Python Programming

- Presentation By Uplatz
- Contact us: https://training.uplatz.com
- Email: info@uplatz.com
- Phone: +44 7836 212635

Uplatz

Python Identifiers, Keywords, Reading Input, Output Formatting

Learning outcomes:

- What is an Identifier
- Keywords
- Reading Input
- Taking multiple inputs from user
- Output Formatting
- Python end parameter

What is an Identifier?

- A Python identifier is a name used to identify a variable, function, class, module, or other object. An identifier starts with a letter A to Z or a to z, or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).
- Python does not allow punctuation characters such as @, $, and % within identifiers. Python is a case sensitive programming language. Thus, **Man** and **man** are two different identifiers in Python. It helps to differentiate one entity from another.

What is an Identifier?

Example:

Age10 <<- Valid
10Age <<- Invalid(identifier cannot start with a digit)
Age_10 <<- Valid
if =10 <<- Invalid (Keywords cannot be used as identifiers.)
_Age_10 <<- Valid
Age@ = 10 <<- Invalid (We cannot use special symbols like **!**, **@**, **#**, **$**, **%** etc. in our identifier.)

Keywords in Python

Keywords are the reserved words in Python.
We cannot use a keyword as
a variable name, function name or any other identifier. They are used to define the syntax and structure of the Python language.
In Python, keywords are case sensitive.
There are 35 keywords in Python 3.7. This number can vary slightly over the course of time.
All the keywords except True, False and None are in lowercase and they must be written as they are.
The list of all the keywords is given below.

Keywords in Python

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

Keywords in Python

We can get the complete list of keywords using python interpreter help utility.

$ Python 3.8.5
>>> **help()**
help> **keywords**

Reading input in Python

Developers often have a need to interact with users, either to get data or to provide some sort of result. Most programs today use a dialog box as a way of asking the user to provide some type of input.

Reading input in Python

2 functions

input()
- Python v3.x

raw_input()
- Python v2.x

Uplatz

Reading input in Python

raw_input ( ) : This function works in older version (like Python 2.x). This function takes exactly what is typed from the keyboard, convert it to string and then return it to the variable in which we want to store.

For example :

```
name = raw_input("Enter your name : ")
Print (name)
```

Reading input in Python

input ( ) : Python has an **input()** function which lets you ask a user for some text **input**. You call this function to tell the program to stop and wait for the user to input the data.

For example:

```
num = input ("Enter number :")
print(num)
```

Reading input in Python

input ( ) : Python has an **input()** function which lets you ask a user for some text **input**. You call this function to tell the program to stop and wait for the user to input the data.

For example:

```
num = input ("Enter number :")
print(num)
```

Reading input in Python

When **input()** function executes program flow will be stopped until the user has given an input.
The text or message display on the output screen to ask a user to enter input value is optional i.e. the prompt, will be printed on the screen is optional.
Whatever you enter as input, input function convert it into a string. if you enter an integer value still input() function convert it into a string. You need to explicitly convert it into an integer in your code using typecasting.

Uplatz

Reading input in Python

Taking integer input from user:

There might be conditions when you might require integer input from user/Console, the following code takes two input(integer/float) from console and typecasts them to integer then prints the sum. For example:

```
num1 = int(input())
num2 = int(input())
  # printing the sum in integer
print(num1 + num2)
```

Taking multiple inputs from user

Developer often wants a user to enter multiple values or inputs in one line. In C++/C user can take multiple inputs in one line using **scanf()** but in Python user can take multiple values or inputs in one line using **split()** method.

Taking multiple inputs from user

Using split() method :

This function helps in getting a multiple inputs from user . It breaks the given input by the specified separator. If separator is not provided then any white space is a separator. Generally, user use a split() method to split a Python string but one can used it in taking multiple input.

Syntax :

input().split(separator, maxsplit)

Uplatz

Taking multiple inputs from user

Using split() method :

Example:

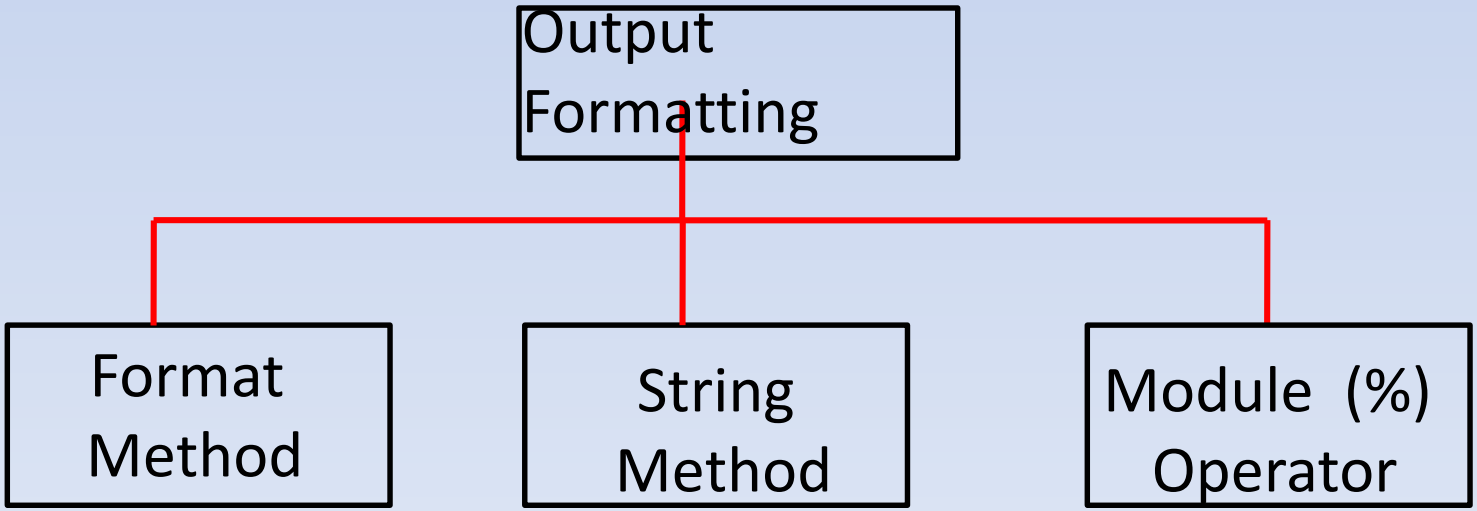
```
x, y = input("Enter a two value: ").split()
print("Number of boys: ", x)
print("Number of girls: ", y)
```

Output Formatting

There are several ways to present the output of a program, data can be printed in a human-readable form, or written to a file for future use. Sometimes user often wants more control the formatting of output than simply printing space-separated values. There are several ways to format output.

Output Formatting

Formatting is done using



Output Formatting

Formatting output using String modulo operator(%) :

The % operator can also be used for string formatting. It interprets the left argument much like a **printf()-**style format string to be applied to the right argument.

In Python, there is no **printf()** function but the functionality of the ancient **printf** is contained in Python.

Output Formatting

Formatting output using String modulo operator(%) :

To this purpose, the modulo operator % is overloaded by the string class to perform string formatting. Therefore, it is often called string modulo (or sometimes even called modulus) operator.

String modulo operator (%) is still available in Python(3.x) and user is using it widely. But nowadays the old style of formatting is removed from the language.

Output Formatting

Formatting output using String modulo operator(%) :

Example:

```
# print integer and float value
print("Rank : %d, Percentage : %f" %(1, 95.333))

# print integer value
print("Total students :%d, Boys :%d" %(240, 120))
```

Output Formatting

Formatting output using String modulo operator(%) :

Example:

print("Total students :%5d, Boys :%d" %(240, 120))

Here **5** means "use at least 5 spaces to display, padding as needed“

print ("Growth is %.3f" %(10.57)) # output 10.570

Output Formatting

Formatting output using format method :
The format() method was added in Python(2.6). Format method of strings requires more manual effort. User use **{}** to mark where a variable will be substituted and can provide detailed formatting directives, but user also needs to provide the information to be formatted. This method lets us concatenate elements within an output through positional formatting.

Output Formatting

Formatting output using format method :

Example:

```
print('I am learning {} from "{}!"'.format('Python', 'Expert'))
# using format() method and referring
# a position of the object
print('{0} and {1}'.format('Python', 'Expert'))
print('{1} and {0}'.format('Python', 'Expert'))
```

A number in the brackets can be used to refer to the position of the object passed into the format() method.

Output Formatting

Formatting output using String method :

In this output is formatted by using string slicing and concatenation operations. The string type has some methods that help in formatting a output in an fancier way. Some of method which help in formatting a output are

str.ljust(), str.rjust(), str.centre()

Output Formatting

Formatting output using String method :

Example:

```
cstr = "I love Python Programming"
# Printing the center aligned string with fillchr
print ("Center aligned string with fillchr: ")
print (cstr.center(40, '#'))
# Printing the left aligned string with "-" padding
print ("The left aligned string is : ")
print (cstr.ljust(40, '-'))
# Printing the right aligned string with "-" padding
print ("The right aligned string is : ")
print (cstr.rjust(40, '-'))
```

Python end parameter in print()

By default python's **print()** function ends with a newline. A programmer with C/C++ background may wonder how to print without newline.
Python's **print()** function comes with a parameter called **'end'.** By default, the value of this parameter is **'\n',** i.e. the new line character. You can end a print statement with any character/string using this parameter.
Example: