AGOG TECHNOLOGIES
Since 2012

Python Book

Contents

❖ Introduction With python :-

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

❖ Why to learn Python :-

- **Python** is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.
- **Python** is a MUST for students and working professionals to become a great Software Engineer especially when they are working in Web Development Domain. I will list down some of the key advantages of learning Python:

- **Python is Interpreted** − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive** − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented** − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language** − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

- **Easy to learn & design To be easy**
  - ➢ Clean & clear syntax
  - ➢ Very few syntax
- **Highly Portable**
  - ➢ Runs almost anywhere - high end servers and workstations, down to windows CE
  - ➢ Uses machine independent byte-code
- **Extensible**
  - ➢ Designed to be extensible using C/C++,
  - ➢ allowing access to many external libraries

❖ Features of Python:-

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.
- Clean syntax plus high-level data types
  - ➢ Leads to fast coding (First language in many universities abroad!)
- Uses white-space to delimit blocks
  - ➢ Humans generally do, so why not the language?
  - ➢ Try it, you will end up liking it
- Variables do not need declaration
  - ➢ Although not a type-less language

❖ Python Productivity:-

- Reduced development time
  - ➢ code is 2-10x shorter than C, C++, Java
- Improved program maintenance
  - ➢ code is extremely readable
- Less training
  - ➢ language is very easy to learn

Python is available on a wide variety of platforms including Linux and Mac OS X. Let's understand how to set up our Python environment.

❖ Local Environment Setup

First open a terminal for that simply type in search box ‘cmd’ and press Enter. Now type “python” to find out if it is already installed and which version is installed.

The most up-to-date and current source code, binaries, documentation, news, etc., is available on the official website of Python https://www.python.org/

❖ Installing python

Python distribution is available for a wide variety of platforms. You need to download only the binary code applicable for your platform and install Python.

❖ Path Setting

The path is stored in an environment variable, which is a named string maintained by the operating system. This variable contains information available to the command shell and other programs.

The **path** variable is named as PATH in Unix or Path in Windows (Unix is case sensitive; Windows is not).

In Mac OS, the installer handles the path details. To invoke the Python interpreter from any particular directory, you must add the Python directory to your path.

AGOG TECHNOLOGIES ❖ **Core Python Concepts :-**

• The Python language has many similarities to Perl, C, and Java. However, there are some definite differences between the languages

First Python Program:-

Type the following text at the Python prompt and press the Enter −

```
>>> print "Hello, Python!"
```

If you are running new version of Python, then you would need to use print statement with parenthesis as in **print ("Hello, Python!");**. However in Python version 2.4.3, this produces the following result −

```
Hello, Python!
```

Script Mode Programming:

Invoking the interpreter with a script parameter begins execution of the script and continues until the script is finished. When the script is finished, the interpreter is no longer active.

Let us write a simple Python program in a script. Python files have extension **.py**. Type the following source code in a test.py file –

```
print "Hello, Python!"
```

We assume that you have Python interpreter set in PATH variable. Now, try to run this program as follows –

```
$ python test.py
```

This produces the following result –

```
Hello, Python!
```


- **Variable Types :-**

  - Variable is a name that is used to refer to memory location. Python • variable is also known as an identifier and used to hold value.
  - In Python, we don't need to specify the type of variable because Python is a infer language and smart enough to get variable type.
  - In Python, we don't need to specify the type of variable because Python is a infer language and smart enough to get variable type.
  - It is recommended to use lowercase letters for the variable name. Rahul and rahul both are two different variables.

- **Identifier Naming:-**

  - ➢ The first character of the variable must be an alphabet or underscore (_).
  - ➢ All the characters except the first character may be an alphabet of lower case(a-z), upper-case (A-Z), underscore, or digit (0-9).
  - ➢ Identifier name must not contain any white-space, or special character

(!, @, #, %, ^, &, *).

- ➢ Identifier name must not be similar to any keyword defined in the language.
- ➢ Identifier names are case sensitive; for example, my name, and MyName is not the same.
- ➢ Examples of valid identifiers: a123, _n, n_9, etc.
- ➢ Examples of invalid identifiers: 1a, n%4, n 9, etc.

• **Declaring Variable and Assigning Values** :-

- ➢ Python does not bind us to declare a variable before using it in the application. It allows us to create a variable at the required time.
- ➢ We don't need to declare explicitly variable in Python. When we assign any value to the variable, that variable is declared automatically.
- ➢ The equal (=) operator is used to assign value to a variable.

```
#!/usr/bin/python

counter = 100          # An integer assignment
miles   = 1000.0       # A floating point
name    = "John"       # A string

print counter
print miles
print name
```

Here, 100, 1000.0 and "John" are the values assigned to *counter*, *miles*, and *name* variables, respectively. This produces the following result −

```
100
1000.0
John
```

• **Data types Of Python:-**

➢ Variables can hold values, and every value has a data-type. Python is a dynamically typed language; hence we do not need to define the type of the variable while declaring it. The interpreter implicitly binds the value with its type.

• **Standard Data Types :-**

➢ A variable can hold different types of values. For example, a person's name must be stored as a string whereas its id must be stored as an integer.
➢ Python provides various standard data types that define the storage method on each of them. The data types defined in Python are given below.

1. **Numbers**
2. **Sequence Type**
3. **Boolean**
4. **Set**
5. **Dictionary**

.

❖ Conditional Statement :

➢ Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions
➢ Decision structures evaluate multiple expressions which produce TRUE or FALSE as outcome.

1. If :-

➢ It is similar to that of other languages.
➢ The if statement contains a logical expression using which data is compared and a decision is made based on the result of the comparison.
➢ Syntax :-

```
If expression:
    Statement
```

2. else-If :-

➢ The if-else statement provides an else block combined with the if statement which is executed in the false case of the condition.
➢ Syntax :-

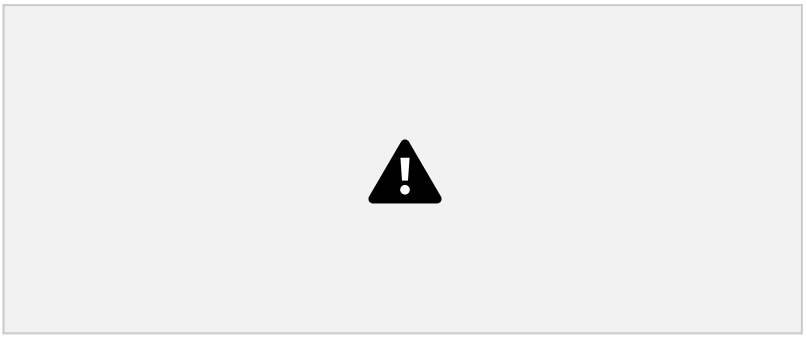
```
If condition :
    # block Statements
Else :
    # another block of statements(else-block)
```

3. Nested if…..else statement :-

➢ There may be a situation when you want to check for another condition after a condition resolves to true.
➢ Syntax :-

```
If expression1:
        # Statement
```

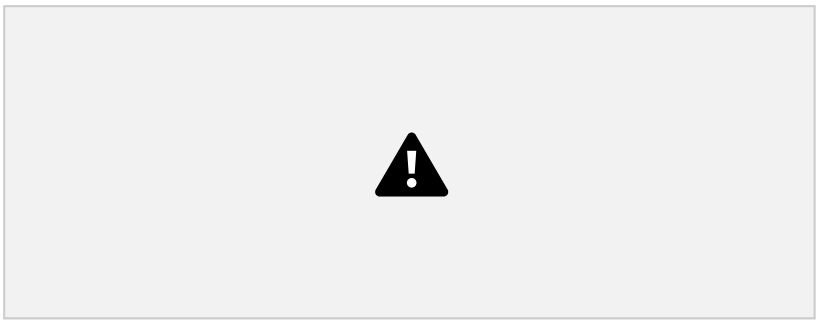
```
        If expression2:
                #statement
Else :
        # statement
```



4. elif statement :-

- ➢ The elif statement works like an if-else-if ladder statement in C. It must be succeeded by an if statement.
- ➢ The syntax of the elif statement is given below.
- ➢ Syntax :-

```
        If expression1:
                # block statements
        elif expression2:
                # block statements
        elif expression3:
                # block statements
        else:
                # block statments
```

❖ Looping

➢ A loop statement allows us to execute a statement or group of statements multiple times.
➢ Python programming language provides following types of loops to handle looping requirements.

▪ **For Loop** :- For loop has the ability to iterate over the items of any sequence, such as a list or a string

Syntax :-

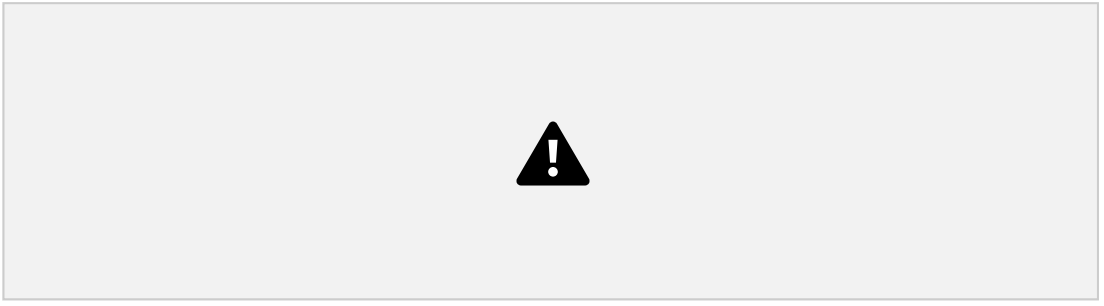
```
for iterating_var in sequence:
    Statements(s)
```

▪ **Nested For Loop** :-

Python allows us to nest any number of for loops inside a **for** loop. The inner loop is executed n number of times for every iteration of the outer loop. The syntax is given below.

Syntax :-

```
for iterating_var1 in sequence: #outer loop
    for iterating_var2 in sequence: #inner loop
        # block statements
# other statements
```

• **While loop :-**

➢ The Python while loop allows a part of the code to be executed until the given condition returns false. It is also known as a pre-tested loop.
➢ It can be viewed as a repeating if statement. When we don't know the number of iterations then the while loop is most effective to use.
➢ Syntax :-

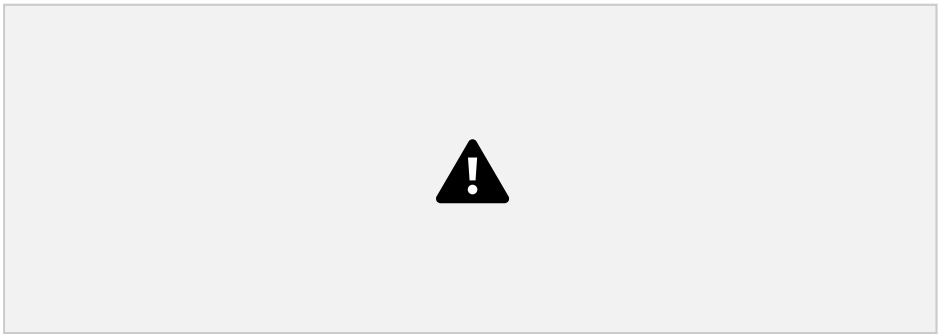
While expression:

Statements

• **Control Statements :-**

➢ Loop control statements change execution from its normal sequence. ➢ When execution leaves a scope, all automatic objects that were created in that scope are destroyed.
➢ Python supports the following control statements

- Break :-
  - ➢ It brings control out of the loop and transfers execution to the statement immediately following the loop



Continue :-

➢ It continues with the next iteration of the loop

Pass :-

➢ The pass statement does nothing.
➢ It can be used when a statement is required syntactically but the program requires no action

❖ String Manipulation :-

➢ Textual data in Python is handled with ”str” objects, or strings. Strings are immutable(fixed/rigid) sequences of Unicode code points.
➢ String literals are written in a variety of ways:
- Single quotes: 'allows embedded "double" quotes'
- Double quotes: "allows embedded 'single' quotes".
- Triple quoted: '''Three single quotes''', """Three double quotes"""

➢ Triple quoted strings may span multiple lines - all associated whitespace will be included in the string literal.

❖ String Functions :-

➢ **str.capitalize()** :-
- Return a copy of the string with its first character capitalized and the rest lowercased.

➢ **str.casefold()** :-
- Return a case folded copy of the string. Case folded strings may be used for caseless matching

➢ **str.center(width[, fillchar])** :-
- Return cantered in a string of length width. Padding is done using the specified fillchar (default is an ASCII space). The original string is returned if width is less than or equal to len(s).

➢ **str.count(sub[, start[, end]])** :-
- Return the number of non-overlapping occurrences of substring sub in the range [start, end].
- Optional arguments start and end are interpreted as in slice notation.

➢ **str.endswith(suffix[, start[, end]])** :-
- Return True if the string ends with the specified suffix, otherwise return False
- Suffix can also be a tuple of suffixes to look for.
- With optional start, test beginning at that position.
- With optional end, stop comparing at that position.

➢ **str.find(sub[, start[, end]])** :-
- Return the lowest index in the string where substring sub is found within the slice s [start: end].
- Optional arguments start and end are interpreted as in slice notation. Return -1 if sub is not found.

➢ **str.isalnum()** :-
- Return true if all characters in the string are alphanumeric and there is at least one character, false otherwise
- A character c is alphanumeric if one of the following returns True:
- c.isalpha(), c.isdecimal(), c.isdigit(), or c.isnumeric()

➢ **str.isidentifier()** :-
- Return true if the string is a valid identifier according to the language definition

➢ **str.islower()** :-
- Return true if all cased characters in the string are lowercase and there is at least one cased character, false otherwise.

➢ **str.istitle()** :-
- Return true if the string is a titlecased string and there is at least one character, for example uppercase characters may only follow uncased characters and lowercase characters only cased ones. Return false otherwise.

➢ **str.isupper()** :-

- Return true if all cased characters in the string are uppercase and there is at least one cased character, false otherwise.

➢ **str.join(iterable)** :-
- Return a string which is the concatenation of the strings in the iterable. A TypeError will be raised if there are any non-string values initerable, including bytes objects. The separator between elements is the string providing this method.

➢ **str.ljust(width[, fillchar])** :-
- Return the string left justified in a string of length width. Padding is done using the specified fillchar (default is an ASCII space). The original string is returned if width is less than or equal to len(s).

➢ **str.lower()** :-
- Return a copy of the string with all the cased characters converted to lowercase.

➢ **str.partition(sep)** :-
- Split the string at the first occurrence of sep, and return a 3-tuple containing the part before the separator, the separator itself, and the part after the separator.
- If the separator is not found, return a 3-tuple containing the string itself, followed by two empty strings.

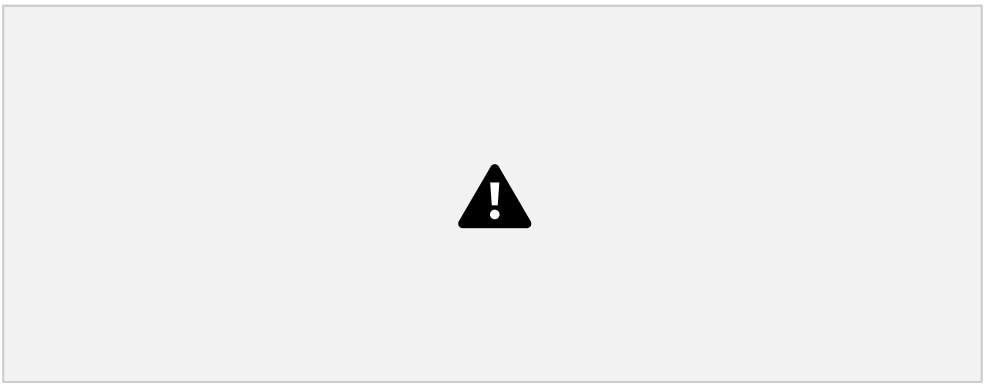
➢ **str.replace(old, new[, count])** :-
- Return a copy of the string with all occurrences of substring old replaced by new.
- If the optional argument count is given, only the first countoccurrences are replaced.

➢ **str.split(sep=None, maxsplit=-1)** :-
- Return a list of the words in the string, using sep as the delimiter string. If maxsplit is given, at most maxsplit splits are done (thus, the list will have at most maxsplit+1 elements).
- If maxsplit is not specified or -1, then there is no limit on the number of splits

➢ **str.swapcase()** :-
- Return a copy of the string with uppercase characters converted to lowercase and vice versa.

➢ **str.title() :-**
- Return a titlecased version of the string where words start with an uppercase character and the remaining characters are lowercase.

Example :



❖ Slicing:-

- ➢ Like other programming languages, it's possible to access individual characters of a string by using array-like indexing syntax. In this we can access each and every element of string through their index number and the indexing starts from 0. Python does index out of bound checking.
- ➢ So, we can obtain the required character using syntax, string_name[index_position]:
- ➢ The positive index_position denotes the element from the starting(0) and the negative index shows the index from the end(-1).
- ➢ To extract substring from the whole string then then we use the syntax like
- ➢ string_name[beginning: end : step]beginning represents the starting index of string end denotes the end index of string which is not inclusive steps denotes the distance between the two words.
- ➢ Example :print x[2:5] # Prints substring stepping up 2nd character
- ➢ Example : print [4:10:2] # from 4th to 10th character
- ➢ print x[-5:-3] # Prints 3rd character from rear from 3 to 5

❖ Lists:

• Python knows a number of compound data types, used to group together other values.

• The list is a most versatile datatype available in Python which can be written as a list of comma-separated values (items) between square brackets. • Important thing about a list is that items in a list need not be of the same type.

