Exemplar_ While loops

January 7, 2024

Exemplar: While loops

0.1 Introduction

Data analysts often work with code that requires repetition. In Python, iterative statements can help automate repetitive processes to make them more efficient.

In this lab, imagine you have transitioned fully into marketing for the movie theater. One of your tasks is analyzing customer purchases at the concession stand and automating messages to alert staff when a customer has earned a coupon.

You will practice using while loops to iterate through data and perform repeated comparisons.

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: Iterate with a `while` loop

While continuing your work as an analyst for a movie theater, you've been asked to create a function that will require a `while` loop. To prepare for the job, first practice using `while` loops.

In this warmup task:

- Create a variable called `candy_purchased` that will act as a counter. Initialize it with a starting value of `0`.
- Write a `while` loop that will continue iterating until `5` candies have been purchased.
  - Each iteration of the loop must print `'Candy purchased: {number}'` and increment `candy_purchased` by one.

Expected output:

```
Candy purchased: 0
Candy purchased: 1
Candy purchased: 2
Candy purchased: 3
Candy purchased: 4
Candy purchased: 5
```

[1]:
```
# Assign `candy_purchased` to an initial value of 0 to act as a counter
candy_purchased = 0

# Create an iterative statement using `while` and `candy_purchased`
# Display how many purchases have been made with each iteration

while candy_purchased <= 5:
    print('Candy purchased: ' + str(candy_purchased))
    # Increment the candy_purchased counter
    candy_purchased += 1
```

```
Candy purchased: 0
Candy purchased: 1
Candy purchased: 2
Candy purchased: 3
Candy purchased: 4
Candy purchased: 5
```

Hint 1

Refer back to what you learned about iterating using `while` loops and comparison operators.

Hint 2

- After creating a counter variable, you can use a `while` loop to iterate while a condition is true.
- Check your syntax and indentation.
- Make sure to increment the `candy_purchased` counter with each iteration of the loop.
- If your output isn't matching the example output, reconsider the order of your operations in the body of the `while` loop.

Hint 3

- The `while` statement can be declared either for `candy_purchased <= 5` or `candy_purchased < 6`.
- For each iteration:
  - First, print `'Candy purchased: {number}'`
  - Then, increment `candy_purchased` by one

1 Task 2: Iterate using `while` and `if`

In the next warmup task, you'll use a `while` loop that has an `if` statement and some Boolean logic.

- Write a `while` loop where `candy_purchased` begins at `0` and stops after 100 purchases have been made. Your code must:
  - Increment `candy_purchased` by one with each iteration
  - Print `'Candy purchased: {number}'` only when a multiple of 10 purchases have been made

Expected output:

```
Candy purchased: 0
Candy purchased: 10
Candy purchased: 20
Candy purchased: 30
Candy purchased: 40
Candy purchased: 50
Candy purchased: 60
Candy purchased: 70
Candy purchased: 80
Candy purchased: 90
Candy purchased: 100
```

[2]:
```
candy_purchased = 0

while candy_purchased <= 100:
    if candy_purchased % 10 == 0:
        print('Candy purchased: ' + str(candy_purchased))
    candy_purchased += 1
```

```
Candy purchased: 0
Candy purchased: 10
Candy purchased: 20
Candy purchased: 30
Candy purchased: 40
Candy purchased: 50
Candy purchased: 60
Candy purchased: 70
Candy purchased: 80
Candy purchased: 90
Candy purchased: 100
```

Hint 1

Refer back to what you learned about comparison operators, `while` loops, `if` statements, and Boolean logic.

Hint 2

After setting the counter to an initial value, initialize the `while` loop on a new line. Make sure the

body of the while loop is indented. Additional iterative statements, such as `if`, are contained in the body of the loop and indented to the same level, though their own body is indented further.

Hint 3

- Define the `while` loop to execute until either `candy_purchased <= 100` or `candy_purchased < 101`.
- Begin the body with an `if` statement that uses the modulo operator to check the `candy_purchased` variable for perfect divisibility by 10 and execute the print statement only when the `if` statement evaluates to `True`.
- Finally, increment the `candy_purchased` counter by one.

2 Task 3: Create a webpage timer function

Now that you have some practice constructing `while` loops, it's time to take on the project you've been asked to complete for the theater.

This task makes use of a function called `sleep()` from the `time` module. Whenever `sleep(n)` is executed, the program will pause for `n` seconds. Run the below cell as an illustrative example to help you understand.

[3]:
```
from time import sleep

n = 3
while n >= 0:
    print(n)
    sleep(1)    # Execution pauses for 1 second
    n = n - 1   # Decrement n by 1
```

```
3
2
1
0
```

For the purposes of this task, pretend that 1 second = 1 minute, so the above cell would represent a 3-minute countdown.

The movie theater has implemented online reservations through their website. From the seat selection page, customers have a limited amount of time to make their selection. You must write a function that counts down and reminds the user that they have limited time to make a selection. When time expires, it must print a timeout message.

- Instantiate a variable called `mins` and assign it a starting value of `10`.
- Write a `while` loop that uses the `sleep()` function and decrements `mins` from `10` minutes. For each minute:
  - Print how many minutes remain as an integer. **BUT:**
  - If there are 5 minutes left, print `Place your reservation soon! 5 minutes remaining.` instead of the integer.

- If there are 2 minutes left, print `Don't lose your seats! 2 minutes remaining.` instead of the integer.
- When zero minutes remain, print `User timed out.` instead of the integer.

Expected output:

```
10
9
8
7
6
Place your reservation soon! 5 minutes remaining.
4
3
Don't lose your seats! 2 minutes remaining.
1
User timed out.
```

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

[4]:
```
from time import sleep

### YOUR CODE HERE ###
mins = 10

while mins >= 0:
    if mins == 5:
        print('Place your reservation soon! 5 minutes remaining.')
    elif mins == 2:
        print('Don\'t lose your seats! 2 minutes remaining.')
    elif mins == 0:
        print('User timed out.')
    else:
        print(mins)
    mins -=1
    sleep(1)
```

```
10
9
8
7
6
Place your reservation soon! 5 minutes remaining.
4
3
Don't lose your seats! 2 minutes remaining.
1
User timed out.
```

Hint 1

- There are four possible print statements to consider for each minute. Consider an if, elif, elif, else branching scheme.
- Test for the special cases first (`mins == 5`, `mins == 2`, `mins == 0`).

Hint 2

- If your code is stuck in an infinite loop, check to make sure that you're decrementing the `mins` variable by one each time a condition that causes a print statement is satisfied.
- If you're having trouble with the `sleep()` function, try writing the code without it. If you can get the code to work without it, experiment with adding the `sleep()` function in different places to see how it affects the code's execution.

Hint 3

One approach is to set a `while` loop to execute when `mins >= 0`. For each iteration: * Check if `mins == 5`. If it does, print the designated message, then decrement `mins` by one and sleep(1). * Check if `mins == 2`. If it does, print the designated message, then decrement `mins` by one and sleep(1). * Check if `mins == 0`. If it does, print the designated message, then decrement `mins` by one. * If none of the above conditions are met, print `mins`, decrement its value by one, and sleep(1).

2.1 Conclusion

What are your key takeaways from this lab?

- Iterative statements play a major role in automating processes that need to be repeated.
- `while` loops allow you to repeat a process until a specific condition is no longer true.
- Comparison operators can be used to check one value against another.
- Variables can be set as counters and incremented by different amounts to help control your `while` loop.

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.