Python

Rappels des bases de python

Antonin Kenzi

09.06.2023

Table des matières

qu’est ce que python ?

Historique de Python :

Python a été créé par Guido van Rossum et sa première version a été publiée en 1991. Le nom du langage a été inspiré par la passion de Guido pour la série télévisée britannique “Monty Python’s Flying Circus”. Depuis lors, Python a connu une croissance exponentielle de sa popularité et est devenu l’un des langages de programmation les plus utilisés dans le monde.

Installation de Python :

Pour commencer à programmer en Python, vous devez d’abord installer l’interpréteur Python sur votre système. Voici les étapes générales pour l’installation de Python :

Rendez-vous sur le site officiel de Python (https://www.python.org) et téléchargez la dernière version stable de Python adaptée à votre système d’exploitation (Windows, macOS, Linux, etc.).

Lancez l’installateur téléchargé et suivez les instructions à l’écran. Assurez-vous de cocher la case “Add Python to PATH” (Ajouter Python au chemin d’accès) lors de l’installation, afin que vous puissiez exécuter des scripts Python depuis n’importe quel répertoire de votre système.

Une fois l’installation terminée, vous pouvez ouvrir l’invite de commande (sur Windows, utilisez PowerShell ou CMD) ou un terminal (sur macOS et Linux) et tapez la commande python ou python3 pour vérifier que Python est correctement installé. Vous devriez voir s’afficher la version de Python installée et l’invite de commande Python (»>) prête à recevoir du code.

Premiers pas avec l’interpréteur Python :

Une fois que vous avez installé Python, vous pouvez commencer à interagir avec l’interpréteur Python en mode interactif. Voici quelques étapes pour démarrer :

Ouvrez l’invite de commande ou le terminal.

Tapez python ou python3 pour lancer l’interpréteur Python.

Vous verrez l’invite de commande Python (»>) prête à recevoir du code.

charte de python

La “Charte de Python” est un ensemble de principes directeurs et de valeurs qui guident le développement et l’évolution du langage Python. Elle est souvent résumée par le document appelé “The Zen of Python” (L’art de Python), qui peut être consulté en tapant import this dans l’interpréteur Python.

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

soit :

```
Beau vaut mieux que laid.
Explicite vaut mieux qu'implicite.
Simple vaut mieux que complexe.
Complexe vaut mieux que compliqué.
Plat vaut mieux que imbriqué.
Dispersé vaut mieux que dense.
La lisibilité compte.
Les cas spéciaux ne sont pas assez spéciaux pour enfreindre les règles.
Bien que la praticité l'emporte sur la pureté.
Les erreurs ne doivent jamais passer silencieusement.
À moins d'être explicitement réduites au silence.
Face à l'ambiguïté, refusez la tentation de deviner.
Il devrait y avoir une façon -- et de préférence une seule -- évidente de le
    faire.
Bien que cette façon puisse ne pas être évidente au premier abord, à moins d'ê
    tre néerlandais.
Maintenant vaut mieux que jamais.
Bien que jamais soit souvent mieux que maintenant.
Si la mise en œuvre est difficile à expliquer, c'est une mauvaise idée.
Si la mise en œuvre est facile à expliquer, c'est peut-être une bonne idée.
Les espaces de noms sont une excellente idée - faisons-en plus !
```

Voici les principes clés de la charte de Python (un peu plus expliqué) :

- Lisibilité compte : Le code Python doit être facile à lire et à comprendre. Il doit favoriser la clarté plutôt que la complexité.
- Explicit is better than implicit (L'explicite vaut mieux que l'implicite) : Le code Python doit être explicite et non ambigu. Les choses doivent être exprimées de manière claire plutôt que laissées à deviner.
- Les exceptions ne doivent pas être ignorées : Les erreurs et les exceptions ne doivent pas être ignorées. Il est préférable de les traiter et de gérer les situations exceptionnelles correctement.
- La simplicité : Le langage Python encourage la simplicité et l'élégance du code. Les solutions simples sont préférées aux solutions compliquées.
- Lisibilité du code : Le code Python doit être écrit de manière à être facilement lu et compris par les autres développeurs. La lisibilité est une priorité.
- Les modules : Les fonctionnalités doivent être regroupées en modules plutôt que d'avoir un code monolithique. Les modules permettent une meilleure organisation et réutilisation du code.
- L'importance de la communauté : Python valorise et encourage une communauté active et participative. La collaboration, le partage de connaissances et l'entraide sont essentiels.
- La diversité : Python accueille et encourage la diversité des utilisateurs, des contributeurs et des idées. Il s'efforce d'être inclusif et ouvert à tous.
- L'évolution continue : Python est un langage en constante évolution. Il s'adapte et se développe pour répondre aux besoins changeants des utilisateurs et de la technologie.

