Introduction à Python

Carlos Gonzalez
carlos.gonzalez@univ-ubs.fr

Langages et outils Informatiques - UEC1601 Python

Université Bretagne Sud - Lorient

8 janvier 2024

Fiche de l'Unité d'Enseignement (UE)

- UE Modélisation info et numériques - UEC1601
- Volume horaire de 16h
  - Cours : 4h
  - TP : 12h
- Déroulement et consignes
  - Cours détaillé sur la plateforme pédagogique (Moodle)
  - Partage des documents uniquement via la plateforme pédagogique
  - Contrôle des connaissances : Examen Final Python 1h et Pandas 1h

Sommaire

Sommaire

Le langage de programmation Python

- Créé en 1989 par Guido van Rossum
- Multiplateforme et gratuit : Python est disponible sur différentes plates-formes, notamment Windows, macOS et Linux. De plus, il est distribué sous une licence open source.
- Version actuelle : La dernière version stable de Python est la version 3.12

Comment la mémoire est-elle gérée en Python ?

La mémoire est gérée en Python de la manière suivante :

- La gestion de la mémoire en Python est prise en charge par l'espace de mémoire privé de Python. Tous les objets et structures de données Python sont situés dans un espace de mémoire privé.
- L'allocation de l'espace mémoire pour les objets Python est effectuée par le gestionnaire de mémoire de Python.
- Python dispose également d'un collecteur intégré, qui recycle toute la mémoire inutilisée pour qu'elle puisse être mise à disposition de l'espace de mémoire.

Comparaison entre Python et Java

Python

- Syntaxe concise et lisible
- Interprété, pas de compilation explicite
- Orienté objet et fonctionnel
- Grande communauté et bibliothèques
- Utilisé pour le développement web, l'analyse de données, l'automatisation

Java

- Syntaxe plus verbeuse
- Compilé en bytecode
- Orienté objet
- Vaste écosystème et outils de développement
- Utilisé pour les applications d'entreprise, Android, systèmes embarqués

Comparaison entre Python et C++

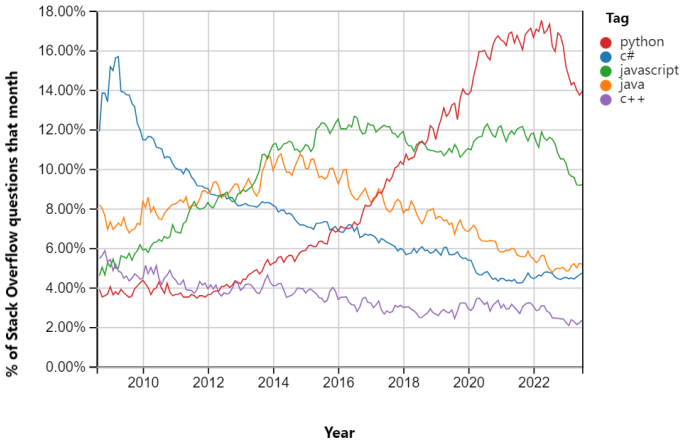
Python

- Syntaxe simple et lisible
- Langage de script, pas de compilation explicite
- Gestion automatique de la mémoire (garbage collection)
- Moins performant que C++
- Convient à la programmation rapide et au prototypage

C++

- Syntaxe complexe et rigide
- Compilation nécessaire pour générer un exécutable
- Contrôle manuel de la mémoire
- Très performant, utilisé pour les applications hautes performances
- Utilisé dans les jeux, les logiciels système, les systèmes embarqués

Tendances en Langages de Programmation



Source : Stack Overflow

Roadmap de Python pour la Science des Données

- **Les Bases :**
  - Introduction à Python : syntaxe, variables, opérations
  - Structures de données : listes, tuples, dictionnaires
  - Contrôle de flux : conditions, boucles
  - Fonctions et modules
- **Avancé :**
  - Manipulation de données avec NumPy et Pandas
  - Visualisation avec Matplotlib et Seaborn
  - Statistiques et analyse exploratoire
  - Introduction à l'apprentissage automatique avec Scikit-Learn
- **Frameworks Web :**
  - Flask : développement d'applications web légères
  - Django : framework web complet pour applications robustes
- **Administration de Packages :**
  - pip : gestion des packages Python
  - conda : gestion des packages pour l'environnement de données
  - Création et gestion d'environnements virtuels

Utilisations de Python

Python est utilisé dans divers domaines, tels que :

- Programmation web
- Analyse de données
- Intelligence artificielle
- Automatisation de tâches
- Et bien d'autres domaines

Sa syntaxe claire et lisible en fait un excellent choix pour les débutants en programmation.

Installation de Python sur Windows

Pour installer Python sur Windows, suivez ces étapes :

1. Rendez-vous sur le site officiel de Python : `https://www.python.org/downloads/windows/`
2. Téléchargez le dernier installateur pour Windows.
3. Exécutez l'installateur et suivez les instructions à l'écran.
4. Cochez l'option "Add Python to PATH" pendant l'installation pour faciliter l'utilisation de Python dans l'invite de commande.
5. Une fois l'installation terminée, vous pouvez ouvrir l'invite de commande et exécuter `python -version` pour vérifier si Python est correctement installé.

Installation de Python sur Linux

L'installation de Python sur la plupart des distributions Linux est assez simple :

1. Ouvrez le terminal.
2. Tapez la commande suivante pour installer Python :
   `sudo apt-get install python3`
3. Une fois l'installation terminée, vous pouvez exécuter `python3 -version` dans le terminal pour vérifier si Python est correctement installé.
4. `Vérifier la version de Linux avec lsb_release -a`

Résultat attendu

```
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 20.04.6 LTS
Release:        20.04
Codename:       focal
```

Installation de Différentes Versions de Python sur Linux

Voici comment installer différentes versions de Python :

Python 3

- Sur la plupart des distributions :
  `sudo apt-get install python3`
- Pour installer Python 3.x :
  `sudo apt-get install python3.x`

Python 2

Python 2 n'est plus pris en charge, mais si nécessaire :
`sudo apt-get install python2`

Gestionnaire de versions (optionnel)

Vous pouvez également utiliser des gestionnaires de versions comme `pyenv` ou `conda` pour gérer différentes versions de Python.

Installation de Python sur macOS

Pour installer Python sur macOS, suivez ces étapes :

1. Rendez-vous sur le site officiel de Python : `https://www.python.org/downloads/macos/`
2. Téléchargez le dernier installateur pour macOS.
3. Exécutez l'installateur et suivez les instructions à l'écran.
4. Une fois l'installation terminée, ouvrez le terminal et exécutez `python3` pour vérifier si Python est correctement installé.

Gestionnaire de paquets

- Anaconda et Miniconda
  - Utilisé pour créer des environnements virtuels et installer des packages,
  - Prend en charge des packages non Python et la résolution des dépendances.
  - Commandes d'installation :
    `conda install package-name`
    `conda create -n env-name package-name`
  - Exemples :
    `conda install numpy`
    `conda create -n myenv numpy`
- Pip Python
  - Utilisé pour installer des packages à partir du Python Package Index (PyPI),
  - Commande d'installation : `pip install package-name`,
  - Exemple : `pip install numpy`

Gestionnaire de paquets

- Anaconda et Miniconda
  - Utilisé pour créer des environnements virtuels et installer des packages,
  - Prend en charge des packages non Python et la résolution des dépendances.
  - Commandes d'installation :
    `conda install package-name`
    `conda create -n env-name package-name`
  - Exemples :
    `conda install numpy`
    `conda create -n myenv numpy`
- Pip Python
  - Utilisé pour installer des packages à partir du Python Package Index (PyPI),
  - Commande d'installation : `pip install package-name`,
  - Exemple : `pip install numpy`

Par où commencer ?

```
Microsoft Windows [Version 10.0.22621.2134]
(c) Microsoft Corporation. All rights reserved.

(base) C:\Users\Krlos>python
Python 3.11.4 | packaged by Anaconda, Inc. | (main, Jul  5 2023, 13:38:37)
[MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Éditeurs de Texte pour le Développement Python

- **Visual Studio Code (VS Code) :**
  - Éditeur de texte léger et extensible de Microsoft.
  - Prise en charge native de Python avec des extensions pour la coloration syntaxique, l'intégration Git, le débogage, etc. nombreuses extensions Python spécifiques.
- **PyCharm :**
  - Environnement de développement intégré (IDE) par JetBrains.
  - Fournit des fonctionnalités avancées pour le développement Python, y compris le débogage, la gestion de projet, les tests unitaires et l'analyse de code.
- **Notepad++ :**
  - Éditeur de texte populaire avec une interface conviviale.
  - Peut être étendu avec des plugins pour la coloration syntaxique Python.
- **Jupyter :**
  - Application web interactive pour créer et partager des documents qui contiennent du code Python, des visualisations et des explications.
  - Idéal pour l'analyse de données exploratoire et l'apprentissage interactif.

Sommaire

Type de Donnée Integer

Le type de données **entier** est utilisé pour stocker des nombres entiers, tels que :

- 1
- 2
- 3
- -10
- 1000

En Python, vous pouvez définir un entier en utilisant la syntaxe suivante :

Exemple

```
[language=Python]
my_integer = 42
negative_integer = -10
```

Type de Donnée Float

Le type de données **Float** (virgule flottante) est utilisé pour stocker des nombres à virgule flottante, tels que :

- 1.69
- 3.14
- -0.5
- 2.0

En Python, vous pouvez définir un nombre à virgule flottante de la manière suivante :

Exemple

```
[language=Python]
my_float = 3.14
negative_float = -0.5
```

Type de Donnée String

Le type de données **String** (chaîne de caractères) est utilisé pour stocker des suites de caractères, comme des mots ou des phrases.
En Python, vous pouvez définir une chaîne de caractères en utilisant des guillemets simples ou doubles :

Exemple

```
[language=Python]
my_string = 'Hello, World!'
another_string = "Python is fun"
```

Type de Donnée Boolean

Le type de données **Boolean** (booléen) est utilisé pour stocker des valeurs de vérité, soit **True** (vrai) ou **False** (faux).
En Python, les opérations de comparaison et les conditions renvoient souvent des valeurs booléennes :

Exemple

```
[language=Python]
is_true = True
is_false = False

x = 5
y = 10
result = x < y  # Cela sera True
```

Exemples d'Utilisation de la Fonction `print`

La fonction `print` est utilisée pour afficher du texte et des données à la sortie standard (généralement la console). Voici quelques exemples d'utilisation :

Exemple 1 : Afficher une Chaîne de Caractères

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

Résultat : Hello, World !

Exemple 2 : Afficher des Variables

```
name = "Alice"
age = 30
print("Nom:", name, "Âge:", age)
```

Résultat : Nom : Alice Âge : 30

Exemples d'Utilisation de la Fonction `print`

Exemple 3 : Formatage de Chaînes

```
x = 5
y = 10
print("La valeur de x est {} et y est {}".format(x, y))
```

Résultat : La valeur de x est 5 et y est 10

Exemple 4 : Utilisation de Séparateurs

```
print("Un", "Deux", "Trois", sep="-")
```

Résultat : Un-Deux-Trois

Les f-strings en Python

Les f-strings, ou format strings, sont une manière concise et puissante de formater et d'insérer des valeurs dans des chaînes de caractères en Python.

- Les f-strings sont définies en utilisant la lettre "f" ou "F" devant la chaîne.
- Les expressions Python à l'intérieur des f-strings sont encadrées par des accolades .
- Les valeurs des expressions sont automatiquement insérées dans la chaîne.

Exemple

```
name = "Toto"
age = 30
formatted_string = f"Bonjour, je m'appelle {name} et j'ai {age} ans."
print(formatted_string)

print(f"Bonjour, je m'appelle {name} et j'ai {age} ans.")
```

Résultat : Bonjour, je m'appelle Toto et j'ai 30 ans.

Méthodes de Manipulation de Chaînes de Caractères

En Python, il existe plusieurs méthodes pour manipuler des chaînes de caractères. Voici quelques-unes des méthodes courantes :

- ‘upper()‘ : Convertit une chaîne en majuscules.
- ‘lower()‘ : Convertit une chaîne en minuscules.
- ‘strip()‘ : Supprime les espaces inutiles au début et à la fin d'une chaîne.
- ‘replace()‘ : Remplace un sous-ensemble de caractères par un autre dans une chaîne.
- ‘split()‘ : Divise une chaîne en une liste de sous-chaînes en fonction d'un séparateur.

Exemples de Méthodes de Manipulation de Chaînes

Voici des exemples d'utilisation de ces méthodes :

```
texte = "Bonjour, Monde !"
majuscules = texte.upper()
minuscules = texte.lower()
stripped = "   texte avec espaces inutiles   ".strip()
remplace = texte.replace("Bonjour", "Hello")
mots = texte.split(",")
```

- ‘majuscules‘ contient "BONJOUR, MONDE !"
- ‘minuscules‘ contient "bonjour, monde !"
- ‘stripped‘ contient "texte avec espaces inutiles"
- ‘remplace‘ contient "Hello, Monde !"
- ‘mots‘ contient ["Bonjour", " Monde !"]

Caractères d'Échappement

En Python, les **caractères d'échappement** (escape characters) sont utilisés pour représenter des caractères spéciaux dans les chaînes de caractères. Ils commencent généralement par un antislash (\).
Voici quelques caractères d'échappement couramment utilisés en Python :

- `\n` : Nouvelle ligne (newline)
- `\'` : Guillemet simple (single quote)
- `\"` : Guillemet double (double quote)
- `\\` : Antislash (backslash)
- `\t` : Tabulation (tab)

Exemple d'Utilisation de Caractères d'Échappement en Python

```
# Utilisation de caractères d'échappement
print("Ligne 1\n Ligne 2")
print(' C\'est une guillemet simple : ')
print("Il a dit : \"Bonjour, Python!\"")
print("Un chemin de fichier : C:\\chemin\\vers\\fichier.txt")
print("Texte avec une\ttabulation")
```

Résultat :
Ligne 1
Ligne 2
C'est une guillemet simple :
Il a dit : "Bonjour, Python !"
Un chemin de fichier : `C:\chemin\vers\fichier.txt`
Texte avec une	tabulation
Inclure des caractères spéciaux dans les chaînes de caractères de manière contrôlée.

Sommaire

Les Opérations Numériques

En Python, vous pouvez effectuer plusieurs opérations numériques de base. Voici quelques-unes des opérations les plus courantes :

- **Addition (+) :** Additionne deux nombres,
- **Soustraction (-) :** Soustrait le deuxième nombre du premier,
- **Multiplication (*) :** Multiplie deux nombres,
- **Division (/) :** Divise le premier nombre par le deuxième.
- **Division Entière (//) :** Divise le premier nombre par le deuxième et renvoie la partie entière du quotient,
- **Modulo (%) :** Renvoie le reste de la division du premier nombre par le deuxième,
- **Exponentiation (** ou pow) :** Élève le premier nombre à la puissance du deuxième.

Les Opérations Numériques

```
a = 10
b = 3

addition = a + b
subtraction = a - b
multiplication = a * b
division = a / b
modulo = a % b
exponentiation = a ** b

print("Addition :", addition)
print("Soustraction :", subtraction)
print("Multiplication :", multiplication)
print("Division :", division)
```

Les Opérateurs

En Python, les opérateurs de comparaison sont utilisés pour comparer des valeurs et renvoyer des valeurs booléennes (True ou False) en fonction de la comparaison. Voici quelques-uns des opérateurs de comparaison les plus couramment utilisés :

- Opérateurs de comparaison :
  - $<$: inférieur
  - $<=$: inférieur ou égal
  - $>$: supérieur
  - $>=$: supérieur ou égal
  - $==$: égal
  - $!=$: différent
- Opérateurs logiques :
  - **and** : Et logique
  - **or** : Ou logique
  - **not** : Non logique

L'opérateur **et** retourne True si toutes les expressions booléennes qu'il combine sont vraies, L'opérateur **ou** retourne True si au moins l'une des expressions booléennes est vraie.

Les Opérateurs

En Python, les opérateurs de comparaison sont utilisés pour comparer des valeurs et renvoyer des valeurs booléennes (True ou False) en fonction de la comparaison. Voici quelques-uns des opérateurs de comparaison les plus couramment utilisés :

- Opérateurs de comparaison :
  - $<$: inférieur
  - $<=$: inférieur ou égal
  - $>$: supérieur
  - $>=$: supérieur ou égal
  - $==$: égal
  - $!=$: différent
- Opérateurs logiques :
  - **and** : Et logique
  - **or** : Ou logique
  - **not** : Non logique

L'opérateur **et** retourne True si toutes les expressions booléennes qu'il combine sont vraies, L'opérateur **ou** retourne True si au moins l'une des expressions booléennes est vraie.

Les Opérateurs

- Opérateurs d'identité :
  - **is :** Est identique à
  - **is not :** N'est pas identique à
- Opérateurs d'affectation :
  - = : Affectation
  - += : Addition et affectation
  - -= : Soustraction et affectation
  - *= : Multiplication et affectation
  - /= : Division et affectation
  - //= : Division entière et affectation
  - %= : Modulo et affectation
  - **= : Exponentiation et affectation

Les Opérateurs

- Opérateurs d'identité :
  - **is** : Est identique à
  - **is not** : N'est pas identique à
- Opérateurs d'affectation :
  - = : Affectation
  - += : Addition et affectation
  - -= : Soustraction et affectation
  - *= : Multiplication et affectation
  - /= : Division et affectation
  - //= : Division entière et affectation
  - %= : Modulo et affectation
  - **= : Exponentiation et affectation

Exemples d'Utilisation des Opérateurs de Comparaison

Voici quelques exemples d'utilisation des opérateurs de comparaison en Python :

Exemples

