1. **INTRODUCTION**

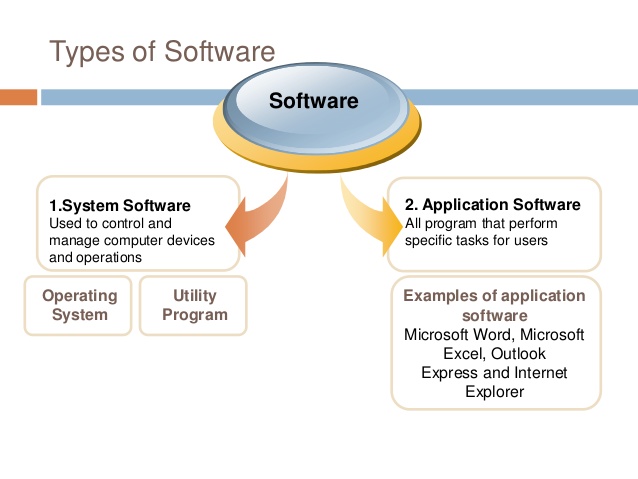
**Software :-**

* “A set of instructions given to the computer to solve a problem is called Software.”
* “Software is a single/collection of programs that performs a particular task.”
* Software is also called Program.
* Different software's are used to solve different problems.

**Types Of Software :-**

* There are two types of software on the basis of their functionality:

1. System Software
2. Application Software



**Application Software :-**

* Application Software includes programs that do real work for user.
* Application Software is created to perform specific tasks for a user.
* It is also known as application package.
* Application software is used to perform various applications on the computer.

Examples

* Microsoft Word
* Microsoft Excel

**Custom software :-**

* Custom software (also known as bespoke software or tailor made software's) is software that is specially developed for some specific organization or other user.

**Packaged Software :-**

* Packaged Software is a Software that is ready-made and available for sale, lease , or license to the general public.

**System Software :-**

* System Software is set of programs to control and manage the operations of a computer hardware.
* It enables application programs to execute properly.
* It consists of the basic operations as follows:
* Saving data on disk

Examples

* Operating Systems
* Utility Programs
* Device Drivers

**Operating System (OS) :-**

* An operating system (OS) is a collection of software that manages computer hardware resources and provides common services for computer programs.
* The operating system is a vital component of the system software in a computer system.
* Application programs usually require an operating system to function

**Utility program :-**

* Utility program is a system software that allows a user to analyze, configure and Maintain the computer.
* It performs a specific task related to the management of computer.

**Device Driver :-**

* A Device Driver is a computer program that operates or controls a particular type of device that is attached to a computer.
* Like printers

### What is a language?

* Language is the medium of communication to share ideas, opinion with each other. For an example, if we want to communicate with someone, we need a language it may be English, Hindi, Spanish or another language. But you need at least one language to communicate with someone (human/person).

### What is a programming language?

* Programming language is the medium of communication between you (a person) and a computer system. It is the set of some instructions written in a specific style (coding) to instruct the computer to do some specific task.

### Types of programming languages

There are basically three types of computer programming languages, they are

1. Low level programming languages
2. High level programming languages
3. Middle level programming languages

**Low level programming languages**

* These are machine dependent programming languages such as Binary and Assembly language.
* Since computer only understand the Binary language that means instructions in the form of 0’s and 1’s so these programming languages are the best way to give Binary Instructions to the computer directly.
* Assembly language needs to be converted in equivalent Binary code, so that computer can understand the instructions written in Assembly. Assembler is used to convert an assembly code to its equivalent Binary code.

**High level programming languages**

* These are the machine independent programming languages, which are easy to write, read, edit and understand.
* The languages like Java, .Net, Pascal, COBOL, C++, C, C#, Python and other
* High level programming languages have some special keywords, functions and class libraries by using them we can easily build a program for the computer.
* Programming translators such as Compilers and Interpreters are the system software’s which converts a program written in particular programming Since, High Level language programs are slower than Low level language programs; still these programming languages are popular to develop User End Applications.

**Middle Level programming language**

* The programming languages that have features of low level and high level programming languages
* C programming languages is the best example of Low Level Programming languages as it has features of low level and high level programming languages both.

**Python Programming Language**

* Python is a general-purpose object-oriented programming language with high-level programming capabilities.
* It has become famous because of its clear and easily understandable syntax, portability and easy to learn.
* Python was developed in the late eighties i.e. late 1980’s by **Guido van Rossum** at the **National Research Institute for Mathematics and Computer Science** in the Netherlands as a successor of ABC language capable of exception handling and interfacing

**Why Python ?**

* Interpreted Language: Python is processed at runtime by Python Interpreter.
* Object-Oriented Language: It supports object-oriented features and techniques of programming.
* Interactive Programming Language:  Users can interact with the python interpreter directly for writing programs.
* Easy language:  Python is easy to learn language especially for beginners.
* Straightforward Syntax:  The formation of python syntax is simple and straightforward which also makes it popular.
* Portable:  Python codes can be run on a wide variety of hardware platforms having the same interface.
* Extendable:  Users can add low level-modules to Python interpreter.
* Scalable:  Python provides an improved structure for supporting large programs then shell-scripts.

## 2. Python Basic Syntax

## Python Keywords

* Keywords are the reserved words in Python.
* They are used to define the syntax and structure of the Python language.
* In Python, keywords are case sensitive.
* There are 33 keywords in Python 3.3. This number can vary slightly in course of time.
* All the keywords except True, False and None are in lowercase and they must be written as it is.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | | | |
| False | class | Finally | is | return |
| None | continue | For | lambda | try |
| True | def | From | nonlocal | while |
| and | del | Global | not | with |
| as | elif | If | or | yield |
| assert | else | Import | pass |  |
| break | except | In | raise |  |

**Python Identifiers**

* Identifier is the name given to entities like class, functions, variables etc. in Python.

**Rules for writing identifiers**

1. Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore
2. An identifier cannot start with a digit.
3. Keywords cannot be used as identifiers.

>>>global = 1

File"<interactive input>", line 1

global = 1

^

SyntaxError: invalid syntax

1. We cannot use special symbols like !, @, #, $, % etc. in our identifier.

>>>a@ = 0

File"<interactive input>", line 1

a@ = 0

^

SyntaxError: invalid syntax

1. Identifier can be of any length.

## Python Statement

* Instructions that a Python interpreter can execute are called statements.
* For example,  a = 1 is an assignment statement.  if statement, for statement, while statement etc

### Multi-line statement

* In Python, end of a statement is marked by a newline character.
* But we can make a statement extend over multiple lines with the line continuation character (\).This is explicit line continuation.

a =1+2+3+ \

4+5+6+ \

7+8+9

* In Python, line continuation is implied inside parentheses ( ), brackets [ ] and braces { }.

a =(1+2+3+

4+5+6+

7+8+9)

* We could also put multiple statements in a single line using semicolons.

a =1; b =2; c =3

**Python Indentation**

* Most of the programming languages like C, C++, Java use braces { } to define a block of code. Python uses indentation.
* A code block (body of a [function](https://www.programiz.com/python-programming/function), [loop](https://www.programiz.com/python-programming/for-loop) etc.) starts with indentation and ends with the first unintended line. The amount of indentation is up to you, but it must be consistent throughout that block.
* Generally four whitespaces are used for indentation and is preferred over tabs.

for i in range(1,11):

print(i)

if i == 5:

break

## Python Comments

* In Python, we use the hash (#) symbol to start writing a comment.
* It extends up to the newline character. Comments are for programmers for better understanding of a program. Python Interpreter ignores comment.

#This is a comment

#print out Hello

### 

### Multi-line comments

* If we have comments that extend multiple lines, one way of doing it is to use hash (#) in the beginning of each line. For example:

#This is a long comment

#and it extends

#to multiple lines

* Another way of doing this is to use triple quotes, either ''' or """.
* These triple quotes are generally used for multi-line strings. But they can be used as multi-line comment as well.

"""This is also a

perfect example of

multi-line comments"""

### Doc string in Python

* Doc string is short for documentation string.
* It is a [string](https://www.programiz.com/python-programming/string) that occurs as the first statement in a module, function, class, or method definition.
* We must write what a function/class does in the doc string.
* Triple quotes are used while writing doc strings.

Example:

def double(num):

"""Function to double the value"""

return 2\*num

* Docstring is available to us as the attribute \_\_doc\_\_ of the function.

>>>print(double.\_\_doc\_\_)

Function to double the value

[Built-in Functions](https://www.programiz.com/python-programming/built-in-function)

* Python provides numerous [built-in functions](https://www.programiz.com/python-programming/built-in-function) that are readily available to us at the Python prompt.
* Some of the functions like input() and print() are widely used for standard input and output operations respectively.

## Python Output

* We use the print() function to output data to the standard output device (screen).

## print('This sentence is output to the screen')

## # Output: This sentence is output to the screen

## a = 5

## print('The value of a is', a)

## # Output: The value of a is 5

Python Input

* + we have the input() function to allow this. The syntax for input() is

input([prompt])

* + where prompt is the string we wish to display on the screen. It is optional.

>>> num = input('Enter a number: ')

Enter a number: 10

>>> num

'10'

## Python Variables

* A variable is a location in memory used to store some data (value).
* They are given unique names to differentiate between different memory locations.
* The rules for writing a variable name is same as the [rules for writing identifiers in Python](https://www.programiz.com/python-programming/keywords-identifier#rules).
* We don't need to declare a variable before using it.

**Variable assignment**

* We use the [assignment operator](https://www.programiz.com/python-programming/operators#arithmetic_operators) (=) to assign values to a variable. Any type of value can be assigned to any valid variable.

a =5

Multiple assignments

* In Python, multiple assignments can be made in a single statement as follows:

a, b, c =5,3.2,"Hello"

* If we want to assign the same value to multiple variables at once, we can do this as

x = y = z ="same"

**Data types in Python**

* Everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes.
* There are various data types in Python.

**Python Numbers**

* Integers
* floating point numbers
* complex numbers come under [Python numbers](https://www.programiz.com/python-programming/numbers) .
* They are defined as int, float and complex class in Python.
* We can use the type() function to know which class a variable or a value belongs
* We can use the isinstance() function to check if an object belongs to a particular class.

Example :

a = 5

print(a, "is of type", type(a))

a = 2.0

print(a, "is of type", type(a))

a = 1+2j

print(a,"iscomplexnumber?",isinstance(1+2j,complex))

* Integers can be of any length, it is only limited by the memory available.
* A floating point number is accurate up to 15 decimal places.
* Complex numbers are written in the form, x + yj,
* where x is the real part and
* y is the imaginary part.

>>> a = 1234567890123456789

>>>a

1234567890123456789

>>> b = 0.1234567890123456789

>>>b

0.12345678901234568

>>> c = 1+2j

>>>c

(1+2j)

### Python List

* [List](https://www.programiz.com/python-programming/list) is an ordered sequence of items. All the items in a list do not need to be of the same type.
* Declaring a list is pretty straight forward. Items separated by commas are enclosed within brackets [ ].

>>> a =[1,2.2,'python']

* We can use the slicing operator [ ] to extract an item or a range of items from a list. Index starts form 0 in Python.

a = [5,10,15,20,25,30,35,40]

# a[2] = 15

print("a[2] = ", a[2])

# a[0:3] = [5, 10, 15]

print("a[0:3] = ", a[0:3])

# a[5:] = [30, 35, 40]

print("a[5:] = ", a[5:])

* Lists are mutable, meaning, value of elements of a list can be altered.

>>> a =[1,2,3]

>>>a[2]=4

>>>a

[1, 2 , 3 ]

Python Tuple

* [Tuple](https://www.programiz.com/python-programming/tuple) is an ordered sequence of items same as list.The only difference is that tuples are immutable. Tuples once created cannot be modified.
* Tuples are used to write-protect data and are usually faster than list as it cannot change dynamically.
* It is defined within parentheses () where items are separated by commas.

