Python - conversão de tipo

Fundição de tipo Python

Do ponto de vista da programação, uma conversão de tipo refere-se à conversão de um objeto de um tipo em outro. Aqui, aprenderemos sobre conversão de tipo na programação Python.

> *Python Type Casting é um processo no qual convertemos um literal de um tipo de dados em outro tipo de dados. Python oferece suporte a dois tipos de conversão -* ***implícita*** *e* ***explícita*** *.*

Em Python existem diferentes tipos de dados, como números, sequências, mapeamentos, etc. Pode haver uma situação em que você tenha os dados disponíveis de um tipo, mas queira usá-los de outra forma. Por exemplo, o usuário inseriu uma string, mas você deseja usá-la como um número. O mecanismo de conversão de tipos do Python permite fazer isso.

Casting implícito em Python

Quando qualquer compilador/interpretador de linguagem converte automaticamente um objeto de um tipo em outro, isso é chamado de **conversão automática ou implícita** . Python é uma linguagem fortemente tipada. Não permite a conversão automática de tipos entre tipos de dados não relacionados. Por exemplo, uma string não pode ser convertida em nenhum tipo de número. No entanto, um inteiro pode ser convertido em um float. Outras linguagens, como JavaScript, são linguagens de tipo fraco, onde um número inteiro é forçado em uma string para concatenação.

Observe que o requisito de memória de cada tipo de dados é diferente. Por exemplo, um objeto **inteiro** em Python ocupa 4 bytes de memória, enquanto um objeto **flutuante** precisa de 8 bytes por causa de sua parte fracionária. Conseqüentemente, o interpretador Python não converte automaticamente um **float** em **int** , porque isso resultará em perda de dados. Por outro lado, **int** pode ser facilmente convertido em **float** definindo sua parte fracionária como 0.

A conversão implícita de **int** para **float** ocorre quando qualquer operação aritmética em operandos **int** e **float** é realizada.

Considere que temos um, **int** e uma variável **float**

```
<<< a=10   # int object
<<< b=10.5 # float object
```

Para realizar sua adição, 10 - o objeto inteiro é atualizado para 10.0. É um float, mas equivalente ao seu valor numérico anterior. Agora podemos realizar a adição de dois carros alegóricos.

```
<<< c=a+b
<<< print (c)
20.5
```

Na conversão implícita de tipo, um objeto Python com tamanho de byte menor é atualizado para corresponder ao tamanho de byte maior de outro objeto na operação. Por exemplo, um objeto booleano é primeiro atualizado para int e depois para float, antes da adição com um objeto de ponto flutuante. No exemplo a seguir, tentamos adicionar um objeto booleano em um float, observe que True é igual a 1 e False é igual a 0.

```
a=True;
b=10.5;
c=a+b;

print (c);
```

Isso produzirá o seguinte resultado:

```
11.5
```

Casting explícito em Python

Embora a conversão automática ou implícita seja limitada à conversão de **int** para **float** , você pode usar as funções integradas do Python int(), float() e str() para realizar conversões explícitas, como string para inteiro.

Função Python int()

A função int() integrada do Python converte um literal inteiro em um objeto inteiro, um float em inteiro e uma string em inteiro se a própria string tiver uma representação literal de inteiro válida.

Usar **int()** com um objeto int como argumento é equivalente a declarar um objeto **int** diretamente.

```
<<< a = int(10)
<<< a
10
```

é o mesmo que -

```
<<< a = 10
<<< a
10
<<< type(a)
<class 'int>
```

Se o argumento para a função **int()** for um objeto flutuante ou uma expressão de ponto flutuante, ele retornará um objeto int. Por exemplo -

```
<<< a = int(10.5) #converts a float object to int
<<< a
10
<<< a = int(2*3.14) #expression results float, is converted to int
<<< a
6
<<< type(a)
<class 'int'>
```

A função **int()** também retorna o inteiro 1 se um objeto booleano for fornecido como argumento.

```
<<< a=int(True)
<<< a
1
<<< type(a)
<class 'int'>
```

String para inteiro

A função **int()** retorna um número inteiro de um objeto string, somente se contiver uma representação inteira válida.

```
<<< a = int("100")
<<< a
100
<<< type(a)
<class 'int'>
<<< a = ("10"+"01")
<<< a = int("10"+"01")
<<< a
1001
<<< type(a)
<class 'int'>
```

No entanto, se a string contiver uma representação não inteira, o Python gerará ValueError.

```
<<< a = int("10.5")
Traceback (most recent call last):
   File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '10.5'
<<< a = int("Hello World")
Traceback (most recent call last):
   File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'Hello World'
```

A função **int()** também retorna números inteiros de strings binárias, octais e hexadecimais. Para isso, a função necessita de um parâmetro base que deve ser 2, 8 ou 16 respectivamente. A string deve ter uma representação binária/octal/hexadecimal válida.

