# py101_lesson_3_hard_1

December 4, 2024

## 1 PY 101 Lesson 3

### 1.1 Practice Problems: Hard

### 1.2 Question 1

Will the following functions return the same results?

```
def first():
    return {
        'prop1': "hi there",
    }

def second():
    return
    {
        'prop1': "hi there",
    }

print(first())
print(second())
```

#### 1.2.1 Answer:

No, because the dictionary isn't indented to be part of the return statement in second(). Specifically the opening curly brace.

### 1.3 Question 2

What does the last line in the following code output?

```
dictionary = {'first': [1]}
num_list = dictionary['first']
num_list.append(2)

print(num_list)
print(dictionary)
```

#### 1.3.1 Answer:

I think it's going to return * print(num_list) = [1, 2] * (dictionary) = {'first': [1]}

because num_list points at dictionary[‘first’] when it’s instantiated, but it’s not the same object.

---

I’m wrong, though. Here’s the text:

Since num_list is a reference to the original list in dictionary, appending to num_list modifies the list. Thus, the original dictionary is also updated.

If, instead of modifying the original dictionary, we want to modify num_list but not dictionary, we have a couple of options: * We can initialize num_list with a reference to a copy of the original list:

```
dictionary = {"first": [1]}
num_list = dictionary["first"].copy()
num_list.append(2)
```

- We can use list slicing which returns a new list:

```
dictionary = {"first": [1]}
num_list = dictionary["first"][:]
num_list.append(2)
```

### 1.4 Question 3:

Given the following similar sets of code, what will each code snippet print?

```
def mess_with_vars(one, two, three):
    one = two
    two = three
    three = one

one = ["one"]
two = ["two"]
three = ["three"]

mess_with_vars(one, two, three)

print(f"one is: {one}")
print(f"two is: {two}")
print(f"three is: {three}")
```

---

```
def mess_with_vars(one, two, three):
    one = ["two"]
    two = ["three"]
    three = ["one"]

one = ["one"]
two = ["two"]
three = ["three"]
```

```
mess_with_vars(one, two, three)

print(f"one is: {one}")
print(f"two is: {two}")
print(f"three is: {three}")
```

---

```
def mess_with_vars(one, two, three):
    one[0] = "two"
    two[0] = "three"
    three[0] = "one"

one = ["one"]
two = ["two"]
three = ["three"]

mess_with_vars(one, two, three)

print(f"one is: {one}")
print(f"two is: {two}")
print(f"three is: {three}")
```

#### 1.4.1 Answer

- First function: the values don't change. `{one}` is going to print `["one"]`, etc
- Second function: again, the values don't change. `{one}` is going to print `["one"]`, etc
- I initially thought this one was going to be the same thing, but this is one of those "lists are weird" kind of thing. Even though the value change happens in the function, I think specifically calling the element is changing the actual list. After some reading, reassigning the reference changes what the variable points to, but doesn't change the original object. In the third case, the function isn't changing the reference, but is actually changing the mutable object the reference points to.

### 1.5 Question 4

Ben was tasked to write a simple Python function to determine whether an input string is an IP address using 4 dot-separated numbers, e.g., 10.4.5.11.

Alyssa supplied Ben with a function named is_an_ip_number. It determines whether a string is a numeric string between 0 and 255 as required for IP numbers and asked Ben to use it. Here's the code that Ben wrote:

```
def is_dot_separated_ip_address(input_string):
    dot_separated_words = input_string.split(".")
    while len(dot_separated_words) > 0:
        word = dot_separated_words.pop()
        if not is_an_ip_number(word):
            break
```

```
    return True
```

Alyssa reviewed Ben's code and said, "It's a good start, but you missed a few things. You're not returning a false condition, and you're not handling the case when the input string has more or less than 4 components, e.g., 4.5.5 or 1.2.3.4.5: both those values should be invalid."

Help Ben fix his code.

[14]:
```
def is_an_ip_number(str):
    if str.isdigit():
        number = int(str)
        return 0 <= number <= 255
    return False

def is_dot_separated_ip_address(input_string):
    split_nums = input_string.split(".")
    if len(split_nums) != 4:
        return False

    for num in split_nums:
        if not is_an_ip_number(num):
            return False
    return True

print(is_dot_separated_ip_address("182.168.0.1") == True)
print(is_dot_separated_ip_address("8.8.8.8") == True)
print(is_dot_separated_ip_address("172.16.254.1") == True)
print(is_dot_separated_ip_address("255.255.255.255") == True)
print(is_dot_separated_ip_address("0.0.0.0") == True)
print(is_dot_separated_ip_address("256.100.50.25") == False)
print(is_dot_separated_ip_address("192.168.1") == False)
print(is_dot_separated_ip_address("192.168.0.256") == False)
print(is_dot_separated_ip_address("abc.def.ghi.jkl") == False)
print(is_dot_separated_ip_address("192.168..1") == False)
```

```
True
True
True
True
True
True
True
True
True
True
```

### 1.6 Question 5

What do you expect to happen when the greeting variable is referenced in the last line of the code below?

```
if False:
    greeting = "hello world"

print(greeting)
```

#### 1.6.1 Answer

I expect there to be a variable not defined error

[15]:
```
if False:
    greeting = "hello world"

print(greeting)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[15], line 4
      1 if False:
      2     greeting = "hello world"
----> 4 print(greeting)

NameError: name 'greeting' is not defined
```

[ ]:
```

```