Python Variables

June 15, 2024

Dr. Labeed Al-Saad

0.1 Variables

Variables are containers for storing data values.

0.2 Creating Variables

Python has no command for declaring a variable.

A variable is created the moment you first assign a value to it.

Example:

[1]:
```
x = 5
y = "John"
print(x)
print(y)
```

```
5
John
```

**Variables do not need to be declared with any particular type, and can even change type after they have been set.

Example:

[2]:
```
x = 4       # x is of type int
x = "Sally" # x is now of type str
print(x)
```

```
Sally
```

0.3 Casting

If you want to specify the data type of a variable, this can be done with casting.

Example

[3]:
```
x = str(3)    # x will be '3'
y = int(3)    # y will be 3
z = float(3)  # z will be 3.0
```

```
print(x)
print(y)
print(z)
```

```
3
3
3.0
```

0.4 Get the Type

**You can get the data type of a variable with the type() function.

Example:

[4]:
```
x = 5
y = "John"
print(type(x))
print(type(y))
```

```
<class 'int'>
<class 'str'>
```

0.5 Single or Double Quotes?

String variables can be declared either by using single or double quotes:

Example:

[8]:
```
x = "John"
print(x)
#double quotes are the same as single quotes:
x = 'John'
print(x)
```

```
John
John
```

0.6 Case-Sensitive

Variable names are case-sensitive.

Example:

This will create two variables:

[9]:
```
a = 4
A = "Sally"
#A will not overwrite a
print(a)
print(A)
```

```
4
Sally
```

0.7 Python - Variable Names

**A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume). Rules for Python variables:

1. A variable name must start with a letter or the underscore character.
2. A variable name cannot start with a number.
3. A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _).
4. Variable names are case-sensitive (age, Age and AGE are three different variables).
5. A variable name cannot be any of the Python keywords.

Example:

Legal variable names:

[10]:
```
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"

print(myvar)
print(my_var)
print(_my_var)
print(myVar)
print(MYVAR)
print(myvar2)
```

```
John
John
John
John
John
John
```

Example about illegal variable names:

[12]:
```
2myvar = "John"
my-var = "John"
my var = "John"

print(2myvar)
print(my-var)
```

```
print(my var)
```

```
  Cell In[12], line 1
    2myvar = "John"
    ^
SyntaxError: invalid decimal literal
```

0.8 Remember that variable names are case-sensitive

0.9 Multi Words Variable Names

Variable names with more than one word can be difficult to read.

There are several techniques you can use to make them more readable:

0.10 Camel Case

Each word, except the first, starts with a capital letter:

[ ]:
```
myVariableName = "John"
```

0.11 Pascal Case

Each word starts with a capital letter:

[ ]:
```
MyVariableName = "John"
```

0.12 Snake Case

Each word is separated by an underscore character:

[ ]:
```
my_variable_name = "John"
```

1 Python Variables - Assign Multiple Values

1.1 Many Values to Multiple Variables

Python allows you to assign values to multiple variables in one line:

**Note: Make sure the number of variables matches the number of values, or else you will get an error.

Example:

[13]:
```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

```
Orange
Banana
Cherry
```

1.2 One Value to Multiple Variables

And you can assign the same value to multiple variables in one line:

Example:

[14]:
```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

```
Orange
Orange
Orange
```

1.3 Unpack a Collection

If you have a collection of values in a list, tuple etc. Python allows you to extract the values into variables. This is called unpacking.

Example:

Unpack a list:

[15]:
```
fruits = ["apple", "banana", "cherry"]
x, y, z = fruits
print(x)
print(y)
print(z)
```

```
apple
banana
cherry
```

**You can also unpacking tuples

2 Python - Output Variables

2.1 Output Variables

The Python print() function is often used to output variables.

Example:

[16]:
```
x = "Python is awesome"
print(x)
```

```
Python is awesome
```

**In the print() function, you output multiple variables, separated by a comma:

Example:

[17]:
```
x = "Python"
y = "is"
z = "awesome"
print(x, y, z)
```

```
Python is awesome
```

**You can also use the + operator to output multiple variables:

Example:

**Notice the space character after "Python" and "is", without them the result would be "Pythonisawesome".

[21]:
```
x = "Python " #there is a space after n
y = "is "     #there is a space after s
z = "awesome" #no space before or after awesome
print(x + y + z)
```

```
Python is awesome
```

**For numbers, the + character works as a mathematical operator:

Example:

[22]:
```
x = 5
y = 10
print(x + y)
```

```
15
```

**In the print() function, when you try to combine a string and a number with the + operator, Python will give you an error:

Example:

[23]:
```
x = 5
y = "John"
print(x + y)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[23], line 3
      1 x = 5
      2 y = "John"
----> 3 print(x + y)

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

**The best way to output multiple variables in the print() function is to separate them with commas, which even support different data types:

Example:

[24]:
```
x = 5
y = "John"
print(x, y)
```

```
5 John
```

3 Python - Global Variables

3.1 Global Variables

Variables that are created outside of a function (as in all of the examples in the previous pages) are known as global variables.

Global variables can be used by everyone, both inside of functions and outside.

Example: Create a variable outside of a function, and use it inside the function

[25]:
```
x = "awesome"

def myfunc():
  print("Python is " + x)

myfunc()
```

```
Python is awesome
```

**If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.

Example:

Create a variable inside a function, with the same name as the global variable

[27]:
```
x = "awesome"

def myfunc():
  x = "fantastic"
  print("Python is " + x)  #this is print local x

myfunc()

print("Python is " + x) #this is print global x
```

```
Python is fantastic
Python is awesome
```

3.2 The global Keyword

Normally, when you create a variable inside a function, that variable is local, and can only be used inside that function.

To create a global variable inside a function, you can use the global keyword.

Example:

If you use the global keyword, the variable belongs to the global scope:

[29]:
```
def myfunc():
  global x
  x = "fantastic"
  print(x)

myfunc()

print("Python is " + x)
```

```
fantastic
Python is fantastic
```

**Also, use the global keyword if you want to change a global variable inside a function.

Example:

To change the value of a global variable inside a function, refer to the variable by using the global keyword:

[30]:
```
x = "awesome"  #this is the global one

def myfunc():
  global x     #here we changed the value of global x with new one
  x = "fantastic" #now this will be the new one

myfunc()

print("Python is " + x) # the x value will be "fantastic" instead of "awesome"
```

```
Python is fantastic
```