**INTRODUCTION**

* Python 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.

**WHY PYTHON?**

* Python works on different platforms (windows, mac, linux, raspberry pi, etc).
* Python has a simple syntax similar to the english language.
* Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
* Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
* Python can be treated in a procedural way, an object-orientated way or a functional way.

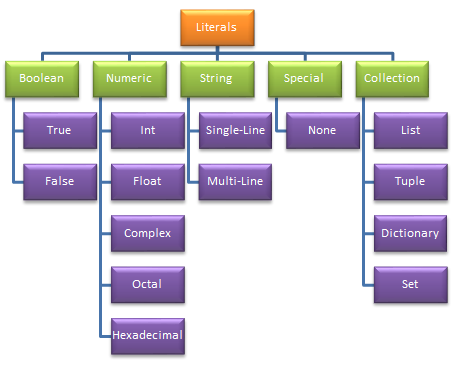
**Features of python:**

* Python works on different platforms (windows, mac, linux, raspberry pi, etc).
* Python has a simple syntax similar to the english language.
* Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
* Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
* Python can be treated in a procedural way, an object-orientated way or a functional way.

* **INTERPRETER**
  + Interpreter used for conversion and execution of the program.
  + Interpreter can run interactively and can be extended with additional high performance modules
* **LIBRARY SUPPORT**
  + Provides large collection of built-in and portable functionality known as
  + Standard library.
  + It supports application-level programming to network programming.
  + It also supports third-party application software for
    - Website construction
    - Numeric programming
    - Serial port access
    - Game development
* **WEB DEVELOPMENT**
  + Python plays a predominant role in web development
  + For example used in scrape websites (i.e) fetching data from websites
* **ARTIFICIAL INTELLIGENCE**
  + It used extensively in artificial intelligence especially in machine
  + Learning (make the machine to think, analyze and make a decisions).
  + The following library files are extensively used in AI.
    - Scikit- learn
    - Keras
    - Tensor flow
    - Opencv
* **GAMES**
  + It is used in computer graphics to develop GUI, desktop applications
    - Pygame for game development
    - Tkinter for GUI
* **BIG DATA**
  + It handles huge volume of data(big data) and supports parallel computing
    - PYDOOP -> used to write map reduce program and process date in HDFS cluster
    - DASK and pyspark -> big data processing
* **DATA SCIENCE**
  + Python in data science
  + Numpy and pandas deal with tabular matrix as well as statistical data.
  + Analyzed data is visualized using matplotlib and seabourn packages

**LITERALS:**

* A ***Literal*** is a notation for representing a fixed value in a source code. It is a sequence of characters that stands for itself.



**BOOLEAN LITERALS:**

Boolean Literals represents two expressions that are, **True** or **False**.

a = **True**

b = **False**

print**((**a+b**)**,bool**(**a+b**))**

**The output of the code above will be.**

1 True

**NUMERIC LITERALS:**

These are the Literals that are used to define values in **digits**or**Numbers.** Some of the Numeric Literals in Python are **Int, Float, Complex, Octal, Hexadecimal**.

c = 23 # Integer Literal

d = 0o257 # Octal Literal

e = 23.657787 # Floating Point Literal

f = 23+5j # Complex Literal

g = 0x18d # Hexadecimal Literal

print**(**"Value of c is ",c**)**

print**(**"Value of d is ",d**)**

print**(**"Value of e is ",e**)**

print**(**"Value of f is ",f.real,f.imag**)**

print**(**"Value of g is ",g**)**

**Note:** Octal and Hexadecimal notations will automatically print their corresponding decimal values.

Value of c is  23

Value of d is  175

Value of e is  23.657787

Value of f is  23.0 5.0

Value of g is  397

**VARIABLES:**

* They are names given to the values used in programs.
* Values in variable are changing one. So it is called as variables. It is also called as identifiers

**RULES:**

* First letter should be alphabets
* From 2nd letter onwards it can be alphabets or numbers.
* It is sensitive language so uppercase and lowercase variables are different
* Start the variable with single underscore indicates that variable is private in python
* Start with double underscore indicates that the variable is strongly private in python
* Special characters are not allowed except underscore

Examples:  x, name, rollno, stud\_name, \_ \_x1, \_x2

**DATA TYPES:**

* No Primitive Data type
* It is used to specify what type of value is going to be stored in the variables
* It is not necessary to specify the data type for a variable.
* If the data type is not specified, python will decide the data type depending upon the value stored in that variable.
* Everything is an object in python.
* Data types are actually classes and the variables declared by users are instances (object) of these classes.

Important data types are

* Numbers  : Integer, float
* String
* Boolean
* Complex
* List
* Tuples
* Dictionary

**String :**

* Any characters within quotes are called string i.e. alphabets, numbers and special characters.
* String is represented with
* „ „   Single quote
* “   “  Double quotes
* “”‟‟   “”‟‟ Triple double quotes
* „‟‟    „‟‟‟    Triple single quotes
* Triple quotes is used to specify multiple lines.

**OPERATORS:**

Operators are used to perform operations on variables and values.

Python divides the operators in the following groups:

* Arithmetic operators
* Assignment operators
* Comparison operators
* Logical operators
* Identity operators
* Membership operators
* Bitwise operators

**ARITHMETIC OPERATORS:**

•It is used to do mathematical calculation like addition, subtraction etc.

