LISP

* Lisp is a programming language that has an overall style that is organized around expressions and functions. Every Lisp procedure is a function, and when called, it returns a data object as its value. It is also commonly referred to as “functions” even though they may have side effects.
* Lisp is the second-oldest high-level programming language in the world which is invented by John McCarthy in the year 1958 at the Massachusetts Institute of Technology.
* The name *LISP* derives from "LISt Processor" [Linked lists](https://en.wikipedia.org/wiki/Linked_list) are one of Lisp's major [data structures](https://en.wikipedia.org/wiki/Data_structure), and Lisp [source code](https://en.wikipedia.org/wiki/Source_code) is made of lists. Thus, Lisp programs can manipulate source code as a data structure, giving rise to the [macro](https://en.wikipedia.org/wiki/Macro_(computer_science)) systems that allow programmers to create new syntax or new [domain-specific languages](https://en.wikipedia.org/wiki/Domain-specific_language) embedded in Lisp.
* **Features of  LISP Programming Language:**

1. It is a machine-independent language
2. It uses iterative design methodology and is easily extensible
3. It allows us to create and update the programs and applications dynamically.
4. It provides high-level debugging.
5. It supports object-oriented programming.
6. It supports all kinds of data types like objects, structures, lists, vectors, adjustable arrays, set, trees,hash-tables, and symbols.
7. It is an expression-based language
8. It can support different decision-making statements like if, when,case, and cond
9. It will also support different iterating statements like do, loop,loopfor, dotimes and dolist.
10. It will support input and output functions
11. By using lisp we can also create our own functions

* These are the features of LISP Programming.
* **Hello World program in LISP:**
* **we can start writing  a string by using the write-line method**
* **Syntax:**
* **(write-line string)**
* **Example:**
* **;this is a comment**
* **(write-line "Hello Geeks")**
* **Output:**
* **Hello Geeks**
* **A variable can not contain double and single quotes, backslash, comma, colon, semicolon, and vertical bar.**
* **Example:**
* **Acceptable – hello,saisravan, etc**
* **Not Acceptable – hell””)0,sat//\*& sra//>vab{,,,,,,,,etc**
* **A variable can not start with a digit but. it can contain any number of digits**
* **Example:**
* **Acceptable – hello88Geeks, r56lisp, ,,,,,etc**
* **Not Acceptable – 40geeks,4klll,….etc**
* **Example Code: For acceptable names**
* **;acceptable naming conventions**
* **(write-line "hello")**
* **;acceptable naming conventions**
* **(write-line "hello99")**

* **;acceptable naming conventions**
* **(write-line "hello geeks")**
* **;acceptable naming conventions**
* **(write-line "hello\_Geek")**

* **;acceptable naming conventions**
* **(write-line "hello123")**
* **Output:**
* **hello**
* **hello99**
* **hello geeks**
* **hello\_Geek**
* **hello123**

**Loops in LISP**

Loops allow executing a set of instructions repeatedly while some condition is true. LISP provides the following types of loops:

**1. dotimes loop:**

The dotimes loop allows executing instructions for a fixed number of times.  
**Syntax:**

(**dotimes**( variableName numberOfIterations ) (

expressions

))

Where,

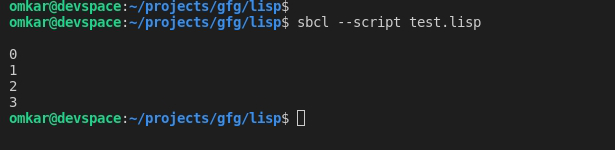
* **variableName =>**Name of the variable, this can be used to get current iteration number
* **numberOfIterations =>**Total number of iterations
* **expressions** **=>** Statements to be executed in each iteration

**Example:**

* Lisp

|  |
| --- |
| (**dotimes** (a 4)    (print a))  (write**-**line "") |

**Output:**

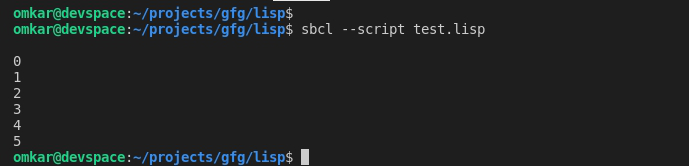


You can also return from the loop if a certain condition is met

* Lisp

|  |
| --- |
| (**dotimes** (i 7)     (**if** (> i 5)         (return)         (print i))      )  (write**-**line "") |

Here, the loop is set to iterate 7 times, however, when the value becomes more than 5, the loop stops because the condition is met.   
**Output:**



**2.  loop:**

The loop construct allows executing some statement(s) repeatedly until it finds a return statement.

