Python

**Python Keywords**

* Reserved Words (total key words 33)
* Can’t use for variable, functions and other identifiers
* Case Sensitive and lower case (except True, False and None)

|  |
| --- |
|  |
| 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 |  |

**Identifiers:**

* Names given to entities like class, functions, variables etc.
* 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 (\_). Names like myClass, var\_1 and print\_this\_to\_screen, all are valid example.
* An identifier cannot start with a digit. 1variable is invalid, but variable1 is perfectly fine.
* Keywords cannot be used as identifiers.
* We cannot use special symbols like !, @, #, $, % etc. in our identifier.
* Identifier can be of any length.

## Python Statement

Instructions that a Python interpreter can execute are called statements.

### Multi-line statement

a = 1 + 2 + 3 + \

4 + 5 + 6 + \

7 + 8 + 9

a = (1 + 2 + 3 +

4 + 5 + 6 +

7 + 8 + 9)

colors = ['red',

'blue',

'green']

a = 1; b = 2; c = 3

## Python 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 unindented line.

for i in range(1,11):

print(i)

if i == 5:

break

if True:

print('Hello')

a = 5

if True: print('Hello'); a = 5

## Python Comments

* In Python, we use the hash (#) symbol to start writing a comment.

#This is a comment

#print out Hello

print('Hello')

### Multi-line comments

#This is a long comment

#and it extends

#to multiple lines

"""This is also a

perfect example of

multi-line comments"""

### Docstring in Python

* Docstring is short for documentation string.
* It is a 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 docstring.
* Triple quotes are used while writing docstrings. For example:

Refer: IntellPython\src\GettingStarted\FunctionStatements.py

## Data types in Python

* There are various data types in Python. Some of the important types are listed below.
* Datatypes are defined as int, float and complex class in Python.
* type() and the isinstance() function to check if an object belongs to a particular class
* Mutable
* Refer IntellPython\src\GettingStarted\Datatype1.py

### Python List

* Ordered Sequence.
* Mostly used
* All items can be different data type
* >>> a = [1, 2.2, 'python']
* Refer /IntellPython/src/GettingStarted/DatatypeList.py

### Python Tuple

* Ordered Sequence.
* Immutable
* Usually faster than list as it cannot change dynamically.
* All items can be different data type
* Refer /IntellPython/src/GettingStarted/DatatypeTuple.py

### Python Strings

* We can use single quotes or double quotes to represent strings.
* Multi-line strings can be denoted using triple quotes, ''' or """.
* Refer /IntellPython/src/GettingStarted/DatatypeString.py

### Python Set

* UnOrdered Sequence.
* Set is defined by values separated by comma inside braces { }.
* Refer /IntellPython/src/GettingStarted/DatatypeSet.py

### Python Dictionary

* Un Ordered Sequence.
* Kind of key value pair- map
* Fast to retrieve data when we have huge amount of data.
* Refer /IntellPython/src/GettingStarted/DatatypeDict.py
* Also Refer data conversion
* Refer /IntellPython/src/GettingStarted/DatatypeCoversion.py

## Functions in Python

### Syntax of Function

def function\_name(parameters):

"""docstring"""

statement(s)

## Simple Example

## def absolute\_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))

## Types of Functions

Basically, 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

**Scope and Lifetime of variables**

def my\_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 (=). Here is an example.

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

## """ This function greets to

## the person with the

## provided message.

## If message is not provided,

## it defaults to "Good

## morning!"

## """

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

## greet("Kate")

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

## Keyword Arguement

>>> # 2 keyword arguments

>>> greet(name = "Bruce",msg = "How do you do?")

>>> # 2 keyword arguments (out of order)

>>> greet(msg = "How do you do?",name = "Bruce")

>>> # 1 positional, 1 keyword argument

>>> greet("Bruce",msg = "How do you do?")

## Arbitary Argument

## def greet(\*names):

## """This function greets all

## the person in the names tuple."""

## # names is a tuple with arguments

## for name in names:

## print("Hello",name)

## greet("Monica","Luke","Steve","John")

A lambda function in python has the following syntax.

### Syntax of Lambda Function in python

lambda arguments: expression

# Program to show the use of lambda functions

double = lambda x: x \* 2

# Output: 10

print(double(5))

## Modules - 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.

## Python Package

Python package is a collection of modules in directories that give a package hierarchy. More elaborately, python packages are a way of structuring python’s module by using “dotted module names”. So A.B actually indicates that B is a sub module which is under a package named A.



## Python List

List is a versatile datatype available in Python. Basically a python list is comma-separated values which are called items. List in python is written within square brackets. Interestingly it’s not necessary for items in a list to be of same types

## 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, tuples are quite similiar to lists, both of them are used in similar situations as well.

However, there are certain advantages of implementing a tuple over a list. Below listed are some of the main advantages:

* We generally use tuple for heterogeneous (different) datatypes and list for homogeneous (similar) datatypes.
* 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.
* Also tuples use parentheses while lists use square brackets.

## 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.

A tuple can have any number of items and they may be of different types (integer, float, list, [string](https://www.programiz.com/python-programming/string) etc.).

Creating a tuple with one element is a bit tricky.

Having one element within parentheses is not enough. We will need a trailing comma to indicate that it is in fact a tuple.

## # only parentheses is not enough

## # Output: <class 'str'>

## my\_tuple = ("hello")

## print(type(my\_tuple))

## # need a comma at the end

## # Output: <class 'tuple'>

## my\_tuple = ("hello",)

## print(type(my\_tuple))

## 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.

So, a tuple having 6 elements will have index from 0 to 5. Trying to access an element other that (6, 7,...) will raise an IndexError.

The index must be an integer, so we cannot use float or other types. This will result into TypeError.

### 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.

### 3. Slicing

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

Slicing can be best visualized by considering the index to be between the elements as shown below. So if we want to access a range, we need the index that will slice the portion from the 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.

We can also assign a tuple to different values (reassignment).

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.

Both + and \* operations result into a new tuple.

# 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

As discussed above, 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).

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

# can't delete items

# if you uncomment line 8,

# you will get an error:

# TypeError: 'tuple' object doesn't support item deletion

#del my\_tuple[3]

# can delete entire tuple

# NameError: name 'my\_tuple' is not defined

del my\_tuple

my\_tuple

## Python Tuple Methods

Methods that add items or remove items are not available with tuple. Only the following two methods are available.

**Python Tuple Method**

* [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

## Other Tuple Operations

### 1. Tuple Membership Test

We can test if an item exists in a tuple or not, using the keyword in.

my\_tuple = ('a','p','p','l','e',)

# In operation

# Output: True

print('a' in my\_tuple)

# Output: False

print('b' in my\_tuple)

# Not in operation

# Output: True

print('g' not in my\_tuple)

### 2. Iterating Through a Tuple

Using a for loop we can iterate though each item in a tuple.

# Output:

# Hello John

# Hello Kate

for name in ('John','Kate'):

print("Hello",name)

### 3. Built-in Functions with Tuple

Built-in functions like all(), any(), enumerate(), len(), max(), min(), sorted(), tuple()etc. are commonly used with tuple to perform different tasks.

**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. |

# **Python Strings**

## What is String in Python?

A string is a sequence of characters.

A character is simply a symbol. For example, the English language has 26 characters.

Computers do not deal with characters, they deal with numbers (binary). Even though you may see characters on your screen, internally it is stored and manipulated as a combination of 0's and 1's.

This conversion of character to a number is called encoding, and the reverse process is decoding. ASCII and Unicode are some of the popular encoding used.

In Python, string is a sequence of Unicode character. Unicode was introduced to include every character in all languages and bring uniformity in encoding. You can [learn more about Unicode](http://docs.python.org/3.3/howto/unicode.html) from here.

## How to create a string in Python?

Strings can be created by enclosing characters inside a single quote or double quotes. Even triple quotes can be used in Python but generally used to represent multiline strings and docstrings.

## How to access characters in a string?

We can access individual characters using indexing and a range of characters using slicing. Index starts from 0. Trying to access a character out of index range will raise an IndexError. The index must be an integer. We can't use float or other types, this will result into TypeError.

Python allows negative indexing for its sequences.

The index of -1 refers to the last item, -2 to the second last item and so on. We can access a range of items in a string by using the slicing operator (colon).

If we try to access index out of the range or use decimal number, we will get errors.

# index must be in range

>>> my\_string[15]

...

IndexError: string index out of range

# index must be an integer

>>> my\_string[1.5]

...

TypeError: string indices must be integers

Slicing can be best visualized by considering the index to be between the elements as shown below.

If we want to access a range, we need the index that will slice the portion from the string.



## How to change or delete a string?

Strings are immutable. This means that elements of a string cannot be changed once it has been assigned. We can simply reassign different strings to the same name.

>>> my\_string = 'programiz'

>>> my\_string[5] = 'a'

...

TypeError: 'str' object does not support item assignment

>>> my\_string = 'Python'

>>> my\_string

'Python'

We cannot delete or remove characters from a string. But deleting the string entirely is possible using the keyword del.

>>> del my\_string[1]

...

TypeError: 'str' object doesn't support item deletion

>>> del my\_string

>>> my\_string

...

NameError: name 'my\_string' is not defined

## Python String Operations

There are many operations that can be performed with string which makes it one of the most used [datatypes in Python](https://www.programiz.com/python-programming/variables-datatypes).

### Concatenation of Two or More Strings

Joining of two or more strings into a single one is called concatenation.

The **+** operator does this in Python. Simply writing two string literals together also concatenates them.

The **\*** operator can be used to repeat the string for a given number of times.

## str1 = 'Hello'

## str2 ='World!'

## # using +

## print('str1 + str2 = ', str1 + str2)

## # using \*

## print('str1 \* 3 =', str1 \* 3)

Writing two string literals together also concatenates them like **+** operator.

If we want to concatenate strings in different lines, we can use parentheses.

>>> # two string literals together

>>> 'Hello ''World!'

'Hello World!'

>>> # using parentheses

>>> s = ('Hello '

... 'World')

>>> s

'Hello World'

### Iterating Through String

Using [for loop](https://www.programiz.com/python-programming/for-loop) we can iterate through a string. Here is an example to count the number of 'l' in a string.

## count = 0

## for letter in 'Hello World':

## if(letter == 'l'):

## count += 1

## print(count,'letters found')

### String Membership Test

We can test if a sub string exists within a string or not, using the keyword in.

>>> 'a' in 'program'

True

>>> 'at' not in 'battle'

False

### Built-in functions to Work with Python

Various built-in functions that work with sequence, works with string as well.

Some of the commonly used ones are enumerate() and len(). The enumerate() function returns an enumerate object. It contains the index and value of all the items in the string as pairs. This can be useful for iteration.

Similarly, len() returns the length (number of characters) of the string.

## str = 'cold'

## # enumerate()

## list\_enumerate = list(enumerate(str))

## print('list(enumerate(str) = ', list\_enumerate)

## #character count

## print('len(str) = ', len(str))

## Python String Formatting

### Escape Sequence

If we want to print a text like -He said, "What's there?"- we can neither use single quote or double quotes. This will result into SyntaxError as the text itself contains both single and double quotes.

>>> print("He said, "What's there?"")

...

SyntaxError: invalid syntax

>>> print('He said, "What's there?"')

...

SyntaxError: invalid syntax

One way to get around this problem is to use triple quotes. Alternatively, we can use escape sequences.

An escape sequence starts with a backslash and is interpreted differently. If we use single quote to represent a string, all the single quotes inside the string must be escaped. Similar is the case with double quotes. Here is how it can be done to represent the above text.

Here is a list of all the escape sequence supported by Python.

|  |  |
| --- | --- |
| Escape Sequence in Python | |
| Escape Sequence | Description |
| \newline | Backslash and newline ignored |
| \\ | Backslash |
| \' | Single quote |
| \" | Double quote |
| \a | ASCII Bell |
| \b | ASCII Backspace |
| \f | ASCII Formfeed |
| \n | ASCII Linefeed |
| \r | ASCII Carriage Return |
| \t | ASCII Horizontal Tab |
| \v | ASCII Vertical Tab |
| \ooo | Character with octal value ooo |
| \xHH | Character with hexadecimal value HH |

Here are some examples

>>> print("C:\\Python32\\Lib")

C:\Python32\Lib

>>> print("This is printed\nin two lines")

This is printed

in two lines

>>> print("This is \x48\x45\x58 representation")

This is HEX representation

### Raw String to ignore escape sequence

Sometimes we may wish to ignore the escape sequences inside a string. To do this we can place r or R in front of the string. This will imply that it is a raw string and any escape sequence inside it will be ignored.

>>> print("This is \x61 \ngood example")

This is a

good example

>>> print(r"This is \x61 \ngood example")

This is \x61 \ngood example

### The format() Method for Formatting Strings

The format() method that is available with the string object is very versatile and powerful in formatting strings. Format strings contains curly braces {} as placeholders or replacement fields which gets replaced.

We can use positional arguments or keyword arguments to specify the order.

The format() method can have optional format specifications. They are separated from field name using colon. For example, we can left-justify <, right-justify > or center ^ a string in the given space. We can also format integers as binary, hexadecimal etc. and floats can be rounded or displayed in the exponent format. There are a ton of formatting you can use. Visit here for all the [string formatting available with the format()](https://www.programiz.com/python-programming/methods/string/format) method.

>>> # formatting integers

>>> "Binary representation of {0} is {0:b}".format(12)

'Binary representation of 12 is 1100'

>>> # formatting floats

>>> "Exponent representation: {0:e}".format(1566.345)

'Exponent representation: 1.566345e+03'

>>> # round off

>>> "One third is: {0:.3f}".format(1/3)

'One third is: 0.333'

>>> # string alignment

>>> "|{:<10}|{:^10}|{:>10}|".format('butter','bread','ham')

'|butter | bread | ham|'

### Old style formatting

We can even format strings like the old sprintf() style used in C programming language. We use the % operator to accomplish this.

>>> x = 12.3456789

>>> print('The value of x is %3.2f' %x)

The value of x is 12.35

>>> print('The value of x is %3.4f' %x)

The value of x is 12.3457

## Common Python String Methods

There are numerous methods available with the string object. The format() method that we mentioned above is one of them. Some of the commonly used methods are lower(), upper(), join(), split(), find(), replace() etc. Here is a complete list of all the [built-in methods to work with strings in Python](https://www.programiz.com/python-programming/methods/string).

>>> "PrOgRaMiZ".lower()

'programiz'

>>> "PrOgRaMiZ".upper()

'PROGRAMIZ'

>>> "This will split all words into a list".split()

['This', 'will', 'split', 'all', 'words', 'into', 'a', 'list']

>>> ' '.join(['This', 'will', 'join', 'all', 'words', 'into', 'a', 'string'])

'This will join all words into a string'

>>> 'Happy New Year'.find('ew')

7

>>> 'Happy New Year'.replace('Happy','Brilliant')

'Brilliant New Year'

# **Python Sets**

## 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().

It can have any number of items and they may be of different types (integer, float, tuple, string etc.). But a set cannot have a mutable element, like [list](https://www.programiz.com/python-programming/list), set or [dictionary](https://www.programiz.com/python-programming/dictionary), as its element.

## # 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)

## 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. The update() method can take [tuples](https://www.programiz.com/python-programming/tuple), lists, [strings](https://www.programiz.com/python-programming/string) or other sets as its argument. In all cases, duplicates are avoided.

## 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.

The following example will illustrate this.

## # initialize my\_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.

## # Output: KeyError: 2

## #my\_set.remove(2)

## Python Set Operations

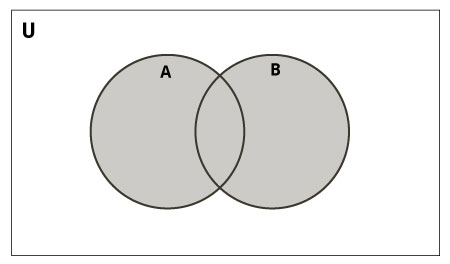
Sets can be used to carry out mathematical set operations like union, intersection, difference and symmetric difference. We can do this with operators or methods.

Let us consider the following two sets for the following operations.

>>> 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().

Try the following examples on Python shell.

# use union function

>>> A.union(B)

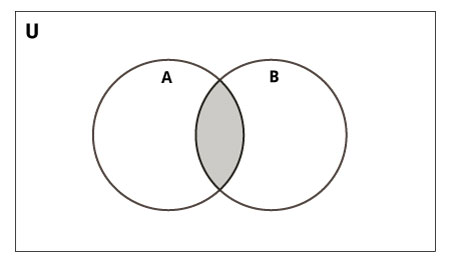
{1, 2, 3, 4, 5, 6, 7, 8}

# 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().

[Powered by DataCamp](https://www.datacamp.com/)

Try the following examples on Python shell.

# use intersection function on A

>>> A.intersection(B)

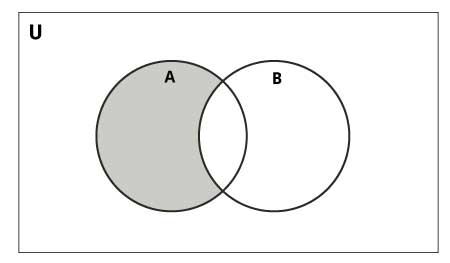
{4, 5}

# 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. Similarly, 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().

Try the following examples on Python shell.

# use difference function on A

>>> A.difference(B)

{1, 2, 3}

# use - operator on B

>>> B - A

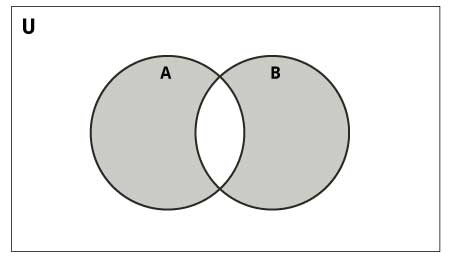
{8, 6, 7}

# 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 B except those that are common in both.

Symmetric difference is performed using ^ operator. Same can be accomplished using the method symmetric\_difference().

## Different Python Set Methods

There are many set methods, some of which we have already used above. Here is a list of all the methods that are available with set objects.

|  |  |
| --- | --- |
| 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 KeyErrorif 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 |

## Other Set Operations

### Set Membership Test

We can test if an item exists in a set or not, using the keyword in.

## # initialize my\_set

## my\_set = set("apple")

## # check if 'a' is present

## # Output: True

## print('a' in my\_set)

## # check if 'p' is present

## # Output: False

## print('p' not in my\_set)

### Iterating Through a Set

Using a for loop, we can iterate though each item in a set.

>>> for letter in set("apple"):

... print(letter)

...

a

p

e

l

### Built-in Functions with Set

Built-in functions like all(), any(), enumerate(), len(), max(), min(), sorted(), sum() etc. are commonly used with set to perform different tasks.

|  |  |
| --- | --- |
| 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. |

## Python Frozenset

Frozenset is a new class that has the characteristics of a set, but its elements cannot be changed once assigned. While tuples are immutable lists, frozensets are immutable sets.

Sets being mutable are unhashable, so they can't be used as dictionary keys. On the other hand, frozensets are hashable and can be used as keys to a dictionary.

Frozensets can be created using the function frozenset().

This datatype supports methods like copy(), difference(), intersection(), isdisjoint(), issubset(), issuperset(), symmetric\_difference() and union(). Being immutable it does not have method that add or remove elements.

## # initialize A and B

## A = frozenset([1, 2, 3, 4])

## B = frozenset([3, 4, 5, 6])

Try these examples on Python shell.

>>> A.isdisjoint(B)

False

>>> A.difference(B)

frozenset({1, 2})

>>> A | B

frozenset({1, 2, 3, 4, 5, 6})

>>> A.add(3)

...

AttributeError: 'frozenset' object has no attribute 'add'

# **Python Arrays**

Arrays are fundamental part of most programming languages. It is the collection of elements of a single data type, eg. array of int, array of string.

However, in Python, there is no native array data structure. So, we use [Python lists](https://www.programiz.com/python-programming/list) instead of an array.

**Note:** If you want to create real arrays in Python, you need to use [NumPy's array](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.array.html) data structure. For mathematical problems, NumPy Array is more efficient.

Unlike arrays, a single list can store elements of any data type and does everything an array does. We can store an integer, a float and a string inside the same list. So, it is more flexible to work with.

[10, 20, 30, 40, 50 ] is an example of what an array would look like in Python, but it is actually a list.

## Create an Array

We can create a Python array with comma separated elements between square brackets[].

#### Example 1: How to create an array in Python?

We can make an integer array and store it to arr.

arr = [10, 20, 30, 40, 50]

## Access elements of an Array

We can access individual elements of an array using index inside square brackets [].

### Array Index

Index is the position of element in an array. In Python, arrays are zero-indexed. This means, the element's position starts with 0 instead of 1.

#### Example 2: Accessing elements of array using indexing

arr = [10, 20, 30, 40, 50]

print(arr[0])

print(arr[1])

print(arr[2])

When we run the above program, the output will be:

10

20

30

Here, the first element of arr is arr[0], second is arr[1], third is arr[2], and so on.

### Negative Indexing

Python programming supports negative indexing of arrays, something that is not available in arrays in most programming languages. This means the index value of -1 gives the last element, and -2 gives the second to last element of an array.

#### Example 3: Accessing elements of array using negative indexing

arr = [10, 20, 30, 40, 50]

print(arr[-1])

print(arr[-2])

When we run the above program, the output will be:

50

40

## Find length of an Array

Python arrays are just lists, so finding the length of an array is equivalent to finding length of a list in Python.

#### Example 4: Find length of an array using len()

brands = ["Coke", "Apple", "Google", "Microsoft", "Toyota"]

num\_brands = len(brands)

print(num\_brands)

When we run the above program, the output will be:

5

As seen from the above example, the len function gives the length of array brands which is 5.

## Add an element to an Array

To add a new element to an array, we use append() method in Python.

#### Example 5: Adding an element in an array using append()

add = ['a', 'b', 'c']

add.append('d')

print(add)

When we run the above program,the output will be

['a', 'b', 'c', 'd']

Here, we used append() method to add 'd'.

## Remove elements from an Array

Python's list implementation of array allows us to delete any elements from an array using del operator.

Similarly, we can also use remove() and pop() methods to remove elements in an array.

#### Example 6: Removing elements of an array using del, remove() and pop()

colors = ["violet", "indigo", "blue", "green", "yellow", "orange", "red"]

del color[4]

colors.remove("blue")

colors.pop(3)

print(color)

When we run the above program, the output will be

['violet', 'indigo', 'green', 'red']

In the above program,

* First, we used del statement to remove element located at index 4, i.e. 'yellow'. Now, the colors array becomes ['violet', 'indigo', 'blue', 'green', 'orange', 'red'].
* Then, we used remove('blue') function to remove element 'blue' from the array. Now the colors array becomes ['violet', 'indigo', 'green', 'orange', 'red'].
* Then, we used pop(3) function to delete element at index 3, i.e. 'orange'. Finally, the colors array becomes ['violet', 'indigo', 'orange', 'red'] as shown in the output.

## Modify elements of an Array

We can change values of elements within an array using indexing and assignment operator (=). We select the position of any element using indexing and use assignment operator to provide a new value for the element.

#### Example 7: Modifying elements of an array using Indexing

fruits = ["Apple", "Banana", "Mango", "Grapes", "Orange"]

fruits[1] = "Pineapple"

fruits[-1] = "Guava"

print(fruits)

When we run the above program, the output will be:

['Apple', 'Pineapple', 'Mango', 'Grapes', 'Guava']

When we print the elements of fruits it shows that Pineapple have replaced Mango at index 1.

We also changed last element of fruits to Guava, using negative indexing.

Thus, we can change and update the elements of array easily.

### Python operators to modify elements in  an Array

In Python arrays, operators like +, \* can also be used to modify elements.

We can use + operator to concatenate (combine) two arrays.

#### Example 8: Concatenating two arrays using + operator

concat = [1, 2, 3]

concat + [4,5,6]

print(concat)

When we run the above program. the output will be:

[1, 2, 3, 4, 5, 6]

Similarly, we can use \* operator to repeat the elements multiple times.

#### Example 8: Repeating elements in array using \* operator

repeat = ["a"]

repeat = repeat \* 5

print(repeat)

When we run the above program, the output will be

['a', 'a', 'a', 'a', 'a']

We repeated string "a" for 5 times, using \* operator.

## Slicing an Array

Python has a slicing feature which allows to access pieces of an array. We, basically, slice an array using a given range (eg. 2nd to 5th position), giving us elements we require. This is done by using indexes separated by a colon [x : y].

We can use negative indexing with slicing too.

#### Example 9: Slicing an array using Indexing

fruits = ["Apple", "Banana", "Mango", "Grapes", "Orange"]

print(fruits[1:4])

print(fruits[ : 3])

print(fruits[-4:])

print(fruits[-3:-1])

When we run the above program, the output will be:

['Banana', 'Mango', 'Grapes']

['Apple', 'Banana', 'Mango']

['Banana', 'Mango', 'Grapes', 'Orange']

['Mango', 'Grapes']

While creating a slice [1:4], slicing starts (inclusive) with left index number, and slicing ends (exclusive) with right index number. This means slicing only prints out the elements of position 1, 2, and 3. Here, position 4 is exclusive so we don't get Orange as an output.

In the code fruits[:3], you can see we didn't include the index on the left. This means, slicing takes all elements until the index on the right (excluding the right index), i.e. first 3 (0, 1, 2) elements of the array.

Likewise, fruits[-4:] prints all elements after second position (1 or -4) i.e 'Banana'. This code is equivalent to fruits[1:].

In the final code, fruits[-3:-1], it all elements starting from index -3 to -2.

## Python Array Methods

Other array operations are also available in Python using list/array methods given as:

|  |  |
| --- | --- |
| Methods | Functions |
| append() | to add element to the end of the list |
| extend() | to extend all elements of a list to the another list |
| insert() | to insert an element at the another index |
| remove() | to remove an element from the list |
| pop() | to remove elements return element at the given index |
| clear() | to remove all elements from the list |
| index() | to return the index of the first matched element |
| count() | to count of number of elements passed as an argument |
| sort() | to sort the elements in ascending order by default |
| reverse() | to reverse order element in a list |
| copy() | to return a copy of elements in a list |

## Multidimensional arrays

All arrays created above are single dimensional. We can also create a multidimensional array in Python. A multidimensional array is an array within an array. This means an array holds different arrays inside it.

#### Example 10: Create a two-dimensional array using lists

multd = [[1,2], [3,4], [5,6], [7,8]]

print(multd[0])

print(multd[3])

print(multd[2][1])

print(multd[3][0])

When we run the above program, the output will be

[1, 2]

[7, 8]

6

7

Here, we have 4 elements and each elements hold another 2 sub-elements.

# **Python Matrix**

## What is a matrix?

A matrix is a two-dimensional data structure. In real-world tasks you often have to store rectangular data table. The table below shows the marks of three students in different subjects.

| S.No | Student Name | Science | English | History | Arts | Maths |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Roy | 80 | 75 | 85 | 90 | 95 |
| 2 | John | 75 | 80 | 75 | 85 | 100 |
| 3 | Dave | 80 | 80 | 80 | 90 | 95 |

Such tables are called matrices or two-dimensional arrays. In python any table can be represented as a list of lists (a list, where each element is in turn a list).

For example:

A = [['Roy',80,75,85,90,95],['John',75,80,75,85,100],['Dave',80,80,80,90,95]]

In the above example A represents a 3\*6 matrix where 3 is number of rows and 6 is number of columns.

## How to create a matrix?

In python, matrix is a nested list. A list is created by placing all the items (elements) inside a square bracket [ ], separated by commas.

#### Example 1: Create a matrix in python

Here's a program that creates a numerical table with 3 rows and 6 columns.

# a is 2-D matrix with integers

a = [['Roy',80,75,85,90,95],

  ['John',75,80,75,85,100],

  ['Dave',80,80,80,90,95]]

#b is a nested list but not a matrix

b= [['Roy',80,75,85,90,95],

  ['John',75,80,75],

  ['Dave',80,80,80,90,95]]

In the above examples a is a matrix as well as nested list where as bis a nested list but not a matrix.

#### Example 2: Create a dynamic matrix using for loop in python

A possible way: you can create a matrix of n\*m elements by first creating a list of nelements (say, of n zeros) and then make each of the elements a link to another one-dimensional list of m elements:

n = 3

m = 4

a = [0] \* n

for i in range(n):

a[i] = [0] \* m

print(a)

When you run the program, the output will be:

[[0 0 0 0],

 [0 0 0 0],

 [0 0 0 0]]

#### Example 3: Create a matrix using numpy in pyhton

Another way to create a matrix is using numpy library.

from numpy import \*

x = range(16)

x = reshape(x,(4,4))

print(x)

When you run the program, the output will be

[[0 1 2 3],

 [4 5 6 7],

 [8 9 10 11],

 [12 13 14 15]]

## How to access elements in a matrix?

There are various ways in which we can access elements of a python matrix.

### List Index

Similar to list we can access elements of a matrix by using square brackets [] after the variable like a[row][col].

#### Example 4: Accessing elements of the matrix in python by using list index

# a is 2-D matrix with integers

a = [['Roy',80,75,85,90,95],

  ['John',75,80,75,85,100],

  ['Dave',80,80,80,90,95]]

print(a[0])

print(a[0][1])

print(a[1][2])

When you run the program, the output will be:

['Roy', 80, 75, 85, 90, 95]

80

80

Here a is a matrix that contains name and marks of the students.

To see all the marks for the student Roy, we have accessed it by a[0] where 0 is 1st row of the matrix. This gives the output ['Roy',80,75,85,90,95].

Similarly, we can view only his marks in Science by accessing the position of it a[0][1]where 0 is 1st row and 1 is the 2nd column of the matrix. This gives the output 80.

If we wanted to view John's marks in English by accessing the position of it a[1][2] where 1 is 2nd row and 2 is the 3rd column of the matrix. This gives the output 80.

### 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 5: Accessing elements of the matrix in python by using negative list index

a = [['Roy',80,75,85,90,95],

  ['John',75,80,75,85,100],

  ['Dave',80,80,80,90,95]]

print(a[-1])

print(a[-1][-2])

print(a[-2][-3])

When you run the above program, the output will be

['Dave', 80, 80, 80, 90, 95]

90

75

Here a is a matrix that contains name and marks of the students.

To see all the marks for the student Dave, we have accessed it by a[-1] where -1 is last row of the matrix. This gives the output ['Dave', 80, 80, 80, 90, 95].

Similarly, we can view only his marks in Arts by accessing the position of it a[-1][-2] where -1 is last row and -2 is the second last column of the matrix. This gives the output 90.

If we wanted to view John's marks in History by accessing the position of it a[-2][-3] where -2 is second last row and -3 is the third last column of the matrix. This gives the output 75.

## Why and how to slice matrix in python?

In the students matrix we store marks for different subjects of three students. Suppose we want to access marks of Science for all 3 students, here we will be using slicing to get the sub elements of the matrix. In python slicing is done using colon(:) with a syntax (start:end:increment) but for matrix we have to do it using numpy library.

We use slicing to get specific sets of sub-elements from it, without any long, drawn out for loops.

#### Example 6: Slicing a matrix in python using colon(:)  and numpy

from numpy import \*

a = array([['Roy',80,75,85,90,95],

  ['John',75,80,75,85,100],

  ['Dave',80,80,80,90,95]])

print(a[:3,[0,1]])

When we run the above program, the output will be

[['Roy',80],

  ['John',75],

['Dave',80]]

Here we have created the matrix a by using array() method from numpy.

Since we have to access Roy's and John's Science marks with their names, we used a[:3,[0,1]] where :3 is for firts three rows and [0,1] is for the first two columns.

## How to change or add elements of a matrix?

In python list are mutable, meaning, their elements can be changed unlike string or tuple.

We can use assignment operator (=) to change an item or a range of items.

#### Example 7: Change elements of a matrix in python

a = [['Roy',80,75,85,90,95],

  ['John',75,80,75,85,100],

  ['Dave',80,80,80,90,95]]

b=a[0]

print(b)

b[1]=75

print(b)

a[2]=['Sam',82,79,88,97,99]

print(a)

a[0][4]=95

print(a)

When we run the above program, the output will be

b=['Roy',80,75,85,90,95]

b=['Roy',75,75,85,90,95]

a= [['Roy',75,75,85,90,95],

  ['John',75,80,75,85,100],

  ['Sam',82,79,88,97,99]]

a=[['Roy',75,75,85,95,95],

  ['John',75,80,75,85,100],

  ['Sam',82,79,88,97,99]]

Here a is a matrix where we have stored name and marks of the students.

We have stored Roy's marks row in the variable b, by using it's row position from the matrix which is 0 so it becomes a[0].

To change Roy's marks in Science, by directly accessing that position where the marks is stored; as the position of that is a[0][1]. So, in b it will be b[1].

We replaced Dave's row with a new student's marks Sam; we directly accessed the row position which is a[2] of Dave's marks and replace it by Sam's marks .

Roy's marks in Arts was entered wrong, we accessed the position a[0][4] where 0 is the first row and 4 is the fifth column of the matrix in which the data is stored and assigned a new value.

### Python operators to add elements in matrix

We can add one row to a matrix using append() method and add a item using insert()method by importing numpy library.

Now we will be adding a new row in the students table or matrix which contains a new student's marks.

#### Example 8: Adding a new row in the matrix in python using append()

from numpy import \*

a = array([['Roy',80,75,85,90,95],

['John',75,80,75,85,100],

['Dave',80,80,80,90,95]])

a= append(a,[['Sam',82,79,88,97,99]],0)

//here 0 is axis that represents the dimensions where 0 stands for row and 1 stands for column

print(a)

When we run the above program, the output will be

[['Roy',80,75,85,90,95],

 ['John',75,80,75,85,100],

 ['Dave',80,80,80,90,95],

 ['Sam',82,79,88,97,99]]

Here we have created the matrix a using array() method from numpy library.

We are using append() method from numpy to add a row in the matrix where a is the matrix, ['Sam',82,79,88,97,99] is the new row and 0 is the axis that represents the row.

#### Example 9: Add a new column in the matrix for economics marks using insert().

from numpy import \*

a = array([['Roy',80,75,85,90,95],

  ['John',75,80,75,85,100],

  ['Dave',80,80,80,90,95]])

a= insert(a,[6],[[73],[80],[85]],axis=1)

//here axis represents the dimensions where 0 stands for row and 1 stands for column

print(a)

When we run the above program, the output will be

[['Roy',80,75,85,90,95,73],

 ['John',80,75,80,75,85,100,80],

 ['Dave',85,80,80,80,90,95,85]]

Here we have created the matrix a using array() method from numpy library.

We are using insert() method from numpy to add a column in the matrix where a is the matrix, [6] is the column where we have to insert the values, [[73],[80],[85]] is the new column and 1 is the axis that represents the column.

### Concatenation of matrix in Python

We can also use + operator to combine two different lists. This is also called concatenation.

#### Example 10: Add a row in the matrix in python using +

a=[['Roy',80,75,85,90,95],

  ['John',75,80,75,85,100],

  ['Dave',80,80,80,90,95]]

a= a+ [['Sam',82,79,88,97,99]]

print(a)

When we run the above program, the output will be

[['Roy',80,75,85,90,95],

 ['John',75,80,75,85,100],

 ['Dave',80,80,80,90,95],

 ['Sam',82,79,88,97,99]]

Here a is a matrix where we have stored name and marks of the students.

We have inserted a new row using + at the end of the matrix.

## How to delete or remove elements from a matrix?

We can delete an entire row of items from a matrix using the method delete from numpylibrary.

#### Example 11: Delete a row of a matrix in python using delete from numpy

from numpy import \*

a = array([['Roy',80,75,85,90,95],

  ['John',75,80,75,85,100],

  ['Dave',80,80,80,90,95]])

a= delete(a,[1],0)

print(a)

When we run the above program, the output will be

[['Roy',80,75,85,90,95],

 ['Dave',80,80,80,90,95]]

Here we have created the matrix a using array() method from numpy library.

We are using delete() method from numpy to delete a row in the matrix where a is the matrix, [1] is the second row and 0 is the axis that represents the row.

### Delete an entire column of a matrix in Python

#### Example 12: Delete columns of a matrix in python using delete from numpy

from numpy import \*

a = array([['Roy',80,75,85,90,95],

  ['John',75,80,75,85,100],

  ['Dave',80,80,80,90,95]])

a= delete(a, s\_[1::2], 1)

print(a)

When we run the above program, the output will be

[['Roy' ,75, 90],

['John', 80, 85],

['Dave', 80, 90]]

Here we have created the matrix a using array() method from numpy library.

We are using delete() method from numpy to delete a column in the matrix where a is the matrix, s\_[1::2] are the columns second, third and fourth to be deleted and 1 is the axis that represents the column.

# **Python List Comprehension**

## List Comprehension vs For Loop in Python

Suppose, we want to separate the letters of the word human and add the letters as items of a list. The first thing that comes in mind would be using [for loop](https://www.programiz.com/python-programming/for-loop).

#### Example 1: Iterating through a string Using for Loop

h\_letters = []

for letter in 'human':

h\_letters.append(letter)

print(h\_letters)

When we run the program, the output will be:

['h','u','m','a','n']

However, Python has an easier way to solve this issue using List Comprehension. List comprehension is an elegant way to define and create lists based on existing lists.

Let’s see how the above program can be written using list comprehensions.

### Example 2: Iterating through a string Using List Comprehension

h\_letters = [ letter for letter in 'human' ]

print( h\_letters)

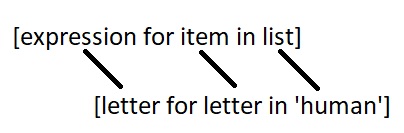
When we run the program, the output will be:

['h', 'u', 'm', 'a', 'n']

In the above example, a new list is assigned to variable h\_letters, and list contains the items of the iterable string 'human'. We call print() function to receive the output.

### Syntax of List Comprehension

[expression for item in list]



We can now identify where list comprehensions are used.

If you noticed, human is a string, not a list. This is the power of list comprehension. It can identify when it receives a string or a tuple and work on it like a [list](https://www.programiz.com/python-programming/list).

You can do that using loops. However, not every loop can be rewritten as list comprehension. But as you learn and get comfortable with list comprehensions, you will find yourself replacing more and more loops with this elegant syntax.

## List Comprehensions vs Lambda functions

List comprehensions aren’t the only way to work on lists. Various built-in functions and [lambda functions](https://www.programiz.com/python-programming/anonymous-function) can create and modify lists in less lines of code.

#### Example 3: Using Lambda functions inside List

h\_letters = list(map(lambda x: x, 'human'))

When we run the program, the output will be

['h','u','m','a','n']

However, list comprehensions are usually more human readable than lambda functions. It is easier to understand what the programmer was trying to accomplish when list comprehensions are used.

## Conditionals in List Comprehension

List comprehensions can utilize conditional statement to modify existing list (or other tuples). We will create list that uses mathematical operators, integers, and [range()](https://www.programiz.com/python-programming/methods/built-in/range).

#### Example 4: Using if with List Comprehension

number\_list = [ x for x in range(20) if x % 2 == 0]

print(number\_list)

When we run the above program, the output will be:

[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

The list ,number\_list, will be populated by the items in range from 0-19 if the item's value is divisible by 2.

#### Example 5: Nested IF with List Comprehension

num\_list = [y for y in range(100) if y % 2 == 0 if y % 5 == 0]

print(num\_list)

When we run the above program, the output will be:

[0, 10, 20, 30, 40, 50, 60, 70, 80, 90]

Here, list comprehension checks:

1. Is y divisible by 2 or not?
2. Is y divisible by 5 or not?

If y satisfies both conditions, y is appended to num\_list.

#### Example 6: if...else With List Comprehension

obj = ["Even" if i%2==0 else "Odd" for i in range(10)]

print(obj)

When we run the above program, the output will be:

['Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd']

Here, list comprehension will check the 10 numbers from 0 to 9. If i is divisible by 2, then Even is appended to the obj list. If not, Odd is appended.

## Nested Loops in List Comprehension

Suppose, we need to compute transpose of a matrix which requires nested for loop. Let’s see how it is done using normal for loop first.

#### Example 7: Transpose of Matrix using Nested Loops

transposed = []

for i in range(2):

transposed\_row = []

for row in matrix:

transposed\_row.append(row[i])

transposed.append(transposed\_row)

print(transposed)

When we run the above program, the output will be:

[[1,3,5,7],[2,4,6,8]]

The above code usestwo for loops to find transpose of a matrix.

We can also perform nested iteration inside a list comprehension. In this section, we will find transpose of a matrix using nested loop inside list comprehension.

#### Example 8: Transpose of a Matrix using List Comprehension

matrix = [[1, 2],[3,4],[5,6],[7,8]]

transpose = [[row[i] for row in matrix] for i in range(2)]

print (transpose)

When we run the above program, the output will be:

[[1,3,5,7],[2,4,6,8]]

In above program, we have a variable matrix which have 4 rows and 2 columns.We need to find transpose of the matrix. For that, we used list comprehension.

**\*\*Note:** The nested loops in list comprehension don’t work like normal nested loops. In the above program, for i in range(2) is executed before row[i] for row in matrix. Hence at first, a value is assigned to i then item directed by row[i] is appended in the transposevariable.

## Key Points to Remember

* List comprehension is an elegant way to define and create lists based on existing lists.
* List comprehension is generally more compact and faster than normal functions and loops for creating list.
* However, we should avoid writing very long list comprehensions in one line to ensure that code is user-friendly.
* Remember, every list comprehension can be rewritten in for loop, but every for loop can’t be rewritten in the form of list comprehension.

# **Python Object Oriented Programming**

## Introduction to OOPs in Python

Python is a multi-paradigm programming language. Meaning, it supports different programming approach.

One of the popular approach to solve a programming problem is by creating objects. This is known as Object-Oriented Programming (OOP).

An object has two characteristics:

* attributes
* behavior

Let's take an example:

Parrot is an object,

* name, age, color are attributes
* singing, dancing are behavior

The concept of OOP in Python focuses on creating reusable code. This concept is also known as DRY (Don't Repeat Yourself).

In Python, the concept of OOP follows some basic principles:

|  |  |
| --- | --- |
| Inheritance | A process of using details from a new class without modifying existing class. |
| Encapsulation | Hiding the private details of a class from other objects. |
| Polymorphism | A concept of using common operation in different ways for different data input. |

## Class

A class is a blueprint for the object.

We can think of class as an sketch of a parrot with labels. It contains all the details about the name, colors, size etc. Based on these descriptions, we can study about the parrot. Here, parrot is an object.

The example for class of parrot can be :

class Parrot:

pass

Here, we use class keyword to define an empty class Parrot. From class, we construct instances. An instance is a specific object created from a particular class.

## Object

An object (instance) is an instantiation of a class. When class is defined, only the description for the object is defined. Therefore, no memory or storage is allocated.

The example for object of parrot class can be:

obj = Parrot()

Here, obj is object of class Parrot.

Suppose we have details of parrot. Now, we are going to show how to build the class and objects of parrot.

#### Example 1: Creating Class and Object in Python

## class Parrot:

## # class attribute

## species = "bird"

## # instance attribute

## def \_\_init\_\_(self, name, age):

## self.name = name

## self.age = age

## # instantiate the Parrot class

## blu = Parrot("Blu", 10)

## woo = Parrot("Woo", 15)

## # access the class attributes

## print("Blu is a {}".format(blu.\_\_class\_\_.species))

## print("Woo is also a {}".format(woo.\_\_class\_\_.species))

## # access the instance attributes

## print("{} is {} years old".format( blu.name, blu.age))

## print("{} is {} years old".format( woo.name, woo.age))

When we run the program, the output will be:

Blu is a bird

Woo is also a bird

Blu is 10 years old

Woo is 15 years old

In the above program, we create a class with name Parrot. Then, we define attributes. The attributes are a characteristic of an object.

Then, we create instances of the Parrot class. Here, blu and woo are references (value) to our new objects.

Then, we access the class attribute using \_\_class \_\_.species. Class attributes are same for all instances of a class. Similarly, we access the instance attributes using blu.name and blu.age. However, instance attributes are different for every instance of a class.

## Methods

Methods are functions defined inside the body of a class. They are used to define the behaviors of an object.

### Example 2 : Creating Methods in Python

## class Parrot:

## 

## # instance attributes

## def \_\_init\_\_(self, name, age):

## self.name = name

## self.age = age

## 

## # instance method

## def sing(self, song):

## return "{} sings {}".format(self.name, song)

## def dance(self):

## return "{} is now dancing".format(self.name)

## # instantiate the object

## blu = Parrot("Blu", 10)

## # call our instance methods

## print(blu.sing("'Happy'"))

## print(blu.dance())

When we run program, the output will be:

Blu sings 'Happy'

Blu is now dancing

In the above program, we define two methods i.e sing() and dance(). These are called instance method because they are called on an instance object i.e blu.

## Inheritance

Inheritance is a way of creating new class for using details of existing class without modifying it. The newly formed class is a derived class (or child class). Similarly, the existing class is a base class (or parent class).

### Example 3: Use of Inheritance in Python

# parent class

class Bird:

def \_\_init\_\_(self):

print("Bird is ready")

def whoisThis(self):

print("Bird")

def swim(self):

print("Swim faster")

# child class

class Penguin(Bird):

def \_\_init\_\_(self):

# call super() function

super().\_\_init\_\_()

print("Penguin is ready")

def whoisThis(self):

print("Penguin")

def run(self):

print("Run faster")

peggy = Penguin()

peggy.whoisThis()

peggy.swim()

peggy.run()

When we run this program, the output will be:

Bird is ready

Penguin is ready

Penguin

Swim faster

Run faster

In the above program, we created two classes i.e. Bird (parent class) and Penguin (child class). The child class inherits the functions of parent class. We can see this from swim()method. Again, the child class modified the behavior of parent class. We can see this from whoisThis() method. Furthermore, we extend the functions of parent class, by creating a new run() method.

Additionally, we use super() function before \_\_init\_\_() method. This is because we want to pull the content of \_\_init\_\_() method from the parent class into the child class.

## Encapsulation

Using OOP in Python, we can restrict access to methods and variables. This prevent data from direct modification which is called encapsulation. In Python, we denote private attribute using underscore as prefix i.e single “ \_ “ or double “ \_\_“.

class Computer:

def \_\_init\_\_(self):

self.\_\_maxprice = 900

def sell(self):

print("Selling Price: {}".format(self.\_\_maxprice))

def setMaxPrice(self, price):

self.\_\_maxprice = price

c = Computer()

c.sell()

# change the price

c.\_\_maxprice = 1000

c.sell()

# using setter function

c.setMaxPrice(1000)

c.sell()

When we run this program, the output will be:

Selling Price: 900

Selling Price: 900

Selling Price: 1000

In the above program, we defined a class Computer. We use \_\_init\_\_() method to store the maximum selling price of computer. We tried to modify the price. However, we can’t change it because Python treats the \_\_maxprice as private attributes. To change the value, we used a setter function i.e setMaxPrice() which takes price as parameter.

## Polymorphism

Polymorphism is an ability (in OOP) to use common interface for multiple form (data types).

Suppose, we need to color a shape, there are multiple shape option (rectangle, square, circle). However we could use same method to color any shape. This concept is called Polymorphism.

### Example 5: Using Polymorphism in Python

class Parrot:

def fly(self):

print("Parrot can fly")

def swim(self):

print("Parrot can't swim")

class Penguin:

def fly(self):

print("Penguin can't fly")

def swim(self):

print("Penguin can swim")

# common interface

def flying\_test(bird):

bird.fly()

#instantiate objects

blu = Parrot()

peggy = Penguin()

# passing the object

flying\_test(blu)

flying\_test(peggy)

When we run above program, the output will be:

Parrot can fly

Penguin can't fly

In the above program, we defined two classes Parrot and Penguin. Each of them have common method fly() method. However, their functions are different. To allow polymorphism, we created common interface i.e flying\_test() function that can take any object. Then, we passed the objects blu and peggy in the flying\_test() function, it ran effectively.

**Key Points to Remember:**

* The programming gets easy and efficient.
* The class is sharable, so codes can be reused.
* The productivity of programmars increases
* Data is safe and secure with data abstraction.

# **Python Objects and Class**

## What are classes and objects in Python?

Python is an object oriented programming language. Unlike procedure oriented programming, where the main emphasis is on functions, object oriented programming stress on objects.

Object is simply a collection of data (variables) and methods (functions) that act on those data. And, class is a blueprint for the object.

We can think of class as a sketch (prototype) of a house. It contains all the details about the floors, doors, windows etc. Based on these descriptions we build the house. House is the object.

As, many houses can be made from a description, we can create many objects from a class. An object is also called an instance of a class and the process of creating this object is called **instantiation**.

## Defining a Class in Python

Like function definitions begin with the keyword [def](https://www.programiz.com/python-programming/keyword-list#def), in Python, we define a class using the keyword [class](https://www.programiz.com/python-programming/keyword-list#class).

The first string is called docstring and has a brief description about the class. Although not mandatory, this is recommended.

Here is a simple class definition.

class MyNewClass:

'''This is a docstring. I have created a new class'''

pass

A class creates a new local [namespace](https://www.programiz.com/python-programming/namespace) where all its attributes are defined. Attributes may be data or functions.

There are also special attributes in it that begins with double underscores (\_\_). For example, \_\_doc\_\_ gives us the docstring of that class.

As soon as we define a class, a new class object is created with the same name. This class object allows us to access the different attributes as well as to instantiate new objects of that class.

## class MyClass:

## "This is my second class"

## a = 10

## def func(self):

## print('Hello')

## # Output: 10

## print(MyClass.a)

## # Output: <function MyClass.func at 0x0000000003079BF8>

## print(MyClass.func)

## # Output: 'This is my second class'

## print(MyClass.\_\_doc\_\_)

When you run the program, the output will be:

10

<function 0x7feaa932eae8="" at="" myclass.func="">

This is my second class

## Creating an Object in Python

We saw that the class object could be used to access different attributes.

It can also be used to create new object instances (instantiation) of that class. The procedure to create an object is similar to a [function](https://www.programiz.com/python-programming/function)call.

>>> ob = MyClass()

This will create a new instance object named ob. We can access attributes of objects using the object name prefix.

Attributes may be data or method. Method of an object are corresponding functions of that class. Any function object that is a class attribute defines a method for objects of that class.

This means to say, since MyClass.func is a function object (attribute of class), ob.func will be a method object.

## Constructors in Python

Class functions that begins with double underscore (\_\_) are called special functions as they have special meaning.

Of one particular interest is the \_\_init\_\_() function. This special function gets called whenever a new object of that class is instantiated.

This type of function is also called constructors in Object Oriented Programming (OOP). We normally use it to initialize all the variables.

## Deleting Attributes and Objects

Any attribute of an object can be deleted anytime, using the del statement. Try the following on the Python shell to see the output.

>>> c1 = ComplexNumber(2,3)

>>> del c1.imag

>>> c1.getData()

Traceback (most recent call last):

...

AttributeError: 'ComplexNumber' object has no attribute 'imag'

>>> del ComplexNumber.getData

>>> c1.getData()

Traceback (most recent call last):

...

AttributeError: 'ComplexNumber' object has no attribute 'getData'

We can even delete the object itself, using the del statement.

>>> c1 = ComplexNumber(1,3)

>>> del c1

>>> c1

Traceback (most recent call last):

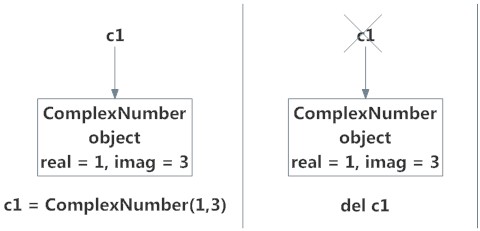
...

NameError: name 'c1' is not defined

Actually, it is more complicated than that. When we do c1 = ComplexNumber(1,3), a new instance object is created in memory and the name c1 binds with it.

On the command del c1, this binding is removed and the name c1 is deleted from the corresponding namespace. The object however continues to exist in memory and if no other name is bound to it, it is later automatically destroyed.

This automatic destruction of unreferenced objects in Python is also called garbage collection.



# **Python Inheritance**

## What is Inheritance?

Inheritance is a powerful feature in object oriented programming.

It refers to defining a new [class](https://www.programiz.com/python-programming/class) with little or no modification to an existing class. The new class is called **derived (or child) class** and the one from which it inherits is called the **base (or parent) class**.

### Python Inheritance Syntax

class BaseClass:

Body of base class

class DerivedClass(BaseClass):

Body of derived class

Derived class inherits features from the base class, adding new features to it. This results into re-usability of code.

### Example of Inheritance in Python

To demonstrate the use of inheritance, let us take an example.

A polygon is a closed figure with 3 or more sides. Say, we have a class called Polygondefined as follows.

class Polygon:

def \_\_init\_\_(self, no\_of\_sides):

self.n = no\_of\_sides

self.sides = [0 for i in range(no\_of\_sides)]

def inputSides(self):

self.sides = [float(input("Enter side "+str(i+1)+" : ")) for i in range(self.n)]

def dispSides(self):

for i in range(self.n):

print("Side",i+1,"is",self.sides[i])

This class has data attributes to store the number of sides, n and magnitude of each side as a list, sides.

Method inputSides() takes in magnitude of each side and similarly, dispSides() will display these properly.

A triangle is a polygon with 3 sides. So, we can created a class called Triangle which inherits from Polygon. This makes all the attributes available in class Polygon readily available in Triangle. We don't need to define them again (code re-usability). Triangle is defined as follows.

class Triangle(Polygon):

def \_\_init\_\_(self):

Polygon.\_\_init\_\_(self,3)

def findArea(self):

a, b, c = self.sides

# calculate the semi-perimeter

s = (a + b + c) / 2

area = (s\*(s-a)\*(s-b)\*(s-c)) \*\* 0.5

print('The area of the triangle is %0.2f' %area)

However, class Triangle has a new method findArea() to find and print the area of the triangle. Here is a sample run.

>>> t = Triangle()

>>> t.inputSides()

Enter side 1 : 3

Enter side 2 : 5

Enter side 3 : 4

>>> t.dispSides()

Side 1 is 3.0

Side 2 is 5.0

Side 3 is 4.0

>>> t.findArea()

The area of the triangle is 6.00

We can see that, even though we did not define methods like inputSides() or dispSides()for class Triangle, we were able to use them.

If an attribute is not found in the class, search continues to the base class. This repeats recursively, if the base class is itself derived from other classes.

## Method Overriding in Python

In the above example, notice that \_\_init\_\_() method was defined in both classes, Triangleas well Polygon. When this happens, the method in the derived class overrides that in the base class. This is to say, \_\_init\_\_() in Triangle gets preference over the same in Polygon.

Generally when overriding a base method, we tend to extend the definition rather than simply replace it. The same is being done by calling the method in base class from the one in derived class (calling Polygon.\_\_init\_\_() from \_\_init\_\_() in Triangle).

A better option would be to use the built-in function super(). So, super().\_\_init\_\_(3) is equivalent to Polygon.\_\_init\_\_(self,3) and is preferred. You can learn more about the [super() function in Python](http://rhettinger.wordpress.com/2011/05/26/super-considered-super/).

Two built-in functions isinstance() and issubclass() are used to check inheritances. Function isinstance() returns True if the object is an instance of the class or other classes derived from it. Each and every class in Python inherits from the base class object.

>>> isinstance(t,Triangle)

True

>>> isinstance(t,Polygon)

True

>>> isinstance(t,int)

False

>>> isinstance(t,object)

True

Similarly, issubclass() is used to check for class inheritance.

>>> issubclass(Polygon,Triangle)

False

>>> issubclass(Triangle,Polygon)

True

>>> issubclass(bool,int)

True

# **Python Operator Overloading**

## What is operator overloading in Python?

[Python operators](https://www.programiz.com/python-programming/operators) work for built-in classes. But same operator behaves differently with different types. For example, the + operator will, perform arithmetic addition on two numbers, merge two lists and concatenate two strings.

This feature in Python, that allows same operator to have different meaning according to the context is called operator overloading.

So what happens when we use them with objects of a user-defined class? Let us consider the following class, which tries to simulate a point in 2-D coordinate system.

## class Point:

## def \_\_init\_\_(self, x = 0, y = 0):

## self.x = x

## self.y = y

Now, run the code and try to add two points in Python shell.

>>> p1 = Point(2,3)

>>> p2 = Point(-1,2)

>>> p1 + p2

Traceback (most recent call last):

...

TypeError: unsupported operand type(s) for +: 'Point' and 'Point'

Whoa! That's a lot of complains. TypeError was raised since Python didn't know how to add two Point objects together.

