Python - Números

Python possui suporte integrado para armazenar e processar dados numéricos (**Python Numbers**). Na maioria das vezes você trabalha com números em quase todas as aplicações Python . Obviamente, qualquer aplicação de computador lida com números. Este tutorial irá discutir sobre diferentes tipos de números Python e suas propriedades.

Python - Tipos de números

Existem três tipos de números integrados disponíveis em Python:

- inteiros **(int)**
- números de ponto flutuante **(float)**
- números **complexos**

Python também possui um tipo de dados booleano integrado chamado **bool** . Pode ser tratado como um subtipo do tipo **int** , pois seus dois valores possíveis **True** e **False** representam os inteiros 1 e 0 respectivamente.

Python - números inteiros

Em Python, qualquer número sem a possibilidade de armazenar uma parte fracionária é um número inteiro. (Observe que se a parte fracionária de um número for 0, isso não significa que seja um número inteiro. Por exemplo, um número 10.0 não é um número inteiro, é um número flutuante com 0 parte fracionária cujo valor numérico é 10.) Um inteiro pode ser zero, positivo ou um número inteiro negativo. Por exemplo, 1234, 0, -55 representam números inteiros em Python.

Existem três maneiras de formar um objeto inteiro. Com (a) representação literal, (b) qualquer expressão avaliada como um número inteiro e (c) usando a função **int()** .

Literal é uma notação usada para representar uma constante diretamente no código-fonte. Por exemplo -

```
>>> a =10
```

No entanto, observe a seguinte atribuição da variável inteira c .

```
a = 10
b = 20
c = a + b

print ("a:", a, "type:", type(a))
print ("c:", c, "type:", type(c))
```

Ele produzirá a seguinte **saída** -

```
a: 10 type: <class 'int'>
c: 30 type: <class 'int'>
```

Aqui, c é de fato uma variável inteira, mas a expressão a + b é avaliada primeiro e seu valor é atribuído indiretamente a c .

O terceiro método de formação de um objeto inteiro é com o valor de retorno da função int(). Ele converte um número de ponto flutuante ou uma string em um número inteiro.

```
>>> a=int(10.5)
>>> b=int("100")
```

Você pode representar um número inteiro como um número binário, octal ou hexadecimal. Porém, internamente o objeto é armazenado como um número inteiro.

Números Binários em Python

Um número que consiste apenas em dígitos binários (1 e 0) e prefixado com **0b** é um número binário. Se você atribuir um número binário a uma variável, ela ainda será uma variável int.

A representa um número inteiro em formato binário, armazena-o diretamente como um literal ou usa a função int(), na qual a base é definida como 2

```
a=0b101
print ("a:",a, "type:",type(a))

b=int("0b101011", 2)
print ("b:",b, "type:",type(b))
```

Ele produzirá a seguinte **saída** -

```
a: 5 type: <class 'int'>
b: 43 type: <class 'int'>
```

Também existe uma função **bin()** em Python. Ele retorna uma string binária equivalente a um número inteiro.

```
a=43
b=bin(a)
print ("Integer:",a, "Binary equivalent:",b)
```

Ele produzirá a seguinte **saída** -

```
Integer: 43 Binary equivalent: 0b101011
```

Números octais em Python

Um número octal é composto apenas por dígitos de 0 a 7. Para especificar que o número inteiro usa notação octal, ele precisa ser prefixado por **0o** (O minúsculo) ou **0O** (O maiúsculo). Uma representação literal do número octal é a seguinte -

```
a=0O107
print (a, type(a))
```

Ele produzirá a seguinte **saída** -

```
71 <class 'int'>
```

Observe que o objeto é armazenado internamente como um número inteiro. O equivalente decimal do número octal 107 é 71.

Como o sistema numérico octal tem 8 símbolos (0 a 7), sua base é 7. Portanto, ao usar a função int() para converter uma string octal em inteiro, você precisa definir o argumento base como 8.

```
a=int('20',8)
print (a, type(a))
```

Ele produzirá a seguinte **saída** -

```
16 <class 'int'>
```

O equivalente decimal do octal 30 é 16.

No código a seguir, dois objetos int são obtidos a partir de notações octais e sua adição é realizada.

```
a=0O56
print ("a:",a, "type:",type(a))

b=int("0O31",8)
print ("b:",b, "type:",type(b))

c=a+b
print ("addition:", c)
```

Ele produzirá a seguinte **saída** -

```
a: 46 type: <class 'int'>
b: 25 type: <class 'int'>
addition: 71
```

Para obter a string octal para um número inteiro, use a função **oct()** .

```
a=oct(71)
print (a, type(a))
```

Números hexadecimais em Python

Como o nome sugere, existem 16 símbolos no sistema numérico hexadecimal. Eles são de 0 a 9 e de A a F. Os primeiros 10 dígitos são iguais aos dígitos decimais. Os alfabetos A, B, C, D, E e F são equivalentes a 11, 12, 13, 14, 15 e 16 respectivamente. Podem ser usadas letras maiúsculas ou minúsculas para esses símbolos de letras.