**Syntax:**

( **loop**

(expressions)

( **when** (condition) (**return** returnValue))

)

Where,

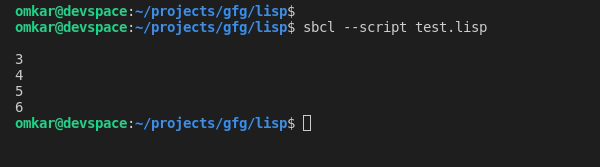
* **expressions** **=>** Statements to be executed in each iteration
* **condition** **=>** Condition that specifies when to exit loop
* **returnValue =>**Value to be returned

**Example:**

* Lisp

|  |
| --- |
| (**defvar** n 3)  (loop      (print n)      (setq n (**+** n 1))      (when (> n 6) (return n))  )  (write**-**line "") |

Here, n is printed and incremented until it becomes greater than 6.  
**Output:**



**3. loop for:**

The loop for construct is similar to the for-loops in popular languages like java, c++, etc. It can also be used to traverse lists and other data structures.

**Syntax:**

( **loop for** variableName **from** startValue **to** endValue **by** changeValue **do**

(expressions)

)

Where,

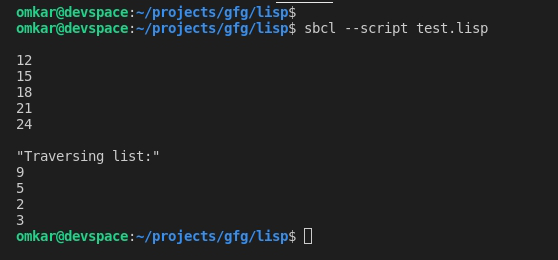
* **variableName =>** Name of the variable, this can be used to get the value of variable in current iteration.
* **startValue =>**Initial value of variable
* **endValue =>** End value of variable (after last iteration)
* **changeValue =>** Specifies by how much to increment or decrement value after every iteration
* **expressions =>** Statements to be executed in each iteration

**Example:**

* Lisp

|  |
| --- |
| ; initialize x**=**12, execute instructions,  ; increment it by 3 and perform iteration until it is less than 25  (loop for x from 12 to 25 by 3 do      (print x)  )  (write**-**line "") |

**Output:**



**4. do:**

The do construct allows a structured form of iteration.  
**Syntax:**

(**do** ((variable\_1 value\_1 updated-value\_1)

(variable\_2 value\_2 updated-value\_2)

(variable\_3 value\_3 updated-value\_3)

...)

(test return\_value)

(s-expressions)

)

Where,

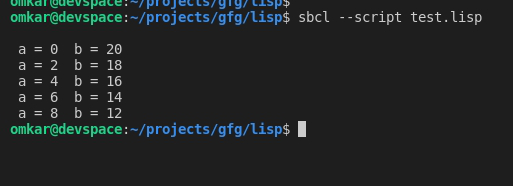
* **variable\_1 =>** First variable
* **value\_1 =>**Initial Value for first variable
* **updated-value\_1 =>** Specifies how to update variable\_1 after each iteration.
* **variable\_2 =>** Second variable
* **value\_2 =>** Initial Value for second variable
* **updated-value\_2 =>** Specifies how to update variable\_2 after each iteration. … similar till **variable\_n**
* **test =>** Condition that specifies when to stop the loop
* **return value =>** Value that is returned at the end of iteration
* **expressions =>** Other statements to be executed in every iteration

The initial values of each variable are evaluated and bound to the respective variable. The updated value in every clause is with respect to an optional update statement which indicates how to update variables in each iteration.  
The test is performed on each iteration and if the test expression results in a non-null or true value, the returned value is evaluated and returned.  
If the last optional s-expression(s) are specified, they’re executed on every iteration while test expression results are true.  
**Example:**

* Lisp

|  |
| --- |
| (do ((a 0 (**+** 2 a))     (b 20 ( **-** b 2)))     ((**=** a b)(**-** a b))     (format **t** "~% a = ~d  b = ~d" a b)  )  (write**-**line "") |

**Output:**



**5. dolist:**

The dolist construct provides an easy way to traverse lists.

**Syntax:**

(**dolist** (listItemVariable list)

(expressions)

)

Where,

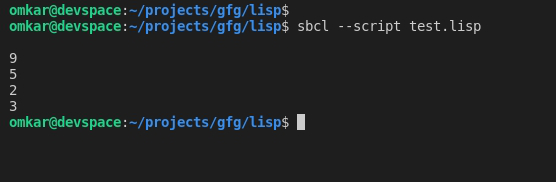
* **listItemVariable =>**Holds value of each list item during every iteration
* **list =>** List to traverse
* **expressions =>** Statements to be executed in every iteration

**Example:**

* Lisp

|  |
| --- |
| (**defvar** nums `(9 5 2 3))  (dolist (item nums)    (print item)  )  (write**-**line "") |

**Output:**



**Logical Operators in LISP**

Common LISP supports 3 types of logical operators on Boolean Values. The arguments of these operators are evaluated conditionally, therefore they are also part of the LISP control Structure.