>>> t =(5,'program',1+3j)

* We can use the slicing operator [] to extract items but we cannot change its value.

t = (5,'program', 1+3j)

# t[1] = 'program'

print("t[1] = ", t[1])

# t[0:3] = (5, 'program', (1+3j))

print("t[0:3] = ", t[0:3])

# Generates error

# Tuples are immutable

t[0] = 10

### Python Strings

* [String](https://www.programiz.com/python-programming/string) is sequence of Unicode characters.
* We can use single quotes or double quotes to represent strings.
* Multi-line strings can be denoted using triple quotes, ''' or """.

>>> s ="This is a string"

>>> s ='''a multiline

* Like list and tuple, slicing operator [ ] can be used with string. Strings are immutable.

s = 'Hello world!'

# s[4] = 'o'

print("s[4] = ", s[4])

# s[6:11] = 'world'

print("s[6:11] = ", s[6:11])

# Generates error

# Strings are immutable in Python

s[5] ='d'

### Python Set

* [Set](https://www.programiz.com/python-programming/set) is an unordered collection of unique items.
* Set is defined by values separated by comma inside braces { }.

a = {5,2,3,1,4}

# printing set variable

print("a = ", a)

# data type of variable a

print(type(a))

* Set have unique values. They eliminate duplicates.

>>> a ={1,2,2,3,3,3}

>>>a

{1,2,3}

* Set are unordered collection, indexing has no meaning. Hence the slicing operator [] does not work.

>>> a ={1,2,3}

>>>a[1]

Traceback(most recent call last):

TypeError:'set'object does not support indexing

Python Dictionary

* [Dictionary](https://www.programiz.com/python-programming/dictionary) is an unordered collection of key-value pairs.
* It is generally used when we have a huge amount of data.
* Dictionaries are optimized for retrieving data.
* We must know the key to retrieve the value.
* Dictionaries are defined within braces {} with each item being a pair in the form key:value. Key and value can be of any type.

>>> d ={1:'value','key':2}

>>>type(d)

<class'dict'>

### Conversion between data types

* We can convert between different data types by using different type conversion functions like
* int()
* float()
* str()
* list()
* tuple()
* dict()
* set()

Examples : **-**

>>>float(5)

5.0

>>>int(10.6)

10

>>>int(-10.6)

-10

>>>float('2.5')

2.5

>>>str(25)

'25'

>>>int('1p')

* We can even convert one sequence to another.

>>>set([1,2,3])

{1,2, 3}

>>>tuple({5,6,7})

(5,6,7)

>>>list('hello')

['h','e','l','l','o']

>>>dict([[1,2],[3,4]])

{1:2,3:4}

>>>dict([(3,26),(4,44)])

# 3. Python Operators

**What are operators in python?**

* Operators are special symbols in Python that carry out arithmetic or logical computation.
* The value that the operator operates on is called the operand.

**Arithmetic operators**

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

|  |  |  |
| --- | --- | --- |
| Operator | Meaning | Example |
| + | Add two operands or unary plus | X + y +2 |
| - | Subtract right operand from the left or unary minus | x - y -2 |
| \* | Multiply two operands | X \* y |
| / | Divide left operand by the right one (always results into float) | X / y |
| % | Modulus - remainder of the division of left operand by the right | x % y (remainder of x/y) |
| // | Floor division - division that results into whole number adjusted to the left in the number line | x // y |
| \*\* | Exponent - left operand raised to the power of right | X \*\* y  (x to the power y) |

Example:

x = 15

y = 4

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

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

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

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

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

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

**Comparison operators**

* Comparison operators are used to compare values.
* It either returns True or False according to the condition.

|  |  |  |
| --- | --- | --- |
| Operator | Meaning | Example |
| > | Greater that - True if left operand is greater than the right | x > y |
| < | Less that - True if left operand is less than the right | x < y |
| == | Equal to - True if both operands are equal | x == y |
| != | Not equal to - True if operands are not equal | x != y |
| >= | Greater than or equal to - True if left operand is greater than or equal to the right | x >= y |
| <= | Less than or equal to - True if left operand is less than or equal to the right | x <= y |

Example :

x = 10

y = 12

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

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

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

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

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

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

**Logical operators**

* Logical operators are the and, or, not operators.

|  |  |  |
| --- | --- | --- |
| Operator | Meaning | Example |
| and | True if both the operands are true | x and y |
| or | True if either of the operands is true | x or y |
| not | True if operand is false (complements the operand) | not x |

Example :

x = True

y = False

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

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

print('not x is',not x)

**Bitwise operators**

* Bitwise operators act on operands as if they were string of binary digits.
* It operates bit by bitFor example, 2 is 10 in binary and 7 is 111.

Let x = 10 (0000 1010 in binary) and

y = 4 (0000 0100in binary)

|  |  |  |
| --- | --- | --- |
| 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) |

**Assignment operators**

* Assignment operators are used in Python to assign values to variables.
* There are various compound operators in Python like a += 5 that adds to the variable and later assigns the same. It is equivalent to a = a + 5.

|  |  |  |
| --- | --- | --- |
| Operator | Example | Equivatent 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 |

**Special operators**

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

**Identity operators**

* + is and is not are the identity operators in Python.
  + They are used to check if two values (or variables) are located on the same part of the memory.
  + Two variables that are equal does not imply that they are identical.

|  |  |  |
| --- | --- | --- |
| Operator | Meaning | Example |
| is | True if the operands are identical (refer to the same object) | x is True |
| is not | True if the operands are not identical (do not refer to the same object) | x is not True |

Example :

y1 = 5

x2 = 'Hello'

y2 = 'Hello'

x3 = [1,2,3]

y3 = [1,2,3]

print(x1 is not y1)

print(x2 is y2)

print(x3 is y3)

### Membership operators

* in and not in are the membership operators in Python.
* They are used to test whether a value or variable is found in a sequence ([string](https://www.programiz.com/python-programming/string), [list](https://www.programiz.com/python-programming/list), [tuple](https://www.programiz.com/python-programming/tuple), [set](https://www.programiz.com/python-programming/set" \o "Python set)and [dictionary](https://www.programiz.com/python-programming/dictionary)).
* In a dictionary we can only test for presence of key, not the value.

|  |  |  |
| --- | --- | --- |
| Operator | Meaning | Example |
| in | True if value/variable is found in the sequence | 5 in x |
| not in | True if value/variable is not found in the sequence | 5 not in x |

Example :

x = 'Hello world'

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

print('H' in x)

print('hello' not in x)

print(1 in y)

print('a' in y)

## 4. Python  Conditional Statements

### The if statement

* The simplest form is the **if** statement, which has the genaral form:

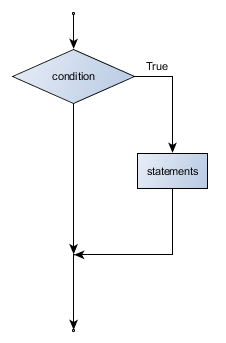
**If** BOOLEAN EXPRESSION:

STATEMENTS

* A few important things to note about if statements:

1. The colon (:) is significant and required. It separates the **header** of the **compound statement** from the **body**.
2. The line after the colon must be indented. It is standard in Python to use four spaces for indenting.
3. All lines indented the same amount after the colon will be executed whenever the BOOLEAN\_EXPRESSION is true.

Flow chart:



Example:

food = 'spam'

**if** food == 'spam':

**print**('Ummmm, my favorite!')

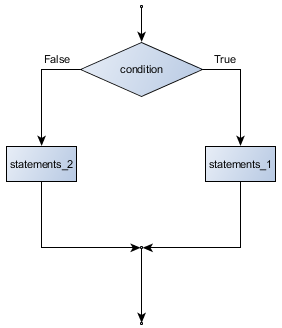
**print**('I feel like saying it 100 times...')

**print**(100 \* (food + '! '))**t**

### if else statement

* It is frequently the case that you want one thing to happen when a condition it true, and **something else** to happen when it is false.

Flow chart:



The syntax :

**if** BOOLEAN EXPRESSION:

STATEMENTS\_1 *# executed if condition evaluates to True*

**else**:

STATEMENTS\_2 *# executed if condition evaluates to False*

* + Each statement inside the if block of an if else statement is executed in order if the Boolean expression evaluates to True.
  + The entire block of statements is skipped if the Boolean expression evaluates to False, and instead all the statements under the else clause are executed.

Example:

**if** True: *# This is always true*

**pass** *# so this is always executed, but it does nothing*

**else**:

**pass**

## else if ladder:

* Sometimes there are more than two possibilities and we need more than two branches.
* One way to express a computation like that is a **chained conditional**:

Syntax:

**if** x < y:

STATEMENTS\_A

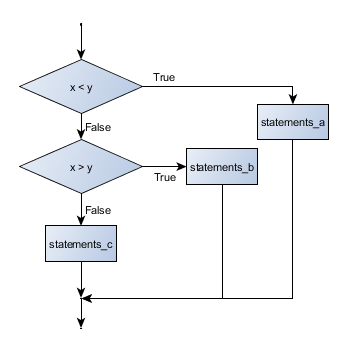
**elif** x > y:

STATEMENTS\_B

**else**:

STATEMENTS\_C

Flowchart :



Example:

**if** choice == 'a':

**print**("You chose 'a'.")

**elif** choice == 'b':

**print**("You chose 'b'.")

**elif** choice == 'c':

**print**("You chose 'c'.")

**else**:

**print**("Invalid choice.")

## Nested conditionals

* One conditional can also be **nested** within another. (It is the same theme of composibility, again!)

Syntax:

**if** x < y:

STATEMENTS\_A

**else**:

**if** x > y:

STATEMENTS\_B

**else**:

STATEMENTS\_C

Example

**if** 0 < x: *# assume x is an int here*

**if** x < 10:

**print**("x is a positive single digit.")

## 5. Python looping Statements

## What is for loop in Python?

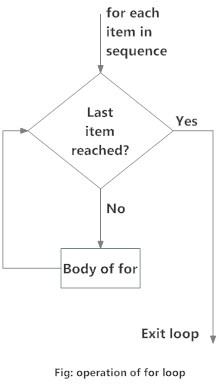
* The for loop in Python is used to iterate over a sequence ([list](https://www.programiz.com/python-programming/list), [tuple](https://www.programiz.com/python-programming/tuple), [string](https://www.programiz.com/python-programming/string)) or other iterable objects.
* Iterating over a sequence is called traversal.

### Syntax

for val in sequence:

Body of for

### Flowchart



Program to find the sum of all numbers stored in a list

numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]

sum = 0

for val in numbers:

sum = sum+val

print("The sum is", sum)

**range() function**

* We can generate a sequence of numbers using range() function.
* We can also define the start, stop and step size as range(start,stop,step size). step size defaults to 1 if not provided.

print(range(10))

# Output: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

print(list(range(2, 8)))

# Output: [2, 3, 4, 5, 6, 7]

print(list(range(2, 20, 3)))

# Output: [2, 5, 8, 11, 14, 17]

* We can use the range() function in for loops to iterate through a sequence of numbers.
* It can be combined with the len() function to iterate though a sequence using indexing. Here is an example.

Example:

genre = ['pop', 'rock', 'jazz']

# iterate over the list using index

for i in range(len(genre)):

print("I like", genre[i])

**for loop with else**

* A for loop can have an optional else block as well.
* The else part is executed if the items in the sequence used in for loop exhausts.
* break statement can be used to stop a for loop. In such case, the else part is ignored.

Example

digits = [0, 1, 5]

for i in digits:

print(i)

else:

print("No items left.")

**What is while loop in Python?**

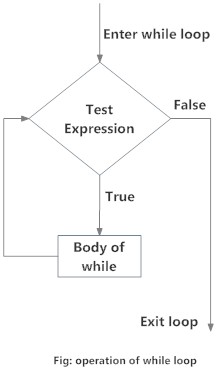
* The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true.

**Syntax**

while test\_expression:

Body of while

**Flowchart**



Example :

n = 10

# initialize sum and counter

sum = 0

i = 1

while i <= n:

sum = sum + i

i = i+1 # update counter

# print the sum

print("The sum is", sum)

## while loop with else

* The else part is executed if the condition in the while loop evaluates to False.
* The while loop can be terminated with a [break statement](https://www.programiz.com/python-programming/break-continue).In such case, the else part is ignored..

counter = 0

while counter < 3:

print("Inside loop")

counter = counter + 1

else:

print("Inside else")

**6. Python Control Statements**

* In Python, break and continue statements can alter the flow of a normal loop.
* Loops iterate over a block of code until test expression is false
* But sometimes we wish to terminate the current iteration or even the whole loop without checking test expression.
* The break and continue statements are used in these cases.

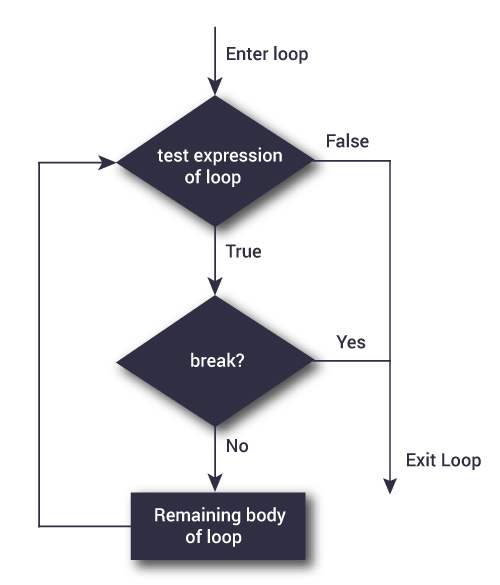
**Python break statement**

* The break statement terminates the loop containing it.
* Control of the program flows to the statement immediately after the body of the loop.
* If break statement is inside a nested loop (loop inside another loop), break will terminate the innermost loop.

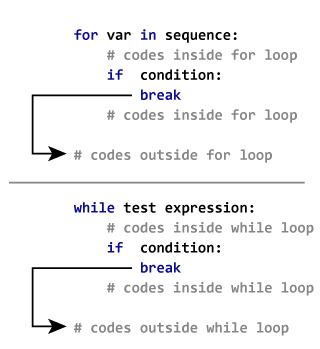
**Syntax**

break

**Flowchart**



The working of break statement in [for loop](https://www.programiz.com/python-programming/for-loop) and [while loop](https://www.programiz.com/python-programming/while-loop) :



Example :

for val in "string":

if val == "i":

break

print(val)

print("The end")

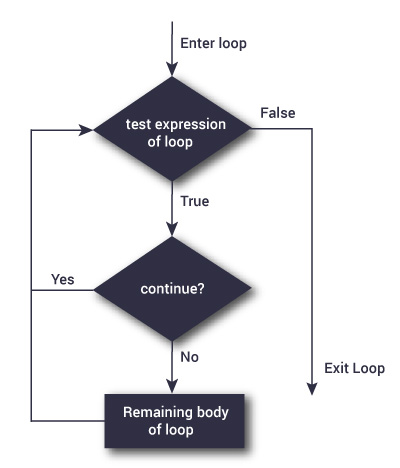
**Python continue statement**

* The continue statement is used to skip the rest of the code inside a loop for the current iteration only.
* Loop does not terminate but continues on with the next iteration.

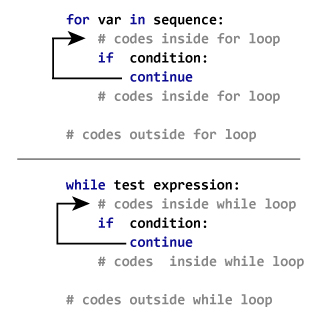
**Syntax**

continue

**Flowchart**



The working of continue statement in for and while loop:



Example

for val in "string":

if val == "i":

continue

print(val)

print("The end")

## pass statement in Python

* In Python programming, pass is a null statement.
* The difference between a [comment](https://www.programiz.com/python-programming/statement-indentation-comments)and pass statement in Python is that, while the interpreter ignores a comment entirely, pass is not ignored.
* However, nothing happens when pass is executed. It results into no operation (NOP).

### Syntax

### pass

* We generally use it as a placeholder.

sequence = {'p', 'a', 's', 's'}

for val in sequence:

pass

* We can do the same thing in an empty function or [class](https://www.programiz.com/python-programming/class) as well.

def function(args):

pass

class example:

pass

## 

## 7. Python Functions

## What is a function ?

* A function is a group of related statements that perform a specific task.
* Functions help break our program into smaller and modular chunks.
* As our program grows larger and larger, functions make it more organized and manageable.
* Furthermore, it avoids repetition and makes code reusable.

## Types of Functions

* we can divide functions into the following two types:

1. [Built-in functions](https://www.programiz.com/python-programming/built-in-function) - Functions that are built into Python.
2. [User-defined functions](https://www.programiz.com/python-programming/user-defined-function) - Functions defined by the users themselves.

### Syntax:

deffunction\_name(parameters):

"""docstring"""

statement(s)

* A function definition which consists of following components.

1. Keyword def marks the start of function header.
2. A function name to uniquely identify it.
3. Parameters (arguments) through which we pass values to a function. They are optional.
4. A colon (:) to mark the end of function header.
5. Optional documentation string (docstring) to describe what the function does.
6. One or more valid python statements that make up the function body. Statements must have same indentation level (usually 4 spaces).
7. An optional return statement to return a value from the function.

**Example:**

def greet(name):

"""This function greets to

the person passed in as

parameter"""

print("Hello, " + name + ". Good morning!")

**How to call a function in python?**

* Once we have defined a function, we can call it from another function, program or even the Python prompt.
* To call a function we simply type the function name with appropriate parameters.

>>>greet('Paul')

Hello, Paul. Good morning!

**Docstring**

* The first string after the function header is called the docstring and is short for documentation string.
* It is used to explain in brief, what a function does.
* This string is available to us as \_\_doc\_\_ attribute of the function.

>>>print(greet.\_\_doc\_\_)

Thisfunction greets to

the person passed into the

name parameter

**The return statement**

* The return statement is used to exit a function and go back to the place from where it was called.

**Syntax**

return [expression\_list]

* This statement can contain expression which gets evaluated and the value is returned.
* If there is no expression in the statement or the return statement itself is not present inside a function, then the function will return the Noneobject.

For example:

>>>print(greet("May"))

Hello, May. Good morning!

None

**Example :**

defabsolute\_value(num):

"""This function returns the absolute

value of the entered number"""

if num >= 0:

return num

else:

return -num

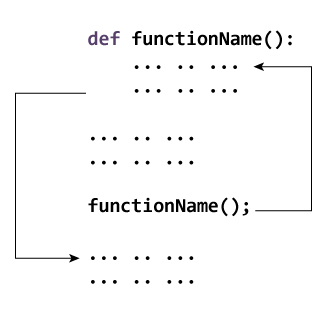
# Output: 2

print(absolute\_value(2))

# Output: 4

print(absolute\_value(-4))

**How Function works in Python?**



**Scope and Lifetime of variables**

* Scope of a variable is the portion of a program where the variable is recognized.
* Parameters and variables defined inside a function is not visible from outside. Hence, they have a local scope.
* Lifetime of a variable is the period throughout which the variable exits in the memory.
* The lifetime of variables inside a function is as long as the function executes.
* They are destroyed once we return from the function. Hence, a function does not remember the value of a variable from its previous calls.

**Example**

defmy\_func():

x = 10

print("Value inside function:",x)

x = 20

my\_func()

print("Value outside function:",x)

**Python Default Arguments**

* Function arguments can have default values in Python.
* We can provide a default value to an argument by using the assignment operator (=).

Example.

def greet(name, msg = "Good morning!"):

"""

This function greets to the person with theprovided message.

If message is not provided, it defaults to "Good morning!"

"""

print("Hello",name + ', ' + msg)

greet("Kate")

greet("Bruce","How do you do?")

## Python Recursive Function

* A [function](https://www.programiz.com/python-programming/function) can call other functions. It is even possible for the function to call itself. These type of construct are termed as recursive functions.

**Example:**

* + Factorial of a number for 6!=1\*2\*3\*4\*5\*6 = 720.

defcalc\_factorial(x):

"""This is a recursive function

to find the factorial of an integer"""

if x == 1:

return 1

else:

return (x \* calc\_factorial(x-1))

num = 4

print("The factorial of", num, "is", calc\_factorial(num))

steps :

calc\_factorial(4)# 1st call with 4

4\*calc\_factorial(3)# 2nd call with 3

4\*3\*calc\_factorial(2)# 3rd call with 2

4\*3\*2\*calc\_factorial(1)# 4th call with 1

4\*3\*2\*1# return from 4th call as number=1

4\*3\*2# return from 3rd call

4\*6# return from 2nd call

24# return from 1st call

**Advantages of Recursion**

1. Recursive functions make the code look clean and elegant.
2. A complex task can be broken down into simpler sub-problems using recursion.
3. Sequence generation is easier with recursion than using some nested iteration.

**Disadvantages of Recursion**

1. Sometimes the logic behind recursion is hard to follow through.
2. Recursive calls are expensive (inefficient) as they take up a lot of memory and time.
3. Recursive functions are hard to debug.

**8. Python Data types**

**Number Data Type**

* Python supports integers, floating point numbers and complex numbers.
* They are defined as int, float and complex class in Python.
* Integers and floating points are separated by the presence or absence of a decimal point. 5 is integer whereas 5.0 is a floating point number.
* Complex numbers are written in the form, x + yj, where x is the real part and y is the imaginary part.

**type() function**

* We can use the type() function to know which class a variable or a value belongs to .

**isinstance() function**

* We can use the  isinstance() function to check if it belongs to a class.

a = 5

# Output: <class 'int'>

print(type(a))

# Output: <class 'float'>

print(type(5.0))

# Output: (8+3j)

c = 5 + 3j

print(c + 3)

# Output: True

print(isinstance(c, complex))

**Number system**

* In Python, we can represent these numbers by appropriately placing a prefix before that number. Following table lists these prefix.

|  |  |
| --- | --- |
| Number System | Prefix |
| Binary | '0b' or '0B' |
| Octal | '0o' or '0O' |
| Hexadecimal | '0x' or '0X' |

Example :

# Output: 107

print(0b1101011)

# Output: 253 (251 + 2)

print(0xFB + 0b10)

# Output: 13

print(0o15)

**Type Conversion**

* We can convert one type of number into another. This is also known as type conversion .
* Operations like addition, subtraction integer to float implicitly (automatically), if one of the operand is float.

>>>1+2.0

3.0

**built-in functions**

* We can also use built-in functions like int(), float() and complex() to convert between types explicitly.
* These function can even convert from [strings](https://www.programiz.com/python-programming/string).

>>>int(2.3)

2

>>>int(-2.8)

-2

>>>float(5)

5.0

>>>complex('3+5j')

(3+5j)

**Python Mathematics**

* Python offers modules like math and random to carry out different mathematics like trigonometry, logarithms, probability and statistics, etc.

Example

import math

# Output: 3.141592653589793

print(math.pi)

# Output: -1.0

print(math.cos(math.pi))

# Output: 22026.465794806718

print(math.exp(10))

# Output: 3.0

print(math.log10(1000))

# Output: 1.1752011936438014

print(math.sinh(1))

# Output: 720

print(math.factorial(6))

example for random module

import random

# Output: random number

print(random.randrange(10,20))

x = ['a', 'b', 'c', 'd', 'e']

# Get random choice

print(random.choice(x))

# Shuffle x

random.shuffle(x)

# Print the shuffled x

print(x)

# Print random element

print(random.random())

**List datatype**

**How to create a list?**

* + In Python programming, a list is created by placing all the items (elements) inside a square bracket [ ], separated by commas.
  + It can have any number of items and they may be of different types (integer, float, string etc.).

# empty list

my\_list = []

# list of integers

my\_list = [1, 2, 3]

# list with mixed datatypes

my\_list = [1, "Hello", 3.4]

* Also, a list can even have another list as an item. This is called nested list.

# nested list

my\_list = ["mouse", [8, 4, 6], ['a']]

**How to access elements from a list?**

* + There are various ways in which we can access the elements of a list.

**List Index**

my\_list = ['p','r','o','b','e']

# Output: p

print(my\_list[0])

# Output: o

print(my\_list[2])

# Output: e

print(my\_list[4])

# Error! Only integer can be used for indexing

# my\_list[4.0]

# Nested List

n\_list = ["Happy", [2,0,1,5]]

# Nested indexing

# Output: a

print(n\_list[0][1])

# Output: 5

print(n\_list[1][3])

**Negative indexing**

* + 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.

Example :

my\_list = ['p','r','o','b','e']

# Output: e

print(my\_list[-1])

# Output: p

print(my\_list[-5])

**How to slice lists in Python?**

* We can access a range of items in a list by using the slicing operator (colon).

Example :

my\_list = ['p','r','o','g','r','a','m','i','z']

# elements 3rd to 5th

print(my\_list[2:5])

# elements beginning to 4th

print(my\_list[:-5])

# elements 6th to end

print(my\_list[5:])

# elements beginning to end

print(my\_list[:])

## How to change or add elements to a list?

* + List are mutable, meaning, their elements can be changed unlike [string](https://www.programiz.com/python-programming/string) or [tuple](https://www.programiz.com/python-programming/tuple).
  + We can use assignment operator (=) to change an item

# mistake values

odd = [2, 4, 6, 8]

# change the 1st item

odd[0] = 1

# Output: [1, 4, 6, 8]

print(odd)

# change 2nd to 4th items

odd[1:4] = [3, 5, 7]

# Output: [1, 3, 5, 7]

print(odd)

* We can add one item to a list using append() method or add several items using extend() method.

odd=[1,3,5]

odd.append(7)

print(odd)

odd.extend([9,11,13])

print(odd)

* We can also use + operator to combine two lists. This is also called concatenation.
* The \* operator repeats a list for the given number of times.

odd=[1,3,5]

# Output: [1, 3, 5, 9, 7, 5]

print(odd +[9,7,5])

#Output: ["re", "re", "re"]

print(["re"]\*3)

* we can insert one item at a desired location by using the method insert() or insert multiple items by squeezing it into an empty slice of a list.

odd = [1, 9]

odd.insert(1,3)

# Output: [1, 3, 9]

print(odd)

odd[2:2] = [5, 7]

# Output: [1, 3, 5, 7, 9]

print(odd)

**How to delete or remove elements from a list?**

* We can delete one or more items from a list using the keyword del.
* It can even delete the list entirely.

Example :

my\_list = ['p','r','o','b','l','e','m']

# delete one item

delmy\_list[2]

# Output: ['p', 'r', 'b', 'l', 'e', 'm']

print(my\_list)

# delete multiple items

delmy\_list[1:5]

# Output: ['p', 'm']

print(my\_list)

# delete entire list

delmy\_list

# Error: List not defined

print(my\_list)

* We can use remove() method to remove the given item or pop() method to remove an item at the given index.
* The pop() method removes and returns the last item if index is not provided. This helps us implement lists as stacks (first in, last out data structure).
* We can also use the clear() method to empty a list.

my\_list = ['p','r','o','b','l','e','m']

my\_list.remove('p')

# Output: ['r', 'o', 'b', 'l', 'e', 'm']

print(my\_list)

# Output: 'o'

print(my\_list.pop(1))

# Output: ['r', 'b', 'l', 'e', 'm']

print(my\_list)

# Output: 'm'

print(my\_list.pop())

# Output: ['r', 'b', 'l', 'e']

print(my\_list)

my\_list.clear()

# Output: []

print(my\_list)

|  |
| --- |
| [Python List Methods](https://www.programiz.com/python-programming/methods/list) |
| [**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 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) |

## List Comprehension:

* List comprehension is an elegant and concise way to create new list from an existing list in Python.
* List comprehension consists of an expression followed by [for statement](https://www.programiz.com/python-programming/for-loop) inside square brackets.

pow2 = [2 \*\* x for x in range(10)]

# Output: [1, 2, 4, 8, 16, 32, 64, 128, 256, 512]

print(pow2)

This code is equivalent to

pow2 =[]

for x in range(10):

pow2.append(2\*\* x)

* A list comprehension can optionally contain more for or [if statements](https://www.programiz.com/python-programming/if-elif-else).
* An optional if statement can filter out items for the new list. Here are some examples.

>>> pow2 =[2\*\* x for x inrange(10)if x >5]

>>> pow2

[64,128,256,512]

>>>odd=[x for x in range(20)if x %2==1]

>>>odd

[1,3,5,7,9,11,13,15,17,19]

### 

|  |  |
| --- | --- |
| Function | Description |
| [all()](https://www.programiz.com/python-programming/methods/built-in/all) | Return True if all elements of the list are true (or if the list is empty). |
| [any()](https://www.programiz.com/python-programming/methods/built-in/any) | Return True if any element of the list is true. If the list is empty, return False. |
| [enumerate()](https://www.programiz.com/python-programming/methods/built-in/enumerate) | Return an enumerate object. It contains the index and value of all the items of list as a tuple. |
| [len()](https://www.programiz.com/python-programming/methods/built-in/len) | Return the length (the number of items) in the list. |
| [list()](https://www.programiz.com/python-programming/methods/built-in/list) | Convert an iterable (tuple, string, set, dictionary) to a list. |
| [max()](https://www.programiz.com/python-programming/methods/built-in/max) | Return the largest item in the list. |
| [min()](https://www.programiz.com/python-programming/methods/built-in/min) | Return the smallest item in the list |
| [sorted()](https://www.programiz.com/python-programming/methods/built-in/sorted) | Return a new sorted list (does not sort the list itself). |
| [sum()](https://www.programiz.com/python-programming/methods/built-in/sum) | Return the sum of all elements in the list. |

## Tuple Datatype:

## What is tuple?

* In Python programming, a tuple is similar to a [list](https://www.programiz.com/python-programming/list).
* The difference between the two is that we cannot change the elements of a tuple once it is assigned whereas in a list, elements can be changed.

### Advantages of Tuple over List

* Since tuple are immutable, iterating through tuple is faster than with list. So there is a slight performance boost.
* Tuples that contain immutable elements can be used as key for a dictionary. With list, this is not possible.
* If you have data that doesn't change, implementing it as tuple will guarantee that it remains write-protected.

## Creating a Tuple

* A tuple is created by placing all the items (elements) inside a parentheses (), separated by comma.
* The parentheses are optional but is a good practice to write it.

my\_tuple = ()

print(my\_tuple)

my\_tuple = (1, 2, 3)

print(my\_tuple)

my\_tuple = (1, "Hello", 3.4)

print(my\_tuple)

my\_tuple = ("mouse", [8, 4, 6], (1, 2, 3))

print(my\_tuple)

Example for without using parenthesis:

my\_tuple = 3, 4.6, "dog"

print(my\_tuple)

# tuple unpacking is also possible

a, b, c = my\_tuple

print(a)

print(b)

print(c)

**Accessing Elements in a Tuple**

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

**1. Indexing**

* + We can use the index operator [] to access an item in a tuple where the index starts from 0.

my\_tuple = ('p','e','r','m','i','t')

print(my\_tuple[0])

print(my\_tuple[5])

n\_tuple = ("mouse", [8, 4, 6], (1, 2, 3))

print(n\_tuple[0][3])

print(n\_tuple[1][1])

**2. Negative Indexing**

* + 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.

my\_tuple = ('p','e','r','m','i','t')

# Output: 't'

print(my\_tuple[-1])

# Output: 'p'

print(my\_tuple[-6])

**3. Slicing**

* We can access a range of items in a tuple by using the slicing operator - colon ":".

my\_tuple = ('p','r','o','g','r','a','m','i','z')

# elements 2nd to 4th

print(my\_tuple[1:4])

# elements beginning to 2nd

print(my\_tuple[:-7])

# elements 8th to end

print(my\_tuple[7:])

# elements beginning to end

print(my\_tuple[:])

**Changing a Tuple**

* 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 datatype like list, its nested items can be changed.

my\_tuple = (4, 2, 3, [6, 5])

#my\_tuple[1] = 9

# but item of mutable element can be changed

# Output: (4, 2, 3, [9, 5])

my\_tuple[3][0] = 9

print(my\_tuple)

* We can use + operator to combine two tuples. This is also called **concatenation**.
* We can also **repeat**the elements in a tuple for a given number of times using the \* operator.

Example :

# Concatenation

# Output: (1, 2, 3, 4, 5, 6)

print((1, 2, 3) + (4, 5, 6))

# Repeat

# Output: ('Repeat', 'Repeat', 'Repeat')

print(("Repeat",) \* 3)

## Deleting a Tuple

* we cannot change the elements in a tuple. That also means we cannot delete or remove items from a tuple.
* But deleting a tuple entirely is possible using the keyword [del](https://www.programiz.com/python-programming/keyword-list#del).

Example :

my\_tuple = ('p','r','o','g','r','a','m','i','z')

# you will get an error:

#del my\_tuple[3]

delmy\_tuple

my\_tuple

## Python Tuple Methods

|  |  |
| --- | --- |
| Method | Description |
| [count(x)](https://www.programiz.com/python-programming/methods/tuple/count) | Return the number of items that is equal to x |
| [index(x)](https://www.programiz.com/python-programming/methods/tuple/index) | Return index of first item that is equal to x |

**Built-in Functions with Tuple**

|  |  |
| --- | --- |
| Function | Description |
| [all()](https://www.programiz.com/python-programming/methods/built-in/all) | Return True if all elements of the tuple are true (or if the tuple is empty). |
| [any()](https://www.programiz.com/python-programming/methods/built-in/any) | Return True if any element of the tuple is true. If the tuple is empty, return False. |
| [enumerate()](https://www.programiz.com/python-programming/methods/built-in/enumerate) | Return an enumerate object. It contains the index and value of all the items of tuple as pairs. |
| [len()](https://www.programiz.com/python-programming/methods/built-in/len) | Return the length (the number of items) in the tuple. |
| [max()](https://www.programiz.com/python-programming/methods/built-in/max) | Return the largest item in the tuple. |
| [min()](https://www.programiz.com/python-programming/methods/built-in/min) | Return the smallest item in the tuple |
| [sorted()](https://www.programiz.com/python-programming/methods/built-in/sorted) | Take elements in the tuple and return a new sorted list (does not sort the tuple itself). |
| [sum()](https://www.programiz.com/python-programming/methods/built-in/sum) | Retrun the sum of all elements in the tuple. |
| [tuple()](https://www.programiz.com/python-programming/methods/built-in/tuple) | Convert an iterable (list, string, set, dictionary) to a tuple. |

## Set Datatype:

## What is a set in Python?

* A set is an unordered collection of items.
* Every element is unique (no duplicates) and must be immutable (which cannot be changed).
* However, the set itself is mutable. We can add or remove items from it.
* Sets can be used to perform mathematical set operations like union, intersection, symmetric difference etc.

## How to create a set?

* A set is created by placing all the items (elements) inside curly braces {}, separated by comma or by using the built-in function set().

Example :

# set of integers

my\_set = {1, 2, 3}

print(my\_set)

# set of mixed datatypes

my\_set = {1.0, "Hello", (1, 2, 3)}

print(my\_set)

# set do not have duplicates

# Output: {1, 2, 3, 4}

my\_set = {1,2,3,4,3,2}

print(my\_set)

# set cannot have mutable items

# here [3, 4] is a mutable list

# If you uncomment line #12,

# this will cause an error.

# TypeError: unhashable type: 'list'

#my\_set = {1, 2, [3, 4]}

# we can make set from a list

# Output: {1, 2, 3}

my\_set = set([1,2,3,2])

print(my\_set)

## How to change a set in Python?

* Sets are mutable. But since they are unordered, indexing have no meaning.
* We cannot access or change an element of set using indexing or slicing. Set does not support it.
* We can add single element using the add() method and multiple elements using the update() method.

