

Module-2

Maheshkumar N

Module-2

- **Functions**, Built-In Functions, Commonly Used Modules, Function Definition and Calling the Function, The return Statement and void Function,
- **Strings**, Creating and Storing Strings, Basic String Operations, Accessing Characters in String by Index Number, String Slicing and Joining, String Methods,
- **Lists**, Creating Lists, Basic List Operations, Indexing and Slicing in Lists, Built-In Functions Used on Lists, List Methods, The del Statement.

Functions

- A Function is a self block of code which is used to organize the functional code.
- Function can be called as a section of a program that is written once and can be executed whenever required in the program, thus making code reusability.
- Function is a subprogram that works on data and produces some output.

◆ Types of Functions:

There are two types of Functions.

- ✓ a) **Built-in Functions:** Functions that are predefined and organized into a library. We have used many predefined functions in Python.
- ✓ b) **User- Defined:** Functions that are created by the programmer to meet the requirements.

Functions

- You can define functions to provide the required functionality. Here are simple rules to define a function in Python.
 - Function blocks begin with the keyword `def` followed by the function name and parentheses (()).
 - Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
 - The first statement of a function can be an optional statement - the documentation string of the function or docstring.
 - The code block within every function starts with a colon (:) and is indented.
 - The statement `return [expression]` exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as `return None`.

Functions

- Example: The following function takes a string as input parameter and prints it on standard screen

```
def printme( str ):
    "This prints a passed string into this function"
    print (str)
    return;
```

Functions

□ Calling a Function

- Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code.
- Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt. Following is the example to call printme() function –

Function definition is here

```
>>> def printme( str ):
    "This prints a passed string into this function"
    print (str)
    return;
```

Now you can call printme function

```
printme("I'm first call to user defined function!")
printme("Again second call to the same function")
```

Functions

How Function works in Python?

Functions

□ Pass by reference vs value

- All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function. For example –

Function definition is here

```
def changeme( mylist ):
```

```
    "This changes a passed list into this function"
```

```
    mylist.append([1,2,3,4]);
```

```
    print ("Values inside the function: ", mylist)
```

```
    return
```

Now you can call changeme function

```
mylist = [10,20,30];
```

```
changeme( mylist );
```

```
print ("Values outside the function: ", mylist)
```


Functions

□ Pass by reference vs value

- All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function. For example –

Function definition is here

```
def changeme( mylist ):
```

```
    "This changes a passed list into this function"
```

```
    mylist=([1,2,3,4]);
```

```
    print ("Values inside the function: ", mylist)
```

```
    return
```

Now you can call changeme function

```
mylist = [10,20,30];
```

```
changeme( mylist );
```

```
print ("Values outside the function: ", mylist)
```

Functions

□ Function Arguments

□ You can call a function by using the following types of formal arguments –

- ❖ Required arguments
- ❖ Keyword arguments
- ❖ Default arguments
- ❖ Variable-length arguments

Functions

□ Required arguments

- Required arguments are the arguments passed to a function in correct positional order. Here, the number of arguments in the function call should match exactly with the function definition.

Function definition is here

```
>>> def printme( str ):
    "This prints a passed string into this function"
    print (str)
    return;
```

Now you can call printme() function

```
printme()
```

- When the above code is executed, it produces the following result –

```
Traceback (most recent call last):
  File "test.py", line 11, in <module>
    printme();
TypeError: printme() takes exactly 1
argument (0 given)
```

Functions

□ Keyword arguments

- Keyword arguments are related to the function calls. When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name.

Function definition is here

```
>>> def printme( str ):
    "This prints a passed string into this function"
    print (str)
    return;
```

Now you can call printme() function

```
printme(str = 'Mahesh')
```

- When the above code is executed, it produces the following result –

Functions

□ **Keyword arguments**

Functions

□ Default arguments

- A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument. The following example gives an idea on default arguments, it prints default age if it is not passed –

Functions

□ Default arguments

Functions

□ Variable-length arguments

- You may need to process a function for more arguments than you specified while defining the function. These arguments are called *variable-length* arguments and are not named in the function definition, unlike required and default arguments.
- Syntax for a function with non-keyword variable arguments is this –

```
def functionname([formal_args,] *var_args_tuple ):  
    "function_docstring"  
    function_suite  
    return [expression]
```


