Python Variables: Declare, Concatenate, Global & Local

What is a Variable in Python?

A Python variable is a reserved memory location to store values. In other words, a variable in a python program gives data to the computer for processing.

Every value in Python has a datatype. Different data types in Python are Numbers, List, Tuple, Strings, Dictionary, etc. Variables can be declared by any name or even alphabets like a, aa, abc, etc.

How to Declare and use a Variable

Let see an example. We will declare variable "a" and print it.

```
a=100
print a
```

Re-declare a Variable

You can re-declare the variable even after you have declared it once.

Here we have variable initialized to f=0.

Later, we re-assign the variable f to value "guru99"

Python 2 Example

```
# Declare a variable and initialize it
f = 0
print f
# re-declaring the variable works
f = 'guru99'
print f
```

Python 3 Example

```
# Declare a variable and initialize it
f = 0
```

```
print(f)
# re-declaring the variable works
f = 'guru99'
print(f)
```

Concatenate Variables

Let's see whether you can concatenate different data types like string and number together. For example, we will concatenate "Guru" with the number "99".

Unlike Java, which concatenates number with string without declaring number as string, Python requires declaring the number as string otherwise it will show a TypeError

For the following code, you will get undefined output -

```
a="Guru"
b = 99
print a+b
```

Once the integer is declared as string, it can concatenate both "Guru" + **str**("99")= "Guru99" in the output.

```
a="Guru"
b = 99
print(a+str(b))
```

Local & Global Variables

In Python when you want to use the same variable for rest of your program or module you declare it a global variable, while if you want to use the variable in a specific function or method, you use a local variable.

Let's understand this difference between local and global variable with the below program.

1. Variable "f" is **global** in scope and is assigned value 101 which is printed in output
2. Variable f is again declared in function and assumes **local** scope. It is assigned value "I am learning Python." which is printed out as an output. This variable is different from the global variable "f" define earlier

3. Once the function call is over, the local variable f is destroyed. At line 12, when we again, print the value of "f" is it displays the value of global variable f=101

Python 2 Example

```
# Declare a variable and initialize it
f = 101
print f
# Global vs. local variables in functions
def someFunction():
# global f
    f = 'I am learning Python'
    print f
someFunction()
print f
```

Python 3 Example

```
# Declare a variable and initialize it
f = 101
print(f)
# Global vs. local variables in functions
def someFunction():
# global f
    f = 'I am learning Python'
    print(f)
someFunction()
print(f)
```

Using the keyword **global,** you can reference the global variable inside a function.

1. Variable "f" is **global** in scope and is assigned value 101 which is printed in output
2. Variable f is declared using the keyword **global**. This is **NOT** a **local variable**, but the same global variable declared earlier. Hence when we print its value, the output is 101
3.

   We changed the value of "f" inside the function. Once the function call is over, the changed value of the variable "f" persists. At line 12, when we again, print the value of "f" is it displays the value "changing global variable"

4.

Python 2 Example

```
f = 101;
print f
# Global vs.local variables in functions
def someFunction():
  global f
print f
f = "changing global variable"
someFunction()
print f
```

Python 3 Example

```
f = 101;
print(f)
# Global vs.local variables in functions
def someFunction():
  global f
print(f)
f = "changing global variable"
someFunction()
print(f)
```

Delete a variable

You can also delete variable using the command **del** "variable name".

```
f = 11;
print(f)
del f
print(f)
```

Python Strings: Replace, Join, Split, Reverse, Uppercase & Lowercase

In Python everything is object and string are an object too. Python string can be created simply by enclosing characters in the double quote.

For example:

var = **"Hello World!"**

Accessing Values in Strings

Python does not support a character type, these are treated as strings of length one, also considered as substring.

We use square brackets for slicing along with the index or indices to obtain a substring.

```
var1 = "Guru99!"
var2 = "Software Testing"
print ("var1[0]:",var1[0])
print ("var2[1:5]:",var2[1:5])
```

Various String Operators

There are various string operators that can be used in different ways like concatenating different string.

Suppose if a=guru and b=99 then a+b= "guru99". Similarly, if you are using a*2, it will "GuruGuru". Likewise, you can use other operators in string.

