Python Data Structures and Boolean

- Boolean
- Boolean and Logical Operators
- Lists
- Comparison operators
- Dictionaries
- Tuples
- Sets

Boolean Variables

Boolean values are the two constant objects FALSE and TRUE.

They are used to represent truth values (other values can also be considered false or true).

In numeric contexts (for example, when used as the argument to an arithmetic operator), they behave like the integers 0 and 1, respectively.

The built-in function bool() can be used to cast any value to a Boolean, if the value can be interpreted as a truth value

They are written as False and True, respectively.

In [2]:
```
False
```
Out[2]:
```
False
```

In [3]:
```
# printing the boolean values
print(True,False)
```
```
True False
```

In [4]:
```
# type function is used to check the type of variables
type(True)
```
Out[4]:
```
bool
```

In [5]:
```
type(False)
```
Out[5]:
```
bool
```

In [11]:
```
my_str='Mukesh_pratice1544'
```

In [8]:
```
my_str1 = 'Mukesh'
```

In [10]:
```
print(my_str1.istitle())# from the above line we can see that stating letter is c
print(my_str.istitle())
```
```
True
False
```

In [14]:
```
print(my_str.isalnum()) #check if all char are numbers
print(my_str.isalpha()) #check if all char in the string are alphabetic
print(my_str.isdigit()) #test if string contains digits
print(my_str.istitle()) #test if string contains title words
print(my_str.isupper()) #test if string contains upper case
print(my_str.islower()) #test if string contains lower case
print(my_str.isspace()) #test if string contains spaces
print(my_str.endswith('4')) #test if string endswith a d
print(my_str.startswith('M')) #test if string startswith H
```
```
False
False
False
False
False
False
False
True
True
```

Boolean and Logical Operators

In [10]:
```
True and True
```
Out[10]:
```
True
```

In [11]:
```
True and False
```
Out[11]:
```
False
```

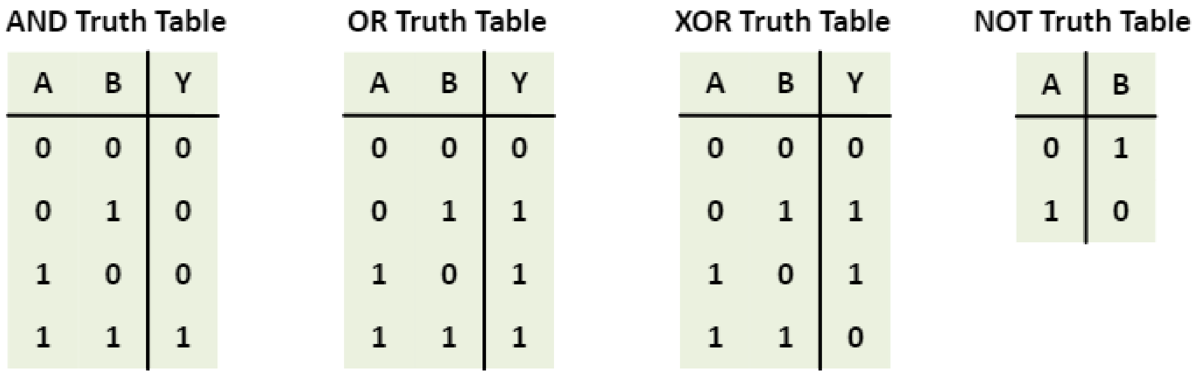
In [12]:
```
True or False
```
Out[12]:
```
True
```

In [13]:
```
True or True
```
Out[13]:
```
True
```

In [15]:
```
str_example='Hello World'
my_str='Mukesh'
```

In [16]:
```
my_str.isalpha() or str_example.isnum()
```
Out[16]:
```
True
```

This is how the boolean works:



AND Truth Table

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

OR Truth Table

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

XOR Truth Table

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

NOT Truth Table

A	B
0	1
1	0

Lists

A list is a data structure in Python that is a mutable, or changeable, ordered sequence of elements. Each element or value that is inside of a list is called an item. Just as strings are defined as characters between quotes, lists are defined by having values between square brackets [ ]

```
In [15]: type([])

Out[15]: list
```

```
In [18]: # creating an empty list
         # one way of creating the list
         lst_example=[]
         # Second way of creating the list is by using the inbuilt function i.e. list
         type(list([1,213,4,3,53,63,56,2,4]))

Out[18]: list
```

```
In [19]: type(lst_example)

Out[19]: list
```

```
In [20]: # generalized method
         lst=list()
```

```
In [21]: type(lst)

Out[21]: list
```

```
In [20]: lst=['C++', 'JAVA', 'PYTHON', 'ML', 'DL', 'CNN']
```

```
In [21]: # len is the built in function for checking the lenght of the element
         len(lst)

Out[21]: 6
```

```
In [17]: type(lst)
```

```
Out[17]: list
```

