(Advanced Python Workshop)

Day-1

Python

- created by Guido van Rossum.
- released in 1991.
- popular general-purpose interpreted, interactive, object-oriented, and high-level programming language.
- used to develop websites, software components, and applications or to work with Data Science, AI, and ML technologies.

Careers with Python

- Game developer
- Web designer
- Python developer
- Full-stack developer
- Machine learning engineer
- Data scientist
- Data analyst
- Data engineer
- DevOps engineer
- Software engineer
- Many more other roles

Characteristics of Python

- Open source.
- It supports functional and structured programming methods as well as OOP.

- It provides very high-level dynamic data types and supports dynamic type checking. (means unlike C or Java you need not to declare a variable).

[5]:
```
a = 5
b = 7
sum = a + b
print(sum)
```

```
12
```

- Python is Interpreted − Python is processed at runtime by the interpreter. So code will be translated line by line. See code below:

[4]:
```
print(3+4)
print(3+a)
```

```
7
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[4], line 2
      1 print(3+4)
----> 2 print(3+a)

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

- Database Connectivity.
- cross-platform language. - can be used on various operating system platforms such as Windows, Linux, Mac OS, Android OS.

Python Popular Editors:

- Google colab,
- Jupyter Notebook,
- VS code,
- IDLE,
- Spyder,
- PyCharm etc

```
[1]: print("Hello, World!")

     Hello, World!
```

Current version: The latest release of Python is 3.x

```
[2]: !python --version

     Python 3.11.7
```

```
[3]: import sys

     print(sys.version)

     3.11.7 | packaged by Anaconda, Inc. | (main, Dec 15 2023, 18:05:47)
```

Or, On terminal simply write:

```
PS C:\PyBatchApp\Project1> Python --version
Python 3.11.8
```

- File Extension for Python file: FileName.py
- File Extension for Python jupyter notebook Notebook: FileName.ipynb
- import will be used to import modules. In <import sys>above sys is a module.

Program 1: Find sum of three numbers.

[6]:
```
a = 5
b = 7
c = 15
sum = a + b +c
print(sum)
```

```
27
```

Program 2: Find sum of three numbers. Input must be taken at run time.

[7]:
```
a = int(input("Enter value for a"))
b = int(input("Enter value for b"))
c = int(input("Enter value for c"))
sum = a + b +c
print(sum)
```

```
Enter value for a 30
Enter value for b 25
Enter value for c 42
97
```

Comment in python

1. Single line comment- using #

[9]:
```
# This is single line comment in Python
```

2. Multi-line comment using **docstring**- triple quotes(“””)

[10]:
```
""" This is multi-line comment
This can be used to give proper
explanation of module or methods"""
```

[10]:
```
' This is multi-line comment\nThis can be used to give proper \nexplanation of module or methods'
```

The docstring can also be printed using .__**doc**__.

[11]:
```
def multiply(a, b):
    """Multiplies the value of a and b"""
    return a*b

# Print the docstring of multiply function
print(multiply.__doc__)
```

```
Multiplies the value of a and b
```

Difference between Comments and Docstring in Python

Comments	Docstrings
They are declared using #	They are declared using “”” “””
Used to increase the readability of the code	Gives a description of the Python modules, functions, and classes
They cannot be accessed	They can be accessed using __doc__

Different Ways to Print in Python

1. **Simple print function**

[13]:
```
#print variable
s = "Hello"
print(s)

x = 5
print(x)
```

```
Hello
5
```

2. **Print with string formatting (%)**

[14]:
```
name = "Alice"
age = 25
print("%s is %d years old." % (name, age))
```

```
Alice is 25 years old.
```

3. Print with the ‘str.format()’ method

[15]:
```
# Example 1: Printing a formatted string using 'str.format()'
name = "Bob"
age = 30
print("{} is {} years old.".format(name, age))
```

```
Bob is 30 years old.
```

[16]:
```
# Example 2: Printing a formatted string with named arguments
product = "Python Course"
price = 99.99
print("{name} costs ${price:.2f}.".format(name=product, price=price))
```