Initiation

Structure de donnée

Scalaires

Les scalaires sont des types de données individuels qui ne peuvent pas être subdivisés.

- **Integers :**
  représentent les nombres entiers, tels que 1, 2, -3, etc.
- **Float :**
  représentent les nombres décimaux, tels que 3.14, 2.5, -0.75, etc.
- **Complex :**
  représentent les nombres complexes, tels que 1+2j, 3-4j, etc.
- **String :**
  représentent les chaînes de caractères, telles que “Bonjour”, “Python”, etc.
- **Boolean :**
  représentent les valeurs de vérité, True ou False.
- **None :**
  représente une valeur spéciale indiquant l’absence de valeur.

Conteneurs

Les conteneurs sont des structures de données qui peuvent contenir plusieurs éléments.

- **Liste(List) :**
  représentées par des crochets [], permettent de stocker une séquence ordonnée d’éléments.
  `Exemples : [1, 2, 3], ["a", "b", "c"].`
- **Tuples (Tuple) :**
  représentés par des parenthèses (), permettent de stocker une séquence ordonnée d’éléments. Les tuples sont immuables, c’est-à-dire qu’ils ne peuvent pas être modifiés une fois créés.
- **Sets (Set) :** représentés par des accolades {}, permettent de stocker des éléments uniques, sans ordre particulier.
- **Dictionnaires (Dictionary) :**
  représentés par des accolades {}, permettent de stocker des paires clé-valeur. Chaque élément du dictionnaire est constitué d’une clé et d’une valeur associée.
  `Exemple : {23: 'deux-trois'}`

Le tupple et le dictionnaire sont des conteneurs hashable, c'est-à-dire qu'ils peuvent être utilisés comme clé dans un dictionnaire ou comme élément d'un set.

```python
# Création d'une Hashtable
hashtable = {}

# Ajout d'éléments à la Hashtable
hashtable["fruit"] = "pomme"
hashtable["animal"] = "chien"
hashtable["couleur"] = "rouge"

# Accès aux éléments de la Hashtable par la clé
print(hashtable["fruit"])  # Affiche "pomme
print(hashtable["animal"])  # Affiche "chien"
print(hashtable["couleur"])  # Affiche "rouge"

# Modification d'un élément dans la Hashtable
hashtable["fruit"] = "banane"
print(hashtable["fruit"])  # Affiche "banane"

# Suppression d'un élément de la Hashtable
del hashtable["animal"]
print(hashtable.get("animal"))  # Affiche None (l'élément a été supprimé)
```

Structure de contrôle

Condition

Les instructions conditionnelles permettent d'exécuter un bloc de code si une condition est vérifiée.

```python
# Exemple d'instruction conditionnelle
if x > 0:
    print("x est positif")
elif x == 0:
    print("x est nul")
else:
    print("x est négatif")
```

Boucle

Les boucles permettent d'exécuter un bloc de code plusieurs fois.

La boucle for est une structure de contrôle en Python utilisée pour itérer sur une séquence d'éléments, tels qu'une liste, un tuple, une chaîne de caractères, etc. La syntaxe générale de la boucle for est la suivante :

```python
i = 0
# Exemple d'instruction break
for i in range(2):
    print(i)
# Résultat :
# 0
# 1
```

La boucle while est une structure de contrôle en Python qui permet d'exécuter un bloc de code tant qu'une condition donnée est évaluée à True. La syntaxe générale de la boucle while est la suivante :

```python
# Exemple de boucle while
i = 0
while i < 2:
    print(i)
    i += 1
# Résultat :
# 0
# 1
```

Les compréhensions de listes (list comprehensions) sont une fonctionnalité puissante de Python qui permet de créer facilement des listes à partir d'une syntaxe concise.

```python
liste_carres_pairs = [x**2 for x in range(1, 11) if x % 2 == 0]
# Affichage de la liste résultante
print(liste_carres_pairs)
# [4, 16, 36, 64, 100]
```

Instruction de contrôle

break : permet de sortir d'une boucle

continue : permet de passer à l'itération suivante d'une boucle

pass : permet de ne rien faire

```python
# Exemple d'instruction break
for i in range(5):
    if i == 2:
        break
    print(i)

# Résultat :
# 0
# 1
```

```python
# Exemple d'instruction continue
for i in range(3):
    if i == 2:
        continue
    print(i)

# Résultat :
# 0
# 1
# 3
```

```python
# Exemple d'instruction pass
for i in range(4):
    if i == 2:
        pass
    else :
        print(i)

# Résultat :
# 0
# 1
# 3
```

Opérateur

Un opérateur est un symbole ou un mot-clé utilisé dans un langage de programmation pour effectuer une opération sur des valeurs ou des variables.

Classic (liste non exhaustive):

1. **Opérateurs arithmétiques**

```
    x = 5 + 3   # Addition
    y = 10 - 2  # Soustraction
    z = 4 * 2   # Multiplication
    w = 15 / 3  # Division
    h = 15//3   # Division entière
```

2. **Opérateurs de comparaison :**

```
a = 5 > 3   # Supériorité
b = 10 != 5  # Inégalité
c = 2 <= 8  # Infériorité ou égalité
```

3. **Opérateurs logiques :**

```
condition1 = (x > 0) and (y < 10)  # Opérateur logique ET
condition2 = (a == True) or (b == True)  # Opérateur logique OU
```

5. **Opérateurs d'affectation :**

```
x = 10  # Affectation simple
x += 5  # Addition et affectation
y -= 3  # Soustraction et affectation
```

6. **Opérateurs de concaténation :**

```
chaine1 = "Bonjour"
resultat = chaine1 + " Python"  # Concaténation de chaînes

noms = ["Alice", "Bob", "Charlie"]
ages = [25, 30, 35]
villes = ["Paris", "New York", "Londres"]

# Utilisation de l'opérateur zip pour regrouper les éléments correspondants
personnes = zip(noms, ages, villes)

# Parcours des éléments regroupés
for personne in personnes:
    nom, age, ville = personne
    print(f"{nom} a {age} ans et vit à {ville}")
```

Particuliers

7. **Opérateurs de membres :**
   . : Accès aux attributs et méthodes d'un objet.
   [] : Accès aux éléments d'une liste, d'un tuple ou d'un dictionnaire.

```python
liste = [1, 2, 3, 4, 5]
liste.append(6)

print(liste)  # Accès à l'élément à l'indice 2 de la liste
# [1,2,3,4,5,6]
```

```python
personne = {
    "nom": "Jean",
    "age": 30,
    "ville": "Paris"
}
print(personne.nom)  # Accès à l'attribut 'nom' de l'objet 'personne'
```

8. **Opérateurs d'appartenance :**
   in : Teste si un élément appartient à une séquence.
   not in : Teste si un élément n'appartient pas à une séquence.

```python
fruits = ["pomme", "banane", "orange"]
print("pomme" in fruits)  # Vérifie si "pomme" est dans la liste 'fruits'
print("raisin" not in fruits) # Vérifie si "raisin" n'est pas dans la liste '
    fruits'
```

9. **Opérateurs d'identité :**
   is : Teste si deux objets sont identiques.
   is not : Teste si deux objets ne sont pas identiques.

```python
# Opérateur 'is'
x = [1, 2, 3]
y = x
print(y is x)  # Vérifie si 'y' et 'x' font référence au même objet
# Résultat
# True

# Opérateur 'is not'
a = 5
b = 10
print(a is not b)  # Vérifie si 'a' et 'b' ne font pas référence au même objet
# Résultat
# True
```

10. **Opérateurs ternaires :**

```python
# Opérateur ternaire
condition = True
resultat = "Condition vérifiée" if condition else "Condition non vérifiée"
```

11. **Opérateur de déréférencement :**

```python
liste = [1, 2, 3, 4, 5]
a, *b, c = liste

print(a)  # Affiche 1
print(b)  # Affiche [2, 3, 4]
print(c)  # Affiche 5

def operate(a, b, **kwargs):
        if 'add' in kwargs:
            print(f"{a}+{b}={a+b}")
        if 'sub' in kwargs:
            print(f"{a}-{b}={a-b}")


# Définition de deux dictionnaires
informations_base = {'nom': 'Alice', 'age': 25}
informations_supplementaires = {'ville': 'Paris', 'profession': 'Ingénieur'}

# Utilisation de l'opérateur ** pour fusionner les dictionnaires
informations_combinees = {**informations_base, **informations_supplementaires}

# Affichage du dictionnaire combiné
print(informations_combinees) # Affiche {'nom': 'Alice', 'age': 25, 'ville': '
    Paris', 'profession': 'Ingénieur'}
```

Fonctions

Une fonction est un bloc de code qui peut être appelé pour effectuer une tâche spécifique. Une fonction peut avoir des paramètres et renvoyer une valeur.

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

- **a** et **b** sont les paramètres de la fonction. Ils peuvent être utilisés dans le corps de la fonction.
- **return** est un mot-clé qui permet de renvoyer une valeur à l'appelant de la fonction.

Il existe de nombreuses fonctions prédéfinies (aussi appelées fonctions intégrées ou fonctions natives) en Python qui sont disponibles pour une utilisation directe dans vos programmes. Voici quelques exemples de fonctions prédéfinies couramment utilisées en Python :

Telles que :

- **print()** : affiche un message à l'écran.
- **len()** : renvoie la longueur d'une séquence.
- **all()** : renvoie True si tous les éléments d'une séquence sont True.
- **any()** : renvoie True si au moins un élément d'une séquence est True.
- **enumerate()** : renvoie un objet énumérable.
- **max()** : renvoie le plus grand élément d'une séquence.
- **min()** : renvoie le plus petit élément d'une séquence.
- **range()** : renvoie une séquence de nombres.

Et bien d'autres encore…

Pour simplifier le code quelques fonctions sont disponibles.

- **map** : applique une fonction à chaque élément d'une séquence.

```
def carre(x):
    return x**2

liste = [1, 2, 3, 4, 5]
resultat = map(carre, liste)
print(list(resultat))  # Renvoie [1, 4, 9, 16, 25]
```

- **filter** : filtre les éléments d'une séquence.

```python
def est_pair(x):
    return x % 2 == 0

liste = [1, 2, 3, 4, 5]
resultat = filter(est_pair, liste)
print(list(resultat))  # Renvoie [2, 4]
```

Lors de la définition d'une fonction en Python, les paramètres spéciaux *args et **kwargs peuvent être utilisés pour accepter un nombre variable d'arguments positionnels et d'arguments nommés.

args :

L'usage de *args permet de capturer un nombre variable d'arguments positionnels (ordre important) et de les regrouper dans un tuple.

kwargs :

L'usage de **kwargs permet de capturer un nombre variable d'arguments nommés et de les regrouper dans un dictionnaire.

```python
def fonction_example(*args, **kwargs):
    for arg in args:
        print("Argument positionnel :", arg)

    for cle, valeur in kwargs.items():
        print("Argument nommé :", cle, "=", valeur)

# Appel de la fonction avec différents arguments
fonction_example(1, 2, 3, nom='Alice', age=25) # Affiche :
# Argument positionnel : 1
# Argument positionnel : 2
# Argument positionnel : 3
# Argument nommé : nom = Alice
# Argument nommé : age = 25
```

Exercice première partie

Énoncé de l'exercice :

Écrivez un programme Python qui demande à l'utilisateur de saisir le nom, l'âge et le pays d'origine de trois personnes **(Utiliser la fonction input())**, puis stocke ces informations dans une liste de dictionnaires. Ensuite, le programme doit afficher les informations de chaque personne en utilisant une boucle for.

Correction :

```python
# Création de la liste pour stocker les informations des personnes
personnes = []

# Saisie des informations pour chaque personne
for i in range(1, 4):
    print("Saisie des informations pour la personne", i, ":")
    nom = input("Nom : ")
    age = input("Âge : ")
    pays = input("Pays d'origine : ")

    # Création d'un dictionnaire pour stocker les informations de la personne
    personne = {"Nom": nom, "Âge": age, "Pays d'origine": pays}

    # Ajout du dictionnaire à la liste des personnes
    personnes.append(personne)

# Affichage des informations de chaque personne
print("\nAffichage des informations :")
for i, personne in enumerate(personnes):
    print("Personne", i + 1, ":")
    print("Nom :", personne["Nom"])
    print("Âge :", personne["Âge"])
    print("Pays d'origine :", personne["Pays d'origine"])
    print()
```

Aprofondisement des bases

Classe et objets

En Python, les classes sont des structures qui permettent de définir des objets avec leurs propres attributs (variables) et méthodes (fonctions). Un objet est une instance d'une classe.

Le terme **self** est une convention utilisée en Python pour faire référence à l'instance d'un objet lui-même à l'intérieur de ses propres méthodes. Il s'agit d'un paramètre implicite que vous devez inclure en tant que premier paramètre dans la définition de chaque méthode d'une classe.

- La méthode **init** est l'initialisateur de classe. Elle est appelée automatiquement lors de la création d'un nouvel objet à partir de la classe.
- La méthode **next** est utilisée dans la définition d'un itérateur. Elle est appelée pour obtenir l'élément suivant d'une séquence lorsque l'itérateur est utilisé.
- La méthode **iter** est utilisée pour retourner un itérateur sur une séquence dans une classe.

```python
class NombrePairs:
    def __init__(self, limite):
        self.limite = limite
        self.nombre = 0

    def __iter__(self):
        return self

    def __next__(self):
        if self.nombre >= self.limite:
            raise StopIteration
        else:
            resultat = self.nombre
            self.nombre += 2
            return resultat


# Utilisation de la classe NombrePairs pour créer l'objet nombres
nombres = NombrePairs(10)  # Crée une instance de la classe avec une limite de 10

for nombre in nombres:
    print(nombre)

# Résultat :
# 0
# 2
# 4
# 6
# 8
```

Gestions des erreurs

Lors de l'exécution d'un programme, des erreurs peuvent survenir. Ces erreurs sont appelées des exceptions. Il est possible de gérer ces exceptions avec des blocs.

1. try/except/else/finally

```python
try:
    # Code susceptible de provoquer une exception
    pass
except:
    # Code à exécuter en cas d'exception levée dans le bloc try
    pass
else:
    # Code à exécuter si aucune exception n'est levée dans le bloc try
    pass
finally:
    # Code à exécuter dans tous les cas
    pass
```

Entrée/sortie

Les entrées/sorties (E/S) sont des opérations essentielles dans la programmation, car elles permettent aux programmes de communiquer avec l'utilisateur et de manipuler des fichiers. En Python, il existe plusieurs moyens de réaliser des E/S.

Bien sûr ! Voici un résumé de la section sur les entrées/sorties en Python, avec un exemple d'utilisation :

Les entrées/sorties (E/S) sont des opérations essentielles dans la programmation, car elles permettent aux programmes de communiquer avec l'utilisateur et de manipuler des fichiers. En Python, il existe plusieurs moyens de réaliser des E/S.

Lecture à partir de fichiers :

Pour lire le contenu d'un fichier, utilisez la fonction open() en mode lecture ('r')

```
fichier = open('exemple.txt', 'r')
ligne = fichier.readline()
print(ligne)
fichier.close()
```

Écriture dans des fichiers :

Pour écrire dans un fichier, ouvrez-le en mode écriture ('w'). Utilisez ensuite la méthode write() pour écrire du texte dans le fichier.

```
fichier = open('nouveau.txt', 'w')
fichier.write("Ceci est un exemple.")
fichier.close()
```

Fermeture de fichiers :

Il est important de fermer correctement les fichiers après les avoir utilisés pour libérer les ressources système. Utilisez la méthode close() pour fermer un fichier. Pour éviter d'oublier de le faire, vous pouvez utiliser le bloc with. Par exemple :

```
with open('exemple.txt', 'r') as fichier:
ligne = fichier.readline()
print(ligne)
```

Entrées de l'utilisateur :

La fonction input() permet de demander à l'utilisateur de saisir des données via la console. Le résultat est généralement stocké dans une variable pour une utilisation ultérieure. Par exemple :

```
1 nom = input("Quel est votre nom ? ")
2 print("Bonjour, " + nom + " !")
```

Formatage de chaînes de caractères :

Pour créer des sorties plus lisibles, vous pouvez formater des chaînes de caractères avec des valeurs variables. Les f-strings sont une méthode pratique pour formater les chaînes de caractères en incluant des variables dans le texte. De plus, cela permettent également d'utiliser des expressions Python dans les accolades. Pour utiliser des f-strings, ajoutez un f avant les guillemets de la chaîne et placez les variables souhaitées entre accolades {}. Par exemple :

```
1 nom = "Alice"
2 age = 25
3 message = f"Je m'appelle {nom} et j'ai {age} ans."
4 print(message)
5 a = 5
6 b = 3
7 resultat = f"Le résultat de {a} + {b} est {a + b}."
8 print(resultat)
```

Des notations permettent de choisir le format d'écritude : 1. {} : Cette notation est utilisée pour insérer une valeur sans spécifier de format particulier. Par exemple : {}.

2. {:<width} : Cette notation spécifie une largeur minimale pour l'insertion et aligne la valeur à gauche. Par exemple : {:<10}.
3. {:>width} : Cette notation spécifie une largeur minimale pour l'insertion et aligne la valeur à droite. Par exemple : {:>10}.
4. {:^width} : Cette notation spécifie une largeur minimale pour l'insertion et centre la valeur. Par exemple : {:^10}.
5. {:.precisionf} : Cette notation spécifie une précision décimale pour les valeurs flottantes. Par exemple : {:.2f} permet d'afficher seulement deux chiffres après la virgule.
6. {:+} : Cette notation force l'affichage d'un signe "+" pour les valeurs numériques positives. Par exemple : {:+}.
7. {:0width} : Cette notation spécifie une largeur minimale pour l'insertion et remplit les espaces vides avec des zéros. Par exemple : {:04}.

Exercice Aprofondisement

Énoncé de l'exercice :

Créez une classe Etudiant qui représente un étudiant avec les attributs suivants :

- nom : le nom de l'étudiant (chaîne de caractères)
- age : l'âge de l'étudiant (entier)
- notes : une liste de notes de l'étudiant (liste de nombres à virgule flottante entre 1 et 6)

La classe Etudiant devrait avoir les méthodes suivantes :

- ajouter_note : ajoute une note à la liste des notes de l'étudiant
- calculer_moyenne : calcule et retourne la moyenne des notes de l'étudiant afficher_informations : affiche les informations de l'étudiant (nom, âge et moyenne des notes)
- Assurez-vous de gérer les erreurs potentielles, comme par exemple si l'âge n'est pas un entier valide, note invalide et/ou si la liste des notes est vide.

Ensuite, créez un programme principal qui demande à l'utilisateur d'entrer les informations d'un étudiant (nom, âge et notes) et utilise la classe Etudiant pour créer une instance de cet étudiant, ajouter les notes, calculer la moyenne et afficher les informations de l'étudiant à l'écran.

Correction :

```python
class Etudiant:
    def __init__(self, nom, age):
        self.nom = nom
        self.age = age
        self.notes = []

    def ajouter_note(self, note):
        if note < 1 or note > 6:
            raise ValueError("Note invalide ! La note doit être entre 1 et 6.")
        self.notes.append(note)

    def calculer_moyenne(self):
        if not self.notes:
            raise ValueError("La liste des notes est vide !")
        return sum(self.notes) / len(self.notes)

    def afficher_informations(self):
        print("Informations de l'étudiant :")
        print("Nom :", self.nom)
        print("Âge :", self.age)
        try:
            moyenne = self.calculer_moyenne()
            print("Moyenne des notes :", moyenne)
        except ValueError as e:
            print("Erreur :", str(e))


# Programme principal
nom = input("Entrez le nom de l'étudiant : ")
age = input("Entrez l'âge de l'étudiant : ")

try:
    age = int(age)
except ValueError:
    print("Erreur : L'âge doit être un entier valide.")
    exit()

etudiant = Etudiant(nom, age)

notes = input("Entrez les notes de l'étudiant (séparées par des espaces) : ").
    split()
for note in notes:
    try:
        note = float(note)
        etudiant.ajouter_note(note)
    except ValueError:
        print("Erreur : La note doit être un nombre valide.")
        exit()

etudiant.afficher_informations()
```

Allez plus loin

Python est un language OpenSource. cela veux dire qu'une communoté alimmente le language par beaucoup de contribution.

Python est un langage de programmation open source. Cela signifie que son code source est disponible publiquement et que la communauté Python peut contribuer à son développement et à son amélioration.

Paquets

Python possède un grand nombre de Paquets qui permettent d'ajouter des fonctionnalités à Python.

Certains Paquets sont inclus dans Python, d'autres doivent être installées. Le site pypi permet de trouver des packages.

