Module 4 Graded Assessment

LATEST SUBMISSION GRADE

100%

1. The format_address function separates out parts of the address string into new strings: house_number and street_name, and returns: "house number X on street named Y". The format of the input string is: numeric house number, followed by the street name which may contain numbers, but never by themselves, and could be several words long. For example, "123 Main Street", "1001 1st Ave", or "55 North Center Drive". Fill in the gaps to complete this function.

1 / 1 point

```python
1 ▾ def format_address(address_string):
2     # Declare variables
3
4     # Separate the address string into parts
5     list1=address_string.split(" ")
6     # Traverse through the address parts
7     st=""
8 ▾  for x in range(len(list1)):
9 ▾    if(x!=0):
10        st+=list1[x]+" "
11      # Determine if the address part is the
12      # house number or part of the street name
13
14    # Does anything else need to be done
15    # before returning the result?
16
17    # Return the formatted string
18    return "house number {} on street named {}".format(list1[0],st.strip())
19
20  print(format_address("123 Main Street"))
21  # Should print: "house number 123 on street named Main Street"
22
23  print(format_address("1001 1st Ave"))
24  # Should print: "house number 1001 on street named 1st Ave"
25
26  print(format_address("55 North Center Drive"))
27  # Should print "house number 55 on street named North Center Drive"
28
```

Run

Reset

✓ Correct

Great work! You've remembered how to work with string methods and use variables for formatting output

2. The highlight_word function changes the given word in a sentence to its upper-case version. For example, highlight_word("Have a nice day", "nice") returns "Have a NICE day". Can you write this function in just one line?

1 / 1 point

```python
1  import re
2 ▾ def highlight_word(sentence, word):
3    return sentence.replace(word, word.upper())
4
5  print(highlight_word("Have a nice day", "nice"))
6  print(highlight_word("Shhh, don't be so loud!", "loud"))
7  print(highlight_word("Automating with Python is fun", "fun"))
8
```

Run

Reset

✓ Correct

Nice job! You're mastering your string skills!

3. A professor with two assistants, Jamie and Drew, wants an attendance list of the students, in the order that they arrived in the classroom. Drew was the first one to note which students arrived, and then Jamie took over. After the class, they each entered their lists into the computer and emailed them to the professor, who needs to combine them into one, in the order of each student's arrival. Jamie emailed a follow-up, saying that her list is in reverse order. Complete the steps to combine them into one list as follows: **the contents of Drew's list, followed by Jamie's list in reverse order**, to get an accurate list of the students as they arrived.

```
 1 ▾ def combine_lists(list1, list2):
 2
 3      return list2+list1[::-1]
 4      # Generate a new list containing the elements of list2
 5      # Followed by the elements of list1 in reverse order
 6
 7
 8   Jamies_list = ["Alice", "Cindy", "Bobby", "Jan", "Peter"]
 9   Drews_list = ["Mike", "Carol", "Greg", "Marcia"]
10
11   print(combine_lists(Jamies_list, Drews_list))
12
```

Run

Reset

✓ **Correct**

Excellent! You're using the list functions correctly, and it shows!

---

4. Use a list comprehension to create a list of squared numbers (n*n). The function receives the variables *start* and *end*, and returns a list of squares of consecutive numbers between *start* and *end* **inclusively**. For example, squares(2, 3) should return [4, 9].

```
 1 ▾ def squares(start, end):
 2      return [ x*x for x in range(start,end+1) ]
 3
 4   print(squares(2, 3)) # Should be [4, 9]
 5   print(squares(1, 5)) # Should be [1, 4, 9, 16, 25]
 6   print(squares(0, 10)) # Should be [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Run

Reset

✓ **Correct**

Right on! You're making the correct calculation, and using the correct range.

---

5. Complete the code to iterate through the keys and values of the car_prices dictionary, printing out some information about each one.

```
 1 ▾ def car_listing(car_prices):
 2      result = ""
 3 ▾    for x,y in car_prices.items():
 4          result += "{} costs {} dollars".format(x,y) + "\n"
 5      return result
 6
 7   print(car_listing({"Kia Soul":19000, "Lamborghini Diablo":55000, "Ford Fiesta"
     :13000, "Toyota Prius":24000}))
```

Run

Reset

✓ **Correct**

You got it! You've correctly gone through the items of the dictionary!

6. Taylor and Rory are hosting a party. They sent out invitations, and each one collected responses into dictionaries, with names of their friends and how many guests each friend is bringing. Each dictionary is a partial list, but Rory's list has more current information about the number of guests. Fill in the blanks to combine both dictionaries into one, with each friend listed only once, and the number of guests from Rory's dictionary taking precedence, if a name is included in both dictionaries. Then print the resulting dictionary.