The common LISP operators are listed below in the table:

| **Operator** | **Syntax** | **Description** |
| --- | --- | --- |
| And | and number1 number2 | This operator takes two numbers which are evaluated left to right. If all numbers evaluate to non-nil, then the value of the last number is returned. Otherwise, nil is returned. |
| Or | or number1 number2 | This operator takes two numbers which are evaluated left to right. If any one number evaluates to non-nil, then the value of the last number is returned. Otherwise, nil is returned. |
| Not | not number | This operator takes one number and returns T(true) if the argument evaluates to NIL |

**Example:**LISP Program that demonstrates Logical operators on numbers

* Lisp

|  |
| --- |
| ;set value 1 to 50  ; set value 2 to 50  (setq val1 50)  (setq val2 50)    ;and operator  (print (and val1 val2))    ;or operator  (print (or val1 val2))    ;**not** operator with value1  (print (**not** val1))    ;**not** operator with value2  (print (**not** val2)) |

**Output:**

1

1

NIL

NIL

**Example 2:**LISP Program to demonstrate Logical operators on Boolean values

* Lisp

|  |
| --- |
| ;set value 1 to T  ; set value 2 to NIL  (setq val1 T)  (setq val2 NIL)    ;and operator  (print (and val1 val2))    ;or operator  (print (or val1 val2))    ;**not** operator with value1  (print (**not** val1))    ;**not** operator with value2  (print (**not** val2)) |

**Output:**

NIL

T

NIL

T

**Arithmetic Operators in LISP**

**Arithmetic operators** are used to perform mathematical operations like addition, subtraction, multiplication, and division.

There are 7 arithmetic operators in LISP that are listed in the below table:

| **Operator** | **Syntax** | **Description** |
| --- | --- | --- |
| Addition Operator(+) | + num1 num2 | Add the two numbers |
| Subtraction Operator(-) | – num1 num2 | Subtract the second number from the first number |
| Multiplication(\*) | \* num1 num2 | Multiply two numbers |
| Division(/) | /  num1 num2 | Divide the two numbers |
| Modulus(mod) | mod num1 num2 | Get the remainder of two numbers |
| Increment(incf) | incf  num value | Increment number by given value |
| Decrement(decf) | decf num value | Decrement number by given value |

**Example 1:**LISP Program that demonstrates arithmetic operators

* Lisp

|  |
| --- |
| ;set value 1 to 300  ; set value 2 to 600  (setq val1 300)  (setq val2 600)    ;addition operation  (print (**+** val1 val2))    ;subtraction operation  (print (**-** val1 val2))    ;multiplication operation  (print (**\*** val1 val2))    ;division operation  (print (**/** val1 val2))    ;modulus operation  (print (MOD val1 val2))    (print (incf val1 val2))    (print (decf val1 val2)) |

**Output:**

900

-300

180000

1/2

300

900

300

**Strings in LISP**

A string is a set of characters. String  are enclosed in double-quotes.

**Example:**

"hello geek","java","python" etc

**Example:**LISP program to display strings

* Lisp

|  |
| --- |
| ;edisplay hello geek  (write**-**line "Hello Geek")    ;display  (write**-**line "Welcome to java") |

**Output:**

Hello Geek

Welcome to java

**String Comparison Functions:**

Used to compare two strings. Further they can be divided into two categories. They are

**Case Sensitive Functions:**

These functions can be represented by mathematical symbols.

| **Symbol** | **Name** | **Syntax** | **Description** |
| --- | --- | --- | --- |
| = | equal to | string= | This operator checks if the values of the operands are all equal or not, if yes then the condition becomes true(T). otherwise it returns NIL |
| /= | not equal to | string/= | This operator checks if the values of the operands are all different or not, if values are not equal then the condition becomes true(T). otherwise, it returns NIL |
| > | greater than | string> | This operator checks if the values of the operands are monotonically increasing. |
| < | less than | string< | This operator checks if the values of the operands are monotonically decreasing. |
| >= | greater than or equal to | string>= | This operator checks if the value of any left operand is less than or equal to the value of its right operand, if yes then the condition becomes true. |
| <= | less than or equal to | string<= | This operator checks if the value of any left operand is greater than or equal to the value of the next right operand, if yes then the condition becomes true. |

