République Algérienne Démocratique et Populaire
Ecole Supérieure en Sciences Appliquées de Tlemcen
Département du Second cycle

Cours: Programmation python

1ere année électronique

Option: informatique industrielle

KARAOUZENE Zoheir
E-mail : zkaraouzene@gmail.com

Programmation python

- **Unité d'Enseignement Transversale (UET1.1.1)**
  - Programmation python(**E2I507**):
    - Coefficients :3
    - Crédit: 3
  - Comptabilité et analyse financière de l'entreprise:
    - Coefficients :2
    - Crédit: 2

Programmation python

- Objectifs de la matière:
  - Le cours de Programmation Python est conçu pour aider les étudiants à développer des compétences en programmation de base avec Python, qui sont essentielles pour leur formation en sciences, en ingénierie et en informatique.

Contenu

Chapitre 1: Introduction à Python
- Présentation du langage Python
- Premier programme Python (Hello World) : Lecture et affichage d'un message
- Concepts de base : variables, types de données, opérateurs

Chapitre 2 : Structures de contrôle
- Structures de contrôle conditionnelles (if, else, elif)
- Boucles (for et while)
- Exercices pratiques avec des structures de contrôle

Chapitre 3 : Fonctions
- Définition et appel de fonctions
- Paramètres et arguments
- Fonctions intégrées de Python

Chapitre 4 : Structures de données
- Listes et opérations sur les listes
- Tuples et ensembles
- Dictionnaires

Chapitre 5 : Programmation d'interfaces graphiques (GUI)
- Introduction à Tkinter : création de fenêtres, de boutons, de champs de texte, etc.
- Création d'une application simple avec Tkinter.

Chapitre 6 : Gestion des erreurs
- Erreurs syntaxiques et d'exécution.
- Exceptions et blocs d'essai/exception.

Programmation python

- Livres:
  - 1001 codes Python pour la modélisation,
  - Algorithmique : techniques fondamentales de programmation, exemples en Python, nombreux exercices corrigés,
  - S'initier à la programmation et à l'orienté objet : avec des exemples en C, C++, C#, Java, Python et PHP,
  - La programmation orientée objet : cours et exercices en UML2, Python, PHP, C#, C++ et Java
- Bibliographie
  - "Python for Everybody" de Charles Severance
  - "Learning Python" de Mark Lutz
  - "Python Crash Course" de Eric Matthes
  - "Automate the Boring Stuff with Python" de Al Sweigart
  - "Python Programming: An Introduction to Computer Science" de John Zelle
  - "Python GUI Programming with Tkinter" de Alan D. Moore
  - "Tkinter GUI Application Development Blueprints" de Bhaskar Chaudhary
  - "Python 3 Object-Oriented Programming" de Dusty Phillips
- Pycharm : outil de développement pour les Tps :

  https://www.jetbrains.com/edu-products/download/other-PCE.html

Chapitre1: Introduction à Python

1. Présentation du langage Python

- Python est un langage de programmation informatique polyvalent, adapté à une variété de tâches de programmation et de développement logiciel.
- La première version publique de ce langage a été rendue disponible en 1991. La version la plus récente de Python est la version 3, plus précisément la version 3.11 qui a été publiée en octobre 2022.
- Pycharm : outil de développement pour les Tps :
  - https://www.jetbrains.com/edu-products/download/other-PCE.html

Chapitre1: Introduction à Python

– Ce langage de programmation présente de nombreuses caractéristiques intéressantes:

- **Multiplateforme** : Il est compatible avec divers systèmes d'exploitation: Windows, Mac OS , Linux, Android, iOS, Raspberry Pi ,supercalculateurs.
- **Gratuit** : Il peut être installé sur autant d'ordinateurs que nécessaire, y compris sur les appareils mobiles.
- **Langage de haut niveau** : Son utilisation ne nécessite pas une connaissance approfondie du fonctionnement interne des ordinateurs.
- **Langage interprété** : chaque ligne de code est lue puis interprétée afin d'être exécutée par l'ordinateur. Contrairement à des langages comme le C ou le C++ qui sont des langages compilés.
- **Orienté objet** : comme touts les langages moderne: offre la possibilité de création d'objets qui interagissent entre eux.
- **Facilité d'apprentissage** : Il est réputé pour sa relative simplicité, ce qui en fait une option attrayante pour les débutants.

Python: langage moderne simple

C++ "Hello World"

```
#include<iostream.h>
main()
{
cout << "Hello World";
}
return 0
```

JAVA "Hello World"

```
class HelloWorldApp
 {
  public static void main(String[] args)
   {
    System.out.println("Hello World!");
   }
 }
```

PYTHON

```
print "Hello World"
```

Chapitre1: Introduction à Python

2. Exemple de programmes Python

- Exemple classique : hello word

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

Ce programme affiche simplement "Hello, World!" à l'écran. C'est souvent le premier programme que les débutants écrivent dans n'importe quel langage de programmation.

- Calcul de la somme de deux nombres entiers

```
# Addition
a = 5
b = 3
result = a + b
print("La somme de", a, "et", b, "est égale à", result)
```

Ce programme réalise une opération d'addition élémentaire et présente le résultat à l'écran,

Remarque : tout ce qui suit le caractère # est ignoré par Python jusqu'à la fin de la ligne et est considéré comme un commentaire. Les commentaires doivent expliquer votre code dans un langage humain.

Chapitre1: Introduction à Python

Instructions de communication d'entrée et de sortie : input et print.

Exemple 1:

```
nom_utilisateur = input("Entrez votre nom : ")
print("Bonjour,", nom_utilisateur)
```

Exemple 2

```
nom = "Alice"
age = 30

# Utilisation d'une f-string pour formater la sortie
print(f"Bonjour, je m'appelle {nom} et j'ai {age} ans.")
```

En Python, la lettre "f" est utilisée pour créer des chaînes de format, souvent appelées f-strings. Les f-strings sont une fonctionnalité introduite dans Python 3.6 et ultérieure qui permet d'insérer des expressions Python dans des chaînes de caractères de manière plus lisible et concise. Lorsque vous utilisez la lettre "f" devant une chaîne, vous indiquez à Python d'évaluer les expressions entre accolades {} et d'incorporer les résultats dans la chaîne.

Chapitre1: Introduction à Python

3. Concepts de base

3-1- Notion de bloc d'instructions et d'indentation

- Un bloc d'instructions est un groupe de lignes de code qui sont regroupées et exécutées ensemble en réponse à une certaine condition ou à un événement. Les blocs d'instructions sont souvent utilisés dans des structures de contrôle telles que les boucles et les structures conditionnelles (comme les instructions if, for, et while).
- L'indentation est la manière dont Python détermine quelles lignes de code font partie d'un même bloc d'instructions. Contrairement à de nombreux autres langages de programmation qui utilisent des accolades {} ou des mots-clés pour délimiter les blocs de code, Python utilise l'indentation, c'est-à-dire l'espacement à gauche des lignes de code, pour indiquer les niveaux de blocs.

Chapitre1: Introduction à Python

- Voici un exemple pour illustrer la notion de blocs d'instructions et d'indentation en Python :

```
if x > 5:
    print("x est supérieur à 5")  # Cette ligne est indentée, elle fait partie du bloc if.
    print("Ceci est toujours dans le bloc if.")  # Cette ligne est également indentée.
else:
    print("x est inférieur ou égal à 5")  # Cette ligne est indentée, elle fait partie du bloc else.
print("Ceci n'est pas indenté, il est en dehors du bloc if-else.")
```

- Dans cet exemple, les lignes de code indentées après les instructions if et else font partie de ces blocs respectifs, tandis que la dernière ligne (sans indentation) est en dehors de ces blocs.
- L'indentation correcte est essentielle en Python, car elle détermine la structure du code et son exécution. Une mauvaise indentation peut entraîner des erreurs de syntaxe ou des comportements inattendus.

Chapitre1: Introduction à Python

- **3.2- Variables:**
  - En Python, les variables sont des conteneurs qui permettent de stocker des valeurs ou des données. Voici les principales caractéristiques des variables en Python :
  - **Déclaration de variables** : En Python, vous n'avez pas besoin de déclarer explicitement le type de variable. Vous pouvez simplement assigner une valeur à une variable, et Python détermine automatiquement le type de variable en fonction de la valeur.

Chapitre1: Introduction à Python

- Exemple:

```
x = 10        # x est une variable de type int (entier)
nom = "Omar"   # nom est une variable de type str (chaîne de caractères)
```

  - **Nom de variables** : Les noms de variables doivent commencer par une lettre (a-z, A-Z) ou un souligné (_) suivi de lettres, de chiffres (0-9) ou de soulignés. Les noms de variables sont sensibles à la casse (par exemple, nom et Nom sont deux variables différentes).
  - **Assignation de valeurs** : Vous pouvez assigner une valeur à une variable en utilisant l'opérateur d'affectation =.
  - Exemple :

```
x = 10
```

Chapitre1: Introduction à Python

- **Types de variables** : Les variables en Python peuvent contenir des données de différents types, tels que des entiers, des chaînes de caractères, des listes, des tuples, etc.
  - Exemple :

```
nom = "Alice"
age = 30
liste_de_courses = ["pommes", "bananes", "oranges"]
```

Chapitre1: Introduction à Python

- **Portée des variables** : La portée d'une variable détermine où cette variable est accessible dans le code. Les variables peuvent avoir une portée locale (définies dans une fonction) ou une portée globale (définies en dehors de toutes les fonctions).
- **Convention de nommage** : Il est recommandé de suivre les conventions de nommage PEP 8, qui suggèrent d'utiliser des noms de variables en minuscules avec des mots séparés par des soulignés (par exemple, nom_de_variable) pour améliorer la lisibilité du code.

Chapitre1: Introduction à Python

- Exemple d'utilisation de variables en Python :

```
prenom =  "Samir"
age = 25
taille = 1.80
est_majeur = True

print("Nom :", prenom)
print("Age :", age)
print("Taille :", taille)
print("Majeur :", est_majeur)
```

- Ce code crée quatre variables (prenom, age, taille, et est_majeur) et affiche leurs valeurs à l'écran.

Chapitre1: Introduction à Python

3.3- types de données

Python prend en charge divers types de données pour représenter différents types de valeurs. Voici quelques-uns des types de données de base en Python:

Entiers (int) : Les nombres entiers sans décimales. Par exemple : `42`, `-10`, `0`.

Nombres à virgule flottante (float): Les nombres avec une partie décimale. Par exemple : `3.14`, `0.5`, `-2.0`.

Chaînes de caractères (str): Les séquences de caractères, généralement utilisées pour représenter du texte. Par exemple : `"Bonjour"`, `'Python'`, `"123"`.

Listes (list) : Les collections ordonnées et modifiables d'éléments.

Par exemple : `[1, 2, 3]`, `['a', 'b', 'c']`.

Tuples (tuple) : Les collections ordonnées et immuables d'éléments.

Par exemple : (1, 2, 3), ('a', 'b', 'c').

Dictionnaires (dict) : Les collections non ordonnées de paires clé-valeur. Par exemple {'nom': 'John', 'âge': 30}.

Ensembles (set) : Les collections non ordonnées d'éléments uniques. Par exemple : `{1, 2, 3}`.

Booléens (bool) : Les valeurs `True` (Vrai) ou `False` (Faux) représentant la logique booléenne.

NoneType (None) : Un type spécial qui représente l'absence de valeur ou une valeur nulle.

Bytes et bytearray : Utilisés pour représenter des données binaires brutes.

Python est également doté d'une fonctionnalité de typage dynamique, ce qui signifie que vous n'avez pas besoin de déclarer explicitement le type de variable ; il est déterminé automatiquement en fonction de la valeur affectée à la variable.

Chapitre1: Introduction à Python

3.4- Opérateurs :

3.4.1- Opérateurs arithmétiques :

+ : Addition

- : Soustraction

* : Multiplication

/ : Division

% : Modulo (reste de la division)

** : Exponentiation (puissance)

l'exemple de code resultat = 2**3 calcule l'équation : 2^3

// : Division entière (retourne le quotient entier)

3.4.2- Opérateurs de comparaison :

= =: Égal à

!= : Différent de

< : Inférieur à

> : Supérieur à

<= : Inférieur ou égal à

>= : Supérieur ou égal à

3.4.3- Opérateurs logiques :

and: ET logique

or : OU logique

not : NON logique

Chapitre1: Introduction à Python

3.4.4- Opérateurs d'appartenance :

in : Appartenance à une séquence (liste, chaîne, tuple, etc.)

not in : Non-appartenance à une séquence

```
liste = [1, 5,7 , 12]

# Vérification de l'appartenance d'un élément à la liste
if 5 in liste:
    print("le nombre 5 est dans la liste")
```

Résulta affiché :

```
le nombre 5 est dans la liste
```

Chapitre1: Introduction à Python

3.4.5- Opérateurs d'identité :

is : Vérifie si deux objets sont identiques (même objet en mémoire)

is not : Vérifie si deux objets ne sont pas identiques

Exemple :

```
a = [1, 2, 3]
b = a  # b pointe vers la même liste que a

# Utilisation de l'opérateur "is" pour vérifier l'identité
if a is b:
    print("a et b sont les mêmes objets en mémoire.")
```

Dans cet exemple, nous avons créé une liste a et une liste b qui pointe vers la même liste que a. Par conséquent, lorsque nous utilisons l'opérateur is, nous vérifions que a et b sont les mêmes objets en mémoire.

Chapitre1: Introduction à Python

3.4.6- Opérateurs d'affectation :

= : Affectation simple
+= : Addition et affectation
-= : Soustraction et affectation
*=: Multiplication et affectation
/= : Division et affectation
%=, **=, //= : Opérations respectives et affectation

Exemple

```python
nombre = 10

# Addition et affectation en une seule opération
nombre += 5

# Affichage du résultat
print(nombre)
```

Dans cet exemple, nous avons initialisé une variable nombre à 10. Ensuite, nous avons utilisé l'opérateur += pour ajouter 5 à la valeur actuelle de nombre et mettre à jour la variable nombre avec le résultat. Après cette opération, nombre contiendra la valeur 15.

Chapitre1: Introduction à Python

3.4.7- Opérateurs de concaténation :

+ : Concaténation de chaînes ou de listes

```
prenom = "Amine"
nom_de_famille = "SAAD"

# Concaténation de prénom et de nom de famille
nom_complet = prenom + " " + nom_de_famille

# Affichage du nom complet
print(nom_complet)
```

Exemple :

Dans cet exemple, nous avons deux chaînes de caractères, prenom et nom_de_famille. En utilisant l'opérateur +, nous les avons concaténées avec un espace entre elles pour former la chaîne nom_complet.

Chapitre1: Introduction à Python

3.4.8- Opérateurs de slicing :

`:` : Utilisé pour extraire une sous-séquence d'une séquence (par exemple, une liste ou une chaîne de caractères).

En Python, l'opérateur de slicing est utilisé pour extraire des parties spécifiques d'une séquence, telle qu'une chaîne de caractères, une liste ou un tuple. L'opérateur de slicing utilise la notation [start:stop:step], où :

start : L'indice de début de la section que vous souhaitez extraire (inclusif).

stop : L'indice de fin de la section que vous souhaitez extraire (exclusif).

step (facultatif) : Le pas qui spécifie l'intervalle entre les indices.

```
ma_liste = [10, 20, 30, 40, 50]

# Extraction d'une sous-liste en utilisant l'opérateur de slicing
sous_liste = ma_liste[1:4]

# Affichage de la sous-liste
print(sous_liste)
```

Le résultat affiché sera :

```
[20, 30, 40]
```

Chapitre1: Introduction à Python

Exercice :

Écrivez un programme Python qui permet à l'utilisateur de saisir les informations suivantes sur un voyage en voiture :

La distance totale du voyage en kilomètres.

La consommation de carburant de la voiture en litres par 100 kilomètres.

Le coût du litre de carburant.

Ensuite, utilisez ces informations pour calculer et afficher le coût total du voyage.

```
# Demandez à l'utilisateur de saisir les informations du voyage
distance_km = float(input("Entrez la distance totale du voyage en kilomètres : "))
consommation_litre_100km = float(input("Entrez la consommation de carburant de la voiture en litres pour 100 kilomètres : "))
cout_litre_carburant = float(input("Entrez le coût du litre de carburant : "))

# Calculez la quantité totale de carburant nécessaire
quantite_carburant = (distance_km / 100) * consommation_litre_100km

# Calculez le coût total du carburant
cout_total = quantite_carburant * cout_litre_carburant

# Affichez le coût total du voyage
print(f"Le coût total du voyage en voiture est de : {cout_total} euros")
```

Chapitre 2 : Les structures de cont

- Les principales structures de contrôle en Python sont:
  - Structure de Contrôle Conditionnelle
  - Structure de Contrôle de Boucle

Chapitre 2 : Les structures de cont

1- Structure de contrôle conditionnelle :

La structure de contrôle conditionnelle permet d'exécuter des instructions en fonction de conditions spécifiées. Les principales instructions conditionnelles en Python sont if, elif (abréviation de "else if") et else.

Syntaxe :

```
if condition1:
    # Code à exécuter si condition1 est vraie
elif condition2:
    # Code à exécuter si condition2 est vraie
else:
    # Code à exécuter si aucune des conditions précédentes n'est vraie
```

Chapitre 2 : Les structures de cont

Exemple :

```python
age = 18

if age < 18:
    print("Vous êtes mineur.")
elif age == 18:
    print("Vous avez 18 ans.")
else:
    print("Vous êtes majeur.")
```

Chapitre 2 : Les structures de cont

2- Structure de Contrôle de Boucle

Les boucles les plus couramment utilisées en Python sont for et while.

Boucle for :

La boucle for est utilisée pour itérer sur une séquence (comme une liste, une chaîne de caractères, ou un intervalle de nombres).

Chapitre 2 : Les structures de cont

Syntaxe :

for element in sequence:

    # Code à exécuter pour chaque élément de la séquence

Exemple concret avec une liste :

```
fruits = ["pomme", "banane", "orange"]

for fruit in fruits:
    print(fruit)
```

Chapitre 2 : Les structures de cont

Boucle while :

La boucle while est utilisée pour répéter un bloc d'instructions tant qu'une condition est vraie.

Syntaxe :

```
while condition:
    # Code à exécuter tant que la   condition est vraie
```

Chapitre 2 : Les structures de cont

- Exemple

```python
compteur = 0

while compteur < 5:
    print(compteur)
    compteur += 1
```

Exercice : 1

Calcul de la somme des nombres pairs :

Écrivez un programme Python qui calcule la somme de tous les nombres pairs d'une liste de nombres entiers.

Exercice : 2

Écrivez un programme Python qui calcule la somme de tous les nombres de 1 à n , n est un entier donné par l'utilisateur,

Chapitre3:Fonctions

- Les fonctions en Python sont des blocs de code réutilisables qui permettent d'accomplir une tâche spécifique.
- Elles sont définies à l'aide du mot-clé `def` suivi du nom de la fonction, des paramètres, et d'un bloc d'instructions. Voici comment définir et utiliser une fonction en Python :

,

```python
def ma_fonction(parametre1, parametre2):
    # Corps de la fonction
    resultat = parametre1 + parametre2
    return resultat  # La valeur renvoyée par la fonction

# Appel de la fonction
resultat_de_la_fonction = ma_fonction(3, 5)
print(resultat_de_la_fonction)  # Affiche 8
```

Chapitre3:Fonctions

- concepts importants liés aux fonctions en Python :
  - Définition de la fonction : def
  - Paramètres : entre parenthèses
  - Instructions dans le corps de la fonction
  - return : retourner une ou plusieurs valeurs (ou objet)
  - Appel de la fonction

Chapitre3:Fonctions

Fonctions de base

1. len(): Cette fonction renvoie la longueur d'une séquence, telle qu'une chaîne de caractères, une liste ou un tuple.

Exemple :
my_list = [1, 2, 3, 4, 5]
length = len(my_list)
print(length) # Affiche 5
```

**2. print():** Utilisée pour afficher des données à la console.

Exemple :
message = "Bonjour, monde!"
print(message)  # Affiche "Bonjour, monde!"
```

3. input(): Cette fonction permet de recevoir une entrée utilisateur depuis la console.

Exemple :
user_input = input("Entrez un nombre : ")
print("Vous avez saisi :", user_input)
```

**4. str(), int(), float():** Ces fonctions sont utilisées pour convertir des valeurs d'un type à un autre. Par exemple, str(42) convertit un entier en chaîne de caractères
```

Chapitre3:Fonctions

5. range(debut,fin): Crée une séquence d'entiers.

Exemple :
```
for i in range(5):
    print(i)  # Affiche les entiers de 0 à 4
```

6. type(): Permet de connaître le type d'une variable.

Exemple :
```
x = 42
print(type(x))  # Affiche <class 'int'>
```

7. max() et min(): Renvoient la valeur maximale et minimale d'une séquence.

Exemple :
```
numbers = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
maximum = max(numbers)
minimum = min(numbers)
print("Maximum :", maximum)  # Affiche 9
print("Minimum :", minimum)  # Affiche 1
```

Chapitre3:Fonctions

8. sum(): Calcule la somme des éléments d'une séquence de nombres.

Exemple :
```
numbers = [1, 2, 3, 4, 5]
total = sum(numbers)
print("Somme :", total)  # Affiche 15
```

9. abs(): Renvoie la valeur absolue d'un nombre.

Exemple :
```
x = -10
absolute_value = abs(x)
print("Valeur absolue :", absolute_value)  # Affiche 10
```

Chapitre3:Fonctions

La bibliothèque `random` de Python :

fournit des fonctions pour la génération de nombres aléatoires. Voici quelques-unes des fonctions les plus couramment utilisées de cette bibliothèque :

1. random(): Cette fonction génère un nombre aléatoire à virgule flottante entre 0 (inclus) et 1 (exclus).

Exemple :

```
import random
random_number = random.random()
print(random_number)  # Affiche un nombre aléatoire entre 0 et 1
```

2. randint(a, b): Génère un nombre entier aléatoire dans la plage de valeurs de `a` (inclus) à `b` (inclus).

Exemple :

```
import random
random_integer = random.randint(1, 10)
print(random_integer)  # Affiche un nombre entier aléatoire entre 1 et 10
```

3. choice(seq): Sélectionne un élément aléatoire à partir d'une séquence (comme une liste ou un tuple).

Exemple :

```
import random
my_list = [10, 20, 30, 40, 50]
random_choice = random.choice(my_list)
print(random_choice)  # Affiche un élément aléatoire de la liste
```

Chapitre3:Fonctions

4. shuffle(seq): Mélange aléatoirement les éléments d'une séquence en place.

Exemple :
```
import random
my_list = [1, 2, 3, 4, 5]
random.shuffle(my_list)
print(my_list)  # Affiche une permutation aléatoire des éléments de la liste
```

5. sample(seq, k): Sélectionne un échantillon aléatoire de taille `k` à partir d'une séquence sans remplacement.

Exemple :
```
import random
my_list = [1, 2, 3, 4, 5]
random_sample = random.sample(my_list, 3)
print(random_sample)  # Affiche un échantillon de 3 éléments de la liste
```

6. uniform(a, b): Génère un nombre aléatoire à virgule flottante dans la plage de valeurs de `a` (inclus) à `b` (inclus).

Exemple :
```
import random
random_float = random.uniform(1.0, 2.0)
print(random_float)  # Affiche un nombre à virgule flottante aléatoire entre 1.0 et 2.0
```

Chapitre3:Fonctions

La bibliothèque `time` en Python : fournit des fonctions pour travailler avec le temps et les dates. Voici quelques-unes des fonctions les plus couramment utilisées de cette bibliothèque :

1. time.time(): Cette fonction renvoie le temps écoulé depuis l'époque (en secondes, sous forme de nombre à virgule flottante).

Exemple :
```
import time
current_time = time.time()
print(current_time)  # Affiche le temps écoulé en secondes depuis l'époque
```

2. time.sleep(seconds): Fait attendre le programme pendant un certain nombre de secondes.

