In [2]:
```
#structured arrays in numpy
import numpy as np
```

In [3]:
```
#structured array example
chocolates=np.zeros(4,dtype={"names":("name","price","quantity"),"formats":("U10","f8","i4")})#template for structured array
print(chocolates.dtype)
chocolates["name"]="Dairy Milk"#assinging name as dairy milk
chocolates["price"]=10.50#assigning price as 10.50 for all elements
chocolates["quantity"]=100#assigning quantity as 100 for all elements
print(chocolates)
```

```
[('name', '<U10'), ('price', '<f8'), ('quantity', '<i4')]
[('Dairy Milk', 10.5, 100) ('Dairy Milk', 10.5, 100)
 ('Dairy Milk', 10.5, 100) ('Dairy Milk', 10.5, 100)]
```

In [4]:
```
chocolates=np.zeros(4,dtype={"names":("name","price","quantity"),"formats":("U10","f8","i4")})#template for structured array
print(chocolates.dtype)
namelist=np.array(["Dairy Milk","Milky bar","5Star","Mars"])
pricelist=np.array([10.00,15.75,20.50,50.00])
quantitylist=np.array([10,20,30,40])
chocolates["name"]=namelist#assinging name as dairy milk
chocolates["price"]=pricelist#assigning respective price to each element with quantity list
chocolates["quantity"]=quantitylist#assigning respective quantites to each element with quantity list
print(chocolates)
```

```
[('name', '<U10'), ('price', '<f8'), ('quantity', '<i4')]
[('Dairy Milk', 10.  , 10) ('Milky bar', 15.75, 20) ('5Star', 20.5 , 30)
 ('Mars', 50.  , 40)]
```

In [15]:
```
print("First row of elements=",chocolates[0])#name of first chcoclate along with all details
print("Names of all chocolates=",chocolates["name"])#to print the list of a particular attribute pass the name of attribute
#getting names of all chcoclates less than 30 usig boolean mask
print("Chocolates less than 30=",chocolates[chocolates["price"]<30]["name"])#returns names of chocolates less than 30
```

```
First row of elements= ('Dairy Milk', 10., 10)
Names of all chocolates= ['Dairy Milk' 'Milky bar' '5Star' 'Mars']
Chocolates less than 30= ['Dairy Milk' 'Milky bar' '5Star']
```

In [21]:
```
#tuple representation of structured arrays
sa=np.dtype([("name","U10"),("height","f8"),("gender","S6")])
a=np.zeros(4,dtype=sa)
a["name"]=["A","B","C","D"]
a["height"]=[6.1,5.6,5,6]
a["gender"]=["female","male","female","male"]
print(a)
```

```
[('A', 6.1, b'female') ('B', 5.6, b'male') ('C', 5. , b'female')
 ('D', 6. , b'male')]
```

In [24]:
```
#advanced compound types
adv_type=np.dtype([("id","i8"),("mat","f8",(3,3))])#we use mat data type to use matrix element in structured array
x=np.zeros(1,dtype=adv_type)#creating a zero array
print(x)#printing array
print(x["mat"])#printing the matrix element of the array
```

```
[(0, [[0., 0., 0.], [0., 0., 0.], [0., 0., 0.]])]
[[[0. 0. 0.]
  [0. 0. 0.]
  [0. 0. 0.]]]
```

In [27]:
```
#record array
sa=np.dtype([("name","U10"),("height","f8"),("gender","S6")])
a=np.zeros(4,dtype=sa)
a["name"]=["A","B","C","D"]
a["height"]=[6.1,5.6,5,6]
a["gender"]=["female","male","female","male"]
name_list=a.view(np.recarray)#creating a age_list object using recarray
print(name_list.name)#we can now directly access name
#less efficient than a["name"]
```

```
['A' 'B' 'C' 'D']
```

In [ ]: