

Python-Study

1. Special Chars:

Write special chars only inside of quotes---> ", ""

`\n` --> new line(always use in quotation)

Example-

code: `print("Welcome to python scripting.\nWe are working with special chars")`

Output:

Welcome to python scripting.

We are working with special chars

`\b` --> back space remove 1 postion

Example-

code: `print("Hello \bWorld")`

Output:

HelloWorld

`\t` --> tab 1 extra spce

Example-

code: `print ("Hello \tWorld")`

Output:

Hello World

`\` - escape symobol

Example-

code: `print('python\'s class')`

Output:

python's class

code:`print("This is a \"python\" class")`

Output:

This is a "python" class

`\a` --> alert sound

2.Variables:

-->decalare can be letters, number or underscore.
-->it should not a keyword.
-->can't contain spaces.
-->it should not start a number.
-->case sensitive.

Exapmle:

Code:

```
x=10
y=4.5
print(x)
print('x')
print(y)
x=23
print(type(x))
print(type(y))
print(x)
```

Output:

```
10
x
4.5
<class 'int'>
<class 'float'>
23
```

Code:

```
del x
print(x)
```

Output:

```
no value
```

Code:

```
x=6
print(id(x))--will show memory location of x value
```

Output:

```
140078623627329
```

3.Data types:

Basics Data types:

3.1. Numbers(integer, float, complex)

Example:

Code:

```
x=3
y=4.5
z=3+4j
print(x,type(x))
print(y,type(y))
print(z,type(z))
```

Output:

```
3 <class 'int'>
4.5 <class 'float'>
(3+4j) <class 'complex'>
```

3.2 Strings:

Example:

Code:

```
my_name="khair"
print(my_name,type(my_name))
```

Output:

```
khair <class 'str'>
```

3.3 Boolean:

Example:

Code:

```
my_name=True
my_new_name=False

print(my_name,type(my_name))
print(my_new_name,type(my_new_name))
```

Output:

```
True <class 'bool'>
False <class 'bool'>
```

4. Type casting or Type conversion:

----> Any data type can be converted into string but reverse is not always true.

----> Any data type can be converted into boolean

bool(any_data_type)= True or False

bool(empty)= False

Example:

```
bool(0)=False
```

```
x=" "
```

```
bool(x)=False
```

```
bool(None)=False
```

```
bool({})=False
```

```
bool(())=False
```

```
bool([])=False
```

```
bool(non_empty)= True
```

Code:

```
x=56
```

```
print(x,type(x))
```

```
y=str(x)
```

```
print(y,type(y))
```

```
z=bool(x)
```

```
print(z,type(z))
```

Output:

```
56 <class 'int'>
```

```
56 <class 'str'>
```

```
True <class 'bool'>
```

5. Working with variables or multiple variables:

Code:

```
x=3;y=5.7;z="Python scripting"
```

```
print(x,y,z)
```

```
print("{} {} {}".format(x,y,z))
```

```
print("{} \n{} \n{}".format(x,y,z))
```

```
print(f"x value is: {x} \ny value is: {y} \nz value is: {z}")
```

Output:

3 5.7 Python scripting

3 5.7 Python scripting

3

5.7

Python scripting

x value is: 3

y value is: 5.7

z value is: Python scripting

6. Input and Output statement:

---> Input and Output function will take input data as string

Code:

```
#simple arithmetic cal
a=4
b=8
result=a+b
print(f'The addition of {a} and {b} is: {result}')
```

Output:

The addition of 4 and 8 is: 12

Code:

```
a=input("Enter a value:")
b=input("Enter b value:")
result=a+b
print(f'The value of result is: {result} and type is: {type(result)}')
```

Output:

Enter a value:4

Enter b value:8

The value of result is: 48 and type is: <class 'str'>

Code:

```
a=int(input("Enter a value:"))
b=int(input("Enter b value:"))
result=a+b
print(f'The value of result is: {result} and type is: {type(result)}')
```

Output:

Enter a value:4
Enter b value:8
The value of result is: 12 and type is: <class 'int'>

Code:

```
a=eval(input("Enter a value:"))  
b=eval(input("Enter b value:"))  
result=a+b  
print(f'The value of result is: {result} and type is: {type(result)}')
```

Output:

Enter a value:4
Enter b value:8
The value of result is: 12 and type is: <class 'int'>

7. Basics Strings Variables:

--->A string is a sequence of characters.

```
my= "P y t h o n"  
Index= [0] [1] [2] [3] [4] [5]  
        [-6] [-5] [-4] [-3] [-2] [-1]
```

7.1 Access particula characters in string:**Code:**

```
my_fav_scripting="Python"  
print(my_fav_scripting)  
print(my_fav_scripting[3])  
print(my_fav_scripting[-1])
```

Output:

Python
h
n

7.2 Slicing Operation:

Code:

```
my_fav_scripting="Python"  
print(my_fav_scripting[0:5])
```

Output:

Pytho

Code:

```
my_fav_scripting="Python"  
print(my_fav_scripting[0:])
```

Output:

Python

7.3 Delete of string:

```
del my_fav_scripting
```

7.4 Length of Strings:

Code:

```
my_fav_scripting="Python"  
print(f'the length of a string is: {len(my_fav_scripting)}')
```

Output:

the length of a string is: 6(index is 0-5)

