Python Statement

By default, the Python interpreter treats a piece of text terminated by hard carriage return (new line character) as one statement. It means each line in a Python script is a statement. (Just as in C/C++/C#, a semicolon ; denotes the end of a statement).

Example: Python Statements

msg="Hello World"

code=123

name="Steve"

However, you can show the text spread over more than one lines to be a single statement by using the backslash (\) as a continuation character. Look at the following examples:

Example: Continuation of Statement

msg="Hello Pythonista \

Welcome to Python Tutorial \

from TutorialsTeacher.com"

Similarly, use the semicolon ; to write multiple statements in a single line.

Example: Multiple Statements in Single Line

msg="Hello World";code=123;name="Steve"

Example: Multi-line Comments

'''

comment1

comment2

comment3

'''

Example: display.py

name="Amar"

age=21

print("Name:", name, end=" ")

print("Age:", age)

>>>python display.py Name: Amar Age: 21

String Formatting

>>> name="Bond"  
>>> "My name is %s." % name  
'My name is Bond.'

You can have multiple parameters too.

>>> name="Bond"  
>>> age=30  
>>> "My name is %s and age is %d years." % (name, age)  
'My name is Bond and age is 30 years.'

You can specify the width of integer and float objects. Here, integers a, b and c will occupy the width of 3 characters in the formatted string. Additional spaces will be padded to the left.

>>> a=1  
>>> b=11  
>>> c=111  
>>> "a=%3d b=%3d c=%3d" % (a, b, c)  
'a=  1 b= 11 c=111'

The following specifies the width of the float variable.

>>> percent=55.50  
>>> "%5.2f" % percent  
'55.50'  
>>> "%6.2f" % percent  
' 55.50'  
>>> "%6.3f" % percent  
'55.500'  
>>> "%7.3f" % percent  
' 55.500'

In the above example, %5.2 specifies the width of float, where 5 is for total characters and 2 is for decimals. So, the result would be '55.50'.

The width of a string can also be specified. The default alignment is right. For left alignment, give a negative sign to width.

>>>'%4s' % 'abc'  
>>>' abc'  
  
>>>'%6s' % 'abc'  
>>>' abc'  
  
>>>'%-6s' % 'abc'  
>>>'abc '  
  
>>>a='abc'  
>>>'%-6s' % a  
>>>'abc '

format() method

The format() method can handle complex string formatting more efficiently. This method of in-built string class provides the ability to do complex variable substitutions and value formatting. This new formatting technique is regarded as more elegant. The general syntax of the format() method is as follows:

string.format(str1, str2,...)

The string itself contains placeholders {}, in which the values of variables are successively inserted.

>>>name="Bill"  
>>>age=25  
>>>"My name is {} and I am {} years old.".format(name, age)  
'My name is Bill and I am 25 years old.'  
  
>>>myStr = "My name is {} and I am {} years old."  
>>>myStr.format(name, age)  
'my name is Bill and I am 25 years old.'

You can also specify formatting symbols by using : instead of %. For example, instead of %s use {:s} and instead of %d use {:d}.

>>> "My name is {:s} and I am {:d} years old.".format(name, age)  
'My name is Bill and I am 25 years old.'

Precision formatting of numbers can be done accordingly.

>>> percent=55.50  
>>> "I have scored {:6.3f} percent marks.".format(percent)  
'I have scored 55.500 percent marks.'

Converting to String

Python has an in-built function str() which returns a printable string representation of any object. In the previous chapter we have used int(), float() and complex() functions. They convert the string representation into integer, float and complex numbers, respectively. The str() function converts any number to a string object.

>>> str(12)  
'12'  
>>> str(6+5j)  
'(6+5j)'  
>>> str(1.11)  
'1.11'

String Formatting

Interpolation of objects of different types at placeholders inside a string is called string formatting. The % operator (otherwise an arithmetic operator used to return the remainder of division) is used to perform string formatting also. Format specification symbols (%d, %c, %f, %s, etc) used in C language are utilized as placeholders in a string.

In the following example, name is a string and age is an integer variable. Their values are inserted in the string with %s and %d format specification symbols, respectively. These symbols are interpolated to values in a tuple with the % operator in front.

>>> name="Bond"  
>>> "My name is %s." % name  
'My name is Bond.'

You can have multiple parameters too.

>>> name="Bond"  
>>> age=30  
>>> "My name is %s and age is %d years." % (name, age)  
'My name is Bond and age is 30 years.'

All C style format specification symbols are permitted.

| Format Symbol | Conversion |
| --- | --- |
| %c | character |
| %s | string conversion via str() prior to formatting |
| %i | signed decimal integer |
| %d | signed decimal integer |
| %u | unsigned decimal integer |
| %o | octal integer |
| %x / %X | hexadecimal integer (lowercase letters) |
| %e / %E | exponential notation (with lowercase 'e') |
| %f | floating point real number |

You can specify the width of integer and float objects. Here, integers a, b and c will occupy the width of 3 characters in the formatted string. Additional spaces will be padded to the left.