Exemple :
```
import time
print("Début de l'attente")
time.sleep(2)  # Fait attendre le programme pendant 2 secondes
print("Fin de l'attente")
```

3. time.localtime(): Renvoie une structure de temps représentant l'heure locale actuelle.

Exemple :
```
import time
local_time = time.localtime()
print(local_time)
```

4. time.strftime(format, time): Cette fonction formate un objet temps (comme celui renvoyé par `time.localtime()`) en une chaîne de caractères en utilisant un format spécifié.

Exemple :
```
import time
current_time = time.localtime()
formatted_time = time.strftime("%Y-%m-%d %H:%M:%S", current_time)
print(formatted_time)
```

Chapitre3:Fonctions

5. time.gmtime(): Renvoie une structure de temps représentant l'heure UTC (temps universel coordonné).

6. time.mktime(time): Convertit une structure de temps en secondes écoulées depuis l'époque.

Exemple :

```
import time
current_time = time.localtime()
timestamp = time.mktime(current_time)
print(timestamp)  # Affiche le temps en secondes depuis l'époque
```

7. time.asctime(): Renvoie une représentation sous forme de chaîne de caractères de l'heure actuelle.

Exemple :

```
import time
current_time = time.asctime()
print(current_time)
```

Chapitre 4 : les listes, tuples et dictionnaires

- Les listes:
  - En Python, une liste est une structure de données qui permet de stocker une collection ordonnée d'éléments.
  - Les listes sont flexibles, offrent la possibilité de stocker des éléments de différents types (nombres, chaînes de caractères, objets, autres listes, etc.).
  - Les listes sont définies en utilisant des crochets `[]` et les éléments sont séparés par des virgules.

Chapitre 4 : les listes, tuples et dictionnaires

Quelques opérations courantes de manipulation des listes en Python :

1. Créer une liste :

```
ma_liste = [1, 2, 3, 4, 5]
```

2. Accéder aux éléments d'une liste par leur index (l'index commence à 0) :

```
premier_element = ma_liste[0]
deuxieme_element = ma_liste[1]
```

3. Modifier un élément de la liste en utilisant son index :

```
ma_liste[2] = 42
```

Chapitre 4 : les listes, tuples et dictionnaires

4. Ajouter des éléments à la fin de la liste avec la méthode `append` :

ma_liste.append(6)

5.Insérer un élément à un emplacement spécifique, utilisez la méthode insert() en spécifiant l'index d'insertion

ma_liste.insert(2, 10) # Insère l'élément 10 à l'index 2

6. Concaténer deux listes avec l'opérateur `+` :

autre_liste = [7, 8, 9]

ma_liste = ma_liste + autre_liste

Chapitre 4 : les listes, tuples et dictionnaires

7. Trouver la longueur d'une liste avec la fonction `len` :

longueur = len(ma_liste)

8. Supprimer un élément de la liste par sa valeur avec la méthode `remove` :

ma_liste.remove(3)

9. Supprimer un élément de la liste par son index avec l'instruction `del` :

del ma_liste[2] # Supprime le troisième élément

Chapitre 4 : les listes, tuples et dictionnaires

10. Vérifier si un élément est présent dans la liste avec l'opérateur `in` :

```
if 4 in ma_liste:
    print("4 est dans la liste.")
```

11. Parcourir une liste avec une boucle `for` :

```
for element in ma_liste:
    print(element)
```

Chapitre 4 : les listes, tuples et dictionnaires

- **Fonctions intégrées pour les listes :**

Python propose plusieurs fonctions intégrées pour effectuer des opérations sur les listes, telles que len(), min(), max(), sum(), sort(), sorted(), etc.

  - La fonction sort() : est une méthode intégrée en Python qui permet de trier les éléments d'une liste en place, c'est-à-dire qu'elle modifie directement la liste d'origine plutôt que de créer une nouvelle liste triée

```
Exemple1 :
ma_liste = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
ma_liste.sort()
Exemple2 :
ma_liste = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
Liste_ord=sorted(ma_liste)
```

Chapitre 4 : les listes, tuples et dictionnaires

- il est possible également de trier la liste dans l'ordre décroissant en utilisant l'argument reverse=True :

```
ma_liste.sort(reverse=True)
```

- Pour trier une liste sans modifier la liste d'origine et créer une nouvelle liste triée, vous pouvez utiliser la fonction sorted(). Par exemple :

```
ma_liste = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
liste_triee_asc = sorted(ma_liste)
liste_triee_desc = sorted(ma_liste,reverse=True)
```

Chapitre 4 : les listes, tuples et dictionnaires

- Exercice 1 :
  - Que fait le programme suivant:

```
liste1 = [10,11,7,18]
liste2=[3,3,2,5]
liste3=[]
liste3.append(liste1)
liste3.append(liste2)
print(liste3)
s=0
for i in range(4):
    s=s+liste3[0][i]*liste3[1][i]
m=s/13
print(m)
```

Chapitre 4 : les listes, tuples et dictionnaires

- Modifier le programme si liste1 et liste2 sont introduites par l'utilisateur :

```
liste1 =[]
liste2=[]
liste3=[]
print("liste1")
for i in range(4):
    x=int(input(f"element {i}: de la premiere liste:"))
    liste1.append(x)
print("liste2")
for i in range(4):
    x=int(input(f"element {i}: de la deuxieme liste:"))
    liste2.append(x)
liste3.append(liste1)
liste3.append(liste2)
print(liste3)
s=0
for i in range(4):
    s=s+liste3[0][i]*liste3[1][i]
m=s/13
print(m)
```

Chapitre 4 : les listes, tuples et dictionnaires

- **<u>Tuples :</u>**

  -En Python, un tuple est une structure de données similaire à une liste

  -les tuples sont immuables, ce qui signifie qu'une fois qu'un tuple est créé, ses éléments ne peuvent pas être modifiés, ajoutés ou supprimés.

  - Les tuples sont définis à l'aide de parenthèses () ou de virgules pour séparer les éléments.

Chapitre 4 : les listes, tuples et dictionnaires

- Voici comment créer et travailler avec des tuples en Python :

Création d'un tuple :

```
mon_tuple = (1, 2, 3)
```

il est possible également créer un tuple sans parenthèses, simplement en séparant les éléments par des virgules :

```
autre_tuple = 4, 5, 6
```

Accès aux éléments d'un tuple :

```
premier_element = mon_tuple[0]  # Accès au premier élément (1)
deuxieme_element = mon_tuple[1]  # Accès au deuxième élément (2)
```

Chapitre 4 : les listes, tuples et dictionnaires

Longueur d'un tuple :

Pour connaître le nombre d'éléments dans un tuple, c'est possible d'utiliser la fonction len().

```
longueur = len(mon_tuple)  # longueur sera égal à 3 dans cet exemple
```

Les tuples sont immuables :

La principale caractéristique des tuples est leur immuabilité. Une fois que vous avez créé un tuple, vous ne pouvez pas changer ses éléments. Par exemple, la tentative de modification d'un élément générera une erreur :

```
mon_tuple[0] = 10  # Cela générera une erreur, car les tuples sont immuables
```

Chapitre 4 : les listes, tuples et dictionnaires

Utilisation de tuples :

- Les tuples sont couramment utilisés dans les cas où vous avez une collection d'éléments qui ne devraient pas être modifiés. Par exemple, vous pouvez utiliser des tuples pour représenter des coordonnées (x, y) ou des enregistrements de données où l'ordre des éléments est important et ne doit pas changer.

- Les tuples peuvent également être utilisés pour retourner plusieurs valeurs à partir d'une fonction, Par exemple :

```
def retourne_coordonnees():
    x = 10
    y = 20
    return x, y

coordonnees = retourne_coordonnees()
```

  - Dans cet exemple, `coordonnees` est un tuple contenant deux valeurs, et il est impossible de les modifier accidentellement.

Chapitre 4 : les listes, tuples et dictionnaires

Dictionnaires

Création d'un dictionnaire :

Un dictionnaire est défini à l'aide d'accolades `{}` et les paires clé-valeur sont séparées par des deux-points `:`.