**Example:** LISP program that demonstrates string case sensitive functions

* Lisp

|  |
| --- |
| ; case**-**sensitive comparison **-** **equal** to  (write (string**=** "Hello Geeks" "Hello Geeks"))      ; case**-**sensitive comparison **-** **equal** to  (write (string**=** "Hello Geeks" "HelloGeeks"))      ; case**-**sensitive comparison **-** **not** **equal** to  (write (string**/=** "Hello Geeks" "Hello Geeks"))      ; case**-**sensitive comparison **-** **not** **equal** to  (write (string**/=** "Hello Geeks" "HelloGeeks"))      ; case**-**sensitive comparison **-**  greater than  (write (string> "Hello Geeks" "Python"))    ; case**-**sensitive comparison **-**  less than  (write (string< "Hello Geeks" "java"))    ; case**-**sensitive comparison **-**  greater than or **equal** to  (write (string>**=** "Hello Geeks" "Python"))      ; case**-**sensitive comparison **-**  less than or **equal** to  (write (string<**=** "Hello Geeks" "java")) |

**Output:**

T

NIL

NIL

5

NIL

0

NIL

0

**Case INSENSITIVE FUNCTIONS**

These functions can be represented by expressions.

| **Name** | **Syntax** | **Description** |
| --- | --- | --- |
| Equal | string-equal | This operator checks if the values of the operands are all equal or not, if yes then the condition becomes true(T). Otherwise, it returns NIL |
| not equal | string-not-equal | This operator checks if the values of the operands are all different or not, if values are not equal then the condition becomes true(T). Else, it returns NIL |
| greater than | string-greaterp | This operator checks if the values of the operands are monotonically increasing. |
| less than | string-lessp | This operator checks if the values of the operands are monotonically decreasing. |
| greater than or equal to | string-not-lessp | This operator checks if the value of any left operand is less than or equal to the value of its right operand, if yes then the condition becomes true. |
| less than or equal to | string-not-greaterp | This operator checks if the value of any left operand is greater than or equal to the value of the next right operand, if yes then the condition becomes true. |

**Example:** Lisp program that demonstrates case insensitive functions

* Lisp

|  |
| --- |
| ; case**-**sensitive comparison **-** **equal** to  (write (string**-equal** "Hello Geeks" "Hello Geeks"))    ;new line  (terpri)    ; case**-**sensitive comparison **-** **equal** to  (write (string**-equal** "Hello Geeks" "HelloGeeks"))    ;new line  (terpri)    ; case**-**sensitive comparison **-** **not** **equal** to  (write (string**-not-equal** "Hello Geeks" "Hello Geeks"))    ;new line  (terpri)    ; case**-**sensitive comparison **-** **not** **equal** to  (write (string**-not-equal** "Hello Geeks" "HelloGeeks"))    ;new line  (terpri)    ; case**-**sensitive comparison **-**  greater than  (write (string**-**greaterp "Hello Geeks" "Python"))    ;new line  (terpri)    ; case**-**sensitive comparison **-**  less than  (write (string**-**lessp "Hello Geeks" "java"))    ;new line  (terpri)    ; case**-**sensitive comparison **-**  greater than or **equal** to  (write (string**-not-**lessp "Hello Geeks" "Python"))    ;new line  (terpri)    ; case**-**sensitive comparison **-**  less than or **equal** to  (write (string**-not-**greaterp "Hello Geeks" "java"))    ;new line  (terpri) |

**Output:**

T

NIL

NIL

5

NIL

0

NIL

0

**Functions in LISP**

A function is a set of statements that takes some input, performs some tasks, and produces the result. Through functions, we can split up a huge task into many smaller functions. They also help in avoiding the repetition of code as we can call the same function for different inputs.

**Defining Functions in LISP:**

Functions in LISP are defined using the **DEFUN** macro. The basic syntax looks like this :

(defun function-name (parameters)

"Documentation string"

body-of-function

)

**Example:**

1. Let’s create a function named ***hello-world*** that doesn’t take any parameters and returns a hello world string.

* Lisp

|  |
| --- |
| (defun hello**-**world ()    (format **t** "Hello, World!")  )  (hello**-**world) |

**Output:**

Hello, World!

2. Function to add two integers

Here we have created a function named ***add-two-number*** which takes two arguments **n1** and **n2**. A Documentation string is used to describe the work done by this function and whatever the value is return by calling the **+** eventually becomes the return value of add-two-number.

