Python Programing:

- Python is a General purpose High level programing Language
- Guido Van Rosaam was developed the python programing in 1989, but it officially realised in 1991

Why Python Programing Is Popular:

1. It is very simple and easy

   Example:

   ```
   if "E" in "Easy":
      print("Yes")
   ```

   Output:

   Yes

   Example:

   ```
   if "Z" in "Easy":
         print("Yes")
   else:
         print ("No")
   ```

   Output:

   No

2. Python programing used in multiple types applications.
   - Web applications,
   - Data Science,
   - AI,
   - Testing,
   - Automation,
   - Gaming,
   - Ecommerce ....e.t.c

3. Python having concise code

   Example: In c programing to print hello world

   ```
   #include <stdio.h>
   int main() {
    // printf() displays the string inside quotation
    printf("Hello, World!");
   }
   ```

   Output: Hello,World!

Trainer Name: G Jagan Mohan Mailid: jaganmohan. gavvala@gmail.com

Example: In python programing to print hello world

```
print("Hello, World!")
```

Output: Hello,World!

4. Python is used in Data Analysis.

Python features:

1. Python supports functional and procedure oriented features. These features taken from C programing.
2. Python supports Objective Oriented Programing. These features taken from C++.
3. Python supports scripting feature. These features taken from Shell Script.
4. Python supports modular programing feature. These feature taken from Modula3.

Python is platform independent:

- We can write python code in any operating system and to run any other operating system Without modification of code.

Python Version:

The current version of the python is 3.10.5. Up to python 2 versions developed based on enhancement of previous versions of python. But Python 3 developed again from scratch, Python doesn't have backward compatibility it means whatever code we written in python 2 which is not executed in python3 similarly.Python3 code is not executed in python2.

How to download and install python:

1. Go to website https://www.python.org/
2. Click on downloads and click on python windows

How to open it:

1. Go to windows start
2. Type python it showing results like IDLE (python 3.10)
3. Click open then python IDLE will be opened.

What is IDLE?

- IDLE stands for Integrated Development Learning Environment.
- When your beginner to the python if you want to practice some basic examples then go

For IDLE.

Trainer Name: G Jagan Mohan Mailid: jaganmohan. gavvala@gmail.com

Example:

```
>>> a=10
>>> b=20
>>> c=a+b
>>> a
10
>>> b
20
>>> c
30
```

Example:

```
>>> x=100
>>> y=2
>>> z=x/y
>>> x
100
>>> y
2
>>> z
50.0
```

- Python IDLE is working based on REPL tool
- REPL stands for Read Evaluate Print Loop
- It works Read------->Evaluate-------->Print then again it expecting new data i.e nothing but Loop

How we can excute python code:

By Using Python IDLE:

1. Open python IDLE
2. Go to file and click on New file
3. Write the code inside the file
4. Save the file with any name but extension with .py eg: abc.py
5. Go to Run and click on Run module.

Example:

tes.py

```
a=10
b=20
c=a+b
print(a+b)
print("Hello")
```

Output:

```
30
Hello
```

Run python code using Text Edtors:

Example:

test.py

```
a=10
b=20
c=a+b
print(a+b)
print("Hello")
```

output:

```
C:\Users\jagan>cd OneDrive

C:\Users\jagan\OneDrive>cd Desktop

C:\Users\jagan\OneDrive\Desktop>cd Python_Batch_4

C:\Users\jagan\OneDrive\Desktop\Python_Batch_4>python  test.py
30
Hello
```

Run python code using IDEs:

IDE standsfor Integrated Development Environment.

There are somany IDEs are there

1.PyCharm

2.Atom

3.Jupiter Not book

e.t.c

Identifiers:

- Identifier is a variable name or function name or class name.

Examples:

```
a=10

y=100


def  test():

----------

-----------


class  Student:

----------

--------

-----------
```

Rules to define identifiers:

1. Allowed charcters to define identifiers are A to Z,a to z, 0 to 9 and only one special symbol i.e _ (under score.)

   Example:

   total=100 ----->valid

   toTaL=200 ------>valid

   total123=5000 ------>valid

total_123=10000 ------>valid

total$=2000 ---------->Invalid

2. Identifier should not start with digit.

   Example:

   total123=1000 ---------->valid

   123total=1000 ---------->Invalid

3. Python is case sensitive programing language.
   Example:

```
abc=1000
ABC=2000
print(abc)
print(ABC)

output:
1000
2000
```

4. There no limit to length of identifier.

   Example

   a=10

   ab=200

   abc=500

   aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa=1000

5. Reserved words should not used as identifier.

   Example

   for=10 invalid

   if=1000 invalid

6. Diffrent types of variables are there

Example

x=10 ---------->Normal variable

_x=20 ---------->Protected variable

__x=200 ---------->Private variable

__x__=30 ---------->magical methods

Reserved words or Key words in python:

- Python having only 33 reserved words.

Example:

test.py

```
import keyword
l=keyword.kwlist
print(l)
print(len(l))
```

Output:

```
['False', 'None', 'True', 'and', 'as', 'assert',
'break', 'class', 'continue', 'def', 'del',
'elif', 'else', 'except', 'finally', 'for',
'from', 'global', 'if', 'import', 'in', 'is',
'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise',
 'return', 'try', 'while', 'with', 'yield']
33
```

Data Types:

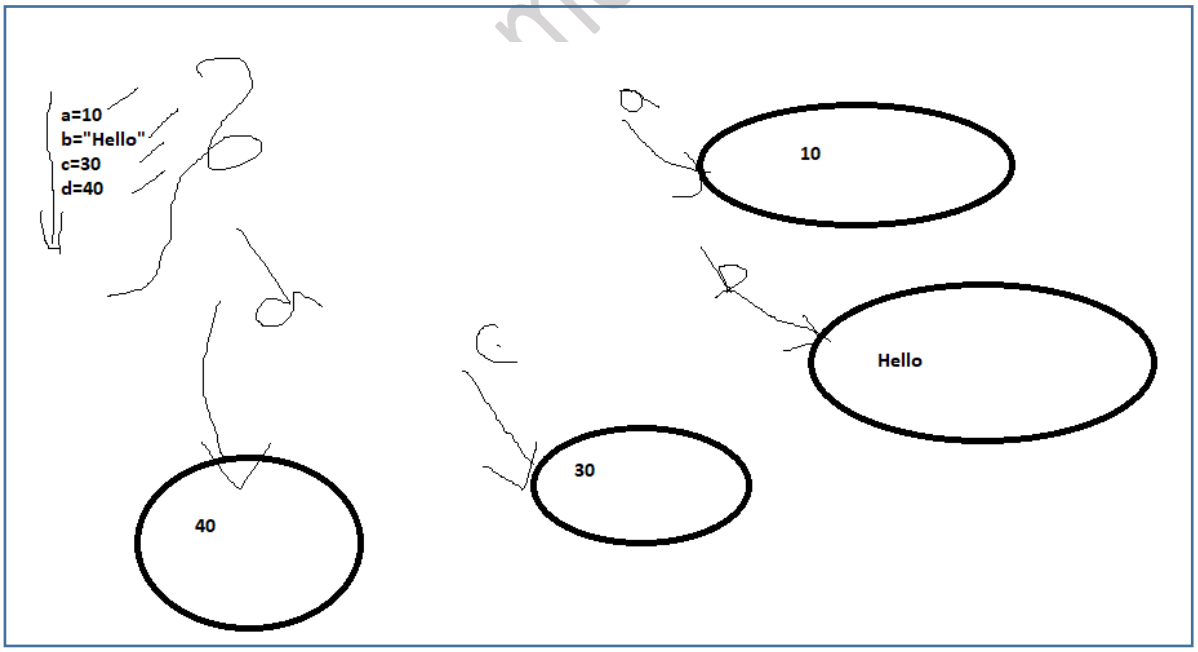
- Python is **dynamically** typed programing language.
- C and Java programing are statically typed programing language.

test.c:

```
main()
{
int a =10;
str b="Hello";
}
```

test.py:

```
a=10
b="Hello"
a="Hai"
```



- "In python everything treated as Object"

Trainer Name: G Jagan Mohan Mailid: jaganmohan. gavvala@gmail.com

Three important functions:

1. type()

- type() is inbuilt function of python by using type() to know data type stored in a variable.

Example:

```
a=10
print(a)
print(type(a))

b=10.5
print(b)
print(type(b))

c="Hello"
print(c)
print(type(c))
```

Output:

```
10
<class 'int'>
10.5
<class 'float'>
Hello
<class 'str'>
```


Trainer Name: G Jagan Mohan Mailid: jaganmohan. gavvala@gmail.com

2. id():

- id() is inbuilt function of python by using id () to know address of object

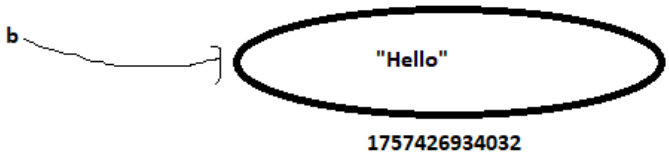
```
a=100
print(id(a))

b="Hello"

print(id(b))
```


```
output:

1916719888
1757426934032
```



Example:

```
a=10
print(a)
print(id(a))
b=20
print(b)
print(id(b))
```

Output:

```
10
1915013072
20
1915013392
```


Trainer Name: G Jagan Mohan Mailid: jaganmohan. gavvala@gmail.com

3. print():

- print() is inbuilt function of python by using print() to print the data

Example:

```
a=10
print(a)
b="Hello"
print(b)
```

Output:

```
10
Hello
```

The following are data types in python.

1. int
2. float
3. complex
4. bool
5. string
6. list
7. tuple
8. set
9. dict
10. range

1. Integer Data Type:

- If you want to mention any integer value we can go for integer data type

Example:

```
a=10
print(a)
print(type(a))
print(id(a))

Output:
10
<class 'int'>
1915013072
```

- We can represent integer data type in following format

1. a=10 ---------->Decimal format
2. a=0b1010 or a=0B1010 ---------->Binary format

   Example:

```
a=0b1010
print(a)
b=0B1111
print(b)
Output:
        10
        15
```

3. b=0o765 or b=0O765 ---------->Octal format

   Example:

```
a=0o765
print(a)
b=0O716
print(b)

Output:
        501
        462
```

4. c=0xABCD or c=0XABCD ----------->Hexa decimal format

Example:

```
a=0xABCD
print(a)
b=0Xabcd
print(b)

Output:
        43981
        43981
```

Note: But internally all representation treated as Decimal representation only.

Base conversions:

- To convert one base (one format) to another base (another format) is known as Base conversion.
- The following are Base conversion functions.
    1. bin() ---------->Given value is converted into Binary
    2. oct() ---------->Given value is converted into Octal
    3. hex() ---------->Given value is converted into hex format

Example:

```
a=10
print(a)
print(bin(a))
b=0b1010
print(b)
print(oct(b))
print(hex(b))

Output:
10
0b1010
10
0o12
0xa
```


Trainer Name: G Jagan Mohan Mailid: jaganmohan. gavvala@gmail.com

2. Float Data Type:

- If you want mention any float point values we can go for float data type

Example

```
a=12.5
print(a)
print(type(a))
```

Output

```
12.5
<class 'float'>
```

- To represent exponential numbers by using float data type.
- Example 1.2*10^4 we can represent this number by using float data type.

Example

```
a=1.2e4
print(a)
print(type(a))
```

Output:

```
12000.0
<class 'float'>
```

3. Complex data type:

- If you want to mention real and imaginary numbers we can go for complex data type

Syntax:

a+bj

Example:

```
a=20+40j
print(a)
print(type(a))
Output:
        (20+40j)
        <class 'complex'>
```

4. Boolean data type:

- True and False are comes under Boolean data type
    - True internally treated as 1
    - False internally treated as 0

Example:

```
        a=True
        b=False
        print(a)
        print(type(a))
        print(b)
        print(type(b))
Output:
        True
        <class 'bool'>
        False
        <class 'bool'>
```

Note:

- Int, Float, Complex and Boolean data types are comes under non iterations objects

5. String data type:

- Any number of sequence of characters enclosed with single or double or triple quotes is known as string

Example

```
s='Hello'
s="Hello"
s='''Hello'''
s="""Hello"""
s='C'
```

Example

```
s1="hello"
print(s1)
s2='hello'
print(s2)
```

Output:

```
hello
hello
```

Example

```
s1= "Hello
    how
        are you"
print(s1)
```

Output:

```
File "test.py", line 1
  s1= "Hello
      ^
SyntaxError: EOL while scanning string literal
```

Note: To define multi line strings compulsory use triple quotes.

Example

```
s1= """Hello
    how
        are you"""
print(s1)

s2= '''Hello
    how
        are you'''
print(s2)
```

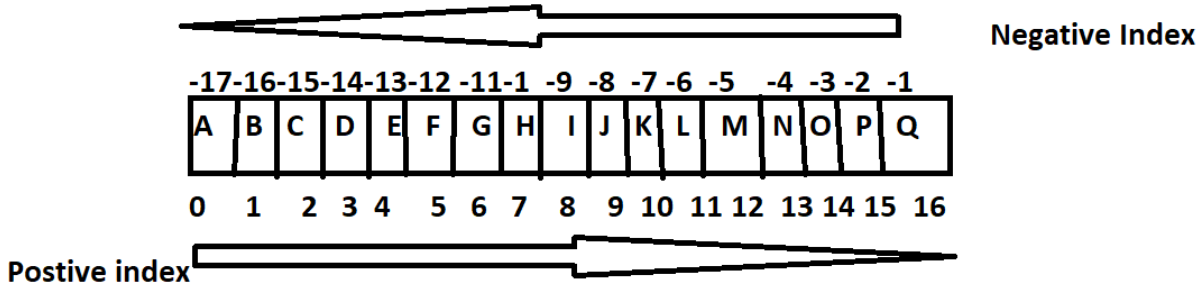
Output:

```
Hello
    how
        are you
Hello
    how
        are you
```

Indexing:

- To access characters in given string we can for Indexing
- Python supports two types index

1. **Positive index** –
   It starts from left to right and starting index value is 0 and end index value is length of string -1(i.e len(string)-1).
2. **Negative index**
   It starts from right to left starting index value is -1 and end index value is – of length string (i.e –(len(string))).



Syntax:

s[index value]

Example:

```
s1="ABCDEFGH"
print(len(s1))
print(s1[0])
print(s1[5])
print(s1[7])
print(s1[len(s1)-1])
print(s1[-1])
print(s1[-5])
print(s1[-8])
print(s1[-len(s1)])
```

Output:

```
8
A
F
H
H
H
D
A
A
```

Example:

```
s="HELLO"
print(s[2])
print(s[4])
print(s[-1])
```

Output:

```
L
O
O
```

Example:

```
s="HELLO"
print(s[100])
```

Output:

```
Traceback (most recent call last):
  File "test.py", line 2, in <module>
    print(s[100])
IndexError: string index out of range
```

Note: If string index value not in range of string index operator raises the IndexError.

Slice operator:

- By using slice operator we can access group of charters from given string

Syntax:

s[start index value: end index value]

- It slice the given string from start index value to end index value -1

Example:

- Given string s=”ABCDEFGH”
- s[4:8] here start index is 4 and end index is 8
- it slice the string from start index to end index -1 i.e 4 to 8 -1
- from 4 to 7 index all the characters EFGH

Example:

```
s="ABCDEFGH"
print(s[4:8])

Output:
EFGH
```

Example:

```
s="ABCDEFGH"
print(s[2:6])

Output:
CDEF
```

- s[:] Here not provided start index value and end index value ,so default start index is 0 and default end index is last character i.e len(string).

Example:

```
s="ABCDEFGH"
print(s[:])
Output:
ABCDEFGH
```

Example:

```
s="ABCDEFGH"
print(s[2:])
print(s[:5])
Output:
        CDEFGH
        ABCDE
```

Example:

```
s="ABCDEFGH"
print(s[2:1000])
Output:
CDEFGH
```

Note: Slice operator never raise any error

+ and * operator apply on strings:

- '+' Operator used for to concatenate the strings.
- '*' Operator used for to repeat the strings specified number of times.

Example:

```
s1=10
s2=20
print(s1+s2)
s3="Hello"
s4="How are you"
print(s3+s4)
output:
30
HelloHow are you
```

Example:

```
s1=10
s2=2
print(s1*s2)
s3="Hello"
s4=4
print(s3*s4)
Output:
20
HelloHelloHelloHello
```

Note: Integer, Float, Bool, Complex and String these data type are known as **Fundamental data types**.

Type casting or Type cohesion:

- The process of converting one data type to another data type is known as Type Casting.

- There are 5 type casting functions:

1.int()

2.float()

3.bool()

4.complex()

5.str()

1.int():

- Int() converts any other data type to integer data type.

Example:

```
a=9.2
print(a)
print(type(a))
print(int(a))
print(type(int(a)))
```

Output:

```
9.2
<class 'float'>
9
<class 'int'>
```

Note: float data type to integer data type conversion is possible.

Example:

```
a=True
print(a)
print(type(a))
print(int(a))
print(type(int(a)))
```

output:

```
True
<class 'bool'>
1
<class 'int'>
```

Example:

```
a=False
print(a)
print(type(a))
print(int(a))
print(type(int(a)))
```

Output:

```
False
<class 'bool'>
0
<class 'int'>
```


Note: bool data type to integer data type conversion is possible.

Example:

```
a=20+30j
print(a)
print(type(a))
print(int(a))

output:
(20+30j)
<class 'complex'>
Traceback (most recent call last):
  File "test.py", line 4, in <module>
    print(int(a))
TypeError: can't convert complex to int
```

Note: complex data type to integer data type conversion is not possible.

Example

```
a="Hello"
print(a)
print(type(a))
print(int(a))

Output:
Hello
<class 'str'>
Traceback (most recent call last):
  File "test.py", line 4, in <module>
    print(int(a))
ValueError: invalid literal for int() with base 10: 'Hello'
```

Example

```
a='1000'
print(a)
print(type(a))
print(int(a))
print(type(int(a)))
```

Output:

```
1000
<class 'str'>
1000
<class 'int'>
```

Note:

String data type integer data type is possible but string contains only numeric characters then only given string converted into integer. If string contains non numeric data is not possible to convert string to integer.

2.float():

- float() which converts any other data type to float data type

Example:

```
a=20
print(a)
print(type(a))
print(float(a))
print(type(float(a)))
```

Output:

```
20
<class 'int'>
20.0
<class 'float'>
```

Note: integer to float conversion is possible

Example:

```
a=True
print(a)
print(type(a))
print(float(a))
print(type(float(a)))
```

Output:

```
True
<class 'bool'>
1.0
<class 'float'>
```

Example:

```
a=False
print(a)
print(type(a))
print(float(a))
print(type(float(a)))
```

Output:

```
False
<class 'bool'>
0.0
<class 'float'>
```

Note: bool data to float conversion is possible

Example:

```
a=10+20j
print(a)
print(type(a))
print(float(a))
```

Output:

```
(10+20j)
<class 'complex'>
Traceback (most recent call last):
  File "test.py", line 4, in <module>
    print(float(a))
TypeError: can't convert complex to float
```

Note: complex data to float conversion is not possible

Example:

```
a="Hello"
print(a)
print(type(a))
print(float(a))
```

Output:

```
Hello
<class 'str'>
Traceback (most recent call last):
  File "test.py", line 4, in <module>
    print(float(a))
ValueError: could not convert string to float: 'Hello'
```

Example:

```
a="1000"
print(a)
print(type(a))
print(float(a))
print(type(float(a)))
```

Output:

```
1000
<class 'str'>
1000.0
<class 'float'>
```

Note:

String data type float data type is possible but string contains only numeric characters then only given string converted into float. If string contains non numeric data is not possible to convert string to float.