Para a representação literal de um número inteiro em notação hexadecimal, prefixe-o com **0x** ou **0X** .

```
a=0XA2
print (a, type(a))
```

Ele produzirá a seguinte **saída** -

```
162 <class 'int'>
```

Para converter uma string hexadecimal em inteiro, defina a base como 16 na função **int()** .

```
a=int('0X1e', 16)
print (a, type(a))
```

Experimente o seguinte trecho de código. É necessária uma string hexadecimal e retorna o número inteiro.

```
num_string = "A1"
number = int(num_string, 16)
print ("Hexadecimal:", num_string, "Integer:",number)
```

Ele produzirá a seguinte **saída** -

```
Hexadecimal: A1 Integer: 161
```

No entanto, se a string contiver qualquer símbolo além do gráfico de símbolos hexadecimais (por exemplo X001), ocorrerá o seguinte erro -

```
Traceback (most recent call last):
  File "C:\Python311\var1.py", line 4, in <module>
    number = int(num_string, 16)
ValueError: invalid literal for int() with base 16: 'X001'
```

A biblioteca padrão do Python possui a função **hex()** , com a qual você pode obter um equivalente hexadecimal de um número inteiro.

```
a=hex(161)
print (a, type(a))
```

Ele produzirá a seguinte **saída** -

```
0xa1 <class 'str'>
```

Embora um número inteiro possa ser representado como binário, octal ou hexadecimal, internamente ele ainda é um número inteiro. Portanto, ao realizar

operações aritméticas, a representação não importa.

```
a=10 #decimal
b=0b10 #binary
c=0O10 #octal
d=0XA #Hexadecimal
e=a+b+c+d

print ("addition:", e)
```

Ele produzirá a seguinte **saída** -

```
addition: 30
```

Python - números de ponto flutuante

Um número de ponto flutuante possui uma parte inteira e uma parte fracionária, separadas por um símbolo de ponto decimal (.). Por padrão, o número é positivo, prefixe um símbolo de traço (-) para um número negativo.

Um número de ponto flutuante é um objeto da classe float do Python. Para armazenar um objeto float, você pode usar uma notação literal, usar o valor de uma expressão aritmética ou usar o valor de retorno da função float().

Usar literal é a maneira mais direta. Basta atribuir um número com parte fracionária a uma variável. Cada uma das instruções a seguir declara um objeto float.

```
>>> a=9.99
>>> b=0.999
>>> c=-9.99
>>> d=-0.999
```

Em Python, não há restrição sobre quantos dígitos após a vírgula decimal um número de ponto flutuante pode ter. Porém, para encurtar a representação, é utilizado o símbolo **E** ou **e .** E significa Dez elevado a. Por exemplo, E4 é 10 elevado a 4 (ou a 4ª potência de 10), e-3 é 10 elevado a -3.

Na notação científica, o número tem um coeficiente e uma parte expoente. O coeficiente deve ser um float maior ou igual a 1, mas menor que 10. Portanto, 1,23E+3, 9,9E-5 e 1E10 são exemplos de floats com notação científica.

```
>>> a=1E10
>>> a
10000000000.0
>>> b=9.90E-5
>>> b
9.9e-05
>>> 1.23E3
1230.0
```

A segunda abordagem para formar um objeto float é indireta, usando o resultado de uma expressão. Aqui, o quociente de dois floats é atribuído a uma variável, que se refere a um objeto float.

```
a=10.33
b=2.66
c=a/b

print ("c:", c, "type", type(c))
```

Ele produzirá a seguinte **saída** -

```
c: 3.8834586466165413 type <class 'float'>
```

A função float() do Python retorna um objeto float, analisando um número ou uma string se tiver o conteúdo apropriado. Se nenhum argumento for fornecido entre parênteses, ele retornará 0,0 e, para um argumento **int** , a parte fracionária com 0 será adicionada.

```
>>> a=float()
>>> a
0.0
>>> a=float(10)
>>> a
10.0
```

Mesmo que o número inteiro seja expresso em binário, octal ou hexadecimal, a função float() retorna um float com parte fracionária como 0.

```
a=float(0b10)
b=float(0O10)
c=float(0xA)
```

```
print (a,b,c, sep=",")
```

Ele produzirá a seguinte **saída** -

```
2.0,8.0,10.0
```

A função **float()** recupera um número de ponto flutuante de uma string que envolve um ponto flutuante, seja no formato de ponto decimal padrão ou com notação científica.

```
a=float("-123.54")
b=float("1.23E04")
print ("a=",a,"b=",b)
```

Ele produzirá a seguinte **saída** -

```
a= -123.54 b= 12300.0
```

Em matemática, o infinito é um conceito abstrato. Fisicamente, um número infinitamente grande nunca pode ser armazenado em qualquer quantidade de memória. Para a maioria das configurações de hardware de computador, entretanto, um número muito grande com 400ª potência de 10 é representado por Inf. Se você usar "Infinity" como argumento para a função float(), ele retornará Inf.

```
a=1.00E400
print (a, type(a))
a=float("Infinity")
print (a, type(a))
```

