortir d'une boucle sans la terminer (à éviter si possible)
<code>pass</code>	Passer un bloc s'il est vide, pour éviter des erreurs (à éviter si possible)

3. Les fonctions

Les fonctions peuvent être situées dans le même fichier, mais doivent être définies avant d'être appelées.

3.1. Créer une fonction

```
def ma_fonction(variable1, variable2...):  
    instructions
```

3.2. Retourner une plusieurs valeurs de la fonction (non obligatoire)

```
return variable
```

ou

```
return variable1, variable2...
```

3.3. Faire un appel à la fonction

```
variable = ma_fonction(valeur1, valeur2...)
```

ou

```
variable1, variable2... = ma_fonction(valeur1, valeur2...)
```

ou (s'il n'y a pas de variable de retour)

```
ma_fonction(valeur1, valeur2...)
```

Remarque : Il est possible d'affecter une valeur par défaut, si aucune valeur n'est fournie aux paramètres de la fonction, sous la forme :

variable=valeur_par_defaut

Il est également possible de créer des fonctions dans des fonctions.

4. Les classes

Les classes peuvent être situées dans le même fichier, mais doivent être définies avant d'être appelées.

4.1. Visibilités

En Python, toutes les déclarations sont publiques et ne peuvent pas être privée. Cependant par convention, pour interdire l'utilisation d'une variable ou méthode en dehors d'une classe, on préfixe les variables et méthodes par `_`.

4.2. Les classes de base

4.2.1. Créer une classe

```
class MaClasse:

    variable_statique3 = valeur3

    def __init__(self, variable1, variable2...):
        self.variable1 = variable1
        self.variable2 = variable2
```

4.2.2. Définir la classe dans un objet

```
mon_objet = MaClasse(valeur1, valeur2...)
```

4.2.3. Ajouter un retour de la classe lors de la conversion en chaîne de caractères

```
def __str__(self):  
    return "message"
```

4.2.4. Créer une méthode (dans une classe)

```
def ma_methode(self, variable1, variable2...):  
    instructions
```

4.2.5. Faire un appel d'une méthode autre que `__init__` et `__str__` ou d'une variable dans une classe

```
self.ma_methode(valeur1, valeur2...)  
self.variable1 = valeur1
```

4.2.6. Faire un appel d'une méthode autre que `__init__` et `__str__` ou d'une variable en dehors d'une classe

```
mon_objet.ma_methode(valeur1, valeur2...)  
mon_objet.variable1 = valeur1
```

4.2.7. Faire un appel d'une variable statique dans une classe et en dehors

```
MaClasse.variable_statique3 = valeur3
```

Il est également possible de créer des fonctions dans des fonctions.

4.3. L'héritage

L'héritage permet à des classes filles de reprendre les mêmes caractéristiques que leur classe mère, et d'ajouter des nouveaux attributs

et/ou méthodes qui leur sont propres. Il est possible de faire un dépassement d'une méthode de la classe mère, c'est-à-dire de créer une méthode du même nom qu'une méthode de la classe mère pour la remplacer. Une classe peut hériter de plusieurs classes.

4.3.1. Créer une classe héritée d'une autre classe

```
class MaClasseHeritee(MaClasse):  
  
    variable_statique6 = valeur6  
  
    def __init__(self, variable1, variable2..., variable4,  
variable5...):  
        MaClasse.__init__(variable1, variable2)  
        self.variable4 = variable4  
        self.variable5 = variable5
```

4.3.2. Définir la classe dans un objet

```
mon_objet = MaClasseHeritee(valeur1, valeur2..., valeur4, valeur5...)
```

5. Les instructions

5.1. Instructions de bases

Instruction	Description
<code>print("texte")</code>	Affiche un texte dans la console avec un retour à la ligne
<code>print(variable)</code> <code>print("Valeur : ", variable)</code>	Affiche une variable dans la console
<code>print("Valeur : ", variable, end = "")</code>	Affiche un texte et/ou une variable sans retour à la ligne
<code>variable = input("Entrer une valeur : ")</code>	Demande une valeur avec le retour dans une variable

<code>type(variable)</code>	Récupérer le type d'une variable
<code>isinstance(mon_objet, MaClasse)</code>	Tester si l'objet est du type de la classe (fonctionne également avec les classes héritées et les types de base)
<code>len(variable)</code>	Obtenir la longueur d'une chaîne de caractères ou d'une liste
<code>assert condition, message</code>	Vérifie que condition est vraie, sinon retourne un message d'erreur
<code>del variable</code>	Supprimer variable de la mémoire (non obligatoire)
<code>n = round(n, d)</code>	Arrondit <i>n</i> au réel à <i>d</i> chiffres après la virgule ou à l'entier si <i>d</i> n'est pas renseigné
<code>global variable</code>	Permet de rappeler une variable du programme principale dans la fonction concernée (à éviter si possible)
<code>help(fonction)</code>	Obtenir les informations écrites dans les ""commentaires"" d'une fonction (fonctionne aussi avec des classes)
<code>exit(code)</code>	Fermer immédiatement le programme avec un code d'erreur facultatif