```
x = 5
y = 10

print(x < y)        # Résultat : True
print(x <= y)       # Résultat : True
print(x > y)        # Résultat : False
print(x >= y)       # Résultat : False
print(x == y)       # Résultat : False
print(x != y)       # Résultat : True
print(x is y)       # Résultat : False
print(x is not y)   # Résultat : True
```

Saisie Utilisateur (User Input) en Python

En Python, vous pouvez obtenir des données entrées par l'utilisateur à l'aide de la fonction `input`. Cette fonction permet à l'utilisateur de saisir des valeurs à partir du clavier.

Exemple

```
name = input("Entrez votre nom : ")
age = input("Entrez votre âge : ")

print("Bonjour,", name, "Vous avez", age, "ans.")
```

Les valeurs saisies seront stockées dans les variables `name` et `age`, puis affichées à l'écran.

Sommaire

Boucle `if` en Python

La boucle `if` est une structure de contrôle qui permet d'exécuter un bloc de code si une condition est vraie. Voici sa syntaxe générale :

```
if condition:
    # Code à exécuter si la condition est vraie
```

La condition est une expression qui renvoie une valeur booléenne (True ou False). Si la condition est vraie, le code à l'intérieur du bloc `if` est exécuté.
La boucle `if` peut également être accompagnée d'une clause `else` pour exécuter un autre bloc de code si la condition est fausse.

Boucle `for` en Python

La boucle `for` est utilisée pour itérer sur une séquence (comme une liste, un tuple ou une chaîne de caractères) et exécuter un bloc de code pour chaque élément de la séquence. Voici sa syntaxe générale :

```
for variable in sequence:
    # Code à exécuter pour chaque élément de la séquence


for i in range(1, 10, 2):  # Initialisation, condition et itération
    print("This is iteration", i)
```

L'exemple de boucle utilise range(1, 10, 2) pour démarrer à 1 et incrémenter de 2 à chaque itération jusqu'à atteindre ou dépasser 10.
La boucle `for` est utile lorsque vous savez combien de fois vous voulez itérer ou lorsque vous devez effectuer une action pour chaque élément d'une séquence.

Boucle `while` en Python

La boucle `while` est utilisée pour exécuter un bloc de code tant qu'une condition est vraie. Voici sa syntaxe générale :

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

```
i = 1  # Initialisation
while i < 6:  # Condition
    print("This is iteration", i)
    i += 1  # Itération
```

La condition est une expression qui renvoie une valeur booléenne. Tant que la condition est vraie, le code à l'intérieur du bloc `while` est exécuté en boucle.
La boucle `while` est utile lorsque vous ne savez pas combien de fois vous devez itérer à l'avance, mais vous itérez jusqu'à ce que la condition devienne fausse.

Accès au jeu Command & Conquer en ligne

```
import time
# Informations d'identification pour Command & Conquer en ligne
nom_utilisateur = 'kylie'
mot_de_passe = 'motdepassesecret'

# Saisie des informations d'identification
nom_utilisateur_saisi = input('Nom d\'utilisateur : ')
mot_de_passe_saisi = input('Mot de passe : ')

# Vérification des informations d'identification
if nom_utilisateur_saisi == nom_utilisateur and mot_de_passe_saisi == mot_de_passe:
    print('Accès autorisé')
    time.sleep(5)
    print('Vous avez maintenant accès au jeu Command & Conquer en ligne.')
elif nom_utilisateur_saisi == nom_utilisateur and mot_de_passe_saisi != mot_de_passe:
    print('Mot de passe incorrect')
elif nom_utilisateur_saisi != nom_utilisateur and mot_de_passe_saisi == mot_de_passe:
    print('Nom d\'utilisateur incorrect')
else:
    print('Veuillez vérifier les deux champs...')
```

Comparaison des Boucles if, for et while

Considérons un exemple : deviner un nombre mystère entre 1 et 10. Nous allons examiner comment les boucles if, for et while peuvent être utilisées pour gérer cette situation.

- **Boucle** `if` : Utilisée pour vérifier si la supposition de l'utilisateur est correcte.
- **Boucle** `for` : Utilisée pour donner des indices à l'utilisateur, limité à un certain nombre d'essais.
- **Boucle** `while` : Utilisée pour permettre à l'utilisateur de deviner jusqu'à ce qu'il trouve la réponse.

L'exemple sera illustré en code dans la diapositive suivante.

Boucle `if` : Deviner le Nombre Mystère

```
nombre_mystere = 7
supposition = int(input("Devinez le nombre mystère entre 1 et 10 : "))

if supposition == nombre_mystere:
    print("Bravo ! Vous avez deviné le nombre mystère.")
else:
    print("Dommage ! Ce n'est pas le nombre mystère.")
```

Boucle `for` : Deviner le Nombre Mystère

```
nombre_mystere = 7

for essai in range(3):
    supposition = int(input("Devinez le nombre mystère entre 1 et 10 : "))

    if supposition == nombre_mystere:
        print("Bravo ! Vous avez deviné le nombre mystère.")
        break
    else:
        print("Ce n'est pas le nombre mystère. Essayez encore.")

print("Fin des essais.")
```

Boucle `while` : Deviner le Nombre Mystère

```
nombre_mystere = 7
essai = 0

while essai < 3:
    supposition = int(input("Devinez le nombre mystère entre 1 et 10 : "))

    if supposition == nombre_mystere:
        print("Bravo ! Vous avez deviné le nombre mystère.")
        break
    else:
        print("Ce n'est pas le nombre mystère. Essayez encore.")
        essai += 1

print("Fin des essais.")
```

Commentaires en Python

- Les commentaires à une ligne commencent par le symbole #.
- Ils peuvent être placés à la fin d'une ligne de code ou sur une ligne distincte.
- Les commentaires aident à expliquer la logique du code.
- Les commentaires multilignes sont placés entre trois guillemets simples ou doubles.

Exemple :

```
# Ceci est un commentaire à une ligne
print("Bonjour, Monde!")

print("Salut") # Commentaire à la fin de la ligne

'''
Ceci est un commentaire multiligne.
Il peut contenir plusieurs lignes de texte
et est ignoré par l'interpréteur Python.
'''
```

Manipulation de Chaînes de Caractères en Python

- Accès par Index : Chaque caractère d'une chaîne peut être accédé en utilisant son index (position).
- Découpage (Slicing) : Une sous-séquence de caractères peut être extraite à l'aide de la notation de découpe.
- Méthodes de Comptage : Méthodes pour compter des occurrences spécifiques dans une chaîne.

```
message = "Bonjour, Monde!"
# Accès par Index
print(message[0])  # Affiche : 'B'

# Découpage
print(message[3:9])  # Affiche : 'jour, '

# Méthode len() pour la longueur de la chaîne
longueur = len(message)
print("Longueur de la chaîne:", longueur)
# Méthode count() pour compter les occurrences
compteur_o = message.count('o')
print("Occurrences de 'o':", compteur_o)
```

Accéder aux Caractères par Numéro d'Index Négatif

Pour localiser un élément vers la fin d'une chaîne, nous pouvons utiliser des indices négatifs en commençant par -1 pour le dernier caractère.
Exemple avec la chaîne "Bonjour Monde" :

B	o	n	j	o	u	r		m	o	n	d	e
-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
chaine = "Bonjour Monde"

premier_caractere = chaine[0]
dernier_caractere = chaine[-1]
avant_dernier_caractere = chaine[-2]

print("Premier caractère:", premier_caractere)  # Affiche : 'B'
print("Dernier caractère:", dernier_caractere)  # Affiche : 'e'
print("Avant-dernier caractère:", avant_dernier_caractere)  # Affiche : 'd'
```

Manipulation de Chaînes de Caractères en Python

- Accès par Index : Chaque caractère d'une chaîne peut être accédé en utilisant son index (position).
- Découpage (Slicing) : Une sous-séquence de caractères peut être extraite à l'aide de la notation de découpe.
- Méthodes de Comptage : Méthodes pour compter des occurrences spécifiques dans une chaîne.

```
message = "Bonjour, Monde!"
# Accès par Index
print(message[0])  # Affiche : 'B'

# Découpage
print(message[3:9])  # Affiche : 'jour, '

# Méthode len() pour la longueur de la chaîne
longueur = len(message)
print("Longueur de la chaîne:", longueur)
# Méthode count() pour compter les occurrences
compteur_o = message.count('o')
print("Occurrences de 'o':", compteur_o)
```

Sommaire

Conversion d'un Entier en Flottant

- En Python, il est possible de convertir un entier en un flottant en utilisant la fonction float().
- Exemple :

```
entier = 42
flottant = float(entier)
```

- Dans cet exemple, la valeur de `entier` (42) est convertie en un flottant, et la variable `flottant` contient maintenant 42.0.
- Cette conversion peut être utile lorsque vous avez besoin de réaliser des opérations mathématiques qui impliquent des nombres à virgule flottante.

