MDIGITZ SOFT SOLUTIONS

PYTHON

What is python

- Python is a high-level programming language.
- It was created by Guido van Rossum, and released in 1991.
- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has simple compared to other languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written.

It is widely used for:

- Web development (server-side).
- Software development.
- Mathematics.
- System scripting.

We can create different types of applications using python

- Web and Internet Development
- Games
- Scientific and computational applications
- Language development
- Operating systems
- GUI-based desktop application*336

Features:

- Easy to learn
- Interpreted language:

If you are familiar with any languages like C++, java. you must first combine it and run it. but in Python, there is no need to compile. It executed line by line and not at all

- Object-oriented
- Dynamically typed programming language
- Free and open source

eg:

```
print(“hello world”)
```

Comments:

Single-line comment:

Using ‘#’ symbol

Eg:

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

print("Cheers, Mate!")
```

Multiline comment:

Eg:

```
 """x = 5

 y = "John"

  print(x)

  print(y)"""
```

Variables:

Variables are containers for storing data values.

Eg:

```
x = 5
 y = "John"
 print(x)

 print(y)
```

Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-Z, 0-9, and _)
- Variable names are case-sensitive (age, Age, and AGE are three different variables)
- A variable name cannot be any of the Python keywords

Eg:

```
myvar = "John"

 my_var = "John"

 _my_var = "John"

 myVar = "John"

 MYVAR = "John"

 myvar2 = "John"
```

Casting:

If you want to specify the data type of a variable, this can be done with casting.

Eg:

```
x = str(3)     # x will be '3'
y = int(3)     # y will be 3
z = float(3)   # z will be 3.0
```

Type():

To define the values of various data types and check their data types we use the type function

Eg:

```
x = 5
y = "John"
print(type(x))
print(type(y))
```

Output:

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

Multiple Values into multiple variables:

Eg:

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

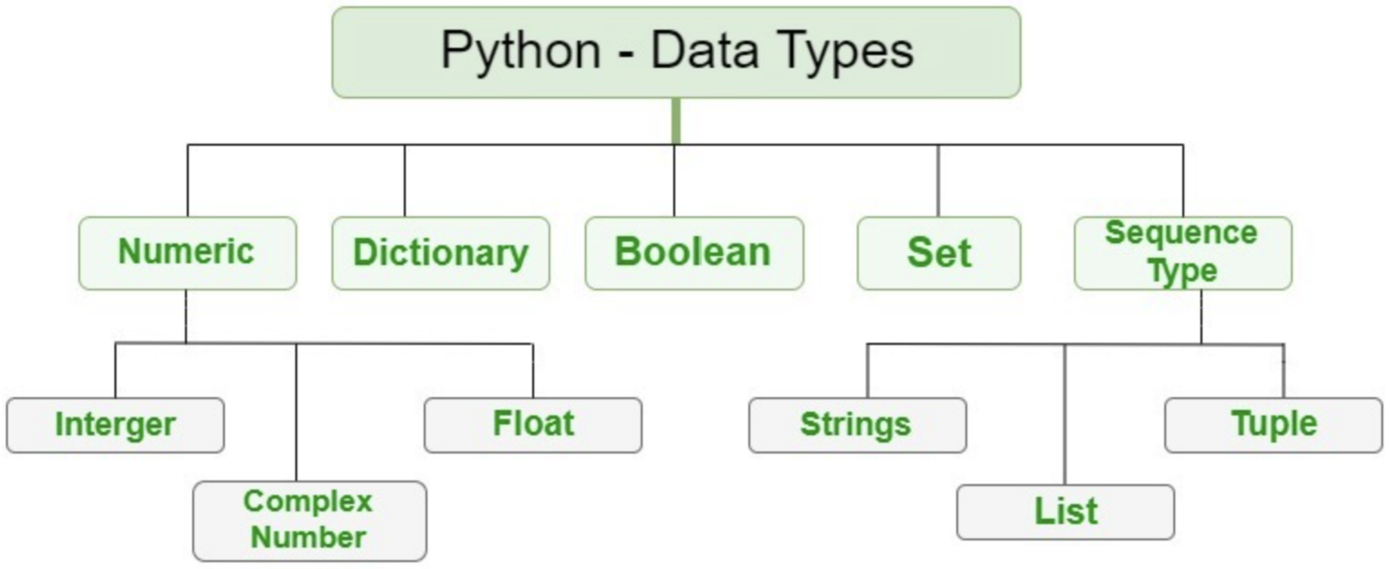
Single values into multiple variables:

Eg:

```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

Data Types:

- **Sequence Type**
- **Boolean**
- **Set**
- **Dictionary**
- **Numeric**



1)Sequence data type:

The sequence Data Type in Python is the ordered collection of similar or different data types.

Types of sequence type:

- **Strings**
- **List**
- **tuple**

Strings:

A string is a collection of one or more characters put in a single quote, double-quote, or triple-quote.

Eg1:

```
print("Hello")
print('Hello')
```

List:

which is an ordered collection of data.

Eg:

L=[“apple”,” orange”,” banana”]

Tuple:

Tuple is also an ordered collection of Python objects.

Tuples are immutable i.e. tuples cannot be modified after it is created.

Eg:

A=(0, 1, 2, 3)

2)Boolean:

Data type with one of the two built-in values, True or False

Eg1:

```
x = bool(5)
print(x)
```

Output:

```
True
```

Eg2:

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

3)Set:

In Python, a Set is an unordered collection of data types that is iterable, mutable, and has no duplicate elements. The order of elements in a set is undefined though it may consist of various elements.

Eg:

```
{'banana', 'apple', 'cherry'}
```

4)Dictionary:

It holds only a single value as an element, a Dictionary holds a key: value pair. Each key-value pair in a Dictionary is separated by a colon: .whereas each key is separated by a ‘comma. the dictionary can also be created by the built-in function dict(). An empty dictionary can be created by just placing it in curly braces{}.

Eg:

```
{'name': 'John', 'age': 36}
```

5)Numeric:

Int

Float

Complex

Eg:

```
x = 1     # int
y = 2.8   # float
z = 1j    # complex
```

Python Operators

Operators are used to perform operations on variables and values.

Types of operators:

- **Arithmetic operators**
- **Assignment operators**
- **Comparison operators**
- **Logical operators**
- **Identity operators**

1)Arithmetic Operators:

Arithmetic operators are used with numeric values to perform common mathematical operations.

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

2)Assignment Operators:

Assignment operators are used to assign values to variables:

3) Comparison Operators:

Comparison operators are used to compare two values:

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

4)Logical Operators:

Logical operators are used to combine conditional statements:

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

5)Identity Operators

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

Q)Write a program to add two numbers?

```
a=10
b=20
print("the sum is",a+b)
```

Q)Write a program to swap two values without using a third variable?

```
x,y=int(input("enter the first number")),int(input("enter the second number"))
x,y=y,x
```

Conditional statement:

- If statement
- If else
- If-elif-else ladder

- ### If statement:

The if statement is a conditional statement in Python, that is used to determine whether a block of code will be executed or not. Meaning if the program finds the condition defined in the if statement to be true, it will go ahead and execute the code block inside the if statement.

Eg:

```
a = 33
b = 200
if b > a:
print("b is greater than a")
```

- ### if -else:

if statement executes the code block when the condition is true. Similarly, the else statement works in conjuncture with the if statement to execute a code block when the defined if the condition is false.

Eg:

```
 a = 33
  b = 33
  if a> b:
print("a is greater than b")
else b> a:
print("b is greater than a")
```

- **If-elif-else**

The elif statement is used to check for multiple conditions and execute the code block within if any of the conditions evaluate to be true.

Eg:

```
a = 200
b = 33
if b > a:
print("b is greater than a")
elif a == b:
print("a and b are equal")
else:
print("a is greater than b")
```

Nested if statement:

A nested *if* statement is considered as *if* within another *if* statement(s).

