**Give an introduction to Python and explain the features of Python?**

**Introduction to Python:**

* Python is a popular programming language.
* It was created by **Guido van Rossum**, and released in 1991
* Python has a **simple syntax** similar to the English language.
* Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
* Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.

**Features of Python :**

* Easy to Code
* Easy to Read
* Free and Open-Source
* Robust Standard Library
* Interpreted
* Portable
* Object-Oriented and Procedure-Oriented
* Extensible
* Expressive
* Support for GUI
* Dynamically Typed
* High-level Language
* Simplify Complex Software Development
* Cross-platform Language
* Dynamic Memory Allocation
* Databases Support
* Large Standard Library

**Uses of Python:**

* web development (server-side),
* software development,
* mathematics,
* system scripting.
* Data analysis and machine learning
* Automation or scripting
* Software testing and prototyping
* Everyday tasks
* ERP and e-commerce systems
* Game development

**Explain Python Keywords?**

Python has a set of keywords that are reserved words.

Keywords have special meaning.

The keywords cannot be used as variable names, function names, or any other identifiers.

In Python, there is an inbuilt[**keyword module**](https://www.geeksforgeeks.org/keyword-module-in-python/) that provides an **[iskeyword() function](https://www.geeksforgeeks.org/check-string-valid-keyword-python/)** that can be used to check whether a given string is a valid keyword or not. Furthermore, we can check the name of the keywords in Python by using the kwlist attribute of the keyword module.

|  |  |
| --- | --- |
| Keyword | Description |
| [and](https://www.w3schools.com/python/ref_keyword_and.asp) | A logical operator |
| [as](https://www.w3schools.com/python/ref_keyword_as.asp) | To create an alias |
| [assert](https://www.w3schools.com/python/ref_keyword_assert.asp) | For debugging |
| [break](https://www.w3schools.com/python/ref_keyword_break.asp) | To break out of a loop |
| [class](https://www.w3schools.com/python/ref_keyword_class.asp) | To define a class |
| [continue](https://www.w3schools.com/python/ref_keyword_continue.asp) | To continue to the next iteration of a loop |
| [def](https://www.w3schools.com/python/ref_keyword_def.asp) | To define a function |
| [del](https://www.w3schools.com/python/ref_keyword_del.asp) | To delete an object |
| [elif](https://www.w3schools.com/python/ref_keyword_elif.asp) | Used in conditional statements, same as else if |
| [else](https://www.w3schools.com/python/ref_keyword_else.asp) | Used in conditional statements |
| [except](https://www.w3schools.com/python/ref_keyword_except.asp) | Used with exceptions, what to do when an exception occurs |
| [False](https://www.w3schools.com/python/ref_keyword_false.asp) | Boolean value, result of comparison operations |
| [finally](https://www.w3schools.com/python/ref_keyword_finally.asp) | Used with exceptions, a block of code that will be executed no matter if there is an exception or not |
| [for](https://www.w3schools.com/python/ref_keyword_for.asp) | To create a for loop |
| [from](https://www.w3schools.com/python/ref_keyword_from.asp) | To import specific parts of a module |
| [global](https://www.w3schools.com/python/ref_keyword_global.asp) | To declare a global variable |
| [if](https://www.w3schools.com/python/ref_keyword_if.asp) | To make a conditional statement |
| [import](https://www.w3schools.com/python/ref_keyword_import.asp) | To import a module |
| [in](https://www.w3schools.com/python/ref_keyword_in.asp) | To check if a value is present in a list, tuple, etc. |
| [is](https://www.w3schools.com/python/ref_keyword_is.asp) | To test if two variables are equal |
| [lambda](https://www.w3schools.com/python/ref_keyword_lambda.asp) | To create an anonymous function |
| [None](https://www.w3schools.com/python/ref_keyword_none.asp) | Represents a null value |
| [nonlocal](https://www.w3schools.com/python/ref_keyword_nonlocal.asp) | To declare a non-local variable |
| [not](https://www.w3schools.com/python/ref_keyword_not.asp) | A logical operator |
| [or](https://www.w3schools.com/python/ref_keyword_or.asp) | A logical operator |
| [pass](https://www.w3schools.com/python/ref_keyword_pass.asp) | A null statement, a statement that will do nothing |
| [raise](https://www.w3schools.com/python/ref_keyword_raise.asp) | To raise an exception |
| [return](https://www.w3schools.com/python/ref_keyword_return.asp) | To exit a function and return a value |
| [True](https://www.w3schools.com/python/ref_keyword_true.asp) | Boolean value, result of comparison operations |
| [try](https://www.w3schools.com/python/ref_keyword_try.asp) | To make a try...except statement |
| [while](https://www.w3schools.com/python/ref_keyword_while.asp) | To create a while loop |
| With | Used to simplify exception handling |
| [yield](https://www.w3schools.com/python/ref_keyword_yield.asp) | To return a list of values from a generator |

**Discuss Identifiers in Python?**

All the variables, class, object, functions, lists, dictionaries etc. in Python are together termed as Python Identifiers. Identifiers are the basis of any Python program. Almost every Python Code uses some or other identifiers.

**Rules for using Python Identifiers:**

* It cannot be a reserved python keyword.
* It can be a combination of A-Z, a-z, 0-9, or underscore.
* It should start with an alphabet character or an underscore ( \_ ).
* It should not contain any special character other than an underscore ( \_ ).
* An identifier name in Python is case-sensitive i.e, sum and Sum are two different identifier.
* It should not contain white space.

**Valid identifiers:**

* var1
* \_var1
* \_1\_var
* var\_1

**Invalid Identifiers**

* !var1
* 1var
* 1\_var
* var#1
* var 1

**Write about python Variables?**

Variables are containers for storing data values.

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

Python has no command for declaring a variable.

A variable is created the moment you first assign a value to it.

Variables do not need to be declared with any particular type, and can even change type after they have been set.

Ex:

x = 5

print(x)  
x = "John"  
print(x)

Output:

5

John

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 name must not be similar to any keyword defined in the language.
* Identifier names are case sensitive; for example, myname, 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 explicitly variable in Python. When we assign any value to the variable, that variable is declared automatically.

The equal (=) operator is used to assign value to a variable.

**Write about Comment statements in Python.**

Comments can be used to explain Python code.

Comments can be used to make the code more readable.

Comments can be used to prevent execution when testing code.

Comments can be used to: Explain the line of code, Provide notes for future reference, Describe how you solved a coding problem, Explain complicated code, and Avoid executing a specific part of code while testing.

Comments in Python are pieces of text that are not executed by the Python interpreter and are used to explain code:

* **Single-line comments**: Start with a hash symbol (#) and continue to the end of the line. You can place the # at the beginning of the line or after the code on the same line.
* **Multi-line comments**: Use multiple # characters or triple double quotes ("""), which can span multiple lines.
* **String literals**: Can be used as comments if they are not assigned to a variable.

## **Creating a Comment**

C omments starts with a #, and Python will ignore them:

#This is a comment

Comments can be placed at the end of a line, and Python will ignore the rest of the line:

Ex:

print("Hello, World!") #This is a comment

## **Multi Line Comments**

Python does not really have a syntax for multi line comments.

To add a multiline comment you could insert a # for each line:

Ex:

#This is a comment

#written in

#more than just one line

print("Hello, World!")

Or, not quite as intended, you can use a multiline string.

Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code, and place your comment inside it:

Ex:

"""  
This is a comment  
written in  
more than just one line  
"""  
print("Hello, World!")

**Explain different datatypes supported by Python.**

Variables and constants can store data of different types.

Python has the following data types built-in by default, in these categories:

|  |  |
| --- | --- |
| Text Type: | Str |
| Numeric Types: | int, float, complex |
| Sequence Types: | list, tuple, range |
| Mapping Type: | Dict |
| Set Types: | set, frozenset |
| Boolean Type: | Bool |
| Binary Types: | bytes, bytearray, memoryview |
| None Type: | NoneType |

Getting the datatype:

You can get the data type of any object by using the **type()** function:

**Integers** – This data type is represented with the help of int class. It consists of positive or negative whole numbers (without fraction or decimal). In Python, there’s no limit to how long integer values are often.

Ex:

x = 20

print(x)

print(type(x))

Output:

20

<class ‘int’>

**Float** – This type is represented by the float class. It is specified by a decimal point.

Ex:

x = 20.5

print(x)

print(type(x))

Output:

20.5

<class 'float'>

**Complex Numbers** – Complex numbers are represented by complex classes. It is specified as (real part) + (imaginary part)j, For example – 4+5j.

Ex: x = 1j

print(x)

print(type(x))

Output:

lj

<class 'complex'>

**Strings:**

The string is a sequence of Unicode characters. A string may be a collection of one or more characters in a a quotation mark, double-quote, or triple quote. It can be represented using as str class.

Ex:

x = "Hello World"

print(x)

print(type(x))

Output:

Hello World

<class ‘str’>

### **List:-**A list is created by placing all the items (elements) inside square brackets [ ], separated by commas.

It can have any number of items and they may or may not be of different types (integer, float, string, etc.).

A list is mutable, which suggests we will modify the list

Ex:

x = ["apple", "banana", "cherry"]

print(x)

print(type(x))

Output:

['apple', 'banana', 'cherry']

<class 'list'>

### **Tuple:-**A tuple is defined as an ordered collection of Python objects. The only difference between tuple and list is that tuples are immutable i.e. tuples can’t be modified after it’s created. It is represented by tuple class. we can represent tuples using parentheses ( ).

Ex:

x = ("apple", "banana", "cherry")

print(x)

print(type(x))

Output:

('apple', 'banana', 'cherry')

<class 'tuple'>

**Set:-**A set is an unordered collection of items. Every set element is exclusive (no duplicates) and must be immutable (cannot be changed).

Ex:

x = {"apple", "banana", "cherry"}

print(x)

print(type(x))

Output:

{'banana', 'cherry', 'apple'}

<class 'set'>

### **Dictionary:-**In Python, Dictionary is an unordered collection of data values, which is used to store data values like a map, which, unlike other Data Types that hold only a single value as an element, a Dictionary consists of key-value pair. Key-value is provided within the dictionary to form it more optimized. In the representation of a dictionary data type, each key-value pair during a Dictionary is separated by a colon: whereas each key’s separated by a ‘comma’.

Ex:

x = {"name" : "John", "age" : 36}

print(x)

print(type(x))

Output:

{'name': 'John', 'age': 36}

<class 'dict'>

**Range**: Range is used to specify range of values.

Ex:

x = range(6)

print(x)

print(type(x))

Output:

range(0, 6)

<class 'range'>

**Frozenset :**Frozenset is unordered and immutable.

Ex:

x = frozenset({"apple", "banana", "cherry"})

print(x)

print(type(x))

Output:

frozenset({'apple', 'cherry', 'banana'})

<class 'frozenset'>

**Boolean:** boolen is used to store True/False values.

Ex:

x = True

print(x)

print(type(x))

output:

True

<class 'bool'>

**None:**None is used to define a null value.

Ex:

x = None

print(x)

print(type(x))

output:

None

<class 'NoneType'>

**Discuss Python operators in detail.**

Operators are used to perform operations on variables and values.

Python divides the operators in the following groups:

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

## **Python Arithmetic Operators:**

Arithmetic operators are used with numeric values to perform common mathematical operations:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Example** |
| + | Addition | x + y |
| - | Subtraction | x - y |
| \* | Multiplication | x \* y |
| / | Division | x / y |
| % | Modulus | x % y |
| \*\* | Exponentiation | x \*\* y |
| // | Floor division | x // y |

Ex- x = 2

y = 5

print(x \*\* y)

#same as 2\*2\*2\*2\*2

Output:- 32

Ex:- x = 15

y = 2

print(x // y)

#the floor division // rounds the result down to the nearest whole number

Output:- 7

## **Python Assignment Operators:**

Assignment operators are used to assign values to variables:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| = | x = 5 | | | | x = 5 |
| += | x += 3 | | | | x = x + 3 |
| -= | x -= 3 | |  | | x = x - 3 |
|  |  | |  | |  |
| \*= | | x \*= 3 | | x = x \* 3 | |
| /= | | x /= 3 | | x = x / 3 | |
| %= | | x %= 3 | | x = x % 3 | |
| \*\*= | | x \*\*= 3 | | x = x \*\* 3 | |
| //= | | x //= 3 | | x = x // 3 | |

Ex:- x = 5

x \*\*= 3

print(x)

Output:- 125

## **Python Comparison Operators:-**

Comparison operators are used to compare two values:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Example** |
| == | Equal | x == y |
| != | Not equal | x != y |
| > | Greater than | x > y |
| < | Less than | x < y |
| >= | Greater than or equal to | x >= y |
| <= | Less than or equal to | x <= y |

Ex:- x = 5

y = 3

print(x == y)

# returns False because 5 is not equal to 3

Output:- False

## **Python Logical Operators:-**

Logical operators are used to combine conditional statements:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| and | Returns True if both statements are true | x < 5 and  x < 10 |
| Or | Returns True if one of the statements is true | x < 5 or x < 4 |
| Not | Reverse the result, returns False  if the result is true | not(x < 5 and x < 10) |

Ex:- x = 5

print(not(x > 3 and x < 10))

# returns True because 5 is greater than 3 AND 5 is less than 10

Output:- False

## **Python Identity Operators:**

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| is | Returns True if both variables are the same object | x is y |
| is not | Returns True if both variables are not the same object | x is not y |

Ex;- x = ["apple", "banana"]

y = ["apple", "banana"]

z = x

print(x is z)

# returns True because z is the same object as x

print(x is y)

# returns False because x is not the same object as y, even if they have the same content

print(x == y)

# to demonstrate the difference betweeen "is" and "==": this comparison returns True because x is equal to y

Output:- True

False

True

Python Membership Operators:

Membership operators are used to test if a sequence is presented in an object:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| in | Returns True if a sequence with the specified value  is present in the object | x in y |
| not in | Returns True if a sequence with the specified value  is not present in the object | x not in y |

Ex;- x = ["apple", "banana"]

print("banana" in x)

# returns True because a sequence with the value "banana" is in the list

Output:- True

## **Python Bitwise Operators:**

Bitwise operators are used to compare (binary) numbers:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Description** |
| & | AND | Sets each bit to 1 if both bits are 1 |
| | | OR | Sets each bit to 1 if one of two bits is 1 |
| ^ | XOR | Sets each bit to 1 if only one of two bits is 1 |
| ~ | NOT | Inverts all the bits |
| << | Zero fill left shift | Shift left by pushing zeros in from the right and let the leftmost bits fall off |
| >> | Signed right shift | Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off |

**Explain standard I/O operations in Python?**

**Python print() Function:**

**Printing to the Screen**

The simplest way to produce output is using the *print* statement where you can pass zero or more expressions separated by commas. This function converts the expressions you pass into a string and writes the result to standard output .

**Ex:** print(“Welcome to the world of Python “)

**Python input() Function:**

The input() function allows user input.

**Ex:**

print("Enter your name:")

x = input()

print("Hello, " + x)

**Discuss Python Type Conversion (Type casting)?**

Type Casting is the method to convert the variable data type into a certain data type .

There can be two types of Type Casting in Python –

1)Implicit Type Casting

2)Explicit Type Casting

**Implicit Type Conversion:**

In this, methods, Python converts data type into another data type automatically. In this process, users don’t have to involve in this process.

Ex:a = 7

print(type(a))

Output:<class 'int'>

Ex:b = 3.0

print(type(b))

Output: <class 'float'>

Ex: c = a + b

print(c)

print(type(c))

Output:<10.0

<class 'float'>

**Explicit Type Casting:**

In this method, Python need user involvement to convert the variable data type into certain data type.

Mainly in type casting can be done with these data type function:

**Int()** : Int() function take float or string as an argument and return int type object.

**float()** : float() function take int or string as an argument and return float type object.

**str()** : str() function take float or int as an argument and return string type object.

**Typecasting to integer:**

x = int(1)

y = int(2.8)

z = int("3")

print(x)

print(y)

print(z)

**Output:**

1

2

3

**Typecasting to float:**

x = float(1)

y = float(2.8)

z = float("3")

w = float("4.2")

print(x)

print(y)

print(z)

print(w)

**Output:**

1.0

2.8

3.0

4.2

**Typecasting to string:**

x = str("s1")

y = str(2)

z = str(3.0)

**Explain Debugging in Python in detail.**

Debugging in Python is the process of identifying and fixing errors in your code so that your program runs smoothly.

Errors are problems in a program that causes the program to stop its execution. On the other hand, exceptions are raised when some internal events change the program’s normal flow.

**[[[Syntax Errors in Python**

When the proper syntax of the language is not followed then a syntax error is thrown.

**Logical Errors (Exception)**

A logical error in Python, or in any programming language, is a type of bug that occurs when a program runs without crashing but produces incorrect or unintended results. Logical errors are mistakes in the program’s logic that lead to incorrect behavior or output, despite the syntax being correct.

**Characteristics of Logical Errors**

1. **No Syntax Error**: The code runs successfully without any syntax errors.
2. **Unexpected Output**: The program produces output that is different from what is expected.
3. **Difficult to Detect**: Logical errors can be subtle and are often harder to identify and fix compared to syntax errors because the program appears to run correctly.
4. **Varied Causes**: They can arise from incorrect assumptions, faulty logic, improper use of operators, or incorrect sequence of instructions.]]]

**some tools and techniques you can use to debug Python.**

* **Breakpoints**

Pause the debugger before it executes certain lines of code. You can add and remove breakpoints before launching the debugger or while it's paused.

* **PDB**

A built-in source code tool that can help correct errors and inaccuracies. You can use pdb.set\_trace() or pdb.breakpoint() to enter debugger mode.

* **Visual Studio Code (VS Code)**

A user-friendly way to debug Python code. You can set breakpoints and step through code to understand how your code is executing.

* **Debugger extensions**

The debugpy extension supports debugging for various types of Python applications.

* **Standalone UI applications**

These applications are designed specifically for debugging. Examples include PuDB and Winpdb.

* **Code-in tools**

Set breakpoints by adding temporary code to your program.

* **IDEs**

Modern development environments like PyCharm, Visual Studio, and PyDev have integrated debugging tools.

The goal of debugging is to discover issues, understand how each line of code affects the corrupted flow, and implement a better solution.

**Explain Indentation in Python?**

Indentation refers to the spaces at the beginning of a code line.

Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.

Python uses indentation to indicate a block of code.

In Python, indentation is the use of spaces or tabs at the beginning of a line of code to indicate a block of code:

Indentation is a requirement in Python, not just a convention, and is essential for code readability. Here are some rules for indentation in Python:

* **Use the same number of spaces**: The number of spaces you use is up to you, but you must use the same number of spaces in the same block of code.
* **Use at least one space**: You must use at least one space.
* **Use a colon**: A colon is used to indicate the start of a new block of code.
* **Use the first unindented statement to end a block**: The first unindented statement indicates the end of a block.
* **Indent each line of code**: Each line of code must be indented by the same amount of whitespace.
* **Don't ignore indentation**: Although indentation may be ignored in a continuous line, it is a good practice to indent.

If you skip indentation or use the wrong number of spaces in the same block of code, Python will give you an IndentationError.

Ex:

if 5 > 2:  
  print("Five is greater than two!")

Python will give you an error if you skip the indentation.

if 5 > 2:  
print("Five is greater than two!")

**Discuss conditional(branching) statements with suitable exapmples.**

|  |  |
| --- | --- |
| **Statement** | **Description** |
|  |  |
| If Statement | The if statement is used to test a specific condition. If the condition is true, a block of code (if-block) will be executed. |
| If - else Statement | The if-else statement is similar to if statement except the fact that, it also provides the block of the code for the false case of the condition to be checked. If the condition provided in the if statement is false, then the else statement will be executed. |
| Nested if Statement | Nested if statements enable us to use if ? else statement inside an outer if statement. |

**Syntax of if...elif...else:**

if condition1:

statements…

elif condition2:

statements…

else:

statements…

**Ex for if..statement:**

a = 200

b = 33

if b > a:

print("b is greater than a")

elif a == b:

print("a and b are equal")

else:

print("a is greater than b")

**output:-**

​a is greater than b

**Ex for Nested if..**

x = 41

if x > 10:

print("Above ten,")

if x > 20:

print("and also above 20!")

else:

print("but not above 20.")

**output:-**

Above ten,

and also above 20!

## **The pass Statement**

if statements cannot be empty, but if you for some reason have an if statement with no content, put in the pass statement to avoid getting an error.

a = 33  
b = 200  
if b > a:  
  pass

## **Explain Repetition(Python Loops)in detail?**

Python has two primitive loop commands:

* while loops
* for loops

## **The while Loop:**

With the while loop we can execute a set of statements as long as a condition is true.

**Syntax for while loop:**

while condition:

statements…

i = 1

while i < 6:

print(i)

i += 1

**output:-**

​1

2

3

4

5

## **The break Statement**

With the break statement we can stop the loop even if the while condition is true:

i = 1

while i < 6:

print(i)

i += 1

**output:-**

​1

2

3

4

5

## **The continue Statement**

With the continue statement we can stop the current iteration, and continue with the next:

i = 0

while i < 6:

i += 1

if i == 3:

continue

print(i)

​# Note that number 3 is missing in the result

**output:-**

​1

2

4

5

6

## **The else Statement**

With the else statement we can run a block of code once when the condition no longer is true:

i = 1

while i < 6:

print(i)

i += 1

else:

print("i is no longer less than 6")

**output:-**

​1

2

3

4

5

i is no longer less than 6

**for loop:**

for loops are used when you have a block of code which you want to repeat a **fixed number of times**.