Operator	Description	Example	
[]	Slice- it gives the letter from the given index	a[1] will give "u" from the word Guru as such (0=G, 1=u, 2=r and 3=u)	`x="Guru"` `print x[1]`
[:]	Range slice-it gives the characters from the given range	x [1:3] it will give "ur" from the word Guru. Remember it will not consider 0 which is G, it will consider word after that is ur.	`x="Guru"` `print x[1:3]`
in	Membership-returns true if a letter exist in the given string	u is present in word Guru and hence it will give 1 (True)	`x="Guru"` `print "u" in x`
not in	Membership-returns true if a letter exist is not in the given string	l not present in word Guru and hence it will give 1	`x="Guru"` `print "l" not in`
r/R	Raw string suppresses actual meaning of escape	Print r'\n' prints \n and print R'/n' prints \n	

Operator	Description	Example	
	characters.		
% - Used for string format	%r - It insert the canonical string representation of the object (i.e., repr(o)) %s- It insert the presentation string representation of the object (i.e., str(o)) %d- it will format a number for display	The output of this code will be "guru 99".	name = 'guru' number = 99 print'%s %d' % (n r)
+	It concatenates 2 strings	It concatenate strings and gives the result	x="Guru" y="99" print x+y
*	Repeat	It prints the character twice.	x="Guru" y="99" print x*2

Some more examples

You can update Python String by re-assigning a variable to another string. The new value can be related to previous value or to a completely different string all together.

```
x = "Hello World!"
print(x[:6])
print(x[0:6] + "Guru99")
```

Note : - Slice:6 or 0:6 has the same effect

Python String replace() Method

The method replace() returns a copy of the string in which the values of old string have been replaced with the new value.

```
oldstring = 'I like Guru99'
newstring = oldstring.replace('like', 'love')
print(newstring)
```

Changing upper and lower case strings

In Python, you can even change the string to upper case or lower case.

```
string="python at guru99"
print(string.upper())
```

Likewise, you can also do for other function as well like capitalize

```
string="python at guru99"
print(string.capitalize())
```

You can also convert your string to lower case

```
string="PYTHON AT GURU99"
print(string.lower())
```

Using "join" function for the string

The join function is a more flexible way for concatenating string. With join function, you can add any character into the string.

For example, if you want to add a colon (:) after every character in the string "Python" you can use the following code.

```
print(":".join("Python"))
```

Reversing String

By using the reverse function, you can reverse the string. For example, if we have string "12345" and then if you apply the code for the reverse function as shown below.

```
string="12345"
print(''.join(reversed(string)))
```

Split Strings

Split strings is another function that can be applied in Python let see for string "guru99 career guru99". First here we will split the string by using the command word.split and get the result.

```
word="guru99 career guru99"
print(word.split(' '))
```

To understand this better we will see one more example of split, instead of space (' ') we will replace it with ('r') and it will split the string wherever 'r' is mentioned in the string

```
word="guru99 career guru99"
print(word.split('r'))
```

Important Note:

In Python, Strings are immutable.

Consider the following code

```
x = "Guru99"
x.replace("Guru99","Python")
print(x)
```

will still return Guru99. This is because x.replace("Guru99","Python") returns **a copy of X with replacements made**

You will need to use the following code to observe changes

```
x = "Guru99"
x = x.replace("Guru99","Python")
print(x)
```

Above codes are Python 3 examples, If you want to run in Python 2 please consider following code.

Python 2 Example

```
#Accessing Values in Strings
var1 = "Guru99!"
var2 = "Software Testing"
print "var1[0]:",var1[0]
print "var2[1:5]:",var2[1:5]
#Some more examples
x = "Hello World!"
print x[:6]
print x[0:6] + "Guru99"
#Python String replace() Method
oldstring = 'I like Guru99'
newstring = oldstring.replace('like', 'love')
print newstring
#Changing upper and lower case strings
```

```
string="python at guru99"
print string.upper()
string="python at guru99"
print string.capitalize()
string="PYTHON AT GURU99"
print string.lower()
#Using "join" function for the string
print":".join("Python")
#Reversing String
string="12345"
print''.join(reversed(string))
#Split Strings
word="guru99 career guru99"
print word.split(' ')
word="guru99 career guru99"
print word.split('r')
x = "Guru99"
x.replace("Guru99","Python")
print x
x = "Guru99"
x = x.replace("Guru99","Python")
print x
```

Python has introduced a .format function which does way with using the cumbersome %d and so on for string formatting.