Introduction to Python

The following concepts shall be covered in the notebook

- Print Statements
- Variables
- Different Data-Types
- String Formatting

Printing output - We shall use a string (sequence of words) to print the sentence to the output

- We have an input cell and output cell in the notebook.
- Passing a value to the print() function shall be used to print the output to the screen
- In python, the sentences are stored in either ' quotes or " quotes
- We can use them to print the values in the following way

In [2]:
```
print('Hello World')
```

```
Hello World
```

In [3]:
```
print("Hello World")
```

```
Hello World
```

We have seen how the output is print using the print in-built function Now , we shall start with the basic values (Numeric/String) in python

- The numeric values that can be used in python are of three types
  - Integer
  - Float
  - Complex Type

- Integer type

In [1]:
```
22
```

Out[1]:
```
22
```

- Floting type

In [5]:
```
22.32
```

Out[5]:
```
22.32
```

- Complex Type

In [6]:
```
3+4j
```

Out[6]:
```
(3+4j)
```

Let us perform some basic operations on the numbers

- Addition
- Subtraction
- Multiplication
- Division
- exponent

In [8]:
```
## Addition
3+4
```

Out[8]:
```
7
```

In [9]:
```
## Subtraction
8-5
```

Out[9]:
```
3
```

In [10]:
```
### Multiplication
3*5
```

Out[10]:
```
15
```

In [11]:
```
### Division
5/3
```

Out[11]:
```
1.6666666666666667
```

In [16]:
```
### Division with two front slashes provides the output as below -
### it rounds off to the nearest integer.
5//3
```

Out[16]:
```
1
```

In [17]:
```
12/2
```

Out[17]:
```
6.0
```

In [18]:
```
12//2
```

Out[18]:
```
6
```

In [22]:
```
## Exponent
2**3
```

Out[22]:
```
8
```

In [23]:
```
## Exponent
4**2
```

Out[23]:
```
16
```

Lets look at how to represent the string data type in Python

When we say string - It is sequence of characters.

- String are generally enclosed in single quotes or double quotes.

In [28]:
```
### Use quotes to represent the data
'Hello World'
```

Out[28]:
```
'Hello World'
```

In [29]:
```
### Print 'Hello World!' to console
print('Hello World!')
```

```
Hello World!
```

In [25]:
```
"Hello World!"
```

Out[25]:
```
'Hello World!'
```

In [30]:
```
### Print "Hello World!" to console
print("Hello World!")
```

```
Hello World!
```

In [26]:
```
### Represent 'This is python' in single quotes
'This is python '
```

Out[26]:
```
'This is python '
```

In [31]:
```
#### Print 'This is python ' in single quotes and
### pass it to the print function
print('This is python ')
```

```
This is python
```

In [27]:
```
#### Represent "This is Python " in double quotes
"This is Python"
```

Out[27]:
```
'This is Python'
```

In [32]:
```
### Print "This is Python" in double quotes
### and pass it to the print function
print("This is Python")
```

```
This is Python
```

How to know which data type each value represents

- Use the type function

In [33]:
```
type(20)
```

Out[33]:
```
int
```

In [34]:
```
type(3.14)
```

Out[34]:
```
float
```

In [35]:
```
type('Hello')
```

Out[35]:
```
str
```

In [36]:
```
type(3+3j)
```

Out[36]:
```
complex
```

How to store values in Python - Use Variables

- Use of variables in python
- Python is a dynamically typed programming language
  - Dynamically Typed programming language is one where reassigment of values for different datatypes is possible

In [1]:
```
# Below x is a variable and it is being
# assigned a integer value of 10
x = 10
```

In [38]:
```
x
```

Out[38]:
```
10
```

In [42]:
```
## check the type of x - It is of type integer.
type(x)
```

Out[42]:
```
int
```

Y is a variable and it is being assigned a float value of 20

In [40]:
```
y = 20.33
```

In [41]:
```
y
```

Out[41]:
```
20.33
```

Check the type of y - It is of type float

In [43]:
```
type(y)
```

Out[43]:
```
float
```

z is a variable and it is being assigned a complex numebr of 3+2j

In [3]:
```
z = 3 + 2j
z
```

Out[3]:
```
(3+2j)
```

Check the type of z it is of complex type.

In [47]:
```
type(z)
```

Out[47]:
```
complex
```

In [50]:
```
z.real
```

Out[50]:
```
3.0
```

A complex number has two parts - real part and imaginary part
imaginary part is got from z.imag

In [4]:
```
z.imag
```

```
2.0
```

Out[4]:

- Exercise - Compute area of triangle with given base and height

In [4]:
```
### area of trainge is given using the formula 0.5*b*h
base = 20
height = 15
area = 0.5 * base * height
print("Area of trinagle is ", area)
```

```
Area of trinagle is  150.0
```

Let us see how strings can be formatted

We want to print within the enclosed single quotes.
We can do as below stands for new line

In [7]:
```
print('hello \new world')
```

```
hello
ew world
```

- Two ways to escape the string character as mentioned below

In [14]:
```
### using the raw format of the string
print(r'hello \new world')
```

```
hello \new world
```

If we want to escape the \ we use another \ to escape it

In [9]:
```
print('hello \\new world')
```

```
hello \new world
```

- to specify tab space in the string

In [10]:
```
## If  use \t then it used as tab
print('hello \t world')
```

```
hello    world
```

- Specify the apostrophe and present it as an escape character

We want to include quote in the output of print statement.
We can print the statement as follows.

In [11]:
```
print('it\'s a new world')
```

```
it's a new world
```

- Multi line statements in print statements

In [12]:
```
### Multiple line can be used in the following way - Triple Single quote
print('''This is how first line
second line
```

```
third line are printed
''')
```

```
This is how first line
second line
third line are printed
```

In [13]:
```
### Multiple line can be used in the following way -Triple Double quote
print("""This is how first line
second line
third line are printed
""")
```

```
This is how first line
second line
third line are printed
```

Formatting string values

- Using index values within curly braces to print the output value

Using format function on top of the stirng value to print the output as follows

In [21]:
```
print('Product of 2 and 3 is {}'.format(2*3))
```

```
Product of 2 and 3 is 6
```

Let say we have more that 2 value and want to
format in the string values it can be done
as following

In [22]:
```
print('Product of {0} and {1} is {2} '.format(2,3,2*3))
```

```
Product of 2 and 3 is 6
```

For string values it will be denoted as following

In [28]:
```
print('{0} {1} {2} {3}'.format("How","are","you","doing ?"))
```

```
How are you doing ?
```

- Expressing float values in print statements

In [26]:
```
print('Value of pi is {}'.format(3.14))
```

```
Value of pi is 3.14
```

- Exercise using string foramtting print the area of triangle in a single line

In [ ]:
```
height = 50
length = 10
print('The area of traingle is {}'.format(0.5 * height * length))
## write code inside the print function
```

using varaibles in print statements

- integer and float variables in print statements

In [30]:
```
x = 10
print('Value is ' , x)
```

```
Value is  10
```

In [31]:
```
## The below throws an error
print('value is' + x)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[31], line 2
      1 ## The below throws an error
----> 2 print('value is' + x)

TypeError: can only concatenate str (not "int") to str
```

In [32]:
```
### To correct the above statement we shall do the following
print('Value is ' + str(x))
```

```
Value is 10
```

- We shall come back to print formatting later as we get into other data structures in python

Let us look at some other functions for integers, float and string

Integer or float value is given -
we need to convert it to string.
The following we can perform it

In [42]:
```
value = 10
str(value)
```

Out[42]:
```
'10'
```

In [43]:
```
fl2 = 30.5
str(fl2)
```

Out[43]:
```
'30.5'
```

Integer/float String value is given,
we need to convert that into int or float value

In [49]:
```
str1 = '10'
print(str1 + ' ' , type(str1))
print(str1 + ' ' , int(str1) , type(int(str1)))
```

```
10  <class 'str'>
10  10 <class 'int'>
```

In [51]:
```
str2 = '10.54'
print(str2 + ' ' , type(str2))
print(str2 + ' ', float(str2), type(float(str2)))
```

```
10.54  <class 'str'>
10.54  10.54 <class 'float'>
```

Python Lesson 2

In this notebook we shall cover the following aspects

- Other in-built functions
- Capture input from the command line
- Operators and Operands in Python

- divmod() function returns a pair of numbers consisting of quotient and remainder

We pass two numbers to divmod - 5,3 : div(5,3)
It returns the quotient of the division between two
numbers and remainder of the division

In [14]:
```
divmod(5,3)
```

Out[14]:
```
(1, 2)
```

- ord() function return point value representation of a character

Finding the unicode point value of a character 'H' - pass 'H'
to the ord() function

In [2]:
```
ord('H')
```

Out[2]:
```
72
```

Finding the unicode point value of a character 'a' - pass 'a' to the ord() function

In [3]:
```
ord('a')
```

Out[3]:
```
97
```

Finding unicode point value of a character 'A'
pass 'A' to the ord() function

In [4]:
```
ord('A')
```

Out[4]:
```
65
```

Finding unicode point value of a character '\n'- pass '\n' to the ord() function

In [22]:
```
ord('\n')
```

Out[22]:
```
10
```

- chr() function is the inverse of ord() - It gives the value of the give integer

chr function that takes input as 10 would return the output as '\n'

In [21]:
```
chr(10)
```

Out[21]:
```
'\n'
```

chr function that takes input as 97 would return the output as 'a'

In [25]:
```
chr(97)
```

Out[25]:
```
'a'
```

chr function that takes input as 97 would return the output as 'A'

In [26]:
```
chr(65)
```

Out[26]:
```
'A'
```

chr function that takes input as 8641 would return the output as %'

In [1]:
```
chr(8541)
```

Out[1]:
```
'⅝'
```

- id() function gives the Cpython address location of the object

We want to get the address of a python object - id() function provides the address of that variable

In [12]:
```
x = 10
id(x)
```

Out[12]:
```
1455356117520
```

Finding the memory location address of a floating value passed to id function

In [13]:
```
y = 20.5
id(y)
```

Out[13]:
```
1455436509232
```

- bin() function - This will provide the binary value of the given input

pass value 10 to the bin() function and
it returns binary representation of the passed value
The binary representation is prefixed with 0b

In [28]:
```
bin(10)
```

Out[28]:
```
'0b1010'
```

pass value -10 to the bin() function and
it returns binary representation of the passed value
The binary representation is prefixed with -0b

In [29]:
```
bin(-10)
```

Out[29]:
```
'-0b1010'
```

- hex() function - Converts the number from string to hexadecimal representation

pass value 10 to the hex() function and
it returns the hexadecimal representation of the passed value
output would 0xa

In [31]:
```
hex(10)
```

Out[31]:
```
'0xa'
```

pass value 8 to the hex() function
and it returns the hexadecimal representation of the passed value
output would 0x8

In [9]:
```
hex(23)
```

Out[9]:
```
'0x17'
```

pass value 17 to the hex() function and
it returns the hexadecimal representation of the passed value
output would 0x8

In [33]:
```
hex(17)
```

Out[33]:
```
'0x11'
```

- oct function - Coverts the number from String representation to octal representation

pass value 10 to the oct() function and
it returns the octal representation of the passed value
output would 0o12 - Prefix is 0o

In [35]:
```
oct(10)
```

Out[35]:
```
'0o12'
```

pass value 15 to the oct() function and </br> it returns the octal representation of the passed value </br>
output would 0o17 - Prefix is 0o </br>

In [36]:
```
oct(15)
```

Out[36]:
```
'0o17'
```

pass value 13 to the oct() function and
it returns the octal representation of the passed value
output would 0o15 - Prefix is 0o

In [37]:
```
oct(13)
```

Out[37]:
```
'0o15'
```

- int() function - Integer values with other bases

default base value of an integer

In [14]:
```
int(10)
```

Out[14]:
```
10
```

Passing a string value '10' for base 2

In [16]:
```
int('10',base=2)
```

Out[16]:
```
2
```

Passing a string value '10001' for base 2 - </br> That is the representation is 10001 in base 2 </br> This number is converted to base 10. </br>

In [17]:
```
int('10001',base=2)
```

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

Passing a string value '0xa' for base 6 - </br> that is the representation is 0xa in base 16 </br> The number is 10 in base 10 </br>

In [8]:
```
int('0xa',base=16)
```

Out[8]:
```
10
```

Converts the number from base 8 to base 10

In [5]:
```
int('0o15', base=8)
```

Out[5]:
```
13
```

Converts the number from base 16 to base 10 -
that is the representation is 0x17 in base 16
The number is 23 in base 10

In [10]:
```
int('0x17',base=16)
```

Out[10]:
```
23
```

- abs() function returns the absolute value. Indicating that it returns the positive value

pass 10.2 value to abs function -
it generated the absolute value of the argument

In [11]:
```
abs(-10)
```

Out[11]:
```
10
```

pass 20.5 value to abs function - it generated
the absolute value of the argument

In [12]:
```
abs(-20.5)
```

Out[12]:
```
20.5
```

- pow() function returns the number for this formula - base** exponent
- base of the number and the exponent it is raised to

pass 2 as the base followed by 3 as the exponent - 2**3 - pow(2,3)

In [13]:
```
pow(2,3)
```

Out[13]:
```
8
```

pass 3 as the base followed by 2 as the exponent - 3**2 - pow(3,2)

In [14]:
```
pow(3,2)
```

Out[14]:
```
9
```

- min() function returns the minimum of two or more values

Pass the values as -5, 10, 30 to the minimuum function
and check the output - min(-5,10.30)

In [12]:
```
min(-5,10,30)
```

Out[12]:
```
-5
```

Pass the value as 1,2,3 to the minimum function and
check the output - min(1,2,3)

In [13]:
```
min(1,2,3)
```

Out[13]:
```
1
```

If we pass only one value in the minimum function - then we get an error

In [14]:
```
min(1)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[14], line 1
----> 1 min(1)

TypeError: 'int' object is not iterable
```

- max() function returns the maximum of two or more values

Pass the values as -5, 10, 30 to the maximum function
and check the output - max(-5,10.30)

In [15]:
```
max(-5,10,30)
```

Out[15]:
```
30
```

Pass the value as 1,2,3 to the maximum function
and check the output - max(1,2,3)

In [16]:
```
max(1,2,3)
```

Out[16]:
```
3
```

- len() function gives us the length of the given string
- We cannot pass integer or floating point value

Let us pass string values to the len() function to check its length - 'Python'
returns length of 6

In [5]:
```
len('Python')
```

Out[5]:
```
6
```

Let us pass string value to len() function
to check its length - 'Hello World!'
return the length of 12

In [6]:
```
len('Hello World!')
```

Out[6]:
```
12
```

- repr() function gives back the printable representation of the object

Pass 'monday' as input to repr. repr('Monday') --
The resultant output is 'Monday'

In [11]:
```
repr('Monday')
```

Out[11]:
```
"'Monday'"
```

Pass repr('monday') as input to print. print(repr('Monday'))
The resultant output is 'Monday'

In [9]:
```
print(repr('Monday'))
```

```
'Monday'
```

String Operations

- Text sequences - String
- String are sequences of text. String sequences are immutable.

We import the string module and use ascii_letters on string.
The output is english alphabets in lower case
and upper case.

In [16]:
```
import string
string.ascii_letters
```

Out[16]:
```
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

The ascii_lowercase shall return

the lower case letters on string.
The output is english alphabets in lower case

In [17]:
```
string.ascii_lowercase
```

Out[17]:
```
'abcdefghijklmnopqrstuvwxyz'
```

the ascii_upppercase shall
return the upper case letters as string.
The output is english alphabets in upper case

In [18]:
```
string.ascii_uppercase
```

Out[18]:
```
'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

The digits shall return the digits as string.
The ouput is '0123456789'

In [19]:
```
string.digits
```

Out[19]:
```
'0123456789'
```

The digits shall return hexadecimal digits as string.
The output is '0123456789abcdefABCDEF'

In [20]:
```
string.hexdigits
```

Out[20]:
```
'0123456789abcdefABCDEF'
```

The digits shall return octal digits as string.
The output is '01234567'

In [21]:
```
string.octdigits
```

Out[21]:
```
'01234567'
```

The digits shall return punctutation marks as string.
The output is '!"#$%&\'()*+,-./:;<=>?@[\]^_`{|}~'

In [22]:
```
string.punctuation
```

Out[22]:
```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

- The Stirng value can be stored in three ways
  - In single quotes
  - In double quotes
  - In triple quotes

Textutal data can be put in single quotes.
for example text_s_quote = 'Tuticorn has two words tuti and corn'

In [2]:
```
text_s_quote = 'Tuticorn has two words tuti and corn'
text_s_quote
```

Out[2]:
```
'Tuticorn has two words tuti and corn'
```

Textual data can be put in double quotes.
For Example text_d_Quote = "Tuticorn has two words tuti and corn"

In [4]:
```
text_d_Quote = "Tuticorn has two words tuti and corn"
text_d_Quote
```

Out[4]:
```
'Tuticorn has two words tuti and corn'
```

Textual data can be put in triple quotes.
It can be multiple line also.
For exampe text_tr_Quote = """ Tuticorn has two words tuti and corn """

In [9]:
```
text_tr_Quote = """
Tuticorn has
two words  - tuti
and corn
"""
text_tr_Quote
```

Out[9]:
```
'\nTuticorn has\ntwo words  - tuti\nand corn\n'
```

- Accessing the string values based on indices - [:] index start from zero

|P|R|B|H|A|S|

==========

|0|1 |2 |3 |4|5|

In [10]:
```
###"""|T|u|t|i|c|o|r|n| |h|a |s |  | t| w| o|  | w| o |r |d |s |22 | t| u| t| i | | a |
###"""|0|1|2|3|4|5|6|7|8|9|10|11|12|13|14|15|16|17|18|19|20|21|22 |23|24|25| 26|27|28|29
### String values can be accessed using index operators.
### text_data = "Tuticorn has two words tuti and corn"
### text_data[0:5]
text_data = "Tuticorn has two words tuti and corn"
text_data[0:5]
```

Out[10]:
```
'Tutic'
```

If string value are provided as [start:] where start=0 or [:] , then entire string will be provided """|T|u|t|i|c|o|r|n| |h|a |s | | t| w| o| | w| o |r |d |s |22 | t| u| t| i | | a | n| d| |c|o|r|n|"""
"""|0|1|2|3|4|5|6|7|8|9|10|11|12|13|14|15|16|17|18|19|20|21|22 |23|24|25| 26|27|28|29| 30|31| ""

In [15]:
```
text_data = "Tuticorn has two words tuti and corn"
text_data[0:]
```

Out[15]:
```
'Tuticorn has two words tuti and corn'
```

If string values are provided as [start:end] where start is some number and
end is some number less than length of the string.
Then it slices only that many end-start+1

"""|T|u|t|i|c|o|r|n| |h|a |s | | t| w| o| | w| o |r |d |s |22 | t| u| t| i | | a | n| d| |c|o|r|n|"""
"""|0|1|2|3|4|5|6|7|8|9|10|11|12|13|14|15|16|17|18|19|20|21|22 |23|24|25| 26|27|28|29| 30|31| ""

start = 3 end = 15 text_data[start:end]

- String methods
  - capitalize()
  - count()
  - endswith

- captilize() - This method on string returns a copy by making the first character of the string in uppercase and the reset lowercased

In [18]:
```
### "tuticorn".capitalize() shall return the output as Tuticorn
"tuticorn".capitalize()
```

Out[18]:
```
'Tuticorn'
```

- count() - This method counts the number of times a given sub-string occurs in a given string

In [19]:
```
### "Tuticorn has two words tuti and corn".count('corn')
## counts the number of time corn has appeared
"Tuticorn has two words tuti and corn".count('corn')
```

Out[19]:
```
2
```

- center() - Centres the string with the necessary padding at the ends

In [30]:
```
### "Tuticorn" centers the string with the necessary padding at the ends.
print("tuticorn".center(12))
"tuticorn".center(12,"=")
```

Out[30]:
```
  tuticorn
'==tuticorn=='
```

- endswith() returns true if the string is available as suffix at the end.

In [33]:
```
### "Tuticorn has two words tuti and corn".endswith("corn") shall return True
"Tuticorn has two words tuti and corn".endswith("corn")
```

Out[33]:
```
True
```

- in operator can also be used to check if a sub-string is availble in String

In [37]:
```
### "corn" in "Tuticorn has two words tuti and corn" returns True
"corn" in "Tuticorn has two words tuti and corn"
```

Out[37]:
```
True
```

In [39]:
```
### "corns" in "Tuticorn has two words tuti and corn" returns False
"corns" in "Tuticorn has two words tuti and corn"
```

Out[39]:
```
False
```

- format on string can be used for formatting the string. The string on which this method is called can contain literal text or replacement fields delimited by braces {}

In [45]:
```
### Passing the value without any index
### "corn can be good during a {}".format("movie")
### {} is replaced with movie
"corn can be good during a {}".format("movie")
```

Out[45]:
```
'corn can be good during a movie'
```

In [46]:
```
#### Index can be also be used to replace the positional
#### argument with the string value
#### "value of two numbers {0} and {1} is {2} ".format(4,5,9)
"value of two numbers {0} and {1} is {2} ".format(4,5,9)
```

Out[46]:
```
'value of two numbers 4 and 5 is 9 '
```

- find() - find function finds the index of the substring in the string </br>
  `If the substring is not found -1 will be returned`

In [50]:
```
#### corn function finds it at the index
## print()"Tuticorn has two words tuti and corn".find("corn")) at 4
print("Tuticorn has two words tuti and corn".find("corn"))
print("Tuticorn has two words tuti and corn".find("has",15,36))
```

```
4
-1
```

- isalpha() - Returns True if all characters in the string are e string are alphabetic and there is at least one character, False otherwise

In [54]:
```
#### "234".isalpha() returns True, "helloworld".isalpha() returns True
print("234".isalpha())
print("helloworld".isalpha())
```

```
False
True
```

- isdecimal() returns True , if the value is decimal (0-9) else it returns false.

In [63]:
```
#### "10.23".isdecimal() returns False "234".isdecimal()
####  returns False,"\u00B3".isdecimal() returns False
print("10.23".isdecimal())
print("234".isdecimal())
print("\u00B3".isdecimal())
```

```
False
True
False
```

- isdigit() returns true if all charaters are digits and atleast one character. Else it returns False. Unicode representations are also considered as digits

In [60]:
```
#### "121".isdigit() returns True, "A123".isdigit()
###  returns False, "\u00B3".isdigit() returns True
```

```
print("121".isdigit())
print("A123".isdigit())
print("\u00B3".isdigit())
```

```
True
False
True
```

- islower() returns true if the case of the all characters is lowercase. Else False otherwise

In [62]:
```
### "tuticorn has two words tuti and corn".islower() returns True.
###  "TUTICORN has two words tuti and corn".islower()
print("tuticorn has two words tuti and corn".islower())
print("TUTICORN has two words tuti and corn".islower())
```

```
True
False
```

- isupper() returns True if the case of all the characters is uppercase. Else Fale otherwise

In [77]:
```
### "TUTICORN HAS TWO WORDS TUTI AND CORN".isupper() returns True.
### "TUTICORN has two words tuti and corn".isupper() returns False
print("TUTICORN HAS TWO WORDS TUTI AND CORN".isupper())
print("TUTICORN has two words tuti and corn".isupper())
```

```
True
False
```

- strip() function returns the copy of the string with the leading characters removed.

In [78]:
```
#### strip() function removes the spaces around the string.
#### " TUTICORN ".strip() removes the spaces and return "TUTICORN"
print(" TUTICORN ".strip())
```

```
TUTICORN
```

- title() function returns the string with the first letter in each word is capitalized.

In [81]:
```
### title() function returns the capitalized word.
### "tuti corn".title() returns Tuti Corn
print("tuti corn".title())
```

```
Tuti Corn
```

Capturing Input from Command line using the input() function

In [92]:
```
### Pass name as input to capture the input from the command line
print(f'My name is', end=' ')
name = input()
```

```
My name is Jayant
```

In [93]:
```
### pass age as input to capture the input from the command line
print('Age is',end=' ')
age = input()
print(type(age))
```

```
Age is 5
<class 'str'>
```

In [95]:
```
###  pass price value as input to capture the input from the command line
print('Price is' , end=' ')
```

```
price = input()
price = float(price)
print(type(price))
```

```
Price is 20.0
<class 'float'>
```

In [96]:
```
### Exercise - Capture three inputs from
### the command line - name, addres and mobile and display the
### details to the screen.
```

In [99]:
```
name = input('Enter your name ')
address = input('Enter your address ')
mobile = input('Enter your mobile number')
print("name of the person is {}, address is {}, Mobile numer is {}".format(name,address,
```

```
Enter your namejayant
Enter your addressIndia
Enter your mobile number9776975664
name of the person is jayant, address is India, Mobile numer is 9776975664
```

Operators and Operands in python

- Arthimetic Operators
- Comprision Operators
- Boolean Operators
- Bitwise Operators
- Identity Operators
- Membership Operators

- Arthimetic Operators
  - Addition
  - Subtraction
  - Multiplication
  - Division
  - Modulo
  - exponentional

- Addition - Adds two numbers ('+')

In [102…
```
### Add two number using '+'
x = 10
y = 13
print(x+y)
```

```
23
```

- Subtraction - Subtract two numbers ('-')

In [103…
```
#### Subtract two number using '-'
x = 13
y = 10
print(x-y)
```

```
3
```

- Multiplication - multiply two numbers ('*')

In [104…
```
#### Multiply two numbers using '*'
x = 10
y = 5
print(x*y)
```

```
50
```

- Division - Division of two number ('/') or ('//')

In [105…
```
#### Divide two numbers using '/'
x = 15
y = 10
print(x/y)
print(x//y)
```

```
1.5
1
```

- Modulo - Modulo of two numbers (%)

In [106…
```
#### Find the remainder of two numbers using %
x = 20
y = 10
print(x%y)
```

```
0
```

In [107…
```
#### Find the remainder of two numbers using %
x = 30
y = 4
print(x%y)
```

```
2
```

- Exponential - exponent of two numbers (**)

In [109…
```
#### Find the exponent of a number - Base and exponent
x = 2
y = 3
print(2**3)
```

```
8
```

- Comprision Operators
  - Less than(<)
  - greater than (>)
  - less than or equal to (<=)
  - equal to (==)
  - greater than or equal to (>=)
  - Not equal to (!=)

In [110…
```
5 < 3
```

Out[110]:
```
False
```

In [111…
```
9 > 4
```

Out[111]:
```
True
```

In [112…
```
10 >= 10
```

Out[112]:
```
True
```

In [113…
```
11 >= 15
```

Out[113]:
```
False
```

In [114…
```
13 <= 14
```

Out[114]:
```
True
```

In [115…
```
12 == 14
```

Out[115]:
```
False
```

In [116…
```
15 == 16
```

Out[116]:
```
False
```

In [117…
```
12 == 12
```

Out[117]:
```
True
```

In [118…
```
23 != 24
```

Out[118]:
```
True
```

In [119…
```
23 != 23
```

Out[119]:
```
False
```

- Boolean Operators
  - and
  - or
  - not

In [121…
```
### Perform the operation as below for 'and' operator-
Check  (x > y) and (y > z) for output
x = 5
y = 2
z = 3
(x>y) and (y>z)
```

Out[121]:
```
False
```

In [122…
```
#### Perform the operation as below for 'or' operator
## - Check  (x > y) or (y > z) for output
x > y or z > y
```

Out[122]:
```
True
```

In [123…
```
#### Perform the operation as below for not operator
#### - Check x != y
print(x!=y)
```

```
True
```

- Bitwise Operators

In [128…
```
#### Bitwise opearators AND
### (0&1 gives 0, 1&0 give 0, 1&1 give 1,0&0 gives 0)
print(0&1)
```

```
0
```

In [127…
```
print(0&0)
```

```
0
```

In [129…
```
print(1&1)
```

```
1
```

In [130…
```
print(1&0)
```

```
0
```

Python-Lesson 03

In this notebook, we shall cover the following concepts

- Bitwise operators
- Membership operators
- Identity Operators
- Conditional Statements
- Python Data Structures
  - List
  - Tuple
  - Sets
  - Dictionary
  - Collections in Python

- Bitwise Operators
- Membership Operators
- Identity Operators

- Bitwise Operators

- Bitwise AND (&) Operator

In [6]:
```
print(bin(3))
print(bin(5)) # 011 & 101 --> 1
```

```
0b11
0b101
```

In [7]:
```
3&5
```

Out[7]:
```
1
```

- Bitwise OR (|) Operator

In [ ]:
```
# 0011 | 0101 -> 0111
```

In [9]:
```
print(3|5)
```

```
7
```

- Bitwise XOR (^) Operator

In [1]:
```
# 1011 ^ 1100 -> 0111 (11^12 -> 7)
print(11^12)
```

