# Python - Tipos de dados

## Visão geral

O computador é um dispositivo de processamento de dados. O computador armazena os dados em sua memória e os processa de acordo com o programa fornecido. Dados são uma representação de fatos sobre um determinado objeto.

Alguns exemplos de dados -

- **Dados dos alunos** - nome, sexo, turma, notas, idade, mensalidade etc.
- **Dados dos livros da biblioteca** – título, autor, editora, preço, páginas, ano de publicação etc.
- **Dados dos funcionários de um escritório** – nome, designação, salário, departamento, filial, etc.

## O que é um tipo de dados?

Um **tipo de dados** representa um tipo de valor e determina quais operações podem ser realizadas nele. Dados numéricos, não numéricos e booleanos (verdadeiro/falso) são os tipos de dados mais óbvios. No entanto, cada linguagem de programação tem a sua própria classificação, refletindo em grande parte a sua filosofia de programação.

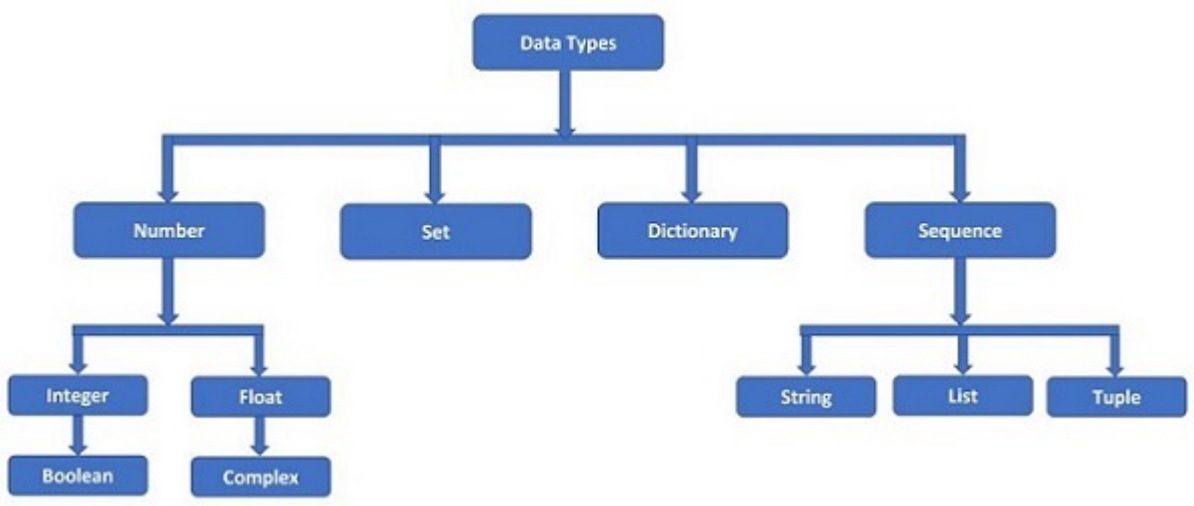
## Tipos de dados em Python

Os tipos de dados Python são usados para definir o tipo de uma variável. Define que tipo de dados vamos armazenar em uma variável . Os dados armazenados na memória podem ser de vários tipos. Por exemplo, a idade de uma pessoa é armazenada como um valor numérico e seu endereço é armazenado como caracteres alfanuméricos.

## Tipos de dados Python

Python possui os seguintes tipos de dados integrados que discutiremos neste tutorial:

| Tipo de dados | Exemplos |
|---|---|
| Numérico | int, float, complexo |
| Corda | str |
| Seqüência | lista, tupla, intervalo |
| Binário | bytes, bytearray, visualização de memória |
| Mapeamento | ditar |
| boleano | bool |
| Definir | conjunto, conjunto congelado |
| Nenhum | NenhumTipo |




## Tipo de dados numéricos Python

Os tipos de dados numéricos do Python armazenam valores numéricos. Objetos numéricos são criados quando você atribui um valor a eles. Por exemplo -

```
var1 = 1        # int data type
var2 = True     # bool data type
var3 = 10.023   # float data type
var4 = 10+3j    # complex data type
```

Python suporta quatro tipos numéricos diferentes e cada um deles possui classes integradas na biblioteca Python, chamadas **int, bool, float** e **complex** respectivamente -

- int (números inteiros com sinal)
- bool (subtipo de números inteiros.)
- float (valores reais de ponto flutuante)
- complexo (números complexos)

A biblioteca padrão do Python possui uma função integrada **type()** , que retorna a classe de um determinado objeto. Aqui, é usado para verificar o tipo de um número inteiro e de ponto flutuante.

```
>>> type(123)
<class 'int'>
>>> type(9.99)
<class 'float'>
```