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| + | Addition |
| - | Subtraction |
| \* | Multiplication |
| / | Division (Results in float) |
| % | Modulo (Remainder) |
| // | Floor Division |
| \*\* | Power (X\*\* Y=> XY) |

EXAMPLE:

x = 15

y = 4

print('x + y =',x+y)

 Output: x + y = 19

print('x - y =',x-y)

 Output: x - y = 11

print('x \* y =',x\*y)

 Output: x \* y = 60

print('x / y =',x/y)

 Output: x / y = 3.75

print('x // y =',x//y)

 Output: x // y = 3

print('x \*\* y =',x\*\*y)

 Output: x \*\* y = 50625

**ASSIGNMENT OPERATORS:**

Assignment operators are used in Python to assign values to variables.

|  |  |  |
| --- | --- | --- |
| Operator | Example | Equivalent to |
| = | x = 5 | x = 5 |
| += | x += 5 | x = x + 5 |
| -= | x -= 5 | x = x - 5 |
| \*= | x \*= 5 | x = x \* 5 |
| /= | x /= 5 | x = x / 5 |
| %= | x %= 5 | x = x % 5 |
| //= | x //= 5 | x = x // 5 |
| \*\*= | x \*\*= 5 | x = x \*\* 5 |
| &= | x &= 5 | x = x & 5 |
| |= | x |= 5 | x = x | 5 |
| ^= | x ^= 5 | x = x ^ 5 |
| >>= | x >>= 5 | x = x >> 5 |
| <<= | x <<= 5 | x = x << 5 |

**COMPARISON OPERATORS:**

          It is used to check the relationship between two operands or two values

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| < | Less than |
| > | Greater than |
| <= | Less than or Equal to |
| >= | Greater than or Equal to |
| == | Equal to |
| != | Not Equal to |

EXAMPLE:

x = 10

y = 12

print('x > y is',x>y)

 Output: x > y is False

print('x < y is',x<y)

 Output: x < y is True

print('x == y is',x==y)

 Output: x == y is False

print('x != y is',x!=y)

 Output: x != y is True

print('x >= y is',x>=y)

 Output: x >= y is False

print('x <= y is',x<=y)

 Output: x <= y is True

**LOGICAL OPERATORS:**

     It is used to check more than one relationship.

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| and | Logical AND (True When both conditions are true) |
| or | Logical OR (True if any one of the condition is true) |
| not | Logical NOT (complements) |

EXAMPLE:

x = True

y = False

print('x and y is',xand y)

Output: x and y is False

print('x or y is',xor y)

Output: x or y is True

print('not x is',not x)

Output: not x is False

**SPECIAL OPERATORS:**

Python language offers some special types of operators like the identity operator or the membership operator.

**IDENTITY OPERATORS:**

• There are two identity operators used to check the identical of the operands

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| is | will return True if operands are having same values |
| is not | will return true if operands are not having the same value |

EXAMPLE:

x1 = 5

y1 = 5

x2 = 'Hello'

y2 = 'Hello'

x3 = [1,2,3]

y3 = [1,2,3]

print(x1 isnot y1)

Output: False

print(x2 is y2)

Output: True

print(x3 is y3)

 Output: False

**MEMBERSHIP OPERATORS:**

• There are two membership operators to check if the value is present in the sequence or not

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| in | returns True if value or the value in the variable is found in sequence |
| not in | returns True if value or value in the variable is not found in sequence |

EXAMPLE:

x = 'Hello world'

y = {1:'a',2:'b'}

print('H'in x)

 Output: True

print('hello'notin x)

 Output: True

print(1in y)

 Output: True

print('a'in y)

Output: False

**BITWISE OPERATORS:**

Bitwise operators act on operands as if they were strings of binary digits. They operate bit by bit

|  |  |  |
| --- | --- | --- |
| Operator | Meaning | Example |
| & | Bitwise AND | x & y = 0 (0000 0000) |
| | | Bitwise OR | x | y = 14 (0000 1110) |
| ~ | Bitwise NOT | ~x = -11 (1111 0101) |
| ^ | Bitwise XOR | x ^ y = 14 (0000 1110) |
| >> | Bitwise right shift | x >> 2 = 2 (0000 0010) |
| << | Bitwise left shift | x << 2 = 40 (0010 1000) |

EXPRESSIONS:

An [expression](http://en.wikipedia.org/wiki/Expression_%28programming%29) is a combination of values, variables, operators, and calls to functions.

If we type an expression at the Python prompt, the interpreter **evaluates** it and displays the result, which is always a *value*:

**>>>**1+1

2

**>>>**len('hello')

5

The *evaluation of an expression* produces a value,A value all by itself is a simple expression, and so is a variable.

**>>>**17

17

**>>>**y =3.14

**>>>**x =len('hello')

**>>>**x

5

**>>>**y

3.14

**Python control structures**

A control structure (or flow of control) is a block of programming that analyses variables and chooses a direction in which to go based on given parameters.

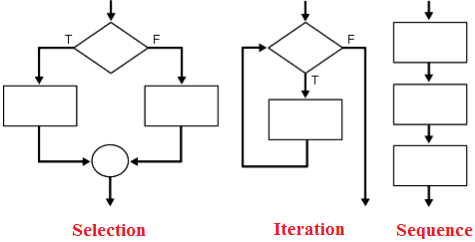
It is the basic **decision-making process** in programming and flow of control determines how a computer program will respond when given certain conditions and parameters.

There are two aspects of computer programming

**Data & Instructions**

To work with data, we need to understand variables and data types; to work with instructions, we need to understand control structures and statements.

**Flow of control** through any given program is implemented with three basic types of control structures: Sequential, Selection and Repetition.



**Sequence (or) Sequential:**

Sequential execution is when statements are executed one after another in order.

**Selection**

Selection used for decisions, branching - choosing between 2 or more alternative paths.

* if
* if...else
* switch

**Repetition**

Repetition used for looping, i.e. repeating a piece of code multiple times in a row.

* while loop
* do..while loop
* for loop

**SELECTION:**

Python programming language provides following types of decision making statements.

* if statements
* if....else statements
* if..elif..else statements
* nested if statements

**INDENTATION IN PYTHON**

* Indentation is a unique feature of Python.
* Indentation of blocks of code describes the program structure.
* In Python, indentation is used to associate and group statements.

**Example:**

if age ==18:

First clause of

if statement                              print(“ You are eligible to vote”)

else:

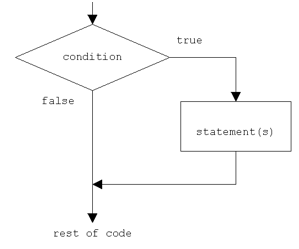
Second clause

of if statement                          print(“ You are not eligible to vote”)

* A **header** in Python is a specific keyword followed by a colon.
* In the above example, the if-else statement contains two headers
* The set of statements following a header in Python is called a **suite** or **block**.
* The statements of a given suite must all be indented to the same amount.
* A header and its associated suite are together called as a **clause**.
* A compound statement in Python may consist of one or more clauses and commonly 4 spaces are used for each level of indentation.

**if statements:**

It is executed based on conditions, if it is true it executes the statements in true block otherwise it skip the statements.



**Syntax:**

**if expression:**

**statements**

Example:

x=20

y=10

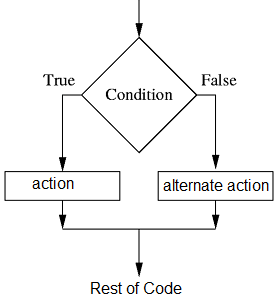
if x > y :

   print(" X is bigger ")

**Output:** X is bigger

**if....else statements:**

It is executed based on conditions, if it is true it executes the statements in true block otherwise it executes false block statements.



**Syntax:**

  if expression:

statements

else:

statements

**Example:**

x=20

y=30

if x > y :

   print(" X is bigger ")

else:

print(“Y is bigger”)

**MULTI-WAY SELECTION:**

* In Python, if statements can occur within another if statement and termed as nested statements.
* Python supports constructing multi-way selection by involving multiple nested if statements and other involving multiple nested if statements and other involving a single if statement executed and the use of elif headers.

**if..elif..else statements:**

The elif is short for else if and is useful to avoid excessive indentation.

**Syntax:**

if expression:

  statements

elif expression:

   statements

else:

   statements

Example:

 x= -10

if x > 0 :

print(" X is Positive no ")

elif x<0:

print(" X is Negative no ")

elif x==0:

print(“ X is Zero”)

else:

print( “Not a valid number”)

**Output:** X is Negative no

**Nested if statements:**

          Placing an if statement inside another if statement is termed as nested if.

**Syntax:**

**if condition:**

**if condition:**

**statements**

**else:**

**statements**

**else:**

**statements**

**Example:**

mark = 72

      if mark > 50:

          if mark > = 80:

            print ("You got A Grade !!")

          elif mark > =60 and mark < 80 :

            print ("You got B Grade !!")

          else:

            print ("You got C Grade !!")

      else:

        print("You failed!!")

**ITERATIVE CONTROL:**

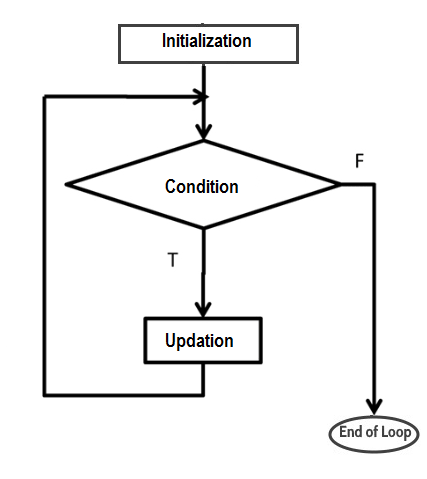
* The statements that allow a set of instructions to be performed repeatedly are referred as iteration statements.
* The iteration statements are also called loops or looping statements.
* Every loop has 3 parts:
* Initialization
* Condition
* Updation

**TYPES:**

* While statement
* Infinite Loops
* Definite Versus Indefinite Loops
* Boolean flags & Indefinite Loops

**3.4.1 While statement:**

* A while statement is an entry controlled loop statement that repeatedly executes a set of statements based on Boolean expression.



**Syntax:**

**while(condition):**

**statements**

**Example:**

x=0

while(x<=5):

print(x)

x+=1

**Output:**

**0**

**1**

**2**

**3**

**4**

**5**

**INFINITE LOOPS:**

* An infinite loop is an iterative control structure that never terminates.
* An infinite for loop has missing test-expression and generally results in programming errors.
* If the condition of a while loop can never be false, an infinite loop will result when executed.
* Example:

x=0

while(x<=5):

    print(x)

* In the above example, incrementing x is missing; therefore the value of x retains with same value in all iterations.

* So the expression x <= 5 is always true. Thus the loop would never terminate.
* Infinite loops can cause a program to hang, so the program must be terminated using ctrl + c to interrupt the execution (terminate abruptly)

**Definite Versus Indefinite Loops:**

* A definite loop is a program in which the number of times the loop will iterate can be determined before the loop is executed.

**Example:**

x=0

while(x<=5):

  print(x)

x+=1

           The while loop executes 6 times, which is a definite number

* An indefinite loop is a program loop in which the number of times that the loop will iterate cannot be determined before the loop is executed.

Example:

n=0

while n>0:

      n=n+1

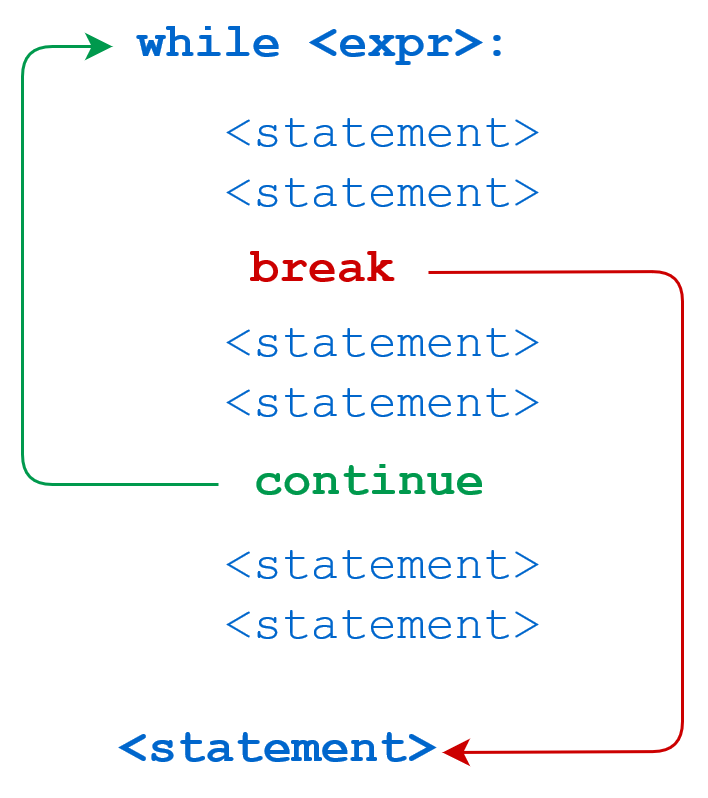
      print(n)

Initially n is 0. The controlling expression n > 0 is already false, so the loop body never executes.

