Annotated follow-along guide_Loops and strings

May 27, 2024

1 Annotated follow-along guide: Loops and strings

This notebook contains the code used in the instructional videos from Module 3: Loops and strings.

1.1 Introduction

This follow-along guide is an annotated Jupyter Notebook organized to match the content from each module. It contains the same code shown in the videos for the module. In addition to content that is identical to what is covered in the videos, you'll find additional information throughout the guide to explain the purpose of each concept covered, why the code is written in a certain way, and tips for running the code.

As you watch each of the following videos, an in-video message will appear to advise you that the video you are viewing contains coding instruction and examples. The in-video message will direct you to the relevant section in the notebook for the specific video you are viewing. Follow along in the notebook as the instructor discusses the code.

To skip directly to the code for a particular video, use the following links:

1. **Section ??**
2. **Section ??**
3. **Section ??**
4. **Section ??**
5. **Section ??**
6. **Section ??**

1. Introduction to while loops

[1]:
```
# Instantiate a counter.
x = 0

# Create a while loop that prints "not there yet," increments x by 1, a
# and prints x until x reaches 5.
while x < 5:
    print('Not there yet, x=' + str(x))
    x = x + 1
    print('x=' + str(x))
```

```
Not there yet, x=0
x=1
Not there yet, x=1
x=2
Not there yet, x=2
x=3
Not there yet, x=3
x=4
Not there yet, x=4
x=5
```

[2]:
```python
# Import the random module to be able to create a (pseudo) random number.
import random

number = random.randint(1,25)                    # Generate random number
number_of_guesses = 0                            # Instantiate guess counter

while number_of_guesses < 5:
    print('Guess a number between 1 and 25: ')   # Tell user to guess number
    guess = input()                              # Produce the user input field
    guess = int(guess)                           # Convert guess to integer
    number_of_guesses += 1                       # Increment guess count by 1

    if guess == number:                          # Break while loop if guess is␣
 ↪correct
        break
    elif number_of_guesses == 5:                 # Break while loop if guess␣
 ↪limit reached
        break
    else:                                        # Tell user to try again
        print('Nope! Try again.')

# Message to display if correct
if guess == number:
    print('Correct! You guessed the number in ' + str(number_of_guesses) + '␣
 ↪tries!')
# Message to display after 5 unsuccessful guesses
else:
    print('You did not guess the number. The number was ' + str(number) + '.')
```

```
Guess a number between 1 and 25:
5
Guess a number between 1 and 25:
22
Guess a number between 1 and 25:
15
Guess a number between 1 and 25:
```

```
7
Guess a number between 1 and 25:
8
You did not guess the number. The number was 25.
```

2. Introduction to for loops

[3]:
```
# Example of for loop with range() function
for x in range(5):
    print(x)
```

```
0
1
2
3
4
```

[4]:
```
# Example of reading in a .txt file line by line with a for loop
with open('zen_of_python.txt') as f:
    for line in f:
        print(line)
print('\nI\'m done.')
```

```
# Get started with Python

# Week 1 - video 7 of 10

# Input/Output: Data comes from different places



#import this

The Zen of Python, by Tim Peters



Beautiful is better than ugly.

Explicit is better than implicit.

Simple is better than complex.

Complex is better than complicated.

Flat is better than nested.

Sparse is better than dense.
```

```
Readability counts.

Special cases aren't special enough to break the rules.

Although practicality beats purity.

Errors should never pass silently.

Unless explicitly silenced.

In the face of ambiguity, refuse the temptation to guess.

There should be one-- and preferably only one --obvious way to do it.

Although that way may not be obvious at first unless you're Dutch.

Now is better than never.

Although never is often better than *right* now.

If the implementation is hard to explain, it's a bad idea.

If the implementation is easy to explain, it may be a good idea.

Namespaces are one honking great idea -- let's do more of those!

I'm done.
```

3. Loops with multiple range parameters

[5]:
```
# Use a for loop to calculate 9!
product = 1
for n in range(1, 10):
    product = product * n

print(product)
```

```
362880
```

[6]:
```
# Define a function that converts Fahrenheit to Celsius.
def to_celsius(x):
     return (x-32) * 5/9

# Create a table of Celsius-->Fahrenheit conversions every 10 degrees, 0-100
for x in range(0, 101, 10):
     print(x, to_celsius(x))
```

```
0 -17.77777777777778
10 -12.222222222222221
```

```
20 -6.666666666666667
30 -1.1111111111111112
40 4.444444444444445
50 10.0
60 15.555555555555555
70 21.11111111111111
80 26.666666666666668
90 32.22222222222222
100 37.77777777777778
```

4. Work with strings

[7]:
```
# Adding strings will combine them.
'Hello' + 'world'
```

[7]:
```
'Helloworld'
```

[8]:
```
# Blank space ("whitespace") is its own character.
'Hello ' + 'world'
```

[8]:
```
'Hello world'
```

[9]:
```
# Including a whitespace when combining strings
'Hello' + ' ' + 'world'
```

[9]:
```
'Hello world'
```

[10]:
```
# Variables containing strings can be added.
greeting_1 = 'Hello '
greeting_2 = 'world'
greeting_1 + greeting_2
```

[10]:
```
'Hello world'
```

[11]:
```
# Strings can be multiplied by integers.
danger = 'Danger! '
danger * 3
```

[11]:
```
'Danger! Danger! Danger! '
```

[12]:
```
# Strings cannot be used with subtraction or division.
danger - 2
```

```
␣
↪---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call␣
↪last)
```

```
<ipython-input-12-50c2ed4b1733> in <module>
  1 # Strings cannot be used with subtraction or division
----> 2 danger - 2


TypeError: unsupported operand type(s) for -: 'str' and 'int'
```

[13]:
```
# Alternate single and double quotes to include one or the other in your string.
quote = '"Thank you for pressing the self-destruct button."'
print(quote)
```

```
"Thank you for pressing the self-destruct button."
```

[14]:
```
# \ is an escape character that modifies the character that follows it.
quote = "\"It's dangerous to go alone!\""
print(quote)
```

```
"It's dangerous to go alone!"
```

[15]:
```
# \n creates a newline.
greeting = "Good day,\nsir."
print(greeting)
```

```
Good day,
sir.
```

[16]:
```
# Using escape character (\) lets you express the newline symbol within a␣
 ↪string.
newline = "\\n represents a newline in Python."
print(newline)
```

```
\n represents a newline in Python.
```

[17]:
```
# You can loop over strings.
python = 'Python'
for letter in python:
    print(letter + 'ut')
```

```
Put
yut
tut
hut
out
nut
```

5. String slicing

[18]:
```
# The index() method returns index of character's first occurrence in string.
pets = 'cats and dogs'
pets.index('s')
```

[18]: 3

[19]:
```
# The index() method will throw an error if character is not in string.
pets.index('z')
```

```
␣
↪---------------------------------------------------------------------------

ValueError                                Traceback (most recent call␣
↪last)

<ipython-input-19-71be11ca684e> in <module>
  1 # The index() method will throw an error if character is not in␣
↪string
----> 2 pets.index('z')


ValueError: substring not found
```

[20]:
```
# Access the character at a given index of a string.
name = 'Jolene'
name[0]
```

[20]: 'J'

[21]:
```
# Access the character at a given index of a string.
name[5]
```

[21]: 'e'

[22]:
```
# Indices that are out of range will return an IndexError.
name[6]
```

```
␣
↪---------------------------------------------------------------------------

IndexError                                Traceback (most recent call␣
↪last)

<ipython-input-22-f2028b57f9ad> in <module>
```

```
      1 # Indices that are out of range will return an IndexError
----> 2 name[6]


IndexError: string index out of range
```

[23]:
```
# Negative indexing begins at the end of the string.
sentence = 'A man, a plan, a canal, Panama!'
sentence[-1]
```

[23]:
```
'!'
```

[24]:
```
# Negative indexing begins at the end of the string.
sentence[-2]
```

[24]:
```
'a'
```

[25]:
```
# Access a substring by using a slice.
color = 'orange'
color[1:4]
```

[25]:
```
'ran'
```

[26]:
```
# Omitting the first value of the slice implies a value of 0.
fruit = 'pineapple'
fruit[:4]
```

[26]:
```
'pine'
```

[27]:
```
# Omitting the last value of the slice implies a value of len(string).
fruit[4:]
```

[27]:
```
'apple'
```

[28]:
```
# The `in` keyword returns Boolean of whether substring is in string.
'banana' in fruit
```

[28]:
```
False
```

[29]:
```
# The `in` keyword returns Boolean of whether substring is in string.
'apple' in fruit
```

[29]:
```
True
```

6. Format strings

[30]:
```
# Use format() method to insert values into your string, indicated by braces.
name = 'Manuel'
number = 3
print('Hello {}, your lucky number is {}.'.format(name, number))
```

```
Hello Manuel, your lucky number is 3.
```

[31]:
```
# You can assign names to designate how you want values to be inserted.
name = 'Manuel'
number = 3
print('Hello {name}, your lucky number is {num}.'.format(num=number, name=name))
```

```
Hello Manuel, your lucky number is 3.
```

[32]:
```
# You can use argument indices to designate how you want values to be inserted.
print('Hello {1}, your lucky number is {0}.'.format(number, name))
```

```
Hello Manuel, your lucky number is 3.
```

[33]:
```
# Example inserting prices into string
price = 7.75
with_tax = price * 1.07
print('Base price: ${} USD. \nWith tax: ${} USD.'.format(price, with_tax))
```

```
Base price: $7.75 USD.
With tax: $8.2925 USD.
```

[34]:
```
# Use :.2f to round a float value to two places beyond the decimal.
print('Base price: ${:.2f} USD. \nWith tax: ${:.2f} USD.'.format(price,␣
 ↪with_tax))
```

```
Base price: $7.75 USD.
With tax: $8.29 USD.
```

[35]:
```
# Define a function that converts Fahrenheit to Celsius.
def to_celsius(x):
    return (x-32) * 5/9

# Create a temperature conversion table using string formatting
for x in range(0, 101, 10):
    print("{:>3} F | {:>6.2f} C".format(x, to_celsius(x)))
```

```
  0 F | -17.78 C
 10 F | -12.22 C
 20 F |  -6.67 C
 30 F |  -1.11 C
 40 F |   4.44 C
 50 F |  10.00 C
```

```
 60 F |  15.56 C
 70 F |  21.11 C
 80 F |  26.67 C
 90 F |  32.22 C
100 F |  37.78 C
```

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.