Append: .append is used to add elements in the list at the last position

```
In [22]: lst.append("I like to learn Data Science")
```

```
In [23]: lst
```

```
Out[23]: ['C++', 'JAVA', 'PYTHON', 'ML', 'DL', 'CNN', 'I like to learn Data Science']
```

```
In [24]: """
         As we know that list are mutable entities so we can add multiple list into a sing
         """
         lst.append(["Kamalesh","Rakesh", "They are also intersted in learning DS"])
```

```
In [25]: lst
```

```
Out[25]: ['C++',
          'JAVA',
          'PYTHON',
          'ML',
          'DL',
          'CNN',
          'I like to learn Data Science',
          ['Kamalesh', 'Rakesh', 'They are also intersted in learning DS']]
```

```
In [26]: # in list indexing operation are accessable and the indexing start with zero
         lst[6]
```

```
Out[26]: 'I like to learn Data Science'
```

```
In [27]: # we can also access sequence element from the list.
         lst[1:6]
```

```
Out[27]: ['JAVA', 'PYTHON', 'ML', 'DL', 'CNN']
```

Insert: It is a built in function that helps to insert the data elements in a specific place

```
In [30]: ## insert in a specific order we need to alloct the index and also element as sho

         lst.insert(0,"C lang")
```

```
In [31]: lst
```

```
Out[31]: ['C lang',
          'C++',
          'JAVA',
          'Kumar',
          'PYTHON',
          'ML',
          'DL',
          'CNN',
          'I like to learn Data Science',
          ['Kamalesh', 'Rakesh', 'They are also intersted in learning DS']]
```

```
In [32]: lst.append(["Hello","World"])
```

```
In [33]: lst
```

```
Out[33]: ['C lang',
          'C++',
          'JAVA',
          'Kumar',
          'PYTHON',
          'ML',
          'DL',
          'CNN',
          'I like to learn Data Science',
          ['Kamalesh', 'Rakesh', 'They are also intersted in learning DS'],
          ['Hello', 'World']]
```

```
In [34]: lst=[1,2,3]
```

```
In [35]: lst.append([4,5])
```

```
In [36]: lst
```

```
Out[36]: [1, 2, 3, [4, 5]]
```

Extend Method: This method is used to extend the existing list.

```
In [39]: lst=[1,2,3,4,5,6]
```

```
In [40]: lst.extend([8,9])
```

```
In [41]: lst
```

```
Out[41]: [1, 2, 3, 4, 5, 6, 8, 9]
```

Various Operations that we can perform in List

```
In [42]: lst=[1,2,3,4,5] # new list
```

```
In [43]: sum(lst) # summing allthe element in the list
Out[43]: 15
```

```
In [44]: lst*5 # printing a list 5 times
Out[44]: [1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
```

Pop() Method: This pop() is used to delete the element from the list

```
In [47]: lst.pop()
Out[47]: 4
```

```
In [48]: lst
Out[48]: [1, 2, 3]
```

```
In [49]: lst.pop(0)
Out[49]: 1
```

```
In [50]: lst
Out[50]: [2, 3]
```

count():Calculates total occurrence of given element of List

```
In [51]: lst=[1,1,2,3,4,5]
         lst.count(1)
Out[51]: 2
```

```
In [52]: #length:Calculates total length of List
         len(lst)
Out[52]: 6
```

```
In [53]: # index(): Returns the index of first occurrence. Start and End index are not nec
         lst.index(1,1,4)
Out[53]: 1
```

```
In [54]: ##Min and Max
         min(lst)
Out[54]: 1
```

In [57]:
```
max(lst)
```
Out[57]:
```
5
```

SETS

A Set is an unordered collection data type that is iterable, mutable, and has no duplicate elements. Python's set class represents the mathematical notion of a set.This is based on a data structure known as a hash table

In [55]:
```
## Defining an empy set

set_var= set()
print(set_var)
print(type(set_var))
```
```
set()
<class 'set'>
```

In [56]:
```
set_var={1,2,3,4,3}
```

In [57]:
```
set_var
```
Out[57]:
```
{1, 2, 3, 4}
```

In [60]:
```
set_var={"Rakesh","Kumar",'Sharma'} # new set
print(set_var) # printing the set values
type(set_var) # checking the type
```
```
{'Sharma', 'Rakesh', 'Kumar'}
```
Out[60]:
```
set
```