```
Python Course costs $99.99.
```

4. Print with the ‘f-string’ syntax

[17]:
```
name = "Carol"
age = 35
print(f"{name} is {age} years old.")
# Output: Carol is 35 years old.
```

```
Carol is 35 years old.
```

5. Print to a file or a stream

[18]:
```
with open('output.txt', 'w') as f:
    # Writing to the file using the print function
    print("This line will be written to a file.", file=f)
```

Here, we open a file named ‘output.txt’ in write mode using the ‘open’ function and the ‘with’ statement. Then, we pass the ‘file’ argument to the ‘print’ function, which writes the output to the file.

sep and end in print()

The end parameter is used to specify a string that will be printed after the output. However, the sep parameter is used as a separator between the items that you want to print.

end	It specifies the string to use as a terminator after the objects are printed. The default value is '\n'.
sep	Specifies the string to use as a separator between the objects being printed. The default value is ' '.

```
[39]: print('India', end = ',')
      India,
[40]: print("Namaste", "India", sep=", ")
      Namaste, India
```

Elements of Python

- Keywords and Identifiers
- Variables
- Data types and type conversion
- Operators in Python
- Expressions in Python

Keywords

- Keywords are the reserved words in Python.
- We cannot use a keyword as a variable name, function name or any other identifier. They are used to define the syntax and structure of the Python language.
- In Python, keywords are case sensitive.

- All the keywords except True, False and None are in lowercase and they must be written as they are.
- Example:
  and, as, assert, await, break, class, continue, def, del, elif, else, except, finally, for, global, if, import, in, is, lambda, not, or, pass, raise, return, try, while, with, yield

Identifier

- An identifier is a name given to entities like objects, class, functions, variables, etc.
- It helps to differentiate one entity from another.
- Rules for writing identifiers:
- Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore _.
- An identifier cannot start with a digit.
- Keywords cannot be used as identifiers.
- Cannot use special symbols like !, @, #, $, % etc. in identifier.
- An identifier can be of any length.

Variable

- A variable is a location in memory used to store some data (value).
- Unique names are given to them to differentiate between different memory locations.
- No need to declare a variable before using it. The declaration happens automatically when a value is assigned to a variable.
- The equal sign (=) is used to assign values to variables.
- The operand to the left of the = (assignment) operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable.
- For example

```
counter = 100          # An integer assignment
miles   = 1000.0       # A floating point
name    = “test"       # A string
```

Multiple Assignment in Variables

- A variable is a location in memory used to store some data (value).
- A single value may be assigned to several variables simultaneously.
  For example
  a = b = c = 1

- It also allows to assign multiple objects to multiple variables.
  For example
  a,b,c = 1,2,"Ram"

Data Types

- Every value in Python has a datatype, that are used to define the operations possible on them and the storage method for each of them.
- Since **everything is an object in Python** programming, data types are actually classes and variables are instance (object) of these classes.
- Python has following standard data types
  Numbers, Boolean, String, List, Tuple, Set, Dictionary

Number Type

The numbers can be integer, float or complex numbers.

Integer: Integer literals are created by any number without a decimal or complex component.
Float: Float literals can be created by adding a decimal component to a number.
Complex: Complex literals can be created by using the notation x+yj where x is the real component and y is the imaginary component.

Boolean Type

Boolean can be defined by typing **True/False** without quotes.

```
b1 = True
b2 = False
```

type() function

We can get the data type of a variable by using type() function.

[19]:
```
x = 5
y = "John"
print(type(x))
print(type(y))
```

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

Type Conversion

The process of converting the value of one data type (integer, string, float, etc.) to another data type is called type conversion.

Python has two types of type conversion.

- Implicit Type Conversion
- Explicit Type Conversion

Implicit Type Conversion

- Python automatically converts one data type to another data type without any user involvement.
- It always converts smaller data types to larger data types to avoid the loss of data.

[21]:
```
a = 4
b = 2.3
c = a + b
print(c)
```

```
6.3
```

In above example, data type of a and b are int and float respectively. Data type of c will be float and its value is 6.3

Explicit Type Conversion

- In Explicit Type Conversion, users convert the data type of an object to required data type.

- The predefined functions like int(), float(), str() are used to perform explicit type conversion.
- This type of conversion is also called typecasting because the user casts/changes the data type of the objects.
- Syntax: <datatype>(expression)

[22]:
```
a = 2.6
c = int(a)
print(c)
```

```
2
```

Operators in Python

Python supports the following types of operators.

- Arithmetic Operators
- Relational Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Arithmetic Operators

Operator	Description	Example
		(a=5, b=3)
+	Adds values on either side of the operator.	a + b = 8
-	Subtracts right hand operand from left hand operand.	a – b = 2
*	Multiplies values on either side of the operator	a * b = 15

Operator	Description	Example
		(a=5, b=3)
+	Adds values on either side of the operator.	a + b = 8
-	Subtracts right hand operand from left hand operand.	a – b = 2
/	Divides left hand operand by right hand operand	a / b = 1.666
%	Divides left hand operand by right hand operand and returns remainder	a % b = 2
**	Performs exponential (power) calculation on operators	a**b =5^3= 125
//	The division of operands where the result is the quotient in which the digits after the decimal point are removed.	a//b=1

Relational Operators

Operator	Description	Example
		a=5, b=3
==	If the values of two operands are equal, then the condition becomes true	a == b is not true.
!=	If values of two operands are not equal, then the condition becomes true.	a!=b is true

>	If the value of the left operand is greater than the value of the right operand, then the condition becomes true.	a > b is true
<	If the value of left operand is less than the value of right operand, then condition becomes true.	a < b is not true
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	a >= b is true
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	a <= b is not true

Assignment Operators

Operator	Description
=	a = b Assigns values from right side operands(b) to left side operand (a)
+=	a+=b is same as a = a + b It adds right operand to the left operand and assign the result to left operand

-=	a-=b is same as a = a - b It subtracts right operand from the left operand and assign the result to left operand
*=	a*=b is same as a = a * b It multiplies right operand with the left operand and assign the result to left operand
/=	a/=b is same as a = a / b It divides left operand with the right operand and assign the result to left operand
%=	a%=b is same as a = a % b It takes modulus using two operands and assign the result to left operand
**=	a**=b is same as a = a ** b Performs exponential (power) calculation on operators and assign value to the left operand
//=	a//=b is same as a = a //b It performs floor division on operators and assign value to the left operand

Logical Operators

Operator	Description
and	True if both the operands are true

or	True if either of the operands is true
not	True if operand is false (complements the operand)

a	**b**	**a and b**	**a or b**	**not a**
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

Bitwise Operators

- Bitwise operators manipulates the data at bit level.
- These are applicable on integer values.
- Types of Bitwise Operators:
    - & (Bitwise and operator)
    - | (Bitwise or operator)
    - ^ (Bitwise XOR operator)
    - ~ (Bitwise one's complement operator)
    - << (Bitwise left-shift operator)
    - >> (Bitwise right-shift operator)

a	b	a & b	a | b	a ^ b	~a
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Operator	Let a = (92)10 =(0101 1100)2 and b = (14)10 = (0000 1110)2	
&	c = (a & b) = 92 & 14	0101 1100 & 0000 1110 ---------- 0000 1100 = $(12)_{10}$
|	c = (a | b) = 92 | 14	0101 1100 | 0000 1110 ---------- 0101 1110 = $(94)_{10}$
^	c = (a ^ b) = 92 | 14	0101 1100 ^ 0000 1110 ---------- 0101 0010 = $(82)_{10}$

~	c = ~a = ~92	c = ~(0101 1100) = 1010 0011
<<	c = a<< 1 = 92<< 1	C = 0101 1100 << 1 = 1011 1000 = $(184)_{10}$
>>	c = a >> 2 = 92>> 2	C = 0101 1100 >> 2 = 0001 0111 = $(23)_{10}$

Membership Operators

- **in** and **not in** are the membership operators in Python.
- They are used to test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary).
- In a dictionary, we can only test for presence of key, not the value.

Operator	**Description**	**Example** **x = {2,3,5}**	**Result**
in	True if value/variable is found in the sequence	5 in x	True
not in	True if value/variable is not found in the sequence	5 not in x	False

Identity Operators

- Identity operators compare the memory locations of two objects.
- **is** and **is not** are the identity operators in Python.
- They are used to check if two values (or variables) are located on the same part of the memory.
- Two variables that are equal does not imply that they are identical

Operator	Description	Example a = 'Hello' b = 'Hello'	Result
is	True if the operands are identical (refer to the same object)	a is b	True
is not	True if the operands are not identical (do not refer to the same object)	a is not b	False

[23]:
```
a = 5
b = 5
print(a is b)
```