3.complex():

- complex() is convert any other data type to complex data type.

Example:

```
a=10+20j
b=0+80j
print(a)
print(b)
```

Output:

```
(10+20j)
80j
```

Format 1: complex(x)

```
a=10
print(a)
print(type(a))
print(complex(a))
print(type(complex(a)))
```

Output:

```
10
<class 'int'>
(10+0j)
<class 'complex'>
```

Note: It is possible to convert integer to complex.

Example:

```
a=10.5
print(a)
print(type(a))
print(complex(a))
print(type(complex(a)))
```

output:

```
10.5
<class 'float'>
(10.5+0j)
<class 'complex'>
```

Note: It is possible to convert float to complex.

Example:

```
a=True
print(a)
print(type(a))
print(complex(a))
print(type(complex(a)))
```

Output:

```
True
<class 'bool'>
(1+0j)
<class 'complex'>
```

Example:

```
a=False
print(a)
print(type(a))
print(complex(a))
print(type(complex(a)))
```

output:

```
False
<class 'bool'>
0j
<class 'complex'>
```

Note: It is possible to convert bool to complex .

Example:

```
a="ABCD"
print(a)
print(type(a))
print(complex(a))
```

Output:

```
ABCD
<class 'str'>
Traceback (most recent call last):
  File "test.py", line 4, in <module>
    print(complex(a))
ValueError: complex() arg is a malformed string
```

Example:

```
a="10100"
print(a)
print(type(a))
print(complex(a))
```

output:

```
10100
<class 'str'>
(10100+0j)
```

Note: string to complex data conversion is possible but string contains only numeric data. If string contains non numeric data it is not possible to convert string to complex

Format 2: complex(x,y)

Example:

```
a=10
b=20
print(a)
print(type(a))
print(b)
print(type(b))
print(complex(a,b))
print(type(complex(a,b)))
```

Output:

```
10
<class 'int'>
20
<class 'int'>
(10+20j)
<class 'complex'>
```

Example:

```
a=10.5
b=20.5
print(a)
print(type(a))
print(b)
print(type(b))
print(complex(a,b))
print(type(complex(a,b)))
```

Output:

```
10.5
<class 'float'>
20.5
<class 'float'>
(10.5+20.5j)
<class 'complex'>
```

Example:

```
a=True
b=True
print(a)
print(type(a))
print(b)
print(type(b))
print(complex(a,b))
print(type(complex(a,b)))
```

Output:

```
True
<class 'bool'>
True
<class 'bool'>
(1+1j)
<class 'complex'>
```

Example:

```
a="Hello"
b="hai"
print(a)
print(type(a))
print(b)
print(type(b))
print(complex(a,b))
```

Output:

```
Hello
<class 'str'>
hai
<class 'str'>
Traceback (most recent call last):
  File "test.py", line 7, in <module>
    print(complex(a,b))
TypeError: complex() can't take second arg if first is a string
```

Example:

```
a="1234"
b="200"
print(a)
print(type(a))
print(b)
print(type(b))
print(complex(a,b))
```

```
Output:
1234
<class 'str'>
200
<class 'str'>
Traceback (most recent call last):
  File "test.py", line 7, in <module>
    print(complex(a,b))
TypeError: complex() can't take second arg if first is a string
```

Note: string to complex data type conversion is not possible with complex(x,y).

4.bool():

- we can convert any data type to bool data is possible

Example:

```
a=10
b=10.5
c=10+20j
d="hello"

print(a)
print(type(a))
print(bool(a))
print(type(bool(a)))

print(b)
print(type(b))
print(bool(b))
print(type(bool(b)))
```

```
print(c)
print(type(c))
print(bool(c))
print(type(bool(c)))

print(d)
print(type(d))
print(bool(d))
print(type(bool(d)))

output:
10
<class 'int'>
True
<class 'bool'>
10.5
<class 'float'>
True
<class 'bool'>
(10+20j)
<class 'complex'>
True
<class 'bool'>
hello
<class 'str'>
True
<class 'bool'>
```


Note: bool() returns True when data is non-zero, non-empty string.

Example:

```
a=0
b=0.0
c=0+0j
d=""

print(a)
print(type(a))
print(bool(a))
print(type(bool(a)))

print(b)
print(type(b))
print(bool(b))
print(type(bool(b)))

print(c)
print(type(c))
print(bool(c))
print(type(bool(c)))

print(d)
print(type(d))
print(bool(d))
print(type(bool(d)))
```

Output:

```
0
<class 'int'>
False
<class 'bool'>
0.0
<class 'float'>
False
<class 'bool'>
0j
<class 'complex'>
False
<class 'bool'>
<class 'str'>
False
<class 'bool'>
```

Note: bool () returns False when data is zero, empty string.

5.str():

Example:

```
a=10
b=10.5
print(a)
print(type(a))
print(str(a))
print(type(str(a)))
print(b)
print(type(b))
print(str(b))
print(type(str(b)))
```

```
Output:
10
<class 'int'>
10
<class 'str'>
10.5
<class 'float'>
10.5
<class 'str'>
```

Example:

```
a=10+20j
print(a)
print(type(a))
print(str(a))
print(type(str(a)))
```

Output:

```
(10+20j)
<class 'complex'>
(10+20j)
<class 'str'>
```

Example:

```
a=True
b=False
print(a)
print(type(a))
print(str(a))
print(type(str(a)))
print(b)
print(type(b))
print(str(b))
print(type(str(b)))
```

```
Output:
True
<class 'bool'>
True
<class 'str'>
False
<class 'bool'>
False
<class 'str'>
```

Note: we can convert any data type to string data type is possible.

Immutable and Mutable:

- Immutable means unable to change and Mutable means change
- In python everything treated as an object. Every object holds by variable. When an object is initiated a unique id generated. Data type of variable defined at run time of program. Once object is created this object never changeable.
- "Once object will created we can't change the state of object is known as Immutable"

Example:

```
x=10
print(x)
print(id(x))
x=x+1  #x=10+1=11
print(x)
print(id(x))


Output:
10
1379387344
11
1379387376
```

Note: "All fundamental data types are immutable"

Example:

```
a="hello world"
b=a[3:6]
print(a)
print(b)
print(id(a))
print(id(b))

output:
hello world
lo
1862119233584
1862119253808
```

Need of immutable:

- Object sharing is possible so that memory utilization improved then performance will be improved.

Example:

```
a=10
b=20
c=30
d=10
e=20
f=30
print(a)
print(id(a))
print(b)
print(id(b))
```

```
print(c)
print(id(c))
print(d)
print(id(d))
print(e)
print(id(e))
print(f)
print(id(f))
```

output:

```
10
1691600848
20
1691601168
30
1691601488
10
1691600848
20
1691601168
30
1691601488
```

Example:

```
a=10.5
b=20.5
c=30.5
d=10.5
e=20.5
f=30.5
print(a)
print(id(a))
print(b)
print(id(b))
print(c)
print(id(c))
print(d)
print(id(d))
print(e)
print(id(e))
print(f)
print(id(f))
```

Output:

```
10.5
2552879190304
20.5
2552879190136
30.5
2552879190280
10.5
2552879190304
20.5
2552879190136
30.5
2552879190280
```

Example:

```
a=True
b=False
c=True
d=False
e=True
f=False
print(a)
print(id(a))
print(b)
print(id(b))
print(c)
print(id(c))
print(d)
print(id(d))
print(e)
print(id(e))
print(f)
print(id(f))
output:
True
1691119840
False
1691119872
True
1691119840
False
1691119872
True
1691119840
False
1691119872
```

Example:

```
a="Hello"
b="Hai"
c="Hello"
d="Hai"
e="Hello"
f="Hai"
print(a)
print(id(a))
print(b)
print(id(b))
print(c)
print(id(c))
print(d)
print(id(d))
print(e)
print(id(e))
print(f)
print(id(f))
```

output:

```
Hello
3027198468368
Hai
3027198469936
Hello
3027198468368
Hai
3027198469936
Hello
3027198468368
Hai
3027198469936
```

Example:

```
a=20+20j
b=22+22j
c=20+20j
d=22+22j
e=20+20j
f=22+22j
print(a)
print(id(a))
print(b)
print(id(b))
print(c)
print(id(c))
print(d)
print(id(d))
print(e)
print(id(e))
print(f)
print(id(f))
```

output:

```
(20+20j)
2537078029424
(22+22j)
2537078029456
(20+20j)
2537078029488
(22+22j)
2537078029520
(20+20j)
2537078029552
(22+22j)
2537078029584
```

Note: Complex data type is not sharing the object to variables

6.List Data Type:

- To store group of elements as single entry is known as collections (or array).

