SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+'
Grade Approved by AICTE, New Delhi & Affiliated to Anna University,
Chennai

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND   MACHINE LEARNING

19AMB302-FULL STACK AI

PYTHON PROGRAMMING AND AI

M.POORNIMA DEVI,AP/AIML

- Python is a high-level, multi-paradigm programming language

- As Python is an interpreter-based language, it is easier to learn compared to some of the other mainstream languages.

- Python is a dynamically typed language with very intuitive data types.

File Handling

The key function for working with files in Python is the open() function.

The open() function takes two parameters; *filename*, and *mode*.

There are four different methods (modes) for opening a file:

"r" - Read - Default value. Opens a file for reading, error if the file does not exist

"a" - Append - Opens a file for appending, creates the file if it does not exist

"w" - Write - Opens a file for writing, creates the file if it does not exist

"x" - Create - Creates the specified file, returns an error if the file exists

PYTHON SCRIPT USING VARIABLES AND KEYWORDS

• A script is a Python file that's intended to be run directly. When you run it, it should do something.

• This means that scripts will often contain code written outside the scope of any classes or functions.

• A module is a Python file that's intended to be imported into scripts or other modules.

• To write a Python script, use an ordinary plain text file and add Python instructions. Scripts often make use of Python modules, which contain external functions, classes, and variables. The Python pip package we can download and install modules, while the import command is used to access the contents of a module.

PYTHON SCRIPT USING VARIABLES

•A Python variable is a name given to a memory location.

•Python Variable is containers that store values.

•Python is not "statically typed". We do not need to declare variables before using them or declare their type. A variable is created the moment we first assign a value to it.

•**Example of Variable in Python**

• Python is a representational name that serves as a pointer to an object.

• Once an object is assigned to a variable, it can be referred to by that name.

Here we have stored "**Geeksforgeeks**" in a variable **var**, and when we call its name the stored information will get printed.

Python3

EXAMPLE:

Var = "Geeksforgeeks"

print(Var)

OUTPUT: Geeksforgeeks

1.Rules for Python variables

- A Python variable name must start with a letter or the underscore character.
- A Python variable name cannot start with a number.
- A Python variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _).
- Variable in Python names are case-sensitive (name, Name, and NAME are three different variables).
- The reserved words(keywords) in Python cannot be used to name the variable in Python
- EXAMPLE:

# valid variable name
geeks = 1
Geeks = 2
Ge_e_ks = 5
_geeks = 6
  geeks_ = 7
GEEKS = 8
  print(geeks, Geeks, Ge_e_ks) **Output:**
1 2 5
6 7 8

2.Variables Assignment in Python

•Here, we will define a variable in python. Here, clearly we have assigned a number, a floating point number, and a string to a variable such as age, salary, and name.

•EXAMPLE # An integer assignment

age = 45

A floating point

salary = 1456.8

A string

name = "John"

print(age)

print(salary)

print(name)

Output:

45

1456.8

John

3.Declaration and Initialization of Variables

•Let's see how to declare a variable and how to define a variable and print the variable.

•EXAMPLE:

```
# declaring the var
Number = 100
# display
print( Number)
```

OUTPUT:100

4.Redeclaring variables in Python

•We can re-declare the Python variable once we have declared the variable and define variable in python already.

```
# declaring the var
Number = 100
# display
print("Before declare: ", Number)
# re-declare the var
Number = 120.3
print("After re-declare:", Number)
```

OUTPUT: Before declare: 100

After re-declare: 120.3

5.Python Assign Values to Multiple Variables

•Python allows assigning a single value to several variables simultaneously with "="
operators.
For example:
a = b = c = 10
print(a)
print(b)
print(c)
OUTPUT:10 10 10

6.Assigning different values to multiple variables

•Python allows adding different values in a single line with "," operators.
a, b, c = 1, 20.2, "GeeksforGeeks"
print(a)
print(b)
print(c)
OUTPUT:
1
20.2
GeeksforGeeks

6.Global and Local Python Variables

•**Local variables** in Python are the ones that are defined and declared inside a function. We can not call this variable outside the function.

