## What is Python

## Python Popular Frameworks and Libraries

## Python Data Structures

# Python Features

## Basic Syntax of Python

## Python Identifiers

# Python Variables

# Python Data Types

# Python Keywords

# Python Literals

## Operators

# Python Loops

# Python String

## Python String Formatting

## Python String functions

# Python List

# Python Tuples

## Python Set

## FrozenSets

# Python Dictionary

## 

## 

## 

## 

## 

## 

## 

## 

## 

## What is Python

Python is an interpreted scripting language also.

Python is *easy to learn* yet powerful and versatile scripting language, which makes it attractive for Application Development.

Python supports *multiple programming patterns*, including object-oriented, imperative, and functional or procedural programming styles.

We don't need to use data types to declare variable because it is *dynamically typed* so

Python makes the development and debugging *fast* because there is no compilation step included in Python development, and the edit-test-debug cycle is very fast.

## Python 2 vs. Python 3

A list of differences between Python 2 and Python 3 are given below:

1. Python 2 uses the function raw\_input() to accept the user's input. While Python 3 uses input() function to accept the user input.
2. In Python 2, the implicit string type is ASCII, whereas, in Python 3, the implicit string type is Unicode.
3. Python 3 doesn't contain the xrange() function of Python 2. The xrange() is the variant of range() function The range() returns a list. Ex:- function range(0,3) contains 0, 1, 2.

## Python Basic Syntax

Python uses the indentation to define a block of code. Generally, we can use four whitespaces to define indentation.

## Why learn Python?

* Easy to use and Learn
* Expressive Language
* Interpreted Language
* Object-Oriented Language
* Open Source Language
* Extensible
* Learn Standard Library
* GUI Programming Support
* Integrated
* Embeddable
* Dynamic Memory Allocation
* Wide Range of Libraries and Frameworks

## Python Popular Frameworks and Libraries

* **Web development (Server-side) -** Django Flask, Pyramid, CherryPy
* **GUIs based applications -** Tk, PyGTK, PyQt, PyJs, etc.
* **Machine Learning -** TensorFlow, PyTorch, **Scikit-learn**, Matplotlib, Scipy, etc.
* **Mathematics -** Numpy, Pandas, etc.

## Python print() Function

The **print()** function displays the given object to (screen) or to the text stream file.

1. print(\*objects, sep=' ', end='\n', file=sys.stdout, flush=False)

Let's explain its parameters one by one.

* **objects -** An object is nothing but a statement that to be printed. The \* sign represents that there can be multiple statements.
* **sep -** The **sep** parameter separates the print values. Default values is ' '.
* **end -** The **end** is printed at last in the statement.
* **file -** It must be an object with a write(string) method.
* **flush -** stream or file is forcibly flushed if it is true. By default, its value is false. Flush() method in Python file handling clears the internal buffer of the file.

Using sep and end argument

1. a = 10
2. print("a =", a, sep='dddd', end='\n\n\n') //a =dddd10 //three new line
3. print("a =", a, sep='0', end='$$$$$') //a =010$$$$$

In the first **print()** statement, we use the **sep** and **end** arguments. The given object is printed just after the **sep** values. The value of end parameter printed at the last of given object. As we can see that, the second **print()** function printed the result after the three black lines.

## Taking Input to the User

Python provides the **input()** function which is used to take input from the user.

1. name = input("Enter a name of student:")
2. print("The student name is: ", name)

By default, the **input()** function takes the string input.If we want to take input as an integer number, we need to typecast the **input()** function into an integer.

1. a = **int**(input("Enter first number: "))
2. b = **int**(input("Enter second number: "))
3. print(a+b)

## Python Data Structures

Data structures are referred which can hold some data together or we can say that they are used to store the data in an organized way.

Python provides built-in data structures such as **list, tuple, dictionary, and set**.

Python List

Python list holds the ordered collection of items. We can store a sequence of items in a list. Python list is mutable which means it can be modified after its creation. The items of lists are enclosed within the square bracket [] and separated by the comma.

1. L1 = ["John", 102, "USA"]
2. L2 = [1, 2, 3, 4, 5, 6]
3. print(type(L1)) //<class 'list'>
4. print(type(L2)) //<class 'list'>

### Python Tuple

Python Tuple is used to store the sequence of immutable Python objects. The tuple is similar to lists since the value of the items stored in the list can be changed.

1. tup = ("Apple", "Mango" , "Orange" , "Banana")
2. print(type(tup))

### Python String

Python string is a sequence of characters. It is a collection of the characters surrounded by single quotes, double quotes, or triple quotes.

1. str1 = "Hi Python" # Creating string using **double** quotes
2. print(str1)
3. str1 = 'Hi Python' # Creating string using single quotes
4. print(str1)
5. str1 = '''Hi Python''' # Creating string using triple quotes
6. print(str1)

**Python doesn't support the character data-type. A single character written as 'p' is treated as a string of length 1.**

### Dictionaries

It stores data in the key-value pair format. Each value is stored corresponding to its key.Keys must be a unique and value can be any type such as integer, list, tuple, etc.

It is a mutable type; we can reassign after its creation.

1. employee = {"Name": "John", "Age": 29, "salary":250000,"Company":"GOOGLE"}
2. print(type(employee))
3. print(employee)

The empty curly braces {} are used to create an empty dictionary.

### Python Sets

A Python set is a collection of unordered elements. Each element in set must be unique and immutable. Sets are mutable.

1. mont= {"January", "February", "March", "April", "May", "June", "July"} #Creating Set
2. print(type(mont))

## Python Modules

Python modules are the program files that contain a Python code or functions. There are two types of modules in the Python - **User-define** modules and **built-in** modules.

A module that the user defines, or we can say that our Python code saved with the **.py** extension, is treated as a user-defined module.

Built-in modules are predefined modules of Python. To use the functionality of the modules, we need to import them into our current working program.

## Python Magic Methods

Python magic method is defined as the special method which adds "magic" to a class. It starts and ends with double underscores, for example, **\_init\_** or **\_str\_.**

The built-in classes define many magic methods. The **dir()** function can be used to see the number of magic methods inherited by a class. It has two prefixes and suffix underscores in the method name.

### Python Iterator

### An iterator is simply an object that can be iterated upon. It returns one object at a time. It can be implemented using the two special methods, **\_\_iter\_\_() and \_\_next\_\_()**.

### Python Generators

The Generators are the easiest way of creating Iterators.

A generator function is defined like a normal function, but whenever it needs to generate a value, it does so with the [yield keyword](https://www.geeksforgeeks.org/python-yield-keyword/) rather than return.

If the body of a def contains yield, the function automatically becomes a Python generator function.

**def** fib(limit):

a, b **=** 0, 1 # Initialize first two Fibonacci Numbers

**while** a < limit:# One by one yield next Fibonacci Number

**yield** a

a, b **=** b, a **+** b

**for** i **in** fib(5):

print(i)(## 0 1 1 2 3)

### Python Decorators

These are used to modify the behavior of the function or class. Decorators provide the flexibility to wrap another function to expand the working of wrapped function, without permanently modifying it.

* A function is an instance of the Object type.
* You can store the function in a variable.
* You can pass the function as a parameter to another function.
* You can return the function from a function.
* You can store them in data structures such as hash tables, lists, …

**def** shout(text):

**return** text.upper()

**def** whisper(text):

**return** text.lower()

**def** greet(func):

# storing the function in a variable

greeting **=** func("""Hi,created by a func passed as an argument.""")

print (greeting)

greet(shout)

greet(whisper)

# Python Features

### 1) Easy to Learn and Use

Python is easy to learn as compared to other programming languages.because There is no use of the semicolon or curly-bracket, the indentation defines the code block.

2) Expressive Language

Python can perform complex tasks using a few lines of code. Ex: the hello world program you simply type **print("Hello World")**.

### 3) Interpreted Language

Python is an interpreted language; it means the Python program is executed one line at a time. advantage of being interpreted language, it makes debugging easy and portable.