Functions

▣ Variable-length arguments

- ▣ An asterisk (*) is placed before the variable name that holds the values of all nonkeyword variable arguments. This tuple remains empty if no additional arguments are specified during the function call. Following is a simple example –

Functions

□ The Anonymous Functions

- Those functions are called anonymous because they are not declared in the standard manner by using the `def` keyword. You can use the `lambda` keyword to create small anonymous functions.
- Lambda forms can take any number of arguments but return just one value in the form of an expression. They cannot contain commands or multiple expressions.
- An anonymous function cannot be a direct call to `print` because `lambda` requires an expression
- Lambda functions have their own local namespace and cannot access variables other than those in their parameter list and those in the global namespace.
- Although it appears that `lambda`'s are a one-line version of a function, they are not equivalent to inline statements in C or C++, whose purpose is by passing function stack allocation during invocation for performance reasons.

Functions

- The syntax of *lambda()* functions contains only a single statement, which is as follows –

`lambda [arg1 [,arg2,.....argn]]:expression`

Functions

□ The return Statement

- The statement `return [expression]` exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as `return None`.

Functions

□ The return Statement

- The statement `return [expression]` exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as `return None`.

Functions

□ The return Statement

Functions

□ Number of Arguments

- By default, a function must be called with the correct number of arguments. Meaning that if your function expects 2 arguments, you have to call the function with 2 arguments, not more, and not less.

```
def progme(a,b):  
    'this is user defined variables'  
    sum=a+b  
    print(sum)  
    return
```

```
progme(10,20)
```

Functions

□ Arbitrary Arguments, *args

- If you do not know how many arguments that will be passed into your function, add a * before the parameter name in the function definition.
- This way the function will receive a tuple of arguments, and can access the items accordingly:

Functions

□ Arbitrary Keyword Arguments, `**kwargs`

- If you do not know how many keyword arguments that will be passed into your function, add two asterisk: `**` before the parameter name in the function definition.
- This way the function will receive a dictionary of arguments, and can access the items accordingly:
- If the number of keyword arguments is unknown, add a double `**` before the parameter name:

Functions

□ Passing a List as an Argument

- You can send any data types of argument to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.
- E.g. if you send a List as an argument, it will still be a List when it reaches the function:

Functions

□ The pass Statement

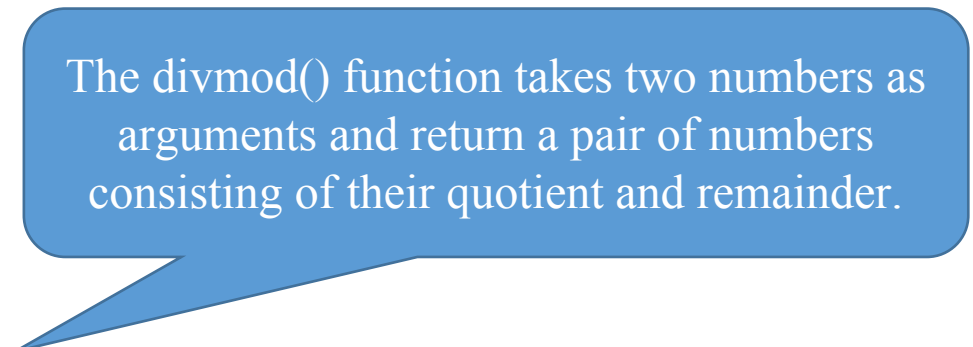
- function definitions cannot be empty, but if you for some reason have a function definition with no content, put in the pass statement to avoid getting an error.

Functions

□ Built-in-functions:

Functions

□ Built-in-functions:



The divmod() function takes two numbers as arguments and return a pair of numbers consisting of their quotient and remainder.