Par exemple :

```
mon_dictionnaire = {"nom": "Omar", "âge": 30, "ville": "Tlemcen"}
```

Accès aux éléments d'un dictionnaire :

Pour accéder à la valeur associée à une clé dans un dictionnaire, vous pouvez utiliser la notation de crochets `[]` en spécifiant la clé comme indice :

```
nom = mon_dictionnaire["nom"]  # Accès à la valeur associée à la clé "nom"
age = mon_dictionnaire["âge"]  # Accès à la valeur associée à la clé "âge"
```

Chapitre 4 : les listes, tuples et dictionnaires

Modification d'un dictionnaire :

Les dictionnaires sont mutables, ce qui signifie que vous pouvez ajouter, modifier ou supprimer des paires clé-valeur. Par exemple, pour changer la valeur associée à une clé existante :

```
mon_dictionnaire["âge"] = 31  # Modifie la valeur associée à la clé "âge"
```

Ajout d'éléments à un dictionnaire :

Pour ajouter une nouvelle paire clé-valeur à un dictionnaire, vous pouvez simplement utiliser la notation des crochets :

```
mon_dictionnaire["profession"] = "Ingénieur"  # Ajoute une nouvelle paire clé-valeur
```

Suppression d'éléments d'un dictionnaire :

Pour supprimer une paire clé-valeur d'un dictionnaire, vous pouvez utiliser le mot-clé `del` :

```
del mon_dictionnaire["ville"]  # Supprime la paire clé-valeur associée à la clé "ville"
```

Chapitre 4 : les listes, tuples et dictionnaires

Vérification de l'existence d'une clé :

Vous pouvez vérifier si une clé existe dans un dictionnaire en utilisant l'opérateur `in` :

```
if "profession" in mon_dictionnaire:
print("La clé 'profession' existe dans le dictionnaire.")
```

Fonctions intégrées pour les dictionnaires :

Python propose plusieurs fonctions et méthodes intégrées pour travailler avec les dictionnaires, notamment `keys()`, `values()`, `items()`, `get()`, `update()`, etc. Ces fonctions permettent de parcourir, d'extraire des clés, des valeurs ou des paires clé-valeur, de fusionner deux dictionnaires, etc.

Les dictionnaires sont très utilisés pour représenter des données structurées, telles que des informations de configuration et des enregistrements de base de données.

Chapitre 4 : les listes, tuples et dictionnaires

1. `**keys**()`: Cette méthode renvoie une vue sur les clés d'un dictionnaire.

```
etudiant = {"nom": "Omar", "age": 20, "matricule": "12345"}

# Utilisation de keys() pour obtenir les clés du dictionnaire
cles = etudiant.keys()

# Affichage des clés
for cle in cles:
    print(cle)
```

2. `**values**()`: Cette méthode renvoie une vue sur les valeurs d'un dictionnaire.

```
etudiant = {"nom": "Alice", "age": 22, "matricule": "54321"}

# Utilisation de values() pour obtenir les valeurs du dictionnaire
valeurs = etudiant.values()

# Affichage des valeurs
for valeur in valeurs:
    print(valeur)
```

Chapitre 4 : les listes, tuples et dictionnaires

3. `**items**()`: Cette méthode renvoie une vue sur les paires clé-valeur d'un dictionnaire.

```
etudiant = {"nom": "Bob", "age": 25, "matricule": "98765"}

# Utilisation de items() pour obtenir les paires clé-valeur du
dictionnaire
paires = etudiant.items()

# Affichage des paires clé-valeur
for cle, valeur in paires:
    print(f"{cle}: {valeur}")
```

Chapitre 4 : les listes, tuples et dictionnaires

4. `**get**()`: Cette méthode permet d'obtenir la valeur associée à une clé donnée, et elle renvoie une valeur par défaut si la clé n'existe pas.

```
etudiant = {"nom": "Caroline", "age": 18}

# Utilisation de get() pour obtenir l'âge de l'étudiant
age = etudiant.get("age", "Âge inconnu")

print(f"Âge de l'étudiant : {age}")

# Si la clé n'existe pas, la valeur par défaut est renvoyée
ville = etudiant.get("ville", "Ville inconnue")
print(f"Ville de l'étudiant : {ville}")
```

Chapitre 4 : les listes, tuples et dictionnaires

5.`**update**()`: Cette méthode permet de mettre à jour un dictionnaire avec les éléments d'un autre dictionnaire ou d'une séquence de paires clé-valeur.

```
etudiant = {"nom": "ahmed", "age": 21}
nouvelles_infos = {"matricule": "55555", "ville": "Tlemcen"}

# Utilisation de update() pour ajouter de nouvelles informations à l'étudiant
etudiant.update(nouvelles_infos)

# Affichage du dictionnaire mis à jour
print(etudiant)
```

Chapitre 4 : les listes, tuples et dictionnaires

- Exemple d'une liste de dictionnaires:

```
etudiant= {}
liste_etudiant=[]

etudiant["nom"]= "anes"
etudiant["age"]=21
liste_etudiant.append(etudiant)

etudiant["nom"]= " imene"
etudiant["age"]=21
liste_etudiant.append(etudiant)

print(liste_etudiant)
```

Chapitre 5 : Les interfaces avec Tkinter

- Tkinter, est une bibliothèque d'interface graphique (GUI) intégrée à Python. permet de créer des applications avec une interface utilisateur (IHM) de manière simple et efficace.
- Son nom provient de "Tk" (Toolkit), qui est une boîte à outils graphique, et "inter" pour interface. Tkinter est inclus avec la plupart des distributions Python, ce qui signifie qu'il est disponible par défaut lorsqu’on installe Python.

Chapitre 5 : Les interfaces avec Tkinter

Caractéristiques de Tkinter :

- Facilité d'utilisation : Tkinter est relativement facile à apprendre et à utiliser, ce qui en fait un bon choix pour les débutants en programmation d'interfaces graphiques.
- Compatibilité : Comme Tkinter est inclus avec Python, il est compatible avec la plupart des plates-formes, y compris Windows, macOS et Linux.
- Widgets : Tkinter offre une variété de widgets (éléments d'interface utilisateur) tels que des boutons, des zones de texte, des étiquettes, etc., que vous pouvez utiliser pour construire votre interface.
- Personnalisable : Vous pouvez personnaliser l'apparence des widgets et de la fenêtre selon vos besoins.

Chapitre 5 : Les interfaces avec Tkinter

- **Création de la fenêtre principale :**

Un exemple très basique de programme Tkinter :

```
import tkinter as tk

# Création de la fenêtre principale
fenetre = tk.Tk()
fenetre.title("Ma Première Application Tkinter")

# Dimensions de la fenêtre (largeur x hauteur + positionX + positionY)
fenetre.geometry("400x300+100+100")

# Lancement de la boucle principale
fenetre.mainloop()
```

Chapitre 5 : Les interfaces avec Tkinter

- Dans cet exemple, une fenêtre est créée avec le titre : « Ma Première Application Tkinter »
- Création de la fenêtre : fenetre = tk.Tk() crée la fenêtre principale de l'application.
- La méthode geometry est utilisée pour spécifier les dimensions de la fenêtre. Le format de la chaîne de géométrie est "largeur x hauteur + positionX + positionY"
- Largeur x Hauteur : La taille de la fenêtre en pixels.
- PositionX + PositionY : La position initiale de la fenêtre sur l'écran. La position est définie par rapport au coin supérieur gauche de l'écran. Dans l'exemple, la fenêtre est positionnée à 100 pixels de la gauche et 100 pixels du haut de l'écran.

Chapitre 5 : Les interfaces avec Tkinter

- L’instruction : fenetre.resizable(False, False) , empêchera la fenêtre d'être redimensionnée horizontalement ou verticalement par l'utilisateur.

Chapitre 5 : Les interfaces avec Tkinter

- **Création d'un bouton :**

```
import tkinter as tk

# Fonction appelée lorsque le bouton est cliqué
def on_button_click(message):
    fenetre.title("Action est lancée")

# Création de la fenêtre principale
fenetre = tk.Tk()
fenetre.title("Exemple de Bouton Tkinter")
fenetre.geometry("400x200+100+100")
# Création d'un bouton
bouton = tk.Button(fenetre, text="Cliquez-moi!", command=on_button_click)
bouton.pack(pady=10, padx=10)  # pady et padx ajoutent des espaces autour du bouton

# Lancement de la boucle principale
fenetre.mainloop()
```

Chapitre 5 : Les interfaces avec Tkinter

- Dans cet exemple :

✓ Nous créons une fonction on_button_click qui sera appelée lorsque le bouto est cliqué.

