KEY_Lesson08_Logic

January 11, 2020

1 Logic

Previously, we learned about Booleans. Booleans are variables that can take two values: `True` or `False`.

We can make a variable a Boolean by setting it equal to `True` or `False`, but as we discussed earlier we can also use comparison operators to generate

`<` less than

`>` greater than

`<=` less than/equal to

`>=` greater than or equal to

`==` equal to

`!=` not equal to

Let's create a basic list to get started

[2]:
```
# create a list called my_list with numbers 1,2,3 inside
my_list = [1, 2, 3]
```

Let's say a list with more than 2 items is considered long. Using the `len` function, we can create a Boolean value that tells us if our list is long or short.

[3]:
```
# create a boolean called long_list that is True if our list has more than 2␣
↪items
long_list = len(my_list) > 2

# print the value of long_list
print(long_list)
```

```
True
```

What if we wanted to create a Boolean that had the opposite value?

Certain pairs of operators are opposites of one another.

`>` and `<=` are opposites

`<` and `>=` are opposites

`==` and `!=` are opposites

So, since we used `>` to figure out if our list was long, we can use `<=` to figure out if our list is short.

[4]:
```
# create a boolean called short_list that is False if our list doesn't have␣
↪more than 2 items
short_list = len(my_list) <= 2

# print the value of short_list
print(short_list)
```

```
False
```

One cool thing about Python is that we can use words in place of some of the operators we talked about earlier. The operator `==` is used to tell if two things are equal, but we can also use `is` in its place.

note to editor: I'm not sure if introducing `is` is a good idea or not given it only works in place of `==` in certain circumstance. I think it's useful to know it exists but obviously explaining when to use it would be too advanced, at least for this lesson.

First, let's use the `==` operator to test if the first item of our list is equal to 1.

[5]:
```
# use == to see if the first item of my_list equals 1
my_list[0] == 1
```

[5]:
```
True
```

Now, let's do the same thing using the `is` operator

[6]:
```
# use is to see if the first item of my_list equals 1
my_list[0] is 1
```

[6]:
```
True
```

In most cases, it doesn't matter whether you use `is` or `==`. Using `is` might help you avoid accidentally using `=` instead of `==`, which is a common mistake.

Another useful python keyword is `not`. Putting `not` in front of a boolean will flip the value to its oposite—remember that `True` and `False` are opposites.

[7]:
```
# use the "not" keyword to calculate the opposite of True
not True
```

[7]:
```
False
```

The `not` keyword can be used with logical operators. Recall that the operator `!=` determines if two things are not equal to each other. We learned above that `is` can be used to compare two objects, too. Then, we can use `is` and `not` together to see if two values are different.

[8]:
```
# use != to see if the first item of my_list is not equal to 5
my_list[0] != 5
```

[8]:
```
True
```

[9]:
```
# use "is not" to see if the first item of my_list is not equal to 5
my_list[0] is not 5
```

[9]:
```
True
```

Just like comparison operators such as `>` and `<=` can work on things like strings and integers (and booleans, too!), there are operators just for booleans.

First, we'll look at `or`.

`or` looks at the two booleans and returns `True` if **at least one** is `True`, and otherwise returns `False`.

[10]:
```
# print the value of True or True
print(True or True)

# print the value of True or False
print(True or False)

# print the value of False or False
print(False or False)
```

```
True
True
False
```

Another really useful boolean operator is `and`.

`and` works similarly to `or`, except it only returns `True` if **both** of the booleans are `True`.

[11]:
```
# print the value of True and True
print(True and True)

# print the value of True and False
print(True and False)

# print the value of False and False
print(False and False)
```

```
True
False
False
```

One kind of problem that `or` and `and` can be useful for solving is testing if a value is inside a certain range.

Say we wanted to see if the length of our list was greater than 5 or less than 4—so any length except 4 or 5.

[12]:
```
# use "or" to see if my_list has less than 4 or more than 5 items
len(my_list) > 5 or len(my_list) < 4
```

[12]:
```
True
```

Say we wanted to solve the opposite problem—we want to see if the length of our list is exactly 4 or 5.

[13]:
```
# use "and" to see if my_list has 4 or 5 items
len(my_list) >=4 and len(my_list) <= 5
```

[13]:
```
False
```

We see that the length of our list **is** less than or equal to 5, but it isn't greater than or equal to 4, so the value is `False`.

This is a bit of a tricky example, so it may take a minute or two for it to sink in. If it is confusing, try making a list with two columns - one representing `>= 4` and one representing `<= 5`. For different example values, place a check in each column if that condition is true, and an X if it is false - only values where all columns have checks will result in `True` using the `and` operator.

Challenge: What about when we use the `or` operator?

Great job! You just learned about logic in Python! You learned: - How to create a boolean using operators - That some operators (for example, `>` and `<=`) are "opposites" - That keywords like `is` and `not` can be used in place of symbol-based operators - How to use `or` and `and` to combine booleans