Functions

		Built-in Functions		
<u>abs()</u>	<u>delattr()</u>	<u>hash()</u>	<u>memoryview()</u>	<u>set()</u>
<u>all()</u>	<u>dict()</u>	<u>help()</u>	<u>min()</u>	<u>setattr()</u>
<u>any()</u>	<u>dir()</u>	<u>hex()</u>	<u>next()</u>	<u>slice()</u>
<u>ascii()</u>	<u>divmod()</u>	<u>id()</u>	<u>object()</u>	<u>sorted()</u>
<u>bin()</u>	<u>enumerate()</u>	<u>input()</u>	<u>oct()</u>	<u>staticmethod()</u>
<u>bool()</u>	<u>eval()</u>	<u>int()</u>	<u>open()</u>	<u>str()</u>
<u>breakpoint()</u>	<u>exec()</u>	<u>isinstance()</u>	<u>ord()</u>	<u>sum()</u>
<u>bytearray()</u>	<u>filter()</u>	<u>issubclass()</u>	<u>pow()</u>	<u>super()</u>
<u>bytes()</u>	<u>float()</u>	<u>iter()</u>	<u>print()</u>	<u>tuple()</u>
<u>callable()</u>	<u>format()</u>	<u>len()</u>	<u>property()</u>	<u>type()</u>
<u>chr()</u>	<u>frozenset()</u>	<u>list()</u>	<u>range()</u>	<u>vars()</u>
<u>classmethod()</u>	<u>getattr()</u>	<u>locals()</u>	<u>repr()</u>	<u>zip()</u>
<u>compile()</u>	<u>globals()</u>	<u>map()</u>	<u>reversed()</u>	<u>__import__()</u>
<u>complex()</u>	<u>hasattr()</u>	<u>max()</u>	<u>round()</u>	

Commonly Used Modules

- Modules in Python are reusable libraries of code having *.py* extension, which implements a group of methods and statements. Python comes with many built-in modules as part of the standard library.
- To use a module in your program, import the module using *import* statement. All the *import* statements are placed at the beginning of the program.
- The syntax for import statement is,
- For example, you can import the *math* module as
 1. `>>>import math`

Commonly Used Modules

□ Import()

- The syntax for using a function defined in a module is,
- ***module_name.function_name()***
- The module name and function name are separated by a dot.
- Here we list some of the functions supported by *math* module.
- >>> import math
- >>> print(math.ceil(5.4))
- 6
- >>> print(math.sqrt(4))
- 2.0
- >>> print(math.pi)
- 3.141592653589793

Commonly Used Modules

□ Dir()

- The built-in function *dir()* returns a sorted list of comma separated strings containing the names of functions, classes and variables as defined in the module.

1. >>> dir(math)

- ['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']

Commonly Used Modules

□ `help()`

- Another built-in function you will find useful is the *help()* function which invokes the built-in help system.
- 1. `>>> help(math.gcd)`
- Help on built-in function gcd in module math:
- `gcd(...)`
- `gcd(x, y) -> int`
- greatest common divisor of x and y

Commonly Used Modules

□ Random module

□ Another useful module in the Python standard library is the random module which generates random numbers.

- The syntax for *random.randint()* function is *random.randint(start, stop)* which generates a integer number between start and stop argument

numbers (including both)

```
1. >>> import random
```

```
2. >>> print(random.random())
```

```
0.2551941897892144
```

```
3. >>> print(random.randint(5,10))
```

```
9
```

Example :OTP generation

Program:

- Program to Demonstrate a Function with and without Arguments

Program:

- Program to Find the Area of Trapezium Using the Formula
 $\text{Area} = (1/2) * (a + b) * h$ Where a and b Are the 2 Bases of Trapezium and h Is the Height

Program:

- Program to Demonstrate Using the Same Variable Name in Calling Function and Function Definition

Program:

- Write a program to display pattern “Pattern-*”

Program:

- Write a program to display pattern “Pattern-*”

Program:

- Write a program to display pattern “Pattern-*”

Program:

- Write a program to display pattern “Pattern-*”

Program:

- Write a program to display pattern “Pattern-*”

Program:

- Write a program to display pattern “A”

Program:

- Write a program to display pattern “A”

Program:

- Write a program to display pattern “EC”

Program:

- Write a program to display pattern “YOUR NAME”

Program:

- Write a program to display pattern “EC”

Program:

- Write a program to display pattern

Program:

- Write a program to display pattern

Program:

- Write a program to display pattern

Program:

- Write a program to display pattern

Program:

- Write a Program to Display the Fibonacci Sequences up to nth Term Where n is Provided by the User

Program:

- Write a Program to Display the Fibonacci Sequences up to nth Term Where n is Provided by the User

Strings:

- A string consists of a sequence of characters, which includes letters, numbers, punctuation marks and spaces.
- To represent strings, you can use a single quote, double quotes or triple quotes.