✓ Ensuite, nous créons la fenêtre principale avec tk.Tk() et un titre.

✓ Nous créons un bouton avec tk.Button, spécifiant le texte affiché sur le bouton avec l'argument text et la fonction à appeler lors du clic avec l'argument command.

✓ Le bouton est placé dans la fenêtre avec la méthode pack.

✓ La boucle principale fenetre.mainloop() est lancée pour afficher la fenêtre et attendre les interactions de l'utilisateur.

Chapitre 5 : Les interfaces avec Tkinter

- La méthode pack() est utilisée pour organiser le placement des widgets dans la fenêtre dans Tkinter. Si vous ne l'utilisez pas, le widget que vous avez créé (comme un bouton) peut ne pas être affiché correctement à l'intérieur de la fenêtre.

- En Tkinter, il existe d'autres méthodes pour organiser les widgets, notamment grid() et place(). Cependant, dans l'exemple précédent, nous avons utilisé pack() car c'est une méthode simple et souvent utilisée pour organiser les widgets de manière linéaire.

Chapitre 5 : Les interfaces avec Tkinter

Création d'un champ de texte :

- Pour créer un champ de texte (entry) en Tkinter, il est possible d'utiliser la classe Entry. Voici un exemple simple qui crée une fenêtre avec un champ de texte :

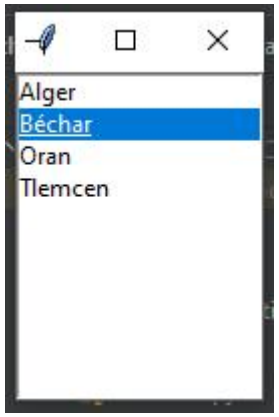
```
import tkinter as tk

def on_button_click():
    user_input = entry.get()
    label_result.config(text=f"Vous avez saisi : {user_input}")

# Création de la fenêtre principale
fenetre = tk.Tk()
fenetre.title("Champ de texte Tkinter")

# Création d'un champ de texte
entry = tk.Entry(fenetre)
entry.pack(pady=10, padx=10)

# Création d'un bouton pour récupérer le texte du champ de texte
button = tk.Button(fenetre, text="Récupérer le texte", command=on_button_click)
button.pack(pady=10, padx=10)

# Création d'un label pour afficher le résultat
label_result = tk.Label(fenetre, text="")
label_result.pack(pady=10, padx=10)

# Lancement de la boucle principale
fenetre.mainloop()
```

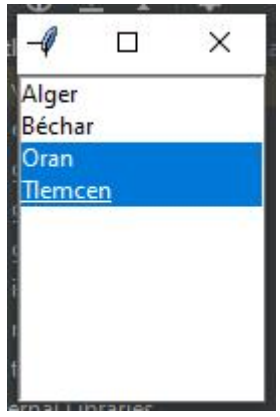
Chapitre 5 : Les interfaces avec Tkinter

- Dans cet exemple, nous avons utilisé la classe Entry pour créer un champ de texte appelé entry. Le texte entré dans ce champ peut être récupéré en utilisant la méthode get().
- La fonction on_button_click est appelée lorsque le bouton est cliqué. Elle récupère le texte du champ de texte à l'aide de entry.get() et met à jour un label (label_result) pour afficher le texte saisi.

Chapitre 5 : Les interfaces avec Tkinter

- **Création de liste de sélection**
  - **Deux types de listes:**
    - **Liste de sélection simple**
    - **Liste de sélection multiple**





Chapitre 5 : Les interfaces avec Tkinter

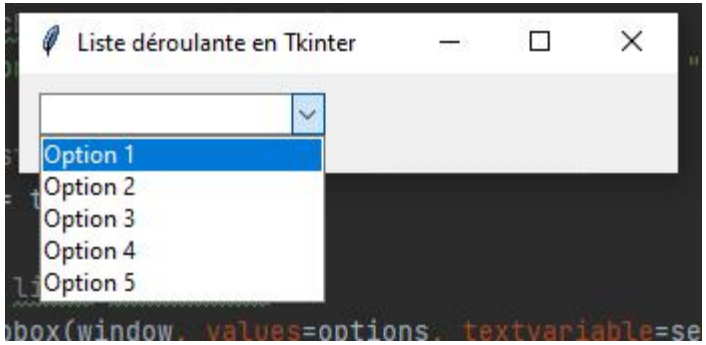
- Exemple d'une liste de sélection simple:
  - Exemple de création :

```
import tkinter as tk
def select(event):
   selected_index = listbox.curselection()
   selected_item = listbox.get(selected_index)
# Création de la fenêtre principale
window = tk.Tk()
window.title("villes")
# Création du widget Listbox
listbox = tk.Listbox(window, selectmode=tk.SINGLE)
#insertion des éléments :
listbox.insert(tk.END, "Alger" )
listbox.insert(tk.END, "Béchar" )
listbox.insert(tk.END, "Oran" )
listbox.insert(tk.END, "Tlemcen" )
# Liaison de la fonction on_select à l'événement de sélection
listbox.bind('<<ListboxSelect>>', select)
# Affichage du widget Listbox
listbox.pack()
# Démarrage de la boucle principale
```

c'est possible de créer une liste de sélection multiple, il suffit de changer selectmode en tk.MULTIPLE

Chapitre 5 : Les interfaces avec Tkinter

- **Création d'une liste déroulante (combobox)**

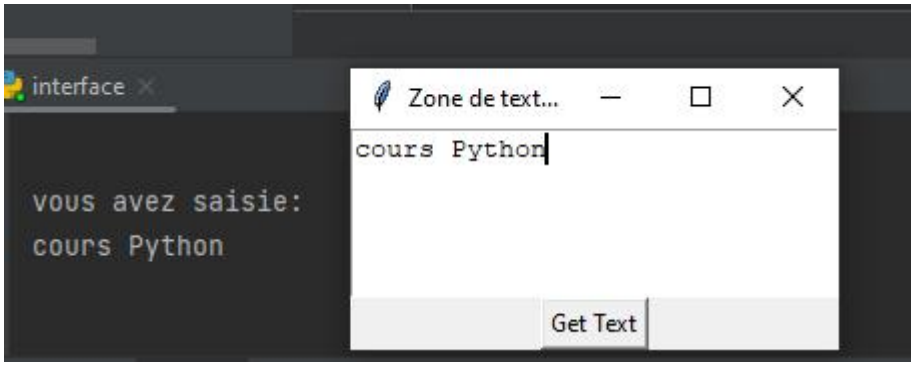



```
def on_select(event):
    selected_value = combo.get()
    print(f"Selected value: {selected_value}")

# Options à afficher dans la liste déroulante
options = ["Option 1", "Option 2", "Option 3",
"Option 4", "Option 5"]
```

```
# Création de la liste déroulante
combo = tk.Combobox(window,
values=options,
textvariable=selected_option)
combo.grid(row=0, column=0, padx=10,
pady=10)
# Configuration de l'événement de sélection
combo.bind("<<ComboboxSelected>>",
on_select)
```

Chapitre 5 : Les interfaces avec Tkinter

- Création d'une zone de texte:




```
def recuperer_text():
    # Cette fonction récupère le contenu
       de la zone de texte
    texte = text_widget.get("1.0", tk.END)
    print("vous avez saisie:")
    print(texte)
```

```
# Création de la zone de texte
text_widget = tk.Text(window, height=5,
width=30)
text_widget.pack()
# Création d'un bouton pour déclencher la
fonction recuperer_text
button = tk.Button(window, text="Get Text",
command=recuperer_text)
button.pack()
```

Chapitre 5 : Les interfaces avec Tkinter

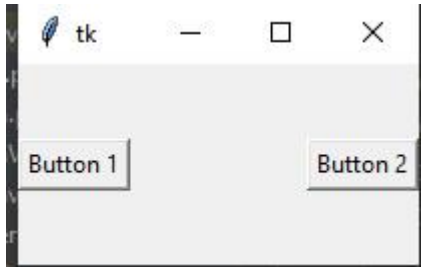
- **Méthodes de placement :**
  - Dans Tkinter, il existe différentes méthodes pour placer des boutons et d'autres objets sur une fenêtre. Les trois gestionnaires de géométrie principaux utilisés pour positionner les widgets sont pack(), grid() et place().