In [61]:
```
"""
From the above example we know that the set is a unordered collection
"""
```
Out[61]:
```
'\nFrom the above example we know that the set is a unordered collection\n'
```

In [62]:
```
## Inbuilt function in sets

set_var.add("Hulk") # add the element at any locatiion.
```

In [63]:
```
print(set_var)
```
```
{'Sharma', 'Hulk', 'Rakesh', 'Kumar'}
```

```
In [64]: # creating multiple sets
         set1={"Mukesh","Kumar",'Sharma'}
         set2={"Rakesh","Kumar",'Sharma','Engg.'}
```

```
In [65]: set2.intersection_update(set1)
```

```
In [66]: set2
```

```
Out[66]: {'Kumar', 'Sharma'}
```

```
In [72]: ##Difference
         set2.difference(set1)
         set2
```

```
Out[72]: set()
```

```
In [73]: set2
```

```
Out[73]: set()
```

```
In [74]: ## Difference update

         set2.difference_update(set1)
```

```
In [71]: print(set2)
```

```
set()
```

Dictionaries

A dictionary is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values.

```
In [75]: dic={}
```

```
In [76]: type(dic)
```

```
Out[76]: dict
```

```
In [77]: type(dict())
```

```
Out[77]: dict
```

```
In [78]: set_ex={1,2,3,4,5}
```

```
In [79]: type(set_ex)
```

```
Out[79]: set
```

In [80]:
```
## Let create a dictionary

my_dict={"Car1": "Audi", "Car2":"BMW","Car3":"Mercidies Benz"}
```

In [81]:
```
type(my_dict)
```

Out[81]:
```
dict
```

In [82]:
```
##Access the item values based on keys

my_dict['Car1']
```

Out[82]:
```
'Audi'
```

In [83]:
```
# We can even loop throught the dictionaries keys

for x in my_dict:
    print(x)
```

```
Car1
Car2
Car3
```

In [84]:
```
# We can even loop throught the dictionaries values

for x in my_dict.values():
    print(x)
```

```
Audi
BMW
Mercidies Benz
```

In [85]:
```
# We can also check both keys and values
for x in my_dict.items():
    print(x)
```

```
('Car1', 'Audi')
('Car2', 'BMW')
('Car3', 'Mercidies Benz')
```

In [86]:
```
## Adding items in Dictionaries

my_dict['car4']='Audi 2.0' # add the new key value pair at the end
```

In [87]:
```
my_dict
```

Out[87]:
```
{'Car1': 'Audi', 'Car2': 'BMW', 'Car3': 'Mercidies Benz', 'car4': 'Audi 2.0'}
```

In [88]:
```
my_dict['Car1']='MAruti' # replacing the value
```

In [89]:
```
my_dict
```

Out[89]:
```
{'Car1': 'MAruti', 'Car2': 'BMW', 'Car3': 'Mercidies Benz', 'car4': 'Audi 2.0'}
```

Nested Dictionary

In [90]:
```
car1_model={'Mercedes':1960}
car2_model={'Audi':1970}
car3_model={'Ambassador':1980}

car_type={'car1':car1_model,'car2':car2_model,'car3':car3_model}
```

In [91]:
```
print(car_type)
```

```
{'car1': {'Mercedes': 1960}, 'car2': {'Audi': 1970}, 'car3': {'Ambassador': 198
0}}
```

In [92]:
```
## Accessing the items in the dictionary

print(car_type['car1'])
```

```
{'Mercedes': 1960}
```

In [93]:
```
print(car_type['car1']['Mercedes'])
```

```
1960
```

Tuples

In [94]:
```
## create an empty Tuples

my_tuple=tuple()
```

In [95]:
```
type(my_tuple)
```

Out[95]:
```
tuple
```

In [96]:
```
my_tuple=()
```

In [97]:
```
type(my_tuple)
```

Out[97]:
```
tuple
```

In [105]:
```
my_tuple=("Mukesh","Kamalesh","Rakesh")
```

In [106]:
```
print(type(my_tuple))
print(my_tuple)
```

```
<class 'tuple'>
('Mukesh', 'Kamalesh', 'Rakesh')
```