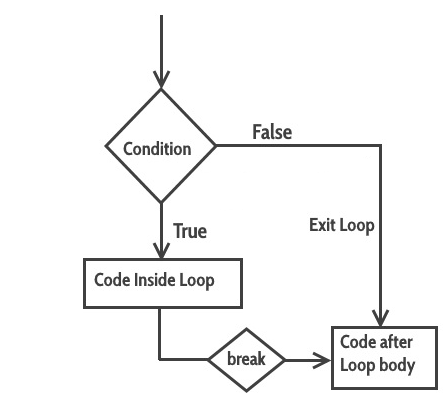
## Break and Continue Statements:

Python provides two keywords that **terminate** a loop iteration prematurely: break and continue.

* break leaves a loop.
* continue jumps to the next iteration. (skipping)



* The Python **break** statement immediately terminates a **loop entirely**. It proceeds to the first statement following the loop.



**Example for break:**

x=10

while True:

  print (x)

  x+=2;

  if x>20:

    break

print("After Break")

**Output:**

10

12

14

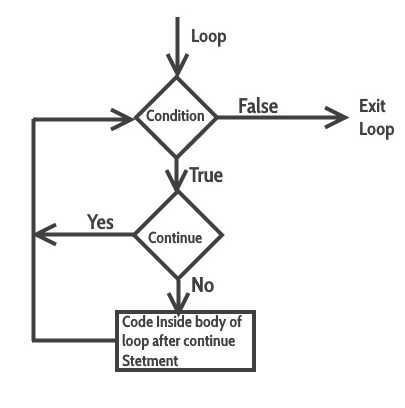
16

18

20

After Break

* The Python **continue** statement immediately terminates the current loop iteration. Execution jumps to the top of the loop.(ie., it skips the current iteration)

* 

**Example for Continue:**

x=0

while x < 50:

  x+=10

  if x==30:

    continue

  print (x)

print("Loop Over")

**Output:**

10

20

40

50

Loop Over

## Else clause on Python while statement

* This is a unique feature of **Python** and not found in most other programming languages. The else clause in **Python while loop** is only executed when your while condition becomes false.

**Syntax:**

while (condition) :

   statement(s)

else:

statement(s)

**Example:**

x = 5

while (x <=10):

  print (x )

  x = x +1

else:

  print(x , "  Inside Else")

**Output:**

5

6

7

8

9

10

11    Inside Else

**One Line While Loop:**

**x,y=0,5**

**while (x<y): x +=1; print(x);**

**BOOLEAN EXPRESSIONS**

* Boolean data type contains two Boolean values true and false in python
* The expressions that result into 0(false) or 1(true) are called Boolean expressions
* Boolean expressions are used to denote the conditions for selection and iterative control statements

**BOOLEAN OPERATORS:**

* In python **and ,or, not** are Boolean or logical operators which relate more than one condition except not operator
* Logical **and** is true only when both it operands and true otherwise it is false
* Logical OR is true when either or both of its operands are true and thus false only when both operands are false
* Logical not reserves truth values that is **NOT FALSE** equals **TRUE** and **NOT TRUE** equals **FALSE**

**Boolean Logic Truth Table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **X** | **Y** | **X AND Y** | **X OR Y** | **NOT X** |
| False | False | False | False | True |
| True | False | False | True | False |
| False | True | False | True | - |
| True | True | True | True | - |

**BOOLEAN FLAGS AND INDEFINITE LOOPS:**

* The condition of a given while loop is denoted by a single Boolean variable, called a Boolean flag.
* Example:

while True:

    print('foo')

This program doesn’t end by its own.so we have to end using ctrl+c.

Clearly, True will never be false. Thus, while True: initiates an infinite loop that will theoretically run forever.

**PYTHON COLLECTIONS (ARRAYS):**

There are four collection data types in the Python programming language:

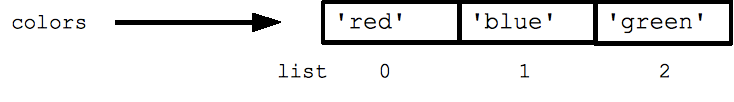
* [**List**](https://www.w3schools.com/python/python_lists.asp) is a collection which is ordered and changeable. Allows duplicate members.
* [**Tuple**](https://www.w3schools.com/python/python_tuples.asp) is a collection which is ordered and unchangeable. Allows duplicate members.
* [**Set**](https://www.w3schools.com/python/python_sets.asp) is a collection which is unordered, unchangeable\*, and unindexed. No duplicate members.
* **Dictionary** is a collection which is ordered\*\* and changeable. No duplicate members.

**LIST:**

* A list in Python is used to store the sequence of various types of data (OR) collection of values or items of different types.
* The items in the list are separated with the comma (,) and enclosed with the square brackets [].

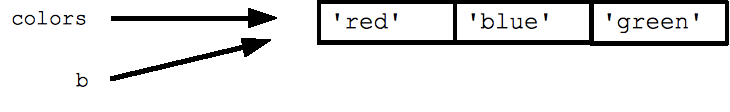
**LIST STRUCTURE:**

colors = ['red', 'blue', 'green']  
  print colors[0]    ## red  
  print colors[2]    ## green  
  print len(colors)  ## 3



* We use the len() function and square brackets [ ] to access data, with the first element at index[0].
* Assignment with an = on lists does not make a copy. Instead, assignment makes the two variables point to the one list in memory.

  b = colors   ## Does not copy the list



### Characteristics of Lists

The list has the following characteristics:

* The lists are ordered.
* The element of the list can access by index.
* The lists are mutable types (ie) we can modify its element after we created
* A list can store the number of various elements.
* An empty list is denoted by an empty pair of square brackets [ ].