7.5 Concatenate two string:

Code:

```
my_str1="Python"  
my_str2="Scripting"  
my_str3=my_str1+my_str2  
print(my_str3)
```

Output:

PythonScripting

7.6 Python-Scripting:

Code:

```
my_str1="Python"  
my_str2="Scripting"  
my_str_s=" "  
my_str3=my_str1+my_str_s+my_str2  
print(my_str3)
```

Output:

Python Scripting

Code:

```
my_str1="Python"  
my_str2="Scripting"  
my_str3=my_str1+" "+my_str2  
print(my_str3)
```

Output:

Python Scripting

Code:

```
my_str1="Python"  
my_str2="Scripting"  
my_str3=my_str1+" "+my_str2+" "+"tutorials"  
print(my_str3)
```

Output:

Python Scripting tutorials

8. Case Conversion:

8.1 lower/upper case:

Code:

```
my_string="Python scripting"  
print(my_string.lower())  
print(my_string.upper())  
print(my_string)  
print(my_string.swapcase())  
print(my_string.title())  
print(my_string.capitalize())
```


Output:

```
python scripting
PYTHON SCRIPTING
Python scripting
pYTHON SCRIPTING
Python Scripting
Python scripting
```

9. Boolean Result Operation:**Code:**

```
my_string="Python"
print(my_string.startswith('P'))
print(my_string.endswith('N'))
print(my_string.islower())
print(my_string.isupper())
```

Output:

```
True
False
False
False
```

10. Join, Center & Zfill string:**10.1 Join:****Code:**

```
x="Python"
y="-".join(x)
print(y)
print("*".join(x))
print("\n".join(x))
print("\t".join(x))
```

Output:

```
P-y-t-h-o-n
P*y*t*h*o*n
P
y
t
h
o
n
```

P y t h o n

10.2 Center:

Code:

```
my_str1="Python"
my_str2="Python Scripting"
my_str3="String Operations"
print(f"{my_str1.center(20)}\n{my_str2.center(20)}\n{my_str3.center(20)}")
```

Output:

```
        Python
    Python Scripting
String Operations
```

10.3 Zfill:

Code:

```
my_str1="Python"
print(my_str1.zfill(10))
```

Output:

```
0000Python
```

11. Strip & Split:

11.1 Strip:

Strip remove space inside quotation or word

Code:

```
x=" Python "
```

```
print(x.strip())
```

Output:

```
Python
```

Code:

```
x="Python is best language Python"
print(x.strip('Python'))
print(x.rstrip('Python'))
print(x.lstrip('Python'))
```

Output:

```
is best language
Python is best language
is best language Python
```

11.2 Split:

Split is separating string where is space its make output into a list.

Code:

```
x="Python is easy and it is better"
print(x.split())
print(x.split('is'))
```

Output:

```
['Python', 'is', 'easy', 'and', 'it', 'is', 'better']
['Python ', ' easy and it ', ' better']
```

12. Count,index and find:**12.1 Count:****Code:**

```
x="Python is easy and it is a common language"
print(x.count('is'))
```

Output:

```
2
```

12.2 Index:

index will give output from left to right

Code:

```
x="Python is easy and it is a common language"
print(x.index('P'))
```

```
x="python is easy and it is a popular language"
print(x.index('p',2))
```

Output:

```
0
27
```

12.3 Find:

Code:

```
x="python is easy and it is a popular language"
print(x.find('p',2))
```

Output:

27

13. Practise String:

Code:

```
given_str="Python Scripting"
print(given_str.center(122))
print(given_str.ljust(122))
print(given_str.rjust(122))
```

Output:

```
Python Scripting
Python Scripting
Python Scripting
```

Example-

```
>>> import os
>>> os.get_terminal_size()
os.terminal_size(columns=140, lines=31)
>>> os.get_terminal_size().columns
140
```

Example-

```
import os
t_w=os.get_terminal_size().columns
given_str=input("Enter the value is:")
print(given_str.center(t_w).title())
print(given_str.ljust(t_w).title())
print(given_str.rjust(t_w).title())
```

Output:

```
Enter the value is:python scripting
Python Scripting
Python Scripting
Python Scripting
```

14. Data Structures:

Data structures are used to store collection of data

```
my_valuse=[3,4,5, 'python', 'script',5.6]
```

There are four built-in data structures:

```
---> List-[]
```

```
---> Tuple-()
```

```
---> Dictionary-{} with key value pair
```

```
---> Set-{} 
```

15. List:

Lists are mutable, strings are not mutable

```
bool(empty_list)==False
```

```
bool(non_empty_list)==True
```

```
my_list=[]
```

Code:

```
my_list1=[3,2,4,"python",5.6]
```

```
print(my_list1[0])
```

```
print(my_list1[3])
```

```
print(my_list1[-1])
```

```
print(my_list1[-2])
```

Output:

```
3
```

```
python
```

```
5.6
```

```
python
```

15.1 List String index[2]:

Code:

```
my_list1=[3,2,4,"python",5.6]
```

```
print(my_list1[3][2])
```

Output:

```
t
```

15.2 List index 1-4:

Code:

```
my_list1=[3,2,4,"python",5.6]  
print(my_list1[1:4])
```

Output:

```
[2, 4, 'python']
```

15.3 index value change:

Code:

```
my_list1=[3,2,4,"python",5.6]  
my_list1[0]=45  
print(my_list1)
```

Output:

```
[45, 2, 4, 'python', 5.6]
```

15.4 Index:

Code:

```
my_list=[3,5,2,7,3,5,9]  
print(my_list.index(5))
```

Output:

```
1
```

15.5 Count:

Code:

```
my_list=[3,5,2,7,3,5,9]  
print(my_list.count(5))
```

Output:

```
2
```

15.6 Copy:

Code:

```
my_new_list=my_list-same memory location
my_one_list=my_list.copy()-different memory location
print(id(my_list), id(my_new_list))
print(id(my_one_list))
```

Output:

```
140302972751168 140302972751168
140302973264384
```

15.7 Append:

Code:

```
my_list=[3,5,2,7,3,5,9]
print(my_list)
my_list.append(56)
print(my_list)
```

Output:

```
[3, 5, 2, 7, 3, 5, 9]
[3, 5, 2, 7, 3, 5, 9, 56]
```

15.8 Insert:

Code:

```
my_list=[3,5,2,7,3,5,9]
my_list.insert(1,45)
print(my_list)
```

Output:

```
[3, 45, 5, 2, 7, 3, 5, 9]
```

15.9 Nested List:

Code:

```
my_list=[3,5,2,7,3,5,9]
my_new_list=[13,26]
my_list.append(my_new_list)
print(my_list)
```

Output:

[3, 5, 2, 7, 3, 5, 9, [13, 26]]

15.10 Extend:**Code:**

```
my_list=[3,5,2,7,3,5,9]
my_new_list=[13,26]
my_list.extend(my_new_list)
print(my_list)
```

Output:

[3, 5, 2, 7, 3, 5, 9, 13, 26]

15.11 Remove:**Code:**

```
my_list=[3,5,2,7,3,5,9]
my_list.remove(7)
print(my_list)
```

Output:

[3, 5, 2, 3, 5, 9]

15.12 POP:**Code:**

```
my_list=[3,5,2,7,3,5,9]
my_list.pop()
print(my_list)
```

Output:

[3, 5, 2, 7, 3, 5]

Code:

```
my_list=[3,5,2,7,3,5,9]
my_list.pop(0)
print(my_list)
```

Output:

[5, 2, 7, 3, 5, 9]

15.13 Reverse:

Code:

```
my_list=[3,5,2,7,3,5,9]
my_list.reverse()
print(my_list)
```

Output:

```
[9, 5, 3, 7, 2, 5, 3]
```

15.14 Sort:

Code:

```
my_list=[3,5,2,7,3,5,9]
my_list.sort()
print(my_list)
```

Output:

```
[2, 3, 3, 5, 5, 7, 9]
```

16. Tuples:

---> Tuples are immutable

```
my_empty=()
my_tuple=(3,4,5)
print(my_tuple)
```

Code:

```
my_tuple=(3,4,[5,6,7],8,9)
print(my_tuple)
print(my_tuple[1])
```

Output:

```
(3, 4, [5, 6, 7], 8, 9)
4
```

Code:

```
my_tuple=(3,4,[5,6,7],8,9)
print(my_tuple[2][1])
```

Output:

```
6
```

16.1 Count:

Code:

```
x=5,2,3,4,5  
print(x.count(5))
```

Output:

2

16.2 index:

Code:

```
x=(5,2,3,4,5)  
print(x.index(5))
```

Output:

0

16.3 Length:

Code:

```
x=(5,2,3,4,5)  
print(len(x))
```

Output:

5

Code:

```
my_tuple=(2,4,7,8,46,65)  
print(my_tuple[3:5])-up to 5
```

Output:

(8, 46)

17. Dictionary:

Code:

```
my_dict={}
print(my_dict,type(my_dict))

my_dict={'fruit':'apple','animal':'fox',1:'one','two':2}
print(my_dict,type(my_dict))
```

Output:

```
{ } <class 'dict'>

{'fruit': 'apple', 'animal': 'fox', 1: 'one', 'two': 2} <class 'dict'>
```

17.1 Get:

Code:

```
my_dict={'fruit':'apple','animal':'fox',1:'one','two':2}
print(my_dict['fruit'])
print(my_dict.get('animal'))
```

Output:

```
apple
fox
```

17.2 Adding Key value pair:

Code:

```
my_dict={'fruit':'apple','animal':'fox',1:'one','two':2}
my_dict['three']=3
print(my_dict)
```

Output:

```
{'fruit': 'apple', 'animal': 'fox', 1: 'one', 'two': 2, 'three': 3}
```

Code:

```
my_dict={'fruit': 'apple', 'animal': 'fox', 1: 'one', 'two': 2, 'three': 3}
print(my_dict.keys())
print(my_dict.values())
print(my_dict.items())
```

Output:

```
dict_keys(['fruit', 'animal', 1, 'two', 'three'])
dict_values(['apple', 'fox', 'one', 2, 3])
dict_items([('fruit', 'apple'), ('animal', 'fox'), (1, 'one'), ('two', 2), ('three', 3)])
```

17.3 Copy:**Code:**

```
my_dict={'fruit': 'apple', 'animal': 'fox', 1: 'one', 'two': 2, 'three': 3}
y=my_dict.copy()
print(id(y),id(my_dict))
print(y)
```

Output:

```
139671763992704 139671763963392
{'fruit': 'apple', 'animal': 'fox', 1: 'one', 'two': 2, 'three': 3}
```