Conversion de Flottant en Entier

- Pour convertir un flottant en entier, utilisez la fonction `int()`.
- Exemple :
  - `flottant = 3.14`
  - `entier = int(flottant)`
- `entier` contient maintenant la valeur 3, car la partie décimale de 3.14 est tronquée.

Conversion de Nombre en Chaîne de Caractères

- Pour convertir un nombre en une chaîne de caractères, utilisez la fonction `str()`.
- Exemple :
  - `nombre = 42`
  - `chaine = str(nombre)`
- `chaine` contient maintenant la chaîne de caractères "42".

Conversion de Chaîne de Caractères en Nombre

- Pour convertir une chaîne de caractères en nombre (entier ou flottant), utilisez les fonctions `int()` ou `float()`.
- Exemple :
  - `chaine_entier = "42"`
  - `chaine_flottant = "3.14"`
  - `entier = int(chaine_entier)`
  - `flottant = float(chaine_flottant)`
- `entier` contient maintenant la valeur 42, et `flottant` contient la valeur 3.14.

Conversion Implicite

- Python effectue automatiquement des conversions implicites lors des opérations entre différents types de données.
- Exemple :
  - `entier = 5`
  - `flottant = 2.5`
  - `resultat = entier + flottant # L'entier est converti en flottant pour l'addition`
- `resultat` contient la valeur 7.5.

Tableau de Conversion de Types

De/Vers	Nombre entier	Nombre flottant	Chaîne de caractères
Nombre entier		`float()`	`str()`
Nombre flottant	`int()`		`str()`
Chaîne de caractères	`int()`	`float()`	

Les fonctions natives mathématiques

Fonctions très utiles comme le calcul du maximum ou de la valeur absolue.

- `abs(-10)` → `10` : calcule la valeur absolue.
- `max(8, 15)` → `15` : calcule le maximum entre deux nombres.
- `max(12, 9, 21)` → `21` : calcule le maximum entre trois nombres.
- `min(7, 14)` → `7` : calcule le minimum entre deux nombres.
- `min(30, 18, 25)` → `18` : calcule le minimum entre trois nombres.
- `round(3.14159)` → `3` : arrondit à l'entier le plus proche.
- `round(2.71828, 2)` → `2.72` : arrondit à deux décimales.

Les fonctions natives mathématiques

Fonctions très utiles comme le calcul du maximum ou de la valeur absolue.

- `abs(-10)` → `10` : calcule la valeur absolue.
- `max(8, 15)` → `15` : calcule le maximum entre deux nombres.
- `max(12, 9, 21)` → `21` : calcule le maximum entre trois nombres.
- `min(7, 14)` → `7` : calcule le minimum entre deux nombres.
- `min(30, 18, 25)` → `18` : calcule le minimum entre trois nombres.
- `round(3.14159)` → `3` : arrondit à l'entier le plus proche.
- `round(2.71828, 2)` → `2.72` : arrondit à deux décimales.

Les fonctions natives mathématiques

Fonctions très utiles comme le calcul du maximum ou de la valeur absolue.

- `abs(-10)` → `10` : calcule la valeur absolue.
- `max(8, 15)` → `15` : calcule le maximum entre deux nombres.
- `max(12, 9, 21)` → `21` : calcule le maximum entre trois nombres.
- `min(7, 14)` → `7` : calcule le minimum entre deux nombres.
- `min(30, 18, 25)` → `18` : calcule le minimum entre trois nombres.
- `round(3.14159)` → `3` : arrondit à l'entier le plus proche.
- `round(2.71828, 2)` → `2.72` : arrondit à deux décimales.

Les fonctions natives mathématiques

Fonctions très utiles comme le calcul du maximum ou de la valeur absolue.

- `abs(-10)` → `10` : calcule la valeur absolue.
- `max(8, 15)` → `15` : calcule le maximum entre deux nombres.
- `max(12, 9, 21)` → `21` : calcule le maximum entre trois nombres.
- `min(7, 14)` → `7` : calcule le minimum entre deux nombres.
- `min(30, 18, 25)` → `18` : calcule le minimum entre trois nombres.
- `round(3.14159)` → `3` : arrondit à l'entier le plus proche.
- `round(2.71828, 2)` → `2.72` : arrondit à deux décimales.

Les fonctions natives mathématiques

Fonctions très utiles comme le calcul du maximum ou de la valeur absolue.

- `abs(-10)` → `10` : calcule la valeur absolue.
- `max(8, 15)` → `15` : calcule le maximum entre deux nombres.
- `max(12, 9, 21)` → `21` : calcule le maximum entre trois nombres.
- `min(7, 14)` → `7` : calcule le minimum entre deux nombres.
- `min(30, 18, 25)` → `18` : calcule le minimum entre trois nombres.
- `round(3.14159)` → `3` : arrondit à l'entier le plus proche.
- `round(2.71828, 2)` → `2.72` : arrondit à deux décimales.

Les fonctions natives mathématiques

Fonctions très utiles comme le calcul du maximum ou de la valeur absolue.

- `abs(-10)` → `10` : calcule la valeur absolue.
- `max(8, 15)` → `15` : calcule le maximum entre deux nombres.
- `max(12, 9, 21)` → `21` : calcule le maximum entre trois nombres.
- `min(7, 14)` → `7` : calcule le minimum entre deux nombres.
- `min(30, 18, 25)` → `18` : calcule le minimum entre trois nombres.
- `round(3.14159)` → `3` : arrondit à l'entier le plus proche.
- `round(2.71828, 2)` → `2.72` : arrondit à deux décimales.

Les fonctions natives mathématiques

Fonctions très utiles comme le calcul du maximum ou de la valeur absolue.

- `abs(-10)` → `10` : calcule la valeur absolue.
- `max(8, 15)` → `15` : calcule le maximum entre deux nombres.
- `max(12, 9, 21)` → `21` : calcule le maximum entre trois nombres.
- `min(7, 14)` → `7` : calcule le minimum entre deux nombres.
- `min(30, 18, 25)` → `18` : calcule le minimum entre trois nombres.
- `round(3.14159)` → `3` : arrondit à l'entier le plus proche.
- `round(2.71828, 2)` → `2.72` : arrondit à deux décimales.

Les fonctions natives mathématiques

Fonctions très utiles comme le calcul du maximum ou de la valeur absolue.

- `abs(-10)` → `10` : calcule la valeur absolue.
- `max(8, 15)` → `15` : calcule le maximum entre deux nombres.
- `max(12, 9, 21)` → `21` : calcule le maximum entre trois nombres.
- `min(7, 14)` → `7` : calcule le minimum entre deux nombres.
- `min(30, 18, 25)` → `18` : calcule le minimum entre trois nombres.
- `round(3.14159)` → `3` : arrondit à l'entier le plus proche.
- `round(2.71828, 2)` → `2.72` : arrondit à deux décimales.

Vérification des Types d'Objets - Les fonctions natives avancées

En Python, il est souvent nécessaire de vérifier le type d'objet d'une variable pour s'assurer de son adéquation. Deux fonctions couramment utilisées à cet effet sont 'isinstance' et 'type()'.

- **La fonction 'isinstance' :** Cette fonction permet de vérifier si le type d'objet d'une variable correspond à un type spécifié. Elle renvoie True si le type spécifié correspond au type de la variable, et False sinon.
  Exemple :

```
>>> x = 42
>>> isinstance(x, int)
True

>>> a = 3.14
>>> isinstance(a, float)
True
```

Vérification des Types d'Objets - Les fonctions natives avancées

En Python, il est souvent nécessaire de vérifier le type d'objet d'une variable pour s'assurer de son adéquation. Deux fonctions couramment utilisées à cet effet sont 'isinstance' et 'type()'.

- **La fonction 'type()' :** Cette fonction retourne le type de données de la variable spécifiée. Elle est utile pour obtenir le type de la variable sans effectuer de comparaison.
  Exemple :

```
>>> y = "Hello, World!"
>>> type(y)
<class 'str'>

>>> d = False
>>> type(d)
<class 'bool'>
```

Suppression de Variables en Python avec `del()`

En Python, la fonction `del()` est utilisée pour détruire explicitement une variable, ce qui signifie que son nom n'est plus accessible par la suite.
Exemple :

```
>>> x = 42
>>> print(x)
42

>>> del x  # Suppression de la variable x
>>> print(x)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
```

Utilisez la fonction `del()` avec précaution, car elle supprime la variable de manière permanente.