```
7
```

- Bitwise << left shift operator - The left shift operator shifts the left bits operand towards left.
- 1100 << 2 means that the two zeros are appended at the right side. 110000 is the output

In [14]:
```
### input is 1100 --> 12 << 2
### output is 110000 (48) is the output
12 << 2
```

Out[14]:
```
48
```

- Bitwise >> right shift operator - The right shift operator shifts the right side bits. The rigt side bit are removed.
- 110011 >> 2 means that the last two bits are removed from the right side. 1100 is the output.

In [21]:
```
### 51 -> in binary is 110011
### and when last two are removed we get 1100 (12) as the output
51 >> 2
```

Out[21]:
```
12
```

- Bitwise one's complement operator (~) x -> -(x+1) . This gives the complement of the binary number
- ~10 (1010) -> -11

In [22]:
```
~10
```

Out[22]:
```
-11
```

In [24]:
```
~-10
```

Out[24]:
```
9
```

In [26]:
```
#### Exercise - check if a number is even using bitwise & operator
x = 11
x&1 == 1  # odd number
```

Out[26]:
```
True
```

In [28]:
```
y = 12
```

```
y&1 == 0 # even number
```

Out[28]:
```
True
```

- Membership Operators - in and not in

In [35]:
```
### We can check the if the values are
### available or not in the given string for operators
## a='hello' in b='hello can i know the time please ?' check a in b
a = 'hello'
b = 'hello can i know the time please ?'
a in b
```

Out[35]:
```
True
```

In [33]:
```
###  We can check the if the values
###  are available or not in the given string for operators
###  a='bell' in b='hello can i know the time please ?' check a not in b
a = 'bell'
b = 'hello can i know the time please ?'
a not in b
```

Out[33]:
```
True
```

In [34]:
```
### We can check the if the values are available
### or not in the given string for operators
## a='bell' in b='hello can i know the time please ?' check a  in b
a = 'bell'
b = 'hello can i know the time please ?'
a in b
```

Out[34]:
```
False
```

- Identity Operators - is and is not

In [36]:
```
### is or is not operators can be used to
### know if certain conditions are met are not
a = 21
b = 33
c = b
print(c is b)
```

Out[36]:
```
True
```

In [37]:
```
print(c is not b)
```

```
False
```

Python Conditional Statements

if expr: </br> statement </br> elif expr: </br> statement </br> elif expr: </br> statement </br> elif expr: </br> statement </br> : </br> : </br> : </br> else:
statement

In [58]:
```
## Simple condition for two numbers
x = 10
y = 20
if y > x:
```

```
    print('Y is greater')
else:
    print('X is greater')
```

```
Y is greater
```

In [43]:
```
### Enter a name to Capture input from the command line
### and write conditions for it.
name = input('Enter name ')
if name == 'Jayant' :
    print('Please start')
elif name == 'Vamshi':
    print('Please turn around')
elif name == 'Mayank':
    print('Please be seated')
else:
    print('Not in the name')
```

```
Enter name Hari
Not in the name
```

In [45]:
```
### Writing a if else block of code to capture
## the statements such that
## we print statements only for age group of 13
age = input('Enter age of participant')
age = int(age)
if age >= 13:
    print('You can start coding')
else:
    print('Please learn basics of scartch')
```

```
Enter age of participant15
You can start coding
```

Hacker Rank Questions

Hacker Rank Question Given an integer, perform the following conditional actions: If n is odd, print Weird If n is even and in the inclusive range of 2 to 5, print Not Weird If n is even and in the inclusive range of to 6 to 20, print Weird If n is even and greater than 20 , print Not Weird n >= 1 and n <= 100

In [ ]:
```
## Summary of above n --> weird ( odd , even and 6 to 20)
# Not weird (even and 2to 5 and > 20)
n = input('Enter a number 1 to 100 ')
n = int(n)
if n % 2 != 0 and n <= 20:
    print('Weird')
elif n%2 == 0 and 2<=n <= 5:
    print('Not weird')
elif n%2 == 0 and 6<=n<=20:
    print('Weird')
else:
    print('Not weird')
```

Loops in Python

- while loop in python

In [35]:
```
### while loops in python
x = 0
while x < 7:
    print(x)
    x += 1
```

```
0
```

```
1
2
3
4
5
6
```

In [36]:
```
### while loops in python
x = 5
while x > 0:
    print(x)
    x -= 1
```

```
5
4
3
2
1
```

- for loop in python

- Range Object - A Range

If we want to generate numbers from 0 to n - then we can use range()

- Generate numbers using range object
- range()- A range object generates a sequence of numbers starting from 0 to n

In [41]:
```
#for loops in python using range object
# - Starting from 0 and ending until 3
for i in range(3):
    print(i)
```

```
0
1
2
```

In [42]:
```
# for loops in python using range object
# Starting from a number 2 and ending until 10.
for i in range(2,10):
    print(i)
```

```
2
3
4
5
6
7
8
9
```

In [43]:
```
# for loops in python using range object
# Staring from a number and ending at a number with difference
for i in range(2,14,2):
    print(i)
```

```
2
4
6
8
10
12
```

Python Data Structures

- List
- Tuple
- Sets
- Dictioary
- Collections in Python

- List - List of comma separated values (items) between square brackets </br>
  - Lists can be heterogenous - can take different values. </br>
  - Lists can be accessed based on indices. </br>

</br> | 0 | 1 | 2 | 3 | 4 | </br> | 1 | 2 | 3 | 4 | 5 | </br> | -5 | -4 | -3 | -2 | -1 | </br>

In [10]:
```
### Creating an empty list - []
numval = []
```

In [1]:
```
numbers = [1,2,3,4,5]
numbers
```

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

In [2]:
```
#### finding the numbers with zeroth index - numbers[0]
numbers[0]
```

Out[2]:
```
1
```

In [4]:
```
#### Finding the numbers from the last - numbers[-1]
numbers[-1]
```

Out[4]:
```
5
```

In [5]:
```
#### finding the number from first (0) to last- numbers[0:]
numbers[0:]
```

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

In [6]:
```
#### finding the numbers from first
#    index (1) to (4) index - numbers[1:4]
numbers[1:4]
```

Out[6]:
```
[2, 3, 4]
```

In [7]:
```
## finding the numbers from second index
## (2) to (5) index - numbers[2:5]
numbers[2:5]
```

Out[7]:
```
[3, 4, 5]
```

In [8]:
```
#### finding the numbers from last index
##   -3 to to the zeroth index -numbers[-3:]
numbers[-3:]
```

Out[8]:
```
[3, 4, 5]
```

- Print all numbers

In [9]:
```
### listing all the numbers from the list - numbers[:]
numbers[:]
```

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

- print length of numbers

In [11]:
```
### length of the list can be found using - len(numbers)
len(numbers)
```

Out[11]:
```
5
```

- Assign value to particular index

In [14]:
```
### We can reassign value to a index in a list
##   - assign a new number to index 0
numbers[0] = 11
```

In [15]:
```
numbers
```

Out[15]:
```
[11, 2, 3, 4, 5]
```

- append() elements to a list

In [16]:
```
## List values can be appended at the end
##  in a list - numbers.append() - numbers.append(6)
```

In [17]:
```
## List of number after the append - numbers
```

Out[17]:
```
[11, 2, 3, 4, 5, 6]
```

- To check the type of the list

In [18]:
```
## Check type of list
type(numbers)
```

Out[18]:
```
list
```

- Elements of list is heterogenous

In [19]:
```
#### Different types of values an be appended to a list
listValues = ['Apple','Banana',23,24,25,26]
listValues
```

Out[19]:
```
['Apple', 'Banana', 23, 24, 25, 26]
```

- Append elements to a list using append()

In [20]:
```
#### List can also be appended with 'Mango' at the list
listValues.append('Mango')
```

In [21]:
```
#### list value can also be appended with [33,45,56]
listValues.append([33,45,56])
```

In [22]:
```
listValues
```

Out[22]:
```
['Apple', 'Banana', 23, 24, 25, 26, 'Mango', [33, 45, 56]]
```

- Create list using the timing values - ['1:30','2:30','3:30','4:30','5:30','6:30','7:30']

In [30]:
```
timings = ['1:30','2:30','3:30','4:30','5:30','6:30','7:30']
timings[0:3]
```

Out[30]:
```
['1:30', '2:30', '3:30']
```

In [31]:
```
## Slice the list value from 2:5
timings[2:5]
```

Out[31]:
```
['3:30', '4:30', '5:30']
```

In [32]:
```
## Slice the list value from 4:
timings[4:]
```

Out[32]:
```
['5:30', '6:30', '7:30']
```

- reverse() a list

In [33]:
```
### To find the reverse of the list - list[::-1]
timings[::-1]
```

Out[33]:
```
['7:30', '6:30', '5:30', '4:30', '3:30', '2:30', '1:30']
```

In [46]:
```
### Alternatively to reverse the list- use the timings.reverse()
timings.reverse()
timings
```

Out[46]:
```
['1:30', '2:30', '3:30', '4:30', '5:30', '6:30', '7:30']
```

- index of a value in the list

In [47]:
```
#### To get the index of the given element - timings.index('2:30')
timings.index('2:30')
```

Out[47]:
```
1
```

- insert an element at a particular index

In [49]:
```
### Insert the value 8 at the index 2 -
```

In [50]:
```
timings.insert(2,8)
```

Out[50]:
```
['1:30', '2:30', 8, '3:30', '4:30', '5:30', '6:30', '7:30']
```

- remove a value from the list

In [51]:
```
### To remove an element from the list we can do this - timings
timings.remove(8)
```

```
timings
```

Out[51]:
```
['1:30', '2:30', '3:30', '4:30', '5:30', '6:30', '7:30']
```

- pop an element from the list

In [52]:
```
### to remove the top most element from the list - timings - timings.pop()
timings.pop()
```

Out[52]:
```
'7:30'
```

- sort() an element from the list

In [56]:
```
### using the sort() list can be sorted - timings.sort()
timings.sort()
```

In [57]:
```
timings
```

Out[57]:
```
['1:30', '2:30', '3:30', '4:30', '5:30', '6:30']
```

- extend an element from the list

In [58]:
```
## Extend the list by adding the elements
## Elements shall be appended at the end of the list
## extend - ['7:30','8:30','9:30']
timings.extend(['7:30','8:30','9:30'])
```

In [59]:
```
timings
```

Out[59]:
```
['1:30', '2:30', '3:30', '4:30', '5:30', '6:30', '7:30', '8:30', '9:30']
```

- Iterate a list

In [61]:
```
for timing in timings:
    print("The time now is {}".format(timing))
```

```
The time now is 1:30
The time now is 2:30
The time now is 3:30
The time now is 4:30
The time now is 5:30
The time now is 6:30
The time now is 7:30
The time now is 8:30
The time now is 9:30
```

- sum of elements in a list

In [3]:
```
### Lets build a list using 10,11,12,13,14,15,16
### The sum of all the elements in the list can be computed
### in two ways  - sum(intList) or using for loops

print('Sum of the list of elements',sum(intList))
print(intList)
```

```
Sum of the list of elements 91
[10, 11, 12, 13, 14, 15, 16]
```

- sum of the elements using for loops

In [4]:
```
intList = 10,11,12,13,14,15,16
slist = 0
for i in intList:
    slist += i
print(slist)
```

```
91
```

- for comprehensions

For comprehension is a simple way of returning the value based on certain input

In [62]:
```
##   Based on the range object
##   lets find the square of each number
y = [x**2 for x in range(5)]
```

In [63]:
```
y
```

Out[63]:
```
[0, 1, 4, 9, 16]
```

In [65]:
```
## We can also include condition
## within the for comrephension
y = [x**2 for x in range(10) if x%2 == 0 ]
print(y)
```

```
[0, 4, 16, 36, 64]
```

In [66]:
```
### return if only list contains letters starting with s
names = ['sunday','super','semi-conductor','non-conductor','mango','apple']
y = [x for x in names if x.startswith('s')]
```

In [67]:
```
y
```

Out[67]:
```
['sunday', 'super', 'semi-conductor']
```

Tuples

- Tuples are immutable objects that can be represented in ()
- Tuples can be accessed using indices
- Tuples are created by passing vlues to ()

In [76]:
```
tup1 = (25,'Mango','Rajesh',40,25,24,25)
tup1[1]
```

Out[76]:
```
'Mango'
```

In [77]:
```
tup1[3]
```

Out[77]:
```
40
```

In [78]:
```
### Tuples can be iterated over using for loop
for x in tup1:
    print(x)
```

```
25
Mango
```

```
Rajesh
40
25
24
25
```

In [75]:
```
## Assigning a value to tuple shows error -
## tuple object does not support assignment
tup1[0] = 35
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[75], line 2
      1 ## Assigning a value to tuple shows error - tuple object does not support assign
ment
----> 2 tup1[0] = 35

TypeError: 'tuple' object does not support item assignment
```

In [79]:
```
## Count function gives the count of the element in the tuple
tup1.count(25)
```

Out[79]:
```
3
```

In [68]:
```
## Exercise - There are two list [2,3,4] and [2,5,6]
## Get a combination (x,y)
## x is the element from first list,
## y is the element from second list -
## Get combinations (x,y) such that x is not equal to y

x = [2,3,4]
y = [2,5,6]
output = [(a,b) for a in x for b in y if x!=y]
print(output)
```

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

- **Sets in Python**
  - Sets contain unique elements
  - Sets are unordered sequence of elements
  - They are represented using the {} (curly braces)

In [80]:
```
elements = {'Mahesh','Mahesh','Suraj','Ravi','Kiran','Surya'}
elements
```

Out[80]:
```
{'Kiran', 'Mahesh', 'Ravi', 'Suraj', 'Surya'}
```

In [6]:
```
## Sets can be constructed using the following
## set keyword and passing a list to it
eles = set(['Mahesh','Mahesh','Suraj','Ravi','Kiran','Surya'])
eles
```

Out[6]:
```
{'Kiran', 'Mahesh', 'Ravi', 'Suraj', 'Surya'}
```

- Iteration over sets in python

In [83]:
```
elements
```

Out[83]:
```
{'Kiran', 'Mahesh', 'Ravi', 'Suraj', 'Surya'}
```

In [84]:
```
for x in elements:
    print(x)
```

```
Suraj
Surya
Ravi
Mahesh
Kiran
```

In [86]:
```
# The set elements are given below -
# We can represent the data as
# els = {'Mahesh','Mahesh','Suraj','Ravi','Kiran','Surya'}
els = {'Mahesh','Mahesh','Suraj','Ravi','Kiran','Surya'}
len(els)
```

Out[86]:
```
5
```

- Adding elements to set

In [87]:
```
### Add the following element to the set elements 'Mitra' using add function
els.add('Mitra')
```

In [88]:
```
els
```

Out[88]:
```
{'Kiran', 'Mahesh', 'Mitra', 'Ravi', 'Suraj', 'Surya'}
```

In [91]:
```
## Lets create one more set els2 = els2 = {'kiran','Mahesh', 'Mitra'}
els2 = {'kiran','Mahesh', 'Mitra'}
```

- seta.difference(setb) - Difference gives the elements that are in seta

In [2]:
```
els.difference(els2)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[2], line 1
----> 1 els.difference(els2)

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

- difference_update() - Removes the elements from the other set

In [98]:
```
els.difference_update(els2)
```

In [99]:
```
els
```

Out[99]:
```
{'Kiran', 'Ravi', 'Suraj', 'Surya'}
```

In [100…
```
els2
```

Out[100]:
```
{'Mahesh', 'Mitra', 'kiran'}
```

In [101…
```
## Create the set and try to perform union on els and els2
els = {'Mahesh','Mahesh','Suraj','Ravi','Kiran','Surya'}
len(els)
```

Out[101]:
```
5
```

- union combines both the sets with unique elements

In [103…
```
els.union(els2)
```

Out[103]:
```
{'Kiran', 'Mahesh', 'Mitra', 'Ravi', 'Suraj', 'Surya', 'kiran'}
```

- intersection provides the common elements between the sets

In [105…
```
els.intersection(els2)
```

Out[105]:
```
{'Mahesh'}
```

In [106…
```
print(els)
print(els2)
```

```
{'Suraj', 'Surya', 'Ravi', 'Mahesh', 'Kiran'}
{'Mitra', 'Mahesh', 'kiran'}
```

- Symmetric difference gives the elements that are not common as one single set.

In [108…
```
## els.symmetric_difference(els2) gives the common elements between both the sets.
els.symmetric_difference(els2)
```

Out[108]:
```
{'Kiran', 'Mitra', 'Ravi', 'Suraj', 'Surya', 'kiran'}
```

- Comprehension for set - Similar to for comprehension for Set
  - return type is Set when set comprehension is perfromed

In [8]:
```
listVal = {'Mahesh','Mahesh','Suraj','Ravi','Kiran','Surya'}
uniEle = {x for x in listVal}
uniEle
```

Out[8]:
```
{'Kiran', 'Mahesh', 'Ravi', 'Suraj', 'Surya'}
```

- **Dictionary in Python** - Dictionaries in Python are basically key,value pairs.
  - In a dictionary, key can be any unique object
  - Keys are unique and non-duplicated
  - Values can be duplicated
  - Dictionaries can be constructed using the dict() or {:}

- Create Dictionary using the dict() keyword

In [13]:
```
idName = dict()
idName[101] = 'Suraj'
idName[102] = 'Mahesh'
idName[103] = 'Surya'
idName[104] = 'Ravi'
idName[105] = 'Kiran'
idName[106] = 'Mitra'
idName
```

Out[13]:
```
{101: 'Suraj',
 102: 'Mahesh',
 103: 'Surya',
 104: 'Ravi',
```

```
 105: 'Kiran',
 106: 'Mitra'}
```

In [37]:
```
### Another way of creating the dictionary is the following
idName = {101: 'Suraj',102: 'Mahesh',103: 'Surya',
          104: 'Ravi', 105: 'Kiran', 106: 'Mitra'}
idName
```

Out[37]:
```
{101: 'Suraj',
 102: 'Mahesh',
 103: 'Surya',
 104: 'Ravi',
 105: 'Kiran',
 106: 'Mitra'}
```

- Retrieve the value of dictionary using key

In [17]:
```
### The dictionary name and the id can be passed to get the value - idName[101]
```

- Check if the key is in the dictionary

In [21]:
```
### Check if a particular key is available in the dictionary - 101 in idName
101 in idName
```

Out[21]:
```
True
```

In [24]:
```
### CHeck if a particular key is not available in the dictionary
#   - 109 not in idName
print(109 not in idName)
print(110 in idName)
```

```
True
False
```

- iterate over the dictionary using keys()

In [34]:
```
#  We can iterate over the dictionary using the keys()
#  function on the dictionary.
for k in idName.keys():
    print(f'Key is {k}')
```

```
Key is 101
Key is 102
Key is 103
Key is 104
Key is 105
Key is 106
```

- Values for the dictionary using values()

In [36]:
```
for v in idName.values():
    print(f'value is {v}')
```

```
value is Suraj
value is Mahesh
value is Surya
value is Ravi
value is Kiran
value is Mitra
```

- iterate over a dictionary using items()

In [33]:
```
for key,value in idName.items():
    print(f'Id is {key}, Name is {value}')
```

```
Id is 101, Name is Suraj
Id is 102, Name is Mahesh
Id is 103, Name is Surya
Id is 104, Name is Ravi
Id is 105, Name is Kiran
Id is 106, Name is Mitra
```

- Iterate over a dictionary using the dictionary object

In [39]:
```
### Using the dictionary object only the dictionary is used
for keyVal in idName:
    print(keyVal)
```

```
101
102
103
104
105
106
```

- Delete a key in the dictionary using the del keyword

In [40]:
```
## Delete the key value pair using the idName[key]
del idName[101]
```

In [41]:
```
idName
```

Out[41]:
```
{102: 'Mahesh', 103: 'Surya', 104: 'Ravi', 105: 'Kiran', 106: 'Mitra'}
```

- To get a value from the dictionar using the get() function or the []

