Python - rappels

Rappels des bases de python

Antonin Kenzi Clément Dieperink

06.06.2023

Table des matières

Cours python : base

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.
- **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'}`

```
# 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
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)
```

- Sets (Set) : représentés par des accolades {}, permettent de stocker des éléments uniques, sans ordre particulier.

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.

Classe

En Python, les classes sont des structures qui permettent de définir des objets avec leurs propres attributs (variables) et méthodes (fonctions).

- 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
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
```

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 :

1. **Opérateurs arithmétiques**

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

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
condition3 = not condition1  # Opérateur logique NON
```

5. **Opérateurs d'affectation :**

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

6. **Opérateurs de concaténation :**

```
chaine1 = "Bonjour"
chaine2 = "Python"
resultat = chaine1 + " " + chaine2  # 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.

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

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.

```
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.

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

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

10. **Opérateurs ternaires :**

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

11. **Opérateur de déréférencement :**

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

print(a)  # Affiche 1
```

```python
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', 'profe
```

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.

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

- **parametre1** et **parametre2** 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.

Des fonctions prédéfinies sont disponibles dans 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.

```python
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 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
```