Um número complexo é composto de duas partes – **real** e **imaginária** . Eles são separados por sinais '+' ou '-'. A parte imaginária tem o sufixo 'j', que é o número imaginário. A raiz quadrada de -1 ($\sqrt{-1}$), é definido como número imaginário. O número complexo em Python é representado como x+yj, onde x é a parte real e y é a parte imaginária. Portanto, 5+6j é um número complexo.

```
>>> type(5+6j)
<class 'complex'>
```

Um número booleano possui apenas dois valores possíveis, representados pelas palavras-chave **True** e **False** . Eles correspondem aos inteiros 1 e 0, respectivamente.

```
>>> type (True)
<class 'bool'>
>>> type(False)
<class 'bool'>
```

Aqui estão alguns exemplos de números -

| interno | bool | flutuador | complexo |
|---|---|---|---|
| 10 | Verdadeiro | 0,0 | 3.14j |
| 0O777 | Falso | 15h20 | 45.j |
| -786 | | -21,9 | 9.322e-36j |

| | | | |
|---|---|---|---|
| 080 | | 32,3+e18 | .876j |
| 0x17 | | -90. | -.6545+0J |
| -0x260 | | -32.54e100 | 3e+26J |
| 0x69 | | 70.2-E12 | 4.53e-7j |

## Exemplo de tipos de dados numéricos

A seguir está um exemplo para mostrar o uso de números inteiros, flutuantes e complexos:

```
# integer variable.
a=100
print("The type of variable having value", a, " is ", type(a))

# boolean variable.
b=True
print("The type of variable having value", b, " is ", type(b))

# float variable.
c=20.345
print("The type of variable having value", c, " is ", type(c))

# complex variable.
d=10+3j
print("The type of variable having value", d, " is ", type(d))
```

## Tipo de dados de sequência Python

Sequência é um tipo de dados de coleção. É uma coleção ordenada de itens. Os itens na sequência têm um índice posicional começando com 0. É conceitualmente semelhante a um array em C ou C++. Existem três tipos de dados de sequência definidos em Python.

- Tipo de dados de string
- Listar tipo de dados
- Tipo de dados tupla

> *As sequências Python são limitadas e iteráveis - sempre que dizemos iterável em Python, significa um tipo de dados de sequência (por exemplo, uma lista).*

## Tipo de dados de string Python

String Python é uma sequência de um ou mais caracteres Unicode, colocados entre aspas simples, duplas ou triplas (também chamadas de vírgulas invertidas). Strings Python são imutáveis, o que significa que quando você executa uma operação em strings, você sempre produz um novo objeto string do mesmo tipo, em vez de alterar uma string existente.

Contanto que a mesma sequência de caracteres esteja entre aspas simples, duplas ou triplas, não importam. Portanto, as seguintes representações de string são equivalentes.

```
>>> 'TutorialsPoint'
'TutorialsPoint'
>>> "TutorialsPoint"
'TutorialsPoint'
>>> '''TutorialsPoint'''
'TutorialsPoint'
```

Uma string em Python é um objeto da classe **str** . Isso pode ser verificado com a função **type()** .

```
>>> type("Welcome To TutorialsPoint")
<class 'str'>
```

Uma string é um tipo de dados não numérico. Obviamente, não podemos realizar operações aritméticas nele. No entanto, operações como **fatiamento** e **concatenação** podem ser realizadas. A classe str do Python define vários métodos úteis para processamento de strings. Subconjuntos de strings podem ser obtidos usando o operador de fatia ([ ] e [:] ) com índices começando em 0 no início da string e trabalhando a partir de -1 no final.

O sinal de mais (+) é o operador de concatenação de strings e o asterisco (*) é o operador de repetição em Python.

## Exemplo de tipo de dados String

```
str = 'Hello World!'

print (str)          # Prints complete string
print (str[0])       # Prints first character of the string
print (str[2:5])     # Prints characters starting from 3rd to 5th
print (str[2:])      # Prints string starting from 3rd character
print (str * 2)      # Prints string two times
print (str + "TEST") # Prints concatenated string
```

Isso produzirá o seguinte resultado -

```
Hello World!
H
llo
llo World!
Hello World!Hello World!
Hello World!TEST
```

## Tipo de dados de lista Python

Listas Python são os tipos de dados compostos mais versáteis. Uma lista Python contém itens separados por vírgulas e entre colchetes ([]). Até certo ponto, as listas Python são semelhantes aos arrays em C. Uma diferença entre eles é que todos os itens pertencentes a uma lista Python podem ser de tipos de dados diferentes, enquanto um array C pode armazenar elementos relacionados a um tipo de dados específico.

```
>>> [2023, "Python", 3.11, 5+6j, 1.23E-4]
```

Uma lista em Python é um objeto da classe **list** . Podemos verificar isso com a função type().

```
>>> type([2023, "Python", 3.11, 5+6j, 1.23E-4])
<class 'list'>
```

Conforme mencionado, um item da lista pode ser de qualquer tipo de dados. Isso significa que um objeto de lista também pode ser um item de outra lista. Nesse caso, torna-se uma lista aninhada.

```
>>> [['One', 'Two', 'Three'], [1,2,3], [1.0, 2.0, 3.0]]
```

> *Uma lista pode conter itens que são números simples, strings, tupla, dicionário, conjunto ou objeto de classe definida pelo usuário também.*

Os valores armazenados em uma lista Python podem ser acessados usando o operador slice ([ ] e [:]) com índices começando em 0 no início da lista e indo até o final -1. O sinal de mais (+) é o operador de concatenação de lista e o asterisco (*) é o operador de repetição. Por exemplo -

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']

print (list)            # Prints complete list
print (list[0])         # Prints first element of the list
print (list[1:3])       # Prints elements starting from 2nd till 3rd
print (list[2:])        # Prints elements starting from 3rd element
print (tinylist * 2)    # Prints list two times
print (list + tinylist) # Prints concatenated lists
```

Isso produz o seguinte resultado -

```
['abcd', 786, 2.23, 'john', 70.2]
abcd
[786, 2.23]
[2.23, 'john', 70.2]
[123, 'john', 123, 'john']
['abcd', 786, 2.23, 'john', 70.2, 123, 'john']
```

## Tipo de dados Tupla Python

Tupla Python é outro tipo de dados de sequência semelhante a uma lista. Uma tupla Python consiste em vários valores separados por vírgulas. Ao contrário das listas, entretanto, as tuplas são colocadas entre parênteses (...).

Uma tupla também é uma sequência, portanto cada item da tupla possui um índice referente à sua posição na coleção. O índice começa em 0.

```
>>> (2023, "Python", 3.11, 5+6j, 1.23E-4)
```

Em Python, uma tupla é um objeto da classe **tupla** . Podemos verificar isso com a função type().

```
>>> type((2023, "Python", 3.11, 5+6j, 1.23E-4))
<class 'tuple'>
```

Como no caso de uma lista, um item na tupla também pode ser uma lista, a própria tupla ou um objeto de qualquer outra classe Python.

```
>>> (['One', 'Two', 'Three'], 1,2.0,3, (1.0, 2.0, 3.0))
```

Para formar uma tupla, o uso de parênteses é opcional. Os itens de dados separados por vírgula sem nenhum símbolo delimitador são tratados como uma tupla por padrão.

```
>>> 2023, "Python", 3.11, 5+6j, 1.23E-4
(2023, 'Python', 3.11, (5+6j), 0.000123)
```

As principais diferenças entre listas e tuplas são: As listas são colocadas entre colchetes ( [ ] ) e seus elementos e tamanho podem ser alterados. Ou seja, as listas são mutáveis, enquanto as tuplas são colocadas entre parênteses ( ( ) ) e não podem ser atualizadas (imutáveis). As tuplas podem ser consideradas listas **somente leitura** . Por exemplo -

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  )
tinytuple = (123, 'john')

print (tuple)               # Prints the complete tuple
print (tuple[0])            # Prints first element of the tuple
print (tuple[1:3])          # Prints elements of the tuple starting from 2nd
print (tuple[2:])           # Prints elements of the tuple starting from 3rd
print (tinytuple * 2)       # Prints the contents of the tuple twice
print (tuple + tinytuple)   # Prints concatenated tuples
```

Isso produz o seguinte resultado -

```
('abcd', 786, 2.23, 'john', 70.2)
abcd
(786, 2.23)
(2.23, 'john', 70.2)
(123, 'john', 123, 'john')
('abcd', 786, 2.23, 'john', 70.2, 123, 'john')
```

O código a seguir é inválido com tupla porque tentamos atualizar uma tupla, o que não é permitido. Caso semelhante é possível com listas -

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  )
list = [ 'abcd', 786 , 2.23, 'john', 70.2  ]
tuple[2] = 1000    # Invalid syntax with tuple
list[2] = 1000     # Valid syntax with list
```

## Tipo de dados do dicionário Python

Os dicionários Python são do tipo tabela hash. Uma chave de dicionário pode ser quase qualquer tipo de Python, mas geralmente são números ou strings. Os valores, por outro lado, podem ser qualquer objeto Python arbitrário.

O dicionário Python é como arrays associativos ou hashes encontrados em Perl e consiste em pares **chave:valor .** Os pares são separados por vírgula e colocados entre chaves {}. Para estabelecer o mapeamento entre chave e valor, o símbolo de ponto e vírgula':' é colocado entre os dois.

```
>>> {1:'one', 2:'two', 3:'three'}
```

**Em Python, dicionário é um objeto da classe dict** integrada . Podemos verificar isso com a função type().