In [42]:
```
idName[102]
```

Out[42]:
```
'Mahesh'
```

In [43]:
```
idName.get(102)
```

Out[43]:
```
'Mahesh'
```

- If the key is not in the dictionary using the get() method we can use the default value

In [49]:
```
#   Lets say we dont have the key/value pair in the dictionary
#   it returns a key erro
idName[109]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[49], line 2
      1 ### Lets say we dont have the key/value pair in the dictionary
----> 2 idName[109]

KeyError: 109
```

In [51]:
```
# If the key is available then it returns the value
idName.get(102,"Value is Not there")
```

Out[51]:
```
'Mahesh'
```

In [46]:
```
idName.get(109,"Key/Value is Not there")
```

Out[46]:
```
'Key/Value is Not there'
```

In [55]:
```
# To get the keys in the reversed order we can
# use reversed on the dictionary
```

In [54]:
```
for i in reversed(idName):
    print(i)
```

```
106
105
104
103
102
```

In [58]:
```
## Collecting the dictionary keys as list - list(idName)
list(idName)
```

Out[58]:
```
[102, 103, 104, 105, 106]
```

- To remove one key,value pair from the dictionary - using popitem()

In [59]:
```
## idName.popitem() will remove one key value pair from the dictionary
idName.popitem()
```

Out[59]:
```
(106, 'Mitra')
```

- To remove one key at a time from the dictionary - using pop()

In [64]:
```
# idName.pop(key) will pop one key value pair
#  for the given key from the dictionary
idName.pop(104)
```

Out[64]:
```
'Ravi'
```

In [65]:
```
# If the key is not available in the dictionary,
# pop() shall given an error on the dictionary
idName.pop(105)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[65], line 2
      1 ### If the key is not available in the dictionary, pop() shall given an error on
 the dictionary
----> 2 idName.pop(105)

KeyError: 105
```

- length of the dictionary - len() function

In [67]:
```
# To get the length of the dictionary
len(idName)
```

Out[67]: 2

- Lets say we want to get the key,value pairs using the dictionary using dictionary comprehension

In [71]:
```
### Dictionary comprehension can be perfromed using the following:
idName = idName = {101: 'Suraj',102: 'Mahesh',103: 'Surya',
                   104: 'Ravi', 105: 'Kiran', 106: 'Mitra'}

# get only the even ids
evenIdName = {k:v for (k,v) in idName.items() if k%2 ==0 }
evenIdName
```

Out[71]:
```
{102: 'Mahesh', 104: 'Ravi', 106: 'Mitra'}
```

Range Function - range()

- Range Function is a built function
- Range function can be used for creating sequence of integer values
- The Range function can be iterated over
- It can also be used to create other data structures like List and Tuples

In [1]:
```
range(10)
```

Out[1]:
```
range(0, 10)
```

- Range function can be iterated using the for loop
- In the code below, we shall user the 'for' keyword and the 'in' keyword and iterate over the range by passing a value. Alos please note that within the 'print' function we can use the 'end' to iterate over the value

In [3]:
```
# Example  - Iterating over the range function
for i in range(10):
    print(i, end=' ')  # Here, in the print function we can use the end as the
                       #separator to print the output in one single line
```

```
0 1 2 3 4 5 6 7 8 9
```

- Range function has other paramenters - we can understand the parameters as something below : startIndex, endIndex and stepSize </br> Let us look at an example below - </br> The start value/index is given as 2 </br> The end value/index is given as 10 </br> the step/value index is given as 5 </br>

In [2]:
```
# start - 4
# end - 18
# step - 2

rng1 = range(4,18,2)# Every alternate number
                    # would be printed with the difference in the numbers as 2
for i in rng1:
    print(i , end=' ')
```

```
4 6 8 10 12 14 16
```

- Creating a range and adding it to list

- In this method we shall create a list outside the range object and then iterate over the range and then add it to list

In [3]:
```
l1 = []
r1 = range(4,18,2)
for i in r1:
    l1.append(i)
print(l1)
print(r1)
```

```
[4, 6, 8, 10, 12, 14, 16]
range(4, 18, 2)
```

- Creating the range object and adding it to the list object using list()

In [4]:
```
r1 = range(4,18,2)
l2  = list(r1)
```

In [5]:
```
l2
```

Out[5]:
```
[4, 6, 8, 10, 12, 14, 16]
```

Lists

- Lists are the objects that are ordered sequentially
- Lists start from index 0
- Lists are iterable objects
- Lists can be iterated over using the for loop
- Lists can be sliced that is they can be indexed over
- Lists can be iterated
- Lists are mutable
- Addition and deletion of elements/values are possible in List.
- We represent lists using the [] - Square brackets.

In [6]:
```
r3 = range(4,18,2)
l3 = list(r3)
```

In [7]:
```
l3
```

Out[7]:
```
[4, 6, 8, 10, 12, 14, 16]
```

- Iterating over lists
  - Using for loop we can iterate over the list We create a range object and then create a list object. We iterate over the list object using the for loop

In [10]:
```
r3 = range(4,18,2)
l3 = list(r3)
for i in l3 :
    if i > 34:
        break
    doublei = i*2
    if doublei not in l3:
```

```
        l3.append(doublei)
    print(i, end=' ')
```

```
4 6 8 10 12 14 16 20 24 28 32
```

- Slicing the list - we use the ':' and '[]' to slice the list objects

- In the example below, we shall build the list using the range object

In [11]:
```
slicList = list(range(15))
slicList[:]
```

Out[11]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

- Slice the l4 list object using the indices from 0 to 5

In [12]:
```
slicList[0:5]
```

Out[12]:
```
[0, 1, 2, 3, 4]
```

- To reveres the list objects in the following format -
  - This will reverse the elements in the reverse format alternate numbers

In [18]:
```
slicList[::-1]
```

Out[18]:
```
[14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

In [19]:
```
slicList[::-2]
```

Out[19]:
```
[14, 12, 10, 8, 6, 4, 2, 0]
```

- Can also add step size for the list
  - Step value is how much needs to be added to the previous element to get the next elements in the list

In [22]:
```
slicList[1:7:2]
```

Out[22]:
```
[1, 3, 5]
```

- Slicing of String values in the list

In [13]:
```
names = ['Abhilash','Bharat','Chandra','Dinesh','Elangovan','Freddy','Girish']
names[:]
```

Out[13]:
```
['Abhilash', 'Bharat', 'Chandra', 'Dinesh', 'Elangovan', 'Freddy', 'Girish']
```

- using the index on the names list - to get the values from 1st index and 2nd index

In [26]:
```
names[1:3]
```

Out[26]:
```
['Bharat', 'Chandra']
```

In [27]:
```
names[1:5]
```

Out[27]:
```
['Bharat', 'Chandra', 'Dinesh', 'Elangovan']
```

- Reversing the list

In [28]:
```
names[::-1]
```

Out[28]:
```
['Girish', 'Freddy', 'Elangovan', 'Dinesh', 'Chandra', 'Bharat', 'Abhilash']
```

- Check the list now and we shall remove and retrieve the elements using pop
  - pop() returns the element and removes the element from the list

In [29]:
```
names.pop()
```

Out[29]:
```
'Girish'
```

In [32]:
```
# As we see in the list below once 'girish' is popped,
# there is no 'girish' element at the end.
names
```

Out[32]:
```
['Abhilash', 'Bharat', 'Chandra', 'Dinesh', 'Elangovan', 'Freddy']
```

- find the length of the list using the len() function

In [33]:
```
len(names)
```

Out[33]:
```
6
```

- remove() - Using the remove function on the list removes the element from the list
  - Lets say if we had two values then it removes only first occurrence from the list

In [34]:
```
names.remove('Freddy')
```

In [35]:
```
# Check the names function and we see that
#  Freddy is no longer available in the names list
names
```

Out[35]:
```
['Abhilash', 'Bharat', 'Chandra', 'Dinesh', 'Elangovan']
```

- Let us add two values to the list and then apply remove -
  - Farhan is added twice using the append() funciton

In [36]:
```
names.append('Farhan')
names.append('Farhan')
```

In [38]:
```
## As we see there are two values with the name 'Farhan'
names
```

Out[38]:
```
['Abhilash', 'Bharat', 'Chandra', 'Dinesh', 'Elangovan', 'Farhan', 'Farhan']
```

- use remove() on the names list and check the output

In [39]:
```
names.remove('Farhan')
names ## There is only one value of 'Farhan' in the list
```

Out[39]:
```
['Abhilash', 'Bharat', 'Chandra', 'Dinesh', 'Elangovan', 'Farhan']
```

- extend() can add multiple elements to the list

In [14]:
```
names.extend(['Giridhar','Hema','Jishnu'])
```

In [15]:
```
## Display the elements with the latest names added to the list
names
```

Out[15]:
```
['Abhilash',
 'Bharat',
 'Chandra',
 'Dinesh',
 'Elangovan',
 'Freddy',
 'Girish',
 'Giridhar',
 'Hema',
 'Jishnu']
```

- Reverse the original list using reverse() function on the list
  - reverse() function reverses the list and generates the output

In [16]:
```
names.reverse()
```

In [17]:
```
names # The list has been reversed and stores the values
      # in the list in reverse order
```

Out[17]:
```
['Jishnu',
 'Hema',
 'Giridhar',
 'Girish',
 'Freddy',
 'Elangovan',
 'Dinesh',
 'Chandra',
 'Bharat',
 'Abhilash']
```

- clear() method is used to empty the lists

In [18]:
```
names.clear()
```

In [19]:
```
names ## In the above cell , clear() was used
      ## to clear the list and now we have no values in the list
```

Out[19]:
```
[]
```

- Iterating over the lists using for loop

In [50]:
```
### Lets us add the friends to the names list
```

In [20]:
```
names = ['Abhilash','Bharat','Chandra','Dinesh','Elangovan','Freddy','Girish']
```

In [21]:
```
for friend in names:
    print(friend, end =',') ## We can print the value as show in this
```