### 4) Cross-platform Language

Python can run equally on different platforms such as Windows, Linux, UNIX, and Macintosh, etc. So, we can say that Python is a portable language.

5) Free and Open Source

Python is freely available for everyone.The open-source means, "Anyone can download its source code without paying any penny."

### 6) Object-Oriented Language

### Python supports object-oriented language and concepts of classes and objects come

### into existence. It supports inheritance, polymorphism, and encapsulation, etc.

7) Extensible

It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in our Python code. It converts the program into bytecode, and any platform can use that byte code.

### 8) Large Standard Library

It provides a vast range of libraries for various fields such as machine learning, web development, and also for scripting. There are various machine learning libraries, such as Tensor flow, Pandas, Numpy, Keras, and Pytorch, etc. Django, flask, pyramids are the popular framework for Python web development.

### 9) GUI Programming Support

Graphical User Interface is used for the developing Desktop application. PyQT5, Tkinter, Kivy are the libraries which are used for developing web applications.

### 10) Integrated

It can be easily integrated with languages like C, C++, and JAVA, etc. Python runs code line by line like C,C++ Java. It makes it easy to debug the code.

### 11. Embeddable

The code of the other programming language can be used in the Python source code. We can use Python source code in another programming language as well.

12. Dynamic Memory Allocation

In Python, we don't need to specify the data-type of the variable. When we assign some value to the variable, it automatically allocates the memory to the variable at run time.

First Python Program

Python provides us the two ways to run a program:

* Using Interactive interpreter prompt
* Using a script file

## Interactive interpreter prompt

To open the interactive mode, open the terminal (or command prompt) and type python (python3 in case you have Python2 and Python3 both installed on your system).

## Using a script file (Script Mode Programming)

interpreter prompt is best to run the single-line statements of the code., we cannot write the code every-time on the terminal. It is not suitable to write multiple lines of code.

We need to open an editor like a notepad, create a file named and save it with the **.py** extension, which stands for **"Python".**

* We need to type the python keyword, followed by the file name
* **Ex:- python abc.py**

### Pros and Cons of Script Mode

The script mode has few advantages and disadvantages as well.

* We can run multiple lines of code.
* Debugging is easy in script mode.
* It is appropriate for beginners and also for experts.

Let's see the disadvantages of the script mode.

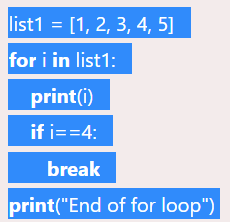
* We have to save the code every time if we make any change in the code.
* It can be tedious when we run a single or a few lines of code.

## Basic Syntax of Python

### Indentation and Comment in Python

Indentation is the most significant concept of the Python programming language. Improper use of indentation will end up **"IndentationError"** in our code.

**Python indentation defines the particular group of statements belonging to the particular block. Like C,** [**C++**](https://www.javatpoint.com/cpp-tutorial)**,** [**java**](https://www.javatpoint.com/java-tutorial) **uses the curly braces {} to define code blocks.**

****

### Types of Comment

**Single-Line Comment -** Single-Line comment starts with the hash # character followed by text for further explanation.

**Multi-Line Comments -** Python doesn't have explicit support for multi-line comments but we can use hash # character to the multiple lines.

## Python Identifiers

Python identifiers refer to a name used to identify a variable, function, module, class, module or other objects. There are few rules to follow while naming the Python Variable.

* A variable name must start with either an English letter or underscore (\_).
* A variable name cannot start with the number.
* Special characters are not allowed in the variable name.
* The variable's name is case sensitive.

# Python Literals

Python literals are a data type and can hold any value type, such as strings, numbers, and more.

# Python Variables

Variable is a name that is used to refer to memory location. A Python variable is also known as an identifier and used to hold value.

In Python, we don't need to specify the type of variable because Python is an inferred language and smart enough to get variable type.

## Identifier Naming

Variables are the example of identifiers. An Identifier is used to identify the literals used in the program. The rules to name an identifier are given below.

* The first character of the variable must be an alphabet or underscore ( \_ ).
* All the characters except the first character may be an alphabet of lower-case(a-z), upper-case (A-Z), underscore, or digit (0-9).
* Identifier name must not contain any white-space, or special character (!, @, #, %, ^, &, \*).
* Identifier names must not be similar to any keyword defined in the language.
* Identifier names are case sensitive; ex-, my name, and MyName is not the same.
* Examples of valid identifiers: a123, \_n, n\_9, etc.
* Examples of invalid identifiers: 1a, n%4, n 9, etc.

## Declaring Variable and Assigning Values

Python does not bind us to declare a variable before using it in the application. It allows us to create a variable at the required time.

We don't need to declare variables explicitly in Python. When we assign any value to the variable, that variable is declared automatically.

## Object Identity

In Python, every created object identifies uniquely in Python. Python provides the guarantee that no two objects will have the same identifier. The built-in **id()** function is used to identify the object identifier.

**The variable b refers to the same object that a points to because Python does not create another object.**

1. a = 50
2. b = a
3. c=50
4. print(id(a)) 140734982691168
5. print(id(b)) 140734982691168
6. print(id(c)) 140734982691168
7. a = 500 //# Reassigned variable a
8. print(id(a)) 2822056960944

We assigned the **b = a, a** and **b** both point to the same object. When we checked by the **id()** function it returned the same number. We reassign **a** to 500; then it referred to the new object identifier.

The multi-word keywords can be created by the following method.

* **Camel Case-** each word or abbreviation in the middle of begins with a capital letter.There is no intervention of whitespace.Ex- nameOfStudent, valueOfVaraible.
* **Pascal Case -** It is the same as the Camel Case, but here the first word is also capital. For example - NameOfStudent, etc.
* **Snake Case -** Words are separated by the underscore. Ex. name\_of\_student, etc.

## Multiple Assignment

Python allows us to assign a value to multiple variables in a single statement, which is also known as multiple assignments.

**1. Assigning a single value to multiple variables. x=y=z=50**

**2. Assigning multiple values to multiple variables: a,b,c=5,10,15**

## Python Variable Types

There are two types of variables in Python - Local variable and Global variable.

### Local Variable

Local variables are the variables that are declared inside the function and have scope within the function.

1. **def** add(): # Declaring a function
2. a, = 20,30 # Defining local variables. They has scope only within a function
3. c = a + b
4. **print**("The sum is:", c)
5. add() # Calling a function

We tried to use local variables outside their scope; it threw the **NameError.**

### Global Variables

Global variables can be used throughout the program, and its scope is in the entire program. We can use global variables inside or outside the function.

A variable declared outside the function is the global variable by default. **Python provides the global keyword to use global variables inside the function. If we don't use the global keyword, the function treats it as a local variable.**

1. x = 101 # Declare a variable and initialize it
2. **def** mainFunction(): # Global variable in function
3. **global** x # printing a global variable
4. **print**(x)
5. x = 'Welcome To Javatpoint' # modifying a global variable
6. **print**(x)
7. mainFunction()
8. **print**(x)

we declare a global variable **x** and assign a value to it. Next, we defined a function and accessed the declared variable using the **global** keyword inside the function. Now we can modify its value. Then, we assigned a new string value to the variable x.

Now, we called the function & proceeded to print **x**. It is printed as a newly assigned value of x.

## Delete a variable

We can delete the variable using the **del** keyword.

1. **del** <variable\_name>

In ex:-, we create a variable x and assign value to it. We deleted variable x, and print it, we get the error **"variable x is not defined"**. variable x will no longer be used in future.

1. x = 6 # Assigning a value to x
2. **print**(x) 6
3. **del** x # deleting a variable.
4. **print**(x) NameError: name 'x' is not defined

## Maximum Possible Value of an Integer in Python

Python doesn't have long int or float data types. It treats all integer values as an **int** type.

There is no limitation number by bits and we can expand to the limit of our memory.Python doesn't have any special data type to store larger numbers.