Example :

# initializemy\_set

my\_set = {1,3}

print(my\_set)

# if you uncomment line 9,

# you will get an error

# TypeError: 'set' object does not support indexing

#my\_set[0]

# add an element

# Output: {1, 2, 3}

my\_set.add(2)

print(my\_set)

# add multiple elements

# Output: {1, 2, 3, 4}

my\_set.update([2,3,4])

print(my\_set)

# add list and set

# Output: {1, 2, 3, 4, 5, 6, 8}

my\_set.update([4,5], {1,6,8})

print(my\_set)

**How to remove elements from a set?**

* A particular item can be removed from set using methods, discard() and remove().
* The only difference between the two is that, while using discard() if the item does not exist in the set, it remains unchanged.
* But remove() will raise an error in such condition.

Example:

# initializemy\_set

my\_set = {1, 3, 4, 5, 6}

print(my\_set)

# discard an element

# Output: {1, 3, 5, 6}

my\_set.discard(4)

print(my\_set)

# remove an element

# Output: {1, 3, 5}

my\_set.remove(6)

print(my\_set)

# discard an element

# not present in my\_set

# Output: {1, 3, 5}

my\_set.discard(2)

print(my\_set)

# remove an element

# not present in my\_set

# If you uncomment line 27,

# you will get an error.

* Similarly, we can remove and return an item using the pop() method.
* We can also remove all items from a set using clear().

Example :

# initializemy\_set

# Output: set of unique elements

my\_set = set("HelloWorld")

print(my\_set)

# pop an element

# Output: random element

print(my\_set.pop())

# pop another element

# Output: random element

my\_set.pop()

print(my\_set)

# clearmy\_set

#Output: set()

my\_set.clear()

print(my\_set)

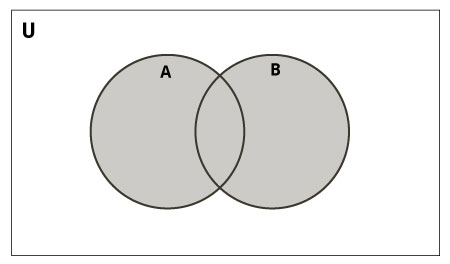
**Python Set Operations**

* Sets can be used to carry out mathematical set operations like union, intersection, difference and symmetric difference.

>>> A = {1, 2, 3, 4, 5}

>>> B = {4, 5, 6, 7, 8}

**Set Union**



* Union of A and B is a set of all elements from both sets.
* Union is performed using | operator. Same can be accomplished using the method union().

Example :

# initialize A and B

A = {1, 2, 3, 4, 5}

B = {4, 5, 6, 7, 8}

# use | operator

# Output: {1, 2, 3, 4, 5, 6, 7, 8}

print(A | B)

Example :

# use union function

>>>A.union(B)

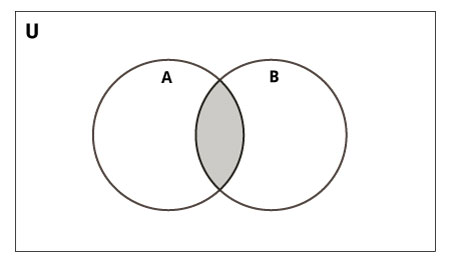
{1,2,3,4,5,6,7,8}

# use union function on B

>>>B.union(A)

{1,2,3,4,5,6,7,8}

**Set Intersection**



* Intersection of A and B is a set of elements that are common in both sets.
* Intersection is performed using & operator. Same can be accomplished using the method intersection().

Example :

# initialize A and B

A = {1, 2, 3, 4, 5}

B = {4, 5, 6, 7, 8}

# use& operator

# Output: {4, 5}

print(A & B)

Examples:

# use intersection function on A

>>>A.intersection(B)

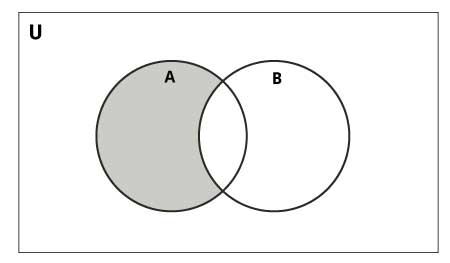
{4,5}

# use intersection function on B

>>>B.intersection(A)

{4,5}

Set Difference



* Difference of A and B (A - B) is a set of elements that are only in A but not in B.
* B - A is a set of element in B but not in A.
* Difference is performed using - operator. Same can be accomplished using the method difference().

Example :

# initialize A and B

A = {1, 2, 3, 4, 5}

B = {4, 5, 6, 7, 8}

# use - operator on A

# Output: {1, 2, 3}

print(A - B)

Example :

>>>A.difference(B)

{1,2,3}

# use - operator on B

>>> B - A

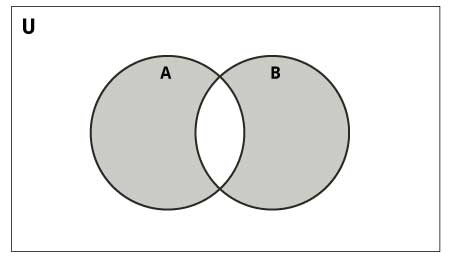
{8,6,7}

# use difference function on B

>>>B.difference(A)

{8,6,7}

**Set Symmetric Difference**



* Symmetric Difference of A and B is a set of elements in both A and Bexceptthose that are common in both.
* Symmetric difference is performed using ^ operator.
* Same can be accomplished using the method symmetric\_difference().

# initialize A and B

A = {1, 2, 3, 4, 5}

B = {4, 5, 6, 7, 8}

# use ^ operator

# Output: {1, 2, 3, 6, 7, 8}

print(A ^ B)

example:

# use symmetric\_difference function on A

>>>A.symmetric\_difference(B)

{1,2,3,6,7,8}

# use symmetric\_difference function on B

>>>B.symmetric\_difference(A)

{1,2,3,6,7,8}

## Different Python Set Methods

|  |  |
| --- | --- |
| Method | Description |
| [add()](https://www.programiz.com/python-programming/methods/set/add) | Add an element to a set |
| [clear()](https://www.programiz.com/python-programming/methods/set/clear) | Remove all elements form a set |
| [copy()](https://www.programiz.com/python-programming/methods/set/copy) | Return a shallow copy of a set |
| [difference()](https://www.programiz.com/python-programming/methods/set/difference) | Return the difference of two or more sets as a new set |
| [difference\_update()](https://www.programiz.com/python-programming/methods/set/difference_update) | Remove all elements of another set from this set |
| [discard()](https://www.programiz.com/python-programming/methods/set/discard) | Remove an element from set if it is a member. (Do nothing if the element is not in set) |
| [intersection()](https://www.programiz.com/python-programming/methods/set/intersection) | Return the intersection of two sets as a new set |
| [intersection\_update()](https://www.programiz.com/python-programming/methods/set/intersection_update) | Update the set with the intersection of itself and another |
| [isdisjoint()](https://www.programiz.com/python-programming/methods/set/isdisjoint) | Return True if two sets have a null intersection |
| [issubset()](https://www.programiz.com/python-programming/methods/set/issubset) | Return True if another set contains this set |
| [issuperset()](https://www.programiz.com/python-programming/methods/set/issuperset) | Return True if this set contains another set |
| [pop()](https://www.programiz.com/python-programming/methods/set/pop) | Remove and return an arbitary set element. Raise KeyError if the set is empty |
| [remove()](https://www.programiz.com/python-programming/methods/set/remove) | Remove an element from a set. If the element is not a member, raise a KeyError |
| [symmetric\_difference()](https://www.programiz.com/python-programming/methods/set/symmetric_difference) | Return the symmetric difference of two sets as a new set |
| [symmetric\_difference\_update()](https://www.programiz.com/python-programming/methods/set/symmetric_difference_update) | Update a set with the symmetric difference of itself and another |
| [union()](https://www.programiz.com/python-programming/methods/set/union) | Return the union of sets in a new set |
| [update()](https://www.programiz.com/python-programming/methods/set/update) | Update a set with the union of itself and others |

### 

### Built-in Functions with Set

|  |  |
| --- | --- |
| Function | Description |
| [all()](https://www.programiz.com/python-programming/methods/built-in/all) | Return True if all elements of the set are true (or if the set is empty). |
| [any()](https://www.programiz.com/python-programming/methods/built-in/any) | Return True if any element of the set is true. If the set is empty, return False. |
| [enumerate()](https://www.programiz.com/python-programming/methods/built-in/enumerate) | Return an enumerate object. It contains the index and value of all the items of set as a pair. |
| [len()](https://www.programiz.com/python-programming/methods/built-in/len) | Return the length (the number of items) in the set. |
| [max()](https://www.programiz.com/python-programming/methods/built-in/max) | Return the largest item in the set. |
| [min()](https://www.programiz.com/python-programming/methods/built-in/min) | Return the smallest item in the set. |
| [sorted()](https://www.programiz.com/python-programming/methods/built-in/sorted) | Return a new sorted list from elements in the set(does not sort the set itself). |
| [sum()](https://www.programiz.com/python-programming/methods/built-in/sum) | Retrun the sum of all elements in the set. |

## Dict Datatype

## What is dictionary in Python?

* Python dictionary is an unordered collection of items.
* While other compound data types have only value as an element, a dictionary has a key: value pair.
* Dictionaries are optimized to retrieve values when the key is known.

## How to create a dictionary?

* Creating a dictionary is as simple as placing items inside curly braces {} separated by comma.
* An item has a key and the corresponding value expressed as a pair, key: value.
* While values can be of any data type and can repeat, keys must be of immutable type and must be unique.

my\_dict={}

# dictionary with integer keys

my\_dict={1:'apple',2:'ball'}

# dictionary with mixed keys

my\_dict={'name':'John',1:[2,4,3]}

# using dict()

my\_dict=dict({1:'apple',2:'ball'})

# from sequence having each item as a pair

my\_dict=dict([(1,'apple'),(2,'ball')])

* we can also create a dictionary using the built-in function dict().
* While indexing is used with other container types to access values, dictionary uses keys.
* Key can be used either inside square brackets or with the get()method.
* The difference while using get() is that it returns None instead of KeyError, if the key is not found.

Example :

my\_dict = {'name':'Jack', 'age': 26}

# Output: Jack

print(my\_dict['name'])

# Output: 26

print(my\_dict.get('age'))

# Trying to access keys which doesn't exist throws error

# my\_dict.get('address')

# my\_dict['address']

**How to change or add elements in a dictionary?**

* Dictionary are mutable. We can add new items or change the value of existing items using assignment operator.
* If the key is already present, value gets updated, else a new key: value pair is added to the dictionary.

Example :

my\_dict = {'name':'Jack', 'age': 26}

# update value

my\_dict['age'] = 27

#Output: {'age': 27, 'name': 'Jack'}

print(my\_dict)

# add item

my\_dict['address'] = 'Downtown'

# Output: {'address': 'Downtown', 'age': 27, 'name': 'Jack'}

print(my\_dict)

**How to delete or remove elements from a dictionary?**

* We can remove a particular item in a dictionary by using the method pop().
* This method removes as item with the provided key and returns the value.
* The method, popitem() can be used to remove and return an arbitrary item (key, value) form the dictionary. All the items can be removed at once using the clear() method.
* We can also use the del keyword to remove individual items or the entire dictionary itself.

