Data Types of Python

- **Booleans** are either True or False.
- **Numbers** can be integers (1 and 2), floats (1.1 and 1.2), fractions (1/2 and 2/3), or even complex numbers.
- **Strings** are sequences of Unicode characters, e.g. an html document.
- **Lists** are ordered sequences of values.
- **Tuples** are ordered, immutable sequences of values.
- **Sets** are unordered bags of values.
- **Dictionaries** are unordered bags of key-value pairs.

Immutable Types

- Strings and Tuples are immutable, which means that once we create them, we can not change them.

- List , Sets and dictionaries are mutable, which means we can add , remove elements from them .

String

- Strings are amongst the most popular types in Python.
- We can create them simply by enclosing characters in quotes.
- Python treats single quotes the same as double quotes.
- Creating strings is as simple as assigning a value to a variable

```
var1 = 'Hello World!'
var1
```

```
'Hello World!'
```

```
var2 = "Python Programming"
var2
```

```
'Python Programming'
```

String: Accessing Elements

- Python does not support a character type; these are treated as strings of length one, thus also considered a substring.

- To access each character, use the square brackets along with the index. We can access a specific element using an integer index which counts from the front of the sequence (starting at ZERO!)

```
var1[0]  # Hello World
```

```
'H'
```

```
var1[1]
```

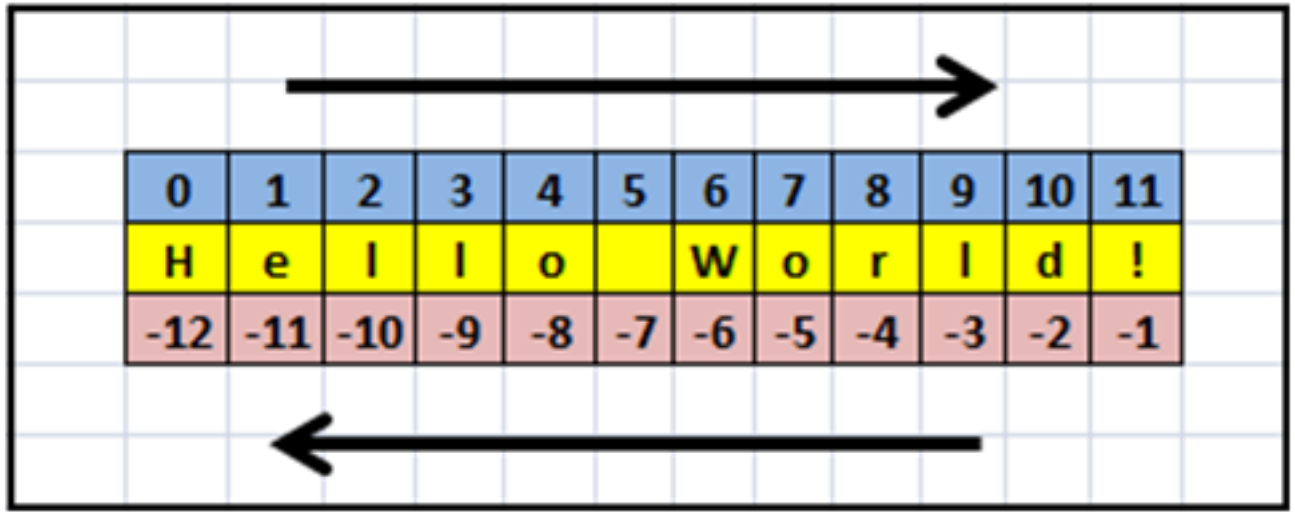
```
'e'
```

```
var2[5] # Python Programming
```

```
'n'
```

Counting Backwards

- We can count from the end of the string using negative numbers also.




```
var1[-1]
```

```
'!'
```

```
var1[-5]
```

```
'o'
```

Index Out of Range!

- If we try to access an element that does not exist, Python will throw an error!

```
var1[100]
```

```
IndexError: string index out of range
```

```
var1[-200]
```

```
IndexError: string index out of range
```

Easy Traversals – The FOR Loop

- Python makes string traversals easy with a FOR loop.

```
for i in var1:
    print(i)
```

```
H
e
l
l
o

W
o
r
l
d
!
```

```
for i in var1:
    if i == 'l':
        print(i)
```

```
l
l
l
```

Grabbing Slices from a String

- The slice operator will clip out part of a sequence.
- It work a lot like the range function, but with a colon that separates the "start" and "end" points.
- Syntax : VAR [START : END : STEP]

```
var1[0:2] # Hello World
```

```
'He'
```

```
var1[3:6]
```

```
'lo '
```

```
var1[3:9:2]
```

```
'l o'
```

Slices – Default Values for Blanks

- If one leave the “start” part blank, it assumes you want zero.

```
var1[:2] # Hello World
```

```
'He'
```

```
var1[:6]
```

```
'Hello '
```

```
var1[:6:2]
```

```
'Hlo'
```

```
var1[5:] # Hello World
```

```
' World!'
```

```
var1[:]
```

```
'Hello World!'
```

```
var1[::-1]
```

```
'!dlroW olleH'
```

- If one leave the “end” blank, it assumes you want until the end of the string

String - Methods

- **format() -** The string format() method formats the given string into a nicer output in Python.

```
name = "Swati"
age = 4
string = "Name : {}, Age : {}".format(name, age)
string
```

```
'Name : Swati, Age : 4'
```

```
string = "Name : {0}, Age : {1}".format(name, age)
string
```

```
'Name : Swati, Age : 4'
```

```
string = "Name : {n}, Age : {a}".format(n=name, a=age)
string
```

```
'Name : Swati, Age : 4'
```

String - Methods

- **Index() -** Returns Index of Substring

```
s1 = 'Python programming is fun.'
s1.index('fun')
```

```
22
```

```
s1.index('n')
```

```
5
```

```
s1.index('n', 14, 20)
```

```
16
```

- **count() -** returns occurrences of substring in string

```
s2 = "Python is awesome, isn't it?"
s2.count('is')
```

```
2
```

- **find() -** Returns the index of first occurrence of substring

```
s1.find('fun')
```

```
22
```

```
s1.find('n')
```

```
5
```

```
s1.find('n', 14, 20)
```

```
16
```

```
s1.find('Java')
```

```
-1
```

String - Methods

- **islower() -** Checks if all Alphabets in a String are Lowercase

```
s1 = 'python academy'
s1.islower()
```

```
True
```

```
s2 = 'python academy#1'
s2.islower()
```

```
True
```

- **isupper() -** Checks if all Alphabets in a String are Uppercase

```
s3 = 'PYTHON ACADEMY'
s3.isupper()
```

```
True
```

- **lower() -** returns lowercased string

```
s4 = 'Python Academy'
s4.lower()
```

```
'python academy'
```

- **upper() -** returns uppercased string

```
s5 = 'Python Academy'
s5.upper()
```

```
'PYTHON ACADEMY'
```

- **swapcase() -** swap uppercase characters to lowercase; vice versa

```
s6 = 'Python Academy'
s6.swapcase()
```

```
'pYTHON aCADEMY'
```

String - Methods

- **capitalize() -** Converts first character to Capital Letter

```
s1 = 'python academy'
s1.capitalize()
```

```
'Python academy'
```

- **lstrip() -** Removes Leading Characters

```
s2 = '  python academy'
s2.lstrip()
```

```
'python academy'
```

- **rstrip() -** Removes Trailing Characters

```
s3 = 'python academy  '
s3.rstrip()
```

```
'python academy'
```

- **isalnum() -** Checks Alphanumeric Character

```
s4 = "PythonAcademy123"
s4.isalnum()
```

```
True
```

- **isalpha() -** Checks if All Characters are Alphabets

```
s4.isalpha()
```

```
False
```

- **isnumeric() -** Checks Numeric Characters

```
s5 = "28212"
s5.isnumeric()
```

```
True
```

List

- A list is a data structure that holds an ordered collection of items i.e. you can store a sequence of items in a list.
- The list of items should be enclosed in square brackets so that Python understands that you are specifying a list.
- Once you have created a list, you can add, remove or search for items in the list.
- Since we can add and remove items, we say that a list is a mutable data type i.e. this type can be altered.

Using List

- A list is represented with [ ] (square brackets) and is created by providing values separated by “,”.

```
x = [1,2,3]
x
```

```
[1, 2, 3]
```

```
y = [10.5,23.45,34.56,45.78]
y
```

```
[10.5, 23.45, 34.56, 45.78]
```

```
z = ['q','r','s','t']
z
```

```
['q', 'r', 's', 't']
```

```
l = [1,34.56,True,'q',4,'r',False]
l
```

```
[1, 34.56, True, 'q', 4, 'r', False]
```

- List allow duplicates of values.

```
x = [10,25,10,38,25]
x
```

```
[10, 25, 10, 38, 25]
```

Using List

- A list contains ordered set of elements , hence can be accessed by using index which starts from 0.

```
x = [10,25,10,38,25]
x[0]
```

```
10
```

```
x[3]
```

```
38
```

```
x[5]
```

```
IndexError: list index out of range
```

```
x[-1]
```

```
25
```

- Index values greater than range then error is generated.
- Reverse index is also possible just like strings.