5.2. Les fichiers

Instruction	Description
<code>fichier = open("nom_du_fichier", "r", encoding = "utf-8")</code>	Ouvre un fichier en lecture seule
<code>fichier = open("nom_du_fichier", "w", encoding = "utf-8")</code>	Crée et ouvre un nouveau fichier en écriture seule (écrase l'ancien fichier si existant)
<code>fichier.close()</code>	Ferme et enregistre le fichier (important pour ne pas bloquer le fichier)

<code>fichier.write(variable)</code>	Écrit le contenu d'une variable dans le fichier (en mode écriture)
<code>variable = fichier.read()</code>	Lit le fichier en entier avec le retour dans une variable (en mode lecture)
<code>variable = fichier.readline()</code>	Lit une ligne du fichier avec le retour dans une variable (en mode lecture) et renvoie une chaîne de caractères vide si la fin du fichier est atteint
<code>with open("nom_du_fichier", "mode", encoding = "utf-8"): instructions_gestion_fichier</code>	Ouvrir le fichier et y exécuter des instructions sur ce fichier (Notes : Si le fichier n'a pas pu être ouvert, le programme continue. La fermeture du fichier se fait automatiquement à la sortie du bloc. Attention ! Ne fonctionne que sur les version 3.10 ou supérieure de Python)

6. Les bibliothèques

6.1. Importer une bibliothèque

`from bibliotheque import *`

ou

`from bibliotheque import fonction1, fonction2...`

(ou `import bibliotheque` mais nécessite d'ajouter `bibliotheque.` devant chaque fonction ou variable... importée).

Si la bibliothèque est introuvable, faire `pip install bibliotheque` dans le shell pyzo ou dans le cmd. Pour Linux, il est nécessaire de passer par un environnement virtuel pour faire des installations avec pip (ou d'utiliser apt).

6.2. Lister les bibliothèques requises

6.2.1. Exporter la liste des bibliothèques requises pour ce projet

```
pip freeze > requirements.txt
```

Des numéros de version sont également associées, il est possible de remplacer les signes == par >=.

6.2.2. Installer toutes les bibliothèques requises

```
pip install -r requirements.txt
```

6.3. Créer un environnement virtuel

Un environnement virtuel permet d'exécuter des programmes Python avec les bibliothèques nécessaires uniquement pour l'application concernée, permettant d'éviter d'installer des bibliothèques utilisées pour seulement quelques projets.

6.3.1. Sur Linux

Créer un venv : `python3 -m venv venv`

Activer le venv : `source venv/bin/activate`

Avec un environnement virtuel, il est possible d'exécuter des commandes python et pip.

6.3.2. Sur Windows

Créer un venv : `python -m venv venv`

Activer le venv : `venv\Scripts\activate`

6.4. Random

Random permet de générer des nombres aléatoires facilement.

Importation : `from random import *`

Instruction	Description
<code>random()</code>	Créer un nombre aléatoire entre 0 compris et 1 exclu
<code>randint(a, b)</code>	Créer un nombre entier aléatoire entre a et b compris
<code>seed(nombre)</code>	Initialiser le générateur pseudo-aléatoire pour reproduire des suites de nombres aléatoires identiques (avec un nombre quelconque pour repérer la suite de nombres aléatoires)

6.5. Turtle

Turtle permet de créer des formes avec des tortues et de les faire déplacer dans une fenêtre graphique.

Importation : `from turtle import *`

Instruction	Description
<code>tortue = Turtle()</code>	Créer une tortue
<code>tortue.forward(n)</code>	Avancer de <i>n</i> pixels
<code>tortue.backward(n)</code>	Reculer de <i>n</i> pixels
<code>tortue.right(a)</code>	Tourner de <i>a</i> degrés dans le sens horaire
<code>tortue.left(a)</code>	Tourner de <i>a</i> degrés dans le sens anti-horaire
<code>tortue.goto(x, y)</code>	Aller vers les coordonnées <i>x</i> et <i>y</i>
<code>tortue.setx(x)</code>	Changer la coordonnée <i>x</i>
<code>tortue.sety(y)</code>	Changer la coordonnée <i>y</i>
<code>tortue.setheading(a)</code>	Régler l'orientation à <i>a</i> degrés
<code>tortue.home()</code>	Réinitialiser les coordonnées à (0,

