**UNIT 2**

**Python Program to Print Hello world!**

A simple program that displays “Hello, World!”. It's often used to illustrate the syntax of the language.

Python is a cross-platform programming language, which means that it can run on multiple platforms like Windows, macOS, Linux, and has even been ported to the Java and .NET virtual machines. It is free and open-source.

Even though most of today's Linux and Mac have Python pre-installed in it, the version might be out-of-date. So, it is always a good idea to install the most current version.

**Source Code**

# This program prints Hello, world!

print('Hello, world!')

**Output**

Hello World

In this program, we have used the built-in print() function to print the string Hello, world! on our screen.

By the way, a string is a sequence of characters. In Python, strings are enclosed inside single quotes, double quotes, or triple quotes.

**Python Basic Input and Output**

## Python Output

In Python, we can simply use the print() function to print output. For example,

print('Python is powerful')

# Output: Python is powerful

[Run Code](https://www.programiz.com/python-programming/online-compiler)

Here, the print() function displays the string enclosed inside the single quotation.

**Syntax of print()**

In the above code, the print() function is taking a single parameter.

However, the actual syntax of the print function accepts **5** parameters

print(object= separator= end= file= flush=)

Here,

* **object** - value(s) to be printed
* **sep** (optional) - allows us to separate multiple **objects** inside print().
* **end** (optional) - allows us to add add specific values like new line "\n", tab "\t"
* **file** (optional) - where the values are printed. It's default value is sys.stdout (screen)
* **flush** (optional) - boolean specifying if the output is flushed or buffered. Default: False

## Example 1: Python Print Statement

print('Good Morning!')

print('It is rainy today')

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Output**

Good Morning!

It is rainy today

In the above example, the print() statement only includes the **object** to be printed. Here, the value for **end** is not used. Hence, it takes the default value '\n'.

**Program Input and the raw\_input() Built-in Function**

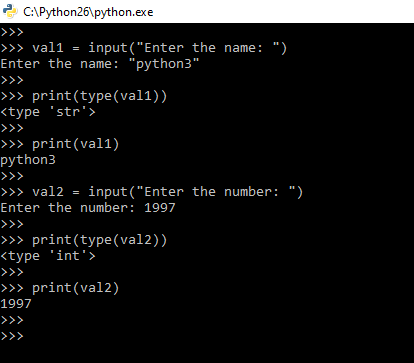
The easiest way to obtain user input from the command-line is with the raw\_input() built-in function. It reads from standard input and assigns the string value to the variable you designate.

**Example program in Python2**

* Python3

|  |
| --- |
| # Python program to demonstrate  # input() function in Python2.x val1 = input("Enter the name: ")  print(type(val1))  print(val1)  val2 = input("Enter the number: ")  print(type(val2))  print(val2) |

**Input and Output**



Here, the value “python3” take from the user and store it in the val1 variable. The function takes the value and type of the input you enter as it is without modifying the type. The type of value in val1 is string type. The value “1997” takes from the user and store it in the variable val2. Now, the type of variable val2 is integer type. We don’t need to explicitly change the variable type.

**raw\_input() function**

Python raw\_input function is used to get the values from the user. We call this function to tell the program to stop and wait for the user to input the values. It is a built-in function. The input function is used only in Python 2.x version. The Python 2.x has two functions to take the value from the user. The first one is input function and another one is raw\_input() function.

**differences in a tabular form -:**

|  |  |  |
| --- | --- | --- |
|  | **input()** | **raw\_input()** |
| **1.** | input() function take the user input. | raw\_input() function takes the input from the user. |
| **2.** | Its syntax is -:  **input(prompt)** | Its syntax is -:  **raw\_input(input)** |
| **3.** | It takes only one parameter that is prompt. | It takes only one parameter that is the input. |
| **4.** | It return the input that it takes. | Its return type is of string. |
| **5.** | It converts the input into a string by removing the trailing newline | It is only introduced in python 2.0 version |
| 6. | blocks until input received | hang ’till user inputs |
| 7. | “hello world” | “hello world” but string |
| 8. | Foo | in snake\_case |

**Python Comments**

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.

## Creating a Comment

Comments starts with a #, and Python will ignore them:

### **Example**

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

## Multiline Comments

Python does not really have a syntax for multiline comments.

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

### **Example**

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

In Python programming, Operators in general are used to perform operations on values and variables. These are standard symbols used for the purpose of logical and arithmetic operations. In this article, we will look into different types of Python operators.

OPERATORS: These are the special symbols. Eg- + , \* , /, etc.

OPERAND: It is the value on which the operator is applied.

**Types of Operators in Python**

Arithmetic Operators

Comparison Operators

Logical Operators

Bitwise Operators

Assignment Operators

Identity Operators and Membership Operators

## ****Arithmetic Operators in Python****

Python [Arithmetic operators](https://www.geeksforgeeks.org/python-arithmetic-operators/) are used to perform basic mathematical operations like**addition, subtraction, multiplication**, and **division**.

In Python 3.x the result of division is a floating-point while in Python 2.x division of 2 integers was an integer. To obtain an integer result in Python 3.x floored (// integer) is used.

| **Operator** | **Description** | **Syntax** |
| --- | --- | --- |
| + | Addition: adds two operands | x + y |
| – | Subtraction: subtracts two operands | x – y |
| \* | Multiplication: multiplies two operands | x \* y |
| / | Division (float): divides the first operand by the second | x / y |
| // | Division (floor): divides the first operand by the second | x // y |
| % | Modulus: returns the remainder when the first operand is divided by the second | x % y |
| \*\* | Power: Returns first raised to power second | x \*\* y |

## Example of Arithmetic Operators in Python

### Division Operators

**Division Operators**allow you to divide two numbers and return a quotient, i.e., the first number or number at the left is divided by the second number or number at the right and returns the quotient.

There are two types of division operators:

1. Float division
2. Floor division

#### **Float division**

The quotient returned by this operator is always a float number, no matter if two numbers are integers. For example:

* Python3

|  |
| --- |
| # python program to demonstrate the use of "/"  print(5/5)  print(10/2)  print(-10/2)  print(20.0/2) |

**Output:**

1.0

5.0

-5.0

10.0

#### **Integer division( Floor division)**

The quotient returned by this operator is dependent on the argument being passed. If any of the numbers is float, it returns output in float. It is also known as Floor division because, if any number is negative, then the output will be floored. For example:

* Python3

|  |
| --- |
| # python program to demonstrate the use of "//"  print(10//3)  print (-5//2)  print (5.0//2)  print (-5.0//2) |

**Output:**

3

-3

2.0

-3.0

## Comparison****Operators in Python****

In Python [Comparison](https://www.geeksforgeeks.org/python-object-comparison-is-vs/)of[Relational operators](https://www.geeksforgeeks.org/relational-operators-in-python/) compares the values. It either returns **True** or **False** according to the condition.

| **Operator** | **Description** | **Syntax** |
| --- | --- | --- |
| > | Greater than: True if the left operand is greater than the right | x > y |
| < | Less than: True if the left operand is less than the right | x < y |
| == | Equal to: True if both operands are equal | x == y |
| != | Not equal to – True if operands are not equal | x != y |
| >= | Greater than or equal to True if the left operand is greater than or equal to the right | x >= y |
| <= | Less than or equal to True if the left operand is less than or equal to the right | x <= y |

= is an assignment operator and == comparison operator.

### Precedence of Comparison Operators in Python

In python, the comparison operators have lower precedence than the arithmetic operators. All the operators within comparison operators have same precedence order.

### **Example of Comparison Operators in Python**

Let’s see an example of Comparison Operators in Python.

* Python3

|  |
| --- |
| # Examples of Relational Operators  a = 13  b = 33    # a > b is False  print(a > b)    # a < b is True  print(a < b)    # a == b is False  print(a == b)    # a != b is True  print(a != b)    # a >= b is False  print(a >= b)    # a <= b is True  print(a <= b) |

**Output**

False

True

False

True

False

True

## ****Logical Operators in Python****

Python [Logical operators](https://www.geeksforgeeks.org/python-logical-operators-with-examples-improvement-needed/) perform **Logical AND**, **Logical OR**, and**Logical NOT** operations. It is used to combine conditional statements.

| **Operator** | **Description** | **Syntax** |
| --- | --- | --- |
| and | Logical AND: True if both the operands are true | x and y |
| or | Logical OR: True if either of the operands is true | x or y |
| not | Logical NOT: True if the operand is false | not x |

### Precedence of Logical Operators in Python

The precedence of Logical Operators in python is as follows:

1. Logical not
2. logical and
3. logical or

### **Example of Logical Operators in Python**

The following code shows how to implement Logical Operators in Python:

* Python3

|  |
| --- |
| # Examples of Logical Operator  a = True  b = False    # Print a and b is False  print(a and b)    # Print a or b is True  print(a or b)    # Print not a is False  print(not a) |

**Output**

False

True

False

## ****Bitwise Operators in Python****

Python [Bitwise operators](https://www.geeksforgeeks.org/python-bitwise-operators/) act on bits and perform bit-by-bit operations. These are used to operate on binary numbers.

| **Operator** | **Description** | **Syntax** |
| --- | --- | --- |
| & | Bitwise AND | x & y |
| | | Bitwise OR | x | y |
| ~ | Bitwise NOT | ~x |
| ^ | Bitwise XOR | x ^ y |
| >> | Bitwise right shift | x>> |
| << | Bitwise left shift | x<< |

### Precedence of Bitwise Operators in Python

The precedence of Bitwise Operators in python is as follows:

1. Bitwise NOT
2. Bitwise Shift
3. Bitwise AND
4. Bitwise XOR
5. Bitwise OR

## ****Assignment Operators in Python****

Python [Assignment operators](https://www.geeksforgeeks.org/assignment-operators-in-python/) are used to assign values to the variables.

| **Operator** | **Description** | **Syntax** |
| --- | --- | --- |
| = | Assign the value of the right side of the expression to the left side operand | x = y + z |
| += | Add AND: Add right-side operand with left-side operand and then assign to left operand | a+=b     a=a+b |
| -= | Subtract AND: Subtract right operand from left operand and then assign to left operand | a-=b     a=a-b |
| \*= | Multiply AND: Multiply right operand with left operand and then assign to left operand | a\*=b     a=a\*b |
| /= | Divide AND: Divide left operand with right operand and then assign to left operand | a/=b     a=a/b |
| %= | Modulus AND: Takes modulus using left and right operands and assign the result to left operand | a%=b     a=a%b |
| //= | Divide(floor) AND: Divide left operand with right operand and then assign the value(floor) to left operand | a//=b     a=a//b |
| \*\*= | Exponent AND: Calculate exponent(raise power) value using operands and assign value to left operand | a\*\*=b     a=a\*\*b |
| &= | Performs Bitwise AND on operands and assign value to left operand | a&=b     a=a&b |
| |= | Performs Bitwise OR on operands and assign value to left operand | a|=b     a=a|b |
| ^= | Performs Bitwise xOR on operands and assign value to left operand | a^=b     a=a^b |
| >>= | Performs Bitwise right shift on operands and assign value to left operand | a>>=b     a=a>>b |
| <<= | Performs Bitwise left shift on operands and assign value to left operand | a <<= b     a= a << b |

## ****Differences between List and Tuple in Python****

List and Tuple in Python are the classes of Python Data Structures. The list is dynamic, whereas the tuple has static characteristics. This means that lists can be modified whereas tuples cannot be modified, the tuple is faster than the list because of static in nature. Lists are denoted by the square brackets but tuples are denoted as parenthesis.