Empty List Creation

- Empty list can be created in two different ways.

```
a = []
a
```

```
[]
```

```
b = list()
b
```

```
[]
```

```
x = [10,25,10,38,25]
x[1]
```

```
25
```

```
x[1] = 5
x[1]
```

```
5
```

```
x
```

```
[10, 5, 10, 38, 25]
```

- Elements in list can be updated since list is mutable.

List Assignment and Equivalence

- List assignment is used to assign all the elements of list to another list.
- Here , memory is allocated for both objects separately. This is called Deep Copy

```
a = [2,4,6]
a
```

```
[2, 4, 6]
```

```
c[0] = 10
c
```

```
[10, 4, 6]
```

```
c = list(a)
c
```

```
[2, 4, 6]
```

```
a
```

```
[2, 4, 6]
```

```
a = [2,4,6]
a
```

```
[2, 4, 6]
```

```
b[1] = 12
b
```

```
[2, 12, 6]
```

```
b = a
b
```

```
[2, 4, 6]
```

```
a
```

```
[2, 12, 6]
```

- Whereas here , same memory is used. This is called Shallow copy.

List slicing

List slicing is same as string slicing .

```
a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
a[1:4]
```

```
[1, 2, 3]
```

```
a[:6]
```

```
[0, 1, 2, 3, 4, 5]
```

```
a[3:]
```

```
[3, 4, 5, 6, 7, 8, 9, 10]
```

```
a[2:8:2]
```

```
[2, 4, 6]
```

List class Methods

- Since list is a class , it has many methods attached wit it. We can check all methods using following help functio .

```
help(list)
```

```
Help on class list in module builtins:

class list(object)
 |  list(iterable=(), /)
 |
 |  Built-in mutable sequence.
 |
```

- Operator overloaded and len() function

```
L1 = [2,3,4,5]
L1
```

```
[2, 3, 4, 5]
```

```
L2 = [6,7,8,9]
L2
```

```
[6, 7, 8, 9]
```

```
L3 = L1 + L2
L3
```

```
[2, 3, 4, 5, 6, 7, 8, 9]
```

```
L4 = 3 * L1
L4
```

```
[2, 3, 4, 5, 2, 3, 4, 5, 2, 3, 4, 5]
```

```
x = len(L1)
x
```

```
4
```

List class Methods (addition)

- Add new element to the list – append , extend , insert.

```
L1 = [2,3,4,5]
L1
```

```
[2, 3, 4, 5]
```

```
L1.append(6)
L1
```

```
[2, 3, 4, 5, 6]
```

```
L2 = [4,5,6]
L2
```

```
[4, 5, 6]
```

```
L1.append(L2)
L1
```

```
[2, 3, 4, 5, 6, [4, 5, 6]]
```

```
L1.extend(L2)
L1
```

```
[2, 3, 4, 5, 6, [4, 5, 6], 4, 5, 6]
```

```
L1.insert(3,7)
L1
```

```
[2, 3, 4, 7, 5, 6, [4, 5, 6], 4, 5, 6]
```

List class Methods (Deletion)

- Delete element from list – del , pop , remove

```
del L1[2]
L1
```

```
[2, 3, 7, 5, 6, [4, 5, 6], 4, 5, 6]
```

```
del L1
L1
```

```
NameError: name 'L1' is not defined
```

```
L1 = [1,2,3,4,5]
L1
```

```
[1, 2, 3, 4, 5]
```

```
x = L1.pop()
x
```

```
5
```

```
L1
```

```
[1, 2, 3, 4]
```

```
x = L1.pop(2)
L1
```

```
[1, 2, 4]
```

```
L1.remove(2)
L1
```

```
[1, 4]
```

```
L1.remove(5)
L1
```

```
ValueError: list.remove(x): x not in list
```

List class Methods (agg & sort)

- Aggregate methods– max , min ,sum , count , index

```
L1 = [4,8,6,7,8,9]
L1
```

```
[4, 8, 6, 7, 8, 9]
```

```
max(L1)
```

```
9
```

```
min(L1)
```

```
4
```

```
sum(L1)
```

```
42
```

```
L1.count(8)
```

```
2
```

```
L1.index(8)
```

```
1
```

```
L1.index(8,2,5)
```

```
4
```

List class Methods (agg & sort)

Sorting :

```
L1 = [5,2,4,9,7]
L1
```

```
[5, 2, 4, 9, 7]
```

Out-place sorting :

```
L2 = sorted(L1)
print(L2)
print(L1)
```

```
[2, 4, 5, 7, 9]
[5, 2, 4, 9, 7]
```

In-place sorting :

```
# Ascending Sort
L1.sort()
L1
```

```
[2, 4, 5, 7, 9]
```

```
# Descending Sort
L1.sort(reverse=True)
L1
```

```
[9, 7, 5, 4, 2]
```