EXAMPLE:

```python
# This function uses local variable s
def f():
        s = "Welcome geeks"
        print(s)
f()
```

Output:

Welcome geeks

•**Global variables** in Python are the ones that are defined and declared outside a function, and we cannot need to use them inside a function.

•EXAMPLE:

```python
# This function has a variable with
# name same as s
def f():print(s)
# Global scope
s = "I love Geeksforgeeks"
f()
```

THANKYOU

SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+'
Grade Approved by AICTE, New Delhi & Affiliated to Anna University,
Chennai

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND   MACHINE LEARNING

19AMB302-FULL STACK AI

M.POORNIMA DEVI,AP/AIML

PYTHON SCRIPT USING KEYWORDS

Global keyword in Python

• Python global is a keyword that allows a user to modify a variable outside of the current scope.

• It is used to create global variables from a non-global scope i.e inside a function.

• Global keyword is used inside a function only when we want to do assignments or when we want to change a variable.

• Global is not needed for printing and accessing.

• **Rules of global keyword**

• If a variable is assigned a value anywhere within the function's body, it's assumed to be local unless explicitly declared as global.

• Variables that are only referenced inside a function are implicitly global.

• We use a global in Python to use a global variable inside a function.

• There is no need to use a global keyword in Python outside a function.

•**Python Keywords**
Keywords in Python are reserved words that can not be used as a variable name, function name, or any other identifier.
•**List of Keywords in Python**

Keyword	Description	Keyword	Description	Keyword	Description
and	It is a Logical Operator	False	Represents an expression that will result in not being true.	nonlocal	It is a non-local variable
as	It is used to create an alias name	finally	It is used with exceptions	not	It is a Logical Operator
assert	It is used for debugging	for	It is used to create Loop	or	It is a Logical Operator
break	Break out a Loop	from	To import specific parts of a module	pass	pass is used when the user doesn't want any code to execute
class	It is used to define a class	global	It is used to declare a global variable	raise	raise is used to raise exceptions or errors.
continue	Skip the next iteration of a loop	if	To create a Conditional Statement	return	return is used to end the execution
def	It is used to define the Function	import	It is used to import a module	True	Represents an expression that will result in true.
del	It is used to delete an object	is	It is used to test if two variables are equal	try	Try is used to handle errors
elif	Conditional statements, same as else-if	in	To check if a value is present in a Tuple, List, etc.	while	While Loop is used to execute a block of statements
else	It is used in a conditional statement	lambda	Used to create an anonymous function	with	with statement is used in exception handling
except	try-except is used to handle these errors	None	It represents a null value	yield	yield keyword is used to create a generator function

True, False, None Keyword in Python

•**True:** This keyword is used to represent a boolean true. If a statement is true, "True" is printed.

•**False:** This keyword is used to represent a boolean false. If a statement is false, "False" is printed.

•**None:** This is a special constant used to denote a null value or a void. It's important to remember, 0, any empty container(e.g. empty list) does not compute to None. It is an object of its datatype – NoneType. It is not possible to create multiple None objects and can assign them to variables.

True, False, and None Use in Python

False is 0, and True is 1.

True + True + True is 3.

True + False + False is 1.

None isn't equal to 0 or an empty list ([]).

1.And, or, not, is and in keyword implementation in Python

2. Iteration Keywords – for, while, break, continue in Python

3. Conditional keywords in Python- if, else, elseif

4. Structure Keywords in Python : def, class, with, as, pass, lambda

5. class in Python

[class](#) keyword is used to declare user defined classes.

Class Keyword in Python

This code defines a Python class named **Dog** with two class attributes, **attr1** and **attr2**, and a method **fun** that prints these attributes. It creates an object **Rodger** from the Dog class, accesses the class attributes, and calls the method. When executed, it prints the values of **attr1** and `**attr2**, and the method displays these values, resulting in the output.

6.as in Python

as keyword is used to create the alias for the module imported. i.e giving a new name to the imported module. E.g import math as mymath.

as Keyword In Python

•This code uses the Python **math** module, which has been imported with the alias **gfg.**

•It calculates and prints the factorial of 5. The **math.factorial**() function is used to calculate the factorial of a number, and in this case, it calculates the factorial of 5, which is 120.

EXAMPLE:

import math as gfg

print(gfg.factorial(5))

OUTPUT:

120

7.pass in Python

pass is the null statement in python. Nothing happens when this is encountered. This is used to prevent indentation errors and used as a placeholder.

Pass Keyword in Python

The code contains a **for** loop that iterates 10 times with a placeholder statement **'pass'**, indicating no specific action is taken within the loop.

8.Return Keywords in Python- Return, Yield

return : This keyword is used to return from the function.

yield : This keyword is used like return statement but is used to return a generator.

EXAMPLE:

```
# Return keyword
def fun():
        S = 0
for i in range(10):
                S += i
        return S
print(fun())
# Yield Keyword
def fun():
        S = 0
for i in range(10):            S += i
                yield S
for i in fun():
        print(i)
```

OUTPUT:45 0 1 3 6 10 15 21 28 36 45

9.Exception Handling Keywords in Python – try, except, raise, finally, and assert

try : This keyword is used for exception handling, used to catch the errors in the code using the keyword except. Code in "try" block is checked, if there is any type of error, except block is executed.

except : As explained above, this works together with "try" to catch exceptions.

finally : No matter what is result of the "try" block, block termed "finally" is always executed.

raise: We can raise an exception explicitly with the raise keyword

assert: This function is used for **debugging purposes**. Usually used to check the correctness of code. If a statement is evaluated to be true, nothing happens, but when it is false, "**AssertionError**" is raised. One can also **print a message with the error, separated by a comma**.

10.del in Python

•<u>del</u> is used to delete a reference to an object. Any variable or list value can be deleted using del.

del Keyword in Python

•In this code, the variables **my_variable1** and **my_variable2** are initially defined and then deleted using the **del** keyword. When you try to print them after deletion, you will encounter a **NameError** because the variables no longer exist.

EXAMPLE:
```
my_variable1 = 20
my_variable2 = "GeeksForGeeks"
print(my_variable1)
print(my_variable2)
del my_variable1
del my_variable2
print(my_variable1)
print(my_variable2)
```
OUTPUT:20 GeeksForGeeks NameError: name 'my_variable1' is not defined

11.Global, Nonlocal in Python

•**global:** This keyword is used to define a variable inside the function to be of a global scope.

•**non-local :** This keyword works similar to the global, but rather than global, this keyword declares a variable to point to variable of outside enclosing function, in case of nested functions.

•In this code, the **'global'** keyword is used to declare global variables **'a'** and **'b'**.

Then, there's a function **'add'** that adds these global variables and prints the result.

The second part of the code demonstrates the **'nonlocal'** keyword.

•The function **fun** contains a variable **var1**, and within the nested function gun, we use nonlocal to indicate that we want to modify the **var1** defined in the outer function **fun.**

•It increments the value of **var1** and prints it.

EXAMPLE:

```python
a = 15
b = 10
def add():
        c = a + b
        print(c)
add()
def fun():
        var1 = 10

        def gun():
                nonlocal var1

                var1 = var10+ 10
                print(var1)

        gun()
fun()
```

Output:
 25
 20

THANKYOU

SNS COLLEGE OF TECHNOLOGY

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND   MACHINE LEARNING

19AMB302-FULL STACK AI

M.POORNIMA DEVI,AP/AIML

•Workflows can be made with Python and other technologies. Database systems are connectable with Python. Files can also be read and changed by it.

•Big data management and advanced mathematical operations can both be done with Python.

•The syntax of Python is straightforward and resembles that of English. Python's syntax enables programmers to create programmes with fewer lines of code than they would be able to with certain other languages.

•Python operates on an interpreter system, allowing for the immediate execution of written code.

•Python provides a lot of built-in functions that ease the writing of code. In this article, you will learn about **Python's built-in functions**, exploring their various applications and highlighting some of the most commonly used ones.

