Exemplar_ For loops

January 7, 2024

Exemplar: For loops

0.1 Introduction

As a data professional, you may need to use methods that involve repetition. For example, when analyzing customer feedback from surveys, you may need to compare averages across categories. In Python, using iterative statements helps automate this task and makes them more efficient.

In this lab, you will practice writing iterative statements using `for` loops.

0.2 Tips for completing this lab

As you navigate this lab, keep the following tips in mind:

- `### YOUR CODE HERE ###` indicates where you should write code. Be sure to replace this with your own code before running the code cell.
- Feel free to open the hints for additional guidance as you work on each task.
- To enter your answer to a question, double-click the markdown cell to edit. Be sure to replace the "[Double-click to enter your responses here.]" with your own answer.
- You can save your work manually by clicking File and then Save in the menu bar at the top of the notebook.
- You can download your work locally by clicking File and then Download and then specifying your preferred file format in the menu bar at the top of the notebook.

0.3 Task 1: Iterating with `if`, `elif`, and `else`

In your work as an analyst, you have received customer feedback for a newly launched product. The feedback is a numerical grade on a scale of 1-10, with 10 being the best. Understanding the number of customers who responded negatively, neutrally, or positively will help determine how to improve the product.

- Define a function called `score_counter` that accepts the following argument:
  - `score_list` - a list of customer-submitted scores, where each score is an integer, 1-10
- The function must iterate over the scores in `score_list` and bin them into three bins:
  - Negative (scores of 1-5)

  - Neutral (scores of 6-8)
  - Positive (scores of 9-10)

- The output of the function must be three statements, each on its own line:
  1. `'Negative: {number_of_negative_scores}'`
  2. `'Neutral: {number_of_neutral_scores}'`
  3. `'Positive: {number_of_positive_scores}'`

Example:

```
 [IN] score_counter([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
[OUT] 'Negative: 5'
      'Neutral: 3'
      'Positive: 2'
```

Note that there is more than one way to solve this problem.

[1]:
```
### YOUR CODE HERE ###
def score_counter(score_list):
    negative_scores = 0
    neutral_scores = 0
    positive_scores = 0

    for score in score_list:
        if score >= 9:
            positive_scores += 1
        elif score >= 6:
            neutral_scores += 1
        else:
            negative_scores += 1

    print('Negative:', negative_scores)
    print('Neutral:', neutral_scores)
    print('Positive:', positive_scores)
```

Hint 1

Refer to what you learned about using `for` loops and `if, elif, else` statements.

Hint 2

- Counters and print statements should not be part of the `for` loop body.
- The body of the `for` loop should be indented as a block below the `for` statement. Each conditional statement within the body of the `for` loop should follow its own indentation syntax.

Hint 3

- Use a counter for each category you're counting and make sure to increment one of the counters by 1 for each iteration of the loop.

- Use the conditional statements `if`, `elif`, and `else` to sort the responses into their buckets. Make sure to include the correct logical and comparitive operators during the branching instructions.

0.3.1 Test your function

Test your function against the following cases by running the cell below.

[2]:
```
# RUN THIS CELL TO TEST YOUR FUNCTION

import random
random.seed(42)

possible_scores = list(range(1,11))
score_list1 = random.choices(possible_scores, weights=[8,8,8,8,8,3,3,4,20,30],␣
↪k=10)
score_list2 = random.choices(possible_scores, weights=[1,2,3,4,5,10,15,15,7,9],␣
↪k=450)
score_list3 = random.choices(possible_scores, weights=[1,2,3,4,4,5,5,10,15,25],␣
↪k=10000)

print('Test 1:')            # SHOULD OUTPUT (neg, neut, pos):
score_counter(score_list1)  # 5, 1, 4
print('\nTest 2:')
score_counter(score_list2)  # 85, 253, 112
print('\nTest 3:')
score_counter(score_list3)  # 1935, 2652, 5413
```

```
Test 1:
Negative: 5
Neutral: 1
Positive: 4

Test 2:
Negative: 85
Neutral: 253
Positive: 112

Test 3:
Negative: 1935
Neutral: 2652
Positive: 5413
```

0.4 Task 2: Create a `for` loop using `if`, `not`, and `in`

There is concern that some of the customer feedback scores are fake. One way to check the integrity of the feedback is to identify how many scores came from verified customers.

In this task, you are provided with two lists: a list of verified customer IDs and a list of all IDs that left feedback scores. You must validate the data by cross-checking the lists against each other.

- Define a function called `id_validator` that takes two lists as arguments:
  - `verified_ids` - a list containing IDs of verified customers
  - `feedback_ids` - a list containing unique IDs of all posters of feedback scores
- The output of the function must be two statements, each on its own line:
  1. `'{number_of_unverified_ids} of {number_of_feedback_ids} IDs unverified.'`
  2. `'{percent} unverified.'`

Example 1:

```
 [IN]  id_validator(verified_ids=['1', '2'], feedback_ids=['1', '2', '3'])
[OUT]  '1 of 3 IDs unverified.'
       '33.33% unverified.'
```

Explanation:

One of the IDs (`'3'`) in `feedback_ids` is not in the list of `verified_ids`.
1/3 of the IDs in `feedback_ids` are unverified.