```
True
```

[24]:
```
a = 5
b = 4
b = b + 1
print(a is b)
```

```
True
```

[25]:
```
a = 5
b = 5
b = b + 1
print(a is b)
```

```
False
```

[26]:
```
a = 5
b = '5'
print(a is b)
```

```
False
```

Conditional Statements

- if statement
- if else statement
- if elif statement
- Nested if else

if statement

Program:

[28]:
```
age = int(input("Enter your age in years: "))
if age >= 18:
    print("You can VOTE")
    print("Congrats!")
```

```
Enter your age in years:  21
You can VOTE
Congrats!
```

if else statement

Program: To print Pass or fail on the basis of marks obtained

[29]:
```
percentage = int(input("Enter your percentage: "))
if percentage > 33:
    print("PASS")
else:
    print("FAIL")
```

```
Enter your percentage:  67
PASS
```

if elif statement

Program: To print grade on the basis of marks obtained

[30]:
```
percentage = int(input("Enter your percentage: "))
if percentage >= 90:
    print("Grade A")
elif percentage >= 80:
    print("Grade B")
elif percentage >= 70:
    print("Grade C")
elif percentage >= 60:
    print("Grade D")
elif percentage >= 50:
    print("Grade E")
else:
    print("Fail")
```

```
Enter your percentage:  67
Grade D
```

Nested if else

Program: To find greatest among three numbers.

[31]:
```
a = int(input("Enter first Number"))
b = int(input("Enter Second Number"))
c = int(input("Enter third Number"))
if a > b:
    if a > c:
        print(a, "is greatest")
    else:
        print(b, "is greatest")
else:
    if b > c:
        print(b, "is greatest")
    else:
        print(c, "is greatest")
```

```
Enter first Number 34
Enter Second Number 56
Enter third Number 27
56 is greatest
```

Loops

Loop statements in Python programming language repeatedly execute a target statement as long as a given condition is true.

- while loop statement
- for in loop statement
- Nested loop statement

while loop

Syntax:

```
while condition:
    statements
```

Program: To find the sum of first n natural numbers.

[33]:
```
num = int(input("Enter a number: "))
sum = 0
while num > 0:
    sum = sum + num
    num = num - 1
print("The Sum is: ", sum)
```

```
Enter a number:  5
The Sum is:  15
```

for loop

Syntax

```
for value in sequence:
    Statements
```

- Here, value is the variable that takes the value of the item inside the sequence on each iteration.
- Loop continues until we reach the last item in the sequence.
- The body of the for loop is separated from the rest of the code using indentation.

Program: To find the sum of all numbers stored in a list.
(we will discuss list sooner)

[35]:
```
numbers = [10, 20, 5, 15, 25, 35, 30]
sum = 0
for val in numbers:
    sum = sum + val
print("The sum of numbers is:", sum)
```

```
The sum of numbers is: 140
```

range() function

- We can generate a sequence of numbers using range() function.
- range(10) will generate numbers from 0 to 9 (10 numbers).

[42]:
```
for i in range(10):
    print(i, end = ',')
```

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

- range() function returns the list, all the operations that can be applied on the list can be used on it.
- We can also define the start, stop and step size as :

  range(start, stop, step_size).

  step_size defaults to 1 if not provided.

[43]:
```
for i in range(1,10):
    print(i, end = ',')
```

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

[44]:
```
for i in range(1,10,2):
    print(i, end = ',')
```

```
1,3,5,7,9,
```

Nested Loops:

- one loop inside another loop.

Syntax:

```
for value1 in sequence:
    for value2 in sequence:
        statements(s)
    statements(s)
```

Program: To print pattern:

[47]:
```
for i in range(1, 5):
    for j in range(i):
        print(i, end=" ")
    print()
```