Example :

squares = {1:1, 2:4, 3:9, 4:16, 5:25}

# remove a particular item

# Output: 16

print(squares.pop(4))

# Output: {1: 1, 2: 4, 3: 9, 5: 25}

print(squares)

# remove an arbitrary item

# Output: (1, 1)

print(squares.popitem())

# Output: {2: 4, 3: 9, 5: 25}

print(squares)

# delete a particular item

del squares[5]

# Output: {2: 4, 3: 9}

print(squares)

# remove all items

squares.clear()

# Output: {}

print(squares)

# delete the dictionary itself

del squares

## Python Dictionary Methods

|  |  |
| --- | --- |
| Method | Description |
| [clear()](https://www.programiz.com/python-programming/methods/dictionary/clear) | Remove all items form the dictionary. |
| [copy()](https://www.programiz.com/python-programming/methods/dictionary/copy) | Return a shallow copy of the dictionary. |
| [fromkeys(seq[, v])](https://www.programiz.com/python-programming/methods/dictionary/fromkeys) | Return a new dictionary with keys from seq and value equal to v (defaults to None). |
| [get(key[,d])](https://www.programiz.com/python-programming/methods/dictionary/get) | Return the value of key. If key doesnot exit, return d (defaults to None). |
| [items()](https://www.programiz.com/python-programming/methods/dictionary/items) | Return a new view of the dictionary's items (key, value). |
| [keys()](https://www.programiz.com/python-programming/methods/dictionary/keys) | Return a new view of the dictionary's keys. |
| [pop(key[,d])](https://www.programiz.com/python-programming/methods/dictionary/pop) | Remove the item with key and return its value or dif key is not found. If d is not provided and key is not found, raises KeyError. |
| [popitem()](https://www.programiz.com/python-programming/methods/dictionary/popitem) | Remove and return an arbitary item (key, value). Raises KeyError if the dictionary is empty. |
| [setdefault(key[,d])](https://www.programiz.com/python-programming/methods/dictionary/setdefault) | If key is in the dictionary, return its value. If not, insert key with a value of d and return d (defaults to None). |
| [update([other])](https://www.programiz.com/python-programming/methods/dictionary/update) | Update the dictionary with the key/value pairs from other, overwriting existing keys. |
| [values()](https://www.programiz.com/python-programming/methods/dictionary/values) | Return a new view of the dictionary's values |

### Built-in Functions with Dictionary

|  |  |
| --- | --- |
| Function | Description |
| [all()](https://www.programiz.com/python-programming/methods/built-in/all) | Return True if all keys of the dictionary are true (or if the dictionary is empty). |
| [any()](https://www.programiz.com/python-programming/methods/built-in/any) | Return True if any key of the dictionary is true. If the dictionary is empty, return False. |
| [len()](https://www.programiz.com/python-programming/methods/built-in/len) | Return the length (the number of items) in the dictionary. |
| cmp() | Compares items of two dictionaries. |
| [sorted()](https://www.programiz.com/python-programming/methods/built-in/sorted) | Return a new sorted list of keys in the dictionary. |

**String Datatype:**

**Introduction to Strings**

* Python has several built-in functions associated with the [string data type](https://www.digitalocean.com/community/tutorials/an-introduction-to-working-with-strings-in-python-3). These functions let us easily modify and manipulate strings.
* Built-in functions are those that are defined in the Python programming language and are readily available for us to use.

**Making Strings Upper and Lower Case**

* The functions str.upper() and str.lower() will return a string with all the letters of an original string converted to upper- or lower-case letters.
* Because strings are immutable data types, the returned string will be a new string. Any characters in the string that are not letters will not be changed.`
* Let’s convert the string Sammy Shark to be all upper case:

ss = "Sammy Shark"

print(ss.upper())

Ouput

SAMMY SHARK

* Now, let’s convert the string to be all lower case:

print(ss.lower())

Ouput

sammy shark

**Boolean Methods**

* Python has some string methods that will evaluate to a [Boolean value](https://www.digitalocean.com/community/tutorials/understanding-boolean-logic-in-python-3). These methods are useful when we are creating forms for users to fill in, for example.
* There are a number of string methods that will return Boolean values:

| **Method** | **True if** |
| --- | --- |
| str.isalnum() | String consists of only alphanumeric characters (no symbols) |
| str.isalpha() | String consists of only alphabetic characters (no symbols) |
| str.islower() | String’s alphabetic characters are all lower case |
| str.isnumeric() | String consists of only numeric characters |
| str.isspace() | String consists of only whitespace characters |
| str.istitle() | String is in title case |
| str.isupper() | String’s alphabetic characters are all upper case |

* Let’s look at a couple of these in action:

number = "5"

letters = "abcdef"

print(number.isnumeric())

print(letters.isnumeric())

Output

True

False

* Now let’s try the Boolean methods that check for case:

movie = "2001: A SAMMY ODYSSEY"

book = "A Thousand Splendid Sharks"

poem = "sammy lived in a pretty how town"

print(movie.islower())

print(movie.isupper())

print(book.istitle())

print(book.isupper())

print(poem.istitle())

print(poem.islower())

Output

False

True

True

False

False

True

**Determining String Length**

* The string method len() returns the number of characters in a string.
* To demonstrate this method, we’ll find the length of a sentence-long string:

open\_source = "Sammy contributes to open source."

print(len(open\_source))

Output

33

* join(), split(), and replace() Methods

balloon = "Sammy has a balloon."

" ".join(balloon)

print(" ".join(balloon))

* use the str.split()method:

print(balloon.split())

Ouput

['Sammy', 'has', 'a', 'balloon.']

* The str.replace() method can take an original string and return an updated string with some replacement.

print(balloon.replace("has","had"))

Ouput

Sammy had a balloon.

# 9. Python File I/O

**What is a file?**

* File is a named location on disk to store related information. It is used to permanently store data in a non-volatile memory (e.g. hard disk).
* Since, random access memory (RAM) is volatile which loses its data when computer is turned off, we use files for future use of the data.
* When we want to read from or write to a file we need to open it first. When we are done, it needs to be closed, so that resources that are tied with the file are freed.

Hence, in Python, a file operation takes place in the following order.

1. Open a file
2. Read or write (perform operation)
3. Close the file

**Open a file**

* Python has a built-in function open() to open a file. This function returns a file object, also called a handle, as it is used to read or modify the file accordingly.

fo= open("filename with path",mode = 'r',encoding = 'utf-8')

* We can specify the mode while opening a file. In mode, we specify whether we want to read 'r', write 'w' or append 'a' to the file.

|  |  |
| --- | --- |
| Mode | Description |
| 'r' | Open a file for reading. (default) |
| 'w' | Open a file for writing. Creates a new file if it does not exist or truncates the file if it exists. |
| 'a' | Open for appending at the end of the file without truncating it. Creates a new file if it does not exist. |
| 't' | Open in text mode. (default) |
| 'b' | Open in binary mode. |
| '+' | Open a file for updating (reading and writing) |

f = open("test.txt") # equivalent to 'r' or 'rt'

f = open("test.txt",'w') # write in text mode

f = open("img.bmp",'r+b') # read and write in binary mode

* Unlike other languages, the character 'a' does not imply the number 97 until it is encoded using ASCII (or other equivalent encodings).
* Moreover, the default encoding is platform dependent. In windows, it is 'cp1252' but 'utf-8' in Linux.

**Close a File**

* When we are done with operations to the file, we need to properly close the file.
* Closing a file will free up the resources that were tied with the file and is done using Python close() method.

f = open("test.txt",encoding = 'utf-8')

# perform file operations

f.close()

A safer way is to use a [try...finally](https://www.programiz.com/python-programming/exception-handling) block.

try:

f =open("test.txt",encoding='utf-8')

# perform file operations

finally:

f.close()

* The best way to do this is using the with statement. This ensures that the file is closed when the block inside with is exited.
* We don't need to explicitly call the close() method. It is done internally.

with open("test.txt",encoding = 'utf-8') as f:

## How to write to File Using Python?

* In order to write into a file in Python, we need to open it in write 'w', append 'a' .
* We need to be careful with the 'w' mode as it will overwrite into the file if it already exists. All previous data are erased.
* Writing a string or sequence of bytes (for binary files) is done using write() method. This method returns the number of characters written to the file.

with open("test.txt",'w',encoding='utf-8')as f:

f.write("my first file\n")

f.write("This file\n\n")

f.write("contains three lines\n")

## How to read files in Python?

* To read a file in Python, we must open the file in ’r’ reading mode.
* We can use the read(size) method to read in size number of data. If size parameter is not specified, it reads and returns up to the end of the file.

>>> f =open("test.txt",'r',encoding='utf-8')

>>>f.read(4)# read the first 4 data

'This'

>>>f.read(4)# read the next 4 data

' is '

>>>f.read()# read in the rest till end of file

'my first file\nThis file\ncontains three lines\n'

>>>f.read()# further reading returns empty sting

''

* we can use readline() method to read individual lines of a file. This method reads a file till the newline, including the newline character.

>>>f.readline()

'This is my first file\n'

>>>f.readline()

'This file\n'

>>>f.readline()

'contains three lines\n'

>>>f.readline()

''

* The readlines() method returns a list of remaining lines of the entire file. All these reading method return empty values when end of file (EOF) is reached.

>>>f.readlines()

['This is my first file\n','This file\n','contains three lines\n']

**seek() method**

We can change our current file cursor (position) using the seek() method.

fileObject.seek(offset[, whence])

**Parameters**

* **Offset** − This is the position of the read/write pointer within the file.
* **Whence** − This is optional and defaults to 0 which means absolute file positioning, other values are 1 which means seek relative to the current position and 2 means seek relative to the file's end.

Return Value

This method does not return any value.

**tell ( ) Methode:**

* We can use the tell() method returns our current position (in number of bytes).

Position = fileObject.tell( )

Return Value

This method return file pointer location.

>>>f.tell() # get the current file position

56

>>>f.seek(0) # bring file cursor to initial position

0

## 10. Python Module & Packages

## What are modules in Python?

* Modules refer to a file containing Python statements and definitions.
* A file containing Python code, for e.g.: example.py, is called a module and its module name would be example.
* We use modules to break down large programs into small manageable and organized files. Furthermore, modules provide reusability of code.
* We can define our most used functions in a module and import it, instead of copying their definitions into different programs.

Create a module (example.py)

# Python Module example

def add(a, b):

"""This program adds two

numbers and return the result"""

result = a + b

return result

## How to import modules in Python?

* We can import the definitions inside a module to another module or the interactive interpreter in Python.
* We use the import keyword to do this. To import our previously defined module example we type the following in the Python prompt.

Syntax :

>>> import example

* Using the module name we can access the function using dot (.) operation. For example:

>>> example.add(4,5.5)

9.5

* Python has a ton of standard modules available.

### Python import statement

* We can import a module using import statement and access the definitions inside it using the dot operator as described above.

Example.

# import statement example

# to import standard module math

import math

print("The value of pi is", math.pi)

**Import with renaming**

# import module by renaming it

import math as m

print("The value of pi is", m.pi)

**Python from...import statement**

* We can import specific names from a module without importing the module as a whole.

Example

# import only pi from math module

from math import pi

print("The value of pi is", pi)

**Import all names**

# import all names from the standard module math

from math import \*

print("The value of pi is", pi)

**The dir() built-in function**

* We can use the dir() function to find out names that are defined inside a module.

Example

>>> dir(example)

['\_\_builtins\_\_',

'\_\_cached\_\_',

'\_\_doc\_\_',

'\_\_file\_\_',

'\_\_initializing\_\_',

'\_\_loader\_\_',

'\_\_name\_\_',

'\_\_package\_\_',

## What are packages?

* We don't usually store all of our files in our computer in the same location.
* We use a well-organized hierarchy of directories for easier access.
* As our application program grows larger in size with a lot of modules, we place similar modules in one package and different modules in different packages.
* This makes a project (program) easy to manage and conceptually clear.
* Similar, as a directory can contain sub-directories and files, a Python package can have sub-packages and modules.
* A directory must contain a file named \_\_init\_\_.py in order for Python to consider it as a package.
* This file can be left empty but we generally place the initialization code for that package in this file.



**Importing module from a package**

* We can import modules from packages using the dot (.) operator.

Example :

import Game.Level.start

* we can import the module without the package prefix as follows.

from Game.Level import start

**11. Python Exception Handling**

* Generally we get two types of errors in any programming Language there are

1. Syntax errors
2. Runtime errors

**1. Syntax errors:-**

* The errors which occurs because of invalid syntax are known as syntax errors.
* Whenever we run the python program then internally we will check for the syntax . If any syntax is written wrongly byte code will not be generated for the python program
* Without generating the byte code program execution will not takes place.
* Python programmers has to provide the solutions to syntax errors.