* Lisp

|  |
| --- |
| (defun add**-**two**-**number (n1 n2)    "Adds two numbers"    (**+** n1 n2)  )  (write(add**-**two**-**number 10 20)) |

**Output:**

30

**Prolog | An Introduction**

**Introduction :**

Prolog is a [logic programming](https://en.wikipedia.org/wiki/Logic_programming) language. It has important role in artificial intelligence. Unlike many other programming languages, Prolog is intended primarily as a declarative programming language. In prolog, logic is expressed as relations (called as Facts and Rules). Core heart of prolog lies at the **logic** being applied. Formulation or Computation is carried out by running a query over these relations.

**Installation in Linux :**

Open a terminal **(Ctrl+Alt+T)** and type:

sudo apt-get install swi-prolog

**Syntax and Basic Fields :**

In prolog, We declare some facts. These facts constitute the Knowledge Base of the system. We can query against the Knowledge Base. We get output as affirmative if our query is already in the knowledge Base or it is implied by Knowledge Base, otherwise we get output as negative. So, Knowledge Base can be considered similar to database, against which we can query. Prolog facts are expressed in definite pattern. Facts contain entities and their relation. Entities are written within the parenthesis separated by comma (, ). Their relation is expressed at the start and outside the parenthesis. Every fact/rule ends with a dot (.). So, a typical prolog fact goes as follows :

Format : relation(entity1, entity2, ....k'th entity).

Example :

friends(raju, mahesh).

singer(sonu).

odd\_number(5).

Explanation :

These facts can be interpreted as :

raju and mahesh are friends.

sonu is a singer.

5 is an odd number.

**Key Features :**  
**1. Unification :** The basic idea is, can the given terms be made to represent the same structure.  
**2. Backtracking :** When a task fails, prolog traces backwards and tries to satisfy previous task.  
**3. Recursion :** Recursion is the basis for any search in program.

**Running queries :**  
A typical prolog query can be asked as :

Query 1 : ?- singer(sonu).

Output : Yes.

Explanation : As our knowledge base contains   
the above fact, so output was 'Yes', otherwise   
it would have been 'No'.

Query 2 : ?- odd\_number(7).

Output : No.

Explanation : As our knowledge base does not   
contain the above fact, so output was 'No'.

**Advantages :**  
**1.**Easy to build database. Doesn’t need a lot of programming effort.  
**2.**Pattern matching is easy. Search is recursion based.  
**3.**It has built in list handling. Makes it easier to play with any algorithm involving lists.

**Disadvantages :**  
**1.** LISP (another logic programming language) dominates over prolog with respect to I/O features.  
**2.** Sometimes input and output is not easy.

**Applications :**

Prolog is highly used in artificial intelligence(AI). Prolog is also used for pattern matching over natural language parse trees.

Python Programming Language

Python is a high-level, general-purpose and a very popular programming language. Python programming language (latest Python 3) is being used in web development, Machine Learning applications, along with all cutting edge technology in Software Industry. Python Programming Language is very well suited for Beginners, also for experienced programmers with other programming languages like C++ and Java.

This specially designed Python tutorial will help you learn Python Programming Language in most efficient way, with the topics from basics to advanced (like Web-scraping, Django, Deep-Learning, etc.) with examples.

Below are some facts about Python Programming Language:

1. Python is currently the most widely used multi-purpose, high-level programming language.
2. Python allows programming in Object-Oriented and Procedural paradigms.
3. Python programs generally are smaller than other programming languages like Java. Programmers have to type relatively less and indentation requirement of the language, makes them readable all the time.
4. Python language is being used by almost all tech-giant companies like – Google, Amazon, Facebook, Instagram, Dropbox, Uber… etc.
5. The biggest strength of Python is huge collection of standard library which can be used for the following:
   * [Machine Learning](https://www.geeksforgeeks.org/machine-learning/)
   * GUI Applications (like [Kivy](https://www.geeksforgeeks.org/kivy-tutorial/" \t "_blank), Tkinter, PyQt etc. )
   * Web frameworks like [Django](https://www.geeksforgeeks.org/django-tutorial/) (used by YouTube, Instagram, Dropbox)
   * Image processing (like [OpenCV](https://www.geeksforgeeks.org/opencv-python-tutorial/), Pillow)
   * Web scraping (like Scrapy, BeautifulSoup, Selenium)
   * Test frameworks
   * Multimedia
   * Scientific computing
   * Text processing and many more..

**Python Language Introduction**

[Python](https://www.geeksforgeeks.org/python-programming-language/) is a widely used general-purpose, high level programming language. It was created by Guido van Rossum in 1991 and further developed by the Python Software Foundation. It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code.

Python is a programming language that lets you work quickly and integrate systems more efficiently.

 Attention geek! Strengthen your foundations with the [**Python Programming Foundation**](https://practice.geeksforgeeks.org/courses/Python-Foundation?utm_source=geeksforgeeks&utm_medium=article&utm_campaign=GFG_Article_Bottom_Python_Foundation) Course and learn the basics.