	0)
<code>tortue.circle(r)</code>	Dessiner un cercle de rayon <i>r</i> pixels
<code>tortue.speed(v)</code>	Régler la vitesse de la tortue, compris entre 0 et 10 inclus (0 : aucune animation ; 1 : lent ; 10 : rapide)
<code>tortue.pos()</code>	Renvoyer la position de la tortue sous la forme (x, y)
<code>tortue.xcor()</code>	Renvoyer la coordonnée x de la tortue
<code>tortue.ycor()</code>	Renvoyer la coordonnée y de la tortue
<code>tortue.heading()</code>	Renvoyer le degré d'inclinaison de la tortue
<code>tortue.up()</code>	Lever la pointe du stylo (pas de dessin quand il se déplace)
<code>tortue.down()</code>	Baisser la pointe du stylo (dessine quand il se déplace)
<code>tortue.width(n)</code>	Régler l'épaisseur du stylo à <i>n</i> pixels
<code>tortue.pencolor(couleur)</code>	Régler la couleur du stylo
<code>tortue.fillcolor(couleur)</code>	Régler la couleur de remplissage
<code>tortue.color(couleur)</code>	Régler la couleur de la tortue
<code>tortue.begin_fill()</code>	À appeler juste avant de dessiner une forme à remplir
<code>tortue.end_fill()</code>	Remplir la forme dessinée après le dernier appel à <code>begin_fill()</code>
<code>tortue.clear()</code>	Supprimer les dessins de la tortue de l'écran
<code>tortue.reset()</code>	Supprimer les dessins de la tortue de l'écran, recentrer la tortue et assigner les variables aux valeurs par défaut
<code>tortue.write(txt, align='left', font = ('Arial', 8, 'normal'))</code>	Écrire le texte txt sur l'écran avec des paramètres facultatifs
<code>stamp_id = tortue.stamp()</code>	Créer un clone d'une tortue et l'enregistrer dans <i>stamp_id</i>
<code>tortue.clearstamp(stamp_id)</code>	Supprimer le clone <i>stamp_id</i> de la

	tortue
<code>tortue.clearstamp()</code>	Supprimer tous les clones de la tortue
<code>tortueBis = tortue.clone()</code>	Créer une copie de la tortue
<code>tortue.undo()</code>	Annuler la ou les dernières (si répété) actions de la tortue
<code>tortue.showturtle()</code>	Afficher la tortue
<code>tortue.hideturtle()</code>	Cacher la tortue
<code>tortue.shape(<i>forme</i>)</code>	Changer la forme de la tortue (qui peut être "arrow", "turtle", "circle", "square", "triangle" ou "classic")
<code>tortue.shape(<i>fichier</i>)</code>	Remplacer la tortue par une image
<code>tortue.onclick(<i>fonction</i>)</code>	Exécuter une fonction au clic de la tortue
<code>tortue.onscreenclick(<i>fonction</i>)</code>	Exécuter une fonction au clic de la souris dans la fenêtre
<code>tortue.onkey(<i>fonction</i>, 'touche')</code>	Exécuter une fonction à l'appui sur une touche du clavier (valeur possible : 'Left', 'Right', 'Top', 'Bottom', 'space'...)
<code>tortue.onrelease(<i>fonction</i>)</code>	Exécuter une fonction au relâchement du clic de la tortue
<code>valeur = tortue.textinput(<i>titre</i>, <i>message</i>)</code>	Créer une fenêtre demandant l'entrée d'un texte
<code>valeur = tortue.numinput(<i>titre</i>, <i>message</i>)</code>	Créer une fenêtre demandant l'entrée d'un nombre
<code>tortue.mainloop()</code>	À mettre à la fin du programme, permet le maintien de la fenêtre turtle
<code>exitonclick()</code>	À mettre à la fin, permet le maintien de la fenêtre et la sortie au clic dans la fenêtre
<code>bye()</code>	Fermer la fenêtre turtle

6.6. Numpy

Numpy permet de créer des tableaux de valeurs d'une taille définie et de pouvoir les afficher de façon lisible son contenu dans une console (fonctionne de la même manière qu'une liste, mais avec des fonctionnalités supprimées).

Installation : `pip install numpy`

Importation : `from numpy import *`

Instruction	Description
<code>tableau = array([val1, val2, val3])</code>	Créer un tableau d'éléments
<code>tableau = zeros(n, type)</code>	Créer un tableau de n éléments rempli de 0 avec pour type int ou float
<code>tableau = empty(n, type)</code>	Créer un tableau non initialisé de n éléments (avec des valeurs aléatoires) (fortement déconseillé)
<code>pi</code>	Obtenir la valeur de π
<code>tableau = arange(d, f, p)</code>	Créer une liste avec des valeurs de d à f non compris avec pour pas p
<code>tableau[i]</code>	Renvoie la valeur du tableau à la position <i>i</i> (l'indice de la première valeur est 0), et permet aussi l'écriture d'une autre valeur

6.7. Math

Math permet d'utiliser les fonctions de base de mathématiques.

