Conditional statements

Sunday, May 5, 2019 10:43 AM

So far, we know how to:

- write some basic statements.
- do the same thing to every element of a list.
- create and print structures.

What if we want to change what we do depending upon the data we are given?

Example 1: we have a list of weights, and want to compute the average of the weights.

```
weights = [10, 0, 23, 34, 21, 20, 0, 17, 2]
sum = 0
count = 0
for w in weights:
    sum = sum + w
    count = count + 1
print(sum/count)
```

We keep a running count of the number of elements, and thus can divide by that number.

A simple application of the "if" statement: missing data.

Example 2: we have a list of weights, where if we don't know the weight, the value 0 is entered. What is the average of the *known* weights?

```
weights = [10, 0, 23, 34, 21, 20, 0, 17, 2]
sum = 0
count = 0
for w in weights:
```

```
      if w > 0:
         sum = sum + w
         count = count + 1
   print(sum/count)
```

In the above, the statements:

```
   sum = sum + w
   count = count + 1
```

are only executed if w > 0.

Example 3: Python has a much more elegant way to deal with missing data: the value **None**. Consider the same data, but with missing data replaced by None:

```
   weights = [10, None, 23, 34, 21, 20, None, 17, 2]
```

This denotes that we don't know the weights whose value is None. Then we can write:

```
   sum = 0
   count = 0
   for w in weights:
      if w is not None:
         sum = sum + w
         count = count + 1
   print(sum/count)
```

This is a very common way to code that data is missing in Python.

Common tests:

a > b	a is greater than b
a < b	a is less than b
a >= b	a is greater than or equal to b
a <= b	a is less than or equal to b
a == b	a is equal to b

a != b	a is not equal to b
a is None	a does not have a value
a is not None	a has a value

Tests can be combined with "logical operators"

- s and t: both s and t are True
- s or t: either s or t is True
- not s: s is False
- try the following

```
print(False and False)
print(False and True)
print(False or True)
print(True and True)
print(not True)
```

  to get a feel for these operators.

"if" statements take an expression that is either True or False. You can learn about if statements by printing the value of these expressions.

- Note that

```
print(None is not None)
```

  prints `False`, while

```
print(1 is not None)
```

  prints `True`
- Also, note that

```
print(2 > 0)
```

  prints `True`, while

```
print(0 > 0)
```

  prints `False`

You can use this to learn about how Python does logic.