>>> a=1  
>>> b=11  
>>> c=111  
>>> "a=%3d b=%3d c=%3d" % (a, b, c)  
'a=  1 b= 11 c=111'

The following specifies the width of the float variable.

>>> percent=55.50  
>>> "%5.2f" % percent  
'55.50'  
>>> "%6.2f" % percent  
' 55.50'  
>>> "%6.3f" % percent  
'55.500'  
>>> "%7.3f" % percent  
' 55.500'

In the above example, %5.2 specifies the width of float, where 5 is for total characters and 2 is for decimals. So, the result would be '55.50'.

The width of a string can also be specified. The default alignment is right. For left alignment, give a negative sign to width.

>>>'%4s' % 'abc'  
>>>' abc'  
  
>>>'%6s' % 'abc'  
>>>' abc'  
  
>>>'%-6s' % 'abc'  
>>>'abc '  
  
>>>a='abc'  
>>>'%-6s' % a  
>>>'abc '

format() method

The format() method can handle complex string formatting more efficiently. This method of in-built string class provides the ability to do complex variable substitutions and value formatting. This new formatting technique is regarded as more elegant. The general syntax of the format() method is as follows:

string.format(str1, str2,...)

The string itself contains placeholders {}, in which the values of variables are successively inserted.

>>>name="Bill"  
>>>age=25  
>>>"My name is {} and I am {} years old.".format(name, age)  
'My name is Bill and I am 25 years old.'  
  
>>>myStr = "My name is {} and I am {} years old."  
>>>myStr.format(name, age)  
'my name is Bill and I am 25 years old.'

You can also specify formatting symbols by using : instead of %. For example, instead of %s use {:s} and instead of %d use {:d}.

>>> "My name is {:s} and I am {:d} years old.".format(name, age)  
'My name is Bill and I am 25 years old.'

Precision formatting of numbers can be done accordingly.

>>> percent=55.50  
>>> "I have scored {:6.3f} percent marks.".format(percent)  
'I have scored 55.500 percent marks.'

String alignment is done with <, > and ^ symbols in the place holder causing left, right and center alignment, respectively. Default is left alignment.

>>> '{:>10}'.format('test')  
'      test'  
>>> '{:<10}'.format('test')  
'test      '  
>>> '{:^10}'.format('test')  
'   test   '

Built-in String Methods

capitalize():

Converts the first character of a string to uppercase letters.

>>> mystr='python'  
>>> mystr.capitalize()  
'Python'

upper()

Replaces the lowercase characters in a string with corresponding uppercase characters.

>>> mystr='Python'  
>>> mystr.upper()  
'PYTHON'

lower()

Replaces the uppercase characters in a string with corresponding lowercase characters.

>>> mystr='PYTHON'  
>>> mystr.lower()  
'python'

title():

Returns the string with the first character of each word converted to uppercase.

>>> mystr='python tutorial from tutorials teacher'  
>>> mystr.title()  
'Python Tutorial From Tutorials Teacher'

find()

The find() method finds the first occurrence of a substring in another string. If not found, the method returns -1.

>>> mystr='Python Tutorial From Tutorials Teacher'  
>>> mystr.find('From')  
16  
>>> mystr.find('xyz')  
-1

Substring 'From' first occurs at position 16 (the count starts from 0). 'xyz' is not found, hence it returns -1.

count()

The count() method returns the number of occurrences of a substring in the given string.

>>> mystr='Python Tutorial From Tutorials Teacher'  
>>> mystr.count('Tutorial')  
2

isalpha()

The isalpha() method returns true if all the characters in a string are alphabetic letters (a-z or A-Z), otherwise it returns false.

>>> mystr='TutorialsTeacher'  
>>> mystr.isalpha()  
True  
>>> mystr='Tutorials Teacher'  
>>> mystr.isalpha()  
False

isdigit()

The isdigit() is method returns true if all the characters in string are digits (0-9), if not, it returns false.

>>> str1='2000'  
>>> str1.isdigit()  
True  
>>> str2='2,000'  
>>> str2.isdigit()  
False

islower()

The islower() method returns true if all the characters in the string are lowercase characters, else it returns false.

>>> str1='python'  
>>> str1.islower()  
True  
>>> str2='Python'  
>>> str2.islower()  
False

isupper()

The isupper() method returns true if all the characters in the string are uppercase characters, else it returns false.

>>> var='TUTORIALSTEACHER'  
>>> var.isupper()  
True  
>>> var='TUTORIALSTeacher'  
>>> var.isupper()  
False