Example:

```
l=[1,2,3,4,5,6,7,8]
print(l)
print(type(l))

Output
[1, 2, 3, 4, 5, 6, 7, 8]
<class 'list'>
```

Features of the List data type:

1. In List order is preserved

Example:

```
l=["A","B","C","D","E","F"]
print(l)
print(type(l))
Output:
['A', 'B', 'C', 'D', 'E', 'F']
<class 'list'>
```

2. Duplicate elements are allowed

Example:

```
l=["A","B","C","D","E","F","A","B","D"]
print(l)
print(type(l))

Output:
['A', 'B', 'C', 'D', 'E', 'F', 'A', 'B', 'D']
<class 'list'>
```

3. List allow Heterogeneous objects.

Example:

```
l=["A","B","C","D","E","F","A","B","D",1,2,3,True,False,20+10j,1.2,2.2]
print(l)
print(type(l))
```

Output:

```
['A', 'B', 'C', 'D', 'E', 'F', 'A', 'B', 'D', 1, 2, 3, True, False, (20+10j), 1.2, 2.2]
<class 'list'>
```

4. List supports index and slicing

Example:

```
l=["A","B","C","D","E","F","A","B","D",1,2,3,True,False,20+10j,1.2,2.2]
print(l)
print(type(l))
print(l[12])
print(l[2])
print(l[2:6])
```

Output:

```
['A', 'B', 'C', 'D', 'E', 'F', 'A', 'B', 'D', 1, 2, 3, True, False, (20+10j), 1.2, 2.2]
<class 'list'>
True
C
['C', 'D', 'E', 'F']
```

5. List data type is mutable

Example:

```
l=[1,2,3,4]
print(l)
print(id(l))
l.append(5)
print(l)
print(id(l))

l[2]=999
print(l)
print(id(l))
l.remove(4)
print(l)
print(id(l))

output:
[1, 2, 3, 4]
2419837933640
[1, 2, 3, 4, 5]
2419837933640
[1, 2, 999, 4, 5]
2419837933640
[1, 2, 999, 5]
2419837933640
```

7. Tuple data type:

1. Tuple is represent ()
2. Tuple is same as List
3. Tuple is immutable
4. Tuple also known as read only version of list
5. Tuple follow index and slicing

Example:

```
t=(1,2,3,4,5)
print(t)
print(type(t))
print(id(t))
```

Output:

```
(1, 2, 3, 4, 5)
<class 'tuple'>
2340929458776
```

Example:

```
t=(1,2,3,4,5)
print(t)
print(t[2])
print(t[4])
print(t[2:5])
```

output:

```
(1, 2, 3, 4, 5)
3
5
(3, 4, 5)
```

Example:

```
t=(1,2,3,4,5)
print(t)
t.append(6)
output:
(1, 2, 3, 4, 5)
Traceback (most recent call last):
  File "test.py", line 3, in <module>
    t.append(6)
AttributeError: 'tuple' object has no attribute 'append'
```

Example:

```
t=(1,2,3,4,5)
print(t)
t[3]=200
Output:
(1, 2, 3, 4, 5)
Traceback (most recent call last):
  File "test.py", line 3, in <module>
    t[3]=200
TypeError: 'tuple' object does not support item assignment
```

Example:

```
t=()
print(t)
print(type(t))

Output:
()
<class 'tuple'>
```

Example:

```
t=(1)
print(t)
print(type(t))
```

Output:

```
1
<class 'int'>
```

Example:

```
t=(1,2)
print(t)
print(type(t))
```

Output:

```
(1, 2)
<class 'tuple'>
```

Example:

```
t=(1,)
print(t)
print(type(t))
```

Output:

```
(1,)
<class 'tuple'>
```

8. Set Data type:

Features of Set data type:

1. Order is not preserved
2. Duplicate elements are not allowed
3. Set with elements s= {1,2,3}or s=set([1,2,3]) or s=set((1,2,3))
4. empty set : set()
5. Index and slicing concepts not applicable for set
6. Heterogeneous objects are allowed
7. Set is mutable data type

Example:

```
s={1,2,3,4,5,6,7,8}
print(s)
print(type(s))
```

Output:

```
{1, 2, 3, 4, 5, 6, 7, 8}
<class 'set'>
```

Example:

```
s={1,2,3,4,5,6,7,8}
print(s[4])
```

Output:

```
Traceback (most recent call last):
  File "test.py", line 2, in <module>
    print(s[4])
TypeError: 'set' object does not support indexing
```

Example:

```
s={1,2,3,4,5,6,7,8}
print(s[4:8])
```

Output:

```
Traceback (most recent call last):
  File "test.py", line 2, in <module>
    print(s[4:8])
TypeError: 'set' object is not subscriptable
```


Trainer Name: G Jagan Mohan Mailid: jaganmohan. gavvala@gmail.com

Example:

```
s={1,2,3,4,5,6,7,8,1,2,3,4,5,6,7,8}
print(s)
```

Output:

```
{1, 2, 3, 4, 5, 6, 7, 8}
```

Example:

```
s={1,2,3,4,5,6,7,8,"A",2.5,20+6j,True}
print(s)
```

Output:

```
{1, 2, 3, 4, 5, 6, 7, 8, 2.5, (20+6j), 'A'}
```

Example:

```
s={}
print(s)
print(type(s))
```

Output:

```
{}
<class 'dict'>
```

Example:

```
s={1,2,3}
print(s)
print(type(s))
```

output:

```
{1, 2, 3}
<class 'set'>
```

Example:

```
s={1}
print(s)
print(type(s))
```

Output:

```
{1}
<class 'set'>
```

Example:

```
s={1,2,3}
print(s)
print(type(s))
```

Output:

```
{1, 2, 3}
<class 'set'>
```

Example

```
s=set()
print(s)
print(type(s))
```

Output:

```
set()
<class 'set'>
```


Trainer Name: G Jagan Mohan Mailid: jaganmohan. gavvala@gmail.com

Example:

```
s={1,2,3,4,5}
print(s)
s.add(6)
print(s)
s.remove(2)
print(s)
```

Output:

```
{1, 2, 3, 4, 5}
{1, 2, 3, 4, 5, 6}
{1, 3, 4, 5, 6}
```

9. Dictionary:

- To represent key and value pairs we can go for dict data

Syntax:

{key1:value1,key2:value2}

Example:

```
d={100:"Jagan",200:"Raj",300:"Ram"}
print(d)
print(type(d))
```

Output:

```
{100: 'Jagan', 200: 'Raj', 300: 'Ram'}
<class 'dict'>
```

Example:

```
d={}
d[10]="Jagan"
d[20]="Raj"
d[30]="Ram"
print(d)
print(type(d))
```

Output:

{10: 'Jagan', 20: 'Raj', 30: 'Ram'}

<class 'dict'>

Features of dictionaries:

1. order is not preserved in dictionary
2. Index and slice concepts not applicable
3. In dictionary duplicate values are allowed but duplicate keys not allowed

   Example:

   d={10:"Jagan",20:"Ram",30:"shyam",40:"Jagan",10:"Raj",41:"Jagan"}

   print(d)

   output:

   {10: 'Raj', 20: 'Ram', 30: 'shyam', 40: 'Jagan', 41: 'Jagan'}

4. Dictionary is mutable data type
5. we can take Heterogeneous data for key and values

10. range():

- We can represent range of sequence of numbers we go to the range data type

  format 1 : range(x)

  ========

  eg:

  r=range(10)

  print(r)

  print(type(r))

  output:

  range(0, 10)

  <class 'range'>

  r=range(100)

  print(r)

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

output:

```
range(0, 100)
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
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56

Trainer Name: G Jagan Mohan Mailid: jaganmohan. gavvala@gmail.com

57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87

Trainer Name: G Jagan Mohan Mailid: jaganmohan. gavvala@gmail.com

88

89

90

91

92

93

94

95

96

97

98

99

Format 2: range(start:end):

Example:

```
r=range(50,100)
print(r)


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

Output:

```
range(50, 100)
50
51
52
53
54
55
56
57
58
```

59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89

Trainer Name: G Jagan Mohan Mailid: jaganmohan. gavvala@gmail.com

90

91

92

93

94

95

96

97

98

99

Format 3: range(start,end,step):

Example:

```
r=range(50,100,2)
print(r)

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

Output:

```
range(50, 100, 2)
50
52
54
56
58
60
62
64
66
68
```

70

72

74

76

78

80

82

84

86

88

90

92

94

96

98

Features of range data type:

1. Range data type is immutable

2 range(x) here we take x value only integer we can't take x is float

Example:

```
r=range(50.5,100,2)
print(r)
for x in  r:
        print(x)
```

output:

```
Traceback (most recent call last):
File "test.py", line 1, in <module>
r=range(50.5,100,2)
TypeError: 'float' object cannot be interpreted as an integer
```

None data type:

- To handle a situation where value is not associate with variable

Example:

```
a=None
print(a)
print(type(a))
```

Output:

```
None
<class 'NoneType'>
```

How we can comment the python code:

- Following code is not correct way of commuting

  Example:

```
print("hello")
print("hello")
"""print("hello")
print("hello")
print("hello")
print("hello")"""
print("hello")
print("hello")
print("hello")
print("hello")
```

Single line comment:

- In python to comment specified line of code use '#' symbol at starting of the line of code.

  Example:

```
print("hello")
print("hello")
#print("Hai")
print("hello")
print("hello")
print("hello")
print("hello")
print("hello")
print("hello")
print("hello")
```

Multiple lines comment

- Multiple lines of comment code in python similar to single line of comment.

Example:

```
print("hello")
print("hello")
#print("Hai")
#print("hello")
#print("hello")
#print("hello")
print("hello")
print("hello")
print("hello")
print("hello")
```

Operators:

- The symbol which is responsible to perform some operations is known as an operator

1. Athematic operators
2. Relational operators
3. Logical operators
4. Bitwise operators
5. Shift operators
6. Assignment operators
7. Equality operators
8. Ternary operators
9. Special operators

1. Arithmetic operators:

- '+' symbol which is used for addition operations
- '-' symbol which is used for subtraction operations
- '*' symbol which is used for multiplication operations
- '/' symbol which is used for division operations
- '%' symbol which is used for modulo operations
- '//' symbol which is used for floor division operations
- '**' symbol which is used for power operations

Example:

```
a=10
b=20
print("Addtion:",a+b)
print("Subtraction:",a-b)
print("Mutliplication:",a*b)
print("Division:",a/b)
print("Reminder:",a%b)
```

Output:

```
Addtion: 30
Subtraction: -10
Mutliplication: 200
Division: 0.5
Reminder: 10
```

Example:

```
a=4
b=2
print("Addtion:",a+b)
print("Subtraction:",a-b)
print("Mutliplication:",a*b)
print("Division:",a/b)
print("Reminder:",a%b)
```

Output:

```
Addtion: 6
Subtraction: 2
Mutliplication: 8
Division: 2.0
Reminder: 0
```

Example:

```
a=4
b=2
print("Division:",a/b)
print("Floor Division:",a//b)
```

Output:

```
Division: 2.0
Floor Division: 2
```

Example:

```
a=4
b=3
print("Division:",a/b)
print("Floor Division:",a//b)
```

Output:

```
Division: 1.3333333333333333
Floor Division: 1
```

Example:

```
a=4.5
b=3.5
print("Division:",a/b)
print("Floor Division:",a//b)
```

Output:

```
Division: 1.2857142857142858
Floor Division: 1.0
```

Example:

```
a=4.5
b=3
print("Division:",a/b)
print("Floor Division:",a//b)
```

Output:

```
Division: 1.5
Floor Division: 1.0
```

Note:

- '/' operator always return float data type only.
- '//' operator returns integer data or float data based given data
- Consider a//b if a and b both are integers floor division operator returns integer value. If you take either a or b float data type floor division returns float data type only.

**** Operator:**

Example:

```
a=10
b=2
print("a power of b",a**b)  # 10^2
```

Output:

```
a power of b 100
```

Example:

```
a=10
b=2.5
print("a power of b",a**b)
```

Output:

```
a power of b 316.22776601683796
```

2. Relational operators:

- Relational operators are
  - '<' Less than
  - '<=' Less than or equal
  - '>' Grete than
  - '>=" Grete than or equal

Note: Relational operators always returns Boolean data only

Example:

```
a=10
b=20
print(a<b)
print(a<=b)
print(a>b)
print(a>=b)
```

Output:

```
True
True
False
False
```

Example:

```
a=20
b=10
print(a<b)
print(a<=b)
print(a>b)
print(a>=b)
```

Output:

```
False
False
True
True
```

Example:

```
a="a"    #unicode of 'a' is 97
b="A"    #unicode of 'A' is 65
print(a<b)  # 97<65
```

Output:

```
False
```

Note: All alphabets and special characters internally having a Unicode or ASCII code values.

By using Unicode or ASCII values relational operators will work

Example:

```
unicode of 'A' is 65
unicode of 'B' is 66
unicode of 'a' is 97
unicode of 'b' is 98
```

ord():

- To get ordinal or Unicode value of specified character we go for ord()

Example:

```
print(ord('A'))
print(ord('B'))
print(ord('a'))
print(ord('b'))
```

Output:

```
65
66
97
98
```

chr():

To get character of specified Unicode or ordinal value we go for chr()

Example:

```
print(chr(97))
print(chr(98))
print(chr(65))
print(chr(66))
```

Output:

```
a
b
A
B
```

Example:

```
print(ord('?'))
print(chr(63))
```

output:

63

?

Example:

```
a="Hello"
b="hello"
print(a<b)
```

Output:

True

Example:

```
a="Helloa"
b="Hellob"
print(a<b)
```

output:

True

Example:

```
a="Helloa"
b="HeLlob"
print(a<b)  #l < L
```

Output:

False

3. Logical operators:

- Logical operators are and,or,not
- 'and' operator:

Truth table for and operator:

A B A and B

True True True

True False False

False True False

False False False

Truth table for 'and' operator **A and B**

A	B	A and B
True	True	True
True	False	False
False	True	False
False	False	False

```
a=10
b=20
print(a and b)

output:
20
```

```
a=20
b=10

print(a and b)

output:
10
```

```
a=0
b=10

print(a and b)

output:
0
```

```
a=20
b=0

print(a and b)

output:
0
```

```
a=0
b=False

print(a and b)

output:
0
```

```
a=False
b=0

print(a and b)
output:
False
```

Example:

```
a=10
b=20
print(a and b)
```

output:

20

Example:

```
a=20
b=10
print(a and b)
```

output:

10

Example:

```
a=0
b=10
print(a and b)
```

Output:

0

Example:

```
a=0
b=False

print(a and b)
```

Output:

0

Example:

```
a=False
b=0
print(a and b)
```

Output:

0

or operator:

Truth table for 'or' operator:

a	b	a or b
True	True	True
True	False	True
False	True	True
False	False	False

Example:

```
a=10
b=20
print(a or b)
```

output:

10

Example:

```
a=10
b=0
print(a or b)
```

output:

10

Example:

```
a=0
b=10
print(a or b)
```

output:

10

Example:

```
a=0
b=False
print(a or b)
```

Output:

False

not operator:

Truth table for not operator

a	not a
True	False
False	True

Example:

```
a=True
print(not a)
```

Output:

False

Example:

```
a=10
print(not a)
```

Output:

False

Example:

```
a=0
print(not a)
```

output:

True

4. Bitwise operators:

- To perform the operations in bitwise we can go for bitwise operators.
  - & Bitwise 'and' operator
  - | Bitwise 'or' operator
  - ^ Bitwise 'xor' operator
  - ~ Bitwise 'compliment or negation' operator.

Note: These operators applicable only for integer and bool data type

Example:

```
a=4
b=5
print( a & b)
```

Output:

4

Example:

```
a=4
b=5
print( a | b)
```

output:

5

Example:

```
a=4
print(~ 4)
```

Output:

-5

Example:

```
a=5
b=4
print(a ^ b)
```

Output:

1

5.Shift operators:

To shift bits either left or right side we can go for shift operators.

- '<< 'it is left shift operator
- '>> 'it is right shift operator

'<< 'it is left shift operator

Example:

```
print(10 << 2)
print(bin(10 << 2))
```

Output:

40

0b101000

'>> 'it is right shift operator

Example:

```
print(10 >> 2)
print(bin(10 >> 2))
```

Output:

2

0b10

6. Assignment operators:

- To assign any value to the variable we can go for assignment operator i.e '='

Example:

- x=10
- a,b,c,d=20,30,40,50

Example:

```
a=10
b=20
c=30
d=40
print(a)
print(b)
print(c)
print(d)
```

output:

```
10
20
30
40
```

Example:

```
a,b,c,d=10,20,30,40
print(a)
print(b)
print(c)
print(d)
```

Output:

```
10
20
30
40
```


Trainer Name: G Jagan Mohan Mailid: jaganmohan. gavvala@gmail.com

Compound assignment operator:

- Assignment operator with some other operator combination is known as compound assignment operator.

Example:

```
x=10
x=x+1
print(x)
```

Output:

11

Example:

```
x=10
x+=1   #  x+=1 ---->x=x+(1)
print(x)
```

Output:

11

Example:

```
x=10
x-=1    # x-=1    ----------> x=x -(1)
print(x)
```

Output:

9

Note: Increment or decrement operators are not in python

Operator	Description	Example	Equivalent
+=	Add the value to the left-hand variable	x += 2	x = x + 2
-=	Subtract the value from the left-hand variable	x -= 2	x = x - 2
*=	Multiple the left-hand variable by the value	x *= 2	x = x * 2
/=	Divide the variable value by the right-hand value	x /= 2	x = x/2
//=	Use integer division to divide the variable's value by the right-hand value	x //= 2	x = x//2
%=	Use the modulus (remainder) operator to apply the right-hand value to the variable	x %= 2	x = x % 2
**=	Apply the power of operator to raise the variable 's value by the value supplied	x **= 3	x = x ** 3

7. Equality operator:

- To compare contentment of an object we go for equality operators

Equality operators are

1.'==' Equality operator

2.'!=' Not equal operator

Example:

```
a=20+30j
b=40+60j
c=20+30j
print(id(a))
print(id(b))
print(id(c))
print(a==b)
print(a==c)
print(a!=b)
print(a!=c)
```


Trainer Name: G Jagan Mohan Mailid: jaganmohan. gavvala@gmail.com

Output:

1565722835888

1565722835920

1565722835952

False

True

True

False

8. Ternary operator:

- '~' is unary operator Eg. ~4
- '+' ,'-','*'..e.t.c are binary operators

Syntax for ternary operator:

x= value1 if condition value2

Example:

```
x = 30  if 10 < 20   else 40
print(x)
```

Output:

30

Example:

```
x = 30  if 10 > 20   else 40
print(x)
```

Output:

40

Example:

```
x=int(input("value 1:"))
y=int(input("value 2:"))
r= x  if x < y  else y
print("min value of given values :",r )
```

output:

```
C:\Users\jagan\OneDrive\Desktop\pythonbatch2>python test.py
value 1:10
value 2:20
min value of given values : 10

C:\Users\jagan\OneDrive\Desktop\pythonbatch2>python test.py
value 1:50
value 2:40
min value of given values : 40