```
>>> type({1:'one', 2:'two', 3:'three'})
<class 'dict'>
```

Os dicionários são colocados entre chaves ({ }) e os valores podem ser atribuídos e acessados usando colchetes ([]). Por exemplo -

```
dict = {}
dict['one'] = "This is one"
dict[2]     = "This is two"

tinydict = {'name': 'john','code':6734, 'dept': 'sales'}


print (dict['one'])       # Prints value for 'one' key
print (dict[2])           # Prints value for 2 key
print (tinydict)          # Prints complete dictionary
print (tinydict.keys())   # Prints all the keys
print (tinydict.values()) # Prints all the values
```

Isso produz o seguinte resultado -

```
This is one
This is two
{'dept': 'sales', 'code': 6734, 'name': 'john'}
['dept', 'code', 'name']
['sales', 6734, 'john']
```

O dicionário do Python não é uma sequência. É uma coleção de itens, mas cada item (par chave:valor) não é identificado pelo índice posicional como em string, lista ou tupla. Conseqüentemente, a operação de fatiamento não pode ser realizada em um dicionário. O dicionário é um objeto mutável, portanto é possível realizar ações de adição, modificação ou exclusão com a funcionalidade correspondente definida na classe dict. Essas operações serão explicadas em um capítulo subsequente.

## Tipo de dados do conjunto Python

Set é uma implementação Python de set conforme definido em matemática. Um conjunto em Python é uma coleção, mas não é uma coleção indexada ou ordenada como string, lista ou tupla. Um objeto não pode aparecer mais de uma vez em um conjunto, enquanto em Lista e Tupla o mesmo objeto pode aparecer mais de uma vez.

Itens separados por vírgula em um conjunto são colocados entre chaves ou colchetes {}. Os itens na coleção definida podem ser de diferentes tipos de dados.

```
>>> {2023, "Python", 3.11, 5+6j, 1.23E-4}
{(5+6j), 3.11, 0.000123, 'Python', 2023}
```

Observe que os itens da coleção definida podem não seguir a mesma ordem em que foram inseridos. A posição dos itens é otimizada pelo Python para realizar operações sobre conjuntos conforme definido em matemática.

**O Set do Python é um objeto da classe set** integrada , como pode ser verificado com a função type().

```
>>> type({2023, "Python", 3.11, 5+6j, 1.23E-4})
<class 'set'>
```

Um conjunto pode armazenar apenas objetos **imutáveis** , como número (int, float, complexo ou bool), string ou tupla. Se você tentar colocar uma lista ou dicionário na coleção set, o Python gerará um **TypeError** .

```
>>> {['One', 'Two', 'Three'], 1,2,3, (1.0, 2.0, 3.0)}
Traceback (most recent call last):
   File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

**Hashing** é um mecanismo da ciência da computação que permite uma busca mais rápida de objetos na memória do computador. **Somente objetos imutáveis são hasháveis** .

Mesmo que um conjunto não permita itens mutáveis, o próprio conjunto é mutável. Conseqüentemente, operações de adição/exclusão/atualização são permitidas em um objeto de conjunto, usando os métodos da classe de conjunto integrada. Python também possui um conjunto de operadores para realizar a manipulação de conjuntos. Os métodos e operadores são explicados nos últimos capítulos

## Tipos de dados booleanos Python

**O tipo booleano** Python é um dos tipos de dados integrados que representa um dos dois valores **True** ou **False** . **A função bool()** do Python permite avaliar o valor de qualquer expressão e retornar True ou False com base na expressão.

## Exemplo de tipos de dados booleanos

A seguir está um programa que imprime o valor das variáveis booleanas aeb -

```
a = True
# display the value of a
print(a)

# display the data type of a
print(type(a))
```

Isso produz o seguinte resultado -

```
true
<class 'bool'>
```

A seguir está outro programa que avalia as expressões e imprime os valores de retorno:

```
# Returns false as a is not equal to b
a = 2
```

```
b = 4
print(bool(a==b))

# Following also prints the same
print(a==b)

# Returns False as a is None
a = None
print(bool(a))

# Returns false as a is an empty sequence
a = ()
print(bool(a))

# Returns false as a is 0
a = 0.0
print(bool(a))

# Returns false as a is 10
a = 10
print(bool(a))
```

Isso produz o seguinte resultado -

```
False
False
False
False
False
True
```



## Obtendo tipo de dados

Em Python, para obter o tipo de dados de uma variável, use o método **type()** .

## Exemplo para obter tipo de dados

Este exemplo demonstra o uso do método type(), ou seja, para obter o tipo de dados das variáveis.

```
a = 10
b = 10.23
c = "TutorialsPoint"
d = True

# Printing types
print(type(a))  # Returns <class 'int'>
print(type(b))  # Returns <class 'float'>
print(type(c))  # Returns <class 'str'>
print(type(d))  # Returns <class 'bool'>
```