Data Types in Python

strings: used to represent text data under quote marks like "ABCD".

integer: used to represent integer numbers like 1, 2, 3, -1, -2, -3.

float: used to represent real numbers like 1.2, 42.42.

boolean: used to represent True or False.

complex: used to represent complex numbers like 1.0 + 2.0j, 1.5 + 2.5j .

Data Types in NumPy

NumPy has some extra data types and refers them with one character, like i for integers, u for unsigned integers etc. List of all data types in NumPy and the characters used to represent them is as follows:

Checking the Data Type of an Array

The NumPy array object has a property called dtype that returns the data type of the array.

Data Types in NumPy NumPy has some extra data types, and refer to data types with one character, like i for integers, u for unsigned integers etc.

Below is a list of all data types in NumPy and the characters used to represent them.

i - integer

b - boolean

u - unsigned integer

f - float

c - complex float

m - timedelta

M - datetime

O - object

S - string

U - unicode string

V - fixed chunk of memory for other type (void)

Program 1: Get the data type of an array object.

In [1]:

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print("The data type of arr is:", arr.dtype)
```

```
The data type of arr is: int32
```

Program 2: Get the data type of an array containing strings.

In [2]:

```
import numpy as np
arr = np.array(['apple', 'banana', 'cherry'])
print("The data type of arr is:", arr.dtype)
```

```
The data type of arr is: <U6
```

Program 3: Create an array with data type string.

In [3]:

```
import numpy as np
arr = np.array([1, 2, 3, 4], dtype='S')
print("The value of arr is", arr)
print("The data type of arr is:", arr.dtype)
```

```
The value of arr is [b'1' b'2' b'3' b'4']
The data type of arr is: |S1
```

What if a Value Can Not Be Converted?

If a type is given in which elements can't be casted then NumPy will raise a ValueError. In Python, ValueError is raised when the type of passed argument to a function is unexpected/incorrect.

Program 5: A non integer string like 'a' can not be converted to integer (will raise an error).

In [4]:

```
import numpy as np
arr = np.array(['a', '2', '3'], dtype='i')
print("The value of arr is:", arr)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_17364/2004014751.py in <module>
      1 import numpy as np
----> 2 arr = np.array(['a', '2', '3'], dtype='i')
      3 print("The value of arr is:", arr)

ValueError: invalid literal for int() with base 10: 'a'
```

Converting Data Type on Existing Arrays

The best way to change the data type of an existing array, is to make a copy of the array with the astype() method. The astype() function creates a copy of the array, and allows us to specify the data type as a parameter. The data type can be specified using a string, like 'f' for float, 'i' for integer etc. or we can use the data type directly like float for float and int for integer.

Change data type from float to integer by using 'i' as parameter value.

In [5]:

```
import numpy as np
arr = np.array([1.1, 2.1, 3.1])
newarr = arr.astype('i')
print("The value of newarr is:", newarr)
print("The data type of newarr is:", newarr.dtype)
```

```
The value of newarr is: [1 2 3]
The data type of newarr is: int32
```

In [6]:

```
import numpy as np
arr = np.array([1.1, 2.1, 3.1])
newarr = arr.astype(int)
print("The value of newarr is:", newarr)
print("The data type of newarr is:", newarr.dtype)
```

```
The value of newarr is: [1 2 3]
The data type of newarr is: int32
```

In [7]:

```
#Change data type from integer to boolean.

import numpy as np
arr = np.array([1, 0, 3])
newarr = arr.astype(bool)
print("The value of newarr is:", newarr)
print("The data type of newarr is:", newarr.dtype)
```

```
The value of newarr is: [ True False  True]
The data type of newarr is: bool
```

class notes

In [8]:

```
a1 = np.arange(12)
print(a1)
print(a1.reshape((3, 4)))
print("C -", a1.reshape(3, 4, order='C'), sep='\n')
print("F -", a1.reshape(3, 4, order='F'), sep='\n')
print("A -", a1.reshape(3, 4, order='A'), sep='\n')
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11]
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
C -
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
F -
[[ 0  3  6  9]
 [ 1  4  7 10]
 [ 2  5  8 11]]
A -
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

In [9]:

```
a1 = np.arange(1, 13).reshape(3, -1)
a2 = np.arange(13, 25).reshape(3, -1)
a30 = np.stack((a1, a2))
print("a30", a30)
a31 = np.stack((a1, a2), axis=1)
print("a31", a31)
a32 = np.stack((a1, a2), axis=2)
print("a32", a32)
```

```
a30 [[[ 1  2  3  4]
  [ 5  6  7  8]
  [ 9 10 11 12]]

 [[13 14 15 16]
  [17 18 19 20]
  [21 22 23 24]]]
a31 [[[ 1  2  3  4]
  [13 14 15 16]]

 [[ 5  6  7  8]
  [17 18 19 20]]

 [[ 9 10 11 12]
  [21 22 23 24]]]
a32 [[[ 1 13]
  [ 2 14]
  [ 3 15]
  [ 4 16]]

 [[ 5 17]
  [ 6 18]
  [ 7 19]
  [ 8 20]]

 [[ 9 21]
  [10 22]
  [11 23]
  [12 24]]]
```

In [10]:

```
a30.sum(axis=0)
```

Out[10]:

```
array([[14, 16, 18, 20],
       [22, 24, 26, 28],
       [30, 32, 34, 36]])
```

In [11]:

```
e = np.array([[[1, 0],
        [0, 0]],

       [[1, 1],
        [1, 0]],

       [[1, 0],
        [0, 1]]])
```

In [12]:

```
e.sum(axis=1)
```

Out[12]:

```
array([[1, 0],
       [2, 1],
       [1, 1]])
```

fancy indexing

In [ ]:

```

```