C:\Users\jagan\OneDrive\Desktop\pythonbatch2>python test.py
value 1:300
value 2:200
min value of given values : 200
```

9. Special operators:

- Special operators are
    1. Identity operator
    2. Membership operators

1**. Identity operator:**

- To compare id's of objects we go for Identity operators.
- Identity operators are
    - is
    - is not

Example:

```
a=10
b=20
c=30
d=10

print(a)
print(id(a))
print(b)
print(id(b))
print(c)
print(id(c))
print(d)
print(id(d))
```

output:

```
10
1691600848
20
1691601168
30
1691601488
10
1691600848
```

Example:

```
a=10
b=20
c=30
d=10
print(a)
print(id(a))
print(b)
print(id(b))
print(c)
print(id(c))
print(d)
print(id(d))
print ( a is d)
print(id(a) == id(d))
print(a is b )
print ( a is not  d)
print(id(a) != id(d))
print( a is not b)
```

Output:

```
10
1691600848
20
1691601168
30
1691601488
10
1691600848
True
True
False
```

False

False

True

2. Membership operators:

- If a particular character or group of characters are a member of given data or not.
- Membership operators are
  - in
  - not in

Example:

```
a=[1,2,3,4,5,6,7,8]
print(a)
print(4 in a)
print(100 in a)
```

Output:

```
[1, 2, 3, 4, 5, 6, 7, 8]
True
False
```

Example:

```
a=[1,2,3,4,5,6,7,8]
print(a)
print(4 not in a)
print(100 not in a)
```

Output:

```
[1, 2, 3, 4, 5, 6, 7, 8]
False
True
```

Example:

```
a="Hello How Are You !"
print("e" in a)
print("Are" in a)
print("are" in a)
print("HYou" in a)
output:
True
True
False
False
```

Module:

- Module is nothing but a group of variables ,classes, functions saved Into a python file

Example: Here XYZ.py is module

XYZ.py

```
a=100
b=200
def total(x,y):
    print("Total:",x+y)

def product(x,y):
    print("Product:",x*y)
```

Example: Here ABC.py is a normal file

- In ABC.py file we are importing XYZ module.

ABC.py

```
import XYZ
print(XYZ.a)
print(XYZ.b)
XYZ.total(100,200)
XYZ.product(5,4)
```

Output:

100

200

Total: 300

Product: 20

Various ways to import module:

1. import XYZ

Example:

```
import XYZ
print(XYZ.a)
print(XYZ.b)
XYZ.total(100,200)
XYZ.product(5,4)
```

Output:

100

200

Total: 300

Product: 20

2. from XYZ import *

Example:

```
from XYZ import *
print(a)
print(b)
total(10,20)
product(2,4)
```

output:

100

200

Total: 30

Product: 8

3. from XYZ import a,product

Example:

```
from XYZ import  a,product
print(a)
product(10,10)
```

Output:

```
100
Product: 100
```

Example:

```
from XYZ import  a,product
print(b)
```

output:

```
Traceback (most recent call last):
  File "ABC.py", line 3, in <module>
    print(b)
NameError: name 'b' is not defined
```

4. import xyz as x

Example:

```
import xyz  as x
print(x.a)
print(x.b)
x.total(2,5)
x.product(10,20)
```

output:

```
100
200
Total: 7
Product: 200
```

5. from xyz import total as t, product as p

Example

```
from xyz  import  total as t, product as p
t(10,20)
p(2,4)
```

output:

Total: 30

Product: 8

Math module:

- We can perform any mathematical operations we can go for math module.

Example:

```
from math import *
print(sqrt(4))
print(sin(90))
```

Output:

2.0

0.8939966636005579

help (module name):

- To get documentation of module we can go for help fumction.

Example:

```
import math
help(math)
```

Output:

Help on built-in module math:

NAME

math

DESCRIPTION

This module is always available. It provides access to the mathematical functions defined by the C standard.

FUNCTIONS

acos(...)

acos(x)

Return the arc cosine (measured in radians) of x.

acosh(...)

acosh(x)

Return the inverse hyperbolic cosine of x.

asin(...)

asin(x)

Return the arc sine (measured in radians) of x.

asinh(...)

asinh(x)

Return the inverse hyperbolic sine of x.

atan(...)

atan(x)

Return the arc tangent (measured in radians) of x.

atan2(...)

atan2(y, x)

Return the arc tangent (measured in radians) of y/x.

Unlike atan(y/x), the signs of both x and y are considered.

atanh(...)

atanh(x)

Return the inverse hyperbolic tangent of x.

ceil(...)

ceil(x)

Return the ceiling of x as an Integral.

This is the smallest integer >= x.

copysign(...)

copysign(x, y)

Return a float with the magnitude (absolute value) of x but the sign

of y. On platforms that support signed zeros, copysign(1.0, -0.0)

returns -1.0.

cos(...)

cos(x)

Return the cosine of x (measured in radians).

cosh(...)

cosh(x)

Return the hyperbolic cosine of x.

degrees(...)

degrees(x)

Convert angle x from radians to degrees.

erf(...)

erf(x)

Error function at x.

erfc(...)

erfc(x)

Complementary error function at x.

exp(...)

exp(x)

Return e raised to the power of x.

expm1(...)

expm1(x)

Return exp(x)-1.

This function avoids the loss of precision involved in the direct evaluation of exp(x)-1 for small x.

fabs(...)

fabs(x)

Return the absolute value of the float x.

factorial(...)

factorial(x) -> Integral

Find x!. Raise a ValueError if x is negative or non-integral.

floor(...)

floor(x)

Return the floor of x as an Integral.

This is the largest integer <= x.

fmod(...)

fmod(x, y)

Return fmod(x, y), according to platform C. x % y may differ.

frexp(...)

frexp(x)

Return the mantissa and exponent of x, as pair (m, e).

m is a float and e is an int, such that x = m * 2.**e.

If x is 0, m and e are both 0. Else 0.5 <= abs(m) < 1.0.

fsum(...)

fsum(iterable)

Return an accurate floating point sum of values in the iterable.

Assumes IEEE-754 floating point arithmetic.

gamma(...)

gamma(x)

Gamma function at x.

gcd(...)

gcd(x, y) -> int

greatest common divisor of x and y

hypot(...)

hypot(x, y)

Return the Euclidean distance, sqrt(x*x + y*y).

isclose(...)

isclose(a, b, *, rel_tol=1e-09, abs_tol=0.0) -> bool

Determine whether two floating point numbers are close in value.

rel_tol

maximum difference for being considered "close", relative to the

magnitude of the input values

abs_tol

maximum difference for being considered "close", regardless of the

magnitude of the input values

Return True if a is close in value to b, and False otherwise.

For the values to be considered close, the difference between them

must be smaller than at least one of the tolerances.

-inf, inf and NaN behave similarly to the IEEE 754 Standard. That

is, NaN is not close to anything, even itself. inf and -inf are

only close to themselves.

isfinite(...)

isfinite(x) -> bool

Return True if x is neither an infinity nor a NaN, and False otherwise.

isinf(...)

isinf(x) -> bool

Return True if x is a positive or negative infinity, and False otherwise.

isnan(...)

isnan(x) -> bool

Return True if x is a NaN (not a number), and False otherwise.

ldexp(...)

ldexp(x, i)

Return x * (2**i).

lgamma(...)

lgamma(x)

Natural logarithm of absolute value of Gamma function at x.

log(...)

log(x[, base])

Return the logarithm of x to the given base.

If the base not specified, returns the natural logarithm (base e) of x.

log10(...)

log10(x)

Return the base 10 logarithm of x.

log1p(...)

log1p(x)

Return the natural logarithm of 1+x (base e).

The result is computed in a way which is accurate for x near zero.

log2(...)

log2(x)

Return the base 2 logarithm of x.

modf(...)

modf(x)

Return the fractional and integer parts of x. Both results carry the sign

of x and are floats.

pow(...)

pow(x, y)

Return x**y (x to the power of y).

radians(...)

radians(x)

Convert angle x from degrees to radians.

sin(...)

sin(x)

Return the sine of x (measured in radians).

sinh(...)

sinh(x)

Return the hyperbolic sine of x.

sqrt(...)

sqrt(x)

Return the square root of x.

tan(...)

tan(x)

Return the tangent of x (measured in radians).

tanh(...)

tanh(x)

Return the hyperbolic tangent of x.

trunc(...)

trunc(x:Real) -> Integral

Truncates x to the nearest Integral toward 0. Uses the __trunc__ magic method.

DATA

e = 2.718281828459045

inf = inf

nan = nan

pi = 3.141592653589793

tau = 6.283185307179586

FILE

(built-in)

Example: Find the area of circle .