Chapitre 5 : Les interfaces avec Tkinter

1. pack()

- pack() organise les widgets en blocs avant de les placer dans le conteneur.
- Il prend en charge les options telles que « side » (côté de l'objet), « fill » (remplissage), « expand » (extension), etc.
- Exemple :

```
import tkinter as tk

root = tk.Tk()

button1 = tk.Button(root, text="Button 1")
button2 = tk.Button(root, text="Button 2")

button1.pack(side="left")
button2.pack(side="right")

root.mainloop()
```

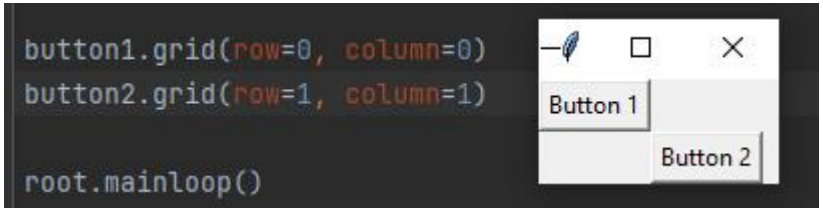


Chapitre 5 : Les interfaces avec Tkinter

2. grid() :

- grid() organise les widgets dans une grille (ligne et colonne) plutôt qu'en blocs.
- Il prend en charge les options telles que `row`, `column`, `sticky`, etc.
- Exemple :

```
import tkinter as tk

root = tk.Tk()

button1 = tk.Button(root, text="Button 1")
button2 = tk.Button(root, text="Button 2")

button1.grid(row=0, column=0)
button2.grid(row=1, column=1)

root.mainloop()
```

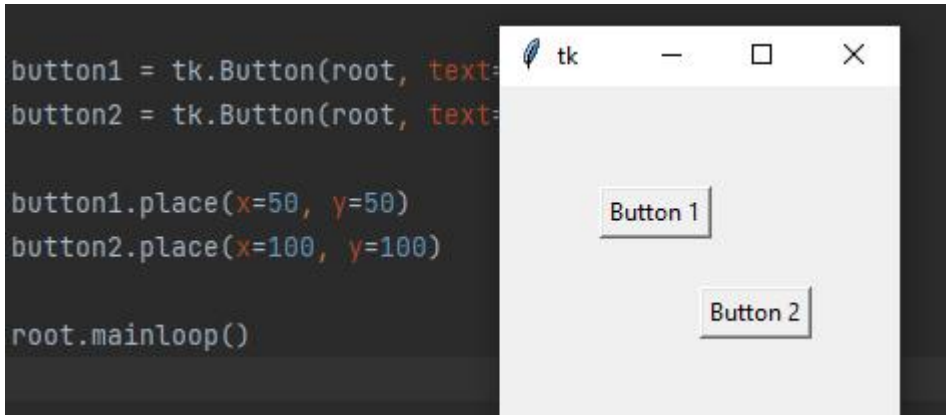


Chapitre 5 : Les interfaces avec Tkinter

3. place() :

- place() permet de placer les widgets à des coordonnées spécifiques ou en relation avec d'autres widgets.
- Il prend en charge les options telles que `x`, `y`, `relx`, `rely`, etc.
- Exemple :

```
import tkinter as tk

root = tk.Tk()

button1 = tk.Button(root, text="Button 1")
button2 = tk.Button(root, text="Button 2")

button1.place(x=50, y=50)
button2.place(x=100, y=100)

root.mainloop()
```



Chapitre 5 : Les interfaces avec Tkinter

- Un autre exemple simple utilisant relx et rely avec la méthode place() dans Tkinter. Ces options permettent de placer les widgets relativement à la taille de la fenêtre parente


```
# Placement relatif des boutons
button1.place(relx=0.2, rely=0.4)  # 20% à droite, 40% en bas
button2.place(relx=0.7, rely=0.6)  # 70% à droite, 60% en bas

root.mainloop()
```

Chapitre 5 : Les interfaces avec Tkinter

Exemple de placement de trois composantes sur une forme :

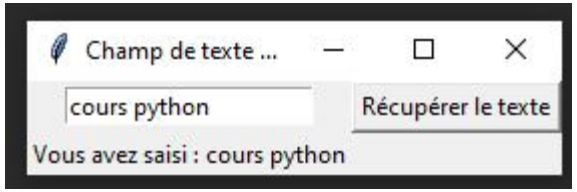
La méthode de placement est : grid

-Les composantes sont : un champs de texte dans la ligne 0 et colonne 0

-Un bouton dans la ligne 0 , colonne 1

-Un label dans la ligne 1 , colonne 0

L'affichage doit être comme celui affiché par lafigure suivante :



Chapitre 5 : Les interfaces avec Tkinter

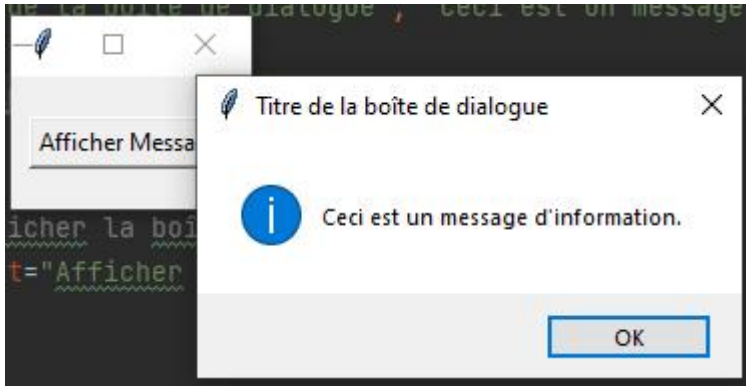
- Les boites de dialogues (messagebox)
  - En Python, les boîtes de dialogue (messagebox) sont souvent utilisées pour afficher des messages informatifs, demander des confirmations de l'utilisateur ou saisir des données. Le module «tkinter » fournit une boîte de dialogue standard appelée « messagebox »

Chapitre 5 : Les interfaces avec Tkinter

- Exemple 1: message d'information

```
import tkinter as tk
from tkinter import messagebox

# Fonction pour afficher une boîte de dialogue
def afficher_message():
    messagebox.showinfo("Titre de la boîte de dialogue", "Ceci est un message
     d'information.")

# Création de la fenêtre principale
fenetre = tk.Tk()

# Création d'un bouton pour afficher la boîte de dialogue
bouton = tk.Button(fenetre, text="Afficher Message", command=afficher_message)
bouton.pack(pady=20)

# Lancement de la boucle principale
fenetre.mainloop()
```

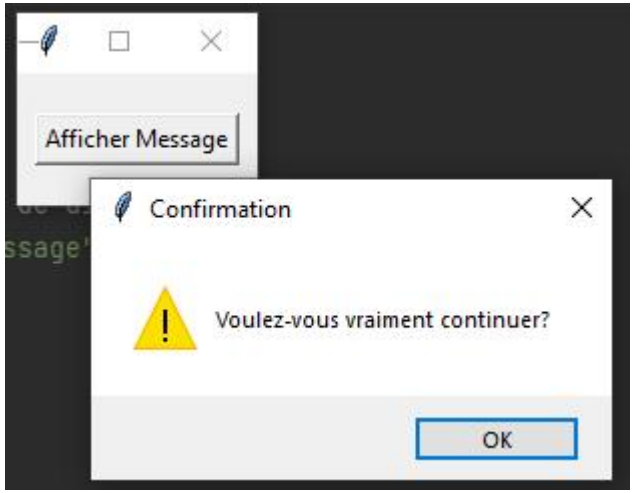
Chapitre 5 : Les interfaces avec Tkinter



Chapitre 5 : Les interfaces avec Tkinter

- Exemple 2: message d'avertissement

```
# Fonction pour afficher une boîte de dialogue
def afficher_message():
    messagebox. showwarning (" Attention", "Voulez vous
    vraiment continuer !!! " )
```

