Python Comments

Python Comments are used to explain the code and make it more readable to another developer. This is very important for the maintenance of the code, or when you are part of a bigger team.

Python comments are not executed by Python Interpreter. These are the program statements that are ignored.

Most developers find it difficult to understand the code without comments. So, even for personnel projects, writing comments helps you in the long run.

How to write Comments in Python?

A comment in Python starts with `#` symbol. Anything that is preceded after `#` till the end of that line, is considered a comment.

We can write any text in the comment. Also, we can comment out part of the program, that we would not want to run, but keep it, during initial stages of an application development.

Comments will be not be considered for execution by Python Interpreter. Therefore, comments are only for developer understanding purpose and the comments does not affect program output or behavior.

Example 1: Python Comment

The following program contains a comment in the first line.

Python Program
```python
#this is a comment
print('Hello World!')
```

The first line starting with `#` is a single line comment.
Output
```
Hello World!
```

The comment is ignored and only the print statement is executed.

Example 2: Comment at the End of Python Statement

We can also write comment after code in that same line. The definition still holds. Anything written after `#` in that line is considered comment.
Python Program
```python
print('Hello World!') #this is a comment
```

Output

```
Hello World!
```

Example 3: Commenting Python Code

We can comment code as well, to prevent Python Interpreter from executing this piece of code.

Python Program

```python
#print("Hello World")
print("Welcome to python Programming")
```

The first print statement is not executed, since Python interprets the line as a comment.

Output

```
Welcome to python Programming
```

Python Multiline Comments

We can write multiple line comments. There is no another symbol or arrangement to write multi-line comments in Python, but we can use a hack.

Example 4: Multiline Comments using

You can write multiline comments using `#` as shown below.

Python Program

```python
#this is a comment
#this is another comment
#one more comment
print('Hello World!')
```

output

```
Hello World!
```

Example 5: Multiline Comments using Multiline String

You can also write a multiple line string, without assigning it to any variable. It does not affect the program output, and therefore can be considered as a comment. An example for multi-line comment using string hack is given below.

Python Program

```python
'''
this is
multi-line string and servers as a comment
'''
print('Hello World!')
```

output
```
Hello World!
```

Python input()

Python input() is a builtin function used to read a string from standard input. The standard input in most cases would be your keyboard.

Syntax – input()

Following is the syntax of input() function.

x = input(prompt)

where prompt is a string, that is printed to the standard output, before input() function starts reading the value input by user. prompt is helpful in letting the users know what kind of input the application is expecting, something like a clue. Prompt is optional though.

input() function returns the string keyed in by user in the standard input.

In detail Working of input()

So, when you call this input() function, Python Interpreter waits for user input from the standard input. When user enters a value in the console input, using keyboard, each character is echoed to the output. This helps user as a feedback of what value is being input. After you enter the value and hit Return or Enter key, the string that is keyed in by you until this last enter key, is returned by the input() function.

Example 1: Python input() Function

In this example, we shall read a string from user using input() function.

Python Program
```
x = input('Enter a value : ')
print('You entered : ', x)
```

Example 2: Read Number using Python input()

By default, input() function returns a string. If you would like to read a number from user, you can typecast the string to int, float or complex, using int(), float() and complex() functions respectively.

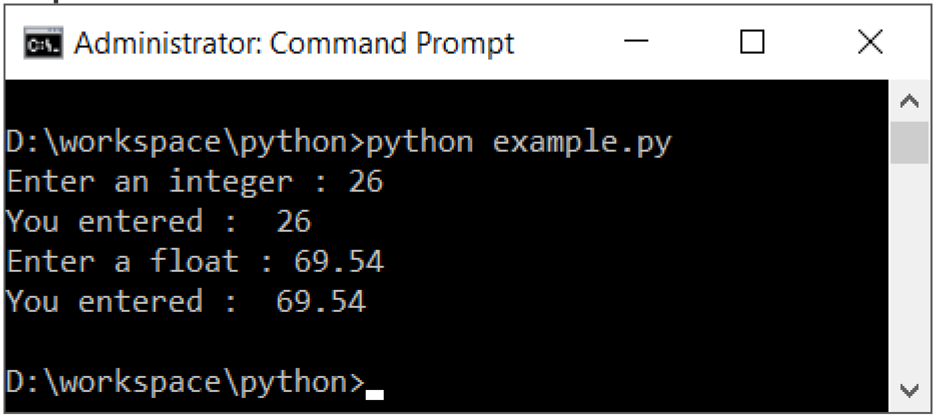
In the following program, we shall read input from user and convert them to integer, float and complex.