```
from math import *
#area of circle=pi*(r**2)
r=10
a=pi*(r**2)
print(a)
```

Output:

314.1592653589793

Example:

```
from math import sqrt as s , pi as p ,sin as s1
print(s(9))
print(p)
print(s1(90))
```

Output:

3.0

3.141592653589793

0.8939966636005579

Input and output statements in python:

Input statements

- To take data from end user (command prompt) we can go for input statements

Example:

```
a=input()
print(a)
```

Output:

```
C:\Users\jagan\OneDrive\Desktop\pythonbatch2>python test.py
6
6
C:\Users\jagan\OneDrive\Desktop\pythonbatch2>python test.py
8
8
```

Example:

```
a=input("Enter number:")
print(a)
```

Output:

```
Enter number:8
8
C:\Users\jagan\OneDrive\Desktop\pythonbatch2>python test.py
Enter number:10
10
```

Note: Whatever data entered by end user is always string data only.

Example:

```
a=input("Enter marks:")
print(a)
print(type(a))
```

Output:

```
C:\Users\jagan\OneDrive\Desktop\pythonbatch2>python test.py
Enter marks:70
70
<class 'str'>
C:\Users\jagan\OneDrive\Desktop\pythonbatch2>python test.py
Enter marks:80
80
<class 'str'>
C:\Users\jagan\OneDrive\Desktop\pythonbatch2>python test.py
Enter marks:70.5
70.5
<class 'str'>
C:\Users\jagan\OneDrive\Desktop\pythonbatch2>python test.py
Enter marks:30+20j
30+20j
<class 'str'>
C:\Users\jagan\OneDrive\Desktop\pythonbatch2>python test.py
Enter marks:True
True
<class 'str'>
```

Example:

```
a=int(input("Enter marks:"))
print(a)
print(type(a))
```

output:

```
C:\Users\jagan\OneDrive\Desktop\pythonbatch2>python test.py
Enter marks:90
90
<class 'int'>
```

```
C:\Users\jagan\OneDrive\Desktop\pythonbatch2>python test.py
Enter marks:100
100
<class 'int'>
```

Example:

```
a=int(input("Enter marks:"))
print(a)
print(type(a))
```

Output:

```
C:\Users\jagan\OneDrive\Desktop\pythonbatch2>python test.py
Enter marks:70
70
<class 'int'>
```

eval():

- eval() function is type casting function which is automatically type caste appropriate data which taken from input statement.

Example:

```
a=eval(input("Enter marks:"))
print(a)
print(type(a))
```

Output:

```
C:\Users\jagan\OneDrive\Desktop\pythonbatch2>python test.py
Enter marks:10
10
<class 'int'>

C:\Users\jagan\OneDrive\Desktop\pythonbatch2>python test.py
Enter marks:10.5
10.5
```

<class 'float'>

C:\Users\jagan\OneDrive\Desktop\pythonbatch2>python test.py

Enter marks:"ABCD"

ABCD

<class 'str'>

C:\Users\jagan\OneDrive\Desktop\pythonbatch2>python test.py

Enter marks:True

True

<class 'bool'>

C:\Users\jagan\OneDrive\Desktop\pythonbatch2>python test.py

Enter marks:20+10j

(20+10j)

<class 'complex'>

Example:

```
data=eval(input("Enter data :"))
print(type(data))
print(data)
```

Output:

C:\Users\jagan\OneDrive\Desktop\Python_sessions>python test.py

Enter data :[1,2,3,4,5,6]

<class 'list'>

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

C:\Users\jagan\OneDrive\Desktop\Python_sessions>python test.py

Enter data :(1,2,3,4,5,6)

<class 'tuple'>

(1, 2, 3, 4, 5, 6)

```
C:\Users\jagan\OneDrive\Desktop\Python_sessions>python test.py
Enter data :{1,2,3,4,5,6,1,2,3,4,5,6}
<class 'set'>
{1, 2, 3, 4, 5, 6}
C:\Users\jagan\OneDrive\Desktop\Python_sessions>python test.py
Enter data :{1:"A",2:"B",3:"C",4:"D"}
<class 'dict'>
{1: 'A', 2: 'B', 3: 'C', 4: 'D'}
```

Command line arguments:

- The arguments which are passed from command prompt are known as command line arguments
- In python sys is predefined module in that module there is one variable i.e argv. In argv variable we can store all command line arguments.

Example:

```
from sys import argv
print(type(argv))
print(argv)
```

Output

```
C:\Users\jagan\OneDrive\Desktop\pythonbatch2>python test.py
<class 'list'>
['test.py']
C:\Users\jagan\OneDrive\Desktop\pythonbatch2>python test.py  1 2 3 4 5 6
<class 'list'>
['test.py', '1', '2', '3', '4', '5', '6']
```

Example:

```
from sys import argv
print(argv)
print(type(argv))
print(argv[0])
print(argv[1:])
```

Output:

['test.py', 'A', 'B', 'C', 'D']

<class 'list'>

test.py

['A', 'B', 'C', 'D']

Example:

```
from sys import argv
args=argv[1:]
print(args)#['10', '20', '30', '40', '50']
total=0
for x in args:
   total=total+int(x)
print("Total:",total)
```

output:

['10', '20', '30', '40', '50']

Total: 150

Output statements:

- To print data in output (print data in console) we can go for output statements

print(): Empty print () used to print empty line in output.

Example:

```
print("Hello")
print("Hello"+"Good morning")
```

Output:

Hello

HelloGood morning

print(string)

Example:

```
print("Hello")
print()
print("Hello"+"Good morning")
```

Output:

```
Hello
HelloGood morning
```

sep=' ':

- In output more than one values are default separated with space.
- If we want any special character is separator between output values we can go for 'sep' attribute .

Syntax:

```
print(value1,value2,valu3,sep='special symbol')
```

Example:

```
a,b,c=10,20,30
print(a,b,c)
print(a,b,c,sep=',')
print(a,b,c,sep='-')
```

Output:

```
10 20 30
10,20,30
10-20-30
```

end=' ':

- Every print() statement print the data in new line. If you have multiple print() statements the data in output printed in multiple lines.
- You want to print the data in single line instead of multiple lines we can go for 'end' attribute.
  Syntax: print("Hello",end='')

Example:

```
a="Hello"
b="How"
c="Are"
d="you"
print(a)
print(b)
print(c)
print(d)
```

Output:

```
Hello
How
Are
you
```

Example:

```
a="Hello"
b="How"
c="Are"
d="you"
print(a,end='')
print(b,end='')
print(c,end='')
print(d)
```

Output:

```
HelloHowAreyou
```

Example:

```
a="Hello"
b="How"
c="Are"
d="you"
print(a,end=' ')
print(b,end=' ')
print(c,end=' ')
print(d)
output:
Hello How Are you
```

Example:

```
a="Hello"
b="How"
c="Are"
d="you"
print(a,end='-')
print(b,end='-')
print(c,end='-')
print(d)
```

Output:

Hello-How-Are-you

.format():

- The .format() method used to print the data in output

  Syntax:

  print("Hello{}hai{} how{}".format(a,b,c))

Example:

```
name="jagan"
sid=222
branch="EEE"
print("Name:",name,"Student Id:",sid,"Branch:",branch)
print("Name:{}  Student Id:{} Baranch:{}".format(name,sid,branch))
print(" Student Id:{} Name:{} Baranch:{}".format(name,sid,branch))
print("Student Id:{} Name:{} Baranch:{}".format(sid,name,branch))
```

Output:

```
Name: jagan Student Id: 222 Branch: EEE
Name:jagan  Student Id:222 Baranch:EEE
 Student Id:jagan Name:222 Baranch:EEE
Student Id:222 Name:jagan Baranch:EEE
```

Example:

```
name=input("Enter Student Name:")
sid=input("Enter student Id:")
branch=input("Enter student Branch:")
print("Student name is {}, id is {} belongs to {} branch".format(name,sid,branch))
```

Output:

```
C:\Users\jagan\OneDrive\Desktop\Python_sessions>python test.py
Enter Student Name:Jagan
Enter student Id:222
Enter student Branch:EEE
Student name is Jagan, id is 222 belongs to EEE branch

C:\Users\jagan\OneDrive\Desktop\Python_sessions>python test.py
Enter Student Name:Raj
Enter student Id:200
Enter student Branch:CSE
Student name is Raj, id is 200 belongs to CSE branch
```

```
C:\Users\jagan\OneDrive\Desktop\Python_sessions>

C:\Users\jagan\OneDrive\Desktop\Python_sessions>python test.py

Enter Student Name:Basha

Enter student Id:400

Enter student Branch:CIVIL

Student name is Basha, id is 400 belongs to CIVIL branch
```

Flow control:

At run time in which order statement going to be executed is decided by flow control

Indentation:

- Indentation in python refers to a tab space.
- To represent block of code with indentation

Example:

```
if 10>20:
   print("Hello")
print("Hai")
```

Example:

```
if (10>20):
  print("Hello")
      print("How")
print("Hai")
```

Output:

```
 File "test.py", line 3
  print("How")
  ^
IndentationError: unexpected indent
```

1. **Conditional statements:**
   - Conditional statements help you to make a decision based on certain conditions. These conditions are specified by a set of conditional statements having Boolean expressions which are evaluated to a Boolean value of true or false.

Conditional statements are

1.if

2.if else

3.if elif else

4.if elif elif .....else.

Example:

```
#WAP to find largest number in given two numbers.
n1=int(input("Enter First Number:"))
n2=int(input("Enter Second  Number:"))
if n1>n2:
   print("Large number:",n1)
else:
   print("Large number:",n2)
```

output:

```
C:\Users\jagan\OneDrive\Desktop\Python_sessions>python test.py
Enter First Number:10
Enter Second  Number:20
Large number: 20

C:\Users\jagan\OneDrive\Desktop\Python_sessions>python test.py
Enter First Number:20
Enter Second  Number:10
Large number: 20

C:\Users\jagan\OneDrive\Desktop\Python_sessions>python test.py
Enter First Number:11
Enter Second  Number:12
Large number: 12
```

```
C:\Users\jagan\OneDrive\Desktop\Python_sessions>python test.py
Enter First Number:12
Enter Second  Number:11
Large number: 12
```

Example:

```
#WAP to find largest number in given three numbers.
n1=int(input("Enter First Number:"))
n2=int(input("Enter Second  Number:"))
n3=int(input("Enter Third  Number:"))

if n1>n2  and n1>n3:
   print("Large number :",n1)
elif n2>n3:
   print("Large number :",n2)
else:
   print("Large number :",n3)
output:
C:\Users\jagan\OneDrive\Desktop\Python_sessions>python test.py
Enter First Number:30
Enter Second  Number:20
Enter Third  Number:10
Large number : 30