Reverse :

```
L = [77,55,88,11]
L
```

```
[77, 55, 88, 11]
```

Out-place reverse :

```
L3 = L[::-1]
print(L3)
print(L)
```

```
[11, 88, 55, 77]
[77, 55, 88, 11]
```

In-place reverse :

```
L.reverse()
L
```

```
[11, 88, 55, 77]
```

List Membership operations

- Since list is a sequence , we can use simple loop for selecting each elements one by one.

```
L1 = [2,3,4,5]
L1
```

```
[2, 3, 4, 5]
```

```
for i in L1:
    print(i)
```

```
2
3
4
5
```

- We can directly check if the element is part of given list

```
x = 4
if x in L1:
     print (x, 'is a member element of ' , L1)
else :
      print (x, 'is not a member element of ' , L1)
```

```
4 is a member element of  [2, 3, 4, 5]
```

Nested list

- List can have list itself as its elements.

```
x = [[2,4,6],[3,2,5]]
x
```

```
[[2, 4, 6], [3, 2, 5]]
```

```
for i in x:
    print(i)
    for j in i:
        print(j)
```

```
[2, 4, 6]
2
4
6
[3, 2, 5]
3
2
5
```

- Print individual elements for following list.

```
y = [[[2],[2,3,4],[6,5]],[[3],[2,5]]]
y
```

```
[[[2], [2, 3, 4], [6, 5]], [[3], [2, 5]]]
```

- To update element ‘6’ of list x and y.

```
x[0][2] = 66
x
```

```
[[2, 4, 66], [3, 2, 5]]
```

```
y[0][2][0] = 66
y
```

```
[[[2], [2, 3, 4], [66, 5]], [[3], [2, 5]]]
```

Splitting strings into list

- The **split()** method returns a list of strings after breaking the given string by the specified separator.

  str_var.split(separator, maxsplit)

- *separator* : The string splits at this specified separator. If is not provided then any white space is a separator.

- *maxsplit* : It is a number, which tells us to split the string into maximum of provided number of times. If it is not provided then there is no limit.

```
s = "This is sample string example."
s
```

```
'This is sample string example.'
```

```
l = s.split()
l
```

```
['This', 'is', 'sample', 'string', 'example.']
```

```
l1 = s.split(' ',2)
l1
```

```
['This', 'is', 'sample string example.']
```

```
s1 = "10,20,30,40"
l2 = s1.split(',')
l2
```

```
['10', '20', '30', '40']
```

Joining list into string

- The **join()** method returns a string in which the elements of sequence have been joined by string separator.

S.join(iterable)

- S is the separator between elements in iterable.
- *iterable* – is the objects capable of returning its members one at a time. Some examples are **List, Tuple, String, Dictionary and Set.**

```
l1 = ['This', 'is', 'sample', 'string', 'example.']
l1
```

```
['This', 'is', 'sample', 'string', 'example.']
```

```
s1 = " ".join(l1)
s1
```

```
'This is sample string example.'
```

```
s2 = "-".join(l1)
s2
```

```
'This-is-sample-string-example.'
```

List comprehension

- List comprehensions are used for creating new list from another iterables.

- As list comprehension returns list, they consists of brackets containing the expression which needs to be executed for each element along with the for loop to iterate over each element.

- Basic syntax:

new_list = [expression for_loop_one_or_more condtions]

Find squares of a number using for loop.

```
numbers = [1, 2, 3, 4]
squares = []

for n in numbers:
    squares.append(n**2)

print(squares)
```

```
[1, 4, 9, 16]
```

```
numbers = [1, 2, 3, 4]
squares = [n**2 for n in numbers]

print(squares)
```

```
[1, 4, 9, 16]
```

List comprehension

- Example 2 : Find common numbers from two list using for loop.

```
list_a = [1,2,3,4]
list_b = [2,3,4,5]
```

```
common_num = []
for a in list_a:
    if a in list_b:
        common_num.append(a)
print(common_num)
```

```
[2, 3, 4]
```

```
common_num = [a for a in list_a if a in list_b]

print(common_num)
```

```
[2, 3, 4]
```

List comprehension

- Example 3: Combine the different numbers from two given list.

```
list_a = [1, 2, 3]
list_b = [2, 7]
```

```
different_num = []
for a in list_a:
    for b in list_b:
        if a != b:
            different_num.append((a, b))
print(different_num)
```

```
[(1, 2), (1, 7), (2, 7), (3, 2), (3, 7)]
```

```
different_num = [(a, b) for a in list_a for b in list_b if a != b]

print(different_num)
```

```
[(1, 2), (1, 7), (2, 7), (3, 2), (3, 7)]
```