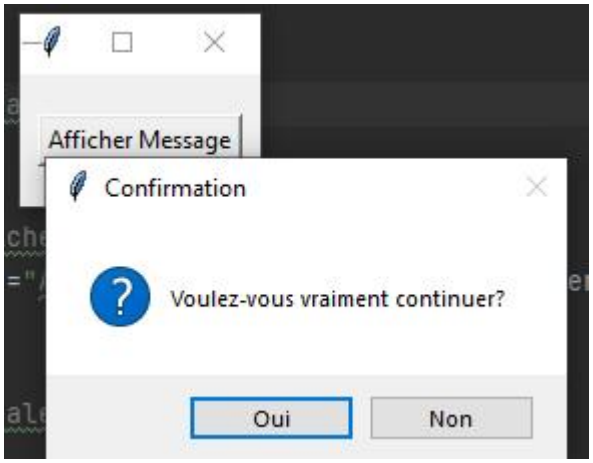


Chapitre 5 : Les interfaces avec Tkinter

- Exemple : boite de dialogue avec réponse (Exemple de askquestion)

```
def demander_confirmation():
    # askquestion renvoie 'yes' ou 'no' en fonction du bouton cliqué
    reponse = messagebox.askquestion("Confirmation", "Voulez-vous vraiment continuer?")

    # Gérer la réponse de l'utilisateur
    if reponse == 'yes':
        messagebox.showinfo("Confirmation", "Action confirmée!")
    else:
        messagebox.showinfo("Confirmation", "Action annulée.")
```

Chapitre 5 : Les interfaces avec Tkinter



Chapitre 5 : Les interfaces avec Tkinter

- **Création de fenêtres secondaires**
  - Généralement les applications graphiques sont des application multi-fenetres
  - Dans le diopo suivant, un exemple simple de création d'une deuxième fenêtre avec un bouton pour la fermer en utilisant le module « tkinter » en Python :

Chapitre 5 : Les interfaces avec Tkinter

```
import tkinter as tk
from tkinter import messagebox

def fermer_deuxieme_fenetre(deuxieme_fenetre):
    deuxieme_fenetre.destroy()

def ouvrir_deuxieme_fenetre():
    # Création de la deuxième fenêtre
    deuxieme_fenetre = tk.Tk()
    deuxieme_fenetre.title("Deuxième Fenêtre")
    # Création d'un bouton pour fermer la deuxième fenêtre
    bouton_fermer = tk.Button(deuxieme_fenetre, text="Fermer la Deuxième Fenêtre",  command=lambda:
        fermer_deuxieme_fenetre(deuxieme_fenetre))
    bouton_fermer.pack(pady=20)
# Création de la fenêtre principale
fenetre = tk.Tk()
fenetre.title("Fenêtre Principale")
# Création d'un bouton pour ouvrir la deuxième fenêtre
bouton_ouvrir = tk.Button(fenetre, text="Ouvrir la Deuxième Fenêtre", command=ouvrir_deuxieme_fenetre)
bouton_ouvrir.pack(pady=20)
# Lancement de la boucle principale
fenetre.mainloop()
```

Chapitre 5 : Les interfaces avec Tkinter

- Dans cet exemple, une fenêtre principale est créée avec un bouton. Lorsque ce bouton est cliqué, la fonction « ouvrir_deuxieme_fenetre » est appelée, créant ainsi une deuxième fenêtre contenant un bouton, lorsqu'il est cliqué, appelle la fonction « fermer_deuxieme_fenetre » pour détruire la deuxième fenêtre.

Chapitre 6- La gestion des exceptions

- La gestion des exceptions en Python est une technique utilisée pour gérer les erreurs et les situations exceptionnelles qui pourraient survenir pendant l'exécution d'un programme. Elle permet de détecter, signaler et traiter les erreurs de manière à éviter que le programme ne s'arrête de manière inattendue.

Chapitre 6- La gestion des exceptions

1. Le bloc « try » ,« except » et « finally » :

   - Le bloc « try » contient le code qui pourrait générer une exception.
   - Le bloc « except » spécifie comment gérer une exception particulière.
   - Le bloc « finally » est exécuté qu'il y ait eu une exception ou non.

Chapitre 6- La gestion des exceptions

- Exemple :

```
try:
    # Code susceptible de générer une exception
    result = 10 / 0
except Exception as e:
    # Code à exécuter en cas d'autres exceptions
    print(f"Une erreur s'est produite : {e}")
finally:
    # Code à exécuter indépendamment de l'existence ou non d'une exception
    print("Fin de l'exécution du bloc try-except")
```

Le message suivant sera affiché:

Une erreur s'est produite : division by zero

Fin de l'exécution du bloc try-except

Chapitre 6- La gestion des exceptions

- Exemple 2 :

```
try:
    # Code susceptible de générer une exception
    result = 10 / 0
except ZeroDivisionError:
    # Code à exécuter en cas d'exception spécifique (ici, division par zéro)
    print("Erreur : division par zéro")
except Exception as e:
    # Code à exécuter en cas d'autres exceptions
    print(f"Une erreur s'est produite : {e}")
finally:
    # Code à exécuter indépendamment de l'existence ou non d'une exception
    print("Fin de l'exécution du bloc try-except")
```

Chapitre 6- La gestion des exceptions

- Le message suivant sera affiché :

  Erreur : division par zéro

  Fin de l'exécution du bloc try-except

Chapitre 6- La gestion des exceptions

- 2. Gérer plusieurs exceptions :
  - C’est possible de spécifier plusieurs blocs `except` pour gérer différents types d'exceptions.

Chapitre 6- La gestion des exceptions

- Exemple :

```
try:
    value = int(input("Entrez un nombre : "))
    result = 10 / value
except ValueError:
    print("Erreur : veuillez entrer un nombre valide.")
except ZeroDivisionError:
    print("Erreur : division par zéro")
except Exception as e:
    print(f"Une erreur s'est produite : {e}")
```

Chapitre 6- La gestion des exceptions

- 3. Utiliser l'instruction « else » :
  - L'instruction « else » est exécutée si aucune exception n'est levée.

Chapitre 6- La gestion des exceptions

- Exemple :

```
try:
    value = int(input("Entrez un nombre : "))
    result = 10 / value
except ValueError:
    print("Erreur : veuillez entrer un nombre valide.")
except ZeroDivisionError:
    print("Erreur : division par zéro")
else:
    print(f"Le résultat est : {result}")
```

Chapitre 6- La gestion des exceptions

- 4.Lever des exceptions :
  - c’est possible également de lever une exception en utilisant l'instruction « raise ».

Exemple:

```
try:
    age = int(input("Entrez votre âge : "))
    if age < 0:
        raise ValueError("L'âge ne peut pas être négatif.")
except ValueError as ve:
    print(f"Erreur : {ve}")
```

Chapitre 6- La gestion des exceptions

- Il existe d’autres erreurs possible a gérer
  - **TypeError :** Se produit lorsqu'une opération est effectuée sur un type de données incorrect.

```
try:
    result = int("abc")
except TypeError:
    print("Erreur de type : la conversion a échoué.")
```

Chapitre 6- La gestion des exceptions

- **FileNotFoundError :** Se produit lorsqu'une tentative est faite pour ouvrir un fichier qui n'existe pas

```
try:
    with open("fichier_inexistant.txt", "r") as file:
        content = file.read()
except FileNotFoundError:
    print("Erreur de fichier non trouvé : le fichier spécifié n'existe pas.")
```

Chapitre 6- La gestion des exceptions

- **IndexError :** Se produit lorsqu'un indice de liste est hors limites.

```
try:
    my_list = [1, 2, 3]
    value = my_list[10]
except IndexError:
    print("Erreur d'indice : l'indice spécifié est hors limites.")
```

Chapitre 6- La gestion des exceptions

- **KeyError :** Se produit lorsqu'une clé est recherchée dans un dictionnaire et que la clé n'est pas présente.

```
try:
    my_dict = {"a": 1, "b": 2}
    value = my_dict["c"]
except KeyError:
    print("Erreur de clé : la clé spécifiée n'existe pas dans le dictionnaire.")
```