Example :

Cars = [“BMW”, ”Audi”, ”RR”, ”Datsun”]

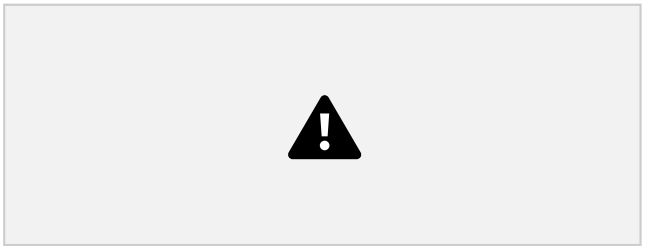
List1 = [10, 20 , 80, 40, 90]

Emp_data = [‘emp1’, 101 , ’vastrapur’ , 18000]

• **Accessing Value in List :**

Lists can be indexed and sliced like a strings.

To access values in lists, use the square brackets for slicing along with the index or indices to obtain value available at that index.



Output:

Lists are mutable type means it is possible to change their content.

You can update single or multiple elements of lists by giving the slice on the left-hand side of the assignment operator, and you can add to elements in a list with the append() method.

- **List Operations :**

1. **“in” operator** :

This operator is used to check if an elements **is present** in the list or not. Return true if element is present in list else return false.

Ex:

```
Cars = [“BMW”, ”Audi”, ”RR”, ”Datsun”]

if ‘Audi’ in Cars:
        Print(“Yes, Audi is in List “)
else:
        print(“No , Its Not Present in List “)
```

2. **“not in” operator** :

This operator is used to check if an elements **is not present** in the list or not. Return true if element is not present in list else return false.

Ex:

```
Cars = [“BMW”, ”Audi”, ”RR”, ”Datsun”]

if ‘alto’ not in Cars:
        Print(“Yes, alto is not in List “)
else:
        print(“No , Its Present in List “)
```

3. With list we can iterate, find the elements, concate the list and so more like…

Ex:

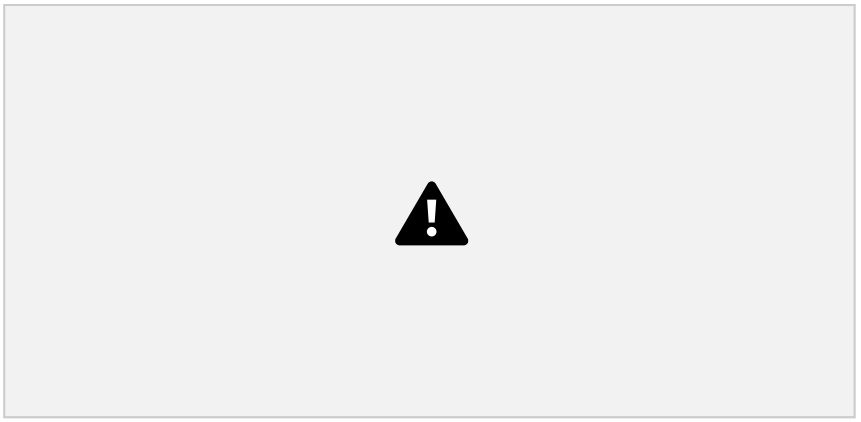
```
Len([1,2,3]) #Find out the Length
[1,2,3]+[4,5,6] #concatenation of 2 list
                [“Hello”]*3 #repetation( “Hello”, “Hello”, “Hello”)

for x in [1,2,3]: #iteration
```

• Buit-in Functions of Lists

Python includes the following list functions –

len(list)	Gives the total length of the list.
max(list)	Returns item from the list with max value
min(list)	Returns item from the list with min value
list(seq)	Converts a tuple into list.

Example:

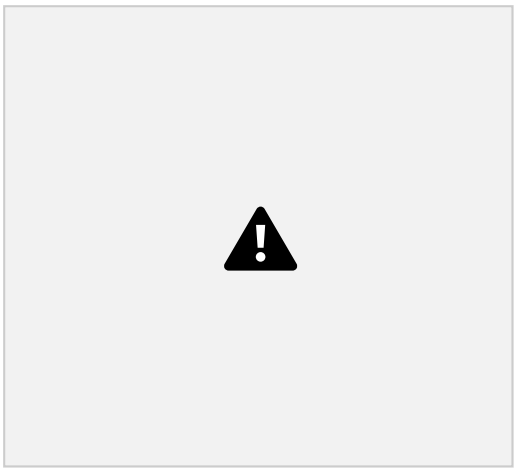


Output:

• Methods of Lists :-

list.append(x)	Add an item to the end of the list. Equivalent to a [len(a):]=[x].
list.extend(L)	Appends the contents of L to list
list.insert(I,x)	Insert an item at a given position. The first argument is the index of the element before which to insert, so a.insert(0,x) inserts at the front of the list,and a.insert(len(a),x) is equivalent to a.append(x).
list.count(obj)	Returns count of how many times obj occurs in list
list.index(obj)	Returns the lowest index in list that obj appears
list.pop(obj=list[-1]	Removes and returns last object or obj from list.
list.reverse()	Reverses Objects of list in place
list.sort([fun])	Sorts objects of list, use compare function If given
List.remove(obj)	Removes object obj from list

Example:



❖ Tuples

- A tuple is a collection of objects which ordered and **immutable**. •

Tuples are sequences, just like lists.

- The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

Example:

fruits=(“Mango”,”Banana”,”Oranges”,10,66)

numbers=(11,22,33,44)

fruits=”Mango”,”Banana”,”Oranges”

Accessing Values in tuples :

- There are various ways in which we can access the elements of a tuple. •

We can use the **index operator [ ] to access** an item in a tuple

- Index starts from 0. The index must be an integer
- Python allows negative indexing for its sequences. The index of -1 refers to the last item, -2 to the second last item and so on.
- We can access a range of items in a tuple by **using the slicing operator ‘ : ‘(colon)**

Example:

Output:

Working :

- Unlike lists, tuples are immutable. This means that elements of a tuple cannot be changed once it has been assigned.
- But if the element is itself a mutable data type like list, its nested items can be changed.
- We can also assign a tuple to different values (reassignment) •

Also we cannot delete or remove items from a tuple.

- But deleting the tuple entirely is possible using the keyword **del**. •

With Tuples we can do concate ,repetation,iterations etc…

More Example with different operations:

• **Buit-in Function of tuple:**

cmp(tuple1, tuple2)	Compares elements of both tuples.
len(tuple)	Gives the total length of the tuple.
max(tuple)	Returns item from the tuple with max value.
min(tuple)	Returns item from the tuple with min value.
tuple(seq)	Converts a list into tuple.

❖ Dictionaries :

- Dictionaries are sometimes found in other languages as “associative memories” or “associative arrays”
- Unlike sequences, which are indexed by a range of numbers, dictionaries are indexed by keys, which can be any immutable type; strings and numbers can always be keys.
- Tuples can be used as keys if they contain only strings, numbers, or tuples; if a tuple contains any mutable object either directly or indirectly, it cannot be used as a key.
- You can’t use lists as keys, since lists can be modified in place using index assignments, slice assignments, or methods like append() and extend(). • The main operations on a dictionary are storing a value with some key and

extracting the value given the key.

- Like lists they can be easily changed, can be shrunk and grown ad libitum at run time. They shrink and grow without the necessity of making copies. Dictionaries can be contained in lists and vice versa.
- A list is an ordered sequence of objects, whereas dictionaries are unordered sets
- But the main difference is that items in dictionaries are accessed via keys and not via their position

Accessing values in dictionaries:

To access dictionary elements, we can use the familiar square brackets along with the key to obtain its value

We can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry.

Example:

Output:

Updating dictionary:

You can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry as shown below in the simple example

Output:

Delete dictionary:

You can either remove individual dictionary elements or clear the entire contents of a dictionary. You can also delete entire dictionary in a single operation.

Properties of Dictionary Keys:

Dictionary values have no restrictions. They can be any arbitrary Python object, either standard objects or user-defined objects. However, same is not true for the keys.

There are two important points to remember about dictionary keys –

1. More than one entry per key not allowed. Which means no duplicate key is allowed. When duplicate keys encountered during assignment, the last assignment wins.

   Ex:

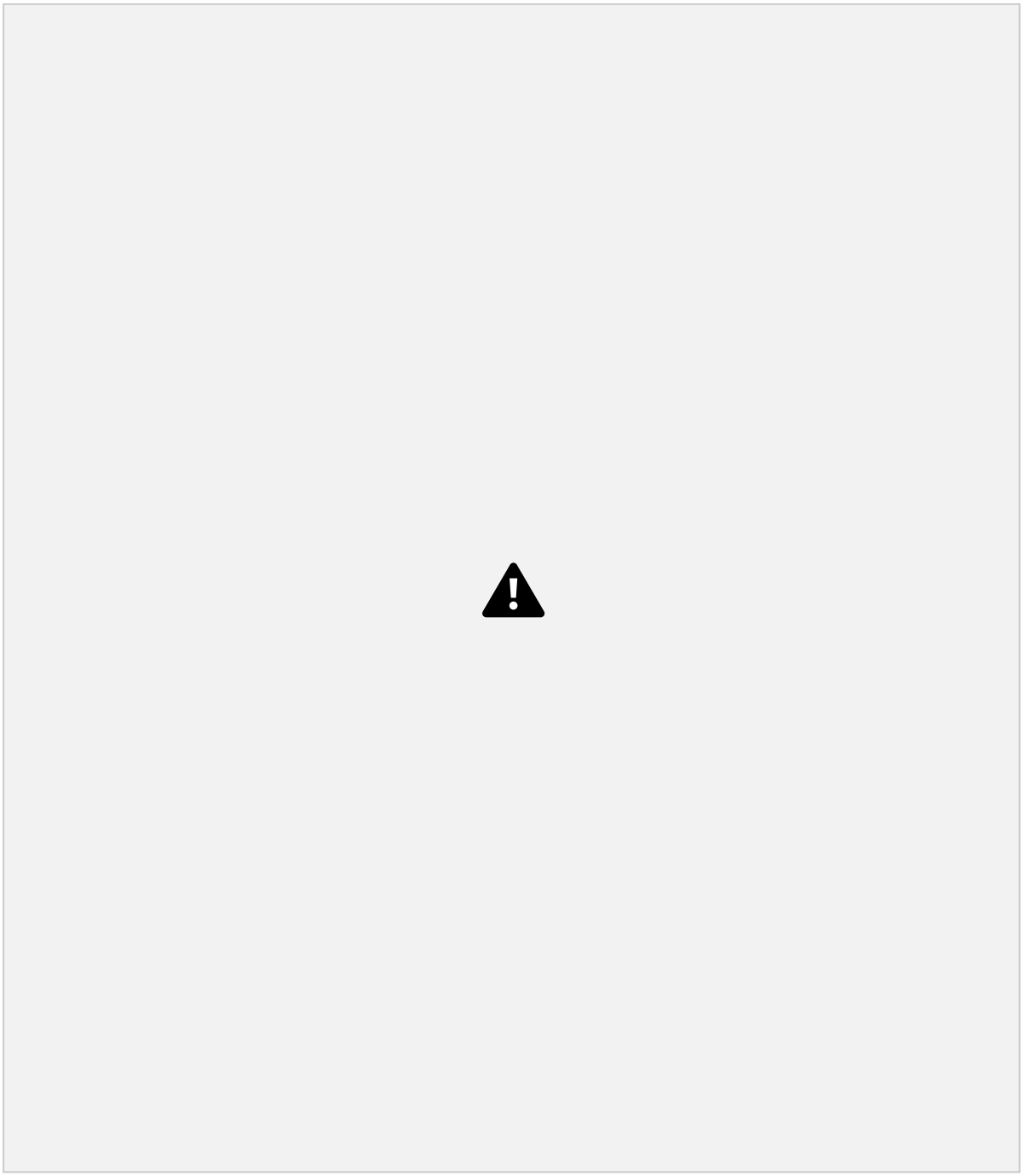
2. Keys must be immutable. Which means you can use strings, numbers or tuples as dictionary keys but something like ['key'] is not allowed.

Ex :

it produces the following result:

Built-in Dictionary Functions & Methods

Python includes following dictionary methods –



❖ Sets:

A Python set is the collection of the unordered items. Each element in the set must be unique, immutable, and the sets remove the duplicate elements. Sets are mutable which means we can modify it after its creation.

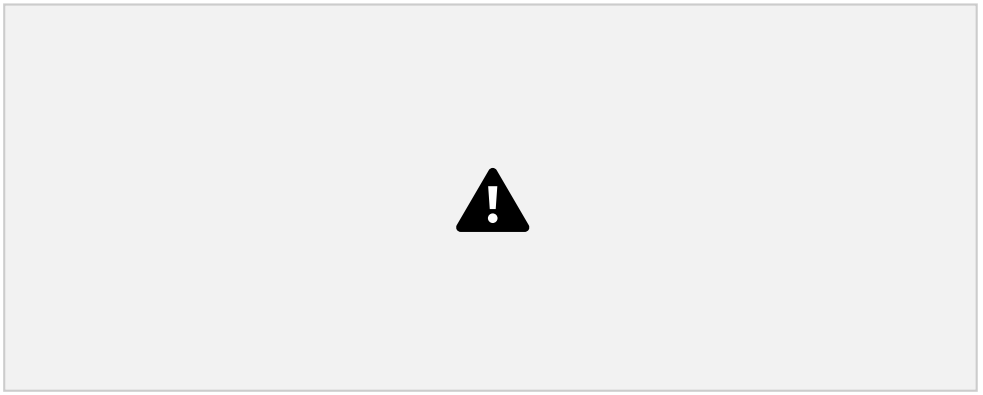
Creating Sets:

The set can be created by enclosing the comma-separated immutable items with the curly braces {}.

Python also provides the set() method, which can be used to create the set by the passed sequence.

Ex:

Output:



❖ Function:-

• **Defining Function :-**

➢ 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.

➢ Syntax :-

➢ Ex :-

• **Type of Function**

Function can be of Built-in Function and user-defined

Functions. **Built-in functions**

➢ Functions that come built into the Python language itself are called built in functions and are readily available to us. Eg: input(),eval(),print() etc…

User defined functions

➢ Functions that we define ourselves to do certain specific task are referred as user-defined functions. Eg:checkNoEvenOdd(20)

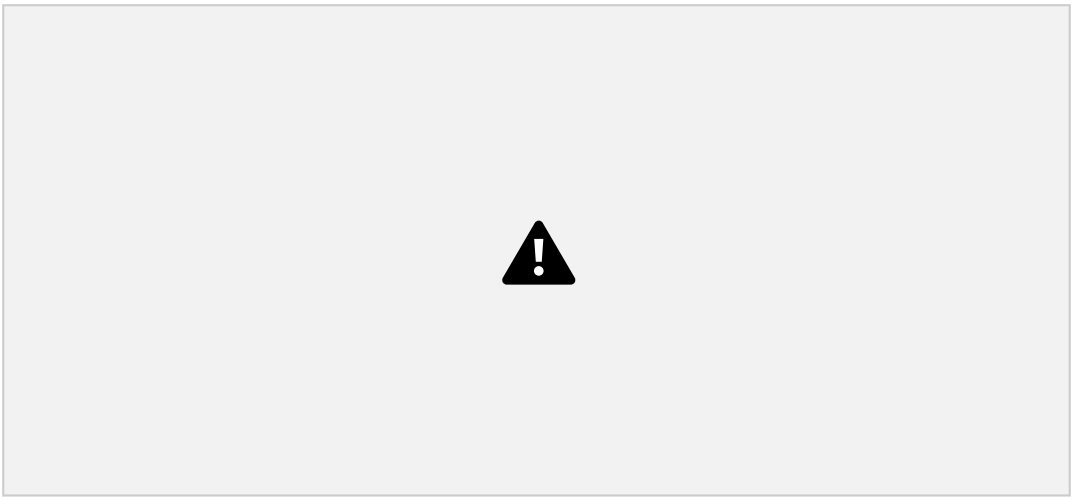
• **Calling Function :-**

➢ 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 −

• **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

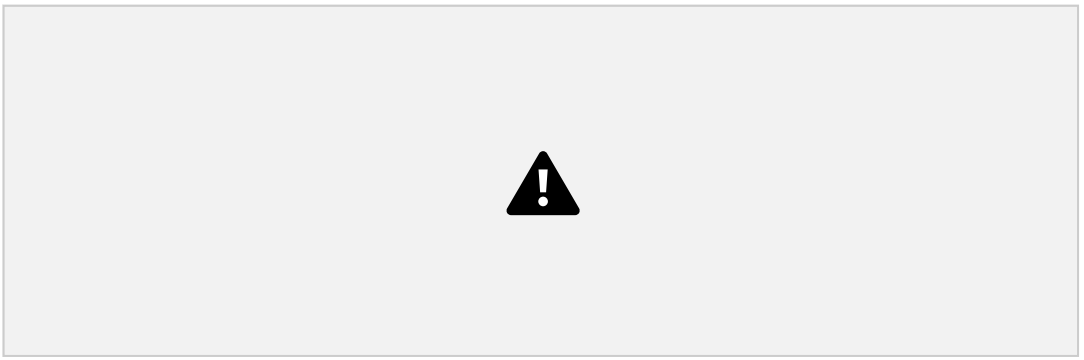


• **Function Arguments :-**

➢ Required arguments
➢ Keyword arguments
➢ Default arguments
➢ Variable-length arguments

○ 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.