## ACCESSING VALUES IN LISTS:

* To access and slice the values in lists, we can use the square brackets

list1 =['physics','chemistry',1997,2000];

list2 =[1,2,3,4,5,6,7];

print"list1[0]: ", list1[0]           ##### Accessing

print"list2[1:5]: ", list2[1:5]     ##### Slicing

**OUTPUT:**

list1[0]:  physics

list2[1:5]:  [2, 3, 4, 5]

* Python provides **negative indexing** which means indices are counted from right. The rightmost of the list has index -1.

Python Lists

**Example:**

list = [1,2,3,4,5]

print(list[-1])

print(list[-3:])

**Output:**

5

[3, 4, 5]

**BASIC LIST OPERATIONS:**

Lists respond to the + and \* operators much like strings; they mean concatenation and

|  |  |  |
| --- | --- | --- |
| **Python Expression** | **Results** | **Description** |
| len([1, 2, 3]) | 3 | Length |
| [1, 2, 3] + [4, 5, 6] | [1, 2, 3, 4, 5, 6] | Concatenation |
| ['Hi!'] \* 4 | ['Hi!', 'Hi!', 'Hi!', 'Hi!'] | Repetition |
| 3 in [1, 2, 3] | True | Membership |
| for x in [1, 2, 3]: print x, | 1 2 3 | Iteration |