Importation : `from math import *`

Instruction	Description
<code>pi</code>	Obtenir la valeur de π
<code>sqrt(nombre)</code>	Renvoyer la racine carrée
<code>log(nombre)</code> ou <code>log(nombre, base)</code>	Utiliser la fonction logarithme
<code>exp(nombre)</code>	Utiliser la fonction exponentielle

<code>degrees(<i>nombre</i>)</code>	Convertir des radians en degrés
<code>radians(<i>nombre</i>)</code>	Convertir des degrés en radians
<code>cos(<i>radians</i>)</code>	Utiliser la fonction cosinus
<code>sin(<i>radians</i>)</code>	Utiliser la fonction sinus
<code>tan(<i>radians</i>)</code>	Utiliser la fonction tangente
<code>acos(<i>nombre</i>)</code>	Utiliser la fonction cosinus ⁻¹
<code>asin(<i>nombre</i>)</code>	Utiliser la fonction sinus ⁻¹
<code>atan(<i>nombre</i>)</code>	Utiliser la fonction tangente ⁻¹

6.8. Time

Time permet de manipuler le temps.

Importation : `from time import *`

Instruction	Description
<code>time()</code>	Retourner l'heure du système en secondes (la différence entre l'exécution de la fonction au début et à la fin peut donner le temps d'exécution du programme)
<code>time_ns()</code>	Retourner l'heure du système en nanosecondes (pour Python 3.7 ou supérieur)
<code>ctime()</code>	Retourner la date et l'heure sous la forme 'Tue Dec 10 16:07:12 2024'
<code>process_time()</code>	Retourner le temps CPU d'exécution du programme en secondes (sans prendre en compte les autres programmes)
<code>process_time_ns()</code>	Retourner le temps CPU d'exécution du programme en nanosecondes (pour Python 3.7 ou supérieur)
<code>sleep(secondes)</code>	Mettre en pause le programme temporairement

7. L'interface graphique

L'interface graphique est réalisée avec la bibliothèque Tkinter.

Importation : `from tkinter import *`

7.1. Syntaxe

```
from tkinter import *

class MaFenetre :
    def __init__(self) :
        self.tk = Tk()

        # Instructions pour paramétrer la fenêtre

        self.tk["menu"] = self.creer_barre_de_menu()
        self.creer_canvas()

    def creer_barre_de_menu(self) :
        ma_barre_de_menu = Menu(self.tk)

        # Instructions pour créer et paramétrer des menus

        return ma_barre_de_menu

    def créer_canvas(self) :
        # Instructions pour créer et paramétrer le panel
```

Il n'est pas obligatoire d'utiliser des classes pour créer des fenêtres.

7.2. Les différents paramètres de la fenêtre

Instruction	Description
<code>tk.title("nom de la fenêtre")</code>	Changer le nom de la fenêtre (en haut à gauche)
<code>tk.iconbitmap("icone.ico")</code>	Changer l'icône de la fenêtre (en haut à gauche et dans la barre des tâches)
<code>tk.after(n, fonction)</code>	Exécuter une fonction après n ms (utile pour faire des animations sans bloquer la fenêtre) (<i>fonction</i> peut être remplacée par <code>lambda:fonction()</code> afin d'ajouter des arguments dans la fonction)
<code>tk.geometry('500x500+200+100')</code>	Redimensionner la fenêtre en 500x500 et placer la fenêtre à x = 200 et y = 100
<code>x = tk.winfo_rootx()</code>	Renvoie la position du bord haut de la fenêtre (sans prendre en compte le bandeau avec le titre et les boutons)
<code>y = tk.winfo_rooty()</code>	Renvoie la position du bord gauche de la fenêtre
<code>tk.bind("<evenement>", fonction)</code>	Créer un événement lié à une action sur la souris ou sur le clavier
<code>tk.mainloop()</code>	Maintenir la fenêtre ouverte (empêche le blocage de la fenêtre sur « Ne répond pas »)

7.3. Gestion de la barre de menu

Instruction	Description
<code>menu1 = Menu(tk)</code>	Créer un menu
<code>menu1.add_command(label = "Nom de la commande", command = fonction)</code>	Ajouter une commande dans un menu
<code>ma_barre_de_menu.add_cascade(label = "Nom du menu", menu = menu1)</code>	Ajouter le menu dans la barre de

7.4. Gestion du panel (zone graphique)

7.4.1. Les différents layouts (méthodes de positionnement)

Instruction	Insertion des widgets	Description
<pre>zg = Canvas(tk, bg = couleur, width = longueur, height = hauteur, parametres) zg.grid(row = 0, column = 0, rowspan = nblignes, colspanspan = nbc colonnes)</pre>	<pre>widget.grid(row = numligne, column = numcolonne)</pre>	Placer un widget avec la méthode de positionnement dans une case (dépend de rowspan ou colspanspan ou les deux)
<pre>zg = Canvas(tk, bg = couleur, width = longueur, height = hauteur, parametres) zg.place(x = 0, y = 0)</pre>	<pre>widget.place(x = abscisse, y = ordonnee)</pre>	Place les widgets de manière absolue grâce à des valeurs personnalisées (fortement déconseillé)

7.4.2. Les différents paramètres

Instruction	Description
<pre>zg.bind("<evenement>", fonction)</pre>	Créer un événement lié à une action sur la souris

7.5. Gestion des widgets et des conteneurs

7.5.1. Instructions en commun pour tous les widgets et conteneurs

Instruction	Description
<code>widget.configure(parametres)</code>	Modifier les paramètres d'un widget
<code>widget.destroy()</code>	Supprimer un widget

7.5.2. Paramètres en commun pour tous les widgets et conteneurs

Paramètre	Description
<code>bg = couleur</code>	Couleur d'arrière-plan
<code>fg = couleur</code>	Couleur de la police
<code>outline = couleur</code>	Couleur de la bordure
<code>width = longueur</code>	Longueur de l'élément
<code>height = hauteur</code>	Hauteur de l'élément
<code>anchor = ancre</code>	Changer l'ancrage de l'élément par rapport à sa position (valeurs possibles : NW, N, NE, E, SE, S, SW, W ou center)

7.5.3. Les différents widgets

7.5.3.1. Label (Texte)

Instruction	Description
<code>widget = Label(tk, text = "Mon texte", parametres)</code>	Créer une zone de texte

7.5.3.2. Entry (Zone de saisie)

Instruction	Description
<code>widget = Entry(tk, textvariable = StringVar(), parametres)</code>	Créer une zone de saisie simple
<code>texteSaisi = widget.get()</code>	Récupérer le texte écrit dans la zone de saisie

7.5.3.3. Button (Bouton)

Instruction	Description
<code>widget = Button(fenetre, text = 'Cliquer ici', parametres, command = fonction)</code>	Créer un bouton

7.5.4. Les différents conteneurs

7.5.4.1. Canvas

Instruction	Description
<code>conteneur = Canvas(tk, parametres)</code>	Créer une zone graphique dans la fenêtre
<code>item1 = conteneur.create_line(x_deb, y_deb, x_fin, y_fin, width = epaisseur, fill = couleur)</code>	Créer une ligne
<code>item2 = conteneur.create_rectangle(x_deb, y_deb, x_fin, y_fin, width = epaisseur, fill = couleur, outline = couleur)</code>	Créer un rectangle
<code>item3 = conteneur.create_oval(x_deb, y_deb, x_fin, y_fin, width = epaisseur, fill = couleur, outline = couleur)</code>	Créer une ellipse
<code>item4 = conteneur.create_text(x, y, text = "Mon texte", font = "Police taille", anchor = ancre)</code>	Créer un texte avec un fond transparent
<code>fichier_imtk = ImageTk.Image.open("nom_du_fichier")</code>	Ouvrir une image en format png ou jpg dans Python (étape

	1 de l'insertion d'une image dans la fenêtre) (nécessite pillow)
<code>fichier_imtk.resize((x, y), ImageTk.Image.ANTIALIAS)</code>	Redimensionner la taille de l'image
<code>image_tk = ImageTk.PhotoImage(image = fichier_imtk)</code>	Ouvrir une image dans tkinter (étape 2 de l'insertion d'une image dans la fenêtre) (nécessite pillow)
<code>item5 = zg.create_image(x, y, image = image_tk, anchor = ancre)</code>	Insérer une image (dernière étape de l'insertion d'une image dans la fenêtre)
<code>variable = conteneur.itemcget(item, "parametre")</code>	Récupérer la valeur d'un paramètre d'un item de la zone graphique
<code>conteneur.itemconfigure(item, parametres)</code>	Modifier un ou plusieurs paramètres d'un item de la zone graphique
<code>conteneur.delete(item)</code>	Supprimer un item de la zone graphique
<code>conteneur.delete("all")</code>	Supprimer tous les items
<code>x, y = conteneur.coords(item)</code>	Récupérer les coordonnées d'un item ayant en argument 2 coordonnées
<code>x_deb, y_deb, x_fin, y_fin = conteneur.coords(item)</code>	Récupérer les coordonnées d'un item ayant en argument 4 coordonnées
<code>conteneur.coords(item, x, y)</code>	Modifier les coordonnées d'un item ayant en argument 2 coordonnées
<code>conteneur.coords(item, x_deb, y_deb, x_fin, y_fin)</code>	Modifier les coordonnées d'un item ayant en argument 4 coordonnées

Remplacer *tk* par le nom de votre conteneur pour placer des éléments à l'intérieur de ce conteneur, ou par *zg* pour placer les éléments dans la zone graphique principale.

7.6. Les événements

7.6.1. Les différents écouteurs d'événements

7.6.1.1. Événements liés à la souris

Événement	Exécution
<ButtonPress-1>	Appui sur le bouton gauche
<ButtonRelease-2>	Relâchement du bouton gauche
<Double-Button-1>	Double clic sur le bouton gauche
<Motion>	Déplacement de la souris
<B1-Motion>	Déplacement avec bouton gauche appuyé
<Enter>	Entrée de la souris dans l'élément
<Leave>	Sortie de la souris de l'élément

7.6.1.2. Événements liés au clavier

Événement	Exécution
<KeyPress>	Appui sur une touche quelconque
<KeyRelease>	Relâchement d'une touche quelconque
<a>	Appui sur la touche a minuscule
<A>	Appui sur la touche A majuscule
<1>	Appui sur la touche 1
<Right>	Appui sur la flèche vers la droite
<Down>	Appui sur la flèche vers le bas
<Up>	Appui sur la flèche vers le haut
<Left>	Appui sur la flèche vers la gauche

<Alt>	Appui sur la touche « Alt »
<Shift>	Appui sur la touche « Majuscule »
<Control>	Appui sur la touche « Ctrl »
<Control-Up>	Combinaison de la touche « Ctrl » et flèche vers le haut

7.6.2. Informations sur les événements

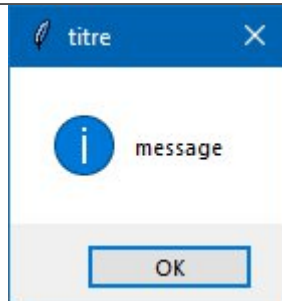
Les fonctions exécutées après un événement doivent obligatoirement avoir en paramètre « event ». Sur cette variable, des informations sur l'événement peuvent être récupérées.

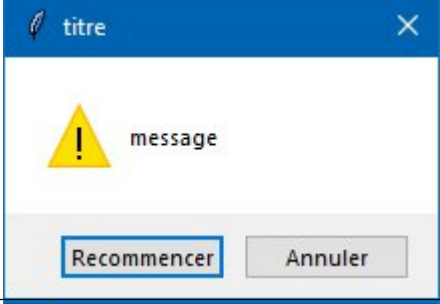
Instruction	Description
event.x	Récupérer la position x de la souris dans l'élément
event.y	Récupérer la position y de la souris dans l'élément

7.7. Les boîtes de dialogues

7.7.1. MessageBox (pour afficher des messages avec un bouton OK)

Importation : `from tkinter import messagebox`

Instruction	Résultat
<code>messagebox.showinfo("titre", "message")</code>	

<code>messagebox.showwarning("titre", "message")</code>	 A Windows-style dialog box titled 'titre' with a yellow warning icon (exclamation mark) and the text 'message'. It has a single 'OK' button at the bottom.
<code>messagebox.showerror("titre", "message")</code>	 A Windows-style dialog box titled 'titre' with a red error icon (X) and the text 'message'. It has a single 'OK' button at the bottom.
<code>reponse = messagebox.askquestion("titre", "message")</code>	 A Windows-style dialog box titled 'titre' with a blue question mark icon and the text 'message'. It has two buttons: 'Oui' and 'Non'.
<code>reponse = messagebox.askokcancel("titre", "message")</code>	 A Windows-style dialog box titled 'titre' with a blue question mark icon and the text 'message'. It has two buttons: 'OK' and 'Annuler'.
<code>reponse = messagebox.askretrycancel("titre", "message")</code>	 A Windows-style dialog box titled 'titre' with a yellow warning icon (exclamation mark) and the text 'message'. It has two buttons: 'Recommencer' and 'Annuler'.
<code>reponse = messagebox.asksyesno("titre", "message")</code>	 A Windows-style dialog box titled 'titre' with a blue question mark icon and the text 'message'. It has two buttons: 'Oui' and 'Non'.

```
reponse =
messagebox.askyesnocancel("titre",
"message")
```



reponse est égal au nom du bouton en minuscule ou au nom du bouton sous la forme : `messagebox.BOUTON`