```
1 ▾ def combine_guests(guests1, guests2):
2     # Combine both dictionaries into one, with each key listed
3
4       guests2.update(guests1)
5     # only once, and the value from guests1 taking precedence
6       return guests2
7   Rorys_guests = { "Adam":2, "Brenda":3, "David":1, "Jose":3, "Charlotte":2,
      "Terry":1, "Robert":4}
8   Taylors_guests = { "David":4, "Nancy":1, "Robert":2, "Adam":1, "Samantha":3,
      "Chris":5}
9
10  print(combine_guests(Rorys_guests, Taylors_guests))
11
```

Run

Reset

✓ Correct

You nailed it! You've figured out the best way to call the update() method, to have the values from the first dictionary added or updated over the second dictionary.

7. Use a dictionary to count the frequency of letters in the input string. Only letters should be counted, not blank spaces, numbers, or punctuation. Upper case should be considered the same as lower case. For example, count_letters("This is a sentence.") should return {'t': 2, 'h': 1, 'i': 2, 's': 3, 'a': 1, 'e': 3, 'n': 2, 'c': 1}.

```
1 ▾ def count_letters(text):
2     result = {}
3     # Go through each letter in the text
4 ▾   for letter in text:
5       # Check if the letter needs to be counted or not
6 ▾     if(letter.upper() >='A' and letter.upper() <='Z'):
7 ▾       if(letter.lower() not in result):
8           result[letter.lower()]=0
9         result[letter.lower()]+=1
10      # Add or increment the value in the dictionary
11    return result
12
13  print(count_letters("AaBbCc"))
14  # Should be {'a': 2, 'b': 2, 'c': 2}
15
16  print(count_letters("Math is fun! 2+2=4"))
17  # Should be {'m': 1, 'a': 1, 't': 1, 'h': 1, 'i': 1, 's': 1, 'f': 1, 'u': 1, 'n'
      : 1}
18
19  print(count_letters("This is a sentence."))
20  # Should be {'t': 2, 'h': 1, 'i': 2, 's': 3, 'a': 1, 'e': 3, 'n': 2, 'c': 1}
```

Run

Reset

✓ Correct

Woohoo! You've remembered the relevant string commands, and how to work with dictionaries.

8. What do the following commands return when animal = "Hippopotamus"?

```
1  >>> print(animal[3:6])
2  >>> print(animal[-5])
3  >>> print(animal[10:])
4
```

- ⚪ ppo, t, mus
- ⚪ ppop, o, s
- ⦿ pop, t, us
- ⚪ popo, t, mus

✓ **Correct**

You got it! When both parts of a string index range are included, the substring starts at first index and ends at second index minus 1. When the index is negative, the character is counted from the end of the string. When the second index is omitted, it goes until the end of the string.

9. What does the list "colors" contain after these commands are executed?

```
1  colors = ["red", "white", "blue"]
2  colors.insert(2, "yellow")
3
```

- ⦿ ['red', 'white', 'yellow', 'blue']
- ⚪ ['red', 'yellow', 'white', 'blue']
- ⚪ ['red', 'yellow', 'blue']
- ⚪ ['red', 'white', 'yellow']

✓ **Correct**

Right on! The insert command inserts the new element into the list at the specified index, shifting the other elements over afterwards.

10. What do the following commands return?

```
1  host_addresses = {"router": "192.168.1.1", "localhost": "127.0.0.1", "google":
   "8.8.8.8"}
2  host_addresses.keys()
3
```

- ⚪ {"router": "192.168.1.1", "localhost": "127.0.0.1", "google": "8.8.8.8"}
- ⚪ ["router", "192.168.1.1", "localhost", "127.0.0.1", "google", "8.8.8.8"]
- ⚪ ['192.168.1.1', '127.0.0.1', '8.8.8.8']
- ⦿ ['router', 'localhost', 'google']

✓ **Correct**

You got it! In dictionaries, the keys() command returns a list of just the keys, which is what this is.