However, the good news is that we can teach this to Python through operator overloading. But first, let's get a notion about special functions.

## Special Functions in Python

Class functions that begins with double underscore \_\_ are called special functions in Python. This is because, well, they are not ordinary. The \_\_init\_\_() function we defined above, is one of them. It gets called every time we create a new object of that class. There are a ton of [special functions in Python](http://docs.python.org/3/reference/datamodel.html#special-method-names).

Using special functions, we can make our class compatible with built-in functions.

>>> p1 = Point(2,3)

>>> print(p1)

<\_\_main\_\_.Point object at 0x00000000031F8CC0>

That did not print well. But if we define \_\_str\_\_() method in our class, we can control how it gets printed. So, let's add this to our class.

## class Point:

## def \_\_init\_\_(self, x = 0, y = 0):

## self.x = x

## self.y = y

## 

## def \_\_str\_\_(self):

## return "({0},{1})".format(self.x,self.y)

Now let's try the print() function again.

>>> p1 = Point(2,3)

>>> print(p1)

(2,3)

That's better. Turns out, that this same method is invoked when we use the built-in function str() or format().

>>> str(p1)

'(2,3)'

>>> format(p1)

'(2,3)'

So, when you do str(p1) or format(p1), Python is internally doing p1.\_\_str\_\_(). Hence the name, special functions.

Ok, now back to operator overloading.

## Overloading the + Operator in Python

To overload the + sign, we will need to implement \_\_add\_\_() function in the class. With great power comes great responsibility. We can do whatever we like, inside this function. But it is sensible to return a Point object of the coordinate sum.

## class Point:

## def \_\_init\_\_(self, x = 0, y = 0):

## self.x = x

## self.y = y

## 

## def \_\_str\_\_(self):

## return "({0},{1})".format(self.x,self.y)

## 

## def \_\_add\_\_(self,other):

## x = self.x + other.x

## y = self.y + other.y

## return Point(x,y)

Now let's try that addition again.

>>> p1 = Point(2,3)

>>> p2 = Point(-1,2)

>>> print(p1 + p2)

(1,5)

What actually happens is that, when you do p1 + p2, Python will call p1.\_\_add\_\_(p2) which in turn is Point.\_\_add\_\_(p1,p2). Similarly, we can overload other operators as well. The special function that we need to implement is tabulated below.

|  |  |  |
| --- | --- | --- |
| Operator Overloading Special Functions in Python | | |
| Operator | Expression | Internally |
| Addition | p1 + p2 | p1.\_\_add\_\_(p2) |
| Subtraction | p1 - p2 | p1.\_\_sub\_\_(p2) |
| Multiplication | p1 \* p2 | p1.\_\_mul\_\_(p2) |
| Power | p1 \*\* p2 | p1.\_\_pow\_\_(p2) |
| Division | p1 / p2 | p1.\_\_truediv\_\_(p2) |
| Floor Division | p1 // p2 | p1.\_\_floordiv\_\_(p2) |
| Remainder (modulo) | p1 % p2 | p1.\_\_mod\_\_(p2) |
| Bitwise Left Shift | p1 << p2 | p1.\_\_lshift\_\_(p2) |
| Bitwise Right Shift | p1 >> p2 | p1.\_\_rshift\_\_(p2) |
| Bitwise AND | p1 & p2 | p1.\_\_and\_\_(p2) |
| Bitwise OR | p1 | p2 | p1.\_\_or\_\_(p2) |
| Bitwise XOR | p1 ^ p2 | p1.\_\_xor\_\_(p2) |
| Bitwise NOT | ~p1 | p1.\_\_invert\_\_() |

Try these sample runs in Python shell.

>>> Point(1,1) < Point(-2,-3)

True

>>> Point(1,1) < Point(0.5,-0.2)

False

>>> Point(1,1) < Point(1,1)

False

Similarly, the special functions that we need to implement, to overload other comparison operators are tabulated below.

|  |  |  |
| --- | --- | --- |
| Comparision Operator Overloading in Python | | |
| Operator | Expression | Internally |
| Less than | p1 < p2 | p1.\_\_lt\_\_(p2) |
| Less than or equal to | p1 <= p2 | p1.\_\_le\_\_(p2) |
| Equal to | p1 == p2 | p1.\_\_eq\_\_(p2) |
| Not equal to | p1 != p2 | p1.\_\_ne\_\_(p2) |
| Greater than | p1 > p2 | p1.\_\_gt\_\_(p2) |
| Greater than or equal to | p1 >= p2 | p1.\_\_ge\_\_(p2) |

## Exception Handling in Python

### Some Built-in Python Exceptions

List of some built-in python exceptions are given below.

1. **Exception :** This is the base class for all kind of the exceptions. All kind of exceptions should be derived from this class
2. **ArithmeticError :** This is the base class for the exception raised for any arithmetic errors.
3. **EOFError :** This exception raise when input() function read End-of-File without reading any data.
4. **ZeroDivisionError :** This exception raise when the second argument of a division or modulo operation is zero
5. **AssertionError :** This exception raise when an **assert** statement fails.
6. **FloatingPointError :** This exception raise when a floating point operation fails.
7. **KeyError :** This exception raise when a mapping (dictionary) key is not found in the set of existing keys.

### Python try expect

While writing the code, some statements might suspicious for raising an error. Hence, those statements should be surrounded with try-except-else block. For example, we will now raise an exception by our code. The following code will raise IndexError Exception.

name = 'Imtiaz Abedin'

print(name[15])

print('This will not print')

If you try running the code, you will get below exception.

Traceback (most recent call last):

File "/home/imtiaz/ExceptionHandling.py", line 2, in

print(name[15])

IndexError: string index out of range

Because the size of the string type object ‘name’ is less than 15 and we are try to access the index no 15. Have a look, the second print statement is not executed for that exception. So program crashes due to exception. So, in the next code we will handle this exception.

name = 'Imtiaz Abedin'

try:

print(name[15])

except IndexError:

print('IndexError has been found!')

print('This will be printed print.')

### Basic Structure of Python Exception Handling

In the previous section, we demonstrate about how exception raised and how to handle that. In this section we will discuss about the basic coding structure for handling exceptions. Therefore, the basic coding structure for Python Exception Handling is given below.

name = 'Imtiaz Abedin'

try:

# Write the suspicious block of code

print(name[15])

except AssertionError: # Catch a single exception

# This block will be executed if exception A is caught

print('AssertionError')

except (EnvironmentError, SyntaxError, NameError) as E: # catch multiple exception

# This block will be executed if any of the exception B, C or D is caught

print(E)

except :

print('Exception')

# This block will be executed if any other exception other than A, B, C or D is caught

else:

# If no exception is caught, this block will be executed

pass

finally:

# This block will be executed and it is a must!

pass

# this line is not related to the try-except block

print('This will be printed.')

### Python Exception Handling Important Points

For undergoing a professional python project you need to be careful about exceptions. A simple exception can ruin your code. So, you need to handle those exceptions. A few important points about handling exceptions are given below.

1. It is better to surround the suspicious code with try-except.
2. Using one try-except block for one line of suspicious code is better that using one try-except block for a block of suspicious code.
3. It is better to catch specific exception class. Using generalized exception class is not that much useful for handling.

### Raising an Exception

You can raise an existing exception by using **raise** keyword. So, you just simply write **raise** keyword and then the name of the exception. If we modify the previous code, we get

def input\_age(age):

try:

if(int(age)<=18):

raise ZeroDivisionError

except ValueError:

return 'ValueError: Cannot convert into int'

else:

return 'Age is saved successfully'

print(input\_age('23')) # This will execute properly

print(input\_age('18')) # This will not execute properly

## try...finally

The try statement in Python can have an optional finally clause. This clause is executed no matter what, and is generally used to release external resources.

For example, we may be connected to a remote data center through the network or working with a file or working with a Graphical User Interface (GUI).

In all these circumstances, we must clean up the resource once used, whether it was successful or not. These actions (closing a file, GUI or disconnecting from network) are performed in the finally clause to guarantee execution.

Here is an example of [file operations](https://www.programiz.com/python-programming/file-operation) to illustrate this.

try:

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

# perform file operations

finally:

f.close()

This type of construct makes sure the file is closed even if an exception occurs.