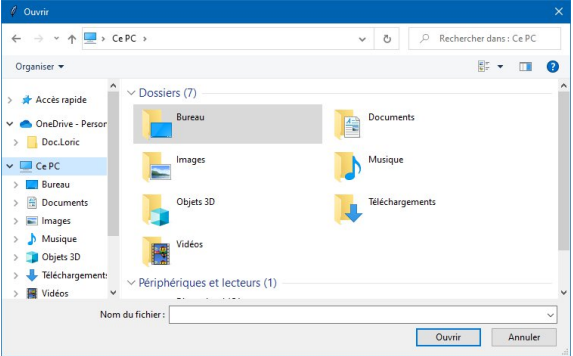
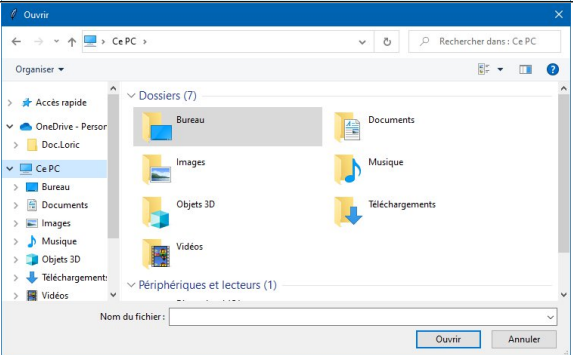
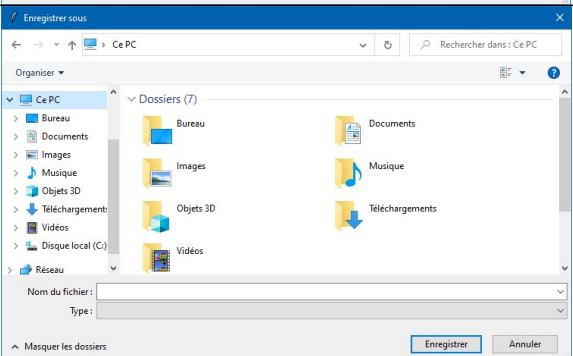
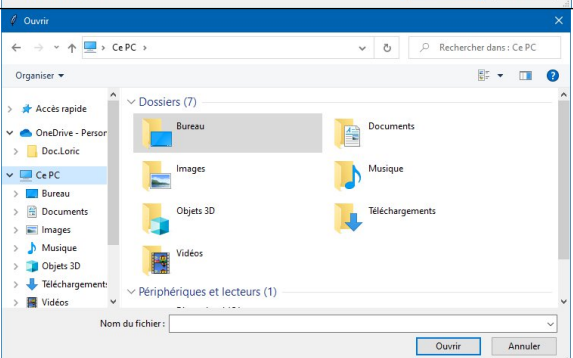
7.7.2. Simpledialog (pour demander une saisie de l'utilisateur)

Importation : `from tkinter.simpledialog import *`

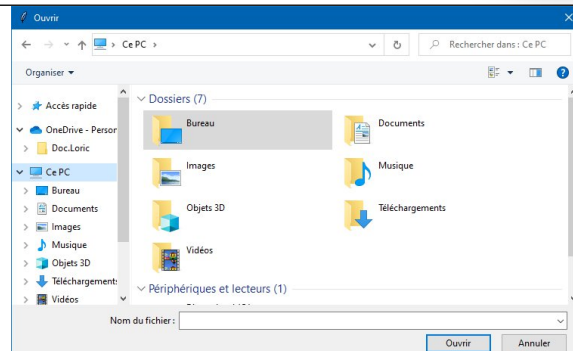
Instruction	Résultat
<pre>reponse = simplifiedialog.askfloat("titre", "message")</pre>	
<pre>reponse = simplifiedialog.askinteger("titre", "message")</pre>	
<pre>reponse = simplifiedialog.askstring("titre", "message")</pre>	

7.7.3. Filedialog (nouvelle méthode pour choisir un fichier)

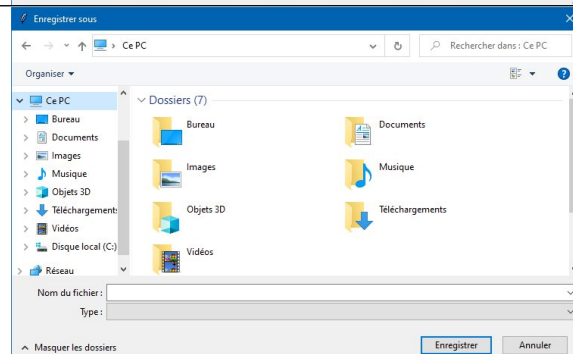
Importation: `from tkinter.colorchooser import *`

Instruction	Résultat
<pre>fichier = filedialog.askopenfile()</pre>	
<pre>fichier = filedialog.askopenfiles()</pre>	
<pre>fichier = filedialog.asksaveasfile()</pre>	
<pre>chemin = filedialog.askopenfilename()</pre>	

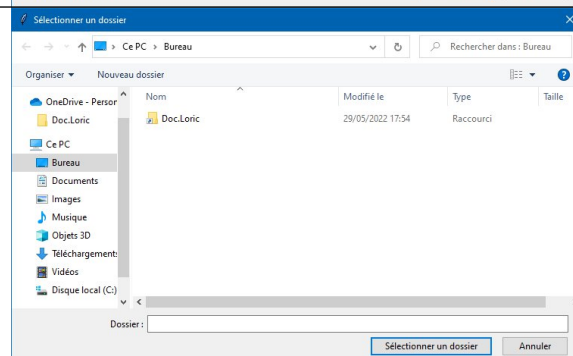
```
chemin =  
filedialog.askopenfilenames()
```



```
chemin =  
filedialog.asksaveasfilename()
```