```
1
2 2
3 3 3
4 4 4 4
```

Program: To print all prime numbers from a list.

[48]:
```
n = [2, 4, 5, 6, 8, 11, 14, 25, 27, 31]
for x in n:
    i = 1;
    flag = 0
    while i <= x:
        if x % i == 0:
            flag = flag + 1
        i = i + 1;
    if flag == 2:
        print(x)
```

```
2
5
11
31
```

break statement

- The break statement is used to exit a for or a while loop.
- The purpose of this statement is to end the execution of the loop (for or while) immediately and the program control goes to the statement after the last statement of the loop.
- If there is an optional else statement in while or for loop it skips the optional clause also.

Example of break Statement

[49]:
```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]
num_sum = 0
count = 0
for x in numbers:
    num_sum = num_sum + x
    count = count + 1
    if count == 5:
        break
print("Sum of first ", count, "integers is: ", num_sum)
```

```
Sum of first  5 integers is:  15
```

continue statement

The continue statement is used in a while or for loop to take the control to the top of the loop without executing the rest statements inside the loop.

Example of continue Statement

[50]:
```
for x in range(7):
    if x == 3 or x == 6:
        continue
    print(x)
```

```
0
1
2
4
5
```

Pass Statement

- The pass statement is used as a placeholder for future code.
- When the pass statement is executed, nothing happens, but you avoid getting an error when empty code is not allowed.

- Empty code is not allowed in loops, function definitions, class definitions, or in if statements.

Example:

Example1:

```
def myfunction():
  pass
```

Ex 2:

```
class Person:
  pass
```

Ex 3:

```
a = 33
b = 200

if b > a:
  pass
```

String

- Strings in python are surrounded by either single quotation marks, or double quotation marks

```
name = 'Manmohan'   #Single quotes
city = "Noida"      #Double quotes
```

- Python does not have a character data type, a single character is simply a string with a length of 1.

```
grade = "A"
```

- Square brackets can be used to access elements of the string.

```
city = "Noida"
print(city[0])
```

```
N
```

Basic operations

- **Traversing:** A string can be traversed by accessing character(s) from one index to another.

```
city = "Noida"
for i in city:
    print(i)
```

```
N
o
i
d
a
```

- **Concatenating:** Concatenate means joining two or more strings. We use + operator to concatenate two given strings.

```
str1 = "Greater"
str2 = " "
str3 = "Noida"
print(str1+str2+str3)
```

```
Greater Noida
```

- **Appending:** Append means add something to the end of string. We use + = operator to add one string at the end of another string.

```
str1 = "Greater"
str2 = "Noida"
str1 += str2
print(str1)
```

```
GreaterNoida
```

- **Multiplying:** Means repeating a string n number of times

```
str1 = "Hello"
print(str1*5)
```

```
HelloHelloHelloHelloHello
```

String Slicing

- A substring of a string is called slice. The slice operation is used to refer to sub-parts of strings.

```
str = "PYTHON"
print("str [1:5] = ", str[1:5])
print("str [:6] = ", str[:6])  #defaults to start of string
print("str [1:] = ", str[1:]) #defaults to end of string
print("str [:] = ", str[:]) #defaults to entire string
print("str [1:20] = ", str[:]) #truncates to length of the string
```

```
str [1:5] =  YTHO
str [:6] =  PYTHON
str [1:] =  YTHON
str [:] =  PYTHON
str [1:20] =  PYTHON
```

- Use negative indexes to start the slice from the end of the string:

P	Y	T	H	O	N
0	1	2	3	4	5
-6	-5	-4	-3	-2	-1

```
str = "PYTHON"
print("str [-1] = ", str[-1])   #last character is accessed
print("str [-6] = ", str[-6])   #first character is accessed
print("str [-2:] = ", str[-2:]) #Second last and last characters
print("str [:-2] = ", str[:-2]) #All except last two
print("str [-5:-2] = ", str[-5:-2]) #from second upto second last
```