Python Built-in Functions List

| Function Name	Function Name
[Python abs() Function](#) , Python aiter() Function [Python all() Function](#), [Python any() Function](#)	[Python exec() Function](#), [Python filter() Function](#), [Python float() Function](#)
Python anext() Function, [Python ascii() Function](#), [Python bin() Function](#)	[Python format() Function](#), [Python frozenset() Function](#)
[Python bool() Function](#), [Python breakpoint() Function](#), [Python bytearray() Function](#)	[Python globals() Function](#)
[Python bytes() Function](#), [Python callable() Function](#), [Python chr() Function](#)	[Python hasattr() Function](#)
[Python classmethod() Function](#), [Python compile() Function](#), [Python complex() Function](#),	[Python tuple() Function](#)
[Python delattr() Function](#), [Python dict() Function](#), [Python dir() Function](#)	[Python type() Function](#)
[Python divmod() Function](#), [Python enumerate() Function](#), [Python eval() Function](#)	

STRINGS DIFFERENT LITERALS

•A literal in Python is a syntax that is used to completely express a fixed value of a specific data type.

•Literals are constants that are self-explanatory and don't need to be computed or evaluated.

•They are used to provide variable values or to directly utilize them in expressions. Generally, literals are a notation for representing a fixed value in source code. They can also be defined as raw values or data given in variables or constants.

• In this article, we will explore the different types of literals in Python, along with examples to demonstrate their usage.
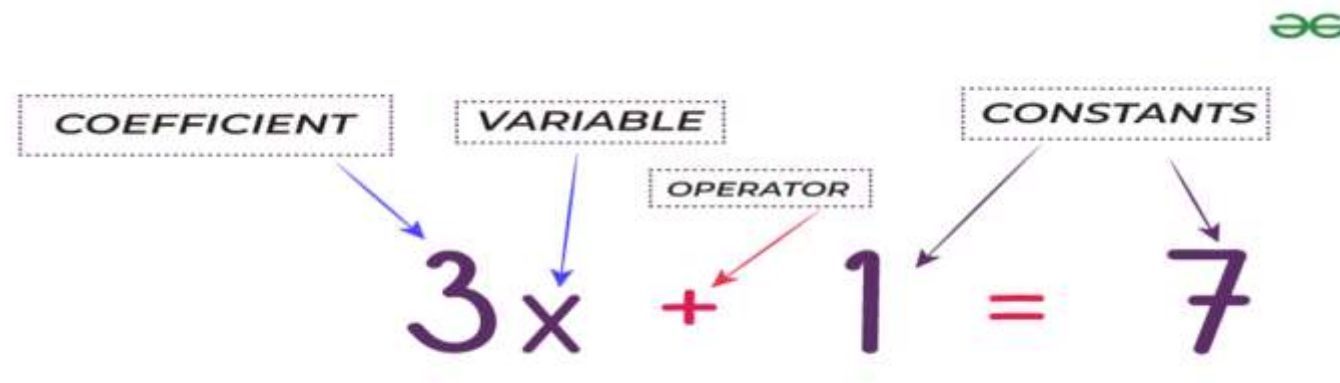
Types of Literals in Python

Python supports various types of literals, such as numeric literals, string literals, Boolean literals, and more. Let's explore different types of literals in Python with examples:

1.String literals
2.Character literal
3.Numeric literals
4.Boolean literals
5.literal Collections
6.Special literals

1.Python String Literals

•A string is literal and can be created by writing a text(a group of Characters) surrounded by a single('), double("), or triple quotes.
• We can write multi-line strings or display them in the desired way by using triple quotes.
•Here **geekforgeeks** is a string literal that is assigned to a variable(s). Here is an example of a Python string literal.