## 

## PYTHON LIST METHODS:

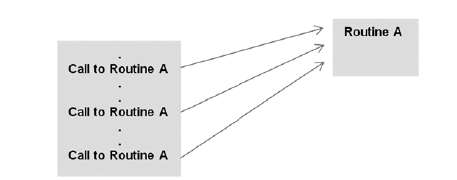
|  |
| --- |
| Methods that are available with list. They are accessed as list.method(). |
| [**append() -**Add an element to the end of the list](https://www.programiz.com/python-programming/methods/list/append) |
| [**extend()**-Add all elements of a list to the another list](https://www.programiz.com/python-programming/methods/list/extend) |
| [**insert()**-Insert an item at the defined index](https://www.programiz.com/python-programming/methods/list/insert) |
| [**remove()**-Removes an item from the list](https://www.programiz.com/python-programming/methods/list/remove) |
| [**pop()**-Removes and returns an element at the given index](https://www.programiz.com/python-programming/methods/list/pop) |
| [**clear()**- Removes all items from the list](https://www.programiz.com/python-programming/methods/list/clear) |
| [**index()**- Returns the index of the first matched item](https://www.programiz.com/python-programming/methods/list/index) |
| [**count()**- Returns the count of the number of items passed as an argument](https://www.programiz.com/python-programming/methods/list/count) |
| [**sort()**- Sort items in a list in ascending order](https://www.programiz.com/python-programming/methods/list/sort) |
| [**reverse()**- Reverse the order of items in the list](https://www.programiz.com/python-programming/methods/list/reverse) |
| [**copy()**- Returns a shallow copy of the list](https://www.programiz.com/python-programming/methods/list/copy) PYTHON TUPLE:  * A tuple is an immutable linear data structure, and it cannot be altered. * The difference between tuples and lists are tuples are denoted by parentheses instead of square brackets as given below:         nums = (10, 20, 30)   * An empty tuple is represented by a set of empty parentheses * Any attempt to alter a tuple is invalid. Thus delete, update, insert and append operations are not defined on tuples.   **Example:**  tuple1 = (10, 20, 30, 40, 50, 60)  print(tuple1)  count = 0  **for** i in tuple1:      print("tuple1[%d] = %d"%(count, i))      count = count+1  **Output:**  (10, 20, 30, 40, 50, 60)  tuple1[0] = 10  tuple1[1] = 20  tuple1[2] = 30  tuple1[3] = 40  tuple1[4] = 50  tuple1[5] = 60    **DIFFERENCE BETWEEN LIST & TUPLE:**   |  |  | | --- | --- | | **LIST** | **TUPLE** | | A list is a mutable linear data structure, it can be altered | A tuple is an immutable linear data structure, and it cannot be altered. | | The List has the variable length. | The tuple has the fixed length. | | The list provides more functionality than a tuple. | The tuple provides less functionality than the list. | | The value of the items can be changed in list | The value of the items cannot be changed in tuple | | The lists are less memory efficient than a tuple. | The tuples are more memory efficient because of its immutability. |   **DICTIONARY:**   * **Dictionary**in Python is a collection of data values * Dictionaries are used to store data values in key:value pairs. * Key-value is provided in the dictionary to make it more optimized. * Values in a dictionary can be of any data type and can be duplicated, whereas keys can’t be repeated and must be *immutable*. * Dictionaries are written with curly brackets, and have keys and values:   **Example:**  thisdict = {   "brand": "Ford",   "model": "Mustang",   "year": 1964 } print(thisdict) Dictionary Items:  * Dictionary items are ordered, changeable, and does not allow duplicates. * Dictionary items are presented in key:value pairs, and can be referred to by using the key name.   thisdict = {   "brand": "Ford",   "model": "Mustang",   "year": 1964 } print(thisdict["brand"])   * Dictionaries are changeable, meaning that we can change, add or remove items after the dictionary has been created. * Dictionaries cannot have two items with the same key:   thisdict = {   "brand": "Ford",   "model": "Mustang",   "year": 1964,   "year": 2020 } print(thisdict) Dictionary Length: To determine how many items a dictionary has, use the len() function:  print(len(thisdict)) Dictionary Items - Data Types: The values in dictionary items can be of any data type:  thisdict = {   "brand": "Ford",   "electric": False,   "year": 1964,   "colors": ["red", "white", "blue"] } type(): thisdict = {   "brand": "Ford",   "model": "Mustang",   "year": 1964 } print(type(thisdict)) ACCESS DICTIONARY ITEMS:Accessing Items  * We can access the items of a dictionary by referring to its key name, inside square brackets   thisdict = {   "brand": "Ford",   "model": "Mustang",   "year": 1964 } x = thisdict["model"]   * There is also a method called get() that will give you the same result  Get Keys:  * The keys() method will return a list of all the keys in the dictionary.   x = thisdict.keys()  **Example:**  car = { "brand": "Ford", "model": "Mustang", "year": 1964 } x = car.keys() print(x) #before the change car["color"] = "white" print(x) #after the change Get Values:  * The values() method will return a list of all the values in the dictionary.   x = thisdict.values()   * Make a change in the original dictionary   car = { "brand": "Ford", "model": "Mustang", "year": 1964 } x = car.values() print(x) #before the change car["year"] = 2020 print(x) #after the change Get Items:  * The items() method will return each item in a dictionary. * Get a list of the key:value pairs   x = thisdict.items()  Example:  car = { "brand": "Ford", "model": "Mustang", "year": 1964 } x = car.items() print(x) #before the change car["year"] = 2020 print(x) #after the change Check if Key Exists:  * To determine if a specified key is present in a dictionary use the in keyword:   thisdict = {   "brand": "Ford",   "model": "Mustang",   "year": 1964 } if "model" in thisdict:   print("Yes, 'model' is one of the keys in the thisdict dictionary") CHANGE DICTIONARY ITEMS:Change Values:  * You can change the value of a specific item by referring to its key name:   thisdict = {   "brand": "Ford",   "model": "Mustang",   "year": 1964 } thisdict["year"] = 2018 Update Dictionary:  The update() method will update the dictionary with the items from the given argument.  * The argument must be a dictionary, or an iterable object with key:value pairs.   thisdict = {   "brand": "Ford",   "model": "Mustang",   "year": 1964 } thisdict.update({"year": 2020}) ADD DICTIONARY ITEMS:Adding Items:  * Adding an item to the dictionary is done by using a new index key and assigning a value to it:   thisdict = {   "brand": "Ford",   "model": "Mustang",   "year": 1964 } thisdict["color"] = "red" print(thisdict) Update Dictionary:  * The update() method will update the dictionary with the items from a given argument. If the item does not exist, the item will be added. * The argument must be a dictionary, or an iterable object with key:value pairs.   thisdict = {   "brand": "Ford",   "model": "Mustang",   "year": 1964 }  thisdict.update({"color": "red"}) REMOVE DICTIONARY ITEMSRemoving Items:  * There are several methods to remove items from a dictionary  Example  * The pop() method removes the item with the specified key name:   thisdict = {   "brand": "Ford",   "model": "Mustang",   "year": 1964 } thisdict.pop("model") print(thisdict)   * The popitem() method removes the last inserted item (in versions before 3.7, a random item is removed instead)   thisdict = {   "brand": "Ford",   "model": "Mustang",   "year": 1964 } thisdict.popitem() print(thisdict)   * The del keyword removes the item with the specified key name   thisdict = {   "brand": "Ford",   "model": "Mustang",   "year": 1964 } del thisdict["model"] print(thisdict)   * The del keyword can also delete the dictionary completely   thisdict = {   "brand": "Ford",   "model": "Mustang",   "year": 1964 } del thisdict print(thisdict) #this will cause an error because "thisdict" no longer exists.   * The clear() method empties the dictionary   thisdict = {   "brand": "Ford",   "model": "Mustang",   "year": 1964 } thisdict.clear() print(thisdict) LOOP DICTIONARIES:Loop Through a Dictionary  * You can loop through a dictionary by using a for loop. * When looping through a dictionary, the return value are the *keys* of the dictionary, but there are methods to return the *values* as well.  Example  * Print all key names in the dictionary, one by one   for x in thisdict:   print(x)   * Print all *values* in the dictionary, one by one   for x in thisdict:   print(thisdict[x])   * You can also use the values() method to return values of a dictionary   for x in thisdict.values():   print(x)   * You can use the keys() method to return the keys of a dictionary   for x in thisdict.keys():   print(x)   * Loop through both *keys* and *values*, by using the items() method   for x, y in thisdict.items():   print(x, y) COPY DICTIONARIES: **Copy a Dictionary**   * You cannot copy a dictionary simply by typing dict2 = dict1, because: dict2 will only be a *reference* to dict1, and changes made in dict1 will automatically also be made in dict2. * There are ways to make a copy, one way is to use the built-in Dictionary method copy().  Example  * Make a copy of a dictionary with the copy() method:   thisdict = {   "brand": "Ford",   "model": "Mustang",   "year": 1964 } mydict = thisdict.copy() print(mydict)   * Another way to make a copy is to use the built-in function dict(). * Make a copy of a dictionary with the dict() function   thisdict = {   "brand": "Ford",   "model": "Mustang",   "year": 1964 } mydict = dict(thisdict) print(mydict) NESTED DICTIONARIES: A dictionary can contain dictionaries, this is called nested dictionaries. Example Create a dictionary that contain three dictionaries:  myfamily = {   "child1" : {     "name" : "Emil",     "year" : 2004   },   "child2" : {     "name" : "Tobias",     "year" : 2007   },   "child3" : {     "name" : "Linus",     "year" : 2011   } }  **DICTIONARY METHODS:**  Python has a set of built-in methods that you can use on dictionaries.     |  |  | | --- | --- | | **Method** | **Description** | | [clear()](https://www.w3schools.com/python/ref_dictionary_clear.asp) | Removes all the elements from the dictionary | | [copy()](https://www.w3schools.com/python/ref_dictionary_copy.asp) | Returns a copy of the dictionary | | [fromkeys()](https://www.w3schools.com/python/ref_dictionary_fromkeys.asp) | Returns a dictionary with the specified keys and value | | [get()](https://www.w3schools.com/python/ref_dictionary_get.asp) | Returns the value of the specified key | | [items()](https://www.w3schools.com/python/ref_dictionary_items.asp) | Returns a list containing a tuple for each key value pair | | [keys()](https://www.w3schools.com/python/ref_dictionary_keys.asp) | Returns a list containing the dictionary's keys | | [pop()](https://www.w3schools.com/python/ref_dictionary_pop.asp) | Removes the element with the specified key | | [popitem()](https://www.w3schools.com/python/ref_dictionary_popitem.asp) | Removes the last inserted key-value pair | | [setdefault()](https://www.w3schools.com/python/ref_dictionary_setdefault.asp) | Returns the value of the specified key. If the key does not exist: insert the key, with the specified value | | [update()](https://www.w3schools.com/python/ref_dictionary_update.asp) | Updates the dictionary with the specified key-value pairs | | [values()](https://www.w3schools.com/python/ref_dictionary_values.asp) | Returns a list of all the values in the dictionary | |

**FUNCTIONS IN PYTHON:**

* Functions are program routines in Python

**PROGRAM ROUTINES:**

* A ***routine*** is a group of instructions performing a specific task and can be called as many times as needed in a specific program.
* It can be predefined in the programming language by the programmer.
* When a routine finishes executing, the control is automatically transferred to the point from where it was called.
* A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity
* Every function is developed for a specific purpose.