Sommaire

Erreur : Syntaxe print

- Erreur : Utilisation incorrecte de la fonction print.
- Exemple incorrect :

```
print "Bonjour, Python !"
```

- Correction :

```
print("Bonjour, Python !")
```

Python SyntaxError

```
a = 88tt1
```

- Ligne de code Python : `a = 88tt1`
- Erreur : `SyntaxError: invalid syntax`

```
a = (88xx1
```

- Ligne de code Python : `a = (88tt1`
- Erreur : `SyntaxError: unexpected EOF while parsing`
- Description : Une erreur de parenthésage est survenue, la parenthèse ouvrante '(' n'a pas de parenthèse fermante correspondante.

Erreur : SyntaxError - Inconsistent Indentation

- Cette erreur se produit lorsque vous utilisez à la fois des tabulations et des espaces pour l'indentation dans votre code.
- Python exige la cohérence : utilisez soit des tabulations, soit des espaces, mais pas les deux.
- Exemple incorrect :

```
if True:
\print("Utilise des tabulations")
    print("Et aussi des espaces")
```

- Correction :

```
if True:
    print("Utilise soit des tabulations, soit des espaces")
```

Erreur : NameError - Variable non définie

- Cette erreur se produit lorsque vous essayez d'utiliser une variable qui n'a pas été définie.
- Exemple incorrect :

```
print(x)
```

- Correction :

```
x = 42
print(x)
```

Erreurs non signalées en Python

Elles existent et ce sont les plus difficiles à détecter, car elles passent sous les radars, aussi bien pour l'interpréteur Python que pour vos yeux.

```
Age = 10
if Age >= 18 :
  print("Majeur")
else :
  print("Mineur")
Age == 20
if Age >= 18 :
  print("Majeur")
else :
  print("Mineur")
```

Quel est le résultat de ce programme ? Vous pensez à "Mineur" puis "Majeur" ? Cela aurait dû être le cas.

Conversion entre Code Unicode et Caractère en Python

En Python, les fonctions `chr()` et `ord()` sont utilisées pour convertir entre un code Unicode et le caractère associé, et vice versa.

- **La fonction** `chr()` : Cette fonction prend un code Unicode comme argument et renvoie le caractère associé.
  Exemple :
  ```
  >>> caractere = chr(65)
  >>> print(caractere)
  'A'
  ```

- **La fonction** `ord()` : Cette fonction prend un caractère comme argument et renvoie son code Unicode.
  Exemple :
  ```
  >>> code_unicode = ord('A')
  >>> print(code_unicode)
  65
  ```

Priorité des Opérations Mathématiques en Python

En Python, les opérations mathématiques sont effectuées en suivant un ordre de priorité standard. Voici les principaux opérateurs par ordre de priorité décroissante :

1. Parenthèses () (les expressions à l'intérieur des parenthèses sont évaluées en premier).
2. Exponentiation **.
3. Multiplication *, Division /, Division entière //, Modulo % (ces opérations sont évaluées de gauche à droite).
4. Addition +, Soustraction - (ces opérations sont évaluées de gauche à droite).

Exemple :

$$2 \cdot (3 + 4)^2 / 7 - 1$$

Le résultat de cette expression sera calculé en respectant l'ordre de priorité.

Sommaire

Listes en Python

- Une liste est une collection d'éléments ordonnés et modifiables.
- Exemple de liste :
  - `ma_liste = [1, 2, 3, 4, 5]`
- Il est possible d'accéder aux éléments par leur indice, commençant à 0.
- Il est possible d'ajouter, de supprimer et de modifier des éléments dans une liste.

Dictionnaires et Tuples

- Un tuple est une collection d'éléments ordonnés et immuables.
- Exemple de tuple :
  - `mon_tuple = (1, 2, 3, 4, 5)`
- Les éléments d'un tuple ne peuvent pas être modifiés après sa création.
- Les tuples sont souvent utilisés pour stocker des données non modifiables, comme les coordonnées géographiques.

Différences entre Listes et Tuples

- Listes :
  - Modifiables (ajouter, supprimer, modifier).
  - Utilisez des crochets : `ma_liste = [1, 2, 3]`.
- Tuples :
  - Immuables (ne peut pas être modifié après création).
  - Utilisez des parenthèses : `mon_tuple = (1, 2, 3)`.

Conversion de Séquences

- Il est possible de convertir des séquences telles que des listes en différents types de séquences en utilisant des constructeurs de type.
- Exemple :
  - `liste = [1, 2, 3]`
  - `mon_tuple = tuple(liste) # Convertir une liste en tuple`

Opérations sur les listes en Python

- Ajouter un élément en fin de liste :
  - Utilisation de la syntaxe : `L.append(element)`
- Insérer un élément à une position précise :
  - Utilisation de la syntaxe : `L.insert(position, element)`
- Retirer l'élément en fin de liste :
  - Utilisation de la commande : `L.pop()`
- Retirer un élément à une position donnée :
  - Utilisation de la commande : `L.pop(position)`

Exemple avec l'équipe de rugby de France :

- Ajout d'un nouveau joueur : `equipe_france.append("Nouveau Joueur")`
- Remplacement d'un joueur existant : `equipe_france[3] = "Joueur Remplacé"`
- Retrait du dernier joueur : `equipe_france.pop()`
- Retrait du joueur en position 2 : `equipe_france.pop(2)`

Opérations sur les listes en Python

- Ajouter un élément en fin de liste :
  - Utilisation de la syntaxe : `L.append(element)`
- Insérer un élément à une position précise :
  - Utilisation de la syntaxe : `L.insert(position, element)`
- Retirer l'élément en fin de liste :
  - Utilisation de la commande : `L.pop()`
- Retirer un élément à une position donnée :
  - Utilisation de la commande : `L.pop(position)`

Exemple avec l'équipe de rugby de France :

- Ajout d'un nouveau joueur : `equipe_france.append("Nouveau Joueur")`
- Remplacement d'un joueur existant : `equipe_france[3] = "Joueur Remplacé"`
- Retrait du dernier joueur : `equipe_france.pop()`
- Retrait du joueur en position 2 : `equipe_france.pop(2)`

Opérations sur les listes en Python

- Ajouter un élément en fin de liste :
  - Utilisation de la syntaxe : `L.append(element)`
- Insérer un élément à une position précise :
  - Utilisation de la syntaxe : `L.insert(position, element)`
- Retirer l'élément en fin de liste :
  - Utilisation de la commande : `L.pop()`
- Retirer un élément à une position donnée :
  - Utilisation de la commande : `L.pop(position)`

Exemple avec l'équipe de rugby de France :

- Ajout d'un nouveau joueur : `equipe_france.append("Nouveau Joueur")`
- Remplacement d'un joueur existant : `equipe_france[3] = "Joueur Remplacé"`
- Retrait du dernier joueur : `equipe_france.pop()`
- Retrait du joueur en position 2 : `equipe_france.pop(2)`

Opérations sur les listes en Python

- Ajouter un élément en fin de liste :
  - Utilisation de la syntaxe : `L.append(element)`
- Insérer un élément à une position précise :
  - Utilisation de la syntaxe : `L.insert(position, element)`
- Retirer l'élément en fin de liste :
  - Utilisation de la commande : `L.pop()`
- Retirer un élément à une position donnée :
  - Utilisation de la commande : `L.pop(position)`

Exemple avec l'équipe de rugby de France :

- Ajout d'un nouveau joueur : `equipe_france.append("Nouveau Joueur")`
- Remplacement d'un joueur existant : `equipe_france[3] = "Joueur Remplacé"`
- Retrait du dernier joueur : `equipe_france.pop()`
- Retrait du joueur en position 2 : `equipe_france.pop(2)`

Opérations sur les listes en Python

- Ajouter un élément en fin de liste :
  - Utilisation de la syntaxe : `L.append(element)`
- Insérer un élément à une position précise :
  - Utilisation de la syntaxe : `L.insert(position, element)`
- Retirer l'élément en fin de liste :
  - Utilisation de la commande : `L.pop()`
- Retirer un élément à une position donnée :
  - Utilisation de la commande : `L.pop(position)`

Exemple avec l'équipe de rugby de France :

- Ajout d'un nouveau joueur : `equipe_france.append("Nouveau Joueur")`
- Remplacement d'un joueur existant : `equipe_france[3] = "Joueur Remplacé"`
- Retrait du dernier joueur : `equipe_france.pop()`
- Retrait du joueur en position 2 : `equipe_france.pop(2)`

Opérations sur les listes en Python

- Ajouter un élément en fin de liste :
  - Utilisation de la syntaxe : `L.append(element)`
- Insérer un élément à une position précise :
  - Utilisation de la syntaxe : `L.insert(position, element)`
- Retirer l'élément en fin de liste :
  - Utilisation de la commande : `L.pop()`
- Retirer un élément à une position donnée :
  - Utilisation de la commande : `L.pop(position)`

Exemple avec l'équipe de rugby de France :