EXAMPLE:

```python
# in single quote
s = 'geekforgeeks'
# in double quotes
t = "geekforgeeks"
# multi-line String
m = '''geek
            for
        geeks'''

print(s)
print(t)
print(m)
```

OUTPUT:
geekforgeeks
geekforgeeks
geek
for
geeks

2.Python Character literal

• It is also a type of Python string literal where a single character is surrounded by single or double quotes.
EXAMPLE:
# character literal in single quote
v = 'n'
# character literal in double quotes
w = "a"
print(v)
print(w)
Output
n
a
3. Python Numeric literal
They are immutable and there are three types of numeric literal:
Integer
Float
Complex

Integer

•Both positive and negative numbers including 0. There should not be any fractional part.

•In this example, We assigned integer literals (0b10100, 50, 0o320, 0x12b) into different variables. Here, '**a**' is a binary literal, '**b**' is a decimal literal, '**c**' is an octal literal, and '**d**' is a hexadecimal literal.

•But on using the print function to display a value or to get the output they were converted into decimal.

•EXAMPLE:

```
# integer literal
# Binary Literals
a = 0b10100
# Decimal Literal
b = 50
# Octal Literal
c = 0o320
# Hexadecimal Literal
d = 0x12b
print(a, b, c, d)
```
OUTPUT:20 50 208 299

Float

•These are real numbers having both integer and fractional parts. In this example, 24.8 and 45.0 are floating-point literals because both 24.8 and 45.0 are floating-point numbers.

 EXAMPLE:

Float Literal

e = 24.8

f = 45.0

print(e, f) **Output**

24.8 45.0

Complex

•The numerals will be in the form of **a + bj**, where '**a**' is the real part and '**b**' is the complex part. **Numeric literal [Complex]**

EXAMPLE:

 z = 7 + 5j

real part is 0 here.

k = 7j

print(z, k)

OUTPUT:(7+5j) 7j

Python Boolean literal
•There are only two [Boolean](#) literals in Python. They are **true** and **false**.
•In Python, **True** represents the value as **1,** and **False** represents the value as **0**. In this example '**a**' is **True** and '**b**' is **False** because 1 is equal to True.
EXAMPLE:
a = (1 == True)
b = (1 == False)
c = True + 3
d = False + 7
print("a is", a)
print("b is", b)
print("c:", c)
print("d:", d)
OUTPUT:
a is True
b is False
c: 4
d: 7

Python literal collections

Python provides four different types of literal collections:

1. List literals
2. Tuple literals
3. Dict literals
4. Set literals

1. List literal

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

EXAMPLE:

```
number = [1, 2, 3, 4, 5]
name = ['Amit', 'kabir', 'bhaskar', 2]
print(number)
print(name)
```

OUTPUT:

```
[1, 2, 3, 4, 5]
['Amit', 'kabir', 'bhaskar', 2]
```

2.Tuple literal

•A tuple is a collection of different data-type. It is enclosed by the parentheses '()' and each element is separated by the comma(,). It is immutable.

EXAMPLE:

even_number = (2, 4, 6, 8)

odd_number = (1, 3, 5, 7)

print(even_number)

print(odd_number)

OUTPUT(2, 4, 6, 8)

(1, 3, 5, 7)

3. Dictionary literal

The dictionary stores the data in the key-value pair. It is enclosed by curly braces '**{}**' and each pair is separated by the commas(,). We can store different types of data in a dictionary. Dictionaries are mutable.

EXAMPLE:alphabets = {'a': 'apple', 'b': 'ball', 'c': 'cat'}

information = {'name': 'amit', 'age': 20, 'ID': 20}

print(alphabets)

print(information)**OUTPUT:**{'a': 'apple', 'b': 'ball', 'c': 'cat'}

{'name': 'amit', 'age': 20, 'ID': 20}

Set literal

•[Set]() is the collection of the unordered data set. It is enclosed by the { } and each element is separated by the comma(,).

EXAMPLE

vowels = {'a', 'e', 'i', 'o', 'u'}

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

print(vowels)

print(fruits)

OUTPUT:

{'o', 'e', 'a', 'u', 'i'}

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

Python Special literal

[Python]() contains one special literal (None). **'None'** is used to define a null variable. If **'None'** is compared with anything else other than a **'None'**, it will return **false**.

EXAMPLE

water_remain = None

print(water_remain)

OUTPUT:

None

THANKYOU