**FUNCTION DEFINITION**

* Python creates a function using the ***’def’*** keyword followed by the function name and parentheses ( ( ) ) and are defined at the top of the program.
* Every function must be defined before it is called.
* The code block within every function starts with a colon (:) and is indented.
* The statement return [expression] exits a function. A return statement with no arguments is the same as return None.

**SYNTAX:**

def function-name(Parameter list): --🡪 Function Header

    statements                                   --🡪  Function Body

    return [expression]

**EXAMPLE:**

defsayHello():

  print("Hello World!!")

sayHello() # call the function here

**OUTPUT:**

Hello World!!

* Points to be remembered:
* sayHello() is a function name, it does not contain any parameters. If it contains any parameters means it should be separated by comma (,) which are termed as formal parameters.
* After (colon : ) function definition starts. If we passes any parameters in definition then formal parameters should be replaced by actual parameters.

**Example:**

defevenOdd(x):

    if (x % 2 == 0):

        print("even")

    else:

        print("odd")

evenOdd(2)

evenOdd(3)

**Example:**

def swap(x, y):

    temp = x

    x = y

    y = temp

x = 2

y = 3

swap(x, y)

print(x)

print(y)

**VALUE RETURNING FUNCTIONS**

* A parameter is the variable which is part of the method’s signature (method declaration).
* Parameters are specified within the pair of parentheses in the function definition, separated by commas.
* When we call the function, we supply the values in the same way.

**EXAMPLE:**

deffindSum(x,y):

  z=x+y

  print("Sum of numbers are :" ,z)

findSum(10,20)

findSum(100,50)

findSum(4,2)

**OUTPUT:**

Sum of numbers are : 30

Sum of numbers are : 150

Sum of numbers are : 6

**EXAMPLE:**

defdispStatus(name,age):

  print("Name is :" ,name)

  print("Age is :" ,age)

dispStatus("John",35)

dispStatus("Dev",50)

**OUTPUT:**

Name is : John

Age is : 35

Name is :Dev

Age is : 50

**DEFAULT ARGUMENTS:**

* Assigning default values for function parameters is known as default arguments. The default value is assigned by using assignment (=) operator.

**EXAMPLE:**

defdispStatus(name,age=25):

  print("Name is :" ,name)

  print("Age is :" ,age)

dispStatus("John")

dispStatus("Dev")

**OUTPUT:**

Name is : John

Age is : 25

Name is :Dev

Age is : 25

**Example:**

defmyFun(x, y=50):

    print("x: ", x)

    print("y: ", y)

myfunc(20)

**OUTPUT:**

 x:  10

 y:  50

**RETURNING A VALUE:**

The return statement is used to return a value from a function.

**EXAMPLE:**

deffindSum(x,y):

  return  x+y

print(findSum(10,20))

print(findSum(100,50))

print(findSum(4,2))

**OUTPUT:**

30

150

6

**NON - VALUE RETURNING FUNCTIONS**

* Functions do not have declared return types.
* A function without an explicit **return** statement returns **None**. In the case of no arguments and no return value, the definition is very simple.
* **A *Non-Value returning function*** is a function thatproduces side effects.
* A ***Side – Effect*** is an action like printing the output on the screen, other than returning a value.
* When a non-value returning functions are called, the function call is just a statement.

**Example:**

defdisp():

  print('hi')

    print('hello')

print(disp())

**Output**

hi

hello

None

**CALLING VALUE RETURNING FUNCTIONS**

* Function calls to value returning functions can be used anywhere where the function’s return value is suitable.
* An expression, for example, can:

1. Contain multiple function calls.
2. A function call can contain another function call as arguments.
3. Conditional expressions may comprise of function calls.
4. The arguments in print function calls may contain function calls.

**ASSIGNING FUNCTIONS TO VARIABLES:**

A function can take multiple arguments, these arguments can be objects, variables (of same or different data types) and functions.

**EXAMPLE:**

deffunc(text):

    returntext.upper()

print(func('Hello'))

sample = func

print(sample('Hello'))

**OUTPUT:**

HELLO

HELLO

When we assign a function to a variable we don't use the () but simply the name of the function.

**EXAMPLE:**

deffindSum(x,y):

  returnx+y

f1 = findSum(10,20)

print(f1)

f1 = findSum(40,20)

print(f1)

**OUTPUT:**

30

60

**FUNCTIONS AS PARAMETERS:**

We can pass functions as parameters to other functions.

**Example:**

defsayHello(func):

  return "Hello " + func

defsayName(name):

  return name

print (sayHello(sayName("Dev")))

**Output:**

Hello Dev

**PARAMETER PASSING**

* Parameter Passing is the process of passing arguments to a function.

**ACTUAL ARGUMENTS VS FORMAL PARAMETERS**

* **Arguments** are values that are passed into function (or method) in the calling function.
* **Parameters** are variables (identifiers) specified in the (header of) function definition.
* When a function is defined, it may have some parameters. These parameters are useful to receive values from outside of the function. They are called ‘***Formal Parameters’***.
* When a function is called, it should pass data or values to the function. These values are called ***‘Actual Arguments’.***
* In the following code, ‘a’ and ‘b’ are called Formal Parameters and ‘num1’ and ‘num2’ are called Actual Arguments.
* With function call **arguments**are what we pass into the function **parameters**. Or it can be said, parameters are names (variables) in a function(or method) definition that specifies types of arguments that the function can accept.
* The correspondence of actual arguments and formal parameters is determined by the order of the arguments passed, and not their names.