While(1<=3)

print “hello”

**2. Runtime errors:-**

* The errors which occurs at the time of execution of the program are known as runtime errors.
* Generally we will get the runtime errors because of program logic, invalid input, memory related problems ,……………
* With respect to runtime errors corresponding classes are available and we call those classes as exception classes. Key error, Index error , Zero division error etc…
* At the time of execution of the python program. If any runtime error is occurred then internally corresponding runtime representation class object will be created.
* If the python program does not contain the code to handle that object then the program will be terminated abnormally.

**Exception handling**

* Exception handling enables you handle errors gracefully and do something meaningful about it. Like display a message to user if intended file not found.
* Python handles exception using try.. except .. block.

Syntax:

try:

# write some code

# that might throw exception

except<ExceptionType>:

# Exception handler, alert the user

* As you can see in try block you need to write code that might throw an exception. When exception occurs code in the try block is skipped.
* If there exist a matching exception type in except  clause then it’s handler is executed.

Let’s take an example:

try:

f = open('somefile.txt', 'r')

print(f.read())

f.close()

except IOError:

print('file not found')

**Note**: The above code is only capable of handling IOError exception. To handle other kind of exception you need to add more except  clause.

* A try  statement can have more than once except  clause, It can also have optional else  and/or finally  statement.

try:

  <body>

except<ExceptionType1>:

  <handler1>

except<ExceptionTypeN>:

  <handlerN>

except:

  <handlerExcept>

else:

  <process\_else>

finally:

  <process\_finally>

* except  clause is similar to elif . When exception occurs, it is checked to match the exception type in except  clause.
* If match is found then handler for the matching case is executed.
* Note that in last except  clause ExceptionType  is omitted. If exception does not match any exception type before the last except  clause, then the handler for last except  clause is executed.

**Note**:  Statements under the else  clause run only when no exception is raised.

**Note**: Statements in finally  block will run every time no matter exception occurs or not.

Now let’s take an example.

try:

   num1, num2 = input("Enter two numbers, separated by a comma : ")

  result = num1 / num2

  print("Result is", result)

exceptZeroDivisionError:

  print("Division by zero is error !!")

exceptSyntaxError:

  print("Comma is missing. Enter numbers separated by comma like this 1, 2")

except:

  print("Wrong input")

else:

  print("No exceptions")

finally:

  print("This will execute no matter what")

**Raising exceptions**

* To raise your exceptions from your own methods you need to use raise  keyword like this

Syntax :-

raise ExceptionClass("Your argument")

example:-

defenterage(age):

   if age < 0:

     raiseValueError("Only positive integers are allowed")

   if age % 2 == 0:

    print("age is even")

  else:

    print("age is odd")

 try:

   num = int(input("Enter your age: "))

   enterage(num)

 exceptValueError:

   print("Only positive integers are allowed")

except:

   print("something is wrong")

## Using Exception objects

* How to access exception object in exception handler code.
* You can use the following code to assign exception object to a variable.

Syntax :-

try:

   # this code is expected to throw exception

except ExceptionType as ex:

   # code to handle exception

Example:-

try:

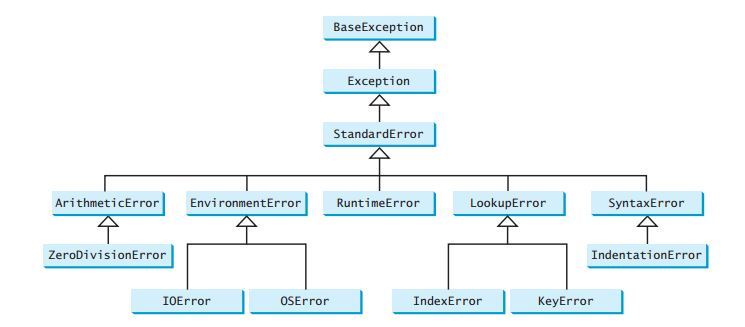
  number = eval(input("Enter a number: "))

  print("The number entered is", number)

exceptNameError as ex:

  print("Exception:", ex)

**Exception types**



**12. Python OOPs Concepts**

* Python is an object-oriented programming language.
* It allows us to develop applications using Object Oriented approach.
* In Python, we can easily create and use classes and objects.

Major principles of object-oriented programming system are given below

* Object
* Class
* Method
* Inheritance
* Polymorphism
* Data Abstraction
* Encapsulation

Object

* Object is an entity that has state and behaviour of class .
* It may be physical and logical.

For example:

mouse, keyboard, chair, table, pen etc.

## Class

* Class can be defined as a collection of objects.
* Class is a blueprint of an object.
* It is a logical entity that has some specific attributes and methods.

For example:

* if you have an employee class then it should contain an attribute and method i.e. an email id, name, age, salary etc.

**Syntax:**

**class** ClassName:

    statement-1>

    . ……………

    . ……………

    . ……………

    statement-N

## Method

* Method is a function that is associated with an object.
* In Python, method is not unique to class instances. Any object type can have methods.

## Inheritance

* Inheritance is a feature of object-oriented programming.
* It specifies that one object acquires all the properties and behaviours of parent object.
* By using inheritance you can define a new class with a little or no changes to the existing class.
* The new class is known as derived class or child class and from which it inherits the properties is called base class or parent class.
* It provides re-usability of the code.

## Polymorphism

* Polymorphism is made by two words "poly" and "morphs". Poly means many and Morphs means form, shape.
* It defines that one task can be performed in different ways.

## Encapsulation

* Encapsulation is also the feature of object-oriented programming.
* It is used to restrict access to methods and variables
* In encapsulation, code and data are wrapped together within a single unit from being modified by accident.

## Data Abstraction

* Data abstraction and encapsulation both are often used as synonyms.
* Both are nearly synonym because data abstraction is achieved through encapsulation.
* Abstraction is used to hide internal details and show only functionalities.

## Python Object Class Example

**class** Student:

**def** \_\_init\_\_(self, rollno, name):

       self.rollno = rollno

       self.name = name

**def** displayStudent(self):

       pr**int** "rollno : ", self.rollno,  ", name: ", self.name

emp1 = Student(121, "Ajeet")

emp2 = Student(122, "Sonoo")

emp1.displayStudent()

emp2.displayStudent()

# Python Constructors

* A constructor is a special type of method (function) which is used to initialize the instance members of the class.
* Constructor can be parameterized and non-parameterized as well.
* Constructor definition executes when we create object of the class.

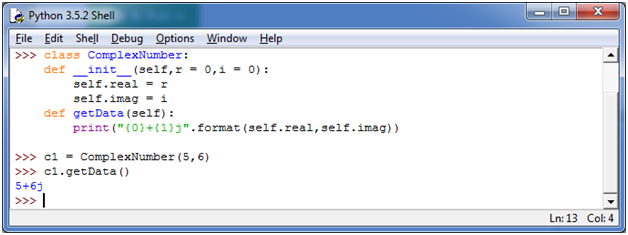
**Creating a Constructor**

* A constructor is a class function that begins with double underscore (\_).
* The name of the constructor is always the same \_\_init\_\_().
* While creating an object, a constructor can accept arguments if necessary.
* When we create a class without a constructor, Python automatically creates a default constructor that doesn't do anything.
* Every class must have a constructor, even if it simply relies on the default constructor.

**Python Constructor Example**

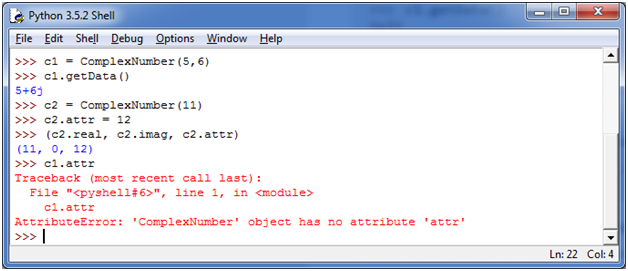
Let's create a class named ComplexNumber, having two functions \_\_init\_\_() function to initialize the variable and getData() to display the number properly.

**See this example:**



You can create a new attribute for an object and read it well at the time of defining the values. But you can't create the attribute for already defined objects.

**See this example:**



**Python Non Parameterized Constructor Example**

**class** Student:

     # Constructor - non parameterized

**def** \_\_init\_\_(self):

**print**("This is non parametrized constructor")

**def** show(self,name):

**print**("Hello",name)

student = Student()

student.show("irfan")

## Python Parameterized Constructor Example

**class** Student:

     # Constructor - parameterized

**def** \_\_init\_\_(self, name):

**print**("This is parametrized constructor")

         self.name = name

**def** show(self):

**print**("Hello",self.name)

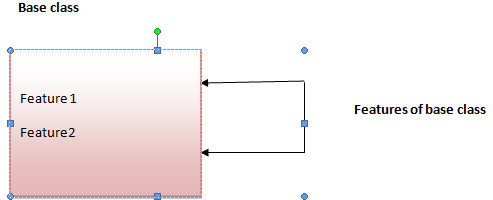
student = Student("irfan")

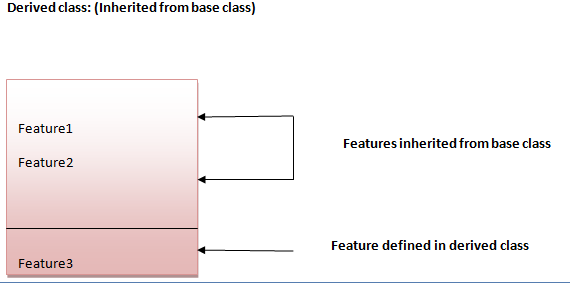
student.show()

# Python Inheritance

* Inheritance is a feature of Object Oriented Programming.
* It is used to specify that one class will get most or all of its features from its parent class.
* It is a very powerful feature which facilitates users to create a new class with a few or more modification to an existing class.
* The new class is called child class or derived class and the main class from which it inherits the properties is called base class or parent class.
* The child class or derived class inherits the features from the parent class, adding new features to it. It facilitates **re-usability of code**.

**Image representation:**





#### Python Inheritance Syntax

class BaseClassName():

<statement-1>

    .

   .

    .

    <statement-N>

class DerivedClassName(BaseClassName):

    <statement-1>

    .

    .

    .

    <statement-N>

## Python Inheritance Example

**class** Animal:

**def** eat(self):

**print** 'Eating...'

**class** Dog(Animal):

**def** bark(self):

**print** 'Barking...'

d=Dog()

d.eat()

d.bark()

**Output:**

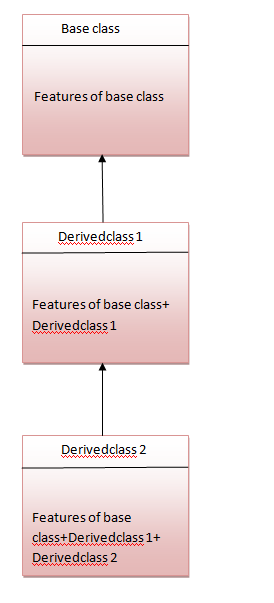
Eating...

Barking...

# Python Multilevel Inheritance

* Multilevel inheritance is also possible in Python like other Object Oriented programming languages.
* We can inherit a derived class from another derived class, this process is known as multilevel inheritance.

**Image representation:**



## Python Multilevel Inheritance Example

**class** Animal:

**def** eat(self):

**print** 'Eating...'

**class** Dog(Animal):

**def** bark(self):

**print** 'Barking...'

**class** BabyDog(Dog):

**def** weep(self):

**print** 'Weeping...'

d=BabyDog()

d.eat()

d.bark()

d.weep()

**Output:**

Eating...

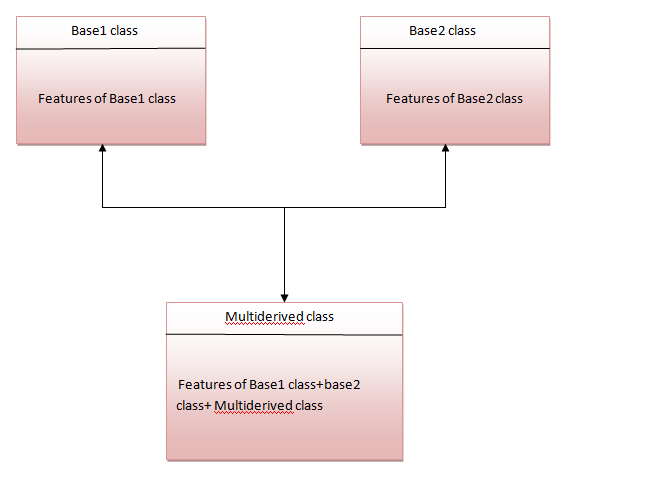
Barking...