17.4 Update:**Code:**

```
my_dict={'fruit': 'apple', 'animal': 'fox', 1: 'one', 'two': 2, 'three': 3}
my_new_dict={'four':4}
my_dict.update(my_new_dict)
print(my_dict)
```

Output:

```
{'fruit': 'apple', 'animal': 'fox', 1: 'one', 'two': 2, 'three': 3, 'four': 4}
```

17.5 POP:**Code:**

```
my_dict={'fruit': 'apple', 'animal': 'fox', 1: 'one', 'two': 2, 'three': 3, 'four': 4}
my_dict.pop('four')
print(my_dict)
```

Output:

```
{'fruit': 'apple', 'animal': 'fox', 1: 'one', 'two': 2, 'three': 3}
```

Code:

```
my_dict={'fruit': 'apple', 'animal': 'fox', 1: 'one', 'two': 2, 'three': 3}
my_dict.popitem()
print(my_dict)
```

Output:

```
{'fruit': 'apple', 'animal': 'fox', 1: 'one', 'two': 2}
```

17.6 Fromkeys:**Code:**

```
keys=['a','e','i','o','u']
new_dict=dict.fromkeys(keys)
print(new_dict)

new_dict['a']='alpha'
print(new_dict)
```

Output:

```
{'a': None, 'e': None, 'i': None, 'o': None, 'u': None}
{'a': 'alpha', 'e': None, 'i': None, 'o': None, 'u': None}
```

17.7 Setdefault:**Code:**

```
my_dict={}
my_dict.setdefault('k',45)-set default take only 2 arguments
my_dict.setdefault('z',87)
print(my_dict)
```

Output:

```
{'k': 45, 'z': 87}
```

18. Sets:**Code:**

```
my_set={4,5,7,2,7,0}
print(my_set)
```

Output:

```
{0, 2, 4, 5, 7}
```

18.1 List to set:

Code:

```
my_li=[1,3,5,6,7]
set(my_li)
print(set(my_li))
```

Output:

```
{1, 3, 5, 6, 7}
```

Code:

```
a={1,2,3,4,5,6}
b={5,6,7,8,9}
print(a.union(b))
print(a.intersection(b))
```

Output:

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

19. Python Operators:

Arithmetic & Assignment Operators---> Takes values as inputs, performs its operation on input values and gives values as outputs.

Comparison, Identity & Membership Operators---> Takes values as inputs, perform its operation on input values and gives output as either True or False.

Logical---> Takes True or False as inputs, performs operation on this inputs and give output as either True or False.

Bitwise---> Takes values as inputs, performs operations on its binary representation and gives output as a value.

19.1 Arithmetic Operators:

- > Addition +
- > Substraction -
- > Multiplication *
- > Division /
- > Modulo %
- > Floor division //
- > Exponential **

Code:

```
x=2+3  
print(x)
```

```
y=4-2  
print(y)
```

```
z=4*2  
print(z)
```

```
e=4/2  
print(e)
```

```
a=2  
b=3  
result=a**b  
print(result)
```

```
c=7%2-remainder  
print(c)
```

```
d=7//2-remove float  
print(d)
```

Output:

```
5  
2  
8  
2.0  
8  
1  
3
```

19.2 Assignment Operators:

```
=--> a=b    a=b
+==> a+=b   a=a+b
-==> a-=b   a=a-b
*==> a*=b   a=a*b
/=--> a/=b   a=a/b
%==> a%=b   a=a%b
```

19.3 Comparison Operators:

```
>--- x>y
<--- x<y
==--- x==y
!=--- x!=y
>=--- x>=y
```

19.4 Identity & Membership Operators:

19.4.1 Identity:

```
--> is
--> is not
```

19.4.2 Membership:

Code:

```
db_user=['db_admin','db_conf','db_installation']
random_user="db_admin"
if random_user in db_user:
    print("yes this user is allowed to start db")
else:
    print("this user is not a valid user")
```

Output:

yes this user is allowed to start db

19.5 Logical Operators:

There are three types of logical operators:

```
---> and(both true then true)
---> or(anyone true then true)
---> not(not of true false, not of false true)
```


20. Conditional Statements:

If is called simple conditional statement.

Used to control the execution of set of lines or block of code or one line.

if expression:

 statement1

 statement2

Comparison operators

identity operators

membership operators

logical operators

Code:

```
usr_str=input("Enter your string: ")
```

```
usr_cnf=input("Do you want to covert your string into lower case? say yes or no ? ")
```

```
if usr_cnf=="yes":
```

```
    print(usr_str.lower())
```

Output:

Enter your string: pYThOn

Do you want to covert your string into lower case? say yes or no ? yes

python

Code:

```
my_even_no=[0,2,4,6,8,10]
```

```
usr_num=eval(input('Enter your number: '))
```

```
if usr_num in my_even_no:
```

```
    print("This is my even number")
```

Output:

Enter your number: 8

This is my even number

20.1 IF-ELSE:

Code:

```
usr_str=input("Enter your string: ")
usr_cnf=input("Do you want to convert your given string into title fmt ? say yes or no ? ")

if usr_cnf=="yes":
    print(usr_str.title())
if usr_cnf=="no":
    print(usr_str)
```

