NumPy - Tipos de dados

NumPy suporta uma variedade muito maior de tipos numéricos do que Python. A tabela a seguir mostra diferentes tipos de dados escalares definidos em NumPy.

Sr. Não.	Tipos e descrição de dados
1	**bool_** Booleano (True ou False) armazenado como um byte
2	**int_** Tipo inteiro padrão (igual a C long; normalmente int64 ou int32)
3	**interno** Idêntico a C int (normalmente int32 ou int64)
4	**interno** Inteiro usado para indexação (igual a C ssize_t; normalmente int32 ou int64)
5	**int8** Bytes (-128 a 127)
6	**int16** Inteiro (-32768 a 32767)
7	**int32** Inteiro (-2147483648 a 2147483647)
8	**int64** Inteiro (-9223372036854775808 a 9223372036854775807)
9	**uint8** Inteiro sem sinal (0 a 255)
10	**uint16** Inteiro sem sinal (0 a 65535)
11	**uint32** Inteiro sem sinal (0 a 4294967295)
12	**uint64** Inteiro sem sinal (0 a 18446744073709551615)
13	**flutuador_**

	Abreviação de float64
14	**float16** Flutuação de meia precisão: bit de sinal, expoente de 5 bits, mantissa de 10 bits
15	**float32** Flutuação de precisão única: bit de sinal, expoente de 8 bits, mantissa de 23 bits
16	**float64** Flutuação de precisão dupla: bit de sinal, expoente de 11 bits, mantissa de 52 bits
17	**complexo_** Abreviação de complexo128
18	**complexo64** Número complexo, representado por dois floats de 32 bits (componentes reais e imaginários)
19	**complexo128** Número complexo, representado por dois floats de 64 bits (componentes reais e imaginários)

Os tipos numéricos NumPy são instâncias de objetos dtype (tipo de dados), cada um com características únicas. Os dtypes estão disponíveis como np.bool_, np.float32, etc.

Objetos de tipo de dados (dtype)

Um objeto de tipo de dados descreve a interpretação de um bloco fixo de memória correspondente a um array, dependendo dos seguintes aspectos -

- Tipo de dados (inteiro, flutuante ou objeto Python)
- Tamanho dos dados
- Ordem de bytes (little-endian ou big-endian)
- No caso do tipo estruturado, os nomes dos campos, tipo de dados de cada campo e parte do bloco de memória ocupado por cada campo.
- Se o tipo de dados for uma submatriz, sua forma e tipo de dados

A ordem dos bytes é decidida prefixando '<' ou '>' ao tipo de dados. '<' significa que a codificação é little-endian (o menos significativo é armazenado no menor endereço). '>' significa que a codificação é big endian (o byte mais significativo é armazenado no menor endereço).

Um objeto dtype é construído usando a seguinte sintaxe -

```
numpy.dtype(object, align, copy)
```

Os parâmetros são -

- **Object** - Para ser convertido em tipo de dados objeto
- **Align** - Se verdadeiro, adiciona preenchimento ao campo para torná-lo semelhante ao C-struct
- **Copy** - Faz uma nova cópia do objeto dtype. Se for falso, o resultado será uma referência ao objeto de tipo de dados integrado

Exemplo 1

```
# using array-scalar type
import numpy as np
dt = np.dtype(np.int32)
print dt
```

Demonstração ao vivo

A saída é a seguinte -

```
int32
```

Exemplo 2

```
#int8, int16, int32, int64 can be replaced by equivalent strir
import numpy as np

dt = np.dtype('i4')
print dt
```

Demonstração ao vivo

A saída é a seguinte -

```
int32
```

Exemplo 3

Demonstração ao vivo

```
# using endian notation
import numpy as np
dt = np.dtype('>i4')
print dt
```

A saída é a seguinte -

```
>i4
```

Os exemplos a seguir mostram o uso do tipo de dados estruturados. Aqui, o nome do campo e o tipo de dados escalar correspondente devem ser declarados.

Exemplo 4

Demonstração ao vivo

```
# first create structured data type
import numpy as np
dt = np.dtype([('age',np.int8)])
print dt
```

A saída é a seguinte -

```
[('age', 'i1')]
```

Exemplo 5

Demonstração ao vivo

```
# now apply it to ndarray object
import numpy as np

dt = np.dtype([('age',np.int8)])
a = np.array([(10,),(20,),(30,)], dtype = dt)
print a
```

A saída é a seguinte -

```
[(10,) (20,) (30,)]
```

Exemplo 6

```
# file name can be used to access content of age column
import numpy as np

dt = np.dtype([('age',np.int8)])
a = np.array([(10,),(20,),(30,)], dtype = dt)
print a['age']
```

Demonstração ao vivo

A saída é a seguinte -

```
[10 20 30]
```

Exemplo 7

Os exemplos a seguir definem um tipo de dados estruturado chamado **estudante** com um campo de string 'nome', um **campo inteiro** 'idade' e um **campo flutuante** 'marcas'. Este dtype é aplicado ao objeto ndarray.

```
import numpy as np
student = np.dtype([('name','S20'), ('age', 'i1'), ('marks', 'f4')])
print student
```

Demonstração ao vivo

A saída é a seguinte -

```
[('name', 'S20'), ('age', 'i1'), ('marks', '<f4')])
```

Exemplo 8

```
import numpy as np

student = np.dtype([('name','S20'), ('age', 'i1'), ('marks', 'f4')])
a = np.array([('abc', 21, 50),('xyz', 18, 75)], dtype = student)
print a
```

Demonstração ao vivo

A saída é a seguinte -

```
[('abc', 21, 50.0), ('xyz', 18, 75.0)]
```

Cada tipo de dados integrado possui um código de caracteres que o identifica exclusivamente.

- **'b'** - booleano
- **'i'** - inteiro (assinado)
- **'u'** - inteiro sem sinal
- **'f'** - ponto flutuante
- **'c'** - ponto flutuante complexo
- **'m'** – timedelta
- **'M'** - data e hora
- **'O'** - objetos (Python)
- **'S', 'a'** - (byte-)string
- **'U'** - Unicode
- **'V'** - dados brutos (vazio)