Weeping

# Python Multiple Inheritance

* Python supports multiple inheritance too.
* It allows us to inherit multiple parent classes.
* We can derive a child class from more than one base (parent) classes.

**Image Representation**



The multi derived class inherits the properties of both classes base1 and base2.

### Python Multiple Inheritance Syntax

**class** Base1:

**pass**

**class** Base2:

**pass**

**class** MultiDerived(Base1, Base2):

**pass**

### Python Multiple Inheritance Example

**class** First(object):

**def** \_\_init\_\_(self):

     super(First, self).\_\_init\_\_()

**print**("first")

**class** Second(object):

**def** \_\_init\_\_(self):

     super(Second, self).\_\_init\_\_()

**print**("second")

**class** Third(Second, First):

**def** \_\_init\_\_(self):

     super(Third, self).\_\_init\_\_()

**print**("third")

ob=Third();

**Output:**

first

second

third

**13. Python Regular Expression**

* Regular expressions are used to identify whether a pattern exists in a given sequence of characters (string) or not.
* They help in manipulating textual data, which is often a pre-requisite for data science projects that involve text mining.
* They are used at the server side to validate the format of email addresses or password during registration.
* They are used for parsing text data files to find, replace or delete certain string, etc.
* In Python, regular expressions are supported by the  [re](https://docs.python.org/3/library/re.html) module.
* That means that if you want to start using them in your Python scripts, you have to import this module with the help of import

import re

* The match( ) function returns a match object if the text matches the pattern. Otherwise it returns None.

pattern = r"Cookie"

sequence = "Cookie"

if re.match(pattern, sequence):

print("Match!")

else:print("Not a match!")

## Wild Card Characters: Special Characters

* Special characters are characters which do not match themselves as seen but actually have a special meaning when used in a regular expression.
* The group() function returns the string matched by the re.
* The most widely used special characters are:
* . - A period. Matches any single character except newline character.

re.search(r'Co.k.e', 'Cookie').group()

'Cookie'

* \w - Lowercase w. Matches any single letter, digit or underscore.

re.search(r'Co\wk\we', 'Cookie').group()

'Cookie'

* \W - Uppercase w. Matches any character not part of \w (lowercase w).

re.search(r'C\Wke', 'C@ke').group()

'C@ke'

* \s - Lowercase s. Matches a single whitespace character like: space, newline, tab, return.

re.search(r'Eat\scake', 'Eat cake').group()

'Eat cake'

* \S - Uppercase s. Matches any character not part of \s (lowercase s).

re.search(r'Cook\Se', 'Cookie').group()

'Cookie'

* \t - Lowercase t. Matches tab.

re.search(r'Eat\tcake', 'Eat cake').group()

'Eat\tcake'

* \n - Lowercase n. Matches newline.
* \d - Lowercase d. Matches decimal digit 0-9.

re.search(r'c\d\dkie', 'c00kie').group()

'c00kie'

* ^ - Caret. Matches a pattern at the start of the string.

re.search(r'^Eat', 'Eat cake').group()

'Eat'

* $ - Matches a pattern at the end of string.

re.search(r'cake$', 'Eat cake').group()

'cake'

* [abc] - Matches a or b or c.
* [a-zA-Z0-9] - Matches any letter from (a to z) or (A to Z) or (0 to 9).
* \A - Uppercase a. Matches only at the start of the string. Works across multiple lines as well.

re.search(r'\A[A-E]ookie', 'Cookie').group()

'Cookie'

* \b - Lowercase b. Matches only the beginning or end of the word.

re.search(r'\b[A-E]ookie', 'Cookie').group()

'Cookie'

* \ - Backslash. If the character following the backslash is a recognized escape character, then the special meaning of the term is taken. For example, \n is considered as newline.

re.search(r'Back\\stail', 'Back\stail').group()

'Back\\stail'

## Repetitions

It becomes quite tedious if you are looking to find long patterns in a sequence. Fortunately, the  re module handles repetitions using the following special characters:

* + - Checks for one or more characters to its left.

re.search(r'Co+kie', 'Cooookie').group()

'Cooookie'

* \* - Checks for zero or more characters to its left.

re.search(r'Ca\*o\*kie', 'Cookie').group()

'Cookie'

* ? - Checks for exactly zero or one character to its left.

re.search(r'Colou?r', 'Color').group()

'Color'

* For example, checking the validity of a phone number in an application:
* {x} - Repeat exactly x number of times.
* {x,} - Repeat at least x times or more.
* {x, y} - Repeat at least x times but no more than y times.

re.search(r'\d{9,10}', '0987654321').group()

'0987654321'

**Search()**

* It returns a corresponding match object if found, else returns None if no position in the string matches the pattern.
* search(pattern, string, flags=0)

pattern = "cookie"

sequence = "Cake and cookie"

re.search(pattern, sequence).group()

**match()**

* Returns a corresponding match object if zero or more characters at the beginning of string match the pattern.
* Else it returns None, if the string does not match the given pattern.
* match(pattern, string, flags=0)

pattern = "C"

sequence1 = "IceCream"

# No match since "C" is not at the start of "IceCream"

re.match(pattern, sequence1)

sequence2 = "Cake"

re.match(pattern,sequence2).group()

### search() versus match()

* The match() function checks for a match only at the beginning of the string (by default) whereas the search() function checks for a match anywhere in the string.

**14. Python - Multithreaded Programming**

**Thread:-**

* Thread is a functionality / logic which can executed simultaneously along with the other part of the program.
* Running several threads is similar to running several different programs concurrently, but with the following benefits −
  + - * Multiple threads within a process share the same data space with the main thread and can therefore share information or communicate with each other more easily than if they were separate processes.
      * Threads sometimes called light-weight processes and they do not require much memory overhead; they are cheaper than processes.
* A thread has a beginning, an execution sequence, and a conclusion. It has an instruction pointer that keeps track of where within its context it is currently running.
  + - * It can be pre-empted (interrupted)
      * It can temporarily be put on hold (also known as sleeping) while other threads are running - this is called yielding.

## Starting a New Thread

Import thread

thread.start\_new\_thread(function,args)

* 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.

Example:

import thread

import time

# Define a function for the thread

def print\_time(threadName, delay):

count=0

while count <5:

time.sleep(delay)

count+=1

print"%s: %s"%(threadName,time.ctime(time.time()))

# Create two threads as follows

try:

thread.start\_new\_thread(print\_time,("Thread-1",2,)) thread.start\_new\_thread(print\_time,("Thread-2",4,))

except:

print"Error: unable to start 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.

### Example

import threading

import time

exitFlag=0

class myThread(threading.Thread):

def \_\_init\_\_(self,threadID, name, counter):

threading.Thread.\_\_init\_\_(self)

self.threadID=threadID

self.name = name

self.counter= counter

def run(self):

print"Starting "+self.name

print\_time(self.name,5,self.counter)

print"Exiting "+self.name

def print\_time(threadName, counter, delay):

while counter:

if exitFlag:

threadName.exit()

time.sleep(delay)

print"%s: %s"%(threadName,time.ctime(time.time()))

counter-=1

# Create new threads

thread1 =myThread(1,"Thread-1",1)

thread2 =myThread(2,"Thread-2",2)

# Start new Threads

thread1.start()

thread2.start()

print"Exiting Main Thread"

## 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.

### Example

import threading

import time

classmyThread(threading.Thread):

def \_\_init\_\_(self,threadID, name, counter):

threading.Thread.\_\_init\_\_(self)

self.threadID=threadID

self.name = name

self.counter= counter

def run(self):

print"Starting "+self.name

# Get lock to synchronize threads

threadLock.acquire()

print\_time(self.name,self.counter,3)

# Free lock to release next thread

threadLock.release()

defprint\_time(threadName, delay, counter):

while counter:

time.sleep(delay)

print"%s: %s"%(threadName,time.ctime(time.time()))

counter-=1

threadLock=threading.Lock()

threads=[]

# Create new threads

thread1 =myThread(1,"Thread-1",1)

thread2 =myThread(2,"Thread-2",2)

# Start new Threads

thread1.start()

thread2.start()

# Add threads to thread list

threads.append(thread1)

threads.append(thread2)

# Wait for all threads to complete

for t in threads:

t.join()

print"Exiting Main Thread"

15. **Python Networking Programming**

* To connect to end points for communication , the set up we are calling networking.
* Python provides two levels of access to network services.
* At a low level, you can access the basic socket support in the underlying operating system, which allows you to implement clients and servers for both connection-oriented and connectionless protocols.
* Python also has libraries that provide higher-level access to specific application-level network protocols, such as FTP, HTTP, and so on.

**What is Sockets?**

* 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.
* Sockets may be implemented over a number of different channel types: Unix domain sockets, TCP, UDP, and so on.

Sockets have their own vocabulary −

|  |  |
| --- | --- |
| **Sr.No.** | **Term & Description** |
| 1 | **Domain**  The family of protocols that is used as the transport mechanism. These values are constants such as AF\_INET, PF\_INET, PF\_UNIX, PF\_X25, and so on. |
| 2 | **Type**  The type of communications between the two endpoints, typically SOCK\_STREAM for connection-oriented protocols and SOCK\_DGRAM for connectionless protocols. |
| 3 | **Protocol**  Typically zero, this may be used to identify a variant of a protocol within a domain and type. |
| 4 | **Hostname**  The identifier of a network interface −   * A string, which can be a host name, a dotted-quad address, or an IPV6 address in colon (and possibly dot) notation * A string "<broadcast>", which specifies an INADDR\_BROADCAST address. * A zero-length string, which specifies INADDR\_ANY, or * An Integer, interpreted as a binary address in host byte order. |
| 5 | **port**  Each server listens for clients calling on one or more ports. A port may be a Fix num port number. |

**The *socket* Module**

* To create a socket, you must use the *socket.socket()* function available in *socket* module, which has the general syntax −

s = socket.socket (socket\_family, socket\_type, protocol=0)

Here is the description of the parameters −

* **socket\_family** − This is either AF\_UNIX or AF\_INET, as explained earlier.
* **socket\_type** − This is either SOCK\_STREAM or SOCK\_DGRAM.
* **protocol** − This is usually left out, defaulting to 0.
* Once you have *socket* object, then you can use required functions to create your client or server program.

Server Socket Methods

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **s.bind()**  This method binds address (hostname, port number pair) to socket. |
| 2 | **s.listen()**  This method sets up and start TCP listener. |
| 3 | **s.accept()**  This passively accept TCP client connection, waiting until connection arrives (blocking). |

Client Socket Methods

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **s.connect()**  This method actively initiates TCP server connection. |

**General Socket Methods**

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **s.recv()**  This method receives TCP message |
| 2 | **s.send()**  This method transmits TCP message |
| 3 | **s.recvfrom()**  This method receives UDP message |
| 4 | **s.sendto()**  This method transmits UDP message |
| 5 | **s.close()**  This method closes socket |
| 6 | **socket.gethostname()**  Returns the hostname. |

A Simple TCP Server

#!/usr/bin/python # This is server.py file

import socket # Import socket module

s = socket.socket() # Create a socket object

host = socket.gethostname() # Get local machine name

port = 12345 # Reserve a port for your service.

s.bind((host, port)) # Bind to the port

s.listen(5) # Now wait for client connection.

while True:

c, addr = s.accept() # Establish connection with client.

print 'Got connection from', addr

c.send('Thank you for connecting')

c.close() # Close the connection

A Simple Client

host and port, reads any available data from the socket, and then exits −

#!/usr/bin/python # This is client.py file

import socket # Import socket module

s = socket.socket() # Create a socket object

host = socket.gethostname() # Get local machine name

port = 12345 # Reserve a port for your service.

s.connect((host, port))

print s.recv(1024)

s.close # Close the socket when done