The *str()* Function

- The *str()* function returns a string which is considered an informal or nicely printable representation of the given object. The syntax for *str()* function is,

str(object)

- It returns a string version of the object. If the object is not provided, then it returns an empty string.

Strings:

□ Basic String Operations

Strings:

□ **Basic String Operations**

- concatenated using + sign and * operator is used to create a repeated sequence of strings.

Strings:

□ *in* and *not in* Membership Operators

- You can check for the presence of a string in another string using *in* and ***not in*** membership operators.
- It returns either a Boolean True or False.

Strings:

□ String Comparison

- You can use (>, <, <=, >=, ==, !=) to compare two strings resulting in either Boolean True or False value.
- Python compares strings using ASCII value of the characters.

Strings:

□ Built-In Functions Used on Strings

Characters with highest and lowest ASCII value are calculated

Strings:

□ Accessing Characters in String by Index Number

- The syntax for accessing an individual character in a string is as shown below.

`string_name[index]`

where *index* is usually in the range of 0 to one less than the length of the string.

- You can access each character in a string using a subscript operator i.e., a square bracket.
- Square brackets are used to perform indexing in a string to get the value at a specific index or position.

Strings:

□ Accessing Characters in String by Index Number

- The index breakdown for the string *"be yourself"* assigned to *word_phrase* string variable is shown below.

```
>>>word_phrase[1]  
'e'
```

The last character in the string is referenced by an index value which is the **(size of the string – 1)** or **(len(string) – 1)**

The negative index breakdown for the string *"be yourself"* assigned to *word_phrase* string variable is shown below.

```
>>> word_phrase[-2]  
'l'
```

Strings:

□ String Slicing and Joining

- The "slice" syntax is a handy way to refer to sub-parts of sequence of characters within an original string.

The syntax for string slicing is,

Strings:

- **Write a program to reverse the given String**
- **1st way**
- **2nd way**
- **3rd way**

Strings:

- **Write Python Code to Determine Whether the Given String Is a Palindrome or Not, Using Slicing.**

Strings:

- **Program to reverse order of words.**
- input: “Learning Python is very Easy”
- output:”Easy Very is Python Learning”

Strings:

- **Program to reverse order of words.**
- input: “Learning Python is very Easy”
- output:”Easy Very is Python Learning”

Strings:

Joining Strings Using *join()* Method:

`string_name.join(sequence)`

- Here sequence can be string or list.
- If the sequence is a string, then *join()* function inserts *string_name* between each character of the string sequence and returns the concatenated string.
- If the sequence is a list, then *join()* function inserts *string_name* between each item of list sequence and returns the concatenated string.
- It should be noted that all the items in the list should be of string type.

Strings:

- **Split Strings Using *split()* Method:**

- The *split()* method returns a list of string items by breaking up the string using the delimiter string.
- The syntax of *split()* method is, ***string_name.split([separator [, maxsplit]])***

Strings:

- **Strings Are Immutable**

- As strings are immutable, it cannot be modified. The characters in a string cannot be changed once a string value is assigned to string variable. However, you can assign different string values to the same string variable.

Strings:

- **Strings Are Immutable**

- All Fundamental Data types are immutable. i.e once we creates an object, we cannot perform any changes in that object. If we are trying to change then with those changes anew object will be created. This non-chageable behaviour is called immutability.

Strings:

String Traversing

Since the string is a sequence of characters, each of these characters can be traversed using the *for* loop.

Program to Demonstrate String Traversing Using the *for* Loop

Strings:

Program to Print the Characters Which Are Common in Two Strings

Strings:

- **Write Python Program to Count the Total Number of Vowels, Consonants and Blanks in a String**

Strings:

- **Write Python Program to Calculate the Length of a String Without Using Built-In *len()* Function**

Strings:

- **Write Python Program to Convert Uppercase Letters to Lowercase and Vice Versa**

Strings:

- The **replace()** method replaces a string with another string:

Strings:

- The **replace()** method replaces a string with another string:

Strings:

- The **split()** method splits the string into substrings if it finds instances of the separator:

Strings:

- String Format

- But we can combine strings and numbers by using the `format()` method!
- The `format()` method takes the passed arguments, formats them, and places them in the string where the placeholders `{}` are:
- Use the `format()` method to insert numbers into strings:

Strings:

- String Format

- The format() method takes unlimited number of arguments, and are placed into the respective placeholders:
- You can use index numbers {0} to be sure the arguments are placed in the correct placeholders:

Strings:

- **Escape Character**

- To insert characters that are illegal in a string, use an escape character.
- An escape character is a backslash \ followed by the character you want to insert.
- An example of an illegal character is a double quote inside a string that is surrounded by double quotes:

- `txt = "We are the so-called "Vikings" from the north."`

- To fix this problem, use the escape character `\`:

- `txt = "We are the so-called \"Vikings\" from the north."`

Lists:

- ✓ Creating Lists,
- ✓ Basic List Operations,
- ✓ Indexing and Slicing in Lists,
- ✓ Built-In Functions Used on Lists,
- ✓ List Methods,
- ✓ The del Statement.

Python Collections (Arrays)

- ✓ There are four collection data types in the Python programming language:
 - ✓ **List:** is a collection which is ordered and changeable. Allows duplicate members.
 - ✓ **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
 - ✓ **Set** is a collection which is unordered and unindexed. No duplicate members.
 - ✓ **Dictionary** is a collection which is unordered, changeable and indexed. No duplicate members.

Lists:

- ✓ A list is a collection which is ordered and changeable. In Python lists are written with square brackets.
- ✓ Python list is a data structure which is used to store various types of data.
- ✓ In Python, lists are mutable i.e., Python will not create a new list if we modify an element of the list.
- ✓ It works as a container that holds other objects in a given order. We can perform various operations like insertion and deletion on list.
- ✓ A list can be composed by storing a sequence of different type of values separated by commas.
- ✓ Python list is enclosed between square([]) brackets and elements are stored in the index basis with starting index 0.

Creating Lists:

- ✓ Lists are constructed using square brackets [] wherein you can include a list of items separated by commas.

You can create an empty list without any items. The syntax is,

```
list_name = [ ]
```

Creating Lists:

✓ Example:

- You access the list items by referring to the index number:
- Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second last item etc.
- You can specify a range of indexes by specifying where to start and where to end the range. Return the third, fourth, and fifth item:

Remember that the first item has index 0.

Creating Lists:

✓ Example:

- To change the value of a specific item, refer to the index number: Change the third item:

Creating Lists:

✓ Loop Through a List:

- Print all items in the list, one by one:
- Check if Item Exists
- Check if "apple" is present in the list:
- To determine how many items a list has, use the len() function:

Creating Lists:

- To add an item to the end of the list, use the `append()` method:
- To add an item at the specified index, use the `insert()` method:
- The `remove()` method removes the specified item:
- There are several methods to remove items from a list:

Creating Lists:

- The `pop()` method removes the specified index, (or the last item if index is not specified):
- The `del` keyword removes the specified index:
- The `del` keyword can also delete the list completely:

Creating Lists:

- The `clear()` method empties the list:
- **Copy a List**
 - There are ways to make a copy, one way is to use the built-in List method `copy()`.
 - You cannot copy a list simply by typing `list2 = list1`, because: `list2` will only be a reference to `list1`, and changes made in `list1` will automatically also be made in `list2`.
- Another way to make a copy is to use the built-in method `list()`.

Creating Lists:

- Join Two Lists
 - There are several ways to join, or concatenate, two or more lists in Python.
-
- Using Append list2 into list1:

Creating Lists:

- Join Two Lists
- Or you can use the `extend()` method, which purpose is to add elements from one list to another list:

Creating Lists:

- **Basic List Operations:**

Creating Lists:

- The `list()` Function:

- The built-in `list()` function is used to create a list. The syntax for `list()` function is,

`list([sequence])`

- where the sequence can be a string, tuple or list itself. If the optional sequence is not specified then an empty list is created.