Eg:

```
x = 41

if x > 10:
 print("Above ten,")
if x > 20:
  print("and also above 20!")
else:
   print("but not above 20.")
```

Programs:

Q)write the program largest of two numbers?

```
n1,n2=int(input(“Enter the first number”)),
      int(input(“Enter the first number”))
If n1>n2:
   print(“The largest is”,n1)
Else:
   print(“the largest”,n2)
```

Q)Write a program to print the numbers are even or odd

```
num1=int(input(“enter the number:”)
If num%2==0:
   print(“the given number is even”)
Else:
   print(“the given number is odd”)
```

Pass Statement:

If statements cannot be empty, but if you for some reason have an if statement with no content, put in the **pass** statement to avoid getting an error.

Eg:

```
  a = 33
 b = 200
if b > a:
  Pass
```

Loops:

Python has two primitive loop commands:

- **while** loops
- **for** loops

While loop:

while loop we can execute a set of statements as long as a condition is true.

Syntax:

```
while expression:

statement(s)
```

Eg:

```
i = 1

while i < 6:
```

```
print(i)

i += 1
```

Q)Write a program to print the first 10 nature numbers?

```
print(“The nature number”)

i=1

While i<=10:

     print(i)

     I++
```

Q)To print odd numbers up to N?

```
n=int(“input(“enter the limit”))

print(“The odd number is:”)

i=1

While i<=n;

 If i%2==1;

   print(i)

i=i+1
```

Break statement:

In the break statement, we can stop the loop even if the while condition is true.

Eg:

```
i = 1
while i < 6:
 print(i)
  if i == 3:
    break
 i += 1
Output:
   1
   2
   3
```

Continue statement:

In the continue statement, we can stop the current iteration, and continue with the next.

Eg:

```
i = 0
while i < 6:
  i += 1
  if i == 3:
  continue
```

```
    print(i)
```

```
Output:

    1

    2

    4

    5

    6
```

FOR LOOP:

for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string.

Syntax:

```
for i in L:
    Statements
```

Eg:

```
L=[“red”,”yellow”,”blue”]
for i in L:
        print(i)
```

Output:

```
Red

Yellow

Blue
```

Eg:

```
For x in “WELCOME”

        print(x)
```

Output:

```
W

E

L

C

O

M

E
```

Q)Write a program to find sum of values in a list?

```
 L=[12,33,45,67]
S=0
For i in L:
s=s+i;
print("sum is:",s)
```

OUTPUT:

```
Sum is :157
```

Range():

To loop through a set of code a specified number of times, we can use the range() function.The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

Syntax:

```
Range(start,stop,step)
```

Eg:

```
For i in range(10):
      print(i)
```

Output:

```
0

1

2

3

4

5

6

7

8

9
```

Syntax2:

For i in range(2,10):

Its means starting with 2,ending with (n-1) 10

Output:

```
2

3

4

5

6

7

8
```

```
9
```

Syntax3:

```
For i in range(2,10,3):
     print(i)
Starting with 2
Ending with 10
Increment by 3
```

Output:

```
2
5
8
```

Q)To print even number using for loop?

```
n=int(input("Enter the limit"))
print("even number are:")
For i in range(1,n+1)
     if(i%2==0)
```

```
            print(i)
```

Output:

```
    Enter the limit:10
    Even numbers are:
          2
          4
          6
          8
         10
```

Q)Write a program to find factorial of a number?

```
     n=int(input("Enter the limit"))
    f=1
   For i in range(1,n+1):
         f=f*i
  print("factorial is",f)
```

Output:

```
   Enter the limit:5
   Factorial is:120
```

Q)write a program to print fibonacci series?

```
n=int(input("Enter the limit")
a,b=0,1
print("The fibnocci is")
print(a)
print(b)
For i in range(3,n+1)
                q=a+b
                 print(q)
                 a,b=b,q
```

Output:

```
Enter the limit:5
The fibonacci is
         0
         1
         1
         2
         3
```

Q)To print multiplication table 5

```
n=int(input("enter the number"))
For i in range(1,11)
```

```
print(i,"*",n"="i*n)
```

Output:

```
Enter the number:5

Multiplication table

            1*5=5

           2*5=10

           3*5=15

           4*5=20

           5*5=25

           6*5=30

          7*5=35

          8*5=40

          9*5=45

         10*5=50
```

Else in For Loop:

The **else** keyword in a **for** loop specifies a block of code to be executed when the loop is finished

Eg:

```
  for x in range(6):
  print(x)
  else:
print("Finally finished!")
```

OUTPUT:

```
0
1
2
3
4
5
Finally finished!
```

Nested for loop:

A nested loop is a loop inside a loop.

Eg:

```python
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
    for y in fruits:
        print(x, y)
```

Output:

```
red apple
red banana
red cherry
big apple
big banana
big cherry
tasty apple
tasty banana
tasty cherry
```

Q) To print first 14 even numbers in the reverse order for loop?

```
    print(“even numbers is”)
    For i in range(14,0,-2):
        print(i)
Output:
     even number is
       14
       12
       10
       8
       6
       4
       2
```

String Slicing:

Used for getting substring.

Syntax:

```
[start:stop:step]
```

Eg:

```
b = "Hello, World!"
print(b[2:5])
```

Output:

```
llo
```

```
str=”WELCOME”
print(str[:3])
```

Start with 0
Stop with 3
Increment by 1

Output:

```
   WEL
```

```
str=" WELCOME"
print(str[4:])
```

Start with 4
Stop with 0
Increment by1

Output:

```
 OME
```

```
str="Hello world"
print(str[2:5]
```

Output:

```
 llo
```

Negative Indexing:

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

Eg:

```
  b = "Hello, World"
  print(b[-5:-2])
```

Output:

```
  Wor
```

```
  str="Welcome"
  print(str[-1:-12:-2])
```

Output:

```
  Eolw
```

```
  a="hello world"
```

```
print(a[: :-1])
```

Output:

```
dlrow olleh
```

Python Collections (Arrays)

- **LIST** is a collection that is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection that is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection that is unordered, unchangeable*, and unindexed. No duplicate members.
- **dictionary** is a collection that is ordered** and changeable. No duplicate members.

Python list:

- Lists are used to store multiple items in a single variable.
- It is represented by a square bracket
- List are mutable(add remove the list)
- It is an ordered collection
- The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.
- Allow duplicate members
- indexing

Eg:

```
L=["apple", "banana", "cherry"]
```

Access Items:

We can access an element from the list using an index

Eg:

```
  L=["red","green","violet","blue","pink"]
  print(L[1])
```

Output:

```
     green
```

Negative Indexing:

Negative indexing means start from the end
1 refers to the last item, -2 refers to the second last item etc

Eg:

```
 thislist = ["apple", "banana", "cherry"]
  print(thislist[-1])
```

Output:

```
   Cherry
```

Change Item Value:

To change the value of a specific item, refer to the index number

Eg:

```
list = ["apple", "banana", "cherry"]
list[1] = "blackcurrant"
```

```
print(list)
```

Output:

```
['apple', 'blackcurrant', 'cherry']
```

Check if an item exists:

```
L=[“red”,”pink”,”black”]
If ‘pink’ in L:
    print(“found”)
else:
   print(“not found”)
```

OUTPUT:

```
Found
```

Length of list:

Using len()

Eg:

```
L=[“pink”,”black”,”red”]
print(“length is:”,len(L))
```

Output:

```
Length is:3
```

Different functions in the list:

2methods for adding items in the list

1)append()
2)insert()

1)append:

To add an item to the end of the list, use the append() method

Eg:

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

Output:

```
['apple', 'banana', 'cherry', 'orange']
```

2)insert()

To insert a list item at a specified index, use the insert() method.The insert() method inserts an item at the specified index:

Eg:

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange")
 print(thislist)
```

Output:

```
['apple', 'orange', 'banana', 'cherry']
```

Items to delete:

1)remove()
2)pop()

1)remove()

The remove() method removes the specified item.

Eg:

```
list = ["apple", "banana", "cherry"]
list.remove("banana")
 print(list)
```

Output:

```
['apple', 'cherry']
```

Eg2:

```
L=[“red”,”green”,”blue”,”red”]
L.remove(“red”)
print(L)
```

Output:

```
[“green”,”blue”,”red”]
```

2)pop()

The pop() method removes the specified index.

Eg:

```
list = ["apple", "banana", "cherry"]
```

```
list.pop(1)
print(list)
```

Output:

['apple', 'cherry']

If you do not specify the index, the `pop()` method removes the last item.

Eg:
list = ["apple", "banana", "cherry"]
list.pop()
print(list)

Output:

```
['apple', 'banana']
```

Clear():

The clear() removes all the elements from the dictionary.The `clear()` method empties the list.The list still remains, but it has no content.

Eg:

```
list = ["apple", "banana", "cherry"]
list.clear()
print(list)
```

Output:

```
[]
```

Delete:

The del keyword also removes the specified index

Eg:

```
list = ["apple", "banana", "cherry"]
del thislist[0]
print(list)
```

OUTPUT:

```
['banana', 'cherry']
```

```
L1=[1,2,3,4,5]

L2=[6,7,8,9,10]

L1.append(L2)

print(L1)
```

Extend:

Extend method add the specified list elements to the end of the list

Eg:

```
L1=[1,2,3,4,5]

L2=[6,7,8,9,10]
```

```
L1.extend(L2)

print(L1)

Output:

 [1,2,3,4,5,6,7,8,9,10]
```

```
L1=[1,2,3,4,5]

L2=[6,,7,8,9,10]

L1.append(L2)

print(L1)

print(L1[5])
```

```
OUTPUT:

[6,7,8,9,10]
```

```
L1=[1,2,3,4,5]

L2=[6,,7,8,9,10]

L1.append(L2)

print(L1)
```

print(L1[5][2])

OUTPUT:

8

SORT:

The `sort()` method sorts the list ascending or descending order.By default is ascending

L1=[100,2,48,7,1]

L1.sort()

Output:

[1,2,7,48,100]

L=[“red”,”orange”,”Black”,”blue”]

Output

[“Black”,”blue’,”orange”,”red”’]

Sort list in descending order:

To sort descending, use the keyword argument `reverse = True`

```
L = [100,48,70,200]

L.sort(reverse=True)

print(“L”)
```

OUTPUT:

```
[200,100,70,48]
```

count()

Returns the number of elements with the specified value

Eg:

```
  L=[“red”,”white”,”green”,”red”]
c=L.count(“red”)
print(c)
```

Output:

```
   2
```

COPY()

EG:

```
L1=[1,2,3,4,5]
L2=L1.copy()
print(L2)
```

OUTPUT:

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

```
L=[1,2,3,4,5]

print(max(L))

print(min(L))

Print(sum(L))
```

Strings are Arrays:

Using the index we can access the character in the array

Eg:

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

Output:

```
e
```

String Length:

To get the length of a string, use the `len()` function.

Output:

```
 a = "Hello, World!"
print(len(a))
```

Output:

```
13
```

Eg: `thislist = ["apple", "banana", "cherry"]`

```
print(len(thislist))
```

```
Output:
  3
```

A list can contain different data types:

Eg:

```
list1 = ["abc", 34, True, 40, "male"]
```

```
  Output:

      ['abc', 34, True, 40, 'male']
```

Range of Indexes:

You can specify a range of indexes by specifying where to start and where to end the range.

Eg:

```
 thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])
```

Output:

```
  ['cherry', 'orange', 'kiwi']
```

Eg:

```
 list = ["apple", "banana", "cherry", "orange", "kiwi", "melon",
"mango"]
  print(thislist[:4])
```

```
Output:

['apple', 'banana', 'cherry', 'orange']
```

```
list = ["apple", "banana", "cherry", "orange", "kiwi", "melon","mango"]
print(thislist[4:])
```

Output:

```
['kiwi', 'melon', 'mango']
```

Reverse Order:

The `reverse()` method reverses the current sorting order of the elements.

Eg:

```
thislist = [ "Orange", "Kiwi", "cherry"]
thislist.reverse()
print(thislist)
```

Output:

```
['cherry', 'Kiwi', 'Orange']
```

Join Lists:

join Two Lists:

There are several ways to join, or concatenate, two or more lists in Python. using the + operator.

Eg:

```
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]
list3 = list1 + list2
print(list3)
```

OUTPUT:
['a', 'b', 'c', 1, 2, 3]

```
   L=[“red”,”white”,”blue”,”yellolw”,”black”]

  output=[]
  For i in L:
    If ‘e’  in i:
        output.append(i)
  print(output)
```

 Output:
   [“red”,”white”,”blue”,”yellow”]

Tuple:

Tuples are used to store multiple items in a single variable.Tuple items are ordered, unchangeable, and allow duplicate values(we cannot change, add or remove items after the tuple has been created.). items are indexed, the first item has index `[0]`, the second item has index `[1]`.**`unchangeable means`** we cannot change, add or remove items after the tuple has been created.

Tuples allow duplicate values:

Eg:

```
t = ("apple", "banana", "cherry", "apple", "cherry")
print(t)
```

Output:

```
    ('apple', 'banana', 'cherry', 'apple', 'cherry')
```

Tuple Length:

use the `len()` function

Eg:
```
thistuple = ("apple", "banana", "cherry")
print(len(thistuple))
```

Output:
```
3
```

Eg:
```
tuple1 = ("abc", 34, True, 40, "male")
```

Output:
```
('abc', 34, True, 40, 'male')
```

Access Tuple Items:

Access tuple items by referring to the index number, inside square brackets

Eg:
```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

Output:
```
Banana
```

Negative Indexing:

Negative indexing means start from the end.`-1` refers to the last item, `-2` refers to the second last item etc.

Eg:
```
thistuple = ("apple", "banana", "cherry")
print(thistuple[-1])
```

```
Output:
 Cherry
```

Range of Indexes:

Eg:

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:5])
```

```
Output:
 ('cherry', 'orange', 'kiwi')
```

Eg:

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[:4])
```

```
Output:
 ('apple', 'banana', 'cherry', 'orange')
```

Eg:

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:])
```

```
Output:
('cherry', 'orange', 'kiwi', 'melon', 'mango')
```

Range of Negative Indexes:

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
```

```
print(thistuple[-4:-1])
```

```
Output:
('orange', 'kiwi', 'melon')
```

Change Tuple Values:

tuple is created, you cannot change its values. Tuples are unchangeable, or immutable as it also is called.You can convert the tuple into a list, change the list, and convert the list back into a tuple

```
Eg:
  x = ("apple", "banana", "cherry")
  y = list(x)
  y[1] = "kiwi"
  x = tuple(y)

  print(x)
```

```
Output:

 ("apple", "kiwi", "cherry")
```

Add Items:

Convert the tuple into a list, add "orange", and convert it back into a tuple

```
Eg:
 L = ("apple", "banana", "cherry")
 y = list(thistuple)
 y.append("orange")
 L = tuple(y)
```

OUTPUT:

```
('apple', 'banana', 'cherry', 'orange')
```

Remove Items:

Tuples are unchangeable, so you cannot remove items from it, but you can use the same workaround as we used for changing and adding tuple items

Eg:

```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.remove("apple")
 thistuple = tuple(y)
```

Output:

```
('banana', 'cherry')
```

The del keyword can delete the tuple completely

Eg:

```
thistuple = ("apple", "banana", "cherry")
del thistuple
print(thistuple) #this will raise an error because the tuple no longer exists
```

Join Tuples:

To join two or more tuples you can use the + operator

Eg:

```
tuple1 = ("a", "b" , "c")
```

```
tuple2 = (1, 2, 3)
tuple3 = tuple1 + tuple2
print(tuple3)
```

Output:

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

Set:

A set is a collection which is *unordered*, *unchangeable*, and *unindexed*.

Eg:

```
thisset = {"apple", "banana", "cherry"}
print(thisset)
```

Output:

```
{'cherry', 'apple', 'banana'}
```

Duplicates Not Allowed:

```
thisset = {"apple", "banana", "cherry", "apple"}
print(thisset)
```

Output:

```
{'banana', 'cherry', 'apple'}
```

Add Items:

To add one item to a set use the `add()` method.

Eg:

```
 thisset = {"apple", "banana", "cherry"}
thisset.add("orange")
print(thisset)
```

Output:

```
{'cherry', 'apple', 'orange', 'banana'}
```

Add Sets:

To add items from another set into the current set, use the `update()` method.

Eg:

```
thisset = {"apple", "banana", "cherry"}
tropical = {"pineapple", "mango", "papaya"}
thisset.update(tropical)
print(thisset)
```

Output:

```
{'apple', 'mango', 'cherry', 'pineapple', 'banana', 'papaya'}
```

Remove Item:

To remove an item in a set, use the `remove()`, or the `discard()` method

Eg:

```
  thisset = {"apple", "banana", "cherry"}

thisset.remove("banana")

print(thisset)
```

Output:

```
{'cherry', 'apple'}
```

clear():

The clear() method empties the set

Eg:
```
thisset = {"apple", "banana", "cherry"}
thisset.clear()
print(thisset)
```

Output:
```
set()
```

The *del* keyword will delete the set completely

eg:
```
thisset = {"apple", "banana", "cherry"}
del thisset
print(thisset)
```

Loop Items:

You can loop through the set items by using a `for` loop:

Eg:
```
thisset = {"apple", "banana", "cherry"}
for x in thisset:
 print(x)
```

Output:
```
cherry
apple
Banana
```

Join Sets:

There are several ways to join two or more sets in Python.You can use the `union()` method that returns a new set containing all items from both sets. the `update()` method that inserts all the items from one set into another

```
union():
    set1 = {"a", "b" , "c"}
  set2 = {1, 2, 3}

set3 = set1.union(set2)
print(set3)


Output:
{'a', 1, 'b', 2, 'c', 3}

update():

set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set1.update(set2)
print(set1)

   Output:
    {'a', 2, 'c', 3, 1, 'b'}
```

Dictionaries:

Dictionaries are used to store data values in key:value pairs.A dictionary is a collection which is ordered*, changeable and do not allow duplicates.It represented clearly braces

```
Eg:
 thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)

Output:
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

Print the "brand" value of the dictionary:

```
Eg:
 thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict["brand"])

Output:
  Ford
```

Duplicates Not Allowed:

Dictionaries cannot have two items with the same key

```
Eg:
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964,
  "year": 2020
}
print(thisdict)


Output:
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 2020}
```

Accessing Items:

You can access the items of a dictionary by referring to its key name, inside square brackets

Print the "brand" value of the dictionary

Eg:

```
 thisdict = {
 "brand": "Ford",
 "model": "Mustang",
 "year": 1964
}
print(thisdict["brand"])
```

Output:

```
Ford
```

Or

Using get method:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x=thisdict.get("brand")
print(x)
```

Output:

```
Ford
```

Length:
Print the number of items in the dictionary:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(len(thisdict))
```

Output:3

Dictionary Items - Data Type:

The values in dictionary items can be of any data type

Eg:

```
  thisdict = {
  "brand": "Ford",
  "electric": False,
  "year": 1964,
  "colors": ["red", "white", "blue"]
}

print(thisdict)
```

Output:

```
{'brand': 'Ford', 'electric': False, 'year': 1964, 'colors': ['red', 'white',
'blue']}
```

Get Keys:

The keys() method will return a list of all the keys in the dictionary.

Output:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

x = thisdict.keys()

print(x)
```

Output:

```
dict_keys(['brand', 'model', 'year'])
```

Adding a key:

Eg:

```
    car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.keys()

print(x)
```

```
car["color"] = "white"

print(x)
```

Output
```
dict_keys(['brand', 'model', 'year'])
dict_keys(['brand', 'model', 'year', 'color'])
```

Print values:

```
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.values()

print(x) #before the change

car["color"] = "white"

print(x)
```

Output:
```
dict_values(['Ford', 'Mustang', 1964, 'white'])
```

Values:

The values() method will return a list of all the values in the dictionary.

