py101_lesson_3_medium_1

December 4, 2024

1 Lesson 3

1.1 Medium 1

1.2 Question 1

Let's do some "ASCII Art": a stone-age form of nerd artwork from back in the days before computers had video screens.

For this practice problem, write a program that outputs The Flintstones Rock! 10 times, with each line prefixed by one more hyphen than the line above it. The output should start out like this:

```
-The Flintstones Rock!
--The Flintstones Rock!
---The Flintstones Rock!
    ...
```

[3]:
```python
for i in range(1,11):
    print(f'{i * "-"}The Flintstones Rock!')
```

```
-The Flintstones Rock!
--The Flintstones Rock!
---The Flintstones Rock!
----The Flintstones Rock!
-----The Flintstones Rock!
------The Flintstones Rock!
-------The Flintstones Rock!
--------The Flintstones Rock!
---------The Flintstones Rock!
----------The Flintstones Rock!
```

1.3 Question 2

Alan wrote the following function, which was intended to return all of the factors of number:

```
def factors(number):
    divisor = number
    result = []
    while divisor != 0:
        if number % divisor == 0:
            result.append(number // divisor)
        divisor -= 1
    return result
```

Alyssa noticed that this code would fail when the input is a negative number, and asked Alan to change the loop. How can he make this work? Note that we're not looking to find the factors for negative numbers, but we want to handle it gracefully instead of going into an infinite loop.

Bonus Question: What is the purpose of number % divisor == 0 in that code?

[7]:
```
def factors(number):
    if number < 0:
        raise ValueError('A negative number was passed to factors(), which it␣
 ↪cannot handle.')
    divisor = number
    result = []
    while divisor != 0:
        if number % divisor == 0:
            result.append(number // divisor)
        divisor -= 1
    return result

factors(10)
factors(-10)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[7], line 13
     10     return result
     12 factors(10)
---> 13 factors(-10)

Cell In[7], line 3, in factors(number)
      1 def factors(number):
      2     if number < 0:
----> 3         raise ValueError('A negative number was passed to factors(),␣
  ↪which it cannot handle.')
      4     divisor = number
      5     result = []

ValueError: A negative number was passed to factors(), which it cannot handle.
```

The suggested answer just changes the while loop from != 0 to > 0, which I'm not sure "handles" negative numbers at all. It ignores them, and is hard to trace.

1.4 Question 3

Alyssa was asked to write an implementation of a rolling buffer. You can add and remove elements from a rolling buffer. However, once the buffer becomes full, any new elements will displace the oldest elements in the buffer.

She wrote two implementations of the code for adding elements to the buffer. What is the key difference between these implementations?

```
def add_to_rolling_buffer1(buffer, max_buffer_size, new_element):
    buffer.append(new_element)
    if len(buffer) > max_buffer_size:
        buffer.pop(0)
    return buffer

def add_to_rolling_buffer2(buffer, max_buffer_size, new_element):
    buffer = buffer + [new_element]
    if len(buffer) > max_buffer_size:
        buffer.pop(0)
    return buffer
```

1.4.1 Answer

The + operator creates a new list instead of adding to the original list.

1.5 Question 4:

What will the following two lines of code output?

```
print(0.3 + 0.6)
print(0.3 + 0.6 == 0.9)
```

1.5.1 Answer

0.9 True —— I'm wrong becuase floats aren't precise enough. math.isclose is a better option.

1.6 Question 5:

What do you think the following code will output?

```
nan_value = float("nan")

print(nan_value == float("nan"))
```

Bonus Question How can you reliably test if a value is nan?

1.6.1 Answer

I said false, which is correct. nan means not a number and you can't use equality to test for it. You can use `math.isnan()`

1.7 Question 6:

What is the output of the following code:

```
answer = 42

def mess_with_it(some_number):
    return some_number + 8

new_answer = mess_with_it(answer)

print(answer - 8)
```

1.7.1 Answer

34

1.8 Question 7

One day, Spot was playing with the Munster family's home computer, and he wrote a small program to mess with their demographic data:

```
munsters = {
    "Herman": {"age": 32, "gender": "male"},
    "Lily": {"age": 30, "gender": "female"},
    "Grandpa": {"age": 402, "gender": "male"},
    "Eddie": {"age": 10, "gender": "male"},
    "Marilyn": {"age": 23, "gender": "female"},
}

def mess_with_demographics(demo_dict):
    for key, value in demo_dict.items():
        value["age"] += 42
        value["gender"] = "other"
```

```
mess_with_demographics(munsters)
```

What did this do?

1.8.1 Answer

.items() returns a "view object," but the view object is dynamically updated *and* dynamically updates the dictionary. SO this does update all of the values in question.

1.9 Question 8

Function and method calls can take expressions as arguments. Suppose we define a function named rps as follows, which follows the classic rules of the rock-paper-scissors game, but with a slight twist: in the event of a tie, it just returns the choice made by both players.

```
def rps(fist1, fist2):
    if fist1 == "rock":
        return "paper" if fist2 == "paper" else "rock"
    elif fist1 == "paper":
        return "scissors" if fist2 == "scissors" else "paper"
    else:
        return "rock" if fist2 == "rock" else "scissors"
```

What does the following code output:

```
print(rps(rps(rps("rock", "paper"), rps("rock", "scissors")), "rock"))
```

1.9.1 Answer

Paper: * (rps(rps(~~rps(“rock”, “paper”)~~, rps(“rock”, “scissors”)), “rock”)) * (rps(rps(**“paper”**, ~~rps(“rock”, “scissors”)~~), “rock”)) * (rps(~~rps(**“paper”, rock**)~~, “rock”)) * (rps(**paper**, “rock”) * paper

1.10 Question 9

Consider these two simple functions:

```
def foo(param="no"):
    return "yes"

def bar(param="no"):
    return (param == "no") and (foo() or "no")
```

What will the following function invocation return?

```
bar(foo())
```

1.10.1 Answer:

- bar(foo())
- bar(yes)
- return (param == “no”) and (foo() or “no”)
- return False and (“yes” or “no”)
- return False

1.11 Question 10

In Python, every object has a unique identifier that can be accessed using the id() function. This function returns the identity of an object, which is guaranteed to be unique for the object’s lifetime. For certain basic immutable data types like short strings or integers, Python might reuse the memory address for objects with the same value. This is known as “interning”.

Given the following code, predict the output:

```
a = 42
b = 42
c = a

print(id(a) == id(b) == id(c))
```

1.11.1 Answer

True. A and C both point to the same thing. I assume because of interning that B will have the same id

[ ]:
```

```