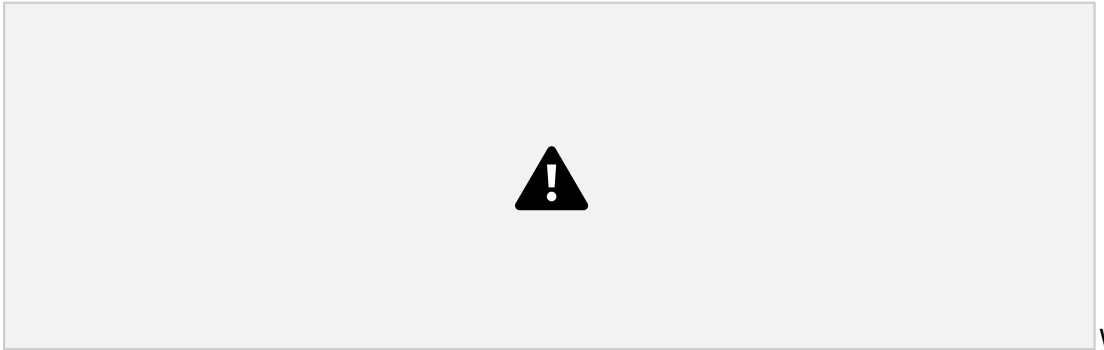


When the above code is executed, it produces the following result :−

- 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.

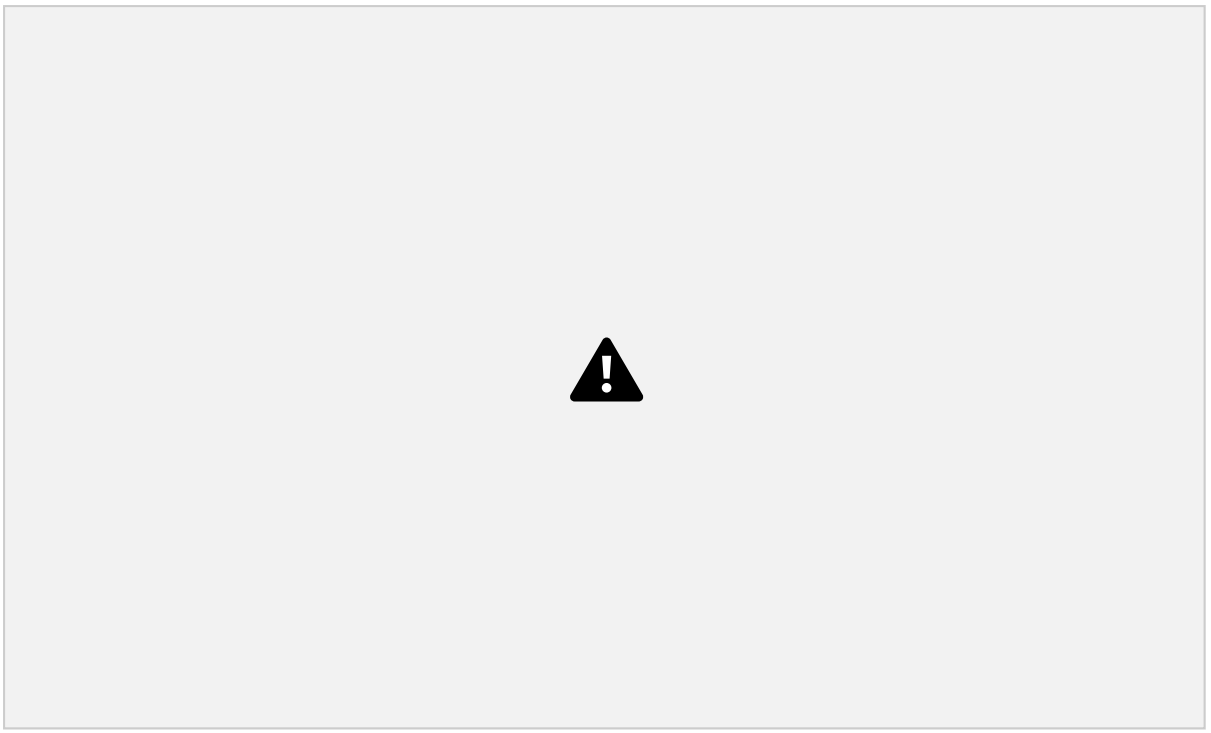
➢ This allows you to skip arguments or place them out of order because the Python interpreter is able to use the keywords provided to match the values with parameters. You can also make keyword calls to the *printme()* function in the following ways



When the above code is executed, it produces the following result :−

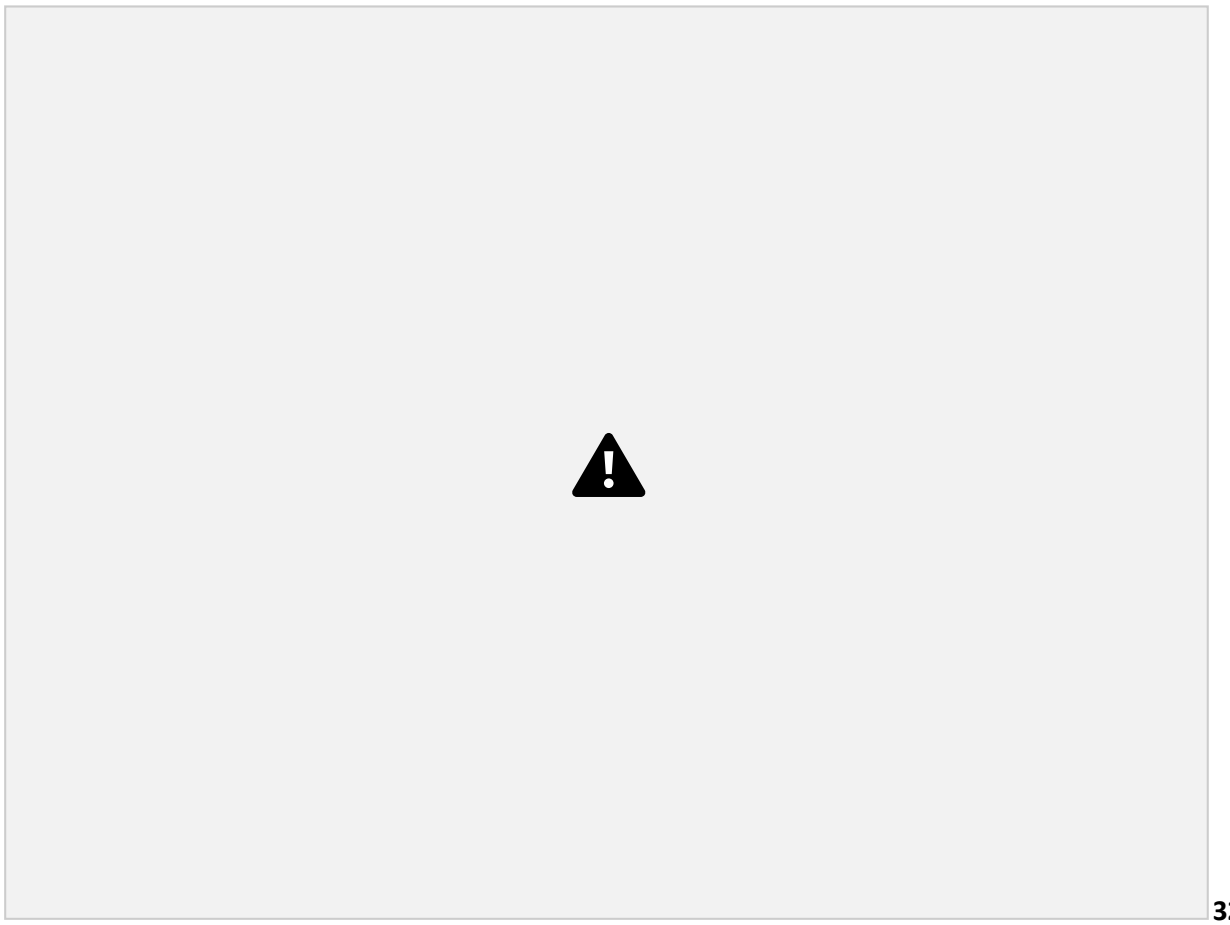
- Default arguments :-

➢ 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



- 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 –



Anonymous functions :-

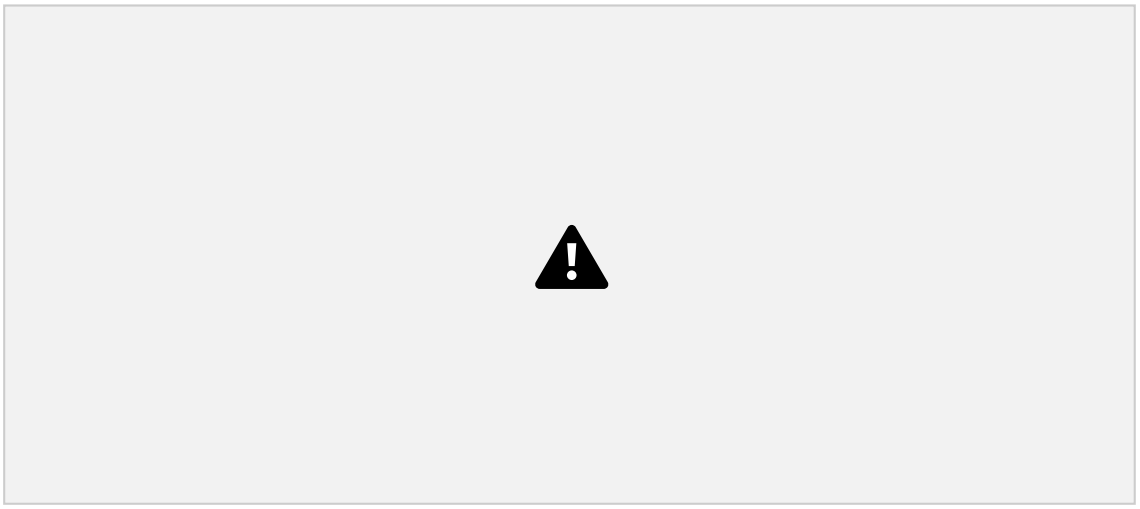
- These functions are called anonymous because they are not declared in the standard manner by using the *def* keyword. You can use the *lambda* keyword to create small anonymous functions.
- Lambda forms can take any number of arguments but return just one value in the form of an expression. They cannot 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.

➢ Syntax :-

Following is the example to show how *lambda* form of function works –

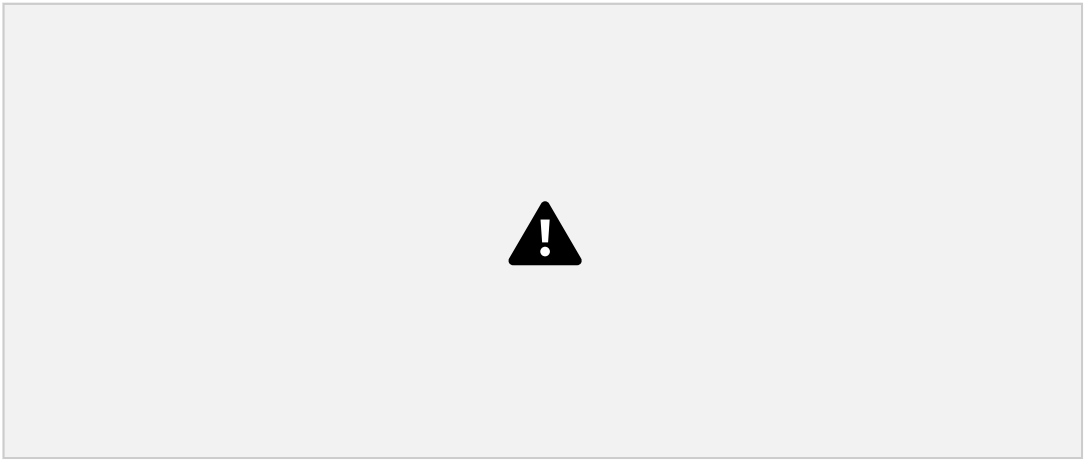


- **Return statement :-**

➢ 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.

➢ All the above examples are not returning any value. You can return a value from a function as follows –

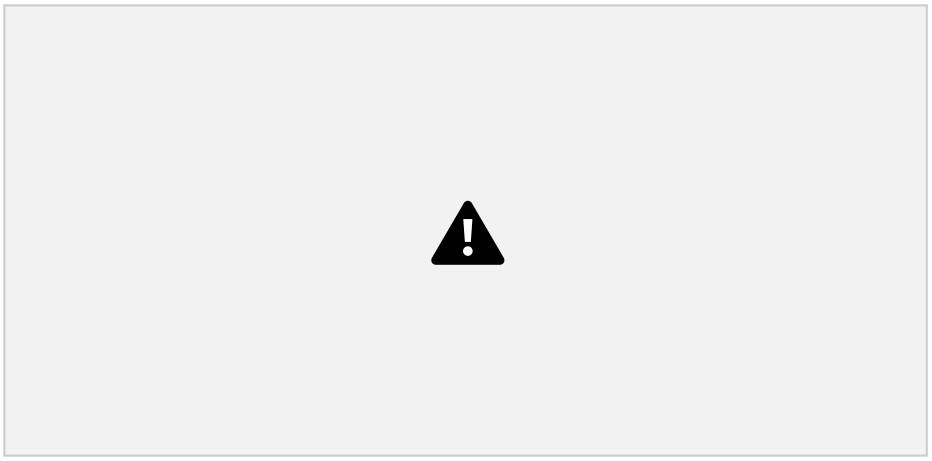
If a variable is assigned a value anywhere within the function's body, it's assumed to be a local unless explicitly declared as global. Eg: def set_globvar_to_one():



Global & Local Variables :-

- Global Variables :-

➢ Defining a variable on the module level makes it a global variable, you don't need to global keyword.
➢ The global keyword is needed only if you want to reassign the global variables in the function/method.
➢ In Python, variables that are only referenced inside a function are implicitly global.
➢ Eg: def set_globvar_to_one():
➢ global globvar
➢ # Needed to modify global copy of globvar
➢ globvar = 1



• Local Variables :-

➢ If a variable is assigned a value anywhere within the function's body, it's assumed to be a local unless explicitly declared as global.
➢ Local variables of functions can't be accessed from outside
➢ Eg : def F()
S="I am a local variable"
Print(s)
F()

Print(s) # Gives NameError: name 's' is not defined

❖ Modules :-

i. Importing module
ii. Math module
iii. Random module
iv. Packages
v. Composition

➢ A module is a file containing Python definitions and statements. ➢ The file name is the module name with the suffix .py appended. ➢ Within a module, the module's name (as a string) is available as the value of the global variable __name__.

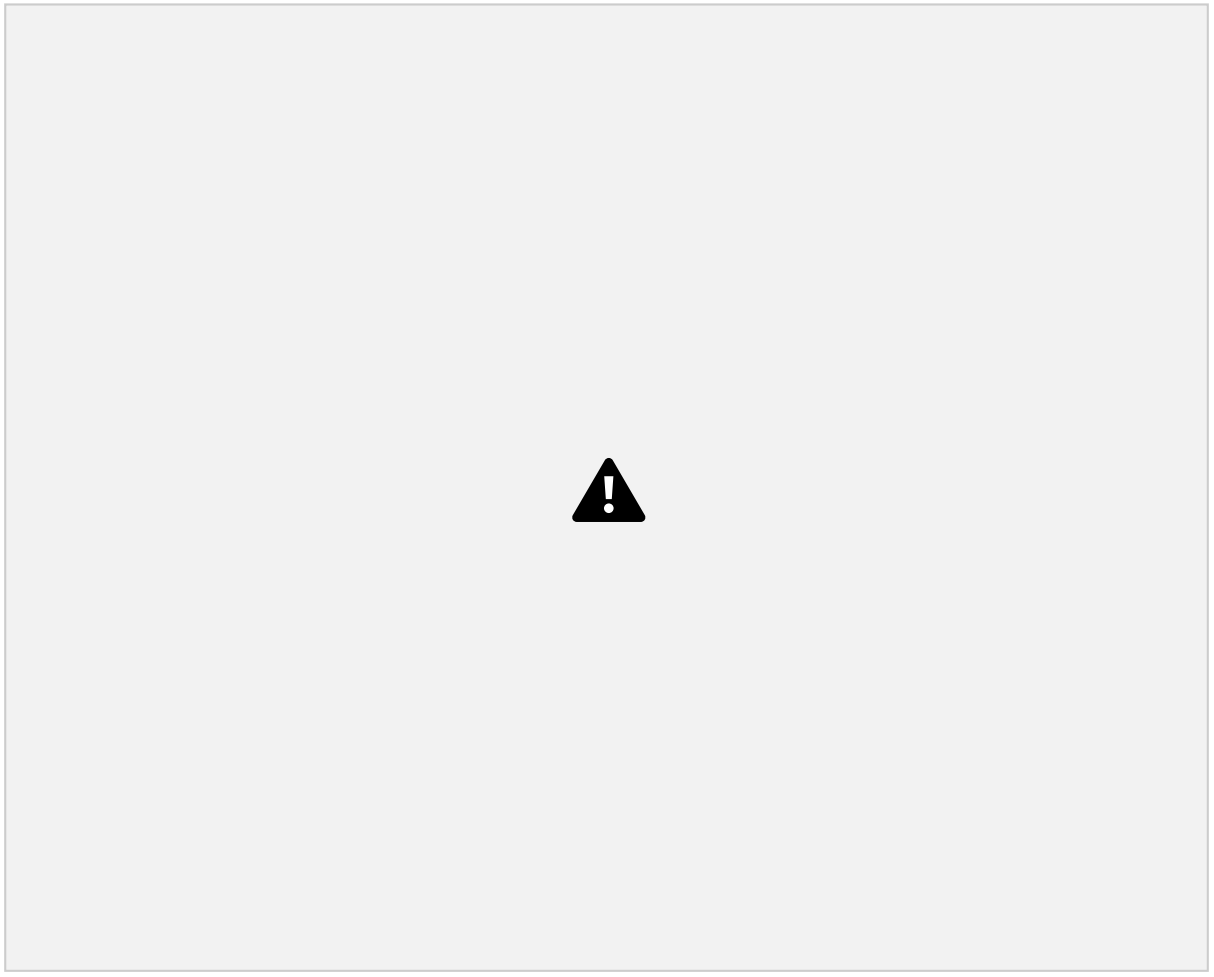
i. **Importing module** :-

- Modules can import other modules.
- It is customary but not required to place all import statements at the beginning of a module (or script, for that matter).
- The imported module names are placed in the importing module's global symbol table. o Eg :import fibo
- Using the module name we can access functions
- fibo.fib(10)
- fibo.fib2(10)
- There is a variant of the import statement that imports names from a module directly into the importing module's symbol table.
- For example:
- from fibo import fib, fib2
- fib(500)
- another way to import is
- from fibo import *