```
str [-1] =  N
str [-6] =  P
str [-2:] =  ON
str [:-2] =  PYTH
str [-5:-2] =  YTH
```

String Methods

Method	Description
lower()	Converts all uppercase characters in a string into lowercase characters and returns it. **Syntax:** str.lower( ) **Example:** *str1 = "NOIda"* *print(str1.lower())*
upper()	Converts all lowercase characters in a string into uppercase characters and returns it. **Syntax:** str.upper( ) **Example:** *str1 = "NOIda"* *print(str1.upper())*
title()	Convert the first character in each word to Uppercase and remaining characters to Lowercase. **Syntax:** str.title( ) **Example:** *str1 = "my NaME is moHIT"* *print(str1.title())*
capitalize()	Converts the first character of the string to a capital letter while making all other characters in the string lowercase letters. **Syntax:** str.capitalize( ) **Example:** *str1 = "my NaME is moHIT"* *print(str1.capitalize())*

casefold()	This method returns a string where all the characters are in lower case. **Syntax:** str.casefold( ) **Example:** *txt = "ß“* *print(txt.casefold())*
center()	This method creates and returns a new string that is padded with the specified character. **Syntax:** str.center(length, char) **Example:** *str1 = “Hello NIET”* *print(str1.center(20,'#'))*
count()	Returns the number of occurrences of a substring in the given string. **Syntax:** str.count(substring) **Example:** *str1 = “This is my first string”* *print(str1.count('i'))*
find()	Returns the lowest index of the substring if it is found in a given string. If it is not found then it returns -1. **Syntax:** str.find(string,start,end) **Example:** *str1 = "This is my first string"* *print(str1.find(“my”))*
format()	Formats specified values in a string *txt1 = "My name is {fname}, I'm {age}".format(fname = "John", age = 36)* *txt2 = "My name is {0}, I'm {1}".format("John",36)* *txt3 = "My name is {}, I'm {}".format("John",36)*
endswith()	Returns true if the string ends with the specified value **Syntax:** str.endswith(value, start, end) **Value:** Required. The value to check if the string ends with **Start:** Optional. An Integer specifying at which position to start the search **End:** Optional. An Integer specifying at which position to end the search
strip()	Removes any whitespace from the beginning or the end. **Syntax:** str.strip() **Example:** *str1 = “ This is my first string ”* *print(str1.strip())*

replace()	Replaces a string with another string. **Syntax:** str.replace(string1, string2) **Example:** *str1 = "This is my first string"* *print(str1.replace("first","new"))*
split()	Returns a list where the text between the specified separator becomes the list items. **Syntax:** str.split() **Example:** *str1 = "This is my first string"* *print(str1.split())*
index()	Returns the position of the first occurrence of a substring in a string. **Syntax:** str.index(string) **Example:** *str1 = "This is my first string"* *print(str1.index("first"))*
isdecimal()	Returns True if all characters in the string are decimals **Syntax:** str.isdecimal() **Example:** x = '\u00b2' print(x.isdecimal())
isdigit()	Returns True if all characters in the string are digits **Syntax:** str.isdigit() **Example:** print('\u00b2'.isdigit()) print('\u2153'.isdigit())
isidentifier()	Returns True if the string is an identifier **Syntax:** str.isidentifier()
islower()	Returns True if all characters in the string are lower case **Syntax:** str.islower()
isupper()	Returns True if all characters in the string are upper case **Syntax:** str.isupper()
isnumeric()	Returns True if all characters in the string are numeric **Example:** print('\u2153'.isnumeric())

List

- Lists are used to store multiple items in a single variable.
- Lists are similar to arrays, declared in other languages.
- List items are indexed, the first item has index [0], the second item has index [1] etc.
- List items are **ordered, changeable**, and **allow duplicate values**.
- Ordered means that the items have a defined order, and that order will not change.
- Changeable means that we can change, add, and remove items in a list after it has been created.
- Duplicate item means that since lists are indexed, lists can have items with the same value.

List Creation

- Lists in Python can be created by just placing the sequence inside the square brackets[ ]

```
a = [1,2,3,4,5,6]
print(a)
```

```
[1, 2, 3, 4, 5, 6]
```

- Creating a list by using list() constructor: It is also possible to use the list() constructor when creating a new list.