To begin with, your interview preparations Enhance your Data Structures concepts with the [**Python DS**](https://practice.geeksforgeeks.org/courses/Data-Structures-With-Python?utm_source=geeksforgeeks&utm_medium=article&utm_campaign=GFG_Article_Bottom_Python_DS) Course. And to begin with your Machine Learning Journey, join the [**Machine Learning - Basic Level Course**](https://practice.geeksforgeeks.org/courses/machine-learning?utm_source=geeksforgeeks&utm_medium=article&utm_campaign=GFG_Article_Bottom_Python_ML)

There are two major Python versions: **Python 2 and Python 3**. Both are quite different.

**Beginning with Python programming:**

**1) Finding an Interpreter:**

Before we start Python programming, we need to have an interpreter to interpret and run our programs. There are certain online interpreters like [**https://ide.geeksforgeeks.org/**](https://ide.geeksforgeeks.org/), http://ideone.com/ or http://codepad.org/ that can be used to run Python programs without installing an interpreter.

***Windows*:**There are many interpreters available freely to run Python scripts like IDLE (Integrated Development Environment) that comes bundled with the Python software downloaded from [**http://python.org/**](http://python.org/).

***Linux*:**Python comes preinstalled with popular Linux distros such as Ubuntu and Fedora. To check which version of Python you’re running, type “python” in the terminal emulator. The interpreter should start and print the version number.

***macOS*:**Generally, Python 2.7 comes bundled with macOS. You’ll have to manually install Python 3 from [**http://python.org/**](http://python.org/).

**2) Writing our first program:**

Just type in the following code after you start the interpreter.

|  |
| --- |
| # Script Begins        print("GeeksQuiz")        # Scripts Ends |

Output:

GeeksQuiz

**Python Language advantages and applications**

Python is a high-level, interpreted, and general-purpose dynamic programming language that focuses on code readability. It has fewer steps when compared to Java and C. It was founded in 1991 by developer Guido Van Rossum. Python ranks among the most popular and fastest-growing languages in the world. Python is a powerful, flexible, and easy-to-use language. In addition, the community is very active there. It is used in many organizations as it supports multiple programming paradigms. It also performs automatic memory management.

**Advantages :**

1. Presence of third-party modules
2. Extensive support libraries(NumPy for numerical calculations, Pandas for data analytics etc)
3. Open source and community development
4. Versatile, Easy to read, learn and write
5. User-friendly data structures
6. High-level language
7. Dynamically typed language(No need to mention data type based on the value assigned, it takes data type)
8. Object-oriented language
9. Portable and Interactive
10. Ideal for prototypes – provide more functionality with less coding
11. Highly Efficient(Python’s clean object-oriented design provides enhanced process control, and the language is equipped with excellent text processing and integration capabilities, as well as its own unit testing framework, which makes it more efficient.)
12. (IoT)Internet of Things Opportunities
13. Interpreted Language
14. Portable across Operating systems

**Applications :**

1. GUI based desktop applications
2. Graphic design, image processing applications, Games, and Scientific/ computational Applications
3. Web frameworks and applications
4. Enterprise and Business applications
5. Operating Systems
6. Education
7. Database Access
8. Language Development
9. Prototyping
10. Software Development

**Organizations using Python :**

1. Google(Components of Google spider and Search Engine)
2. Yahoo(Maps)
3. YouTube
4. Mozilla
5. Dropbox
6. Microsoft
7. Cisco
8. Spotify
9. Quora

**Python 3 basics**

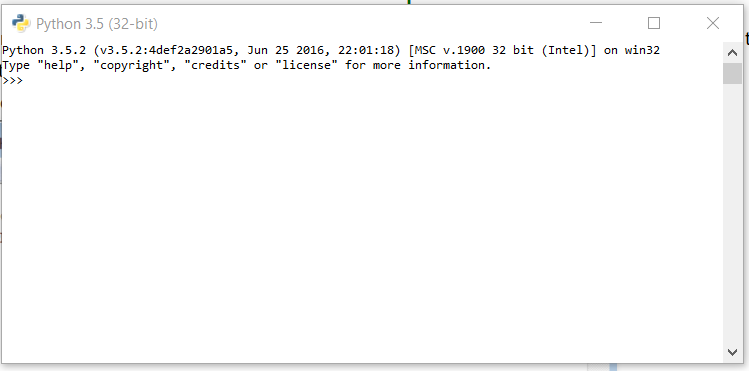
[Python](https://www.geeksforgeeks.org/python-tutorial-learn-python-3-with-examples/) was developed by Guido van Rossum in the early 1990s and its latest version is 3.7.1, we can simply call it as Python3. Python 3.0 was released in 2008. and is interpreted language i.e it’s not compiled and the interpreter will check the code line by line. This article can used to learn very basics of [Python programming language](https://www.geeksforgeeks.org/python-programming-language/).