Code:

```
usr_str=input("Enter your string: ")
usr_cnf=input("Do you want to convert your given string into title fmt ? say yes or no ? ")

if usr_cnf=="yes":
    print(usr_str.title())
else:
    print(usr_str)
```

Output:

Enter your string: python scripting
Do you want to convert your given string into title fmt ? say yes or no ? yes
Python Scripting

Enter your string: welcome to python
Do you want to convert your given string into title fmt ? say yes or no ? no
welcome to python

Code:

```
a=eval(input("Enter the first number: "))
b=eval(input("Enter the second number: "))

if a > b:
    print(f'{a} is greater than {b}')
elif a < b:
    print(f'{a} is less than {b}')
elif a==b:
    print(f'{a} and {b} is equal')
```

Output:

Enter the first number: 4
Enter the second number: 4
4 and 4 is equal

Code:

```
num=eval(input("Enter your number between 1-10: "))
if num==1:
    print("one")
elif num==2:
    print("two")
elif num==3:
    print("three")
elif num==4:
    print("four")
elif num==5:
    print("five")
elif num==6:
    print("six")
elif num==7:
    print("seven")
elif num==8:
    print("eight")
elif num==9:
    print("nine")
elif num==10:
    print("ten")
elif num not in [1,2,3,4,5,6,7,8,9,10]:
    print("your number not in range please select from 1-10")
```

Output:

```
Enter your number between 1-10: 10
ten
Enter your number between 1-10: 11
your number not in range please select from 1-10
```

Code:

```
num=eval(input("Enter your number between 1-10: "))

if num in [1,2,3,4,5,6,7,8,9,10]:
    if num==1:
        print("one")
    elif num==3:
        print("three")
    elif num==4:
        print("four")
    elif num==5:
        print("five")
    elif num==6:
        print("six")
    elif num==7:
```

```
        print("seven")
    elif num==8:
        print("eight")
    elif num==9:
        print("nine")
    else:
        print("ten")
else:
    print("out of range number please select 1-10 range")
```

Output:

Enter your number between 1-10: 10
ten
Enter your number between 1-10: 23
out of range number please select 1-10 range

Code:

```
num=eval(input("Enter your number between 1-10: "))

num_word={1:'one',2:'two',3:'three',4:'four',5:'five',6:'six',7:'seven',8:'eight',9:'nine',10:'ten'}
if num in [1,2,3,4,5,6,7,8,9,10]:
    print(num_word.get(num))
else:
    print("out of range number please select 1-10 range")
```

Output:

Enter your number between 1-10: 4
four

Enter your number between 1-10: 3
three

Enter your number between 1-10: 11
out of range number please select 1-10 range

21. Python Modules:

A module is a file containing Python definitions and statements. That means, module containing python functions, classes and variables.

21.1 Use of module:

---> Reusability

help("modules")-to get all the default modules.

Code:

```
my_value=46354787980434
```

```
import modules
```

```
print(modules.my_value)
```

Output:

46354787980434

Code:

```
import math
```

```
print(math.pi)
```

Output:

3.141592653589793

Types of Modules:

----> Default Modules

import -default modules name

----> Third Party Modules

pip -is used to get third party modules.

21.2 Method-1:

```
import math
```

```
print(math.pi)
```

```
print(math.pow(3,2))
```

21.3 Method-2:

```
import math as m
```

```
print(m.pi)
```

```
print(m.pow(2,3))
```

21.4 Method-3:

```
from math import *  
print(pi)  
print(pow(4,2))
```

21.5 Method-4:

```
from math import pi,pow  
print(pi)  
print(pow(4,2))
```

21.6 Getpass:

Code:

```
import getpass  
db_pass=getpass.getpass(prompt="Enter your db password: ")  
print(f"The entered password is: {db_pass}")
```

Output:

```
Enter your db password:  
The entered password is: python
```

21.7 SYS Module:

The sys module provides function and variables used to manipulate different parts of the Python runtime environment

```
import sys
```

22. OS Modules:

```
---> os  
---> os.path  
---> os.system()  
---> os.walk()
```

22.1 os.sep:

```
pwd  
/home/khair
```

```
import os  
>>>  
>>> print(os.sep)  
/
```

```
import os
>>> print(os.getcwd())
/home/khair
```

23. Loops(for and while)Modules:

23.1 Read path is a directory or file:

Code:

```
import os
path=input("Enter your path: ")
if os.path.isfile(path):
    print(f"The given path: {path} is a file")
else:
    print("The given path:{path} is a directory")
```

Output:

Enter your path: /home/khair
The given path:{path} is a directory

Enter your path: Desktop/Python-Exercise/
The given path:{path} is a directory

Enter your path: Desktop/Python-Exercise/addition.py
The given path: Desktop/Python-Exercise/addition.py is a file

Code:

```
import os
path=input("Enter your path: ")
if os.path.exists(path):
    print(f"The given path: {path} is a valid")
else:
    print("The given path:{path} is not in this host")
```

Output:

Enter your path: /khair/Python
The given path:{path} is not in this host

Code:

```
import os
path=input("Enter your path: ")
if os.path.exists(path):
    print(f"The given path: {path} is a valid")
    if os.path.isfile(path):
        print(f"The given path: {path} is a file")
    else:
        print(f"The given path:{path} is a directory")
else:
    print(f"The given path:{path} is not in this host")
```