```
Abhilash,Bharat,Chandra,Dinesh,Elangovan,Freddy,Girish,
```

- Exercise - Add names from the input prompt and store the values in the list

In [3]:
```
frNames= []
while True:
    addName = input(' Would you like to add a name ? Type yes/y or no/n : ')
    addName = addName.lower()
    if addName == 'yes' or  addName == 'y' :
        name = input('Enter the name of the friend :  ')
        frNames.append(name)
    elif addName == 'no' or addName == 'n':
        print('No more names are added -- Good bye !!')
        print('The friends list are {}'.format(frNames))
        break
    else:
        print('Wrong Input -- Can accept only yes/y or no/n .. Exiting....')
        break
```

```
 Would you like to add a name ? Type yes/y or no/n : What
Wrong Input -- Can accept only yes/y or no/n .. Exiting....
```

- Elements of the list can be sorted using sort() function

In [58]:
```
frNames.sort()
```

In [60]:
```
frNames
```

Out[60]:
```
['Bindu', 'Chandra', 'abhishek']
```

- geting the index of a 'Chandra' from the list using index()

In [62]:
```
frNames.index('Chandra')
```

Out[62]:
```
1
```

- sum() function on the list object
- sum() function is used to calculate the sum of the iterable objects
- In the below example, we pass list to sum function
- len() function gives the length of the iterable or the list
- Average/mean can be computed using the sum() and len() function

In [3]:
```
## Lets build a list using 10,11,12,13,14,15,16 -
## The sum of all the elements in the list can be computed
## in two ways  - sum(intList) or using for loops
intList = [10,11,12,13,14,15,17]
print('Sum of the list of elements',sum(intList))
print(intList)
numOfEle = len(intList)
meanIntLIst = sum(intList)/numOfEle
print(meanIntLIst)
```

```
Sum of the list of elements 92
[10, 11, 12, 13, 14, 15, 17]
13.142857142857142
```

- sum of the elements using the for loop

In [64]:
```
intList = 10,11,12,13,14,15,16
slist = 0
for i in intList:
    slist += i
print(slist)
```

```
91
```

- list comprehensions are writing simple code in a single line

In [65]:
```
## Based on the range object lets find the square of each number
y = [x**2 for x in range(5)]
```

In [66]:
```
y
```

Out[66]:
```
[0, 1, 4, 9, 16]
```

- Condition in the List Comprehension

In [67]:
```
## We can also include condition within the for comrephension
y = [x**2 for x in range(10) if x%2 == 0 ]
print(y)
```

```
[0, 4, 16, 36, 64]
```

- Condition with the string values/list

In [6]:
```
### return if only list contains letters starting with s
names = ['sunday','super','semi-conductor','non-conductor','mango','apple']
y = [x for x in names if x.startswith('s')]
```

In [7]:
```
y
```

Out[7]:
```
['sunday', 'super', 'semi-conductor']
```

In [8]:
```
del y[0]
```

In [9]:
```
y
```

Out[9]:
```
['super', 'semi-conductor']
```

Tuples

- `Tuples` are immutable objects that can be represented in ()
  - We cannot add,modify or remove elements from Tuples
- Tuples can be accessed using indices
- Tuples are created by passing values to ()
- Tuples can iterated using for loop
- Tuple values can be assigned to multiple variables in a single line

In [12]:
```
tup1 = (25,'Mango','Rajesh',40,25,24,25)
tup1[1]
```

Out[12]:
```
'Mango'
```

In [13]:
```
tup1[3]
```

Out[13]:
```
40
```

In [72]:
```
### Tuples can be iterated over using for loop
for x in tup1:
    print(x)
```

```
25
Mango
Rajesh
40
25
24
25
```

In [73]:
```
#  Assigning a value to tuple shows error
#  - tuple object does not support assignment
tup1[0] = 35
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[73], line 2
      1 ## Assigning a value to tuple shows error - tuple object does not support assign
ment
----> 2 tup1[0] = 35

TypeError: 'tuple' object does not support item assignment
```

In [14]:
```
## Count function gives the count of the element in the tuple
tup1.count(25)
```

Out[14]:
```
3
```

- A negative index starts from the end of the tuple. It starts at -1

In [15]:
```
tup1[-1]
```

Out[15]:
```
25
```

- len() function on the tuple. when applied on tuple gives the length of the tuple

In [78]:
```
len(tup1)
```

Out[78]:
```
7
```

- Accessing last element using the len() function

In [79]:
```
tup1[len(tup1) - 1]
```

Out[79]:
```
25
```

- As the tuples are immutable we cannot add, modify , remove or replace the elements in tuples
- If the tuples contain mutable datastructures however we can modify the elements within them </br>
  Let us look at the example below for more details

In [80]:
```
names = (['Rajesh','Mukesh'],['Sangeeta','Rithika'])
```

In [84]:
```
names[0]
```

Out[84]:
```
['Rajesh', 'Mukesh']
```

In [85]:
```
names[1]
```

Out[85]:
```
['Sangeeta', 'Rithika']
```

In [82]:
```
# This kind of direct assignment to the tuple is not possible ,
# where we are assigning the list elements directly to the tuples first value
names[0] = ['Abhishek','Brijesh']
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[82], line 3
      1 # This kind of direct assignment to the tuple is not possible ,
      2 # where we are assigning the list elements directly to the tuples first value
----> 3 names[0] = ['Abhishek','Brijesh']

TypeError: 'tuple' object does not support item assignment
```

- To access list inside tuple we shall use two indices like the ones used in matrices
- please note that the lists are mutable objects , whereas tuples are not mutable

In [87]:
```
## We access the inside elements using the syntax below
names[0][0] = 'Abhishek Bachan'
```

In [88]:
```
names
```

Out[88]:
```
(['Abhishek Bachan', 'Mukesh'], ['Sangeeta', 'Rithika'])
```

In [89]:
```
names[0][1] = 'Mukesh Khanna'
```

In [90]:
```
names
```

Out[90]:
```
(['Abhishek Bachan', 'Mukesh Khanna'], ['Sangeeta', 'Rithika'])
```

- Earlier we have seen the assignment of multilpe values in a single line
- Lets us assign multiple values to a single variable and see how the value is assigned

In [16]:
```
a,b = 10,20
```

In [17]:
```
val1 = a,b,a+b
```

In [18]:
```
## We see the value assigned is a tuple
val1
```

Out[18]:
```
(10, 20, 30)
```

- A tuple is returned from a funcion from where it is called if the function returns multiple values

In [94]:
```
def studentDetails():
    name = 'Ajay'
```

```
    subject = 'Python'
    course = 'BBA'
    return name,subject, course
```

In [95]:
```
details = studentDetails()
```

In [96]:
```
details
```

Out[96]:
```
('Ajay', 'Python', 'BBA')
```

- In order to hold values in different variables from a tuple

In [19]:
```
a,b ,c = (10,20,30)
```

In [20]:
```
print('Values are {} , {} , {}'.format(a,b,c))
```

```
Values are 10 , 20 , 30
```

- type and len functions on tuple

In [99]:
```
t1 = (10,'Chocolate',20,'Amul')
```

In [100…
```
t1
```

Out[100]:
```
(10, 'Chocolate', 20, 'Amul')
```

In [101…
```
type(t1)
```

Out[101]:
```
tuple
```

In [102…
```
len(t1)
```

Out[102]:
```
4
```

In [103…
```
type(t1),len(t1)
```

Out[103]:
```
(tuple, 4)
```

- Converting tuples to list and back to tuples

In [104…
```
t1
```

Out[104]:
```
(10, 'Chocolate', 20, 'Amul')
```

In [105…
```
l1 = list(t1)
```

In [106…
```
l1
```

Out[106]:
```
[10, 'Chocolate', 20, 'Amul']
```

In [107…
```
l1[1] = 'Butter Scotch'
```

In [108…
```
l1
```

Out[108]:
```
[10, 'Butter Scotch', 20, 'Amul']
```

In [109…
```
type(l1)
```

Out[109]:
```
list
```

- Converting the list back to tuple

In [110…
```
t2 = tuple(l1)
```

In [111…
```
t2
```

Out[111]:
```
(10, 'Butter Scotch', 20, 'Amul')
```

Exercise

In [1]:
```
## Exercise - There are two list [2,3,4] and [2,5,6]
## -Get a combination (x,y)
## x is the element from first list,
## y is the element from second list - Get combinations (x,y)
## such that x is not equal to y

x = [2,3,4]
y = [2,5,6]
output = [(a,b) for a in x for b in y if a!=b]
print(output)
```

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

- **Sets in Python**
  - Sets contain unique elements
  - Sets are unordered sequence of elements
  - They are represented using the {} (curly braces)

In [113…
```
elements = {'Mahesh','Mahesh','Suraj','Ravi','Kiran','Surya'}
elements
```

Out[113]:
```
{'Kiran', 'Mahesh', 'Ravi', 'Suraj', 'Surya'}
```

In [5]:
```
# Sets can be constructed using the following set
# keyword and passing a list to it
eles = set(['Z','A','A','Mahesh','Mahesh','Suraj','Ravi','Kiran','Surya'])
eles
```

Out[5]:
```
{'A', 'Kiran', 'Mahesh', 'Ravi', 'Suraj', 'Surya', 'Z'}
```

In [3]:
```
eles[0]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[3], line 1
----> 1 eles[0]

TypeError: 'set' object is not subscriptable
```

- Iteration over sets in python

In [115…
```
elements
```

Out[115]: {'Kiran', 'Mahesh', 'Ravi', 'Suraj', 'Surya'}

In [116…
```
for x in elements:
    print(x)
```