**There is no limit.**

### Print Single and Multiple Variables in Python

1. a = 5
2. b = 6
3. print(a) # printing single variables
4. **print**(a,b) # printing multiple variables
5. Print(1, 2, 3, 4, 5, 6, 7, 8) # separate the variables by the comma

**a)Tokens:**

* The tokens can be defined as a punctuator mark, reserved words, and each word in a statement.
* The token is the smallest unit inside the given program.

There are following tokens in Python:

* Keywords.
* Identifiers.
* Literals.
* Operators.

# Python Data Types

Python is a dynamically typed language; hence we do not need to define the type of the variable while declaring it. The interpreter implicitly binds the value with its type.

data types defined in Python are given below.

1. [Numbers](https://www.javatpoint.com/python-data-types#numbers)
2. [Sequence Type](https://www.javatpoint.com/python-data-types#SequenceType)
3. [Boolean](https://www.javatpoint.com/python-data-types#Boolean)
4. [Set](https://www.javatpoint.com/python-data-types#Set)
5. [Dictionary](https://www.javatpoint.com/python-data-types#dictionary)



### Numbers

The integer, float, and complex values belong to a Python Numbers data-type. Python creates Number objects when a number is assigned to a variable.the **isinstance()** function is used to check an object belongs to a particular class.

1. a = 5
2. **print**("The type of a", type(a)) #int
3. b = 40.5
4. **print**("The type of b", type(b)) #float
5. c = 1+3j
6. **print**("The type of c", type(c)) #complex
7. **print**(" c is a complex number", isinstance(1+3j,complex)) #true

Python supports three types of numeric data.

1. **Int -** Integer value can be any length such as integers 10, 2, 29, -20, -150 etc. Python has no restriction on the length of an integer. Its value belongs to **int**
2. **Float -** Float is used to store floating-point numbers like 1.9, 9.902, 15.2, etc. It is accurate up to 15 decimal points.
3. **complex -** A complex number contains an ordered pair, i.e., x + iy where x and y denote the real and imaginary parts, respectively. Ex: 2.14j, 2.0 + 2.3j, etc.

### **Sequence Type**

### A)**String** : The string can be defined as the sequence of characters In Python, we can use single, double, or triple quotes to define a string.

operator + is used to concatenate two strings as the operation *"a"+"bc"* returns *"abc"*.

The operator \* is known as a repetition operator as the operation "ab" \*2 returns 'abab'.

1. str1 = 'hello' #string str1
2. str2 = ' how are you' #string str2
3. **print** (str1[0:2]) #he #printing first two character using slice operator
4. **print** (str1[4]) #o #printing 4th character of the string
5. **print** (str1\*2) #hello hello #printing the string twice
6. **print** (str1 + str2) #hello how are you #printing the concatenation of str1 and str2

### **B)List:** Python Lists are similar to arrays in C. list can contain data of different types. items stored in list are separated with a comma & enclosed within brackets [].

We can use slice [:] operators to access the data of the list. operator (+) and repetition operator (\*) works with the list in the same way as they were working with the strings.

1. list1 = [1, "hi", "Python", 2]
2. **print**(type(list1)) #Checking type of given list
3. **print** (list1) #[1, 'hi', 'Python', 2] #Printing the list1
4. **print** (list1[3:]) #[2] # List slicing
5. **print** (list1[0:2]) #[1, 'hi'] # List slicing
6. **print** (list1 + list1) # List Concatenation using + operator
7. **print** (list1 \* 3) # List repetition using \* operator

### **C)Tuple:** A tuple is same as List except A tuple is a read-only data structure as we can't modify the size and value of the items of a tuple.The items of the tuple are separated with a comma (,) and enclosed in parentheses ().

### **Dictionary**

Dictionary is an unordered set of a key-value pair of items. It is like an associative array or a hash table where each key stores a specific value. Key can hold any primitive data type, whereas value is an arbitrary Python object.

Items in the dictionary are separated with the comma (,) and enclosed in the curly braces {}.

1. d = {1:'Jimmy', 2:'Alex', 3:'john', 4:'mike'}
2. **print** (d) # Printing dictionary
3. **print**("1st name is "+d[1]) #1st name is Jimmy
4. **print**("2nd name is "+ d[4]) #2nd name is mike
5. **print** (d.keys()) #dict\_keys([1, 2, 3, 4])
6. **print** (d.values()) //dict\_values(['Jimmy', 'Alex', 'john', 'mike'])

### **Set:** Python Set is the unordered collection of the data type. It is iterable, mutable, and has unique elements. it may return the changed sequence of the element. Set is created by using a built-in function **set(),** or a sequence of element is passed in curly {}

### And separated by the comma. It can contain various types of values.

1. set1 = set() # Creating Empty set
2. set2 = {'James', 2, 3,'Python'}
3. **print**(set2) //{3, 'Python', 'James', 2} #Printing Set value
4. set2.add(10) # Adding element to the set
5. **print**(set2) //{'Python', 'James', 3, 2, 10}
6. set2.remove(2) #Removing element from the set
7. **print**(set2) //{'Python', 'James', 3, 10}

# Python Keywords

Python keywords are unique words reserved with defined meanings .You'll never need to import any keyword into your program because they're permanently present.

Python's built-in methods and classes are not the same as the keywords. Built-in methods and classes are constantly present; however, they are not as limited in their application as keywords.

**You can not assign a particular meaning to Python keywords in our code. You'll get a message of SyntaxError if you attempt to do the same. If you attempt to assign anything to a built-in method or type, you will not receive a SyntaxError message; however, it is still not a smart idea.**

**Python contains thirty-five keywords in the most recent version,**

| False | await | else | import | async | with | pass |
| --- | --- | --- | --- | --- | --- | --- |
| None | break | except | in | elif | assert | raise |
| True | class | finally | is | if | del | return |
| and | continue | for | lambda | or | global | try |
| as | def | from | nonlocal | yield | not | while |

**#for printing the available keywords**

1. **import keyword**
2. **print( keyword.kwlist ) # displaying the complete keyword list using "kwlist()."**
3. **print( True == 3 ) //False**
4. **print( False == 0 ) //True**
5. **print( True + True + True) //3**

**The None Keyword:** None is a Python keyword that means "nothing." None is known as nil, null, or undefined in different computer languages.

If a function does not have a return clause, it will give None as the default output:

| **Mathematical Operations** | **Operations in Other Languages** | **Python Keyword** |
| --- | --- | --- |
| **AND, ∧** | **&&** | **and** |
| **OR, ∨** | **||** | **or** |
| **NOT, ¬** | **!** | **not** |
| **CONTAINS, ∈** |  | **in** |
| **IDENTITY** | **===** | **is** |

**The in Keyword:** The in keyword of Python is a robust confinement checker, also known as a membership operator. If you provide it an element to seek and a container or series to seek into, it will give True or False, depending on if that given element was located in the given container:

1. <an\_element> **in** <a\_container>
2. container = "Javatpoint"
3. **print**( "p" **in** container ) //True
4. **print**( "P" **in** container ) //False
5. **print**( container.**\_\_contains\_\_**(“p”) ) //True

Lists, dictionaries, tuples, strings, or any data type with the method **\_\_contains\_\_**(), or we can iterate over it will work with the in keyword.

**The is Keyword:** In Python, it's used to check the identification of objects.

The == operation is used to determine whether two arguments are identical. It also determines whether two arguments relate to the unique object.

When the objects are the same, it gives True; otherwise, it gives False.

1. **print**( True **is** True ) //True
2. **print**( False **is** True ) //False
3. **print**( [] == [] ) //True
4. **print**( [] **is** [] ) //False
5. **print**( {} == {} ) //True
6. **print**( {} **is** {} ) //False

Strings and tuples, unlike lists and dictionaries, are unchangeable. As a result, two equal strings or tuples are also identical. They're both referring to the unique memory region.

### The nonlocal Keyword

Nonlocal keyword usage is fairly analogous to global keyword usage. The keyword nonlocal is designed to indicate that a variable within a function that is inside a function, i.e., a nested function is just not local to it, implying that it is located in the outer function.

We must define a non-local parameter with nonlocal if we ever need to change its value under a nested function. Otherwise, the nested function creates a local variable using that title.