```
a = list(("Abhi", 10, 75))
print(a)
```

```
['Abhi', 10, 75]
```

- List items are indexed and can be accessed by referring to the index number.

```
a = [1,2,3,4,5,6]
print(a[2])
```

```
3
```

- List items can also be accessed by referring to the negative index number. -1 refers to the last item, -2 refers to the second last item etc.

```
list1 = [1,2,3,4,5,6]
print(list1[-2])
```

```
5
```

List Slicing

- We use [] operator also called slicing operator to take subset of a list from original list.
- Syntax: list_name[start:end:step-size]

```
list1 = [1, 2, 3, 4, 5, 6, 8, 9, 10]
list2 = list1[1:7:2]
print(list2)
```

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

Change List Items

```
list1 = ["apple", "banana", "mango"]
print("Before changing list is",list1)
list1[1] = "orange"
print("After changing list is",list1)
```

```
Before changing list is ['apple', 'banana', 'mango']
After changing list is ['apple', 'orange', 'mango']
```

Change a Range of Item Values

To change the value of items within a specific range, define a list with the new values, and refer to the range of index numbers where you want to insert the new values.

```
list1 = [10, 20, 30, 40, 50, 60, 70, 80]
print("Before changing list is",list1)
list1[1:5] = [1, 2, 3, 4]
print("After changing list is",list1)
```

```
Before changing list is [10, 20, 30, 40, 50, 60, 70, 80]
After changing list is [10, 1, 2, 3, 4, 60, 70, 80]
```

List Methods

Method	Description	Syntax	Example	output
append()	Appends an element to the list	list.append(obj)	list=[1,2,3,4,5] list.append(8) print(list)	[1,2,3,4,5,8]
count()	count number of times an element appears in the list	list.count(obj)	list=[1,2,3,4,5] print(list.count(3))	1

index()	Returns lowest index of element in list	list.index(obj)	list=[1,2,3,4,5] print(list.index(3))	2
insert()	Insert object at the specified index in the list	list.insert(pos, elmnt)	list=[1,2,3,4,5] list.insert(3, 10) print(list)	[1,2,3,10,4,5]
pop()	Removes element at specified index from list(by default last)	list.pop(index)	list=[1,2,3,4,5] print(list.pop()) print(list)	5 [1,2,3,4]
remove()	removes the elements in the list	list.remove(obj)	list=[1,2,3,4,5] list.remove(1) print(list)	[2,3,4,5]
reverse()	reverse the elements in the list	list.reverse()	list=[1,2,3,4,5] list.reverse()	[5, 4, 3, 2, 1]
sort()	sorts the elements in the list	list.sort(reverse=True/False)	list=[5,2,4,1,3] list.sort()	[1,2,3,4,5]
extend()	adds the elements in a list to the end of another list	list1.extend(list2)	list1=[1,2,3,4,5] list2=[6,7] list1.extend(list2) print(list1)	[1,2,3,4,5,6,7]
copy()	returns a copy of the specified list	x=list.copy()	list=[1,2,3,4,5] x=list.copy() print(x)	[1,2,3,4,5]
clear()	removes all the elements from a list	list.clear()	list=[1,2,3,4,5] list.clear() print(list)	[ ]

List Comprehension

List comprehension help programmers to create list in a concise way.
This is mainly beneficial to make new lists where each element is obtained by applying some operations to each member of another sequence or iterable.

syntax:
List =[expression for variable in sequence]

Example: Program to make a list of cubes using List comprehension

```
cubes = [i**3 for i in range(11)]
print(cubes)
```

```
[0, 1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

Example: Program to find even numbers from a list using List comprehension

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_numbers = [x for x in numbers if x % 2 == 0]
print(even_numbers)
```

```
[2, 4, 6, 8, 10]
```

Example: Program for converting a list of strings to uppercase using List comprehension

```
words = ["hello", "world", "python", "list", "comprehension"]
uppercase_words = [word.upper() for word in words]
print(uppercase_words)
```

```
['HELLO', 'WORLD', 'PYTHON', 'LIST', 'COMPREHENSION']
```