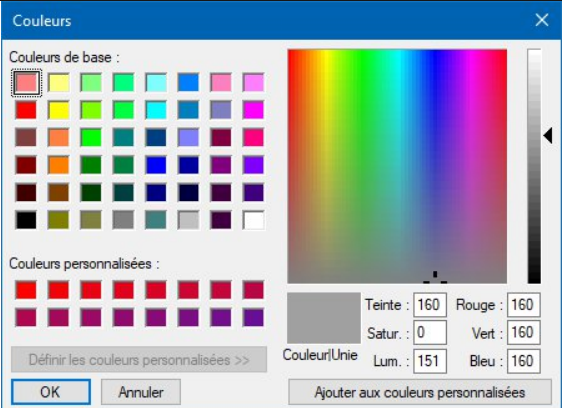


```
chemin =  
filedialog.askdirectory()
```



7.7.4. Colorchooser (pour choisir une couleur)

Importation : `from tkinter.colorchooser import *`

Instruction	Résultat
<pre>reponse = colorchooser.askcolor()</pre>	

reponse est sous la forme : `((r, g, b), "#hexhex")`

7.8. Les fenêtres modales

Les fenêtres modales sont des fenêtres filles de la fenêtre principale. Elles appartiennent donc à la même application.

7.8.1. Syntaxe

```
from tkinter import *

class MaFenetreModale :
    def __init__(self, tk_parent) :
        self.tk = Toplevel(tk_parent)

        # Instructions pour paramétrer la fenêtre

        self.tk["menu"] = self.creer_barre_de_menu()
        self.creer_canvas()

    def creer_barre_de_menu(self) :
        ma_barre_de_menu = Menu(self.tk)

        # Instructions pour créer et paramétrer des menus

        return ma_barre_de_menu

    def creer_canvas(self) :
        # Instructions pour créer et paramétrer le panel
```

7.8.2. Les différents paramètres de la fenêtre modale

Instruction	Description
<code>tk.grab_set()</code>	Interdire le retour sur la fenêtre mère tant que la fenêtre fille n'est pas fermée
<code>tk.grab_release()</code>	Débloquer l'accès à la fenêtre mère

8. Les tests

8.1. Les tests unitaires avec pytest

Les tests unitaires permettent de vérifier le bon fonctionnement des fonctions ou méthodes de manière isolée.

Installation : `pip install pytest`

Tous les fichiers Python de tests doivent être de préférence placés dans un dossier `test`, et doivent obligatoirement commencer par `test_...`

8.1.1. Exécuter les tests unitaires

8.1.1.1. Exécuter tous les tests

Commande : `pytest`

8.1.1.2. Exécuter un seul fichier de test

Commande : `pytest test_fichier.py`

8.1.1.3. Exécuter une fonction ou une classe de test

Commande : `pytest test_fichier.py::test_ma_fonction`

8.1.2. Test unitaire simple

```
def test_ma_fonction():  
    assert ma_fonction(valeur1, valeur2...) == valeur_attendue1  
    assert ma_fonction(valeur3, valeur4...) == valeur_attendue2  
    ...
```

8.1.3. Tests unitaires regroupés

```
class TestMaClasse:
    def test_ma_methode1(self):
        ...

    def test_ma_methode2(self):
        ...

    ...
```

8.1.4. Vérification qu'une erreur s'est bien produite

Exemple :

```
def test_diviser_par_zero():
    with pytest.raises(ValueError) as excinfo:
        diviser(10, 0)
    assert "Cannot divide by zero" in str(excinfo.value)
```

Ou

```
def test_diviser_par_zero():
    with pytest.raises(ValueError, match="Cannot divide by zero"):
        diviser(10, 0)
```

9. Les bonnes habitudes à avoir

9.1. Les conventions de nommage

Convention de nommage	Règles
<i>ma_variable</i>	snake_case
<i>ma_fonction</i>	snake_case
<i>MaClasse</i>	CamelCase
<i>ma_methode</i>	snake_case
<i>nom_des_fichiers</i>	snake_case
<i>nom_des_packages</i>	snake_case
<i>nom_des_bibliotheques</i>	snake_case

9.2. Les tests unitaires

9.2.1. Le pattern Given-When-Then

Le pattern Given-When-Then consiste à séparer une fonction de test en 3 parties :

- Given (Étant donné) : Contient toutes les données et conditions nécessaires.
- When (Quand/Lorsque) : Contient l'action principale que nous testons. Elle devrait idéalement se limiter à un seul appel de fonction ou une seule opération.
- Then (Alors) : Vérifie que le résultat correspond aux attentes. Les assertions doivent être précises et ne vérifier que ce qui est directement lié à l'action du When.

Exemple :

```
def test_diviser():  
    a = 8  
    b = 2  
    resultat_attendu = 4  
  
    resultat = diviser(a, b)  
  
    assert resultat == resultat_attendu
```