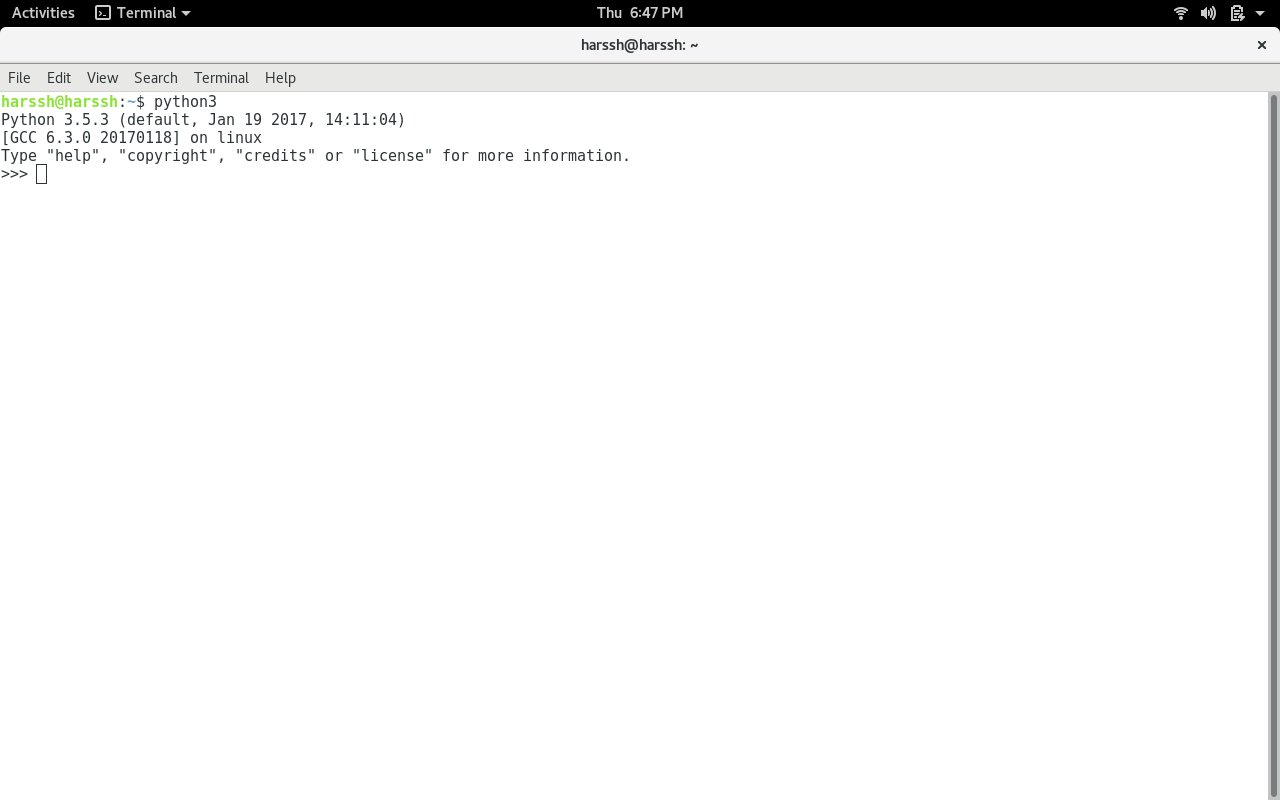
|  |
| --- |
| # Python code for "Hello World"  # nothing else to type...see how simple is the syntax.    print("Hello World") |

**Note:**Please note that Python for its scope doesn’t depend on the braces ( { } ), instead it uses indentation for its scope.  
Now moving on further **Lets start our basics of Python**. I will be covering the basics in some small sections. Just go through them and trust me you’ll learn the basics of Python very easily.

[Introduction and Setup](https://www.geeksforgeeks.org/python-language-introduction/)

1. If you are on **Windows OS** download Python by [Clicking here](https://www.python.org/downloads/windows/) and now install from the setup and in the start menu type IDLE.IDLE, you can think it as an Python’s IDE to run the Python Scripts.