Pour utiliser un module ou un package, il faut l'importer.

```
1 import math
2
3 print(math.pi)  # Affiche la valeur de pi
```

Il est possible d'importer uniquement une partie d'un module.

```
1 from math import pi
2
3 print(pi)  # Affiche la valeur de pi
```

Dans ces paquets, il y a des modules qui sont très utiles pour le développement d'applications.

math

Le module math contient des fonctions mathématiques.

random

Le module random contient des fonctions pour générer des nombres aléatoires.

datetime

Le module datetime contient des classes pour manipuler des dates et des heures.

os

Le module os contient des fonctions pour interagir avec le système d'exploitation.

sys

Le module sys contient des fonctions et des variables qui permettent d'interagir avec l'interpréteur Python.

NamedTuple

Le module collections contient la classe NamedTuple qui permet de créer des tuples nommés.

Cela permet de créer des objets immuables avec des attributs nommés. Ce qui rend le code plus lisible.

```python
from collections import namedtuple

# Définition d'un NamedTuple pour représenter une personne
Personne = namedtuple('Personne', ['nom', 'age', 'ville'])

# Création d'une instance de Personne
alice = Personne('Alice', 25, 'Paris')

# Accès aux éléments du tuple nommé
print(alice.nom)   # Affiche 'Alice'
print(alice.age)   # Affiche 25
print(alice.ville) # Affiche 'Paris'
```

Gestionnaire de paquets

Le téléchargement de paquets Python se fait généralement à l'aide d'un gestionnaire de paquets tel que pip. Un gestionnaire de paquets est un outil qui facilite l'installation, la mise à jour et la suppression de paquets ou de modules Python.

Par exemple, pour installer le paquet numpy, vous pouvez exécuter `pip install numpy`. Le gestionnaire de paquets téléchargera automatiquement les fichiers nécessaires à partir du Python Package Index (PyPI) et les installera sur votre système.

Numpy

Site : https://numpy.org/

Numpy est un paquets qui permet de manipuler des tableaux multidimensionnels et des matrices. Ce paquets est très utilisé pour tout ce qui est des calculs scientifiques.

```python
import numpy as np
from collections import namedtuple

# Définition d'un NamedTuple pour représenter une coordonnée
Coordonnee = namedtuple('Coordonnee', ['x', 'y'])

# Création d'un tableau NumPy de coordonnées
coordonnees = np.array([Coordonnee(1, 2), Coordonnee(3, 4), Coordonnee(5, 6)])

# Accès aux éléments du tableau
print(coordonnees[0].x)  # Affiche 1
print(coordonnees[1].y)  # Affiche 4

# Broadcasting

# Création de deux tableaux NumPy
a = np.array([1, 2, 3])
b = np.array([10, 20, 30])

# Addition des tableaux
result = a + b

print(result) # Affiche [11 22 33]

#récupérer les deux éléments d'un tableau qui vérifient une condition

a = np.array([1, 2, 3, 4, 5, 6])

b = a[a > 3] # Récupère les éléments de a qui sont supérieurs à 3
print(b) # Affiche [4 5 6]

#récupérer les deux première éléments d'un tableau
print(a[:2]) # Affiche [1 2]
#récupérer les deux dernière éléments d'un tableau
print(a[-2:]) # Affiche [5 6]
```

Matplotlib

Site : https://matplotlib.org/

Matplotlib est une bibliothèque de visualisation de données en Python très populaire. Elle offre une grande flexibilité pour créer des graphiques et des visualisations de manière simple et efficace.

```python
import matplotlib.pyplot as plt
import numpy as np


# Création d'une figure
fig = plt.figure()

#Creation d'un vecteur x
x = np.linspace(0, 2, 100)

# Création d'un vecteur y
y = x ** 2
# Création d'un graphique
plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('y')
plt.title('y = x^2')
plt.grid(True)

#savegarde du graphique
fig.savefig('graphique.png')

# Affichage du graphique
plt.show()
```

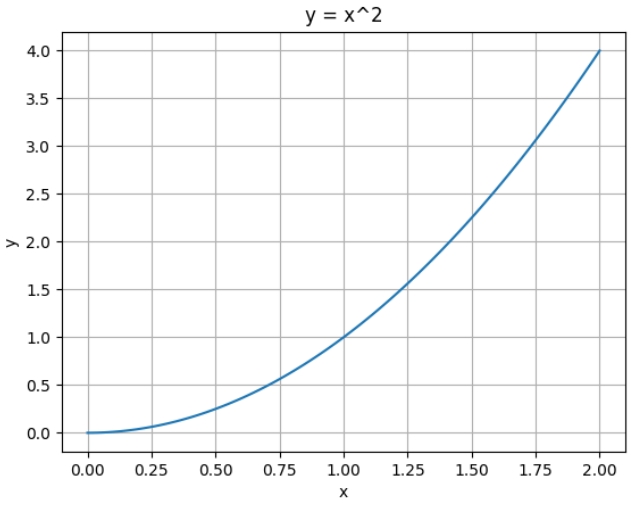

Figure 1: Résultat de l'exemple

Click

Site : https://click.palletsprojects.com/en/8.1.x/

Click est un paquets qui permet de créer des interfaces en ligne de commande.