## ****Differences between List and Tuple in Python****

| **Sno** | **LIST** | **TUPLE** |
| --- | --- | --- |
| 1 | [Lists](https://www.geeksforgeeks.org/python-lists/) are [mutable](https://www.geeksforgeeks.org/mutable-vs-immutable-objects-in-python/) | [Tuples](https://www.geeksforgeeks.org/python-tuples/) are immutable |
| 2 | The implication of iterations is Time-consuming | The implication of iterations is comparatively Faster |
| 3 | The list is better for performing operations, such as insertion and deletion. | A Tuple data type is appropriate for accessing the elements |
| 4 | Lists consume more memory | Tuple consumes less memory as compared to the list |
| 5 | Lists have several built-in methods | Tuple does not have many built-in methods. |
| 6 | Unexpected changes and errors are more likely to occur | In a tuple, it is hard to take place. |

## Python List vs Python Tuple

### Test whether tuples are immutable and lists are mutable

Here we are going to compare the list and tuple mutability tests.

* Python3

|  |
| --- |
| # Creating a List with  # the use of Numbers  # code to test that tuples are mutable  List = [1, 2, 4, 4, 3, 3, 3, 6, 5]  print("Original list ", List)  List[3] = 77  print("Example to show mutability ", List) |

**Output:**

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

Example to show mutability [1, 2, 4, 77, 3, 3, 3, 6, 5]

Sample programs

1.Write a Python program to test whether a number is within 100 of 1000 or 2000.

**Sample Solution**:-

**Python Code:**

def near\_thousand(n):

return ((abs(1000 - n) <= 100) or (abs(2000 - n) <= 100))

print(near\_thousand(1000))

print(near\_thousand(900))

print(near\_thousand(800))

print(near\_thousand(2200))

Sample Output:

True

True

False

False

Explanation:

* The said script defines a function called near\_thousand(n) that takes an integer as an argument and returns True if the input is within 100 of either 1000 or 2000, and False otherwise. The function uses the abs() function, which returns the absolute value of a number, to determine the difference between the input and 1000 or 2000.
* Now the script calls the function four time with the input values of 1000, 900, 800 and 2200 respectively.
* When the input value is 1000, it returns True because the difference between 1000 and 1000 is 0, which is less than 100.
* When the input value is t 900, it returns True because the difference between 900 and 1000 is 100, which is less than 100.
* When the input value is 800, it returns False because the difference between 800 and 1000 is 200, which is not less than or equal to 100.
* When the input value is 2200, it returns False because the difference between 2200 and 2000 is 200, which is not less than or equal to 100

2. Write a Python program to display the first and last colors from the following list.

color\_list = ["Red","Green","White" ,"Black"]

**Sample Solution**:-

**Python Code:**

color\_list = ["Red","Green","White" ,"Black"]

print( "%s %s"%(color\_list[0],color\_list[-1]))

Copy

Sample Output:

Red Black

**Explanation:**

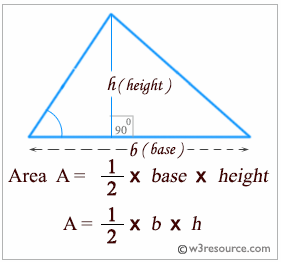
The said code creates a list named "color\_list" which includes four different colors, "Red", "Green", "White", and "Black". Then, it utilizes string formatting to print the first (Index 0) and the last (Index -1) color of the list together with a space in between. The output would be "Red Black".

**3. Write a Python program that will accept the base and height of a triangle and compute its area.**

**Python: Area of a triangle**

A triangle is a polygon with three edges and three vertices. It is one of the basic shapes in geometry. A triangle with vertices A, B, and C is denoted triangle ABC.

* Vertex of a triangle: The point at which two sides of a triangle meet.
* Altitude of a triangle: The perpendicular segment from a vertex of a triangle to the line containing the opposite side.
* Base of a triangle: The side of a triangle to which an altitude is drawn.
* Height of a triangle: The length of an altitude.



**Sample Solution**:-

**Python Code:**

b = int(input("Input the base : "))

h = int(input("Input the height : "))

area = b\*h/2

print("area = ", area)

Copy

Sample Output:

Input the base : 20

Input the height : 40

area = 400.0

**Explanation:**

The said code prompts the user to input an integer value for the base and height of a triangle. Stores these values in variables "b" and "h" respectively. It then calculates the area of the triangle by multiplying the base and height and dividing the result by 2 and assigns the result to the variable "area". Finally, it prints the calculated area.