ii. Math module :-

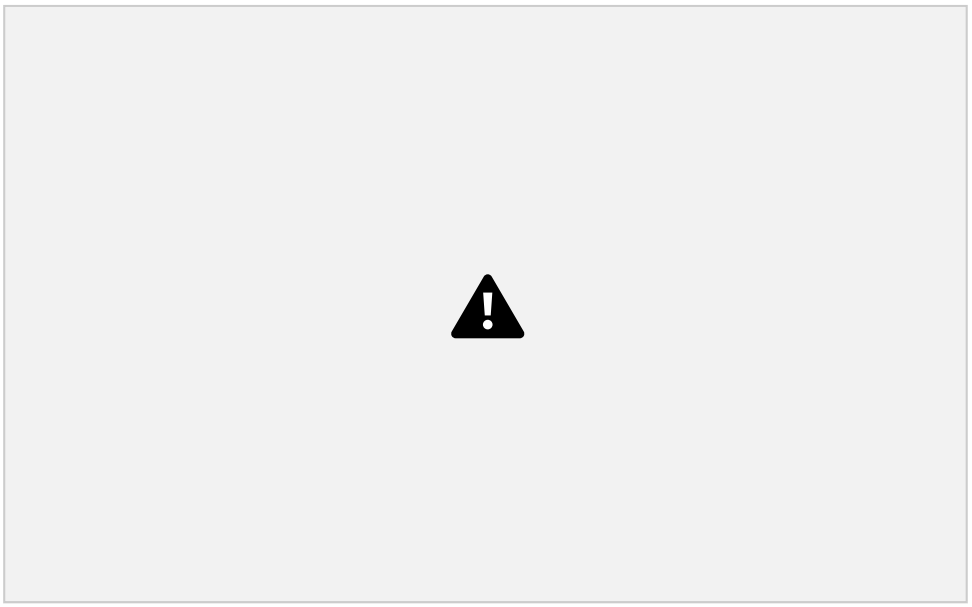
- This module is always available.
- It provides access to the mathematical functions defined by the C standard.
- This functions are divided into some categories like Number-theoretic and representation functions, Power and logarithmic functions, Trigonometric functions, Angular conversion, Hyperbolic functions, Special functions. • Constants
- math.pi : The mathematical constant π = 3.141592..., to available precision.
- math.e : The mathematical constant e = 2.718281..., to available precision.,

- math.inf : A floating-point positive infinity. (For negative infinity, use -math.inf.) Equivalent to the output of float('inf').
- math.nan : A floating-point “not a number” (NaN) value. Equivalent to the output of float('nan').



iii. Random module :-

- Python offers random module that can generate random numbers
- These are pseudo-random number as the sequence of number generated depends on the seed.
- Random module functions :-



iv. Packages :-

- The packages in python facilitate the developer with the application development environment by providing a hierarchical directory structure where a package contains sub-packages, modules, and sub-modules. The packages are used to categorize the application level code efficiently.
- Let's create a package named Employees in your home directory. Consider the following steps.
  1) Create a directory with name Employees on path /**home**.
  2) Create a python source file with name ITEmployees.py on the path /**home**/**Employees**.

ITEmployees.py

Similarly, create one more python file with name BPOEmployees.py and create a function getBPONames().

Now, the directory Employees which we have created in the first step contains two python modules. To make this directory a package, we need to include one more file here, that is __init__.py which contains the import statements of the modules defined in this directory.

__init__.py

Now, the directory **Employees** has become the package containing two python modules. Here we must notice that we must have to create __init__.py inside a directory to convert this directory to a package.

To use the modules defined inside the package Employees, we must have to import this in our python source file. Let's create a simple python source file at our home directory (/home) which uses the modules defined in this package.

Test.py

AGOG TECHNOLOGIES

❖ **Files I/O** :-

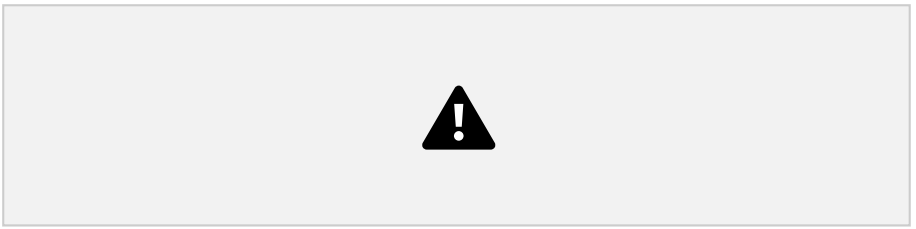
- Printing on screen
- Reading data from keyboard
- Opening and closing file
- Reading and writing files
- Functions

○ Printing on screen :-

- "print()" is use to print on screen.
- print(value, ..., sep=' ', end='\n', file=sys.stdout)
- Prints the values to a stream, or to sys.stdout by default.
- Optional keyword arguments:
- file: a file-like object (stream); defaults to the current sys.stdout. • sep: string inserted between values, default a space.
- end: string appended after the last value, default a newline.

○ Reading data from keyboard :-

- To read data from keyboard "input()" is use.
- The input of the user will be returned as a string without any changes. • If this raw input has to be transformed into another data type needed by the algorithm, we can use either a casting function or the eval function.



○ Opening & Closing File :-

- **"open()" is use to open the file**
- It returns file object.
- syntax
- open(fileName,mode)
- fileName: Name of the file that we wants to open.
- mode: 'r' (only for reading), 'w' (only for writing), 'a' (for append) , r+ (for read and write).
- Normally, files are opened in text mode, that means, you read and write strings from and to the file, which are encoded in a specific encoding.
- If encoding is not specified, the default is platform dependent .
- 'b' appended to the mode opens the file in binary mode.
- **Close the file**

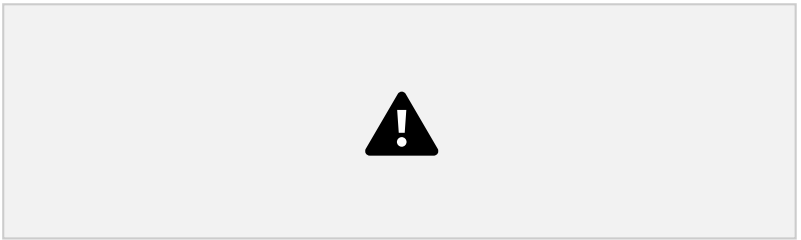
• call f.close() to close it and free up any system resources taken up by the open file.

- **Reading & writing files :-**

• file.read(size) :This function is use to read a file's contents.
• If size is ommited or negative then the entire content is returnd.
• If the end of the file has been reached, f.read() will return an empty string (' '). • f.readline() reads a single line from the file; a newline character (\n) is left at the end of the string, and is only omitted on the last line of the file if the file doesn't end in a newline.
• For reading lines from a file, you can loop over the file object.
• This is memory efficient, fast, and leads to simple code: for line in f: print(line, end='')
• **f.write(string)** , writes the contents of string to the file, returning the number of characters written
• Other types of objects need to be converted – either to a string (in text mode) or a bytes object (in binary mode) –before writing them.
Eg :value = ('the answer', 42)
s = str(value) # convert the tuple to string
f.write(s)

- **File function :-**

• f.tell() : It returns an integer giving the file object's current position in the file represented as number of bytes from the beginning of the file when in binary mode and an opaque number when in text mode.
• f.seek(): To change the file object's position.
• f.seek(offset, from_what)
• Eg : f.seek(5)
• f.seek(-3,2)



❖ Exception :-

• An exception can be defined as an unusual condition in a program resulting in the interruption in the flow of the program.

• Whenever an exception occurs, the program stops the execution, and thus the further code is not executed. Therefore, an exception is the run-time errors that are unable to handle to Python script. An exception is a Python object that represents an error • Exception Handling

• Exception
• Except clause
• Try …. finally clause
• User Defined Exceptions

➢ Exception Handling :-

• If the Python program contains suspicious code that may throw the exception, we must place that code in the **try** block. The **try** block must be followed with the **except** statement, which contains a block of code that will be executed if there is some exception in the try block.

try:

```
# block of code
except Exception1:
        # block of code
Except Exception2:
        # block of code
```

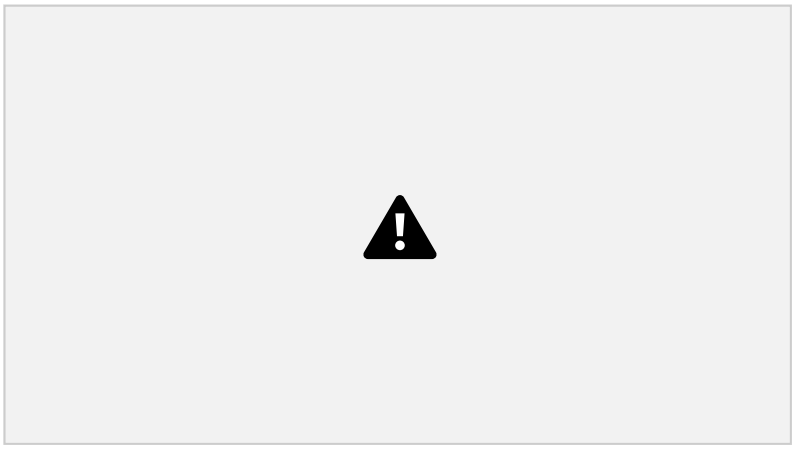
Ex:-

➢ **Except clause:-**

A try statement may have more than one except clause for different exceptions. But at most one except clause will be executed.

Eg :

```
import sys
try:
 ……
 ……
except IOError:
 …….
except :
 sys.exc_info()[0]
```



➢ Try …..finally clause :-

try statement had always been paired with except clauses. But there is another way to use it as well.

The try statement can be followed by a finally clause.

Finally clauses are called clean-up or termination clauses, because they must be executed under all circumstances, i.e. a "finally" clause is always executed regardless if an exception occurred in a try block or not.

Syntax:

try:

……

……

finally :

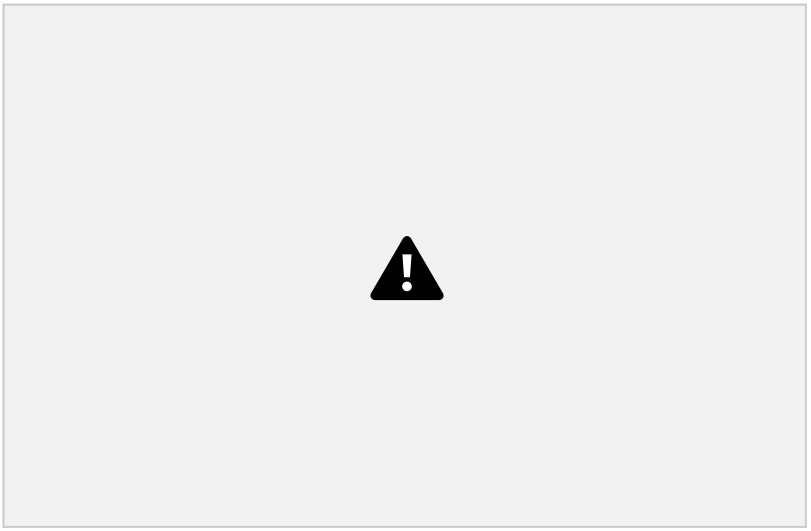
……

➢ User defined function :-

- Python also allows you to create your own exceptions by deriving classes from the standard built-in exceptions.

  class MyNewError(Exception):

  pass



❖ Advance Python :-

- ➢ Oops in python
- ➢ Regular Expression
- ➢ CGI
- ➢ File upload
- ➢ Database
- ➢ Networking
- ➢ Multithreading
- ➢ GUI programming

➢ Oops Concept :-

1. Class & Objects
2. Attributes
3. Inheritance
4. Overloading
5. Overriding
6. Data handling

• Class & Object :-

Class :-

A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.

Python classes provide all the standard features of Object Oriented Programming: the class inheritance mechanism allows multiple base classes, a derived class can override any methods of its base class or classes, and a method can call the method of a base class with the same name.

The class definition looks like :

```
class ClassName:
        Statement 1
         Statement 2
        …….
        Statement N
```

- ○ The statements inside a class definition will usually be function definitions, but other statements are also allowed.
- ○ When a class definition is entered, a new namespace is created, and used as the local scope—thus, all assignments to local variables go into this new namespace. ○ In particular, function definitions bind the name of the new function here

Object :-

- A unique instance of a data structure that's defined by its class. An object comprises both data members (class variables and instance variables) and methods.
- Class objects support two kinds of operations: attribute references and instantiation
- Attribute references use the standard syntax used for all attribute references in Python: obj.name
- Valid attribute names are all the names that were in the class's namespace when the class object was created..
- Class instantiation uses function notation.
- Just pretend that the class object is a parameterless function that returns a new instance of the class
- For example : x = MyClass()
- creates a new instance of the class and assigns this object to the local variable x.

Member Method in class :-

- The class_suite consists of all the component statements defining class members, data attributes and functions
- The class attributes are data members (class variables and instance variables) and methods, accessed via dot notation.
- Eg. displayDetails()

Static method in python :-

- Static methods in Python are similar to those found in Java or C++. ▪
A static method does not receive an implicit first argument.

- To declare a static method, use this idiom:

- class C:

```
def f(arg1, arg2, ...):
 f = staticmethod(f)
        staticmethod(function) -> method
```

- Convert a function to be a static method
- It can be called either on the class (e.g. C.f()) or on an instance
- (e.g. C().f()). The instance is ignored except for its class.

• **Inheritance :-**

The transfer of the characteristics of a class to other classes that are derived from it.

• **Overloading:-**

1) Method over loading :-

- Overloading is the ability to define the same method, with the same name but with a different number of arguments and types
- It's the ability of one function to perform different tasks, depending on the number of parameters or the types of the parameters.

2) Operator over loading :-

- The assignment of more than one function to a particular operator. •

Overriding :-

Whereas in the method **overriding**, methods or functions must have the same name and same signatures. ... Whereas method **overriding** is done between parent class and child class methods.

• **Data handling :-**

An object's attributes may or may not be visible outside the class definition. You need to name attributes with a double underscore prefix, and those attributes then are not be directly visible to outsiders.

➢ Regular Expression :

The regular expressions can be defined as the sequence of characters which are used to search for a pattern in a string. The module re provides the support to use regex in the python program. The re module throws an exception if there is some error while using the regular expression.

• match function :-

This method matches the regex pattern in the string with the optional flag. It returns true if a match is found in the string otherwise it returns false.

• search function :-

This method returns the match object if there is a match found in the string.

• findall :-

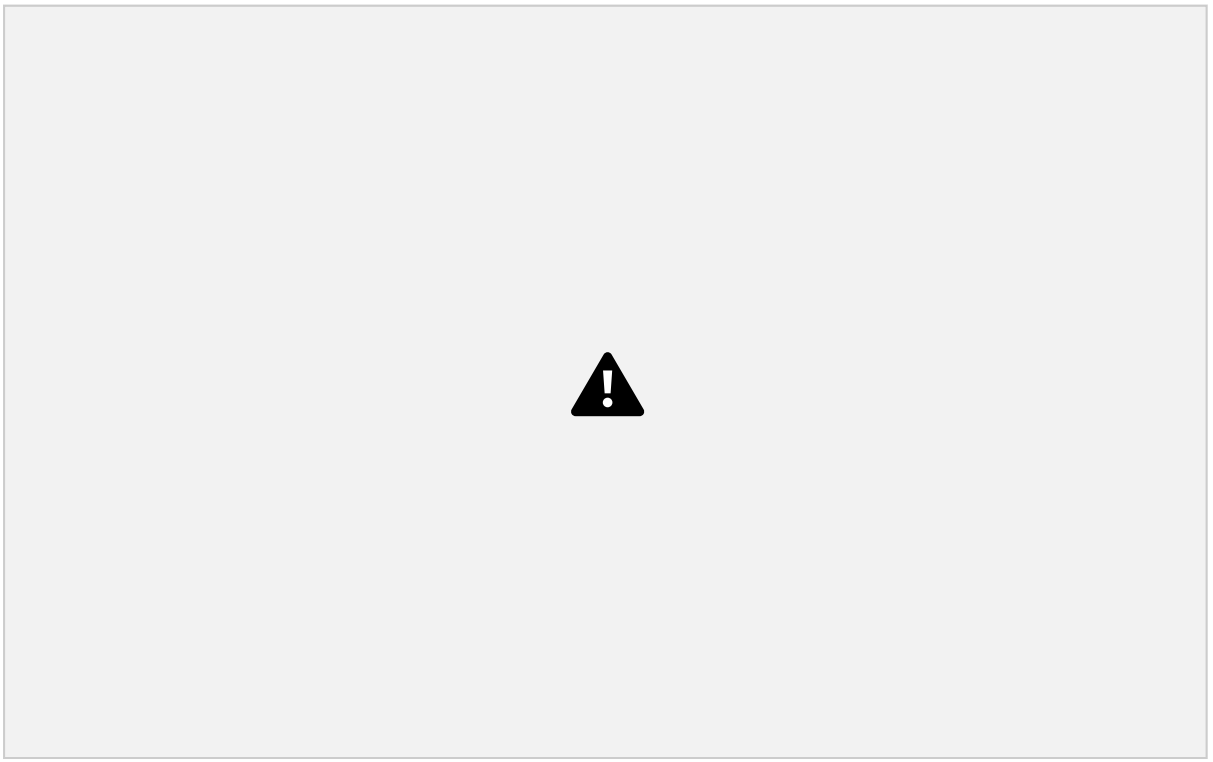
It returns a list that contains all the matches of a pattern

in the string.

• split :-

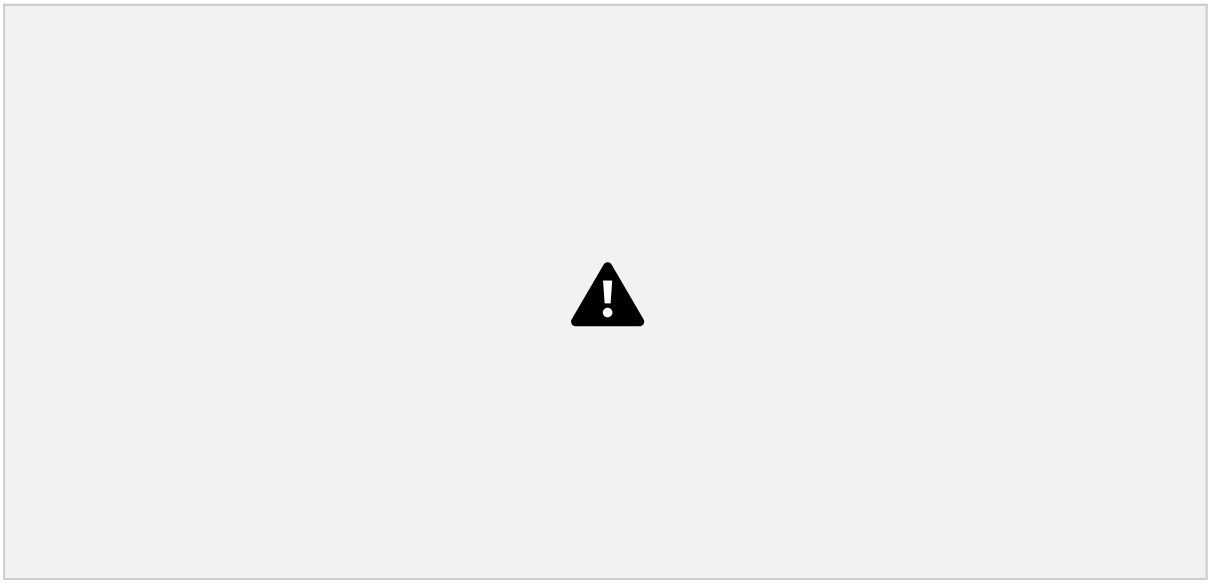
Returns a list in which the string has been split in each match.

• sub :- Replace one or many matches in the string.



Sets

A set is a group of characters given inside a pair of square brackets. It represents the special meaning.

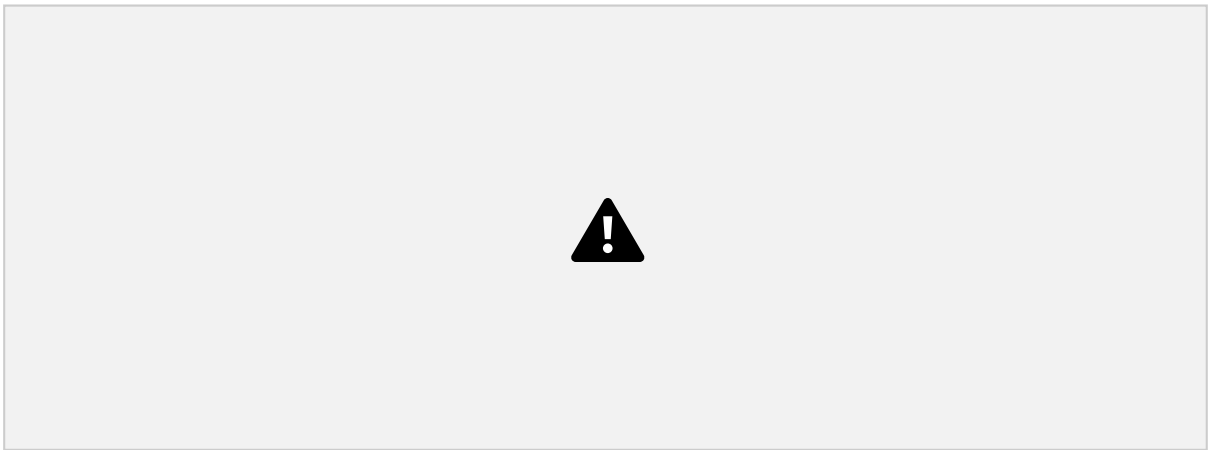


➢ CGI programing :-

What is CGI ? :-

The Common Gateway Interface, or CGI, is a standard for external gateway programs to interface with information servers such as HTTP servers.

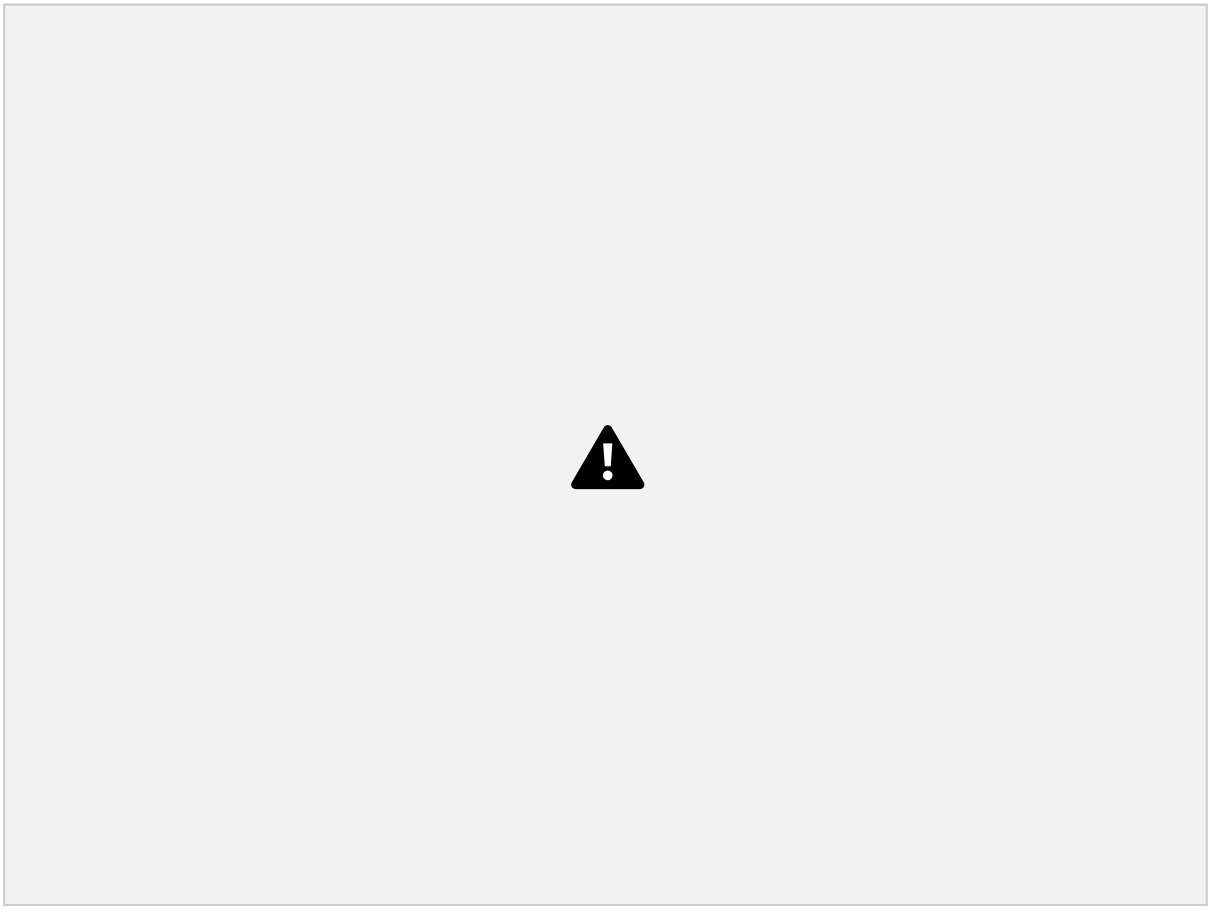
CGI Architecture Diagram :-

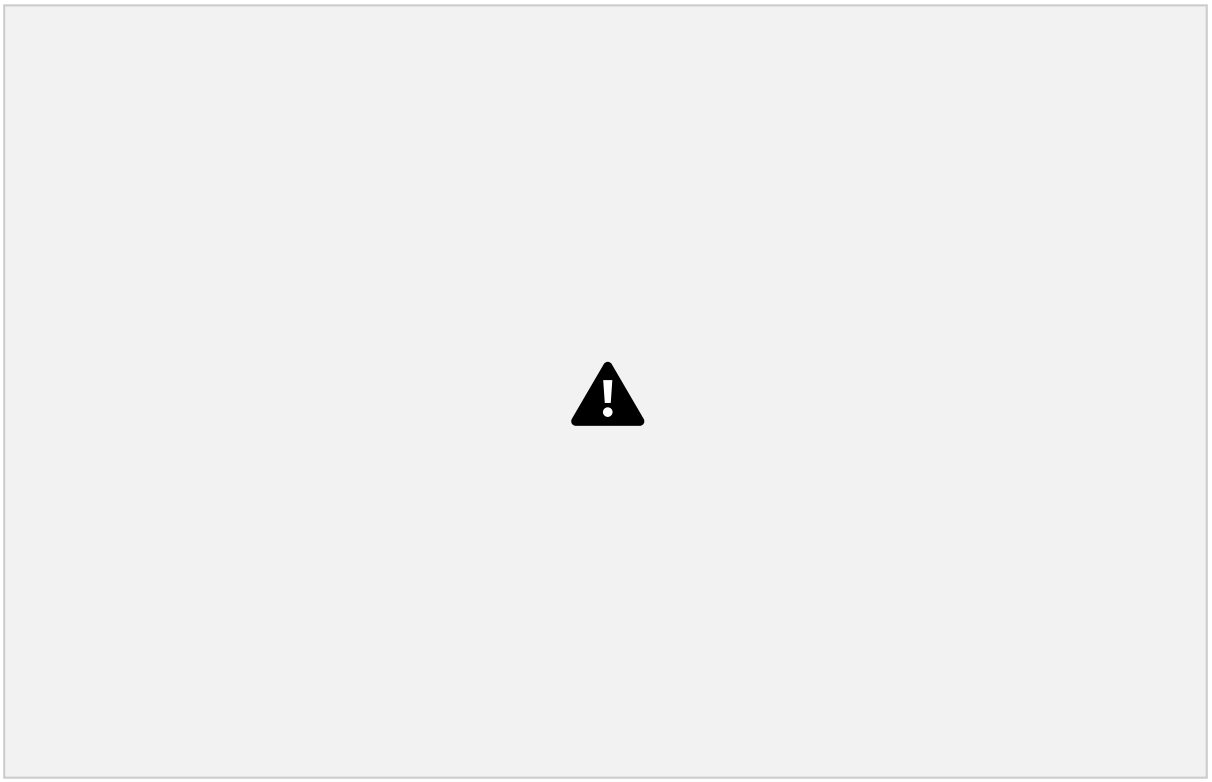


CGI environment variables :-

On next page.

AGOG TECHNOLOGIES





Get and Post method :-

You must have come across many situations when you need to pass some information from your browser to web server and ultimately to your CGI Program. Most frequently, browser uses two methods two pass this information to web server. These methods are GET Method and POST Method.

Cookies :-

You must have come across many situations when you need to pass some information from your browser to web server and ultimately to your CGI Program. Most frequently, browser uses two methods two pass this information to web server. These methods are GET Method and POST Method.

In many situations, using cookies is the most efficient method of remembering and tracking preferences, purchases, commissions, and other information required for better visitor experience or site statistics.

How it works :-

Your server sends some data to the visitor's browser in the form of a cookie. The browser may accept the cookie. If it does, it is stored as a plain text record on the visitor's hard drive. Now, when the visitor arrives at another page on your site, the cookie is available for retrieval. Once retrieved, your server knows/remembers what was stored.

Cookies are a plain text data record of 5 variable-length fields :

Expires − The date the cookie will expire. If this is blank, the cookie will expire when the visitor quits the browser.

Domain − The domain name of your site.

Path − The path to the directory or web page that sets the cookie. This may be blank if you want to retrieve the cookie from any directory or page.

Secure − If this field contains the word "secure", then the cookie may only be retrieved with a secure server. If this field is blank, no such restriction exists.

Name=Value − Cookies are set and retrieved in the form of key and value pairs.

❖ File upload :-

To upload a file, the HTML form must have the enctype attribute set to multipart/form-data. The input tag with the file type creates a "Browse" button.

```
<html>
<body>
 <form enctype=”multipart/form-data”
                    action=”save_file.py” method=”psot”>
<p>File:<input type=”file” name=”filename”/></p>
<p><input type=”submit” value=”Upload” /></p>
 </form>
</body>
</html>
```

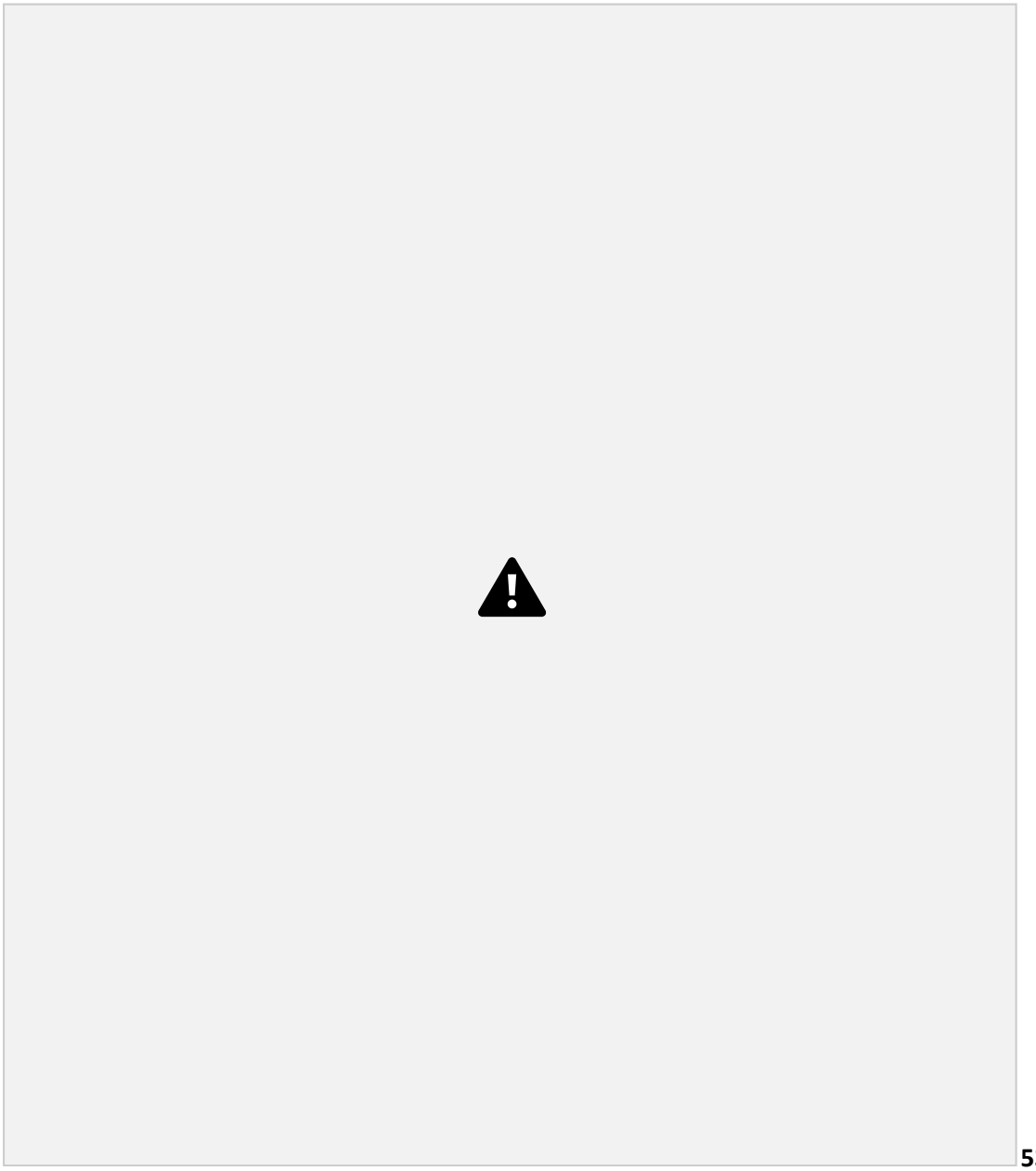
❖ Database :-

- Introduction
- Connections
- Executing queries
- Transaction
- Handling error

Introduction :-

- SQLite is a C library that provides a lightweight disk-based database that doesn't require a separate server process and allows accessing the database using a nonstandard variant of the SQL query language.
- Connections
- To use the module, you must first create a Connection object that represents the database. Here the data will be stored in the example.db file:
- importsqlite3
- conn=sqlite3.connect('example.db')
- You can also supply the special name :memory: to create a database in RAM. • Executing queries
- Once you have a Connection, you can create a Cursor object and call its execute() method to perform SQL commands.
- There are execute(),executemany(),executescript() methods to execute queries. • Transaction
- By default, the sqlite3 module opens transactions implicitly before a Data Modification Language (DML) statement (i.e.INSERT/UPDATE/DELETE/REPLACE), and commits transactions implicitly before a non-DML, non-query statement .
- Atomicity: Either a transaction completes or nothing happens at all. • Consistency: A transaction must start in a consistent state and leave the system in a consistent state.
- Isolation: Intermediate results of a transaction are not visible outside the current transaction

- Durability: Once a transaction was committed, the effects are persistent, even after a system failure.
- The Python DB API 2.0 provides two methods to either commit or rollback a transaction.