Elle permet de créer des commandes avec des options et des arguments.

Voici quelques points clés concernant Click :

1. Définition des commandes :
   Définissez facilement des commandes en tant que fonctions Python avec le décorateur @click.command().
2. Options et arguments :
   Utilisez les décorateurs de Click pour définir des options et des arguments associés à vos commandes.
3. Parsing des arguments :
   Click gère la conversion et la validation des arguments en fonction de leurs types définis.
4. Gestion des erreurs :
   Définissez des gestionnaires d'erreurs personnalisés pour afficher des messages ou effectuer des actions spécifiques en cas d'erreur
5. Génération d'aide automatique :
   Click génère automatiquement une aide complète pour vos commandes et options.
6. Personnalisation de l'interface :
   Utilisez les options de personnalisation de Click pour ajuster l'apparence de l'interface de ligne de commande.

```python
import click

@click.group()
def cli():
    pass

@cli.command()
@click.option('--name', prompt='Your name', help='Your name')
def greet(name):
    click.echo(f"Hello, {name}!")

@cli.command()
@click.option('--count', type=int, default=1)
def repeat(count):
    for _ in range(count):
        click.echo("Hello!")

if __name__ == '__main__':
    cli()
```

```
$ python script.py greet
Your name: John
Hello, John!

$ python script.py repeat --count 3
Hello!
Hello!
Hello!
```

Flask

Site : https://flask.palletsprojects.com/en/2.3.x/

Flask est un paquets qui permet de créer des applications web.

Il est possible de créer des routes qui permettent de définir des URL.

Souvent on ajoute des templates HTML avec le paquets **jinja2** et des zone de remplissage par variables avec **mustache**.
Code python :

```python
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')

def hello_world():
    return render_template('index.html', name='John')
```

Code Html:

```html
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Mon site</title>
</head>
<body>
    <h1>Hello, {{ name }}!</h1>
</body>
</html>
```

Environnement virtuel

Un environnement virtuel est un environnement Python isolé qui permet de gérer les dépendances de projets Python.

Il permet de créer un environnement de développement spécifique à un projet.

Il est possible d'installer des paquets spécifiques à un projet sans les installer globalement sur le système.

Il est possible de créer un environnement virtuel avec le module venv.

Installation :

1. Installer le module venv

```
pip install virtualenv
```

2. Créer un dossier pour le projet
3. Se placer dans le dossier
4. Créer l'environnement virtuel

```
python -m venv env
```

5. Activer l'environnement virtuel

- Windows

```
env\Scripts\activate.bat
```

- Linux

```
source env/bin/activate
```

6. Désactiver l'environnement virtuel

- Windows

```
env\Scripts\deactivate.bat
```

- Linux

```
deactivate
```

outils logiciels

Ipython

IPython est un shell interactif pour le langage de programmation Python qui offre des fonctionnalités telles que la complétion de code, l'historique des commandes, la coloration syntaxique, etc.

Installation :

```
pip install ipython
```

Lançer Ipython

```
ipython
```

Jupyter

Jupyter est une application web utilisée pour programmer dans plus de 40 langages de programmation, dont Python, Julia, Ruby, R, ou encore Scala2. Jupyter est une évolution du projet IPython. Son nom est un hommage aux trois langages de programmation principaux supportés par Jupyter, à savoir Julia, Python et R.

Installation :

```
pip install jupyter
```

Lancer le server Jupyter

```
jupyter notebook
```

Ouvrir le navigateur et taper l'url : http://localhost:8888/ ou clique sur le lien qui s'affiche dans la console.

Il est possible de lancer Jupyter notebook et d'ouvrir une console python dans le même environnement virtuel.

```
ipython console --existing
```

A vous de jouer

Bonne suite d'aprentisage ou de rappel. Pensé que ce n'est jamais vraiment fini