Week 6: Conditionals, Iteration

Data 8 Tutoring

# 1 Conditional Statements and Iteration

## Key Concepts

**Conditional Statements**

We can use conditional statements and iteration to create functions that perform different operations based on certain conditions. As a reference, here is how conditional statements work:

if x > 10:

Do something

elif x > 5:

Do something

else:

Do something

**Iteration**

For loops in Python allow us to do two different operations. First, they allow us to iterate through arrays, manipulating each element as we wish. Alternatively, we can use for loops to repeat lines of code many times. Examples of how for loops can be used are below.

for item in some\_array:

print(item)

or

for i in np.arange(1000):

print(“Hello”)

## Practice Problems

**Question 1.** Examine the function, then answer the questions below. It has been written purposely vague!

def mystery\_function(x):

if (x > 0):

return “Positive”

elif (x < 0):

return “Negative”

else:

return “Neither”

**1.1** What would mystery\_function(10) return?

“Positive”

**1.2** What does mystery\_function(-1) return?

“Negative”

**1.3** What does mystery\_function(0) return?

“Neither”

**Question 2.** The for loop statement below stores the length of each name in names in a new array called lengths.

lengths = make\_array()

names = make\_array(‘Bob’, ‘Sarah’, ‘Michael’, ‘Sam’)

for name in names:

lengths = np.append(lengths, len(name))

**2.1** For each iteration below, fill in the value of name as well as what lengths looks like.

Iteration 1: name = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ , lengths = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Iteration 2: name = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ , lengths = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Iteration 3: name = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ , lengths = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Iteration 4: name = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ , lengths = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Iteration 1: name = ‘Bob’ , lengths = array(3)

Iteration 2: name = ‘Sarah’ , lengths = array(3, 5)

Iteration 3: name = ‘Michael’ , lengths = array(3, 5, 7)

Iteration 4: name = ‘Sam’ , lengths = array(3, 5, 7, 3)

**2.2** Now, let’s say that instead of storing lengths, we want to store the name as long as the length of the name is greater than 4. Fill in the following for loop statement such that longer contains these names.

longer = make\_array()

for name in \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_:

if \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_:

longer = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

longer = make\_array()

for name in names:

if len(name) > 4:

longer = np.append(longer, name)

**2.3** What names would longer contain after the for loop executes?

longer would contain the names Sarah and Michael.

**2.4** Finally, look at this last for loop below. What values does i take on throughout? How is i used as compared to the way name is used in the previous for loops?

counter = 0

for i in np.arange(1000):

counter = counter + 1

i takes on values that go from 0 to 999. In the first iteration, i takes on the value of 0, in the second, it takes on the value of 1, etc. The way i is used here is different than the way name was used in the previous for loop because we cared about the value that name took on throughout the for loop. For this for loop, we don’t really care about i - we just want to repeat a line of code 1000 times. We’ll use for loops like the one above many times throughout this course for the purpose of simulations!

**Question 3.** Suppose you have an array called salaries, containing the salary information of 5 individuals. You would like to determine what percentage of the total salaries each individual's salary comprises. You want to output an array, proportion where the ith element of proportion corresponds to what percentage of the total salary salary.item(i) is.

For example, if salaries was equal to an array [1,2,3,1,3], then proportion.item(0) would be 0.1.

**3.1** Your friend writes some code, but it doesn’t work! Find the error that your friend made. What would the code output if executed as is? How would you fix it?

salaries = make\_array(25, 50, 100, 25, 100)

for salary in salaries:

proportion = make\_array()

total = sum(salaries)

percentage = salary/total

proportion = np.append(proportion, percentage)

This code makes a new empty array and appends to it with each iteration of the for loop. Therefore, at the end of iteration, the proportion array will only have a single element, which represents the proportion of the total that the last element is.

In order to fix this, we would want to move the line proportion = make\_array() outside the for loop, so we can append to the same array every time. That way, we’ll have an array of each element’s proportion of the total.

**3.2** You fix the error described above, but in doing so, break something else. Again, find the error in the code below. What would the code output if executed as is? How would you fix it?

salaries = make\_array(25, 50, 100, 25, 100)

proportion = make\_array()

for salary in salaries:

total = sum(salaries)

percentage = salary/total

np.append(proportion, percentage)

This code as is appends to the proportion array at the end of the for loop, but doesn’t re-assign the the name proportion to the newly appended array. Therefore, it’ll append to proportion every time, but it won’t keep track of the updated proportion. At the end of the for loop, proportion will be empty, since we never re-assign what proportion is.

In order to fix this, we want to reassign proportion to the appended version of proportion, by replacing the last line with proportion = np.append(proportion, percentage).