➢ Networking :-

- Socket
- Socket module
- Methods
- Client and server
- Internet modules

• Socket :-

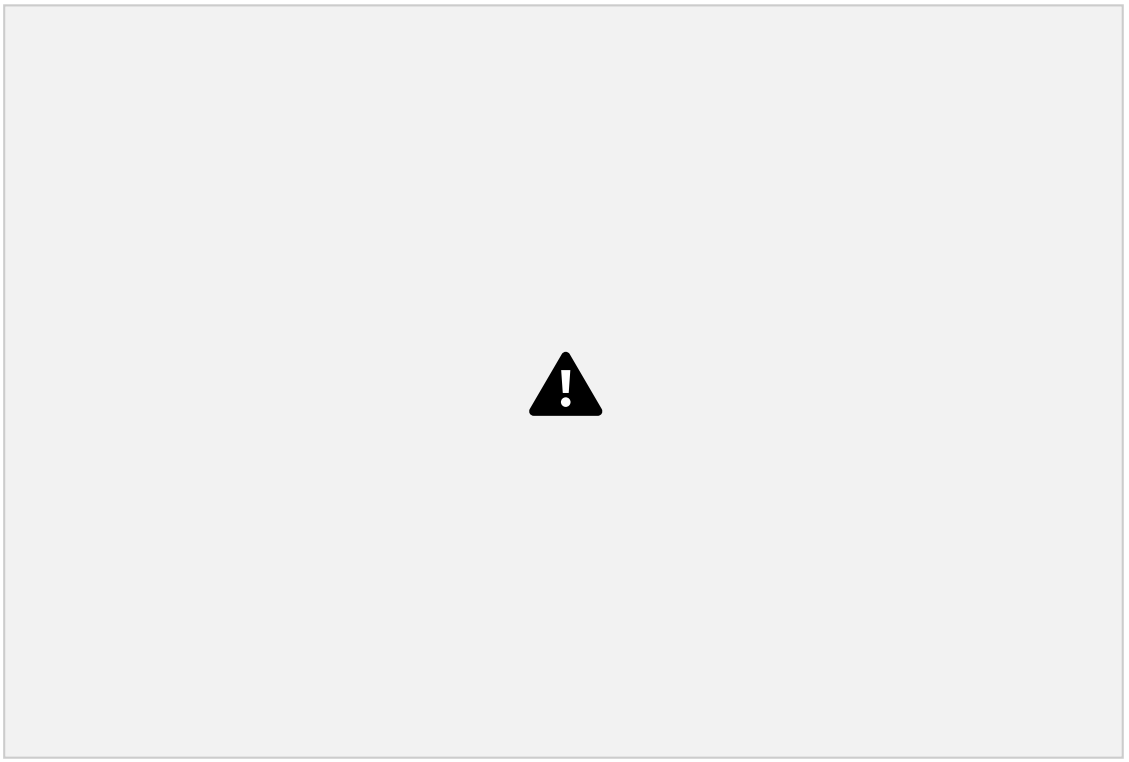
- Sockets are the endpoints of a bidirectional communications channel.
- Sockets may communicate within a process, between processes on the same machine, or between processes on different continents.

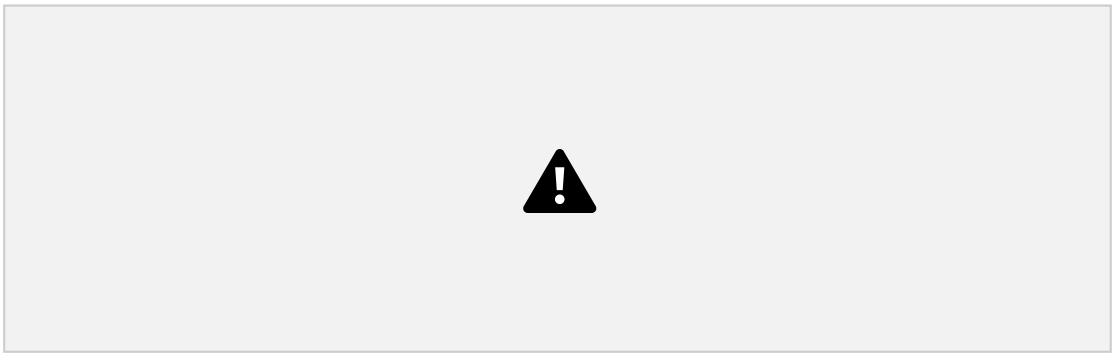
- **Socket module :-**
    - To create/initialize a socket, we use the socket.socket() method defined in the Python's socket module.
    - Its syntax is as follows. o sock_obj = socket.socket(socket_family, socket_type, protocol=0)
    - socket_family: This is either AF_UNIX or AF_INET,
    - AF_INET: IP version 4 or IPv4
    - AF_UNIX : Unix socket
    - socket_type

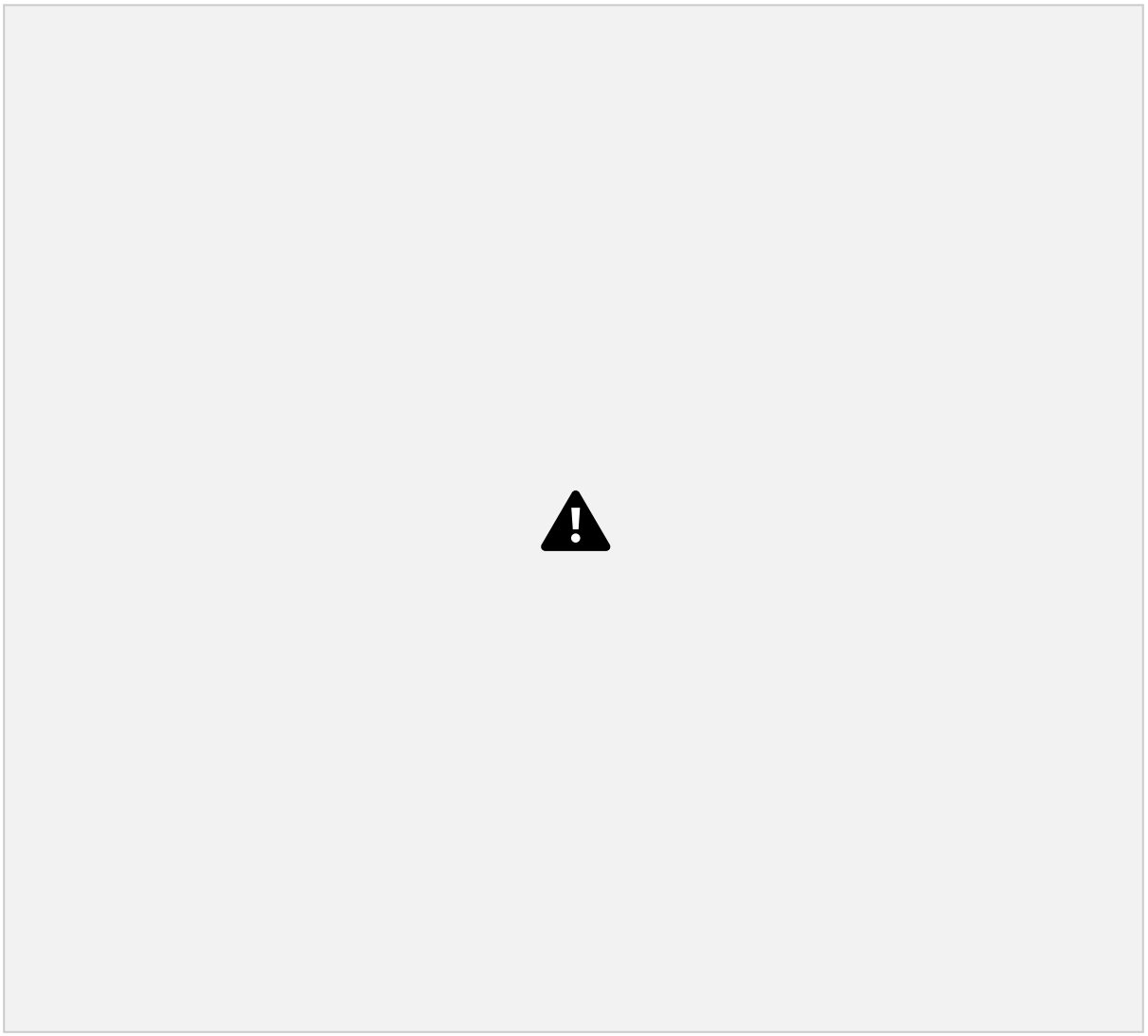
- **Socket_type :-**
    - SOCK_STREAM (for connection-oriented protocols e.g. TCP), or
    - SOCK_DGRAM (for connectionless protocols e.g. UDP).
    - SOCK_RAW (For Raw socket)

- **Protocol :-**
    - It's usually used with raw sockets. Like IPPROTO_ICMP,
    - IPPROTO_IP,IPPROTO_RAW,IPPROTO_TCP,IPPROTO_UDP

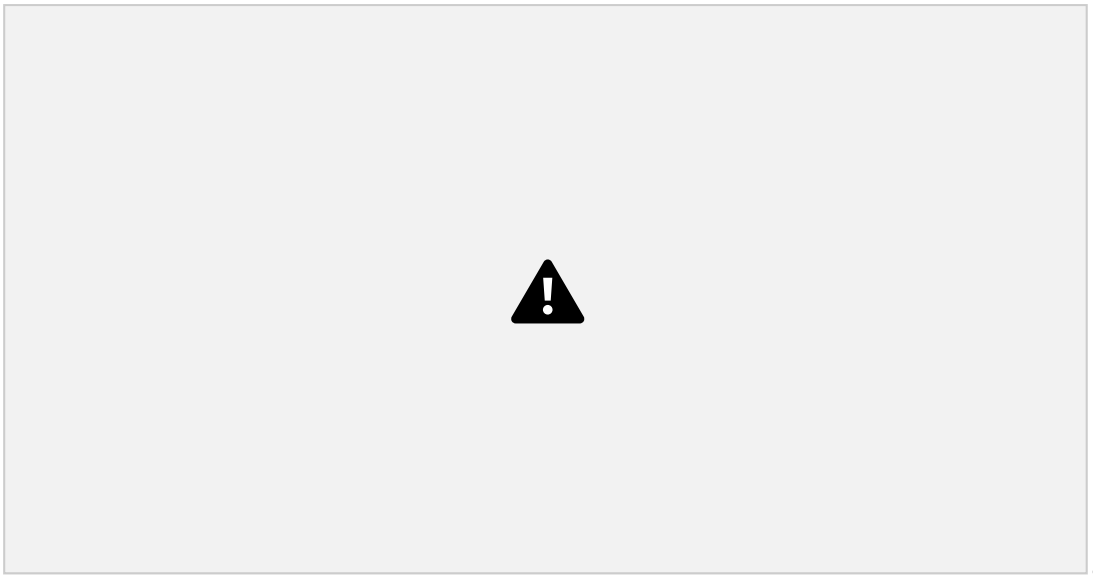
- **Methods :-**



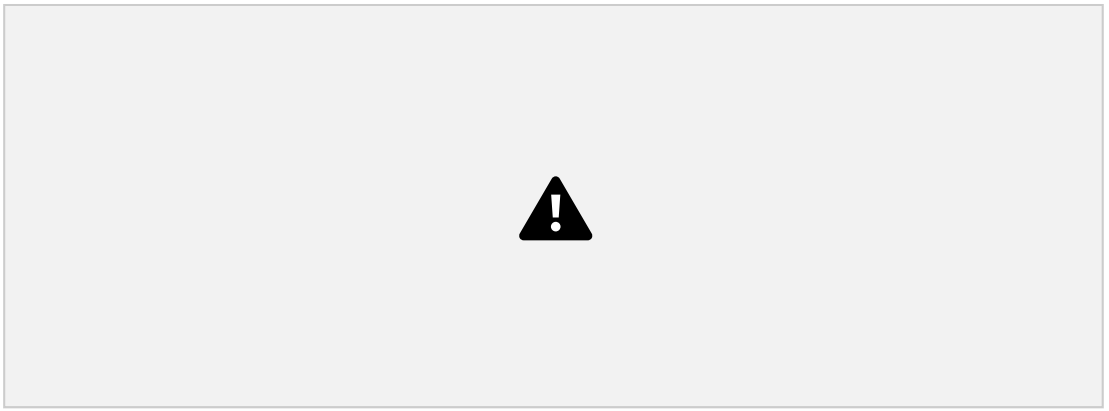




Simple Server :-



Simple Client :-



Result :-

➢ **Multithreading :-**

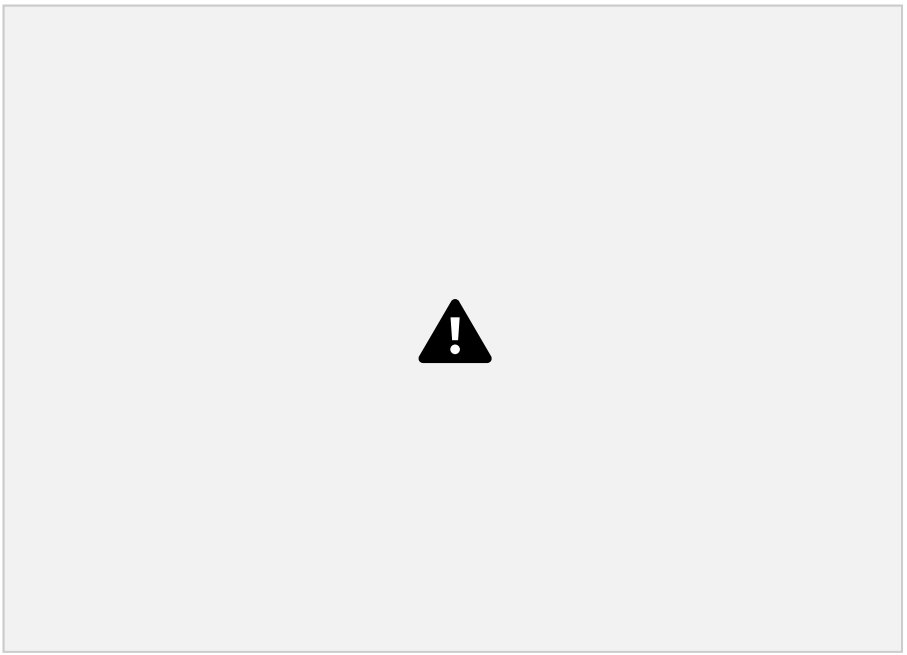
What is thread?

- A Thread or a Thread of Execution is defined in computer science as the smallest unit that can be scheduled in an operating system.
- Threads are usually contained in processes.
- More than one thread can exist within the same process.
- Every process has at least one thread, i.e. the process itself.
- A process can start multiple threads.

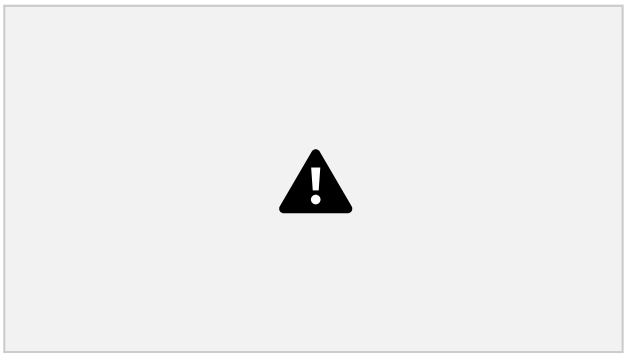
How To start New Thread?

- This method call enables a fast and efficient way to create new threads in both Linux and Windows
- The method call returns immediately and the child thread starts and calls function with the passed list of *args*. When function returns, the thread terminates.
- Here, *args* is a tuple of arguments; use an empty tuple to call function without passing any arguments. *kwargs* is an optional dictionary of keyword arguments.

Example:



Output:



The *Threading* Module

The newer threading module included with Python 2.4 provides much more powerful, high-level support for threads than the thread module discussed in the previous section.

The *threading* module exposes all the methods of the *thread* module and provides some additional methods −

- **threading.activeCount()** − Returns the number of thread objects that are active.
- **threading.currentThread()** − Returns the number of thread objects in the caller's thread control.
- **threading.enumerate()** − Returns a list of all thread objects that are currently active.

In addition to the methods

- **run()** − The run() method is the entry point for a thread.
- **start()** − The start() method starts a thread by calling the run method. •

join([time]) − The join() waits for threads to terminate.

- **isAlive()** − The isAlive() method checks whether a thread is still executing. •

getName() − The getName() method returns the name of a thread. •

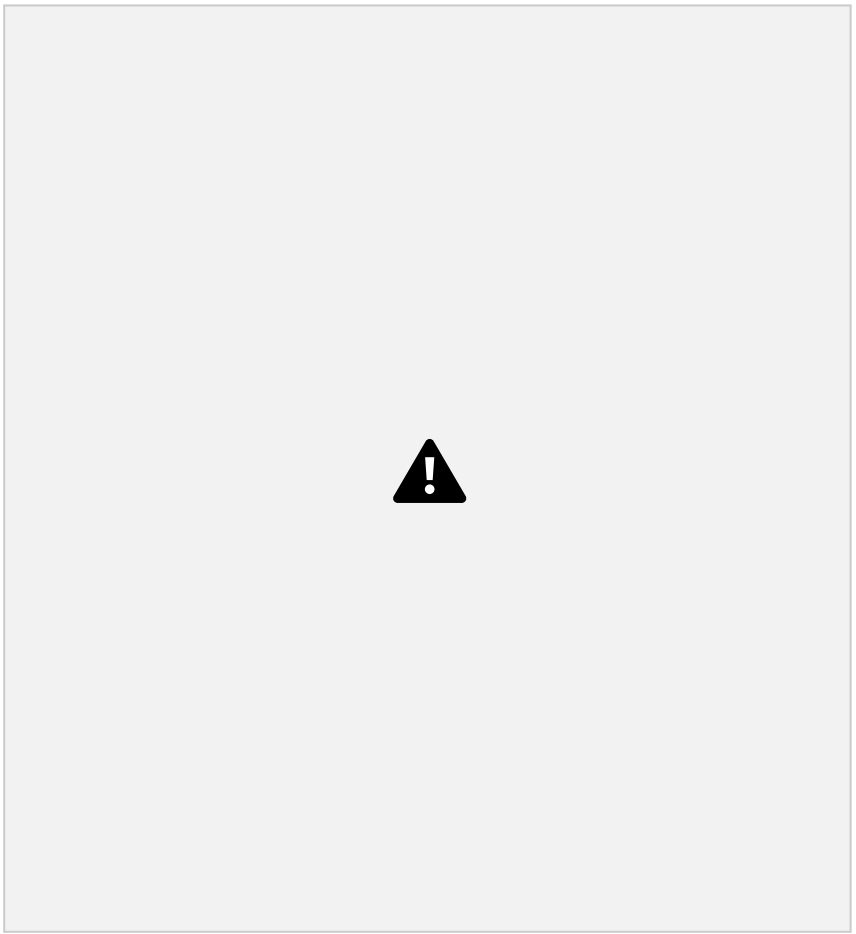
setName() − The setName() method sets the name of a thread.

Creating Thread Using *Threading* Module

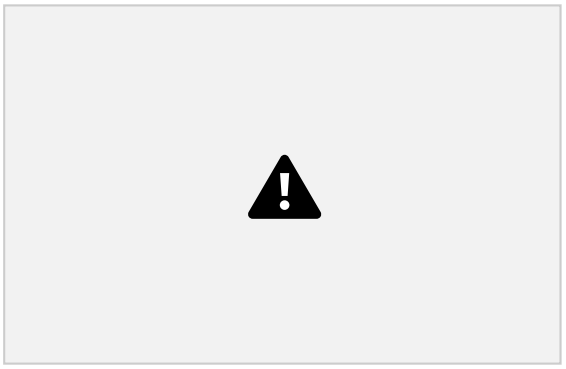
To implement a new thread using the threading module, you have to do the following

- Define a new subclass of the *Thread* class.
- Override the *__init__(self [,args])* method to add additional arguments.
- Then, override the run(self [,args]) method to implement what the thread should do when started.

Once you have created the new *Thread* subclass, you can create an instance of it and then start a new thread by invoking the *start()*, which in turn calls *run()* method.



Output:



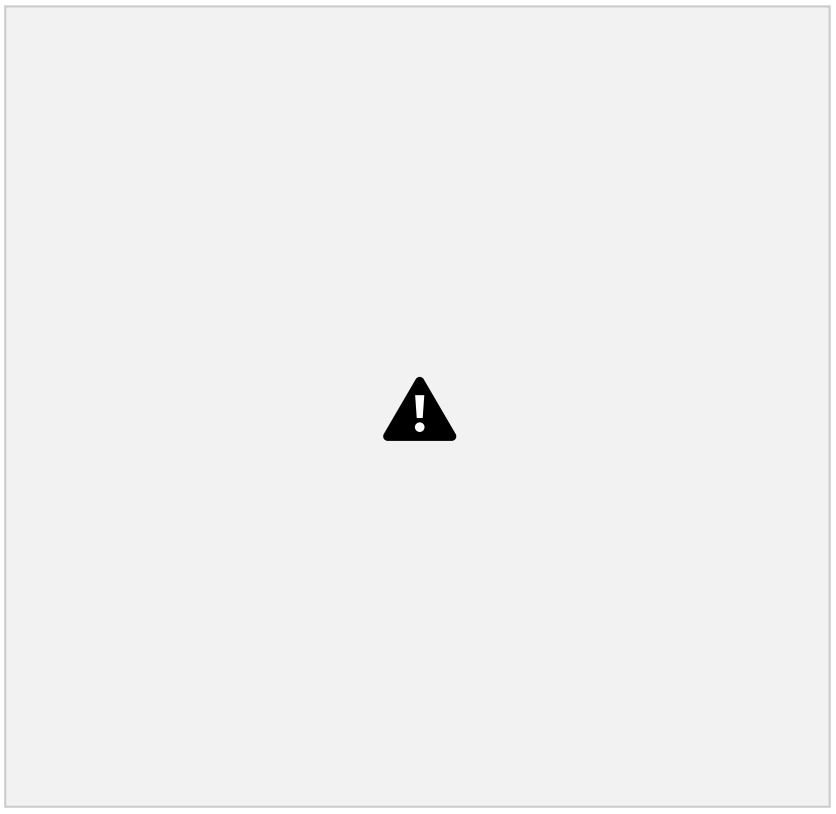
Synchronizing Threads

The threading module provided with Python includes a simple-to-implement locking mechanism that allows you to synchronize threads. A new lock is created by calling the *Lock()* method, which returns the new lock.

The *acquire(blocking)* method of the new lock object is used to force threads to run synchronously. The optional *blocking* parameter enables you to control whether the thread waits to acquire the lock.

If *blocking* is set to 0, the thread returns immediately with a 0 value if the lock cannot be acquired and with a 1 if the lock was acquired. If blocking is set to 1, the thread blocks and wait for the lock to be released.

The *release()* method of the new lock object is used to release the lock when it is no longer required.



Multithreaded Priority Queue

The *Queue* module allows you to create a new queue object that can hold a specific number of items. There are following methods to control the Queue

- **get()** − The get() removes and returns an item from the queue. •

put() − The put adds item to a queue.

- **qsize()** − The qsize() returns the number of items that are currently in the queue.
- **empty()** − The empty( ) returns True if queue is empty; otherwise, False. •

full() − the full() returns True if queue is full; otherwise, False.

❖ Django :

• Introduction:

- Django is a Web Application Framework which is used to develop web applications.
- Django is a web application framework written in Python programming language. It is based on MVT (Model View Template) design pattern. The Django is very demanding due to its rapid development feature. It takes less time to build application after collecting client requirement.
- This framework uses a famous tag line:**The web framework for**

perfectionists with deadlines.

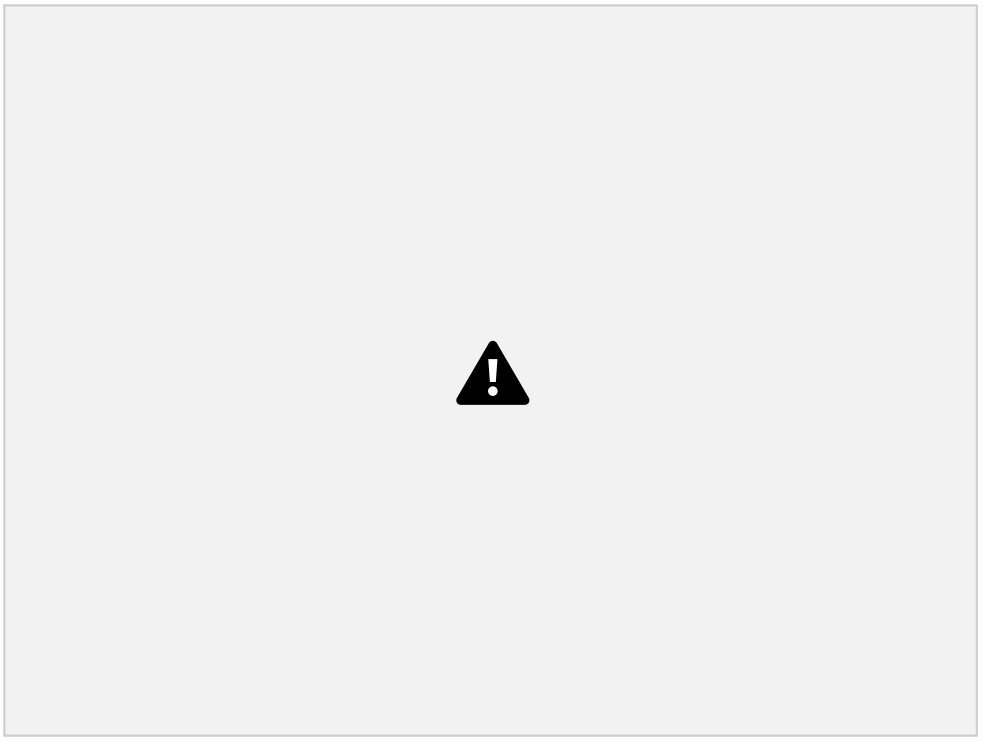
- By using Django, we can build web applications in very less time. Django is designed in such a manner that it handles much of configure things automatically, so we can focus on application development only.
- Django was released on 21, July 2005. Its current stable version is 2.0.3 which was released on 6 March, 2018.

Features of Django:

- Rapid Development
- Secure
- Scalable
- Fully loaded
- Versatile
- Open Source
- Vast and Supported Community

How Django Works:

- When a request comes to a web server, it's passed to Django which tries to figure out what is actually requested
- It takes a web page address first and tries to figure out what to do. This part is done by Django's urlresolver (note that a website address is called a URL – Uniform Resource Locator – so the name urlresolver makes sense).
- It is not very smart – it takes a list of patterns and tries to match the URL. Django checks patterns from top to bottom and if something is matched, then Django passes the request to the associated function (which is called view).



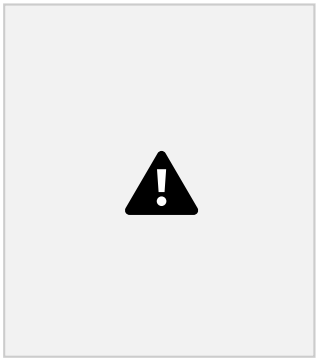
• Django Installation:

Virtual environment

- Django requires **pip** to start installation. Pip is a package manager system which is used to install and manage packages written in python. For Python 3.4 and higher versions **pip3** is used to manage packages.
- Before we install Django we will get you to install an extremely useful tool to help keep your coding environment tidy on your computer
- It's possible to skip this step, but it's highly recommended.
- So, let's create a virtual environment (also called a virtualenv). Virtualenv will isolate your Python/Django setup on a per-project basis.
- To create a new virtualenv,you need to open the console and run C:\Python35\python –m venv myvenv.
- Working with virtualenv
- The command above will create a directory called 'myvenv' that contains our virtual environment.
- Start your virtual environment by running :
- Command line **: C:\Users\Name\djangoproject >myvenv\Scripts\activate**
- Now that you have your virtualenv started ,you can install Django • Before we do that, we should make sure we have the latest version of pip, the software that we use to install Django.
- pip install django==2.0 or pip install django (by this install latest version of Django)

Start The Project

- The first step is to start a new Django project.
- Basically, this means that we'll run some scripts provided by Django that will create the skeleton of a Django project for us.
- This is just a bunch of directories and files that we will use later. • Command-line**: (myvenv)C:\Users\Name\djangoproject>django-admin.py startproject mysite .** • On Windows don't forget to add the period (or dot) '.' at the end • Here 'django-admin.py' is a script that will create the directories and files for you.
- You should now have a directory structure which looks like this:



- Here 'manage.py' is a script that helps with management of the site. With it we will be able (amongst other things) to start a web server on our computer without installing anything else
- The settings.py file contains the configuration of your website. • urls.py file contains a list of patterns used by urlresolver.

Database Setup

There's a lot of different database software that can store data for your site. We'll use the default one, sqlite3.

This is already set up in this part of your mysite/settings.py file.

mysite/settings.py

```
DATABASES = { 'default': {
                    'ENGINE': 'django.db.backends.sqlite3',
                   'NAME': 'mydatabase',
                    }
            }
```

To connect with MySQL, **django.db.backends.mysql** driver is used to establishing a connection between application and database. Let's see an example.

We need to provide all connection details in the settings file. The settings.py file of our project contains the following code for the database

For mysql database :

```
DATABASES = {
'default': {
'ENGINE': 'django.db.backends.mysql',
'NAME': 'djangoApp',
'USER':'root',
'PASSWORD':'mysql',
'HOST':'localhost',
'PORT':'3306'
}
}
```

For Postgres database :

```
DATABASES = { 'default': {
                    'ENGINE': 'django.db.backends.postgresql_psycopg2',
                    'NAME': 'todo',
                    'USER': 'postgres',
                    'PASSWORD': 'admin',
                    'HOST': 'localhost',
                    'PORT': '5432',
                    }
            }
```

Now migration is needed for settings configuration.For this we needs to run. Command-line**: (myvenv)C:\Users\Name\djangoproject>** python manage.py migrate

Starting the web server : Command-line **:**
myvenv)C:\Users\Name\djangoproject> $ python manage.py runserver

Now all you need to do is check that your website is running. Open your browser (Firefox, Chrome, Safari, Internet Explorer or whatever you use) and enter this address:

AGOG TECHNOLOGIES http://127.0.0.1:8000/

Create Application :

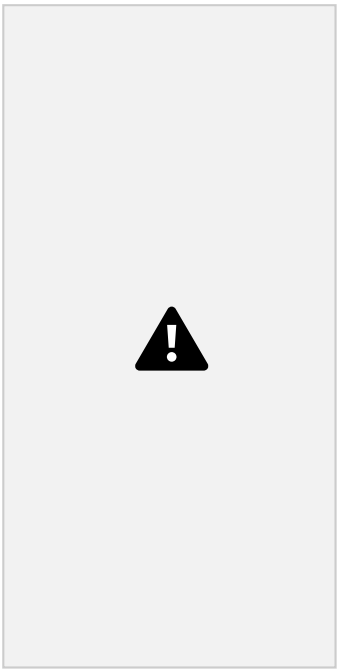
• A model in Django is a special kind of object --it is saved in the database • Model in database as a spreadsheet with columns(fields) and rows (data). • Creating an application

• To create an application we need to run the following command in the console **$ python manage.py startapp appname**

• You will notice that a new 'appname' directory is created and it contains a number of files now.

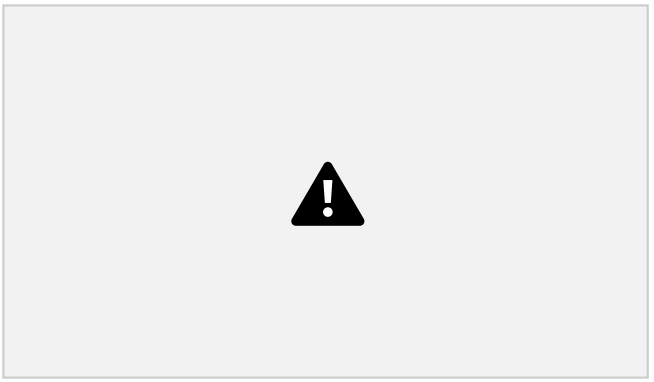
• The directories and files in our project should look like this.

Here I am give app name 'app1'



After creating an application, we also need to tell Django that it should use it. We do that in the file mysite/settings.py.

We need to find INSTALLED_APPS and add a line containing 'app1 just above .

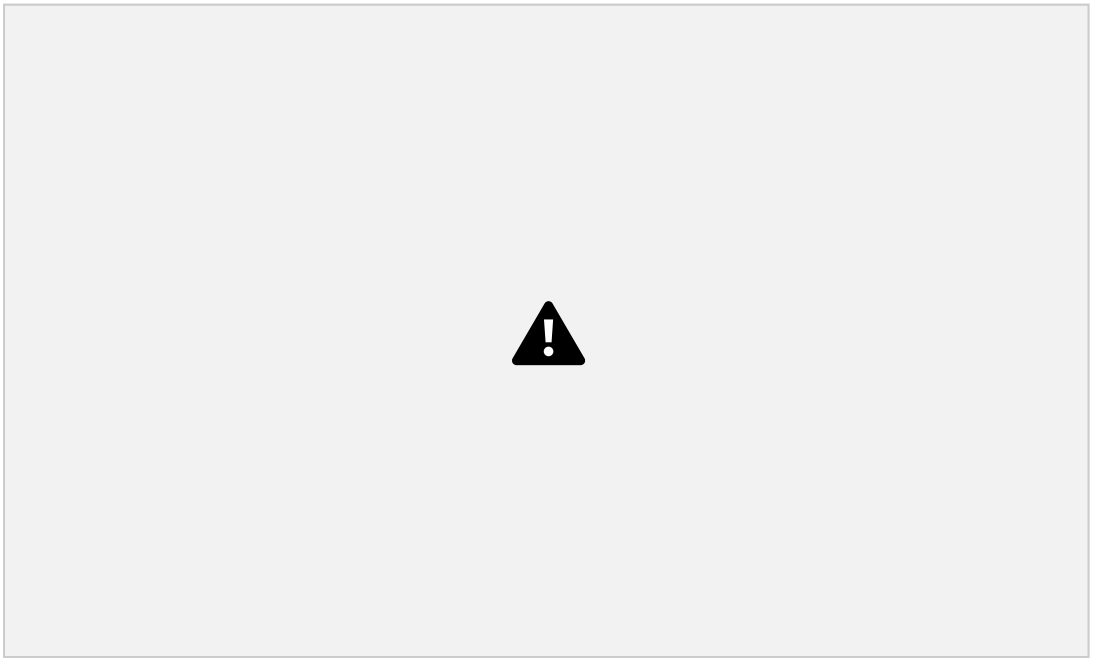


Creating Model

- Creating a app1 post model:
- In the app1/models.py file we define all objects called Models – this is a place in which we will define our blog post.
- app1/models.py
- class Post(models.Model):
- -------
- -------
- Create tables for models in your database.
- The last step here is to add our new model to our database
- First we have to make Django know that we have some changes in our model. (We have just created it!)
- Go to your console window and type
- python manage.py makemigrations
- Django prepared a migration file for us that we now have to apply to our database. Type
- python manage.py migrate

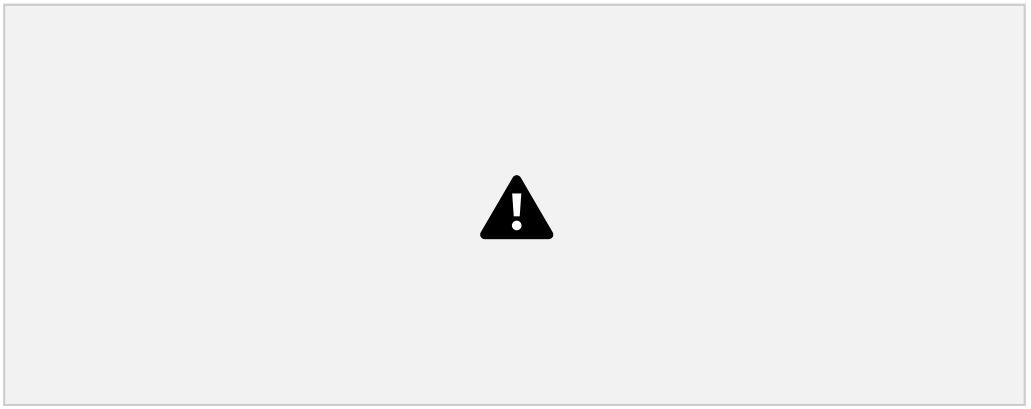
Design your model

Although you can use Django without a database, it comes with an object-relational mapper in which you describe your database layout in Python code.



