Lecture 8

String: Strings are collections of alphabets, words or other characters. In Python, you can create strings by enclosing a sequence of characters within a pair of single or double quotes.

You can also apply the + operations on two or more strings to **concatenate** them.

```
x = 'Cake'
y = 'Cookie'
x + ' & ' + y

# 'Cake & Cookie'
```

Repeat

```
x * 2

# 'CakeCake'
```

You can also **slice** strings, which means that you select parts of strings:

```
# Range Slicing

z1 = x[2:]

print(z1)

# Slicing

z2 = y[0] + y[1]

print(z2)


# ke
Co
```

Capitalize strings

```
str.capitalize('cookie')

#'Cookie'
```

Retrieve the **length** of a string in characters. Note that the spaces also count towards the final result:

```
str1 = "Cake 4 U"
str2 = "404"
len(str1)

# 8
```

Check whether a string consists of **only digits**

```
str1.isdigit()

#False


str2.isdigit()

#True
```

Replace parts of strings with other strings

```
str1.replace('4 U', str2)

#'Cake 404'
```

Find **substrings** in other strings; Returns the lowest index or position within the string at which the substring is found:

```
str1 = 'cookie'
str2 = 'cook'
str1.find(str2)

#0

str1 = 'I got you a cookie'
str2 = 'cook'
str1.find(str2)

#12
```

Data Type Conversion

Implicit Data Type Conversion

```
x=1
```

Explicit Data Type Conversion (**type casting**)

```
x = 2
y = "The Godfather: Part "
fav_movie = y + x

#show error

x = 2
y = "The Godfather: Part "
fav_movie = (y) + str(x)
print(fav_movie)

#The Godfather: Part 2
```

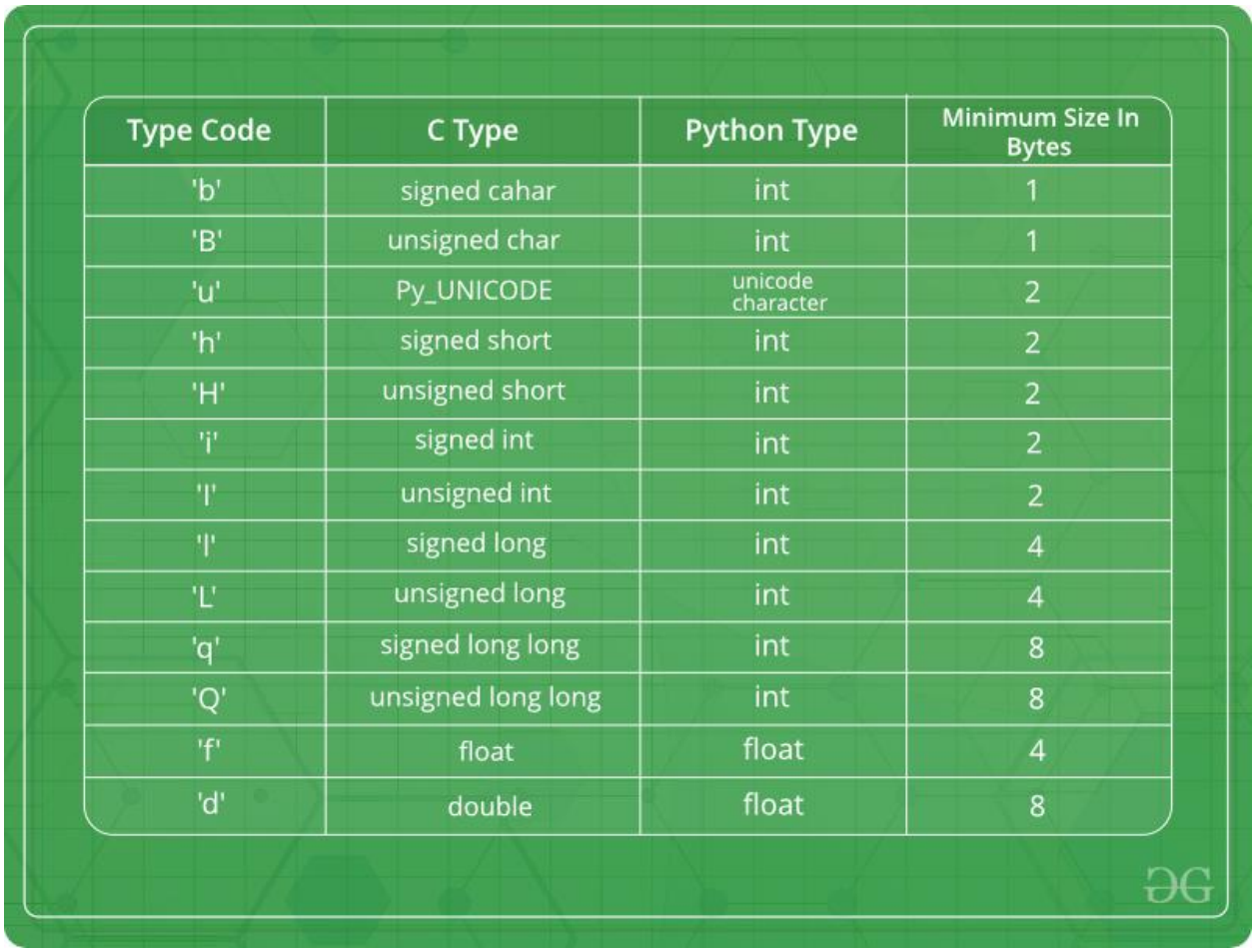
Data structures are a way of organizing and storing data so that they can be accessed and worked with efficiently. They define the relationship between the data, and the operations that can be performed on the data. The keyword “Abstract” is used as, how those operations are working that is totally hidden from the user. There are many various kinds of data structures defined that make it easier for the data scientists and the computer engineers, alike to concentrate on the main picture of solving larger problems rather than getting lost in the details of data description and access. This is known as data abstraction. Generally, data structures can be divided into two categories in computer science: **primitive** and **non-primitive** data structures.

Primitive: Eg: Integer, Float, Strings, Boolean

Non-Primitive Array, List,….. etc..

Array

An array is a vector containing homogeneous elements i.e. belonging to the same data type. Elements are allocated with contiguous memory locations allowing easy modification, that is, addition, deletion, accessing of elements. In Python, we have to use the **array** module to declare arrays.



Type Code	C Type	Python Type	Minimum Size In Bytes
'b'	signed cahar	int	1
'B'	unsigned char	int	1
'u'	Py_UNICODE	unicode character	2
'h'	signed short	int	2
'H'	unsigned short	int	2
'i'	signed int	int	2
'I'	unsigned int	int	2
'l'	signed long	int	4
'L'	unsigned long	int	4
'q'	signed long long	int	8
'Q'	unsigned long long	int	8
'f'	float	float	4
'd'	double	float	8

 # Python program to demonstrate
 # Creation of Array

```
# importing "array" for array creations
import array as arr

# creating an array with integer type
a = arr.array('i', [1, 2, 3])

# printing original array
print ("The new created array is : ", end =" ")
for i in range (0, 3):
    print (a[i], end =" ")
print()

# creating an array with float type
b = arr.array('d', [2.5, 3.2, 3.3])
```

```
# printing original array
print ("The new created array is : ", end =" ")
for i in range (0, 3):
    print (b[i], end =" ")
```

 # Python program to demonstrate
 # accessing of element

```
# importing array module
import array as arr

# array with int type
a = arr.array('i', [1, 2, 3, 4, 5, 6])

# accessing element of array
print("Access element is: ", a[0])

# accessing element of array
print("Access element is: ", a[3])

# array with float type
b = arr.array('d', [2.5, 3.2, 3.3])

# accessing element of array
print("Access element is: ", b[1])

# accessing element of array
print("Access element is: ", b[2])
```

 # Python program to demonstrate
 # Removal of elements in a Array

```
# importing "array" for array operations
import array
# initializing array with array values
# initializes array with signed integers
arr = array.array('i', [1, 2, 3, 1, 5])
# printing original array
print ("The new created array is : ", end ="")
for i in range (0, 5):
    print (arr[i], end =" ")


# using pop() to remove element at 2nd position
```

```
print ("The popped element is : ", end ="")
print (arr.pop(2))
# printing array after popping
print ("The array after popping is : ", end ="")
for i in range (0, 4):
    print (arr[i], end =" ")
# using remove() to remove 1st occurrence of 1
arr.remove(1)
# printing array after removing
print ("The array after removing is : ", end ="")
for i in range (0, 3):
    print (arr[i], end =" ")
```

 # Python program to demonstrate
 # silicing of elements in a Array

```
# importing array module
import array as arr

# creating a list
l = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

a = arr.array('i', l)
print("Intial Array: ")
for i in (a):
    print(i, end =" ")

# Print elements of a range
# using Slice operation
Sliced_array = a[3:8]
print("\nSlicing elements in a range 3-8: ")
print(Sliced_array)

# Print elements from a
# pre-defined point to end
Sliced_array = a[5:]
print("\nElements sliced from 5th "
      "element till the end: ")
print(Sliced_array)

# Printing elements from
# beginning till end
Sliced_array = a[:]
print("\nPrinting all elements using slice operation: ")
print(Sliced_array)
```