- Ajout d'un nouveau joueur : `equipe_france.append("Nouveau Joueur")`
- Remplacement d'un joueur existant : `equipe_france[3] = "Joueur Remplacé"`
- Retrait du dernier joueur : `equipe_france.pop()`
- Retrait du joueur en position 2 : `equipe_france.pop(2)`

Opérations sur les listes en Python

- Ajouter un élément en fin de liste :
  - Utilisation de la syntaxe : `L.append(element)`
- Insérer un élément à une position précise :
  - Utilisation de la syntaxe : `L.insert(position, element)`
- Retirer l'élément en fin de liste :
  - Utilisation de la commande : `L.pop()`
- Retirer un élément à une position donnée :
  - Utilisation de la commande : `L.pop(position)`

Exemple avec l'équipe de rugby de France :

- Ajout d'un nouveau joueur : `equipe_france.append("Nouveau Joueur")`
- Remplacement d'un joueur existant : `equipe_france[3] = "Joueur Remplacé"`
- Retrait du dernier joueur : `equipe_france.pop()`
- Retrait du joueur en position 2 : `equipe_france.pop(2)`

Opérations sur les listes en Python

- Ajouter un élément en fin de liste :
  - Utilisation de la syntaxe : `L.append(element)`
- Insérer un élément à une position précise :
  - Utilisation de la syntaxe : `L.insert(position, element)`
- Retirer l'élément en fin de liste :
  - Utilisation de la commande : `L.pop()`
- Retirer un élément à une position donnée :
  - Utilisation de la commande : `L.pop(position)`

Exemple avec l'équipe de rugby de France :

- Ajout d'un nouveau joueur : `equipe_france.append("Nouveau Joueur")`
- Remplacement d'un joueur existant : `equipe_france[3] = "Joueur Remplacé"`
- Retrait du dernier joueur : `equipe_france.pop()`
- Retrait du joueur en position 2 : `equipe_france.pop(2)`

Opérations sur les listes en Python

- Ajouter un élément en fin de liste :
  - Utilisation de la syntaxe : `L.append(element)`
- Insérer un élément à une position précise :
  - Utilisation de la syntaxe : `L.insert(position, element)`
- Retirer l'élément en fin de liste :
  - Utilisation de la commande : `L.pop()`
- Retirer un élément à une position donnée :
  - Utilisation de la commande : `L.pop(position)`

Exemple avec l'équipe de rugby de France :

- Ajout d'un nouveau joueur : `equipe_france.append("Nouveau Joueur")`
- Remplacement d'un joueur existant : `equipe_france[3] = "Joueur Remplacé"`
- Retrait du dernier joueur : `equipe_france.pop()`
- Retrait du joueur en position 2 : `equipe_france.pop(2)`

Opérations sur la liste de joueurs de rugby français

- Création de la liste de joueurs :
  - `joueurs_francais = ["Antoine Dupont", "Julien Marchand", "Thomas Ramos"]`

Ajout d'un nouveau joueur :

- Utilisation de `append` : `joueurs_francais.append("Damian Penaud ")`

Retrait du dernier joueur :

- Utilisation de `pop` : `joueurs_francais.pop()`

Insertion d'un joueur à la position 2 :

- Utilisation de `insert` : `joueurs_francais.insert(2, "Damian Penaud ")`

Liste mise à jour des joueurs :

- Antoine Dupont
- Julien Marchand
- Damian Penaud
- Thomas Ramos

Opérations sur la liste de joueurs de rugby français

- Création de la liste de joueurs :
  - `joueurs_francais = ["Antoine Dupont", "Julien Marchand", "Thomas Ramos"]`

Ajout d'un nouveau joueur :

- Utilisation de append : `joueurs_francais.append("Damian Penaud ")`

Retrait du dernier joueur :

- Utilisation de pop : `joueurs_francais.pop()`

Insertion d'un joueur à la position 2 :

- Utilisation de `insert` : `joueurs_francais.insert(2, "Damian Penaud ")`

Liste mise à jour des joueurs :

- Antoine Dupont
- Julien Marchand
- Damian Penaud
- Thomas Ramos

Opérations sur la liste de joueurs de rugby français

- Création de la liste de joueurs :
  - `joueurs_francais = ["Antoine Dupont", "Julien Marchand", "Thomas Ramos"]`

Ajout d'un nouveau joueur :

- Utilisation de `append` : `joueurs_francais.append("Damian Penaud ")`

Retrait du dernier joueur :

- Utilisation de `pop` : `joueurs_francais.pop()`

Insertion d'un joueur à la position 2 :

- Utilisation de `insert` : `joueurs_francais.insert(2, "Damian Penaud ")`

Liste mise à jour des joueurs :

- Antoine Dupont
- Julien Marchand
- Damian Penaud
- Thomas Ramos

Opérations sur la liste de joueurs de rugby français

- Création de la liste de joueurs :
  - `joueurs_francais = ["Antoine Dupont", "Julien Marchand", "Thomas Ramos"]`

Ajout d'un nouveau joueur :

- Utilisation de `append` : `joueurs_francais.append("Damian Penaud ")`

Retrait du dernier joueur :

- Utilisation de `pop` : `joueurs_francais.pop()`

Insertion d'un joueur à la position 2 :

- Utilisation de `insert` : `joueurs_francais.insert(2, "Damian Penaud ")`

Liste mise à jour des joueurs :

- Antoine Dupont
- Julien Marchand
- Damian Penaud
- Thomas Ramos

Opérations sur la liste de joueurs de rugby français

- Création de la liste de joueurs :
  - `joueurs_francais = ["Antoine Dupont", "Julien Marchand", "Thomas Ramos"]`

Ajout d'un nouveau joueur :

- Utilisation de `append` : `joueurs_francais.append("Damian Penaud ")`

Retrait du dernier joueur :

- Utilisation de `pop` : `joueurs_francais.pop()`

Insertion d'un joueur à la position 2 :

- Utilisation de `insert` : `joueurs_francais.insert(2, "Damian Penaud ")`

Liste mise à jour des joueurs :

- Antoine Dupont
- Julien Marchand
- Damian Penaud
- Thomas Ramos

Opérations sur la liste de joueurs de rugby français

- Création de la liste de joueurs :
  - `joueurs_francais = ["Antoine Dupont", "Julien Marchand", "Thomas Ramos"]`

Ajout d'un nouveau joueur :

- Utilisation de `append` : `joueurs_francais.append("Damian Penaud ")`

Retrait du dernier joueur :

- Utilisation de `pop` : `joueurs_francais.pop()`

Insertion d'un joueur à la position 2 :

- Utilisation de `insert` : `joueurs_francais.insert(2, "Damian Penaud ")`

Liste mise à jour des joueurs :

- Antoine Dupont
- Julien Marchand
- Damian Penaud
- Thomas Ramos

Opérations sur la liste de joueurs de rugby français

- Création de la liste de joueurs :
  - `joueurs_francais = ["Antoine Dupont", "Julien Marchand", "Thomas Ramos"]`

Ajout d'un nouveau joueur :

- Utilisation de `append` : `joueurs_francais.append("Damian Penaud ")`

Retrait du dernier joueur :

- Utilisation de `pop` : `joueurs_francais.pop()`

Insertion d'un joueur à la position 2 :

- Utilisation de `insert` : `joueurs_francais.insert(2, "Damian Penaud ")`

Liste mise à jour des joueurs :

- Antoine Dupont
- Julien Marchand
- Damian Penaud
- Thomas Ramos

Opérations sur la liste de joueurs de rugby français

- Création de la liste de joueurs :
  - `joueurs_francais = ["Antoine Dupont", "Julien Marchand", "Thomas Ramos"]`

Ajout d'un nouveau joueur :

- Utilisation de `append` : `joueurs_francais.append("Damian Penaud ")`

Retrait du dernier joueur :

- Utilisation de `pop` : `joueurs_francais.pop()`

Insertion d'un joueur à la position 2 :

- Utilisation de `insert` : `joueurs_francais.insert(2, "Damian Penaud ")`

Liste mise à jour des joueurs :

- Antoine Dupont
- Julien Marchand
- Damian Penaud
- Thomas Ramos

Opérations sur la liste de joueurs de rugby français

- Création de la liste de joueurs :
  - `joueurs_francais = ["Antoine Dupont", "Julien Marchand", "Thomas Ramos"]`

Ajout d'un nouveau joueur :

- Utilisation de `append` : `joueurs_francais.append("Damian Penaud ")`

Retrait du dernier joueur :

- Utilisation de `pop` : `joueurs_francais.pop()`

Insertion d'un joueur à la position 2 :

- Utilisation de `insert` : `joueurs_francais.insert(2, "Damian Penaud ")`

Liste mise à jour des joueurs :

- Antoine Dupont
- Julien Marchand
- Damian Penaud
- Thomas Ramos

Opérations sur la liste de joueurs de rugby français

- Création de la liste de joueurs :
  - `joueurs_francais = ["Antoine Dupont", "Julien Marchand", "Thomas Ramos"]`

Ajout d'un nouveau joueur :

- Utilisation de `append` : `joueurs_francais.append("Damian Penaud ")`

Retrait du dernier joueur :

- Utilisation de `pop` : `joueurs_francais.pop()`

Insertion d'un joueur à la position 2 :

- Utilisation de `insert` : `joueurs_francais.insert(2, "Damian Penaud ")`

Liste mise à jour des joueurs :

- Antoine Dupont
- Julien Marchand
- Damian Penaud
- Thomas Ramos

Opérations sur la liste de joueurs de rugby français

- Création de la liste de joueurs :
  - `joueurs_francais = ["Antoine Dupont", "Julien Marchand", "Thomas Ramos"]`

Ajout d'un nouveau joueur :

- Utilisation de `append` : `joueurs_francais.append("Damian Penaud ")`

Retrait du dernier joueur :

- Utilisation de `pop` : `joueurs_francais.pop()`

Insertion d'un joueur à la position 2 :

- Utilisation de `insert` : `joueurs_francais.insert(2, "Damian Penaud ")`

Liste mise à jour des joueurs :

- Antoine Dupont
- Julien Marchand
- Damian Penaud
- Thomas Ramos

Opérations sur la liste de joueurs de rugby français

- Création de la liste de joueurs :
  - `joueurs_francais = ["Antoine Dupont", "Julien Marchand", "Thomas Ramos"]`

Ajout d'un nouveau joueur :

- Utilisation de `append` : `joueurs_francais.append("Damian Penaud ")`

Retrait du dernier joueur :

- Utilisation de `pop` : `joueurs_francais.pop()`

Insertion d'un joueur à la position 2 :

- Utilisation de `insert` : `joueurs_francais.insert(2, "Damian Penaud ")`

Liste mise à jour des joueurs :

- Antoine Dupont
- Julien Marchand
- Damian Penaud
- Thomas Ramos

Dictionnaires en Python

- Un dictionnaire est une collection d'éléments associés à des clés.
- Exemple de dictionnaire :
  - `mon_dictionnaire = {'nom': 'Alice', 'âge': 30, 'ville': 'Paris'}`
- Les éléments peuvent être accédés en utilisant leurs clés.
- Les éléments peuvent être ajoutés, supprimés et modifiés dans un dictionnaire.

Ensembles en Python

- Un ensemble est une collection d'éléments non ordonnés et uniques.
- Exemple d'ensemble :
  - `mon_ensemble = {1, 2, 3, 4, 5}`
- Les ensembles ne permettent pas de doublons.
- On peut effectuer des opérations ensemblistes telles que l'union, l'intersection et la différence sur les ensembles.

Différences entre Dictionnaires et Ensembles

- Dictionnaires :
  - Collection d'éléments associés à des clés.
  - Utilisez des accolades : `{'nom': 'Alice', 'âge': 30}`.
- Ensembles :
  - Collection d'éléments uniques et non ordonnés.
  - Utilisez des accolades : `{1, 2, 3}`.

La Fonction len()

La fonction 'len()' est utilisée pour obtenir la longueur (le nombre d'éléments) d'une séquence ou d'une collection en Python. Voici un exemple :

Exemple : Utilisation de len()

```
# Une liste d'exemple
ma_liste = [1, 2, 3, 4, 5]

# Obtenir la longueur de la liste
longueur = len(ma_liste)

print("La longueur de la liste est", longueur)
```

Résultat : La longueur de la liste est 5

La Fonction range()

La fonction ‘range()‘ est utilisée pour générer une séquence de nombres dans une plage donnée. Voici un exemple :

Exemple : Utilisation de range()

```
# Générer une séquence de nombres de 0 à 4
ma_sequence = range(5)

# Afficher la séquence
for nombre in ma_sequence:
    print(nombre)
```

Résultat :

```
0
1
2
```

Utilisation de len() et range() en Python

Imaginez que vous ayez une boîte de bonbons, et vous voulez compter combien de bonbons il y a à l'intérieur.

Exemple : Compter les bonbons

```
boite_de_bonbons = ["Fraise", "Chocolat", "Citron", "Menthe"]
nombre_de_bonbons = len(boite_de_bonbons)

for i in range(nombre_de_bonbons):
    print(f"Bonbon {i+1} : {boite_de_bonbons[i]}")
```

Le résultat de l'exemple précédent est le suivant :

```
Bonbon 1 : Fraise
Bonbon 2 : Chocolat
Bonbon 3 : Citron
Bonbon 4 : Menthe
```

Utilisation de len() et range() en Python

Imaginez que vous ayez une boîte de bonbons, et vous voulez compter combien de bonbons il y a à l'intérieur.

Exemple : Compter les bonbons

```
boite_de_bonbons = ["Fraise", "Chocolat", "Citron", "Menthe"]
nombre_de_bonbons = len(boite_de_bonbons)

for i in range(nombre_de_bonbons):
    print(f"Bonbon {i+1} : {boite_de_bonbons[i]}")
```

Le résultat de l'exemple précédent est le suivant :

```
Bonbon 1 : Fraise
Bonbon 2 : Chocolat
Bonbon 3 : Citron
Bonbon 4 : Menthe
```

Les Fonctions en Python

- Une fonction en Python est un bloc de code réutilisable qui effectue une tâche spécifique.
- Les fonctions sont définies à l'aide du mot-clé `def` suivi du nom de la fonction et de parenthèses contenant les paramètres (si nécessaire).
- Une fonction peut prendre des paramètres en entrée, effectuer des opérations et renvoyer un résultat.
- Les fonctions facilitent la réutilisation du code et la modularité.

Exemple de Fonction Python

Voici un exemple simple d'une fonction Python qui calcule la somme de deux nombres :

```
def somme(a, b):
    resultat = a + b
    return resultat

x = 5
y = 3
resultat_somme = somme(x, y)
print(resultat_somme)  # Affiche 8
```

Cette fonction prend deux paramètres (a et b), effectue une addition, et renvoie le résultat.

Valeurs par Défaut pour les Arguments de Fonction en Python

- En Python, on peut définir des valeurs par défaut pour les arguments de fonction.
- Cela signifie que si un argument n'est pas spécifié lors de l'appel de la fonction, il prendra la valeur par défaut que vous avez définie.
- Cela permet d'appeler la fonction avec un nombre variable d'arguments.

Exemple d'Utilisation de Valeurs par Défaut

Voici un exemple de fonction Python qui utilise des valeurs par défaut pour les arguments :

```
def add_numbers(a=7, b=8):
    sum = a + b
    print('Somme :', sum)

# Appel de la fonction avec deux arguments
add_numbers(2, 3)

# Appel de la fonction avec un argument
add_numbers(a=2)

# Appel de la fonction sans argument
add_numbers()
```

Fonctions Récursives en Python

- Une fonction récursive est une fonction qui s'appelle elle-même pour résoudre un problème,
- Les fonctions récursives ont deux parties principales : le cas de base et le cas récursif,
- Le cas de base est la condition qui arrête la récursion,
- Le cas récursif est l'appel de la fonction à elle-même avec des arguments différents,
- Les fonctions récursives peuvent être utiles pour résoudre des problèmes qui peuvent être décomposés en sous-problèmes similaires.

Exemple de Fonction Récursive Python

Voici un exemple de fonction Python récursive qui calcule le factoriel d'un nombre :

```
def factorial(x):
    """Ceci est une fonction récursive
    pour trouver le factoriel d'un entier"""

    if x == 1:
        return 1
    else:
        return (x * factorial(x-1))

num = 3
print("Le factoriel de", num, "est", factorial(num))
```

Portée Locale

En Python, une variable déclarée à l'intérieur d'une fonction a une **portée locale**. Cela signifie qu'elle n'est accessible qu'à l'intérieur de cette fonction.

Exemple de Portée Locale

```
def ma_fonction():
    x = 10
    print(x)

ma_fonction()
# Ceci fonctionnera
print(x)
# Cela générera une erreur
```

Portée Globale

Une variable déclarée en dehors de toutes les fonctions a une **portée globale**. Cela signifie qu'elle est accessible depuis n'importe où dans le script.

Exemple de Portée Globale

```
x = 10

def ma_fonction():
    print(x)

ma_fonction()
# Ceci fonctionnera
print(x)
# Ceci fonctionnera également
```

Portée Non Locale

Une variable déclarée comme non locale dans une fonction se réfère à une variable dans la portée englobante.

Exemple de Portée Non Locale

```
x = 10

def ma_fonction():
    x = 5
    def ma_autre_fonction():
        nonlocal x
        x = 20
    ma_autre_fonction()
    print(x)

ma_fonction()
# Ceci affichera 20
```

Sommaire

Utilisation du Module Python

Supposons que vous avez un module Python nommé ‘example.py‘ contenant la fonction ‘add‘ :

```
# example.py

def add(a, b):
    result = a + b
    return result
```

Script Python :

```
import example

resultat = example.add(4, 4)
print(resultat)  # Affiche 8
```

Importation des Modules de la Bibliothèque Standard

La bibliothèque standard Python contient bien plus de 200 modules. Nous pouvons importer un module selon nos besoins.
Par exemple, si nous voulons obtenir la valeur de pi, nous importons d'abord le module math et utilisons math.pi.