Eg:
```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

x = thisdict.values()

print(x)


Output:
  dict_values(['Ford', 'Mustang', 1964])
```

Change value in the dictionary:

Eg:

```
  car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.values()

print(x)

car["year"] = 2020

print(x)

Output:
dict_values(['Ford', 'Mustang', 2020])
```

Add a new item to the original dictionary

```
Eg:
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.values()

print(x)

car["color"] = "red"

print(x)

Output:
dict_values(['Ford', 'Mustang', 1964, 'red'])
```

Items:

Get a list of the key:value pairs

```
Eg:
  thisdict = {
 "brand": "Ford",
 "model": "Mustang",
 "year": 1964
}

x = thisdict.items()

print(x)

Output:

dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)]
```

Removing items in the dictionary:

The `pop()` method removes the item with the specified key name

Eg:
```
  thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)
```

Output:

```
{'brand': 'Ford', 'year': 1964}
```

Delete:

The `del` keyword removes the item with the specified key name:

Output:
```
    thisdict ={
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict["model"]
print(thisdict)
```

Output:

```
{'brand': 'Ford', 'year': 1964}
```

Example for delete a full dictionary:

```
  thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict
print(thisdict)
```

Loop Through a Dictionary:

Example:

Print all key names in the dictionary, one by one:

```
     thisdict ={
  "brand": "Ford",
 "model": "Mustang",
   "year": 1964}
   for x in thisdict:
  print(x)
```

Output:

```
   brand
   model
   year
```

Print all values in the dictionary, one by one:

Eg:

```
 thisdict =       {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict:
  print(thisdict[x])
```

Output:

```
  Ford
Mustang
1964
```

You can also use the `values()` method to return values of a dictionary

Eg:

```
thisdict =     {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict.values():
  print(x)
```

Output:

```
 Ford

Mustang

1964
```

Loop through both *keys* and *values*, by using the `items()` method:

Eg:

```
  thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x, y in thisdict.items():
  print(x, y)
```

Output:

```
brand Ford
model Mustang
year 196
```

To concatinate 2 list:

```
L=[1,4,6,7]
l2=[“a”,”b”,”c”]

print(L+L2)
```

Output

```
   [1,4,6,7,”a”,”b”,”c”]
```

Functions:

A function is a block of code which only runs when it is called.
You can pass data, known as parameters, into a function.A function is defined using the `def` keyword:

A function can return data as a result.

Syntax:

```
def functionname():
      Staments
```

Some Benefits of Using Functions:

Increase Code Readability
Increase Code Reusability

Eg:

```
def fun():
print("Welcome to GFG")
```

Function calling:

After creating a function in Python we can call it by using the name of the function followed by parenthesis containing parameters of that particular function.

Eg:

```
def fun():
      print("Welcome to GFG")
 # Driver code to call a function
 fun()
```

Eg:

```
   def my_function():
  print("Hello from a function")

  my_function()
```

OUTPUT:

```
      Hello from a function
```

Q)Add two numbers?

```
     def sum():
           x,y=10,20
           print("sum is",x+y)
    sum()
```

Output:

```
   Sum is 30
```

Arguments:

Arguments are the values passed inside the parenthesis of the function. A function can have any number of arguments separated by a comma.

Eg:

```
  def my_function(fname):
          print(fname + " Refsnes")

my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

Output:

```
      Emil Refsnes
      Tobias Refsnes
      Linus Refsnes
```

The terms parameter and argument can be used for the same thing: information that are passed into a function.

Eg:

```
   def hey(name):
         print(“my name is” +name)
        hey(“anil”)

     Or

def (naame):
      print(“my name is”,+name)
value=”crossboards”)
hey(value)
```

Return Values:

To let a function return a value, use the return statement:

```
    Def find_sum():
          Return 10+20
print(find_sum())
```

Eg:

```
def find(num):
   return num*num
print(find(4))
```

Output:

```
16
```

PASS MULTIPLE ARGUMENTS:

```
    def h(name,age):
         print("my name is" +name+ "age:" +str(ag))

h("cross",34)
```

The pass Statement

function definitions cannot be empty, but if you for some reason have a function definition with no content, put in the pass statement to avoid getting an error.

Eg:

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