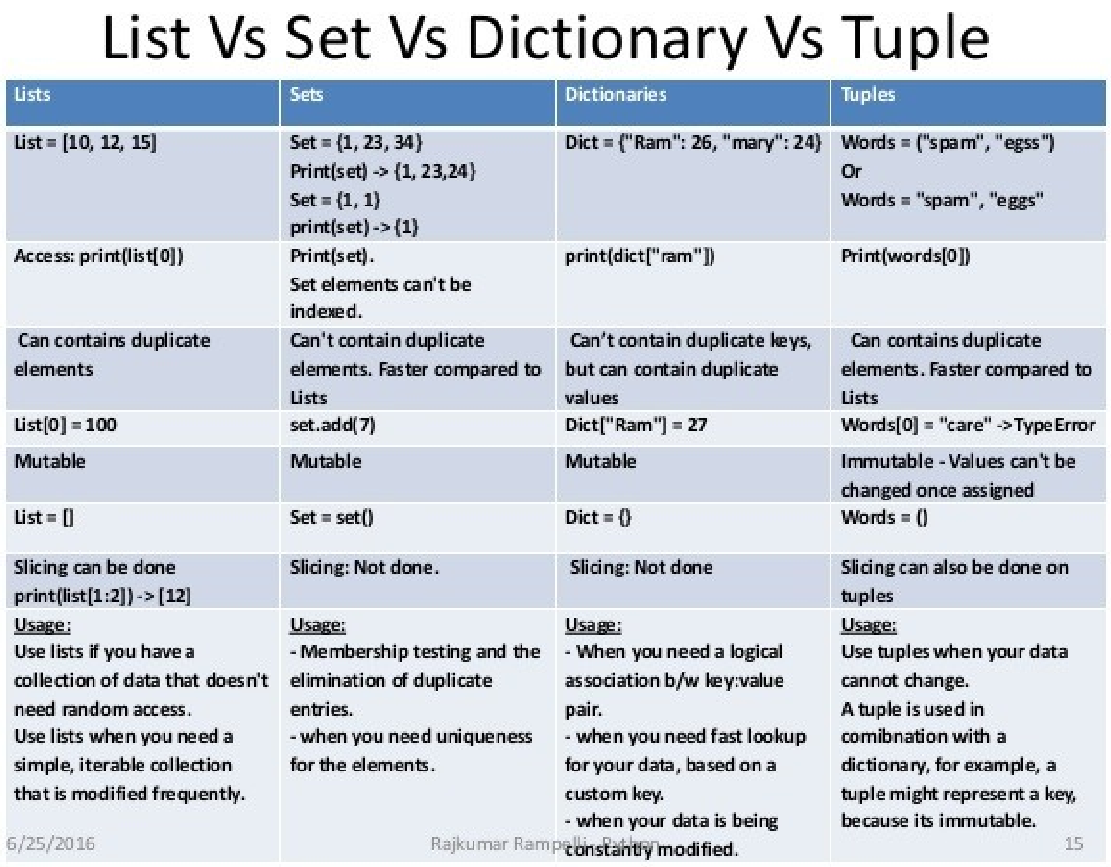
In [108]:
```
type(my_tuple)
```

Out[108]:
```
tuple
```

```
In [109]: ## Inbuilt function
          my_tuple.count('Mukesh')

Out[109]: 1
```

```
In [110]: my_tuple.index('Kamalesh')

Out[110]: 1
```

```
In [111]: '''
          Note: Always indexing start with 0 position only.
          '''

Out[111]: '\nNote: Always indexing start with 0 position only.\n'
```



List Vs Set Vs Dictionary Vs Tuple

Lists	Sets	Dictionaries	Tuples

List = [10, 12, 15]	Set = {1, 23, 34}
	Print(set) -> {1, 23,24}
	Set = {1, 1}
	print(set) -> {1}
		Dict = {"Ram": 26, "mary": 24}	Words = ("spam", "egss")
			Or
			Words = "spam", "eggs"

Access: print(list[0])	Print(set). Set elements can't be indexed.	print(dict["ram"])	Print(words[0])
Can contains duplicate elements	Can't contain duplicate elements. Faster compared to Lists	Can't contain duplicate keys, but can contain duplicate values	Can contains duplicate elements. Faster compared to Lists
List[0] = 100	set.add(7)	Dict["Ram"] = 27	Words[0] = "care" -> TypeError
Mutable	Mutable	Mutable	Immutable - Values can't be changed once assigned
List = []	Set = set()	Dict = {}	Words = ()
Slicing can be done print(list[1:2]) -> [12]	Slicing: Not done.	Slicing: Not done	Slicing can also be done on tuples
Usage: Use lists if you have a collection of data that doesn't need random access. Use lists when you need a simple, iterable collection that is modified frequently.	Usage: - Membership testing and the elimination of duplicate entries. - when you need uniqueness for the elements.	Usage: - When you need a logical association b/w key:value pair. - when you need fast lookup for your data, based on a custom key. - when your data is being constantly modified.	Usage: Use tuples when your data cannot change. A tuple is used in comibnation with a dictionary, for example, a tuple might represent a key, because its immutable.

6/25/2016 Rajkumar Rampelli - Python 15

```
In [ ]:
```