**EXAMPLE:**

***Sample Program for adding two numbers using Actual Arguments and Formal parameters***

**def**add\_num(a,b): # formal parameters a and b

sum=a+b;

**return** sum;  #return value

num1=int(input("input the number one: "))

num2=int(input("input the number one :"))

print("The sum is",add\_num(num1,num2)) # actual parameters

**Output**

input the number one: 20

input the number one :30

The sum is 50

**KEYWORD ARGUMENTS IN PYTHON**

* Python allows functions to be called using **Keyword arguments**. When we call functions in this way, the order (position) of the arguments can be changed.
* It is also known as **“named arguments”.**
* A **keyword argument** is an argument passed to a function or method which is preceded by a *keyword* and an equals sign. The general form is:

function(key=value)

* Where function is the function name, keyword is the keyword argument and value is the value or object passed as that keyword.

* A Keyword argument is usually specified by the parameter name.

**Example**

          defmyfunc(child3,child2,child1):

print('The Youngest Child is'+child3)

myfunc(child1='Hema',child2='Nema',child3='Rema')

***Output***

The Youngest Child is Rema

**ANONYMOUS FUNCTION**

* In Python, an anonymous function means that a function is without a name.
* The def keyword is used to define the normal functions and the lambda keyword is used to create anonymous functions.

**EXAMPLE:**

def cube(x): return x\*x\*x

cube\_v2 = lambda x : x\*x\*x

print(cube(7))

print(cube\_v2(7))

**RECURSION:**

* The term [Recursion](https://www.geeksforgeeks.org/recursion/#:~:text=The%20process%20in%20which%20a,%20can%20be%20solved%20quite%20easily.) can be defined as the process of defining something in terms of itself, it is a process in which a function calls itself directly or indirectly.

**Example:**

defrecursive\_factorial(n):

   if n == 1:

       return n

   else:

       return n \* recursive\_factorial(n-1)

num=6

ifnum< 0:

   print("Invalid input ! Please enter a positive number.")

elifnum == 0:

   print("Factorial of number 0 is 1")

else:

   print("Factorial of number", num, "=", recursive\_factorial(num))

Output:

Factorial of a number 6 = 720

**STRINGS:**

* A string is a list of characters in order.
* A character is anything you can type on the keyboard in one keystroke, like a letter, a number, or a backslash.
* Strings can have spaces: "hello world".
* An empty string is a string that has 0 characters.
* Python strings are immutable

**STRING MANIPULATION:**

To manipulate strings, we can use some of Pythons built-in methods.

**Creation:**

word = "Hello World"

>>> print word

Hello World

**Accessing:**

Use [ ] to access characters in a string

word = "Hello World"

letter=word[0]

>>>print letter

H

### Length:

word = "Hello World"

>>>len(word)

11

### Finding:

word = "Hello World"

>>> print word.count('l') # count how many times l is in the string

3

>>>printword.find("H") # find the word H in the string

0

>>>printword.index("World") # find the letters World in the string

6

### Count:

s = "Count, the number of spaces"

>>> print s.count(' ')

8

### Slicing:

Use [ # : # ] to get set of letter

Always indexing starts from 0

word = "Hello World"

print word[0] #get one char of the word

print word[0:1] #get one char of the word (same as above)

print word[0:3] #get the first three char

print word[:3] #get the first three char

print word[-3:] #get the last three char

print word[3:] #get all but the three first char

print word[:-3] #get all but the three last character

word = "Hello World"

word[start:end] # items start through end-1

word[start:] # items start through the rest of the list

word[:end] # items from the beginning through end-1

word[:] # a copy of the whole list

### Split Strings

word = "Hello World"

>>>word.split(' ') # Split on whitespace

['Hello', 'World']

### Startswith / Endswith:

word = "hello world"

>>>word.startswith("H")

True

>>>word.endswith("d")

True

>>>word.endswith("w")

False

### Repeat Strings:

print "."\* 10 # prints ten dots

>>> print "." \* 10

..........

### 

### Replacing

word = "Hello World"

>>>word.replace("Hello", "Goodbye")

'Goodbye World'

### Changing Upper and Lower Case Strings:

string = "Hello World"

>>> print string.upper()

HELLO WORLD

>>> print string.lower()

hello world

>>> print string.title()

Hello World

>>> print string.capitalize()

Hello world

>>> print string.swapcase()

hELLOwORLD

### 

### Reversing:

string = "Hello World"

>>> print ' '.join(reversed(string))

d l r o W o l l e H

### 

### Strip:

Python strings have the strip(), lstrip(), rstrip() methods for removing  
any character from both ends of a string.

 If the characters to be removed are not specified then white-space will be removed

>>> print word.strip(' ')

Hello World

strip() #removes from both ends

lstrip() #removes leading characters (Left-strip)

rstrip() #removes trailing characters (Right-strip)

>>>word = " xyz "

>>> print word

xyz

>>> print word.strip()

xyz

>>> print word.lstrip()

xyz

>>> print word.rstrip()

xyz

### Concatenation:

To concatenate strings in Python use the “+” operator

"Hello " + "World" # = "Hello World"

"Hello " + "World" + "!"# = "Hello World!"

### Join:

>>> print ":".join(word) # #add a : between every char

H:e:l:l:o: :W:o:r:l:d

>>> print " ".join(word) # add a whitespace between every char

H e l l o W o r l d

### Testing:

A string in Python can be tested for truth value.

The return type will be in Boolean value (True or False)

word = "Hello World"

word.isalnum() #check if all char are alphanumeric

word.isalpha() #check if all char in the string are alphabetic

word.isdigit() #test if string contains digits

word.istitle() #test if string contains title words

word.isupper() #test if string contains upper case

word.islower() #test if string contains lower case

word.isspace() #test if string contains spaces

word.endswith('d') #test if string endswith a d

word.startswith('H') #test if string startswith H