It will look somehow this :  


1. If you are on **Linux/Unix-like** just open the terminal and on 99% linux OS Python comes preinstalled with the OS.Just type ‘python3’ in terminal and you are ready to go.  
   It will look like this :  
   

*The ” >>> ” represents the python shell and its ready to take python commands and code.*

[Variables and Data Structures](https://www.geeksforgeeks.org/python-set-2-variables-expressions-conditions-and-functions/)

In other programming languages like C, C++, and Java, you will need to declare the type of variables but in Python you don’t need to do that. Just type in the variable and when values will be given to it, then it will automatically know whether the value given would be an int, float, or char or even a String.

|  |
| --- |
| # Python program to declare variables  myNumber **=** 3  **print**(myNumber)    myNumber2 **=** 4.5  print(myNumber2)    myNumber **=**"helloworld"  print(myNumber) |

**Output:**

3

4.5

helloworld

See, how simple is it, just create a variable and assign it any value you want and then use the print function to print it. Python have 4 types of built in Data Structures namely [List](https://www.geeksforgeeks.org/python-set-3-strings-lists-tuples-iterations/), [Dictionary](https://www.geeksforgeeks.org/python-set-4-dictionary-keywords-python/), [Tuple](https://www.geeksforgeeks.org/python-set-3-strings-lists-tuples-iterations/) and Set.

**List** is the most basic Data Structure in python. List is a mutable data structure i.e items can be added to list later after the list creation. It’s like you are going to shop at the local market and made a list of some items and later on you can add more and more items to the list.  
append() function is used to add data to the list.

|  |
| --- |
| # Python program to illustrate a list    # creates a empty list  nums **=** []    # appending data in list  nums.append(21)  nums.append(40.5)  nums.append("String")    print(nums) |

**Output:**

[21, 40.5, String]

**Comments:**

# is used for single line comment in Python

""" this is a comment """ is used for multi line comments

**Input and Output**

In this section, we will learn how to take input from the user and hence manipulate it or simply display it. input() function is used to take input from the user.

|  |
| --- |
| # Python program to illustrate  # getting input from user  name **=** input("Enter your name: ")    # user entered the name 'harssh'  print("hello", name) |

**Output:**

hello harssh

|  |
| --- |
| # Python3 program to get input from user    # accepting integer from the user  # the return type of input() function is string ,  # so we need to convert the input to integer  num1 **=** int(input("Enter num1: "))  num2 **=** int(input("Enter num2: "))    num3 **=** num1 **\*** num2  print("Product is: ", num3) |

**Output:**

Enter num1: 8 Enter num2: 6 ('Product is: ', 48)

**Selection**

Selection in Python is made using the two keywords ‘if’ and ‘elif’ and else (elseif)

|  |
| --- |
| # Python program to illustrate  # selection statement    num1 **=** 34  **if**(num1>12):  **print**("Num1 is good")  **elif**(num1>35):      print("Num2 is not gooooo....")  **else**:  **print**("Num2 is great") |

**Output:**

Num1 is good

**Functions**

You can think of functions like a bunch of code that is intended to do a particular task in the whole Python script. Python used the keyword ‘def’ to define a function.  
**Syntax:**

def function-name(arguments):

#function body

|  |
| --- |
| # Python program to illustrate  # functions  **def** hello():      print("hello")      print("hello again")  hello()    # calling function  hello() |

**Output:**

hello

hello again

hello

hello again

Now as we know any program starts from a ‘main’ function…lets create a main function like in many other programming languages.

|  |
| --- |
| # Python program to illustrate  # function with main  **def** getInteger():      result **=** int(input("Enter integer: "))  **return** result    **def** Main():  **print**("Started")        # calling the getInteger function and      # storing its returned value in the output variable      output **=** getInteger()      print(output)    # now we are required to tell Python  # for 'Main' function existence  **if** \_\_name\_\_**==**"\_\_main\_\_":      Main() |

**Output:**

Started

Enter integer: 5

[Iteration (Looping)](https://www.geeksforgeeks.org/loops-and-loop-control-statements-continue-break-and-pass-in-python/)

As the name suggests it calls repeating things again and again. We will use the most popular ‘for’ loop here.

|  |
| --- |
| # Python program to illustrate  # a simple for loop    **for** step **in** range(5):      print(step) |

**Output:**

0

1

2

3

4

[Modules](https://www.geeksforgeeks.org/python/#Modules%20in%20Python)

Python has a very rich module library that has several functions to do many tasks. You can read more about Python’s standard library . ‘import’ keyword is used to import a particular module into your python code. For instance consider the following program.

|  |
| --- |
| # Python program to illustrate  # math module  **import** math    **def** Main():      num **=** **-**85        # fabs is used to get the absolute      # value of a decimal      num **=** math.fabs(num)  **print**(num)      **if** \_\_name\_\_**==**"\_\_main\_\_":      Main() |

**Output:**

85.0

Python modules

Numpy

Pandas

JSON

CSV

MySql

Open Cv

MongoDB

Tkinter

Kivy