1. **def** the\_outer\_function():
2. var = 10
3. **def** the\_inner\_function():
4. nonlocal var
5. var = 14
6. **print**("The value inside the inner function: ", var) //14
7. the\_inner\_function()
8. **print**("The value inside the outer function: ", var) //14
9. the\_outer\_function()

### **Exception Handling Keywords - try, except, raise, finally, & assert**

**try:** This keyword is designed to handle exceptions and is used in conjunction with the keyword except to handle problems in the program. When there is some kind of error, the program inside the "try" block is verified, but the code in that block is not executed.

**except:** As previously stated, this operates in conjunction with "try" to handle exceptions.

**finally:** Whatever outcome of the "try" section, the "finally" box is implemented every time.

**raise:** The raise keyword could be used to specifically raise an exception.

**assert:** This method is used to help in troubleshooting. Often used to ensure that code is correct. Nothing occurs if an expression is interpreted as true; however, if it is false, "AssertionError" is raised. An output with the error, followed by a comma, can also be printed.

### The pass Keyword: A null sentence is called a pass. It serves as a stand-in for something else. When it is run, nothing occurs.

Let's say we possess a function that has not been coded yet however we wish to do so in the long term. If we write just this in the middle of code,

1. **def** function\_pass( arguments ):

**O/P: IndentationError: expected an indented block after function definition on** line

as shown, IndentationError will be thrown. Rather, we use the pass command to create a blank container.

1. **def** function\_pass( arguments ):
2. **pass**

### **The return Keyword:** The return expression is used to leave a function and generate a result.

The None keyword is returned by default if we don't specifically return a value.

### **The del Keyword:** del keyword is used to remove any reference to an object. Every entity is an object. We can use the del command to remove a variable reference.

1. var1 = var2 = 5
2. **del** var1

# Python Literals

Python literals are a data type and can hold any value type, such as strings, numbers, and more.

Python Literals can be defined as data that is given in a variable or constant.Python supports the following literals:

### 1. String literals:

String literals can be formed by enclosing a text in the quotes. We can use both single as well as double quotes to create a string.

"Aman" , '12345'

**Types of Strings:** There are two types of Strings supported in Python:

**a) Single-line String**- Strings that are terminated within a single-line.

text1='hello'

**b) Multi-line String -** A piece of text that is written in multiple lines.

There are two ways to create multiline strings:

**1) Add a black slash at the end of each line.**

1. text1='hello\
2. user'
3. **print**(text1) ///'hellouser'

**2) Using triple quotation marks:-**

1. str2='''''welcome
2. to
3. SSSIT'''
4. **print** str2

**O/P:-- welcome**

**to**

**SSSIT**

### II. Numeric literals:

Numeric Literals are immutable. Numeric literals can belong to the following four different numerical types.

| **Int(signed integers)** | **Long(long integers)** | **float(floating point)** | **Complex(complex)** |
| --- | --- | --- | --- |
| Numbers( can be both positive and negative) with no fractional part.eg: 100 | Integers of unlimited size followed by lowercase or uppercase L eg: 87032845L | Real numbers with both integer and fractional part eg: -26.2 | In the form of a+bj where a forms the real part and b forms the imaginary part of the complex number. eg: 3.14j |

### III. Boolean literals:

A Boolean literal can have any of the two values: True or False.

### IV. Special literals.

Python contains one special literal i.e., **None.**

None is used to specify that field that is not created. It is also used for the end of lists in Python.

1. val1=10
2. val2=None
3. **print**(val1) //10
4. **print**(val2) //None

### V. Literal Collections.

Python provides the four types of literal collection such as List literals, Tuple literals, Dict literals, and Set literals.

**List:**

* List contains items of different data types. Lists are mutable i.e., modifiable.
* The values stored in List are separated by comma(,) and enclosed within square brackets([]). We can store different types of data in a List.

1. list=['John',678,20.4,'Peter']
2. list1=[456,'Andrew']
3. **print**(list) //['John', 678, 20.4, 'Peter']
4. **print**(list + list1) //['John', 678, 20.4, 'Peter', 456, 'Andrew']

## Operators

## Membership Operators

| **Operator** | **Description** |
| --- | --- |
| in | If the first operand is found in the second operand, it is evaluated to be true (list, tuple, dictionary). |
| not in | If the first operand is not present in the second operand, evaluation is true (list, tuple, or dictionary). |

## Identity Operators

| **Operator** | **Description** |
| --- | --- |
| is | If the references on both sides point to the same object, it is determined to be true. |
| is not | If the references on both sides do not point at the same object, it is determined to be true. |

### Python Comments:

### Single-Line Comments

1. Multi-Line Comments: python does not support multi line comments.

**Using String Literals:**

Because Python overlooks string expressions that aren't allocated to a variable, we can utilize them as comments.

1. **'it is a comment extending to multiple lines'**

**We can observe that on running this code, there will be no output; thus, we utilize the strings inside triple quotes(""") as multi-line comments.**

### Python Docstring

The strings enclosed in triple quotes that come immediately after the defined function are called Python docstring.

1. **def add(x, y):**
2. **"""This function adds the values of x and y"""**
3. **return x + y**
4. **# Displaying the docstring of the add function**
5. **print( add.\_\_doc\_\_ ) //This function adds the values of x and y**

# Python If-else statements

1. **if condition:**
2. **#block of statements**
3. **else:**
4. **#another block of statements (else-block)**

## The elif statement

The elif statement enables us to check multiple conditions and execute the specific block of statements depending upon the true condition among them.

1. **if expression 1:**
2. **# block of statements**
3. **elif expression 2:**
4. **# block of statements**
5. **elif expression 3:**
6. **# block of statements**

# Python Loops

| **SN** | **Name of the loop** | **Loop Type & Description** |
| --- | --- | --- |
| 1 | While loop | Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body. |
| 2 | For loop | This type of loop executes a code block multiple times and abbreviates the code that manages the loop variable. |
| 3 | Nested loops | We can iterate a loop inside another loop. |

| **Name of the control statement** | **Description** |
| --- | --- |
| Break statement | This command terminates the loop's execution and transfers the program's control to the statement next to the loop. |
| Continue statement | This command skips the current iteration of the loop. The statements following the continue statement are not executed once the Python interpreter reaches the continue statement. |
| Pass statement | The pass statement is used when a statement is syntactically necessary, but no code is to be executed. |

## The for Loop

Python for loop is designed to repeatedly execute a code block while iterating through a list, tuple, dictionary, or other iterable objects of Python. The process of traversing a sequence is known as iteration.

1. **for** value **in** sequence:
2. { code block }
3. numbers = [4, 2, 6, 7, 3, 5, 8, 10] # Creating a sequence which is a tuple of nums
4. square = 0 # variable to store the square of the number
5. squares = [] # Creating an empty list
6. **for** value **in** numbers: # Creating a for loop
7. square = value \*\* 2
8. squares.append(square)
9. **print**("The list of squares is", squares) [16, 4, 36, 49, 9, 25, 64, 100]

The range() Function

The **range**() **function** returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and stops before a specified number.

With the help of the range() function, we may produce a series of numbers. range(10) will produce values between 0 and 9. (10 numbers).

We can give specific start, stop, and step size values in the manner range(start, stop, step size). If the step size is not specified, it defaults to 1.

1. **print**(range(15)) //range(0, 15)
2. **print**(list(range(15))) //[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
3. **print**(list(range(4, 9))) //[4, 5, 6, 7, 8]
4. **print**(list(range(5, 25, 4))) //[5, 9, 13, 17, 21]

## While Loop

While loops are used in Python to iterate until a specified condition is met. statement in the program that follows the while loop is executed once the condition changes to false.

1. **while** <condition>:
2. { code block }
3. counter = 0
4. **while** counter < 10: # # Initiating the loop and giving the condition
5. counter = counter + 3
6. **print**("Python Loops")

**Output:** Python Loops

Python Loops

Python Loops

Python Loops

1. my\_list = [3, 5, 6, 8, 4]
2. **for** iter\_var **in** range( len( my\_list ) ):
3. my\_list.append(my\_list[iter\_var] + 2)
4. **print**( my\_list ) //[3, 5, 6, 8, 4, 5, 7, 8, 10, 6]

### Python While Loop with List

### We will use a Python while loop to square every number of a list

### list\_ = [3, 5, 1, 4, 6] //# initializing a list

### squares = []

### # programing a **while** loop

### **while** list\_: # until list is not empty **this** expression will give **boolean** True after that False

### squares.append( (list\_.pop())\*\*2) //pop method use to remove element at specific position and return that value

### print( squares ) // [36, 16, 1, 25, 9]

### Single statement while Block

1. counter = 0
2. **while** (count < 3): **print**("Python Loops")

### Continue Statement

It returns the control to the beginning of the loop

1. **for** string **in** "Python Loops": # Initiating the loop
2. **if** string == "o" **or** string == "p" **or** string == "t":
3. **continue**
4. **print**('Current Letter:', string) //P y h n L s

### Break Statement

It stops the execution of the loop when the break statement is reached.

1. **for** string **in** "Python Loops": # Initiating the loop
2. **if** string == 'L':
3. **break**
4. **print**('Current Letter: ', string) //P y t h o n

### Pass Statement

Pass statements are used to create empty loops. Pass statement is also employed for classes, functions, and empty control statements.

We can simply insert a pass in places where empty code is prohibited, such as loops, functions, class definitions, or if-else statements.

1. **for** a **in** "Python Loops": # program to show how the pass statement works
2. **pass**
3. **print**( 'Last Letter:', a)

**O/P:-**  Last Letter: s

1. '''''pass acts as a placeholder. We can fill this place later on'''
2. sequence = {"Python", "Pass", "Statement", "Placeholder"}
3. **for** value **in** sequence:
4. **if** value == "Pass":
5. **pass** # leaving an empty if block using the pass keyword
6. **else**:
7. **print**("Not reached pass keyword: ", value)

**O/P:-** Not reached pass keyword: Python

Not reached pass keyword: Placeholder

Not reached pass keyword: Statement

## Nested Loops

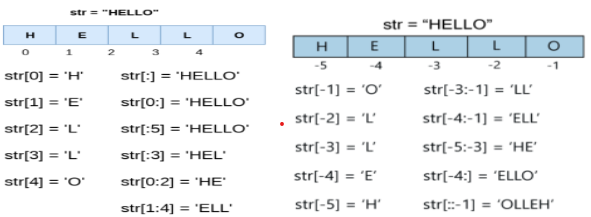
If we have a piece of script that we want to run a number of times and then another piece of script inside that script that we want to run B number of times, we employ a "nested loop." When working with an iterable in lists, these are widely utilized in Python.

1. **import** random
2. numbers = [ ]
3. **for** val **in** range(0, 11):
4. numbers.append( random.randint( 0, 11 ) )
5. **for** num **in** range( 0, 11 ):
6. **for** i **in** numbers:
7. **if** num == i:
8. **print**( num, end = " " ) //0 2 4 5 6 7 8 8 9 10

## Python Continue vs. Pass

| **Headings** | **continue** | **pass** |
| --- | --- | --- |
| **Definition** | The continue statement is utilized to skip the current loop's remaining statements, go to the following iteration, and return control to the beginning. | The pass keyword is used when a phrase is necessary syntactically to be placed but not to be executed. |
| **Action** | It takes the control back to the start of the loop. | Nothing happens if the Python interpreter encounters the pass statement. |
| **Application** | It works with both the Python while and Python for loops. | It performs nothing; hence it is a null operation. |
| **Syntax** | It has the following syntax: -: continue | Its syntax is as follows:- pass |
| **Interpretation** | It's mostly utilized within a loop's condition. | During the byte-compile stage, the pass keyword is removed. |

# Python String



## Reassigning Strings

Updating the content of the strings is as easy as assigning it to a new string. The string object doesn't support item assignment i.e. A string can only be replaced with a new string since its content cannot be partially replaced. **Strings are immutable in Python.**

1. str = "HELLO"
2. str[0] = "h" //error
3. **print**(str)

## Deleting the String

As we know that strings are immutable. We cannot delete or remove the characters from the string. But we can delete the entire string using the **del** keyword.

1. str = "JAVATPOINT"
2. **del** str[1]

## String Operators

| **Operator** | **Description** |
| --- | --- |
| + | It is known as a concatenation operator used to join the strings given either side of the operator. |
| \* | It is known as a repetition operator. It concatenates multiple copies of the same string. |
| [] | It is known as a slice operator. It is used to access the sub-strings of a particular string. |
| [:] | It is known as a range slice operator. It is used to access the characters from the specified range. |
| in | It is known as a membership operator. It returns if a particular substring is present in the specified string. |
| not in | It is also a membership operator and does the exact reverse of in. It returns true if a particular substring is not present in the specified string. |
| r/R | **It is used to specify the raw string.** Raw strings are used in the cases where we need to print the actual meaning of escape characters such as "C://python". To define any string as a raw string, the character r or R is followed by the string. |
| % | It is used to perform string formatting. It makes use of the format specifiers used in C programming like %d or %f to map their values in python. |

1. str = "Hello"
2. str1 = " world"
3. **print**(str\*3) # prints HelloHelloHello
4. **print**(str+str1)# prints Hello world
5. **print**(str[4]) # prints o
6. **print**(str[2:4]); # prints ll
7. **print**('w' **in** str) # prints false as w is not present in str
8. **print**('wo' **not** **in** str1) # prints false as wo is present in str1.
9. **print**(r'C://python37') # prints C://python37 as it is written
10. **print**("The string str : %s"%(str)) # prints The string str : Hello

## Python String Formatting

### Escape Sequence

Let's suppose we need to write the text as - They said, "Hello what's going on?"- the given statement can be written in single quotes or double quotes but it will raise the **SyntaxError** as it contains both single and double-quotes.

1. str = "They said, "Hello what's going on?""
2. **print**(str) //SyntaxError: invalid syntax

We can use the triple quotes to accomplish this problem but Python provides the escape sequence.

**The backslash(/)** symbol denotes the escape sequence. The backslash can be followed by a special character and it is interpreted differently. The single quotes inside the string must be escaped. We can apply the same as in the double quotes.

1. # using triple quotes
2. **print**('''''They said, "What's there?"''') // They said, "What's there?"
3. # escaping single quotes
4. **print**('They said, "What\'s going on?"') //They said, "What's going on?"
5. # escaping double quotes
6. **print**("They said, \"What's going on?\"") //They said, "What's going on?"

The list of an escape sequence is given below:

| **Escape Sequence** | **Description** | **Example** |
| --- | --- | --- |
| \newline | It ignores the new line. | print("Python1 \  Python2 \  Python3")  **Output:**Python1 Python2 Python3 |
| \\ | Backslash | print("\\")  **Output:**\ |
| \' | Single Quotes | print('\'')  **Output:**' |
| \\'' | Double Quotes | print("\"")  **Output:**" |
| \a | ASCII Bell | print("\a") |
| \b | ASCII Backspace(BS) | print("Hello \b World")  **Output:**Hello World |
| \f | ASCII Formfeed | print("Hello \f World!")  Hello World! |
| \n | ASCII Linefeed | print("Hello \n World!")  **Output:**Hello  World! |
| \r | ASCII Carriage Return(CR) | print("Hello \r World!")  **Output:**World! |
| \t | ASCII Horizontal Tab | print("Hello \t World!")  **Output:**Hello World! |
| \v | ASCII Vertical Tab | print("Hello \v World!")  **Output:**Hello  World! |
| \ooo | Character with octal value | print("\110\145\154\154\157")  **Output:**Hello |
| \xHH | Character with hex value. | print("\x48\x65\x6c\x6c\x6f")  **Output:**Hello |

## The format() method

The **format()** method is the most flexible and useful method in formatting strings. The curly braces {} are used as the placeholder in the string and replaced by the **format()** method argument. Let's have a look at the given an example:

1. **print**("{} and {} both are friend".format("Dev","Abhishek")) # Using Curly braces
2. //Dev and Abhishek both are friend
3. **print**("{1} and {0} best players ".format("Virat","Rohit")) #Positional Argument
4. //Rohit and Virat best players
5. **print**("{a},{b},{c}".format(a = "Dev", b = "Peter", c = "Ricky")) #Keyword Argument //Dev,,Peter,Ricky

## Python String functions

## Python provides many in-built functions that are used for string handling.

| Method | Description |
| --- | --- |
| [capitalize()](https://www.javatpoint.com/python-string-capitalize-method) | It capitalizes the first character of the String. This function is deprecated in python3 |
| [casefold()](https://www.javatpoint.com/python-string-casefold-method) | It returns a version of s suitable for case-less comparisons. |
| [center(width ,fillchar)](https://www.javatpoint.com/python-string-center-method) | It returns a space padded string with the original string centred with equal number of left and right spaces. |
| [count(string,begin,end)](https://www.javatpoint.com/python-string-count-method) | It counts the number of occurrences of a substring in a String between begin and end index. |
| decode(encoding = 'UTF8', errors = 'strict') | Decodes the string using codec registered for encoding. |
| [encode()](https://www.javatpoint.com/python-string-encode-method) | Encode S using the codec registered for encoding. Default encoding is 'utf-8'. |
| [endswith(suffix ,begin=0,end=len(string))](https://www.javatpoint.com/python-string-endswith-method) | It returns a Boolean value if the string terminates with a given suffix between begin and end. |
| [expandtabs(tabsize = 8)](https://www.javatpoint.com/python-string-expandtabs-method) | It defines tabs in a string to multiple spaces. The default space value is 8. |
| [find(substring ,beginIndex, endIndex)](https://www.javatpoint.com/python-string-find-method) | It returns the index value of the string where substring is found between begin index and end index. |
| [format(value)](https://www.javatpoint.com/python-string-format-method) | It returns a formatted version of S, using the passed value. |
| [index(subsring, beginIndex, endIndex)](https://www.javatpoint.com/python-string-index-method) | It throws an exception if string is not found. It works the same as the find() method. |
| [isalnum()](https://www.javatpoint.com/python-string-isalnum-method) | It returns true if the characters in the string are alphanumeric i.e., alphabets or numbers and there is at least 1 character. Otherwise, it returns false. |
| [isalpha()](https://www.javatpoint.com/python-string-isalpha-method) | It returns true if all the characters are alphabets and there is at least one character, otherwise False. |
| [isdecimal()](https://www.javatpoint.com/python-string-isdecimal-method) | It returns true if all the characters of the string are decimals. |
| [isdigit()](https://www.javatpoint.com/python-string-isdigit-method) | It returns true if all the characters are digits and there is at least one character, otherwise False. |
| [isidentifier()](https://www.javatpoint.com/python-string-isidentifier-method) | It returns true if the string is the valid identifier. |
| [islower()](https://www.javatpoint.com/python-string-islower-method) | It returns true if the characters of a string are in lower case, otherwise false. |
| [isnumeric()](https://www.javatpoint.com/python-string-isnumeric-method) | It returns true if the string contains only numeric characters. |
| [isprintable()](https://www.javatpoint.com/python-string-isprintable-method) | It returns true if all the characters of s are printable or s is empty, false otherwise. |
| [isupper()](https://www.javatpoint.com/python-string-isupper-method) | It returns false if characters of a string are in Upper case, otherwise False. |
| [isspace()](https://www.javatpoint.com/python-string-isspace-method) | It returns true if the characters of a string are white-space, otherwise false. |
| [istitle()](https://www.javatpoint.com/python-string-istitle-method) | It returns true if the string is titled properly and false otherwise. A title string is the one in which the first character is upper-case whereas the other characters are lower-case. |
| [isupper()](https://www.javatpoint.com/python-string-isupper-method) | It returns true if all the characters of the string(if exists) is true otherwise it returns false. |
| [join(seq)](https://www.javatpoint.com/python-string-join-method) | It merges the string representation of the given sequence. |
| len(string) | It returns the length of a string. |
| [ljust(width[,fillchar])](https://www.javatpoint.com/python-string-ljust-method) | It returns the space padded strings with the original string left justified to the given width. |
| [lower()](https://www.javatpoint.com/python-string-lower-method) | It converts all the characters of a string to Lowercase. |
| [lstrip()](https://www.javatpoint.com/python-string-lstrip-method) | It removes all leading whitespaces of a string and can also be used to remove a particular character from leading. |
| [partition()](https://www.javatpoint.com/python-string-partition-method) | It searches for the separator sep in S, and returns the part before it, the separator itself, and the part after it. If the separator is not found, return S and two empty strings. |
| maketrans() | It returns a translation table to be used in the translation function. |
| [replace(old,new[,count])](https://www.javatpoint.com/python-string-replace-method) | It replaces the old sequence of characters with the new sequence. The max characters are replaced if max is given. |
| [rfind(str,beg=0,end=len(str))](https://www.javatpoint.com/python-string-rfind-method) | It is similar to find but it traverses the string in backward direction. |
| [rindex(str,beg=0,end=len(str))](https://www.javatpoint.com/python-string-rindex-method) | It is the same as an index but it traverses the string in backward direction. |
| [rjust(width,[,fillchar])](https://www.javatpoint.com/python-string-rjust-method) | Returns a space padded string having the original string right justified to the number of characters specified. |
| [rstrip()](https://www.javatpoint.com/python-string-rstrip-method) | It removes all trailing whitespace of a string and can also be used to remove a particular character from trailing. |
| [rsplit(sep=None, maxsplit = -1)](https://www.javatpoint.com/python-string-rsplit-method) | It is the same as split() but it processes the string from the backward direction. It returns the list of words in the string. If Separator is not specified then the string splits according to the white-space. |
| [split(str,num=string.count(str))](https://www.javatpoint.com/python-string-split-method) | Splits the string according to the delimiter str. The string splits according to the space if the delimiter is not provided. It returns the list of substring concatenated with the delimiter. |
| [splitlines(num=string.count('\n'))](https://www.javatpoint.com/python-string-splitlines-method) | It returns the list of strings at each line with newline removed. |
| [startswith(str,beg=0,end=len(str))](https://www.javatpoint.com/python-string-startswith-method) | It returns a Boolean value if the string starts with given str between begin and end. |
| strip([chars]) | It is used to perform lstrip() and rstrip() on the string. |
| [swapcase()](https://www.javatpoint.com/python-string-swapcase-method) | It inverts the case of all characters in a string. |
| title() | It is used to convert the string into the title-case i.e., The string meEruT will be converted to Meerut. |
| [translate(table,deletechars = '')](https://www.javatpoint.com/python-string-translate-method) | It translates the string according to the translation table passed in the function |
| [upper()](https://www.javatpoint.com/python-string-upper-method) | It converts all the characters of a string to UpperCase. |
| [zfill(width)](https://www.javatpoint.com/python-string-zfill-method) | Returns original string leftpadded with zeros to a total of width characters; intended for numbers, zfill() retains any sign given (less one zero). |

# Python List

A list in Python is used to store the sequence of various types of data.

Python lists are mutable types. The items in the list are separated with the comma (,) and enclosed with the square brackets [].

## Characteristics of Lists

* The lists are ordered.
* The element of the list can be accessed by index.
* The lists are the mutable type.
* A list can store the number of various elements.

### Ordered List Checking

1. a = [ 1, 2, "Ram", 3.50, "Rahul", 5, 6 ]
2. b = [ 1, 2, 5, "Ram", 3.50, "Rahul", 6 ]
3. a == b //False
4. a = [ 1, 2, "Ram", 3.50, "Rahul", 5, 6]
5. b = [ 1, 2, "Ram", 3.50, "Rahul", 5, 6]
6. a == b //True

**We can get the sub-list of the list using the following syntax.**

list\_varible(start:stop:step)

1. **start** denotes the starting index position of the list.
2. The **stop** denotes the last index position of the list.
3. The **step** is used to skip the nth element within a **start:stop**

* list = [1,2,3,4,5,6,7]
* **print**(list[1:6:2]) //[2, 4, 6]

## Updating List Values

List values can be updated by using the slice and assignment operator. Python also provides append() and insert() methods, which can be used to add values to the list.

1. list = [1, 2, 3, 4, 5, 6] # updating list values
2. **print**(list) //[1, 2, 3, 4, 5, 6]
3. list[2] = 10 # It will assign value to the value to the second index
4. **print**(list) //[1, 2, 10, 4, 5, 6]
5. list[1:3] = [89, 78] # Adding multiple-element
6. **print**(list) //[1, 89, 78, 4, 5, 6]
7. list[-1] = 25 # It will add value at the end of the list
8. **print**(list) //[1, 89, 78, 4, 5, 25]

The list elements can also be deleted by using the **del** keyword. Python also provides us the **remove()** method if we do not know which element is to be deleted from the list.

l.append(input("Enter the item:"))

list.remove(2)

## Python List Operations

The concatenation (+) and repetition (\*) operators work in the same way as they were working with the strings. The different operations of list are

1. Repetition
2. Concatenation
3. Length
4. Iteration //The for loop is used to iterate over the list elements.
5. Membership //It returns true if a particular item exists in a particular list.

## Python List Built-in Functions

1. len()
2. max()
3. min()

**Example: 1-** Write the program to remove the duplicate element of the list.

1. list1 = [1,2,2,3,55,98,65,65,13,29]
2. list2 = [] # Declare an empty list that will store unique values
3. **for** i **in** list1:
4. **if** i **not** **in** list2:
5. list2.append(i)
6. **print**(list2) //[1, 2, 3, 55, 98, 65, 13, 29]

# Python Tuples

* Tuples are an immutable data type, which means that once they have been generated, their elements cannot be changed.
* Since tuples are ordered sequences, each element has a specific order that will never change.

1. nested\_tuple = ("Python", {4: 5, 6: 2, 8:2}, (5, 3, 5, 6)) # Creating a nested tuple
2. print("A nested tuple: ", nested\_tuple) //('Python', {4: 5, 6: 2, 8: 2}, (5, 3, 5, 6))

**Tuples can be constructed without using parentheses.** This is known as triple packing.

1. tuple\_ = 4, 5.7, "Tuples", ["Python", "Tuples"]
2. print(tuple\_) //(4, 5.7, 'Tuples', ['Python', 'Tuples'])

The construction of a tuple from a single member might be hard.

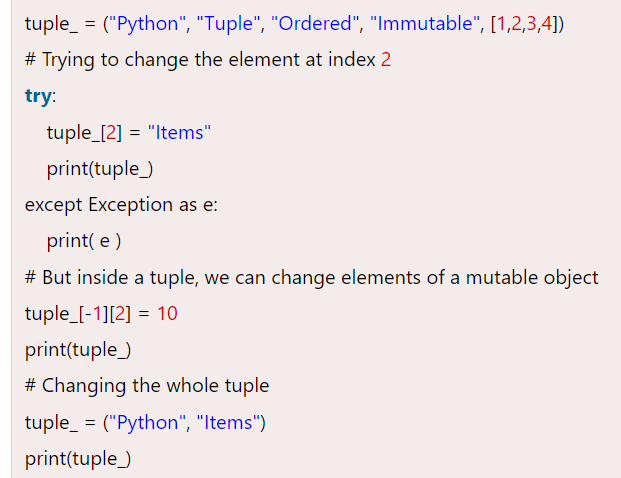
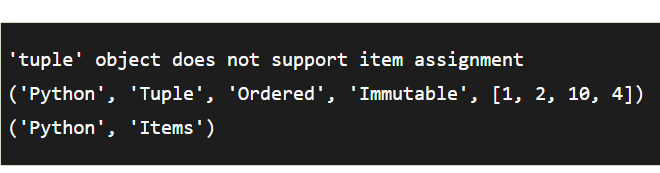
Simply adding parenthesis around the element is insufficient. To be recognised as a tuple, the element must be followed by a comma.

1. tup=()
2. print( type(tup) ) //<class 'tuple'>
3. single\_tuple = ("Tuple")
4. print( type(single\_tuple) ) //<class 'str'>
5. single\_tuple = ("Tuple",) # Creating a tuple that has only one element
6. print( type(single\_tuple) ) //<class 'tuple'>
7. single\_tuple = "Tuple", # Creating tuple without parentheses
8. print( type(single\_tuple) ) //<class 'tuple'>

**Count (val) Method:** The number of times the specified element occurs in the tuple is returned by the count() function of Tuple.

### Changing a Tuple

This suggests that we are unable to change a tuple's elements once they have been defined. **The nested elements of an element can be changed,** though, if the element itself is a changeable data type like a list.

# Python Set

A Python set is the collection of the unordered items. Each element in the set must be unique, mutable, and the sets remove the duplicate elements.

## Creating a set

The set can be created by enclosing the comma-separated immutable items with the curly braces {}. Python also provides the set() method, which can be used to create the set by the passed sequence.

### Example 1: Using curly braces

Days = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"}

### Example 2: Using set() method

Days = set(["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"])

**Set can contain any type of element such as integer, float, tuple etc**. **But mutable elements (list, dictionary, set) can't be members of set.**

1. set1 = {1,2,3, "JavaTpoint", 20.5, 14} # Creating a set which have immutable elem
2. **print**(type(set1)) //<class 'set'>
3. set2 = {1,2,3,["Javatpoint",4]} #Creating a set which have mutable element
4. **print**(type(set2)) //error

Creating an empty set is a bit different because empty curly {} braces are also used to create a dictionary as well. So Python provides the set() method used without an argument to create an empty set.

1. set3 = {} # Empty curly braces will create dictionary
2. **print**(type(set3)) //<class 'dict'>
3. set4 = set() # Empty set using set() function
4. **print**(type(set4)) //<class 'dict'>

## Adding items to the set

Python provides the **add()** method and **update()** method which can be used to add some particular item to the set. The **add**() method is used to add a single element whereas the **update**() method is used to add multiple elements to the set.

Months = set(["January","February", "March", "April", "May", "June"])

Months.add("July"); //can add single item at a time

Months.update(["July","August","Sept","October"]); //can add multiple item at a time

## Removing items from the set

Python provides the **discard()** method and **remove()** method which can be used to remove the items from the set.

difference between these functions, using **discard**() function if the item does not exist in the set then the set remains unchanged whereas **remove**() method will throw an error.

We can also use the pop() method to remove the item. Generally, the pop() method will always remove the last item but the set is unordered, so we can't determine which element will be popped from the set.

## Python Set Operations

Set can be performed mathematical operations such as union, intersection, difference, and symmetric difference. Python provides the facility to carry out these operations with operators or methods.

### Union of two Sets

The union of two sets is calculated by using the pipe (|) operator or using **union() method**. Union of the two sets contains all the items that are present in both the sets.

1. Days1 = {"Monday","Tuesday","Wednesday","Thursday", "Sunday"}
2. Days2 = {"Friday","Saturday","Sunday"}
3. **print**(Days1|Days2) #printing the union of the sets
4. **print**(Days1.union(Days2)) #printing the union of the sets

### Intersection of two sets

### Intersection of two sets can be performed by **and &** operator or **intersection() function**.

1. Days1 = {"Monday","Tuesday", "Wednesday", "Thursday"}
2. Days2 = {"Monday","Tuesday","Sunday", "Friday"}
3. **print**(Days1&Days2) #prints the intersection of the two sets

**print**(set1.intersection(set2)) #prints the intersection of the two sets

## The intersection\_update() method

The **intersection\_update()** method removes the items from the original set that are not present in both the sets (all the sets if more than one are specified).

The **intersection\_update()** method is different from the intersection() method since it modifies the original set by removing the unwanted items, on the other hand, the intersection() method returns a new set.

