11_python_data_types

November 27, 2023

1 Python Data Types

En programmation, `data type` (type de données) est un concept important. Les variables peuvent stocker des données de différents types, et les différents types peuvent faire différentes choses. Python a les types de données suivants intégrés par défaut, dans ces catégories:

Data Type	Classes	Description
Numeric	int, float, complex	Stoque les valeurs numériques
String	str	Stoque les chaînes de caractères
Sequenece	list, tuple, range	Stoque des collection d'objets
Mapping	dict	Stoque les données sour forme de key/value
Boolean	bool	Stoque soit `True` soit `False`
Set	set, frozenset	Stoque une collection d'objet, **unique**

Comme tout est objet en python, chaque `Data Type` sont des `Classes` et les variables sont des `instances(Objet)` des ces classes.

2 Python Numeric Data Type

Les données numériques sont utilisées pour stocker des valeurs numériques. Les objets numériques sont créés lorsque vous assignez une valeur numérique à eux.

`Integers`, `Nombres à virgule flottante` et `Nombres complexes` sont tous des types de données numériques. Ils sont définis comme `int`, `float` et `complex` dans Python.

- `int` - Stoque des nombres entiers positifs ou négatifs sans décimales, sans limite de taille.

- `float` - Stoque des nombres, positifs ou négatifs, contenant un ou plusieurs décimales. Les nombres à virgule flottante peuvent également être scientifiques (avec un "e" pour indiquer la puissance de 10). ils sont précis jusqu'à 15 chiffres décimaux.
- `complex` - Stoque des nombres complexes, ils sont de la forme `a + bj`, où `a` et `b` sont des nombres et `j` (ou `J`) représente la racine carrée de -1 (qui est un nombre imaginaire). La partie réelle est `a`, et la partie imaginaire est `b`. Les nombres complexes ne peuvent pas être convertis en un autre type de nombre. `a` et `b` sont de type `float`.

Nous pouvons utiliser la fonction `type()` pour connaître le type de données d'une variable.

[ ]:
```
num1 = 5
print(num1, 'is of type', type(num1))

num2 = 2.0
print(num2, 'is of type', type(num2))

num3 = 1+2j
print(num3, 'is of type', type(num3))
```

- **5** est un integer, `type()` retourne `int` car la classe de `num1` est `<class 'int'>`
- **2.0** est un float, `type()` retourne `float` car la classe de `num2` est `<class 'float'>`
- **1+2j** est un complex, `type()` retourne `complex` car la classe de `num3` est `<class 'complex'>`

3 Python String Data Type

Les chaînes de caractères sont utilisées pour stocker des valeurs textuelles. Une chaîne est une séquence de caractères. Dans Python, les chaînes sont écrits avec des guillemets simples ou doubles.

[ ]:
```
name = 'Name'
print(name)

message = f"Python\n for {name} beginners"
print(message)
```

4 Python List Data Type

Les listes sont des collections ordonnées et modifiables. Dans Python, les listes sont écrites avec des crochets `[]`. Elles peuvent contenir des éléments de différents ou même types de données.

[ ]:
```
languages = ["Swift", "Java", "Python"]
```

Ici, nous avons crée une liste nommeée `languages` contenant trois `strings`

4.0.1 Accéder aux éléments d'une liste

Vous pouvez accéder aux éléments d'une liste en utilisant leurs index. L'index d'une liste commence à 0.

[ ]:
```
# access element at index 0
print(languages[0])   # Swift
languages[0] = "C++"
print(languages[0])
```

5 Python Tuple Data Type

Les tuples sont des collections ordonnées et non modifiables. Dans Python, les tuples sont écrits avec des parenthèses `()`. Ils peuvent contenir des éléments de différents ou même types de données. Les tupoles sont comme les listes, sauf qu'elles sont immuables (ne peuvent pas être modifiées).

[ ]:
```
product = ('Xbox', 499.99)
```

Ici, nous avons crée un `tuple` nommé `product` contenant un `string` et un `float`.

Pour accéder aux éléments d'un tuple, nous utilisons leurs index. L'index d'un tuple commence à 0. (Comme pour les listes)

[ ]:
```
# create a tuple
product = ('Microsoft', 'Xbox', 499.99)

# access element at index 0
print(product[0])   # Microsoft
product = ("Apple", "Macbook pro")
print(product[0])   # Microsoft


# access element at index 1
print(type(product[1])) # -> float
```

6 Python Set Data Type

Les sets sont utilisés pour stocker plusieurs éléments dans un seul conteneur. Les sets sont des collections non ordonnées et non indexées. Dans Python, les sets sont écrits avec des accolades `{}`. En mathématique un `Set` est une collection d'éléments uniques. En Python, les sets sont utilisés pour stocker des éléments uniques.

[ ]:
```
# create a set named student_id
student_id = {112, 114, 116, 118, 115, 119, 114, 114, 114}

# display student_id elements
print(student_id)

# display type of student_id
print(len(student_id)
```

7 Python Dictionary Data Type

Les dictionnaires sont utilisés pour stocker des données sous forme de paires clé/valeur (key/value). Un dictionnaire est une collection non ordonnée, modifiable et indexée. Dans Python, les dictionnaires sont écrits avec des accolades `{}`.

> **Note:** Les dictionnaires ne peuvent pas avoir deux clés avec le même nom. Les clés sont uniques dans un dictionnaire, alors que les valeurs peuvent ne pas l'être.
>
> **Note:** Ne pas confondre les `dictionnaires` et les `sets`. Ils sont tous les deux écrits avec des accolades `{}`, mais les sets n'ont pas de clés et les dictionnaires ont des clés. Les sets sont utilisés pour stocker des éléments uniques, et les dictionnaires sont utilisés pour stocker des données sous forme de paires clé/valeur.

[ ]:
```
# create a dictionary named capital_city
capital_city = {'Neppal': 'Kathmandu', 'Italy': "Rome", 'England': 'London'}
print(capital_city)

# On accède aux éléments du dictionnaire par leur clé
print(capital_city['Nepal'])  # prints Kathmandu

#print(capital_city['Kathmandu'])  # throws error message

# On peut aussi utiliser la méthode get()
print(capital_city.get('Nepal'))  # prints Kathmandu

print(capital_city.get('Kathmandu'))  # prints None

print(capital_city.get('Kathmandu', 'Country not found'))  # prints City not␣
↪found
```

8 Python Type Conversion

En programmation, il est parfois nécessaire de convertir les données d'un type à un autre. Python a deux types de conversion de données:

- **Implicit Type Conversion** - Ces conversions de données sont effectuées automatiquement par Python. On ne peut pas les contrôler.
- **Explicit Type Conversion** - Ces conversions de données sont effectuées par l'utilisateur en utilisant les fonctions prédéfinies Python comme `int()`, `float()`, `str()` etc.

8.0.1 Exemple 1: Conversion implicite de type

Dans cet exemple Python convertit le type `int` au type `float` implicitement (automatiquement). Nous n'avons pas besoin de faire quoi que ce soit. Python favorise la conversion du type de données inférieur (entier) vers le type de données supérieur (flottant) pour éviter toute perte de données.

[ ]:
```
integer_number = 123
float_number = 1.23
```

```
print((float_number))
float_number += True
print((float_number))
```

Dans l'exemple ci-dessus, nous avons ajouté un `int` et un `float`, et Python a converti le type `int` en `float` avant de faire l'addition. Le résultat du calcul est un `float`. Python a converti le type `int` en `float` implicitement (automatiquement). Lorsque Python rencontre une opération où les types de données sont différents, il effectue automatiquement une conversion pour garantir que les données de plus petite taille ne soient pas perdues. Par exemple, si vous effectuez une opération entre un entier (un type de données plus petit) et un flottant (un type de données plus grand), Python convertira automatiquement l'entier en flottant avant d'effectuer l'opération afin de préserver toutes les informations contenues dans l'entier. Cela garantit qu'aucune donnée ne soit perdue lors de la manipulation des différentes variables et types de données dans votre programme Python.

> **Note:** Python ne peut pas convertir le type `string` en `int` implicitement (automatiquement). Cela générera une erreur. Par exemple, si on essaye d'additionner un `string` et un `int`, Python ne peut pas convertir le `string` en `int` implicitement (automatiquement). Cela générera une erreur de type `TypeError`. La solution est d'utiliser la conversion explicite de type.

8.0.2 Exemple 2: Conversion explicite de type

La conversion explicite de type est faite par le programmeur en utilisant les fonctions prédéfinies Python comme `int()`, `float()`, `str()` etc. Ce type de conversion est également appelé `typecasting` car le programmeur force le type de données à changer.

[ ]:
```
num_string = '12.23'
num_integer = 0b1
f = 12.0
f /= 1

print("Data type of num_string before Type Casting:",type(num_string))

# explicit type conversion
num_string = float(num_string)

print("Data type of num_string after Type Casting:",type(num_string))

num_sum = num_integer + num_string

print("Sum:",num_sum)
print("Data type of num_sum:",type(num_sum))
```

Dans l'example ci-dessus, nous avons créé deux variables `num_string` et `num_integer` de type `str`et `int` respectivement.

`num_string = int(num_string)`

Ici, nous avons converti la variable `num_string` de type `str` en type `int` en utilisant la fonction

`int()`.

Une fois la conversion effectuée, python est capable d'additionner les deux variables `num_string` et `num_integer` car elles sont toutes les deux de type `int`.

9 Conclusion

1. `TypeConversion` est la conversion d'un type de données en un autre type de données.
2. Python a deux types de conversion de données:
   - **Implicit Type Conversion** - Ces conversions de données sont effectuées automatiquement par Python. On ne peut pas les contrôler.
   - **Explicit Type Conversion** - Ces conversions de données sont effectuées par l'utilisateur en utilisant les fonctions prédéfinies Python comme `int()`, `float()`, `str()` etc.
3. Python favorise la conversion du type de données inférieur (entier) vers le type de données supérieur (flottant) pour éviter toute perte de données.
4. Python ne peut pas convertir le type `string` en `int` implicitement (automatiquement). Cela générera une erreur. La solution est d'utiliser la conversion explicite de type. appellée `typecasting` car le programmeur force le type de données à changer.