```
Kiran
Suraj
Mahesh
Ravi
Surya
```

In [9]:
```
## The set elements are given below - We can represent the data as
# els = {'Mahesh','Mahesh','Suraj','Ravi','Kiran','Surya'}
els = {'Mahesh','Mahesh','Suraj','Ravi','Kiran','Surya'}
len(els)
```

Out[9]: 5

In [10]:
```
### Add the following element to the set elements 'Mitra' using add function
els.add('Mitra')
```

In [11]:
```
els
```

Out[11]: {'Kiran', 'Mahesh', 'Mitra', 'Ravi', 'Suraj', 'Surya'}

In [7]:
```
## Lets create one more set els2 = els2 = {'kiran','Mahesh', 'Mitra'}
els2 = {'kiran','Mahesh', 'Mitra'}
```

- seta.difference(setb) - Difference gives the elements that are in sets

In [ ]:
```
{'Kiran', 'Mahesh', 'Mitra', 'Ravi', 'Suraj', 'Surya'}
{'kiran','Mahesh', 'Mitra'}
```

In [11]:
```
els.difference(els2)
```

Out[11]: {'Kiran', 'Ravi', 'Suraj', 'Surya'}

In [12]:
```
els2.difference(els)
```

Out[12]: {'kiran'}

- difference_update() - Removes the elements from the other set

In [12]:
```
els.difference_update(els2)
```

In [13]:
```
els
```

Out[13]: {'Kiran', 'Ravi', 'Suraj', 'Surya'}

In [14]:
```
els2
```

Out[14]: {'Mahesh', 'Mitra', 'kiran'}

In [18]:
```
## Create the set and try to perform union on els and els2
els = {'Mahesh','Mahesh','Mitra','Suraj','Ravi','Kiran','Surya'}
len(els)
```

Out[18]: 6

- union combines both the sets with unique elements

In [19]:
```
els.union(els2)
```

Out[19]:
```
{'Kiran', 'Mahesh', 'Mitra', 'Ravi', 'Suraj', 'Surya', 'kiran'}
```

- intersection provides the common elements between the sets

In [20]:
```
els.intersection(els2)
```

Out[20]:
```
{'Mahesh', 'Mitra'}
```

In [21]:
```
print(els)
print(els2)
```

```
{'Ravi', 'Kiran', 'Mahesh', 'Suraj', 'Mitra', 'Surya'}
{'kiran', 'Mahesh', 'Mitra'}
```

- Symmetric difference gives the elements that are not common as one single set.

In [22]:
```
## els.symmetric_difference(els2) gives the common elements between both the sets.
els.symmetric_difference(els2)
```

Out[22]:
```
{'Kiran', 'Ravi', 'Suraj', 'Surya', 'kiran'}
```

- Comprehension for set - Similar to for comprehension for Set
  - return type is Set when set comprehension is perfromed

In [13]:
```
listVal =  {'Mahesh','Mahesh','Suraj','Ravi','Kiran','Surya'}
uniEle = set()
for name in listVal:
    name = 'Mr ' + name
    uniEle.add(name)
print(uniEle)
```

```
{'Mr Mahesh', 'Mr Suraj', 'Mr Ravi', 'Mr Surya', 'Mr Kiran'}
```

In [14]:
```
listVal = {'Mahesh','Mahesh','Suraj','Ravi','Kiran','Surya'}
uniEle = {'Mr ' + x for x in listVal}
uniEle
```

Out[14]:
```
{'Mr Kiran', 'Mr Mahesh', 'Mr Ravi', 'Mr Suraj', 'Mr Surya'}
```

- list can be created using [] or list()
- tuple can be created using () or tuple()
- Dictionary can be created using {:} or dict()
- sets can be created using {} or set()

Why are we using data strcutures

- We want to store multiple values
- List : Ordered set of values and we want to iterate over the values
  `Index based accessing`

- Set : Store only unique elements then we shall go for set or we do not want to allow duplicate elements
- Tuple : We do not want to modify the values while storage and no modification of values
- Dictionary : We want identifiers for values or key/value pairs

- **Dictionary in Python** - Dictionaries in Python are basically key,value pairs.
  - In a dictionary, key can be any unique object
  - Keys are unique and non-duplicated
  - Values can be duplicated
  - Dictionaries can be constructed using the dict() or {:}

- Create Dictionary using the dict() keyword

In [16]:
```
idName = dict()
idName[101] = 'Suraj'
idName[102] = 'Mahesh'
idName[103] = 'Surya'
idName[104] = 'Ravi'
idName[105] = 'Kiran'
idName[106] = 'Mitra'
idName[107] = 'Mahesh B'
print(idName)
```

```
{101: 'Suraj', 102: 'Mahesh', 103: 'Surya', 104: 'Ravi', 105: 'Kiran', 106: 'Mitra', 10
7: 'Mahesh B'}
```

In [17]:
```
### Another way of creating the dictionary is the following
idName = {101: 'Suraj',102: 'Mahesh',103: 'Surya',
          104: 'Ravi', 105: 'Kiran', 106: 'Mitra',
          109:'Akhilesh'}
print(idName)
```

```
{101: 'Suraj', 102: 'Mahesh', 103: 'Surya', 104: 'Ravi', 105: 'Kiran', 106: 'Mitra', 10
9: 'Akhilesh'}
```

- Retrieve the value in the dictionary using key

In [32]:
```
### The dictionary name and the id can be passed to get the value - idName[101]
### dictionaryName[keyValue]
idName[101]
```

Out[32]:
```
'Suraj'
```

- Check if the key is in the dictionary

In [33]:
```
### Check if a particular key is available in the dictionary - 101 in idName
101 in idName
```

Out[33]:
```
True
```

In [18]:
```
### CHeck if a particular key is not available in the dictionary - 109 not in idName
print(109 not in idName)
print(110 in idName)
```

```
False
False
```

- iterate over the dictionary using keys()

In [36]:
```
idName.keys()
```

Out[36]:
```
dict_keys([101, 102, 103, 104, 105, 106, 109])
```

In [35]:
```
# We can iterate over the dictionary using the keys() function
# on the dictionary.
for k in idName.keys():
    print(f'Key is {k}')
```

```
Key is 101
Key is 102
Key is 103
Key is 104
Key is 105
Key is 106
Key is 109
```

- Values for the dictionary using values()

In [139…
```
for v in idName.values():
    print(f'value is {v}')
```

```
value is Suraj
value is Mahesh
value is Surya
value is Ravi
value is Kiran
value is Mitra
```

- iterate over a dictionary using items()

In [22]:
```
for key,value in idName.items():
    print(f'Id is {key}, Name is {value}')
    print('Id is {} , Name is {}'.format(key,value))
```

```
Id is 101, Name is Suraj
101 Suraj
Id is 102, Name is Mahesh
102 Mahesh
Id is 103, Name is Surya
103 Surya
Id is 104, Name is Ravi
104 Ravi
Id is 105, Name is Kiran
105 Kiran
Id is 106, Name is Mitra
106 Mitra
Id is 109, Name is Akhilesh
109 Akhilesh
```

- Iterate over a dictionary using the dictionary object

In [141…
```
### Using the dictionary object only the dictionary is used
for keyVal in idName:
    print(keyVal)
```

```
101
102
103
```

```
104
105
106
```

- Delete a key in the dictionary using the del keyword

In [142…
```
## Delete the key value pair using the idName[key]
del idName[101]
```

In [143…
```
idName
```

Out[143]:
```
{102: 'Mahesh', 103: 'Surya', 104: 'Ravi', 105: 'Kiran', 106: 'Mitra'}
```

- To get a value from the dictionar using the get() function or the []

In [144…
```
idName[102]
```

Out[144]:
```
'Mahesh'
```

In [145…
```
idName.get(102)
```

Out[145]:
```
'Mahesh'
```

- If the key is not in the dictionary using the get() method we can use the default value

In [37]:
```
# Lets say we dont have the key/value pair in the dictionary
# it returns a key erro
idName[110]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[37], line 2
      1 ### Lets say we dont have the key/value pair in the dictionary it returns a key
 erro
----> 2 idName[110]

KeyError: 110
```

In [147…
```
# If the key is available then it returns the value
idName.get(102,"Value is Not there")
```

Out[147]:
```
'Mahesh'
```

In [39]:
```
idName.get(110,"Key/Value is Not there")
```

Out[39]:
```
'Key/Value is Not there'
```

In [149…
```
# To get the keys in the reversed order
# we can use reversed on the dictionary
```

In [40]:
```
for i in reversed(idName):
    print(i)
```

```
109
106
105
104
103
```

```
102
101
```

In [151…
```
## Collecting the dictionary keys as list - list(idName)
list(idName)
```

Out[151]:
```
[102, 103, 104, 105, 106]
```

- To remove one key,value pair from the dictionary - using popitem()

In [41]:
```
## idName.popitem() will remove one key value pair from the dictionary
idName.popitem()
```

Out[41]:
```
(109, 'Akhilesh')
```

- To remove one key at a time from the dictionary - using pop()

In [42]:
```
# idName.pop(key) will pop one key value pair for
# the given key from the dictionary
idName.pop(104)
```

Out[42]:
```
'Ravi'
```

In [44]:
```
# If the key is not available in the dictionary,
# pop() shall given an error on the dictionary
idName.pop(105)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[44], line 2
      1 ### If the key is not available in the dictionary, pop() shall given an error on
 the dictionary
----> 2 idName.pop(105)

KeyError: 105
```

In [43]:
```
# If the key is not available in the dictionary,
# pop() shall given an error on the dictionary
idName.pop(105)
```

Out[43]:
```
'Kiran'
```

- length of the dictionary - len() function

In [156…
```
# To get the length of the dictionary
len(idName)
```

Out[156]:
```
2
```

- Lets say we want to get the key,value pairs using the dictionary using dictionary comprehension

In [157…
```
### Dictionary comprehension can be perfromed using the following:
idName = {101: 'Suraj',102: 'Mahesh',103: 'Surya', 104: 'Ravi',
          105: 'Kiran', 106: 'Mitra'}
# get only the even ids
evenIdName = {k:v for (k,v) in idName.items() if k%2 ==0 }
evenIdName
```

Out[157]: {102: 'Mahesh', 104: 'Ravi', 106: 'Mitra'}