String binária para inteiro

A string deve ser composta apenas por 1 e 0, e a base deve ser 2.

```
<<< a = int("110011", 2)
<<< a
51
```

O equivalente decimal do número binário 110011 é 51.

String octal para inteiro

A string deve conter apenas de 0 a 7 dígitos e a base deve ser 8.

```
<<< a = int("20", 8)
<<< a
16
```

O equivalente decimal do octal 20 é 16.

String hexa-decimal para inteiro

A string deve conter apenas os símbolos hexadecimais, ou seja, 0-9 e A, B, C, D, E ou F. A base deve ser 16.

```
<<< a = int("2A9", 16)
<<< a
681
```

O equivalente decimal de Hexadecimal 2A9 é 681. Você pode verificar facilmente essas conversões com o aplicativo de calculadora no Windows, Ubuntu ou Smartphones.

A seguir está um exemplo para converter número, float e string em tipos de dados inteiros:

```
a = int(1)     # a will be 1
b = int(2.2)   # b will be 2
c = int("3")   # c will be 3

print (a)
print (b)
print (c)
```

Isso produz o seguinte resultado -

```
1
2
3
```


Função Python float()

O **float()** é uma função integrada em Python. Ele retorna um objeto float se o argumento for um literal float, um número inteiro ou uma string com representação de ponto flutuante válida.

Usar float() com um objeto float como argumento é equivalente a declarar um objeto float diretamente

```
<<< a = float(9.99)
<<< a
9.99
<<< type(a)
<class 'float'>
```

é o mesmo que -

```
<<< a = 9.99
<<< a
9.99
<<< type(a)
<class 'float'>
```

Se o argumento para a função **float()** for um número inteiro, o valor retornado será um ponto flutuante com parte fracionária definida como 0.

```
<<< a = float(100)
<<< a
100.0
<<< type(a)
<class 'float'>
```

A função **float()** retorna o objeto float de uma string, se a string contiver um número de ponto flutuante válido, caso contrário, ValueError será gerado.

```
<<< a = float("9.99")
<<< a
9.99
<<< type(a)
<class 'float'>
<<< a = float("1,234.50")
Traceback (most recent call last):
   File "<stdin>", line 1, in <module>
ValueError: could not convert string to float: '1,234.50'
```

O motivo do ValueError aqui é a presença de vírgula na string.

Para fins de conversão de string em float, a notação científica de ponto flutuante também é considerada válida.

```
<<< a = float("1.00E4")
<<< a
10000.0
<<< type(a)
<class 'float'>
<<< a = float("1.00E-4")
<<< a
0.0001
<<< type(a)
<class 'float'>
```

A seguir está um exemplo para converter número, float e string em tipo de dados float:

```
a = float(1)     # a will be 1.0
b = float(2.2)   # b will be 2.2
c = float("3.3") # c will be 3.3

print (a)
print (b)
print (c)
```

Isso produz o seguinte resultado -

```
1.0
2.2
3.3
```


Função Pythonstr()

Vimos como um Python obtém um número inteiro ou flutuante da representação de string correspondente. A função **str()** funciona de forma oposta. Ele envolve um

número inteiro ou um objeto float entre aspas (') para retornar um objeto str. A função **str()** retorna a representação em string de qualquer objeto Python. Nesta seção, veremos diferentes exemplos da função **str()** em Python.

A função str() possui três parâmetros. O primeiro parâmetro (ou argumento) obrigatório é o objeto cuja representação de string desejamos. Outros dois operadores, codificação e erros, são opcionais.

Executaremos a função str() no console Python para verificar facilmente se o objeto retornado é uma string, com aspas (').

Inteiro para string

Você pode converter qualquer número inteiro em uma string da seguinte maneira:

```
<<< a = str(10)
<<< a
'10'
<<< type(a)
<class 'str'>
```

Flutuar para String

A função str() converte objetos de ponto flutuante com notações de ponto flutuante, notação padrão com um ponto decimal separando a parte inteira e fracionária, e a notação científica em objeto string.

```
<<< a=str(11.10)
<<< a
'11.1'
<<< type(a)
<class 'str'>
<<< a = str(2/5)
<<< a
'0.4'
<<< type(a)
<class 'str'>
```

No segundo caso, uma expressão de divisão é dada como argumento para a função str(). Observe que a expressão é avaliada primeiro e depois o resultado é convertido em string.

