KEY_Lesson03_Variables_Types

January 11, 2020

1 Variables

In the last lesson we learned that we can use variables to store values in Python, similar to how we use variables in algebra. Each variable is assigned a value, and then we can use the variable name to reference the value it represents.

Variables are very helpful in coding. Not only do they prevent us from having to remember and type the values over and over again, it allows us to update certain values when we have new information. For example, let's say we have a variable `apples` that stores the number of apples we have in our refrigerator.

[1]:
```
apples = 2
```

Now let's say I went to the store and bought half a dozen apples. Now I want to update the value of `apples` without having to remember how many we started with. How should I do this?

[2]:
```
# add 6 to apples and print apples
apples + 6
print(apples)
```

```
2
```

[3]:
```
# add 6 to apples and update apples
# print apples
apples += 6
print(apples)
```

```
8
```

Notice that we can't just add 6 to `apples` and have Python know we mean for it to save that result, **we need to explicitly tell Python to update the value of `apples`**. To do this we use the `+=` operator. This adds 6 to the previous value of `apples` (2) and updates the variable `apples` to store this new value. This is called **incrementing**.

Let's say I had some friends coming over for dinner and I wanted to make an apple pie for dessert. This used up 4 apples. How do you think we should update the variable `apples` to reflect the number of apples that remain in the reifrigerator?

[4]:
```
# subtract 4 from apples and update it
# print apples
apples -= 4
print(apples)
```

```
4
```

This is an example of **decrementing**.

2 Types

So far we have only used variables to represent numbers. In fact, Python allows us to store several different kinds of data and assign their values to variables. These basic kinds of data come in pre-defined flavors, called **types**. Each type has certain key ways to identify it and certain rules that it has to follow. All variables fall into some type, and you can use the `type()` function to determine the type of any variable. Let's test this with our `apples` variable from above.

[5]:
```
type(apples)
```

[5]:
```
int
```

2.0.1 Numeric Types

`int` stands for integer, which means a whole number or a number that does not have any decimal value. You may have heard this word before in your math classes. Integers follow basic mathematical rules and can be manipulated with standard mathematical operators, like we saw in the previous lesson.

Another numerical type is called `float`, which stands for floating point number. This type represents numbers with decimal values. Let's see an example.

[6]:
```
# create float variable oranges, print type
oranges = 3.5
type(oranges)
```

[6]:
```
float
```

`float` variables are very similar to `int` variables, since they are both numeric types and abide by the same basic mathematical rules. We can even convert between one type to another.

[7]:
```
# int to float
apples = float(apples)
print(apples)
type(apples)
```

```
4.0
```

[7]: float

[8]:
```
# float to int
oranges = int(oranges)
print(oranges)
type(oranges)
```

```
3
```

[8]: int

Notice that here Python rounded down. When you use `int` in Python, the decimal place is just cut off so the number is always rounded down.

[9]:
```
# add float to int
oranges += 0.3
print(oranges)
type(oranges)
```

```
3.3
```

[9]: float

[10]:
```
# add int to float
oranges += 4
print(oranges)
type(oranges)
```

```
7.3
```

[10]: float

2.0.2 Strings

When you think of data science, you may think mainly of numbers and statistics, but sometimes we need to analyze data in other forms as well. Often we need to work with text data, which are called **strings** in Python and designated using `str`. Strings are comprised of characters, like letters or symbols, or a bunch of characters put together, like words. Strings are defined using single quotations `' '` or double quotations `" "`. You can use either as long as they match at the start and end of the string. This is what separates strings from varaible or function names and tells Python to view it as a string value. Remember that everything in programming is extremely literal, so we have to be very explicit about what we mean when writing code.

We used a string in our very first line of code from the last lesson - can anyone remember what it was?

[11]:
```
# create string variable name, print type
name = "Marlena"
```

```
type(name)
```

[11]:
```
str
```

Like the numeric types, strings have to follow some basic rules for how they can behave in Python. While we can't perform mathematical addition with strings, we can combine strings together using the + operator. This is called **concatenating**.

[12]:
```
# print first name + last name
print("Marlena" + "Duda")

# add space in between
print("Marlena" + " " + "Duda")
```

```
MarlenaDuda
Marlena Duda
```

We can also use the multiplication operator on strings. Can anyone guess what that might do?

[13]:
```
# use multiplication operator on string
"Hello World " * 10
```

[13]:
```
'Hello World Hello World Hello World Hello World Hello World Hello World Hello
World Hello World Hello World Hello World '
```

However, not all the operators we learned in the last lesson work with strings. We can't subtract or divide strings, and we also can't add numeric values to strings or multiply strings with other strings. Let's see what kinds of errors arise when we try to break these string rules.

[14]:
```
# uncomment these lines one by one to see the resulting error messages

#"Hello World" - "Hello"

#"Marlena"/"Duda"

#"Marlena" * "GWC"

#"GWC" + 2019
```

Though there is a clear distinction between numeric and string variables, string variables can contain numbers, which are seen as numeric characters in this context.

[15]:
```
# create string variable date with today's date
date = "7/15/2019"
```

2.0.3 Booleans

One last basic data type in Python is called the boolean, or `bool`, which is a special type of variable that can take only on the value of `True` or `False`. Since it only has two possible values, a boolean is referred to as a *binary* variable.

[16]:
```
# create boolean variable and print type
approved = True
type(approved)
```

[16]:
```
bool
```

Boolean variables can be assigned directly using the special reserved words of `True` or `False` like we saw above, but they can also be generated as the result of comparison operators:

`<` less than

`>` greater than

`<=` less than or equal to

`>=` greater than or equal to

`==` equal to

`!=` not equal to

Pay special attention to the `==` operator - this tests whether or not two values are exactly equal to each other, and is an important distinction from the `=` operator, which assigns a value to a variable.

[17]:
```
# generate boolean variables using comparison operators
check_equal = apples == 7
print(check_equal)
type(check_equal)
```

```
False
```

[17]:
```
bool
```

While we've seen that booleans take on the values of `True` or `False`, they also have a numeric representation: `True` = 1 and `False` = 0. As such, boolean variables follow mathematical rules and can be manipulated using mathematical operators. This can be useful when you want to quickly count up the number of `True` values in a list, for example. Let's play with this below.

[18]:
```
yes = True
no = False

# add, substract, multiply boolean variables
print(no - yes)
print(yes + no)
print(yes * no)
```

```
# add, subtract, multiple boolean and int variables
print(no - 10)
print(yes + 10)
print(yes * 10)
```

```
-1
1
0
-10
11
10
```

Now you're an expert in the basic Python data types: * `int`: integer * `float`: decimal number * `string`: text * `bool`: True/False