1. a = {"Devansh", "bob", "castle"}
2. b = {"castle", "dude", "emyway"}
3. c = {"fuson", "gaurav", "castle"}
4. a.intersection\_update(b, c)
5. **print**(a) //{‘castle’}

## Difference between the two sets

## The difference of two sets can be calculated by using the subtraction (-) operator or **intersection()** method. Suppose there are two sets A and B, and the difference is A-B that denotes the resulting set will obtain that element of A, which is not present in the set B.

1. Days1 = {"Monday", "Tuesday", "Wednesday", "Thursday"}
2. Days2 = {"Monday", "Tuesday", "Sunday"}
3. //using subtraction operator
4. **print**(Days1-Days2) #{"Wednesday", "Thursday" will be printed}
5. //using difference method && prints the diffof the two sets Days1 and Days2
6. **print**(Days1.difference(Days2)) # {'Thursday', 'Wednesday'}

## Symmetric Difference of two sets

Symmetric difference of two sets is calculated by the ^ operator or **symmetric\_difference()** method. Symmetric differs from sets, it removes that element which is present in both sets.

1. a = {1,2,3,4,5,6}
2. b = {1,2,9,8,10}
3. c = a^b //**Using ^ operator**
4. **print**(c) //{3, 4, 5, 6, 8, 9, 10}
5. c = a.symmetric\_difference(b) //**Using symmetric\_difference() method**

## Set comparisons

Python allows us to use the comparison operators i.e., <, >, <=, >= , == with the sets by using which we can check whether a set is a subset, superset, or equivalent to another set. boolean true or false is returned depending upon the items present inside the sets.

1. Days1 = {"Monday", "Tuesday", "Wednesday", "Thursday"}
2. Days2 = {"Monday", "Tuesday"}
3. Days3 = {"Monday", "Tuesday", "Friday"}
4. #Days1 is the superset of Days2 hence it will print true.
5. **print** (Days1>Days2) //True
6. #prints false since Days1 is not the subset of Days2
7. **print** (Days1<Days2) //False
8. #prints false since Days2 and Days3 are not equivalent
9. **print** (Days2 == Days3) //False

## FrozenSets

The frozen sets are the immutable form of the normal sets, i.e., the items of the frozen set cannot be changed and therefore it can be used as a key in the dictionary.

The frozenset() method is used to create the frozenset object. The iterable sequence is passed into this method which is converted into a frozen set as a return type of method.

1. Frozenset = frozenset([1,2,3,4,5])
2. **print**(type(Frozenset))
3. **print**("\nprinting the content of frozen set...")
4. **for** i **in** Frozenset:
5. **print**(i);
6. Frozenset.add(6) #error bcoz cannot change content of Frozenset after creation

## Frozenset for the dictionary

If we pass the dictionary as the sequence inside the frozenset() method, it will take only the keys from the dictionary and return a frozenset that contains the key of the dictionary as its elements.

## Python Built-in set method

| **Method** | **Description** |
| --- | --- |
| [add(item)](https://www.javatpoint.com/python-set-add-method) | It adds an item to the set. It has no effect if the item is already present in the set. |
| clear() | It deletes all the items from the set. |
| copy() | It returns a shallow copy of the set. |
| difference\_update(....) | It modifies this set by removing all items that are also present in the specified sets. |
| [discard(item)](https://www.javatpoint.com/python-set-discard-method) | It removes the specified item from the set. |
| intersection() | It returns a new set that contains only the common elements of both the sets. (all the sets if more than two are specified). |
| intersection\_update(....) | It removes the items from the original set that are not present in both the sets (all the sets if more than one are specified). |
| Isdisjoint(....) | Return True if two sets have a null intersection. |
| Issubset(....) | Report whether another set contains this set. |
| Issuperset(....) | Report whether this set contains another set. |
| [pop()](https://www.javatpoint.com/python-set-pop-method) | Remove and return an arbitrary set element that is the last element of the set. Raises KeyError if the set is empty. |
| [remove(item)](https://www.javatpoint.com/python-set-remove-method) | Remove an element from a set; it must be a member. If the element is not a member, raise a KeyError. |
| symmetric\_difference(...) | Remove an element from a set; it must be a member. If the element is not a member, raise a KeyError. |
| symmetric\_difference\_update(....) | Update a set with the symmetric difference of itself and another. |
| union(....) | Return the union of sets as a new set.(i.e. all elements that are in either set.) |
| update() | Update a set with the union of itself and others. |

# Python Dictionary

* Python Dictionary is used to store the data in a key-value pair format.
* It is a mutable data-structure.
* The elements Keys and values are employed to create the dictionary.
* Keys must consist of just one element.
* Value can be any type such as list, tuple, integer, etc.

Python provides built-in function **dict()** method which is also used to create a dictionary.

The empty curly braces {} are used to create an empty dictionary.

1. Dict = {} # Creating an empty Dictionary
2. **print**(Dict)
3. Dict = dict({1: 'MS', 2: 'Google', 3:'book'}) # Creating a Dictionary with dict()
4. **print**(Dict) //{1: 'MS', 2: 'Google', 3: 'book'}
5. Dict = dict([(4, 'Preet'), (2, 'Var')]) # Creating a Dictionary with each item as a Pair
6. **print**(Dict) //{4: 'Preet', 2: 'Var'}

#### Note: If the key-value is already present in the dictionary, the value gets updated. Otherwise, the new keys added in the dictionary.

1. # for loop to print all the keys of a dictionary
2. Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}
3. **for** x **in** Employee:
4. **print**(x)
5. **for** x **in** Employee: #for loop to print all the values of the dictionary
6. **print**(Employee[x])
7. **for** x **in** Employee.values(): #loop to print values using values() method.
8. **print**(x)
9. **for** x **in** Employee.items(): #for loop to print items using items() method
10. **print**(x)
11. **for** x,y **in** Employee.items():
12. **print**(x,y)

## Properties of Dictionary Keys

1. In the dictionary, we cannot store multiple values for the same keys. If we pass more than one value for a single key, then the value which is last assigned is considered as the value of the key.

2. In python, the key cannot be any mutable object. We can use numbers, strings, or tuples as the key, but we cannot use any mutable object like the list as the key.

Built-in Dictionary Functions

A Python dictionary can be used with a handful of the methods that Python provides.

* **any()**

The any() method returns True indeed if one dictionary key does have a Boolean expression of True, much like it does for lists and tuples.

**The any() function returns True if any item in an iterable is true, otherwise it returns False. If the iterable object is empty, the any() function will return False.**

Note: When used in a dictionary, the any() function checks if any of the *keys* are true, not the *values*.

1. dict = {1: "Ayan", 2: "Bunny", 3: "Ram", 4: "Bheem"}
2. any({'':'','':'','3':''}) //True
3. any({0,0,""}) //False

* **all()**

Unlike in any() method, all() only returns True if each of the dictionary's keys contain a True Boolean value.

1. dict = {1: "Ayan", 2: "Bunny", 3: "Ram", 4: "Bheem"}
2. all({1:'',2:'','':''}) //False

* **sorted()**

The sorted() method returns an ordered series of the dictionary's keys, much like it does with lists as well as tuples.Initial Python dictionary is not changed by ascending sorting.

The sorted() function returns a sorted list of the specified iterable object.

Note: You cannot sort a list that contains BOTH string values & numeric values.

1. dict = {7: "Ayan", 5: "Bunny", 8: "Ram", 1: "Bheem"}
2. sorted(dict,reverse=False) ///[ 1, 5, 7, 8]

## Built-in Dictionary methods

| **clear()** | It is used to delete all the items of the dictionary. |
| --- | --- |
| **copy()** | It returns a shallow copy of the dictionary. |
| **pop()** | eliminates the element using the defined key. |
| **popitem()** | removes the most recent key-value pair entered. |
| **keys()** | It returns all the keys of the dictionary. |
| **items()** | It returns all the key-value pairs as a tuple. |
| **get()** | It is used to get the value specified for the passed key. |
| **update()** | It updates the dictionary by adding the key-value pair of dict2 to this dictionary.  dict\_demo.update({3: "TCS"}) |
| **values()** | It returns all the values of the dictionary. |