Ele produzirá a seguinte **saída** -

```
inf <class 'float'>
inf <class 'float'>
```

Mais uma dessas entidades é Nan (significa Not a Number). Representa qualquer valor indefinido ou não representável.

```
>>> a=float('Nan')
>>> a
```

```
Nan
```

Python - Números Complexos

Nesta seção, conheceremos em detalhes sobre o tipo de dados Complexo em Python. Os números complexos encontram suas aplicações em equações matemáticas e leis em eletromagnetismo, eletrônica, óptica e teoria quântica. As transformadas de Fourier usam números complexos. Eles são usados em cálculos com funções de onda, projeto de filtros, integridade de sinal em eletrônica digital, radioastronomia, etc.

Um número complexo consiste em uma parte real e uma parte imaginária, separadas por "+" ou "-". A parte real pode ser qualquer número de ponto flutuante (ou ele próprio um número complexo). A parte imaginária também é flutuante/complexa, mas multiplicada por um número imaginário.

Em matemática, um número imaginário "i" é definido como a raiz quadrada de -1 ($\sqrt{-1}$). Portanto, um número complexo é representado como “x+yi”, onde x é a parte real e “y” é o coeficiente da parte imaginária.

Muitas vezes, o símbolo “j” é usado em vez de “I” para o número imaginário, para evitar confusão com seu uso como corrente na teoria da eletricidade. Python também usa “j” como número imaginário. Portanto, “x+yj” é a representação de um número complexo em Python.

Assim como os tipos de dados int ou float, um objeto complexo pode ser formado com representação literal ou usando a função complex(). Todas as declarações a seguir formam um objeto complexo.

```
>>> a=5+6j
>>> a
(5+6j)
>>> type(a)
<class 'complex'>
>>> a=2.25-1.2J
>>> a
(2.25-1.2j)
>>> type(a)
<class 'complex'>
>>> a=1.01E-2+2.2e3j
>>> a
(0.0101+2200j)
```

```
>>> type(a)
<class 'complex'>
```

Observe que a parte real, bem como o coeficiente da parte imaginária, devem ser flutuantes e podem ser expressos em notação de ponto decimal padrão ou em notação científica.

A função complex() do Python ajuda a formar um objeto de tipo complexo. A função recebe argumentos para parte real e imaginária e retorna o número complexo.

Existem duas versões da função complex(), com dois argumentos e com um argumento. O uso de complex() com dois argumentos é simples. Utiliza o primeiro argumento como parte real e o segundo como coeficiente da parte imaginária.

```
a=complex(5.3,6)
b=complex(1.01E-2, 2.2E3)
print ("a:", a, "type:", type(a))
print ("b:", b, "type:", type(b))
```

Ele produzirá a seguinte **saída** -

```
a: (5.3+6j) type: <class 'complex'>
b: (0.0101+2200j) type: <class 'complex'>
```

No exemplo acima, usamos xey como parâmetros flutuantes. Eles podem até ser de tipos de dados complexos.

```
a=complex(1+2j, 2-3j)
print (a, type(a))
```

Ele produzirá a seguinte **saída** -

```
(4+4j) <class 'complex'>
```

Surpreso com o exemplo acima? Coloque “x” como 1+2j e “y” como 2-3j. Tente realizar o cálculo manual de "x+yj" e você saberá.

```
complex(1+2j, 2-3j)
=(1+2j)+(2-3j)*j
=1+2j +2j+3
=4+4j
```

Se você usar apenas um argumento numérico para a função complex(), ele o tratará como o valor da parte real; e a parte imaginária é definida como 0.

```
a=complex(5.3)
print ("a:", a, "type:", type(a))
```

Ele produzirá a seguinte **saída** -

```
a: (5.3+0j) type: <class 'complex'>
```

A função complex() também pode analisar uma string em um número complexo se seu único argumento for uma string com representação de número complexo.

No trecho a seguir, o usuário é solicitado a inserir um número complexo. É usado como argumento. Como o Python lê a entrada como uma string, a função extrai dela o objeto complexo.

```
a= "5.5+2.3j"
b=complex(a)
print ("Complex number:", b)
```

Ele produzirá a seguinte **saída** -

```
Complex number: (5.5+2.3j)
```

A classe complexa integrada do Python tem dois atributos **real** e **imag** - eles retornam o real e o coeficiente da parte imaginária do objeto.

```
a=5+6j
print ("Real part:", a.real, "Coefficient of Imaginary part:", a.imag)
```

Ele produzirá a seguinte **saída** -

```
Real part: 5.0 Coefficient of Imaginary part: 6.0
```

A classe complexa também define um método conjugate(). Ele retorna outro número complexo com o sinal da componente imaginária invertido. Por exemplo, o conjugado de x+yj é x-yj.

```
>>> a=5-2.2j
>>> a.conjugate()
```

```
(5+2.2j)
```