Python Program

```
x = int(input('Enter an integer : '))
print('You entered : ', x)

x = float(input('Enter a float : '))
print('You entered : ', x)
```

Output



```
D:\workspace\python>python example.py
Enter an integer : 26
You entered :  26
Enter a float : 69.54
You entered :  69.54

D:\workspace\python>
```

Example 3: Python input() – without prompt message

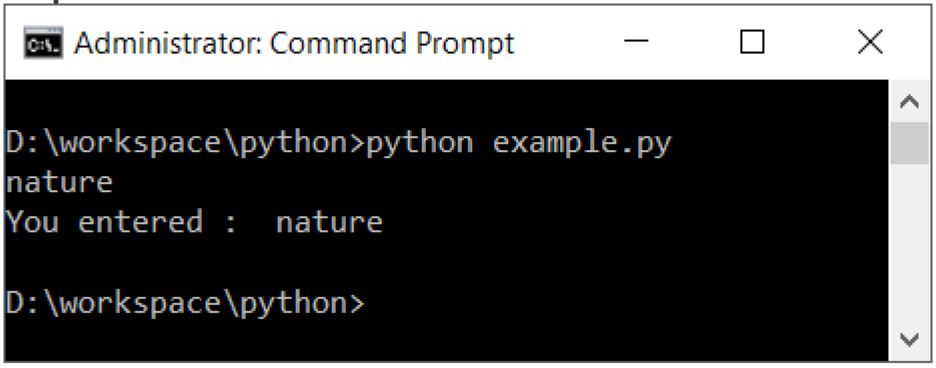
Prompt string is an optional argument to input() function.

In this example, let us not provide a string argument to input() and read a string from user.

Python Program

```
x = input()
print('You entered : ', x)
```

Output



```
D:\workspace\python>python example.py
nature
You entered :  nature

D:\workspace\python>
```

Python print() - Print String to Console Output Example.

Print String to Console Output in Python

Most often, we print data to console. Be it the result we want to display, or intermediate results that we want to check while debugging our program.

In this tutorial, we will learn how to print a string to console with some well detailed examples.

To print a string to console output, you can use Python built-in function – print().

Syntax of print() function

The syntax of Python print() function is

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

where

- **objects** are comma separated objects that has to be printed to the file.
- **sep** is the separator used between objects. **sep** can be a string or a character. By default **sep** is space.
- **end** is the character or a string that has to appended after the **objects**.
- **file** is the place where our print function has to write the objects. By default it is **sys.stdout**, in other words system standard output which is console output.

If you would like to override the default values of sep, end, file or flush parameters, you can do so by giving the corresponding argument. We will see more about them in the following examples.

Example 1: Basic Python Print Statement

Following is a basic example, where we print the string `Hello World` to console.

Python Program

```
print("Hello World")
```

Output

```
Hello World
```

You can also specify multiple strings separated by comma to the print function. All those strings will be considered as `*objects` parameter.

Python Program

```
print("Hello", "World")
```

Output

```
Hello World
```

In this example, we have provided two objects to the print() statement. Since the default separator is **' '** and the **end** character is new line, we got **Hello** and **World** printed with space between them and a new line after the objects.

Example 2: Print with Separator

Now, we will provide a separator to the print function.

Python Program

```
print("Hello", "World", sep='-')
```

Output

```
Hello-World
```

We have overridden the default separator with a character. You can also use a string for a separator.

Python Program

```
print("Hello", "World", sep=', - ')
```

Output

```
Hello, - World
```

Example 3: Print with different ending character

May be you do not want a new line ending. So, with `print()` function's `end` parameter, you can specify your own `end` character or string when it prints to the console.

Python Program

```
print("Hello", "World", end='.')
```

Output

```
Hello World.
```