Pontos flutuantes em notações científicas usando E ou e e com potência positiva ou negativa são convertidos em string com a função str().

```
<<< a=str(10E4)
<<< a
'100000.0'
<<< type(a)
<class 'str'>
<<< a=str(1.23e-4)
<<< a
'0.000123'
<<< type(a)
<class 'str'>
```

Quando a constante booleana é inserida como argumento, ela é cercada por (') para que True se torne 'True'. Objetos Lista e Tupla também podem receber argumentos para a função str(). A string resultante é a lista/tupla cercada por (').

```
<<< a=str('True')
<<< a
'True'
<<< a=str([1,2,3])
<<< a
'[1, 2, 3]'
<<< a=str((1,2,3))
<<< a
'(1, 2, 3)'
<<< a=str({1:100, 2:200, 3:300})
<<< a
'{1: 100, 2: 200, 3: 300}'
```

A seguir está um exemplo para converter número, float e string em tipo de dados string:

```
a = str(1)     # a will be "1"
b = str(2.2)   # b will be "2.2"
c = str("3.3") # c will be "3.3"

print (a)
print (b)
print (c)
```

Isso produz o seguinte resultado -

```
1
2.2
3.3
```


Conversão de tipos de sequência

List, Tuple e String são tipos de sequência do Python. Eles são uma coleção ordenada ou indexada de itens.

Uma string e uma tupla podem ser convertidas em um objeto de lista usando a função **list()** . Da mesma forma, a função **tuple()** converte uma string ou lista em uma tupla.

Tomaremos um objeto de cada um desses três tipos de sequência e estudaremos sua interconversão.

```
<<< a=[1,2,3,4,5]   # List Object
<<< b=(1,2,3,4,5)   # Tupple Object
<<< c="Hello"       # String Object

### list() separates each character in the string and builds the list
<<< obj=list(c)
<<< obj
['H', 'e', 'l', 'l', 'o']

### The parentheses of tuple are replaced by square brackets
<<< obj=list(b)
<<< obj
[1, 2, 3, 4, 5]

### tuple() separates each character from string and builds a tuple of charac
<<< obj=tuple(c)
<<< obj
('H', 'e', 'l', 'l', 'o')
```

```
### square brackets of list are replaced by parentheses.
<<< obj=tuple(a)
<<< obj
(1, 2, 3, 4, 5)

### str() function puts the list and tuple inside the quote symbols.
<<< obj=str(a)
<<< obj
'[1, 2, 3, 4, 5]'

<<< obj=str(b)
<<< obj
'(1, 2, 3, 4, 5)'
```

Assim, o recurso de conversão de tipo explícito do Python permite a conversão de um tipo de dados para outro com a ajuda de suas funções integradas.

Funções de conversão de tipo de dados

Existem várias funções integradas para realizar a conversão de um tipo de dados para outro. Estas funções retornam um novo objeto representando o valor convertido.

Sr. Não.	Descrição da função
1	**Função Python int()** Converte x em um número inteiro. base especifica a base se x for uma string.
2	**Função longa() do Python** Converte x em um número inteiro longo. base especifica a base se x for uma string.
3	**Função Python float()** Converte x em um número de ponto flutuante.
4	**Função complexa() do Python** Cria um número complexo.
5	**Função Pythonstr()** Converte o objeto x em uma representação de string.

6	**Função Python repr()** Converte o objeto x em uma string de expressão.
7	**Função de avaliação Python()** Avalia uma string e retorna um objeto.
8	**Função Tupla() do Python** Converte s em uma tupla.
9	**Função lista() do Python** Converte s em uma lista.
10	**Função Python set()** Converte s em um conjunto.
11	**Função Python dict()** Cria um dicionário. d deve ser uma sequência de tuplas (chave, valor).
12	**Função Python frozenset()** Converte s em um conjunto congelado.
13	**Função Python chr()** Converte um número inteiro em um caractere.
14	**Função Unichr() do Python** Converte um número inteiro em um caractere Unicode.
15	**Função Python ord()** Converte um único caractere em seu valor inteiro.
16	**Função hexadecimal() do Python** Converte um número inteiro em uma string hexadecimal.
17	**Função Python oct()** Converte um número inteiro em uma string octal.