Output:

```
Enter your path: Desktop/Python-Exercise/addition.py
The given path: Desktop/Python-Exercise/addition.py is a valid
The given path: Desktop/Python-Exercise/addition.py is a file
```

23.2 Introduction to the Loops:**Code:**

```
import os
import sys

path=input("Enter your directory path : ")
if os.path.exists(path):
    df_l=os.listdir(path)
    print(df_l)
else:
    print("please provide a valid path")
    sys.exit()

p1=os.path.join(path,df_l[0])
p2=os.path.join(path,df_l[1])

if os.path.isfile(p1):
    print(f"{p1} is a file ")
else:
    print(f"{p1} is a directory")

if os.path.isfile(p2):
    print(f"{p2} is a file ")
else:
    print(f"{p2} is a directory")
```


Output:

Enter your directory path : /home/khair/Desktop/Python-Exercise

```
['indentation_usage.py', 'boolean_result.py', 'usage_of_cmnts.py', 'Test1.py', 'input_output.py',  
'Pynative-Exercise5.py', 'if.py', 'printfunction.py', 'Test3.py', 'loops.readme', 'Pynative-Exercise3.py',  
'basic_String.py', 'Pynative-Exercise4.py', 'Pynative-Exercise2.py', 'binarysearch.py', 'loops.py',  
'string_join_fill.p', 'string_join_fill.py', 'special_chars.py', 'selection', 'tuple.py', 'Hello_word.py', '100  
Days of Code', 'working_with_varaibles.py', 'even.py', 'if_else_two.py', '.git', 'Pynative-Exercise9.py',  
'Pynative-Exercise8.py', 'dict.py', 'type_conversion.py', 'if_else.py', 'README.md', 'Pynative-  
Exercise7.py', 'addition.py', 'count_index_find.py', 'Test2.py', 'Pynative-Exercise-1.py', 'data_types.py',  
'printfunction.readme', 'oop.py', 'list.py', 'string_practise.py', 'memarship.py',  
'working_multiple_variables.py', 'Pynative-Exercise6.py', 'sets.py']  
/home/khair/Desktop/Python-Exercise/indentation_usage.py is a file  
/home/khair/Desktop/Python-Exercise/boolean_result.py is a file
```

Enter your directory path : /Desktop/Python-Exercise/100\ Days\ of\ Code/
please provide a valid path

23.3 Loops:

Code:

```
my_list=[4,5,6,7,8,5,3,4,9]  
for each_value in [4,5,6,7,8,5,3,4,9]:  
    print(each_value)
```

or

```
my_list=[4,5,6,7,8,5,3,4,9]  
for each_value in my_list:  
    print(each_value)
```

Output:

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

23.4 Two types of loops:

---> while loop

---> for loop

The for loop in python is used to iterate over a sequence(list, tuple, string) or other iterable objects.

23.5 List-Code:

Code:

```
for x in [4,5,6]:  
    print('*****')
```

Output:

```
*****  
*****  
*****
```

Code:

```
my_list=[3,4,34,5,67,89,23]
```

```
for each in my_list:  
    rem=each%2  
    if rem==0:  
        print(f"{each} is even")  
    else:  
        print(f"{each} is odd")
```

Output:

```
3 is odd  
4 is even  
34 is even  
5 is odd  
67 is odd  
89 is odd  
23 is odd
```

23.6 Tuple-Code:

Code:

```
for value in (4,5,7,"Hi"):  
    print('*****')
```

Output:

```
*****  
*****  
*****  
*****
```

23.7 String-Code:

Code:

```
for each_char in "python":  
    print("*****",each_char)
```

Output:

```
***** p  
***** y  
***** t  
***** h  
***** o  
***** n
```

Code:

```
usr_str=input("Enter your string: ")  
index=0  
for each_char in usr_str:  
    print(f'{each_char}-->{index}')  
    index+=1
```

Output:

```
Enter your string: python  
p-->0  
y-->1  
t-->2  
h-->3  
o-->4  
n-->5
```

23.8 Python range function:

```
range()  
---> Built-in function  
---> Genarates integars as a list
```

Code:

```
print(list(range(5)))
```

Output:

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

23.9 Syntax:

range(start,stop,step)
by default start=0, step=1

Code:

```
print(list(range(0,10,2)))
```

Output:

```
[0, 2, 4, 6, 8]
```

Code:

```
my_list=[5,6,7,34,"Python"]  
  
for each_index in range(len(my_list)):  
    print(f'Index-->{each_index}: value-->{my_list[each_index]}')
```

Output:

```
Index-->0: value-->5  
Index-->1: value-->6  
Index-->2: value-->7  
Index-->3: value-->34  
Index-->4: value-->Python
```

23.10 For Loops string,list,tuple,dictionary:

Code:

```
my_list=[4,5,6,7,8,5,3,4,9]  
for each_value in my_list:  
    print(each_value)  
  
my_string="python scripting is easy"  
for each_char in my_string:  
    print(each_char)  
  
my_list=[(1,2),(3,5),(7,9)]  
for each_item in my_list:  
    print(each_item)  
  
my_list=[(1,2),(3,5),(7,9)]  
for f,s in my_list:  
    print(f)
```

```
my_dic={'a':1, 'b':2, 'c':3}
for each in my_dic.items():
    print(each)

my_dic={'a':1, 'b':2, 'c':3}
for keys,values in my_dic.items():
    print(keys,values)
```

Output:

4

5

6

7

8

5

3

4

9

p

y

t

h

o

n

s

c

r

i

p

t

i

n

g

i

s

e

a

s

y

(1, 2)

(3, 5)

(7, 9)

1

3

7

```
('a', 1)
('b', 2)
('c', 3)
a 1
b 2
c 3
```

23.11 While:

---> The while loop in python is used to iterate over a block of code as long as the test expression(condition) is true.

---> We generally use this loop when we don't know beforehand, the number of times to iterate.

Code:

```
value=4
while value<=6789:
    print(value)
    value=value+456
```

Output:

```
4
460
916
1372
1828
2284
2740
3196
3652
4108
4564
5020
5476
5932
6388
```

Code:

```
cnt=1
while cnt<=5:
    print("hello")
    cnt=cnt+1
```

Output:

hello
hello
hello
hello
hello

23.12 While loop control statement:**Code:**

```
for each in [3,4,56,7,8]:  
    print(each)  
    if each==56:  
        break  
print("after loop")
```

Output:

3
4
56
after loop

Code:

```
for each in range(1,11):  
    if each==7:  
        continue  
    print(each)
```

Output:

1
2
3
4
5
6
8
9
10

24. Working with text file using python:

```
open---> w
      ---> a
      ---> r
# Creation of empty file

fo=open('newfile.txt','w')
fo.close()
```

Code:

```
fo=open('demo.txt','w')
print(fo.mode)
print(fo.readable())
print(fo.writable())
fo.write("This a first line\n")
fo.write("This a second line")
fo.close()
```

Output:

```
w
False
True
```

Also create a text file with text in 2 line

24.1 how copy content from one file to another:

Code:

```
sfile="demo.txt"
dfile="newdemo.txt"

sfo=open(sfile,'r')
content=sfo.read()
sfo.close()

dfo=open(dfile,'w')
dfo.write(content)
dfo.close()
```

Output:

demo file content copy in newdemo

25. Working with JSON file:

JSON(Javascript Object Notation) is a popular data format used for representing structured data.

```
import json
```

```
req_file="myjson.json"
```

```
fo=open(req_file,'r')
```

```
print(json.load(fo))----> json.load(fo) is used to conver json data into dictionary.
```

```
fo.close()
```

26. Introduction to functions:

---> A function is a block of code for some specific operation.

---> Function code is re-usable

---> It will execute only when it is called

26.1 Function uses:

---> Code Resability

----> Improve Modularity

Code:

```
import os
```

```
import time
```

```
import platform
```

```
def mycode(cmd1,cmd2):
```

```
    print("Please wait. Cleaning the screen....")
```

```
    time.sleep(2)
```

```
    os.system(cmd1)
```

```
    print("Please wait finding the list of dir and files")
```

```
    time.sleep(2)
```

```
    os.system(cmd2)
```

```
if platform.system()=="Windows":
```

```
    mycode("cls", "dir")
```

```
else:
```

```
    mycode('clear', 'ls -lrt')
```

Output:

Please wait finding the list of dir and files

/home/khair

26.2 How to define a function:

```
def display():
```

```
display()
```

Code:

Simple code:

```
print("Welcome to python")
print("python Scripting")
print("We are good with basics")
print("we know how to write function")
print("Function is easy to write")
print("Function is easy to graps")

def welcome_msg():
    print("Welcome to python")
    print("python Scripting")
    return None
def known_concepts():
    print("We are good with basics")
    print("we know how to write function")
    return None
def new_concepts():
    print("Function is easy to write")
    print("Function is easy to graps")
    return None

welcome_msg()
known_concepts()
new_concepts()
```

Output:

```
Welcome to python
python Scripting
We are good with basics
we know how to write function
Function is easy to write
Function is easy to graps
```

26.3 Functions with arguments:

Code:

```
def get_result():
    result=num+10
    print(f'Your result is: {result}')
    return None
num=eval(input("Enter your number: "))
get_result()
```

Output:

Enter your number: 20
Your result is: 30

Code:

```
def get_result(num):
    result=num+10
    print(f'Your result is: {result}')
    return None
def main():
    num=eval(input("Enter your number: "))
    get_result(num)
    return None
main()
```

Output:

Enter your number: 20
Your result is: 30

Code:

```
def get_result(value): # parameters/positional arguments
    result=value+10
    print(f'Your result is: {result}')
    return None
def main():
    num=eval(input("Enter your number: "))
    get_result(num) #Arguments
    return None
main()
```

Output:

Enter your number: 30
Your result is: 40

Code:

```
def get_add(a,b):
    aresult=a+b
    print(f'The addition of {a} and {b} is: {areult}')
    return None
def get_sub(a,b):
    sresult=a-b
    print(f'The sub of {a} and {b} is: {sresult}')
    return None
def main():
    a=eval(input("Enter your number: "))
    b=eval(input("Enter your number: "))
    get_add(a,b)
    get_sub(b,a)
    return None
main()
```

Output:

```
Enter your number: 4
Enter your number: 7
The addition of 4 and 7 is: 11
The sub of 7 and 4 is: 3
```

26.4 Functions with arguments & Return value:**Code:**

```
def get_add(a,b):
    aresult=a+b
    return aresult

def main():
    a=eval(input("Enter your number: "))
    b=eval(input("Enter your number: "))
    aresult=get_add(a,b)
    print(f'The addition of {a} and {b} is: {areult}')
    return None
main()
```