Exemple d'Importation du Module math

```
# Importation du module math standard
import math

# Utilisation de math.pi pour obtenir la valeur de pi
print("La valeur de pi est", math.pi)
```

Importation en Python avec Renommage

En Python, nous pouvons également importer un module en lui donnant un autre nom (renommage).
Par exemple, nous pouvons importer le module math et le renommer en utilisant ‘as‘.

Exemple d'Importation avec Renommage

```
# Importation du module en le renommant
import math as m

# Utilisation de m.pi pour obtenir la valeur de pi
print(m.pi)
```

Importation de Tous les Noms en Python

En Python, nous pouvons importer tous les noms (définitions) d'un module en utilisant la construction suivante :

Exemple d'Importation de Tous les Noms depuis le Module math

```
# Importation de tous les noms depuis le module math standard
from math import *

print("La valeur de pi est", pi)
```

Ici, nous avons importé toutes les définitions depuis le module math. Cela inclut tous les noms visibles dans notre portée, à l'exception de ceux commençant par un trait de soulignement (définitions privées).

La Fonction dir() en Python

La fonction intégrée ‘dir()‘ en Python est utilisée pour lister tous les noms (identifiants) dans un module ou dans l’espace de noms courant.

Utilisation de la Fonction dir()

Pour lister les noms dans un module, on peut utiliser la syntaxe suivante :

```
import nom_du_module
print(dir(nom_du_module))
```

Voyons comment cela fonctionne avec le module ‘math‘ :

```
import math
print(dir(math))
```

Liste des Noms dans le Module math

Voici la liste des noms (identifiants) dans le module ‘math‘ :

```
['__doc__', '__file__', '__loader__', '__name__', '__package__',
 '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2',
 'atanh', 'ceil', 'comb', 'copysign', 'cos', 'cosh', 'degrees',
 'dist', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial',
 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf',
 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log',
 'log10', 'log1p', 'log2', 'modf', 'nan', 'perm', 'pi', 'pow',
 'prod', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh',
 'tau', 'trunc']
```

Importer un Module depuis un Package

Lorsque vous avez un package Python contenant plusieurs modules, on peut importer un module spécifique depuis ce package.

Syntaxe d'Importation

Pour importer un module depuis un package, utilisez la syntaxe suivante :

```
from nom_du_package import nom_du_module
```

Par exemple, si vous avez un package ‘monpackage‘ contenant un module ‘monmodule‘, on peut l'importer comme suit :

```
from monpackage import monmodule
```

Exemple d'Importation d'un Module depuis un Package

Supposons que vous ayez un package 'monpackage' avec un module 'monmodule.py' contenant une fonction 'ma_fonction'.

Exemple d'Importation

```
from monpackage import monmodule

# Appel de la fonction ma_fonction depuis monmodule
monmodule.ma_fonction()
```

Différence entre Modules et Packages

En Python, les modules et les packages sont utilisés pour organiser et structurer le code, mais ils servent des objectifs légèrement différents.

Module Python

Un module Python est un fichier contenant des définitions et des instructions Python. Il peut être utilisé pour regrouper du code lié dans un seul endroit.

Package Python

Un package Python est un répertoire qui contient un ensemble de modules. Il est utilisé pour organiser un grand projet en plusieurs modules pour une meilleure gestion.

En résumé, un module est un fichier, tandis qu'un package est un répertoire qui contient des fichiers (modules).

Exemple de Module Python

Voici un exemple de module Python nommé ‘monmodule.py‘ :

```
# monmodule.py

def ma_fonction():
    print("Ceci est ma fonction")

ma_variable = 42
```

Exemple de Package Python

Voici un exemple de package Python contenant deux modules : module1.py et module2.py :

```
monpackage/
    __init\__.py'
    module1.py
    module2.py
```

Les fichiers module1.py et module2.py peuvent contenir des définitions et des fonctions liées.

Exemple : Utilisation du Module Requests

```
import requests

# Définir l'URL
url = 'https://jsonplaceholder.typicode.com/posts/1'

# Envoyer la requête GET
response = requests.get(url)

# Vérifier le code d'état
if response.status_code == 200:
    print('Requête réussie !')
    data = response.json() #Données de la réponse :
    print(data)
else:
    print('Échec de la requête avec le code d\'état :', response.status_code)
```

Classes en Python

En Python, une classe est un modèle ou un plan pour créer des objets. Un objet est une instance d'une classe. Voici comment définir une classe en Python :

Exemple : Définition d'une Classe

```
class Personne:
    def __init__(self, nom, age):
        self.nom = nom
        self.age = age

    def presenter(self):
        print(f"Je m'appelle {self.nom} et j'ai {self.age} ans.")
```

Création d'Objets

Une fois que vous avez défini une classe, von peut créer des objets (instances) de cette classe. Voici comment créer un objet et appeler une méthode de la classe :

Exemple : Création d'un Objet

```
# Créer un objet de la classe Personne
personne1 = Personne("Toto", 30)

# Appeler la méthode presenter()
personne1.presenter()
```

Résultat : "Je m'appelle Toto et j'ai 30 ans."

Création de Multiples Objets

```
# Définition de la classe Voiture
class Voiture:
    def __init__(self, marque, modele):
        self.marque = marque
        self.modele = modele

# Création de plusieurs objets Voiture
voiture1 = Voiture("Toyota", "Camry")
voiture2 = Voiture("Honda", "Accord")
voiture3 = Voiture("Ford", "Mustang")

# Affichage des détails des voitures
print(f"Voiture 1 : {voiture1.marque} {voiture1.modele}")
print(f"Voiture 2 : {voiture2.marque} {voiture2.modele}")
print(f"Voiture 3 : {voiture3.marque} {voiture3.modele}")
```

Fonction dans une Classe Python

```
class Chambre: # Créer une classe
    longueur = 0.0
    largeur = 0.0

    def calculer_surface(self): # Méthode pour calculer la surface
        print("Surface de la Chambre =", self.longueur * self.largeur)

chambre_etude = Chambre() # Créer un objet de la classe Chambre

chambre_etude.longueur = 42.5
chambre_etude.largeur = 30.8

chambre_etude.calculer_surface()
```

Sommaire

Utilisation du Module os

Le module ‘os‘ en Python permet d’interagir avec le système d’exploitation.

```
import os

cwd = os.getcwd() # Obtenir le répertoire de travail actuel
print("Répertoire de travail actuel :", cwd)

files = os.listdir(cwd) # Lister les fichiers dans un répertoire
print("Fichiers dans le répertoire :", files)

new_directory = "nouveau_dossier" # Créer un nouveau répertoire
os.mkdir(new_directory)
print("Nouveau répertoire créé :", new_directory)

os.rmdir(new_directory) # Supprimer un répertoire
print("Répertoire supprimé :", new_directory)
```

Utilisation de ‘with‘ pour la Gestion des Fichiers

On peut utiliser ‘with‘ pour simplifier la gestion des fichiers. Il se charge automatiquement de la fermeture du fichier, même en cas d'exception.

Exemple : Utilisation de ‘with‘

```
# Utilisation de ‘with‘ pour lire un fichier
with open("more_names.txt") as f:
    content = f.read()

# Utilisation de ‘with‘ pour écrire dans un fichier
with open("names.txt", "w") as f:
    f.write(content)
```

Applications - Listes

```
A = [0, 10, 20,30]
B = A
B[2] = 40
print(A)
print(B)
```

le résultat de l'affichage est-il correct ?
[0, 10, 20, 30] print(L)
[0, 10, 40, 30] print(K)

Variables locales et variables globales

```
a = 1

def test(b):
    print(b)
    print(c)
    c = 5
    print(c)
    print(b)
test(a)
print(c)
print(b)
```

Identifiez les erreurs dans le code,
Assurez-vous que toutes les variables sont correctement déclarées et que leur portée est appropriée.

Demo

- Calculs Simples,
- Conversion de température de Celsius en Fahrenheit. La formule de conversion de Celsius en Fahrenheit est la suivante :

$$Fahrenheit = (Celsius \times \frac{9}{5}) + 32$$

Demo

```
# Exercice 2 : Conversion de Température

#Ce programme Python contient plusieurs erreurs

celsius = input("Entrez la température en degrés Celsius : ")
fahrenheit =
# Afficher la Variable fahrenheit
print("Température en degrés Fahrenheit :")
# Formatage de Chaînes
print("Température en degrés Fahrenheit :")
#f-strings
print("Température en degrés Fahrenheit :")
```