Design your URLs

- A clean, elegant URL scheme is an important detail in a high-quality Web application. Django encourages beautiful URL design and doesn't put any cruft in URLs, like .php or .asp.
- To design URLs for an app, you create a Python module called a URLconf. A table of contents for your app, it contains a mapping between URL patterns and Python callback functions. URLconfs also serve to decouple URLs from Python code.
- Here's what a URLconf might look like for the Reporter/Article example above:

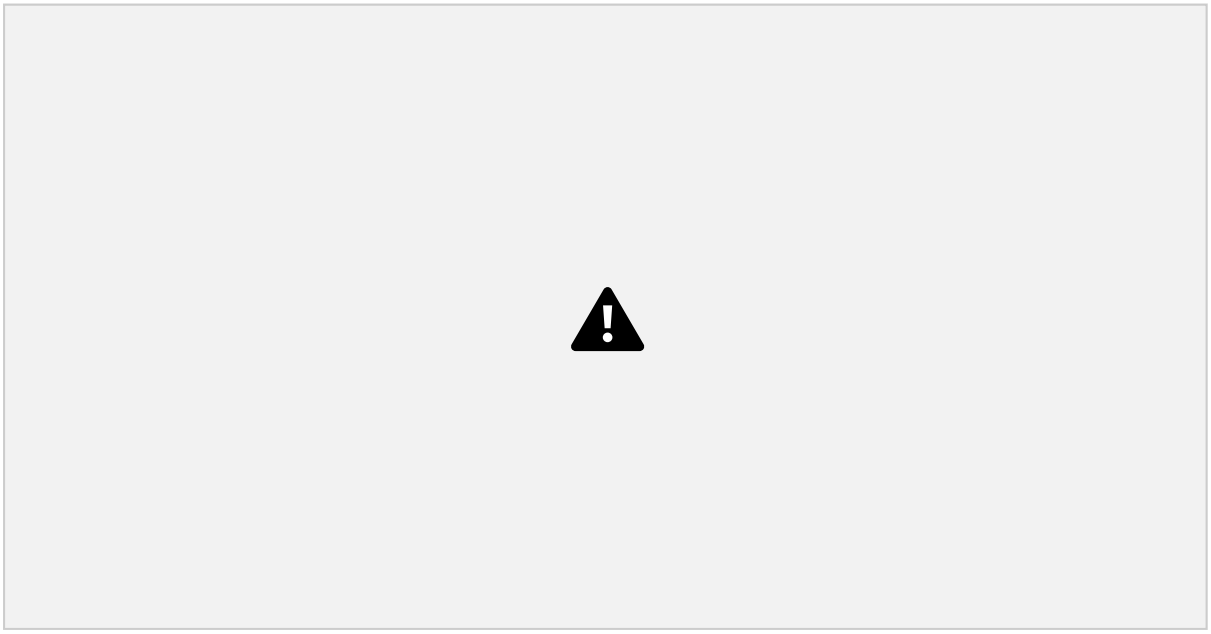


Design your templates[html pages]

• The code above loads the news/year_archive.html template.

• Django has a template search path, which allows you to minimize redundancy among templates. In your Django settings, you specify a list of directories to check for templates with DIRS. If a template doesn't exist in the first directory, it checks the second, and so on.

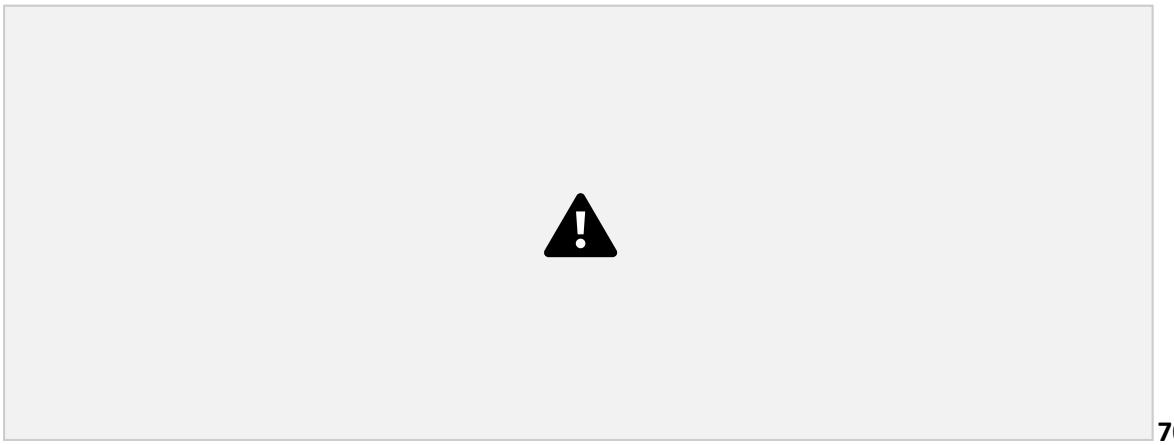
• Let's say the news/year_archive.html template was found. Here's what that might look like:



Write your views

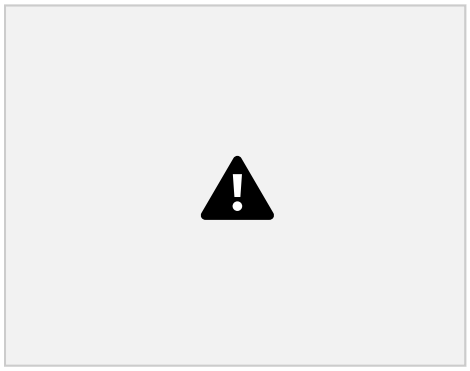
• Each view is responsible for doing one of two things: Returning an **HttpResponse** object containing the content for the requested page, or raising an exception such as **Http404**. The rest is up to you.

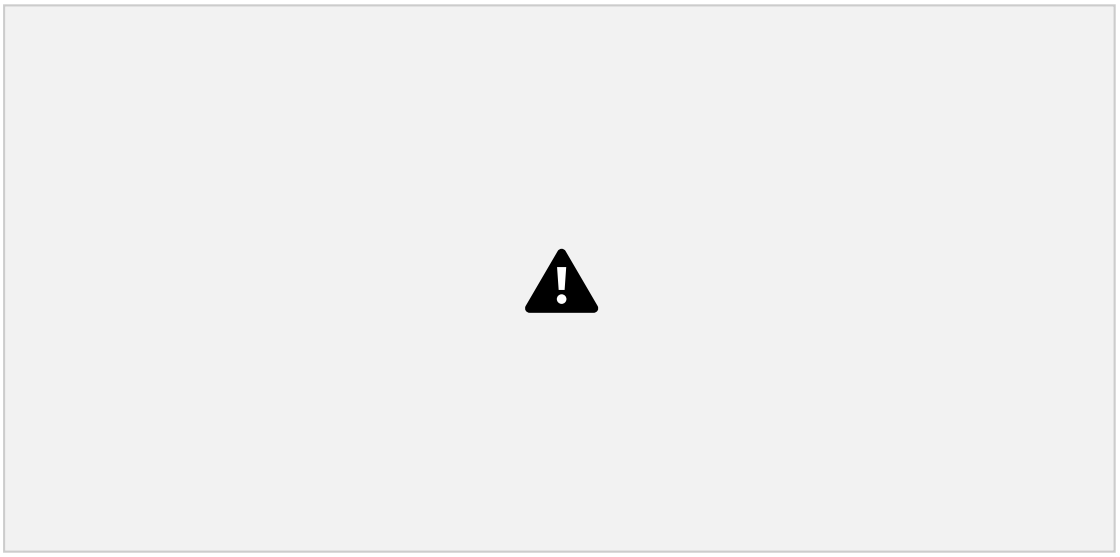
• Generally, a view retrieves data according to the parameters, loads a template and renders the template with the retrieved data. Here's an example view for **year_archive** from above:



Django Admin

- To add, edit and delete the posts we've just modeled, we will use Django admin.
- Let's open the app1/admin.py file and replace its contents with this: • app1/admin.py
- from django.contrib import admin
- from .models import Post
- admin.site.register(Post)
- To log in, you need to create a superuser - a user account that has control over everything on the site. Go back to the command line, type • python manage.py createsuperuser and press enter
- (myvenv) ~/$ python manage.py createsuperuser
- Username: admin
- Email address: admin@admin.com
- Password:
- Password (again):
- Superuser created successfully





AGOG TECHNOLOGIES

Make a new project by simple following steps :

1.md project_folder

2.cd project_folder

3.python -m venv myenv

4.myenv\Scripts\activate

5.pip install django==2.2

6.django-admin startproject project_name

. 7.python manage.py migrate

8.python manage.py createsuperuser

9.python manage.py startapp app_name

❖ Artificial Intelligence:

Artificial Intelligence which is ready to create a new revolution in the world by making intelligent machines. The Artificial Intelligence is now all around us. It is currently working with a variety of subfields, ranging from general to specific, such as self-driving cars, playing chess, proving theorems, playing music, Painting, etc.

Artificial Intelligence is composed of two words **Artificial** and **Intelligence**, where Artificial defines *"man-made,"* and intelligence defines *"thinking power",* hence AI means *"a man-made thinking power."*

So, we can define AI as:

"It is a branch of computer science by which we can create intelligent machines which can behave like a human, think like humans, and able to make decisions."

➢ Why Artificial Intelligence?

- With the help of AI, you can create such software or devices which can solve real world problems very easily and with accuracy such as health issues, marketing, traffic issues, etc.
- With the help of AI, you can create your personal virtual Assistant, such as Cortana, Google Assistant, Siri, etc.
- With the help of AI, you can build such Robots which can work in an environment where survival of humans can be at risk.
- AI opens a path for other new technologies, new devices, and new Opportunities.

➢ What Comprises to Artificial Intelligence?

- Artificial Intelligence is not just a part of computer science even it's so vast and requires lots of other factors which can contribute to it. To create the AI first we should know that how intelligence is composed, so the Intelligence is an intangible part of our brain which is a combination of **Reasoning, learning, problem-solving perception, language understanding, etc**.

➢ Advantages of Artificial Intelligence :

- **High Accuracy with less errors:** AI machines or systems are prone to less errors and high accuracy as it takes decisions as per pre-experience or information.
- **High-Speed:** AI systems can be of very high-speed and fast-decision making, because of that AI systems can beat a chess champion in the Chess game.

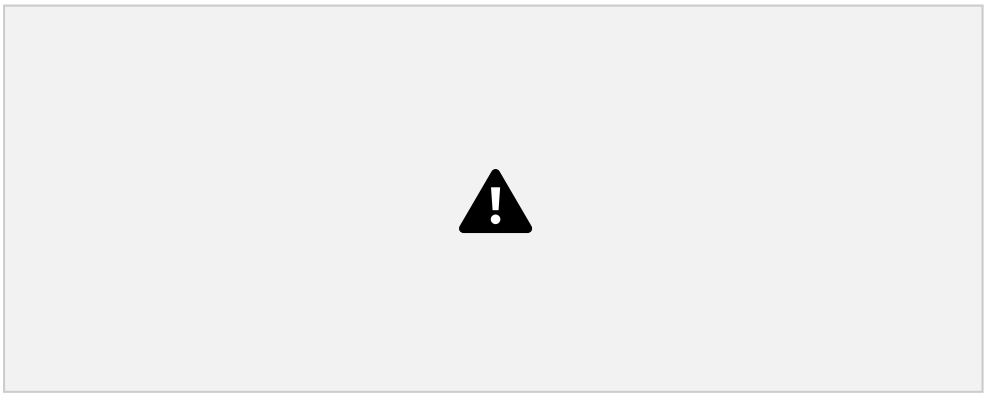
- **High reliability:** AI machines are highly reliable and can perform the same action multiple times with high accuracy.
- **Useful as a public utility:** AI can be very useful for public utilities such as a self-driving car which can make our journey safer and hassle-free, facial recognition for security purpose, Natural language processing to communicate with the human in human-language, etc.
- **Useful for risky areas:** AI machines can be helpful in situations such as defusing a bomb, exploring the ocean floor, where to employ a human can be risky.

- **Digital Assistant:** AI can be very useful to provide digital assistant to the users such as AI technology is currently used by various E-commerce websites to show the products as per customer requirement.

➢ **Application of AI :**

- AI in Astronomy
- AI in Healthcare
- AI in Gaming
- AI in Finance
- AI in Data Security
- AI in Social Media
- AI in Travel and Transport
- AI in Automotive Industry
- AI in Robotics
- AI in Entertainment
- AI in Agriculture
- AI in E-commerce
- AI in Education

➢ **Types of AI :**



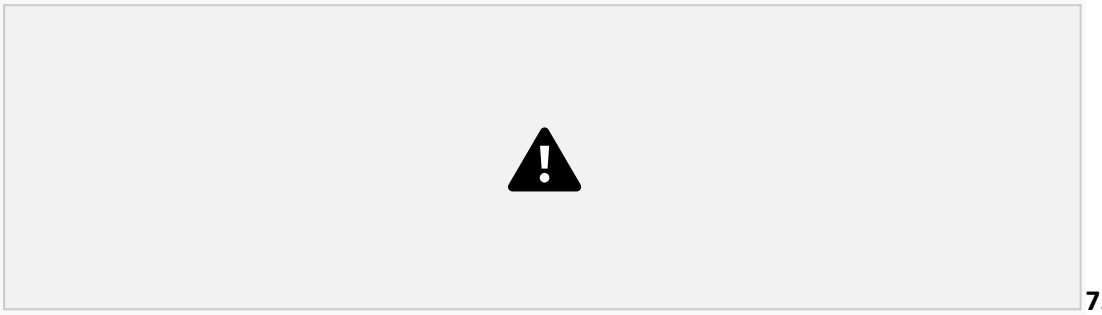
❖ Machine Learning:

Machine learning is a growing technology which enables computers to learn automatically from past data.

Machine learning uses various algorithms for **building mathematical models and making predictions using historical data or information**. Currently, it is being used for various tasks such as **image recognition**, **speech recognition**, **email filtering**, **Facebook auto-tagging**, **recommender system**, and many more.

Machine learning enables a machine to automatically learn from data, improve performance from experiences, and predict things without being explicitly programmed.

➢ **How does Machine Learning Works :**



➢ **Features of Machine Learning :**

- Machine learning uses data to detect various patterns in a given dataset.
- It can learn from past data and improve automatically.
- It is a data-driven technology.
- Machine learning is much similar to data mining as it also deals with the huge amount of the data.

➢ **Types of Machine Learning:**

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning
- Supervised learning:-
  - Supervised learning is a type of machine learning method in which we provide sample labeled data to the machine learning system in order to train it, and on that basis, it predicts the output.
  - The system creates a model using labeled data to understand the datasets and learn about each data, once the training and processing are done then we test the model by providing a sample data to check whether it is predicting the exact output or not.
  - The goal of supervised learning is to map input data with the output data. The supervised learning is based on supervision, and it is the same as when a student learns things in the supervision of the teacher. The example of supervised learning is **spam filtering**.
  - Supervised learning can be grouped further in two categories of algorithms:
  - **Classification**
  - **Regression**

- Unsupervised Learning:
  - Unsupervised learning is a learning method in which a machine learns without any supervision.

  - The training is provided to the machine with the set of data that has not been labeled, classified, or categorized, and the algorithm needs to act on that data without any supervision. The goal of unsupervised learning is to restructure the input data into new features or a group of objects with similar patterns.
    - In unsupervised learning, we don't have a predetermined result.

The machine tries to find useful insights from the huge amount of data. It can be further classifieds into two categories of algorithms:

- **Clustering**
- **Association**

• Reinforcement Learning:

- Reinforcement learning is a feedback-based learning method, in which a learning agent gets a reward for each right action and gets a penalty for each wrong action. The agent learns automatically with these feedbacks and improves its performance.
- In reinforcement learning, the agent interacts with the environment and explores it. The goal of an agent is to get the most reward points, and hence, it improves its performance.
- The robotic dog, which automatically learns the movement of his arms, is an example of Reinforcement learning.