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

**Syntax for for loop:**

for val in sequence:

statements…

**Ex1:**for x in "banana":

print(x)

**output:-**

​b

a

n

a

n

a

**Ex2:**

fruits = ["apple", "banana", "cherry"]

for x in fruits:

print(x)

**output:-**

​apple

banana

cherry

## **The break Statement:**

With the break statement we can stop the loop before it has looped through all the items:

**Ex:**

fruits = ["apple", "banana", "cherry"]

for x in fruits:

print(x)

if x == "banana":

break

**output:-**

​apple

banana

## **The continue Statement:**

With the continue statement we can stop the current iteration of the loop, and continue with the next:

**Ex:**

fruits = ["apple", "banana", "cherry"]

for x in fruits:

if x == "banana":

continue

print(x)

**output:-**

**​**apple

cherry

## **The range() Function:**

To loop through a set of code a specified number of times, we can use the range() function,

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

**Ex:**

for x in range(6):

print(x)

**output:**

0

1

2

3

4

5

## **Else in For Loop:**

The else keyword in a for loop specifies a block of code to be executed when the loop is finished:

**Ex:**

for x in range(6):

print(x)

else:

print("Finally finished!")

​**output :-**

0

1

2

3

4

5

Finally finished!

## **Nested Loops:**

A nested loop is a loop inside a loop.

The "inner loop" will be executed one time for each iteration of the "outer loop":

**Ex:**

adj = ["red", "big", "tasty"]

fruits = ["apple", "banana", "cherry"]

​for x in adj:

for y in fruits:

print(x, y)

**output:-**

​red apple

red banana

red cherry

big apple

big banana

big cherry

tasty apple

tasty banana

tasty cherry

## **The pass Statement:**

for loops cannot be empty, but if you for some reason have a for loop with no content, put in the pass statement to avoid getting an error.

**Ex:**

for x in [0, 1, 2]:

pass

# having an empty for loop like this, would raise an error without the pass statement

**Discuss python Strings , string operations and String Methods indetails.**

Strings in python are surrounded by either single quotation marks, or double quotation marks.

'hello' is the same as "hello".

You can display a string literal with the print() function:

a = "Hello"

print(a)

output: Hello

**Multiline Strings:**

You can assign a multiline string to a variable by using three quotes:

a = """we are using

python strings"""  
print(a)

Output: we are using

Python strings

**Strings are Arrays:**

a = "Hello, World!"  
print(a[1])

output: e

**Looping Through a String(Traversing a string):**

Since strings are arrays, we can loop through the characters in a string, with a for loop.

for x in "banana":  
  print(x)

output:b

a

n

a

n

a

**String Length:**

To get the length of a string, use the len() function.

a = "Hello, World!"  
print(len(a))

output: 13

**Check String:**

To check if a certain phrase or character is present in a string, we can use the keyword in.

txt = "The best things in life are free!"

print("free" in txt)

output: True

**Use it in an if statement:**

txt = "The best things in life are free!"  
if "free" in txt:  
  print("Yes, 'free' is present.")

output: Yes, 'free' is present.

**Slicing Strings:**

You can return a range of characters by using the slice syntax.

Specify the start index and the end index, separated by a colon, to return a part of the string.

b = "Hello, World!"

print(b[2:5])

output:llo

b = "Hello, World!"  
print(b[:5]) #slice from the start

print(b[2:]) #slice to the end

print(b[-5:-2]) #negative indexing

**Python - Modify Strings:**

**Upper Case:**

The upper() method returns the string in upper case:

a = "Hello, World!"  
print(a.upper())

output: HELLO,WORLD

**Lower Case:**

The lower() method returns the string in lower case:

a = "Hello, World!"  
print(a.lower())

output: hello,world

**Remove Whitespace:**

Whitespace is the space before and/or after the actual text, and very often you want to remove this space.

The strip() method removes any whitespace from the beginning or the end:

a = " Hello, World! "  
print(a.strip()) # returns "Hello, World!"

**Replace String:**

The replace() method replaces a string with another string:

a = "Tom and Jerry"  
print(a.replace("T", "J"))

output: Jom and Jerry

**Split String:**

The split() method returns a list where the text between the specified separator becomes the list items.

a = "Hello, World!"  
print(a.split(",")) # returns ['Hello', ' World!']

['Hello', ' World!']

String Concatenation

**String Concatenation:**

To concatenate, or combine, two strings you can use the + operator.

a = "Hello"  
b = "World"  
c = a + b  
print(c)

output: HelloWorld

**To add a space between them, add a " ":**

a = "Hello"  
b = "World"  
c = a + " " + b  
print(c)

output: Hello World

**String Format:**

age = 36  
txt = "My name is John, I am " + age  
print(txt)

But we can combine strings and numbers by using the format() method!

The format() method takes the passed arguments, formats them, and places them in the string where the placeholders {} are:

age = 36  
txt = "My name is John, and I am {}"  
print(txt.format(age))

output: My name is John, and I am 36

The format() method takes unlimited number of arguments, and are placed into the respective placeholders:

quantity = 3  
itemno = 567  
price = 49.95  
myorder = "I want {} pieces of item {} for {} dollars."  
print(myorder.format(quantity, itemno, price))

output: I want 3 pieces of item 567 for 49.95 dollars.

quantity = 3

itemno = 567

price = 49.95

myorder = "I want to pay {2} dollars for {0} pieces of item {1}."

print(myorder.format(quantity, itemno, price))

output: I want 3 pieces of item 567 for 49.95 dollars.

**Escape Character**

To insert characters that are illegal in a string, use an escape character.

An escape character is a backslash \ followed by the character you want to insert.

An example of an illegal character is a double quote inside a string that is surrounded by double quotes:

You will get an error if you use double quotes inside a string that is surrounded by double quotes:

txt = "We are the so-called "Vikings" from the north."

To fix this problem, use the escape character \":

txt = "We are the so-called \"Vikings\" from the north."

print(txt)

output:

We are the so-called "Vikings" from the north.

|  |  |  |
| --- | --- | --- |
| **Code** | **Result** | **Try it** |
| \' | Single Quote | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_string_escape2) |
| \\ | Backslash | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_string_backslash) |
| \n | New Line | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_string_newline) |
| \r | Carriage Return | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_string_r) |
| \t | Tab | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_string_t) |
| \b | Backspace |  |

**String Methods**

Python has a set of built-in methods that you can use on strings.

All string methods returns new values. They do not change the original string.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [capitalize()](https://www.w3schools.com/python/ref_string_capitalize.asp) | Converts the first character to upper case |
| [casefold()](https://www.w3schools.com/python/ref_string_casefold.asp) | Converts string into lower case |
| [center()](https://www.w3schools.com/python/ref_string_center.asp) | Returns a centered string |
| [count()](https://www.w3schools.com/python/ref_string_count.asp) | Returns the number of times a specified value occurs in a string |
| [encode()](https://www.w3schools.com/python/ref_string_encode.asp) | Returns an encoded version of the string |
| [endswith()](https://www.w3schools.com/python/ref_string_endswith.asp) | Returns true if the string ends with the specified value |
| [expandtabs()](https://www.w3schools.com/python/ref_string_expandtabs.asp) | Sets the tab size of the string |
| [find()](https://www.w3schools.com/python/ref_string_find.asp) | Searches the string for a specified value and returns the position of where it was found |
| [format()](https://www.w3schools.com/python/ref_string_format.asp) | Formats specified values in a string |
| format\_map() | Formats specified values in a string |
| [index()](https://www.w3schools.com/python/ref_string_index.asp) | Searches the string for a specified value and returns the position of where it was found |
| [isalnum()](https://www.w3schools.com/python/ref_string_isalnum.asp) | Returns True if all characters in the string are alphanumeric |
| [isalpha()](https://www.w3schools.com/python/ref_string_isalpha.asp) | Returns True if all characters in the string are in the alphabet |
| [isdecimal()](https://www.w3schools.com/python/ref_string_isdecimal.asp) | Returns True if all characters in the string are decimals |
| [isdigit()](https://www.w3schools.com/python/ref_string_isdigit.asp) | Returns True if all characters in the string are digits |
| [isidentifier()](https://www.w3schools.com/python/ref_string_isidentifier.asp) | Returns True if the string is an identifier |
| [islower()](https://www.w3schools.com/python/ref_string_islower.asp) | Returns True if all characters in the string are lower case |
| [isnumeric()](https://www.w3schools.com/python/ref_string_isnumeric.asp) | Returns True if all characters in the string are numeric |
| [isprintable()](https://www.w3schools.com/python/ref_string_isprintable.asp) | Returns True if all characters in the string are printable |
| [isspace()](https://www.w3schools.com/python/ref_string_isspace.asp) | Returns True if all characters in the string are whitespaces |
| [istitle()](https://www.w3schools.com/python/ref_string_istitle.asp) | Returns True if the string follows the rules of a title |
| [isupper()](https://www.w3schools.com/python/ref_string_isupper.asp) | Returns True if all characters in the string are upper case |
| [join()](https://www.w3schools.com/python/ref_string_join.asp) | Joins the elements of an iterable to the end of the string |
| [ljust()](https://www.w3schools.com/python/ref_string_ljust.asp) | Returns a left justified version of the string |
| [lower()](https://www.w3schools.com/python/ref_string_lower.asp) | Converts a string into lower case |
| [lstrip()](https://www.w3schools.com/python/ref_string_lstrip.asp) | Returns a left trim version of the string |
| [maketrans()](https://www.w3schools.com/python/ref_string_maketrans.asp) | Returns a translation table to be used in translations |
| [partition()](https://www.w3schools.com/python/ref_string_partition.asp) | Returns a tuple where the string is parted into three parts |
| [replace()](https://www.w3schools.com/python/ref_string_replace.asp) | Returns a string where a specified value is replaced with a specified value |
| [rfind()](https://www.w3schools.com/python/ref_string_rfind.asp) | Searches the string for a specified value and returns the last position of where it was found |
| [rindex()](https://www.w3schools.com/python/ref_string_rindex.asp) | Searches the string for a specified value and returns the last position of where it was found |
| [rjust()](https://www.w3schools.com/python/ref_string_rjust.asp) | Returns a right justified version of the string |
| [rpartition()](https://www.w3schools.com/python/ref_string_rpartition.asp) | Returns a tuple where the string is parted into three parts |
| [rsplit()](https://www.w3schools.com/python/ref_string_rsplit.asp) | Splits the string at the specified separator, and returns a list |
| [rstrip()](https://www.w3schools.com/python/ref_string_rstrip.asp) | Returns a right trim version of the string |
| [split()](https://www.w3schools.com/python/ref_string_split.asp) | Splits the string at the specified separator, and returns a list |
| [splitlines()](https://www.w3schools.com/python/ref_string_splitlines.asp) | Splits the string at line breaks and returns a list |
| [startswith()](https://www.w3schools.com/python/ref_string_startswith.asp) | Returns true if the string starts with the specified value |
| [strip()](https://www.w3schools.com/python/ref_string_strip.asp) | Returns a trimmed version of the string |
| [swapcase()](https://www.w3schools.com/python/ref_string_swapcase.asp) | Swaps cases, lower case becomes upper case and vice versa |
| [title()](https://www.w3schools.com/python/ref_string_title.asp) | Converts the first character of each word to upper case |
| [translate()](https://www.w3schools.com/python/ref_string_translate.asp) | Returns a translated string |
| [upper()](https://www.w3schools.com/python/ref_string_upper.asp) | Converts a string into upper case |
| [zfill()](https://www.w3schools.com/python/ref_string_zfill.asp) | Fills the string with a specified number of 0 values at the beginning |

**Explain String Comparison in Python .**

**Method 1: Using Relational Operators**

The relational operators compare the Unicode values of the characters of the strings from the zeroth index till the end of the string. It then returns a boolean value according to the operator used.

print("Sdhr" == "Sdhr")

print("Sdhr" < "sdhr")

print("Sdhr" > "sdhr)

print("Sdhr" != "Sdhr")

**Output:**

True

True

False

False

**Method 2: Using is and is not**

The == operator compares the values of both the operands and checks for value equality. Whereas is operator checks whether both the operands refer to the same object or not. The same is the case for != and is not.

str1 = "Sdhr"

str2 = "Sdhr"

str3 = str1

print("ID of str1 =", hex(id(str1)))

print("ID of str2 =", hex(id(str2)))

print("ID of str3 =", hex(id(str3)))

print(str1 is str1)

print(str1 is str2)

print(str1 is str3)

str1 += "s"

str4 = "Sdhr"

print("\nID of changed str1 =", hex(id(str1)))

print("ID of str4 =", hex(id(str4)))

print(str1 is str4)

**output:**

ID of str1 = 0x7f6037051570

ID of str2 = 0x7f6037051570

ID of str3 = 0x7f6037051570

True

True

True

ID of changed str1 = 0x7f60356137d8

ID of str4 = 0x7f60356137a0

False

The object ID of the strings may vary on different machines. The object IDs of str1, str2 and str3 were the same therefore they the result is True in all the cases. After the object id of str1 is changed, the result of str1 and str2 will be false. Even after creating str4 with the same contents as in the new str1, the answer will be false as their object IDs are different.

**Explain Traversing a string in Python?**

Traversing a string means accessing all the elements of the string one after the other by using the subscript. A string can be traversed using for loop or while loop.

Strings in python are surrounded by either single quotation marks, or double quotation marks.

'hello' is the same as "hello".

You can display a string literal with the print() function:

a = "Hello"

print(a)

output: Hello

**Multiline Strings**

You can assign a multiline string to a variable by using three quotes:

a = """we are using

python strings"""  
print(a)

Output: we are using

Python strings

**Looping Through a String**

Since strings are arrays, we can loop through the characters in a string, with a for loop.

for x in "banana":  
  print(x)

output:b

a

n

a

n

a