Output:

```
Enter your number: 4
Enter your number: 8
The addition of 4 and 8 is: 12
```

Code:

```
def display(**karg):  
    print(karg)  
    return None  
  
display(b=5,a=4)  
display(a=4,b=5,c=6)  
display(x=5,y="Hi",z=6.7,user="root")
```

Output:

```
{'b': 5, 'a': 4}  
{'a': 4, 'b': 5, 'c': 6}  
{'x': 5, 'y': 'Hi', 'z': 6.7, 'user': 'root'}
```

26.5 How to use one script in another:**Code(script):**

```
def addtion(a,b):  
    print(f"The addition of {a} and {b} is {a+b}")  
    return None  
def sub(a,b):  
    print(f"The sub of {a} and {b} is {a-b}")  
    return None  
  
x=7  
y=4  
addition(x,y)  
sub(x,y)
```

Output:

```
The addition of 7 and 4 is 11  
The sub of 7 and 4 is 3
```

Code:

```
import script  
  
def mul(a,b):  
    print(f"The mul of {a} and {b} is {a*b}")  
    return None  
  
x=20  
y=10  
script.addition(x,y)  
script.sub(x,y)  
mul(x,y)
```

Output:

The addition of 20 and 10 is 30

The sub of 20 and 10 is 10

The mul of 20 and 10 is 200

Code(script):

```
def addition(a,b):
    print(f"The addition of {a} and {b} is {a+b}")
    return None
def sub(a,b):
    print(f"The sub of {a} and {b} is {a-b}")
    return None
def main():
    x=7
    y=4
    addition(x,y)
    sub(x,y)
    return None
if __name__=="__main__":
    main()
```

Output:

The addition of 7 and 4 is 11

The sub of 7 and 4 is 3

Code(script2):

```
import script

def mul(a,b):
    print(f"The mul of {a} and {b} is {a*b}")
    return None

def main():
    x=20
    y=10
    mul(x,y)
    script.addition(x,y)
    script.sub(x,y)
    return None
if __name__=="__main__": ## donot show script output which imported, beacuse if you import script
__name__==script so it will not call scirpt one main function so in script two script output will not
show.
    main()
```

Output:

The mul of 20 and 10 is 200
The addition of 20 and 10 is 30
The sub of 20 and 10 is 10

27. Working with paramiko:

----> Paramiko module used to work with remote server

----> Server need to have SSHV2 protocol to work with paramiko

----> Paramiko module will create a SSH Client and by using this it will connect to remote server and execute commands.

Two ways to connect with remote server:

----> Using username and password

----> using username and cryptographic key

pip install paramiko-If not present.

```
import paramiko
```

Code:

```
#!/usr/bin/python3
import paramiko
ssh = paramiko.SSHClient()
ssh.set_missing_host_key_policy(paramiko.AutoPolicy())
ssh.connect(hostname='hostname/IP',username='ec2-user',password='paramiko123',port=22)
stdin, stdout, stderr = ssh.exec_command('free -m')
print(stdout.read())
```

28.Introduction to OOps:

---> OOps is a combination of class and object.

---> To create a template/Blueprint.

Code:

```
class emp:
    count=0
    def get_name_age_salary(self,name,age,salary):
        self.name=name
        self.age=age
        self.salary=salary
    def increase_count_for_emp(self):
        emp.count=emp.count+1
        return None
    def display_details(self):
        print(f'The name is: {self.name}\nThe age is: {self.age}\nThe salary is: {self.salary}')
        return None

emp1=emp()
emp2=emp()

emp1.get_name_age_salary('Jhon',34,45000)
emp1.increase_count_for_emp()
emp2.get_name_age_salary('Adnan',36,54000)
emp2.increase_count_for_emp()

print(emp.count)
```

Output:

2

28.1 Constructor of a Class:

Code:

```
class Emp:
    count=0
    def __init__(self):
        Emp.count=Emp.count+1
        return None
    def display(self):
        print("This is a display method")
        return None

emp1=Emp()
```



```
emp2=Emp()

print('The number of objects created from Emp class are:',Emp.count)
```

Output:

The number of objects created from Emp class are: 2

Code:

---> You can create your attribute while creating your object

```
class Emp(object):
    def __init__(self,name,salary):
        self.name=name
        self.salary=salary
        return None
    def display(self):
        print(f'This name is: {self.name}\nThe salary is: {self.salary}')
        return None
```

```
emp1=Emp('Ramu',56000)
emp2=Emp("Naren",90000)
```

```
emp1.display()
emp2.display()
```

Output:

```
This name is: Ramu
The salary is: 56000
This name is: Naren
The salary is: 90000
```

28.2 Inheritance from one class:

Code:

```
class Tomcat(object):-----Tomcat Super Classs
    def __init__(self,home,ver):
        self.home=home
        self.version=ver
        return None
    def display(self):
        print(self.home)
        print(self.version)
        return None
class Apache(Tomcat):----Apache Sub Class
    def __init__(self,home,ver):
```

```
        self.home=home
        self.version=ver
        return None
tom_ob=Tomcat('/home/tomcat9','7.6')
apa_ob=Apache("/etc/httpd",'2.4')

tom_ob.display()
apa_ob.display()
```

Output:

```
/home/tomcat9
7.6
/etc/httpd
2.4
```