Example 2:

```
 [IN]  id_validator(verified_ids=['1a', '2b', '3c'], feedback_ids=['1a', '4d'])
[OUT]  '1 of 2 IDs unverified.'
       '50.0% unverified.'
```

Explanation:

One of the IDs (`'4d'`) in `feedback_ids` is not in the list of `verified_ids`.
1/2 of the IDs in `feedback_ids` are unverified.

Note that there is more than one way to solve this problem.

[3]:
```
### YOUR CODE HERE ###
def id_validator(verified_ids, feedback_ids):
    unverified_feedback = 0
    for id in feedback_ids:
        if id not in verified_ids:
            unverified_feedback += 1
    percent_unverified = unverified_feedback/len(feedback_ids)*100
    print(unverified_feedback, 'of', len(feedback_ids), 'IDs unverified.')
    print(str(round(percent_unverified, 2)) + '% unverified.')
```

Hint 1

Refer to what you learned about using `for` loops, `if` statements, and `not` and `in` operators.

Hint 2

Any lines of code within a `for` loop's body should be indented, including other conditional statements.

Hint 3

- Use a counter variable to keep track of the number of IDs you're counting.
- Use an `if` statement to check if an ID from one list is in, or not in, the other list.

0.4.1 Test your function

Test your function against the following cases by running the cell below.

[4]:
```
# RUN THIS CELL TO TEST YOUR FUNCTION
import ada_c2_labs as lab

# TEST SCENARIOS:                                    # SHOULD OUTPUT:
print('Test 1:')
ver1, fb1 = lab.lists_gen(8, 20, 15, 15)             # 4 of 15 IDs unverified.
id_validator(ver1, fb1)                              # 26.67% unverified.

print('\nTest 2:')
ver2, fb2 = lab.lists_gen(8, 1000, 900, 600)         # 357 of 900 IDs unverified.
id_validator(ver2, fb2)                              # 39.67% unverified.

print('\nTest 3:')
ver3, fb3 = lab.lists_gen(8, 15000, 14925, 13788)    # 1208 of 14925 IDs␣
 ↪unverified.
id_validator(ver3, fb3)                              # 8.09% unverified.
```

```
Test 1:
4 of 15 IDs unverified.
26.67% unverified.

Test 2:
357 of 900 IDs unverified.
39.67% unverified.

Test 3:
1208 of 14925 IDs unverified.
8.09% unverified.
```

1 Task 3: Create a nested loop

You've been asked to calculate the total value of all purchases made by each customer to understand how many customers have spent $100+.

- Write a function called `purchases_100` that accepts the following argument:

  - `sales` - a list of lists where each inner list contains the prices of items purchased by a unique customer.
- The function must return the number of customers whose purchases summed to $100 or more.

Example:

```
sales = [[2.75], [50.0, 50.0], [150.46, 200.12, 111.30]]

 [IN]  purchases_100(sales)
[OUT]  2
```

Note that there is more than one way to solve this problem.

[5]:
```
### YOUR CODE HERE ###

def purchases_100(sales):
    count = 0                              # Set a counter of totals > 100
    for customer in sales:                 # Loop over each inner list
        customer_total = 0                 # Set 0 value of purchases for the␣
↪inner list
        for purchase in customer:          # For price in inner list:
            customer_total += purchase     # Add the price to value of purchases␣
↪for inner list
        if customer_total >= 100:          # After looping over all prices in␣
↪inner list, if
                                           # total >= 100
            count+=1                       # Increment the counter
    return count
```

Hint 1

Refer to what you learned about nested `for` loops.

Hint 2

Try setting: * A counter for the number of inner lists in `sales` that sum to >= 100 * A variable that increments with each price in a given inner list to capture a final sum of the contents of that list.

Hint 3

- One approach would be to:
  1. Set a counter to 0
  2. Loop over each customer in the sales list
  3. Set the customer's total purchase price to 0
  4. For each purchase price in that customer's list, increase the customer's total purchase price by that price
  5. If the customer's total purchse price >= 100, increment the counter by 1
  6. Return the counter

1.0.1 Test your function

Test your function against the following cases by running the cell below.

[6]:
```
# RUN THIS CELL TO TEST YOUR FUNCTION
import ada_c2_labs as lab
sales1 = lab.sales_data_generator(n_customers=10, seed=1)    # SHOULD OUTPUT:
print('Test 1:', purchases_100(sales1))                      # 5

sales2 = lab.sales_data_generator(n_customers=150, seed=18)
print('Test 2:', purchases_100(sales2))                      # 46

sales3 = lab.sales_data_generator(n_customers=1275, seed=42)
print('Test 3:', purchases_100(sales3))                      # 470
```

```
Test 1: 5
Test 2: 46
Test 3: 470
```

2 Conclusion

What are your key takeaways from this lab?

- Iterative statements play a major role in automating repetitive analyses.
- You can use `for` loops to allow you to repeat a process a specified number of times.
- `if`, `elif`, `else`, logical, and comparitive operators can be used together within a `for` loop to specify which values to iterate through.
- Python built in functions are versatile and can be applied within loops, functions, or stand alone.

Congratulations! You've completed this lab. However, you may not notice a green check mark next to this item on Coursera's platform. Please continue your progress regardless of the check mark. Just click on the "save" icon at the top of this notebook to ensure your work has been logged.