C:\Users\jagan\OneDrive\Desktop\Python_sessions>python test.py
Enter First Number:10
Enter Second  Number:30
Enter Third  Number:20
Large number : 30
```

```
C:\Users\jagan\OneDrive\Desktop\Python_sessions>python test.py
Enter First Number:10
Enter Second  Number:20
Enter Third  Number:30
Large number : 30
```

Example:

```
n=int(input("Enter a number:"))
if 1<=n<=100:
   print("Yes")
else:
   print("No")
```

Output:

```
C:\Users\jagan\OneDrive\Desktop\Python_sessions>python test.py
Enter a number:10
Yes
C:\Users\jagan\OneDrive\Desktop\Python_sessions>python test.py
Enter a number:90
Yes
```

```
C:\Users\jagan\OneDrive\Desktop\Python_sessions>python test.py
Enter a number:1
Yes
C:\Users\jagan\OneDrive\Desktop\Python_sessions>python test.py
Enter a number:100
Yes
C:\Users\jagan\OneDrive\Desktop\Python_sessions>python test.py
Enter a number:101
No
```

Example:

```
n=int(input("Enter a number:"))
if n in range(1,101):
   print("Yes")
else:
   print("No")
```

output:

```
C:\Users\jagan\OneDrive\Desktop\Python_sessions>python test.py
Enter a number:1
Yes

C:\Users\jagan\OneDrive\Desktop\Python_sessions>python test.py
Enter a number:90
Yes

C:\Users\jagan\OneDrive\Desktop\Python_sessions>python test.py
Enter a number:100
Yes

C:\Users\jagan\OneDrive\Desktop\Python_sessions>python test.py
Enter a number:101
No
```

Example:

```
n=int(input("Enter a number:"))
if n%2==0:
   print("Even Number")
else:
   print("odd Number")
```

Output:

C:\Users\jagan\OneDrive\Desktop\Python_sessions>python test.py

Enter a number:2

Even Number

C:\Users\jagan\OneDrive\Desktop\Python_sessions>python test.py

Enter a number:8

Even Number

C:\Users\jagan\OneDrive\Desktop\Python_sessions>python test.py

Enter a number:998

Even Number

C:\Users\jagan\OneDrive\Desktop\Python_sessions>python test.py

Enter a number:667

odd Number

C:\Users\jagan\OneDrive\Desktop\Python_sessions>python test.py

Enter a number:789

odd Number

Example:

```
list_of_states=["AP","TS","TN","KA","KL"]
print(list_of_states)
state=input("Enter Your State Name:")
if state in list_of_states:
   if state=="AP":
      print("AMVARVATHI")
   elif state=="TS":
      print("HYD")
   elif state=="TN":
      print("CHEN")
   elif state=="KA":
      print("BANG")
   else:
      print("Tiruvanthapur")
else:
   print("Your Entered State not found")
```

Output:

```
C:\Users\jagan\OneDrive\Desktop\Python_sessions>python test.py
['AP', 'TS', 'TN', 'KA', 'KL']
Enter Your State Name:AP
AMVARVATHI
C:\Users\jagan\OneDrive\Desktop\Python_sessions>python test.py
['AP', 'TS', 'TN', 'KA', 'KL']
Enter Your State Name:TS
HYD
C:\Users\jagan\OneDrive\Desktop\Python_sessions>python test.py
['AP', 'TS', 'TN', 'KA', 'KL']
Enter Your State Name:TN
CHEN
```

```
C:\Users\jagan\OneDrive\Desktop\Python_sessions>python test.py
['AP', 'TS', 'TN', 'KA', 'KL']
Enter Your State Name:KA
BANG

C:\Users\jagan\OneDrive\Desktop\Python_sessions>python test.py
['AP', 'TS', 'TN', 'KA', 'KL']
Enter Your State Name:KL
Tiruvanthapur
C:\Users\jagan\OneDrive\Desktop\Python_sessions>python test.py
['AP', 'TS', 'TN', 'KA', 'KL']
Enter Your State Name:Odisa
Your Entered State not found
```

2. **Iterative Statements:**

    1. for loop
    2. while loop

1. for loop:

- If you have sequence of elements (eg: list,tuple,set,dict,range) to perform any operation on each element in given sequence then we go for loop.

Syntax:

```
for   temapary_varible in sequence:
```

Example:

```
for x in ['A','B','C','D']:
   print(x)
```

Output:

```
A
B
C
D
```

Example:

```
for n in range(1,10):
   print(n)
```

output:

1

2

3

4

5

6

7

8

9

Example:

```
for n in range(1,10):
   print("sequre of {} is {}".format(n,n**2))
```

Output:

sequre of 1 is 1

sequre of 2 is 4

sequre of 3 is 9

sequre of 4 is 16

sequre of 5 is 25

sequre of 6 is 36

sequre of 7 is 49

sequre of 8 is 64

sequre of 9 is 81

Example:

```
n=range(1,101)
print(n)
even_list=[]
odd_list=[]
for x in n:
   if  x%2==0:
      even_list.append(x)
   else:
      odd_list.append(x)
print(even_list)
print(odd_list)
```

Output:

```
range(1, 101)
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22,
24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44,
 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66,
 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88,
 90, 92, 94, 96, 98, 100]
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25,
 27, 29, 31, 33, 35, 37, 39, 41,
 43, 45, 47, 49, 51, 53, 55, 57,
 59, 61, 63, 65, 67, 69, 71, 73,
 75, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97, 99]
```

2. while loop:

- To perform some operations on sequences based on some condition.

Example:

```
n=5
total=0
i=1
while i<=n:       #1<=5        2<=5       3<=5       4<=5       5<=5       6<=5
   print(i)       #1           2          3          4          5
   total=total+i  #total=0+1=1  total=1+2=3   total=3+3=6  total=6+4=10 total=10+5=15
   i=i+1          #i=1+1=2          i=2+1=3     i=3+1=4     i=4+1=5     i=5+1=6
print("Total:",total)
```

Output:

```
1
2
3
4
5
Total: 15
```

Example:

```
s="Hello"
n=len(s)-1#n=5-1=4
i=0
while i<=n:
   print(s[i])
   i=i+1
output:
H
e
l
l
o
```

Example:

```
l=[1,2,3,4,5,6,7,8]
i=0
n=len(l)-1
while i<=n:
   print(l[i])
   i=i+1
```

Output:

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

Example:

```
l=[1,2,3,4,5,6,7,8,9,10]
i=0
n=len(l)-1
even_list=[]
odd_list=[]
while i<=n:
   if l[i]%2==0:
      even_list.append(l[i])
   else:
      odd_list.append(l[i])
   i=i+1
print(even_list)
print(odd_list)
```

Output:

[2, 4, 6, 8, 10]

[1, 3, 5, 7, 9]

Infinite loops:

- Because of programmers mistake the conditions of loops always True. Such type of loops keep on iterating these loops are known as Infinite loops.

Example:

```
n=5
total=0
i=1
while i<=n:
   print(i)
```

Output:

```
 1
1
1
1
1
1
1
1
1
1
1
Traceback (most recent call last):
  File "test.py", line 6, in <module>
    print(i)
KeyboardInterrupt
```

Example:

```
while True:
   print("Hello")
```

output:

```
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Traceback (most recent call last):
  File "test.py", line 2, in <module>
    print("Hello")
KeyboardInterrupt
```

3. Transfer statements:

1. break

2 .continue

1.break:

- Break the loop based on some condition.

Example:

```
cart=[10,20,30,40,500,700,300]
for item in cart:
   print("Price:",item)
```

Output:

Price: 10

Price: 20

Price: 30

Price: 40

Price: 500

Price: 700

Price: 300

Example:

```
cart=[10,20,30,40,500,700,300,100,20,40]
for item in cart:
   if item>500:#700>500
      print("This item insurance required")
        break
   print("Price:",item)
print("End")
```

output:

Price: 10

Price: 20

Price: 30

Price: 40

Price: 500

This item insurance required

End

2. continue:

- Skip the current iteration based on some condition.

Example:

```
cart=[10,20,30,40,500,700,300,100,20,40]
for item in cart:
   if item>500:#700>500
      print("This item insurance required")
      continue
   print("Price:",item)
print("End")
```

Output:

Price: 10

Price: 20

Price: 30

Price: 40

Price: 500

This item insurance required

Price: 300

Price: 100

Price: 20

Price: 40

End

Example:

```
cart=[10,20,30,40,500,700,300,100,20,40,600,300,800,20]
for item in cart:
   if item>500:#700>500
      print("This item insurance required for the item:",item )
      continue
   print("Price:",item)
print("End")
```

Output:

```
Price: 10
Price: 20
Price: 30
Price: 40
Price: 500
This item insurance required for the item: 700
Price: 300
Price: 100
Price: 20
Price: 40
This item insurance required for the item: 600
Price: 300
This item insurance required for the item: 800
Price: 20
End
```

Pass statement::

- if required some empty block we write pass statement

Example:

```
for x in range(10,20):
```

Output:

```
  File "test.py", line 3
                    ^
SyntaxError: unexpected EOF while parsing
```

Example:

```
for x in range(10,20):
   pass
```

del statement:

- del is key word in python to delete object

Example:

```
x=10
print(x)
del x
print(x)
```

Output:

```
10
Traceback (most recent call last):
  File "test.py", line 4, in <module>
    print(x)
NameError: name 'x' is not defined
```

None:

- if want to delete current object but don't want to delete variable we can go for None

Example:

```
x=10
print(x)
x=None
print(x)
```

Output:

```
10
None
```


Trainer Name: G Jagan Mohan Mailid: jaganmohan. gavvala@gmail.com