Lists:

- Indexing in Lists:

- The syntax for accessing an item in a list is,

list_name[index]

Lists:

- Indexing in Lists:

- The syntax for accessing an item in a list is,

list_name[index]

Negative Indexing

Lists:

- Modifying Items in Lists:

Lists:

- **Modifying Items in Lists:**

- When you assign an existing list variable to a new variable, an assignment (=) on lists does not make a new copy.
- Instead, assignment makes both the variable names point to the same list in memory.

Lists:

- Slicing in Lists:

- Slicing of lists is allowed in Python wherein a part of the list can be extracted by specifying index range along with the colon (:) operator which itself is a list.
- The syntax for list slicing is,
 - where both start and stop are integer values (positive or negative values).
 - List slicing returns a part of the list from the start index value to stop index value which includes the start index value but excludes the stop index value.
 - Step specifies the increment value to slice by and it is optional.

Lists:

- Slicing in Lists:

Built-In Functions Used on Lists

len()

max()

min()

sorted()

list()

range()

range()

range()

range()

range()

range()

range()

range()

range()

range()

range()

range()

range()

Lists:

- Built-in Functions Used on Lists:

Lists:

- List Methods:

Lists:

- Nested Lists:

- The syntax for nested lists is,
- Each list inside another list is separated by a comma. For example.

Lists:

- Nested Lists:

Lists:

- Two-Dimensional Lists:
- Two-dimensional list: a list that contains other lists as its elements
 - Also known as nested list
 - Common to think of two-dimensional lists as having rows and columns
 - Useful for working with multiple sets of data
- To process data in a two-dimensional list need to use two indexes
- Typically use nested loops to process

Lists:

- Two-Dimensional Lists:

Lists:

- **The del Statement:**

- You can remove an item from a list based on its index rather than its value.
- The difference between del statement and pop() function is that the del statement does not return any value while the pop() function returns a value.
- The del statement can also be used to remove slices from a list or clear the entire list

Lists:

- **Summary:**

- Lists are a basic and useful data structure built into the Python language.
- Built-in functions include *len()*, which returns the length of the list; *max()*, which returns the maximum element in the list; *min()*, which returns the minimum element in the list and *sum()*, which returns the sum of all the elements in the list.
- An individual elements in the list can be accessed using the index operator [].
- Lists are mutable sequences which can be used to add, delete, sort and even reverse list elements.
- The *sort()* method is used to sort items in the list.
- The *split()* method can be used to split a string into a list.
- Nested list means a list within another list.

Lists:

- Program:
- Program to Dynamically Build User Input as a List

Lists:

- Program:
- Program to Illustrate Traversing of Lists Using the for loop

Lists:

- Program:
- Program to Illustrate Traversing of Lists Using the for loop

Lists:

- Program:
- Program to Display the Index Values of Items in List

Lists:

- Program:
- Write Python Program to Sort Numbers in a List in Ascending Order Using Bubble Sort by Passing the List as an Argument to the Function Call.

Lists:

- Program:
- Input Five Integers (+ve and -ve). Find the Sum of Negative Numbers, Positive Numbers and Print Them.
Also, Find the Average of All the Numbers and Numbers Above Average.

Lists:

- Program:

- Input Five Integers (+ve and -ve). Find the Sum of Negative Numbers, Positive Numbers and Print Them. Also, Find the Average of All the Numbers and Numbers Above Average.

Lists:

- Program:
- Find Mean, Variance and Standard Deviation of List Numbers
- Steps to find the variance is to first calculate the mean, then traverse through each element in the list and subtract it with mean and square the result which is also called as the squared difference.
- Finally, find the mean of those squared differences to get the variance.
- Standard deviation is the square root of variance.

Lists:

- Program:

- Find Mean, Variance and Standard Deviation of List Numbers
- Steps to find the variance is to first calculate the mean, then traverse through each element in the list and subtract it with mean and square the result which is also called as the squared difference.
- Finally, find the mean of those squared differences to get the variance.
- Standard deviation is the square root of variance.

Lists:

- Program:
- Write a Program to Find the Transpose of a Matrix

Lists:

- Program:
- Write a Program to Find the Transpose of a Matrix

Lists:

- Program:
- Write Python Program to Add Two Matrices