

Practice Quiz: Strings

Total points 5

Question 1

The `is_palindrome` function checks if a string is a palindrome. A palindrome is a string that can be equally read from left to right or right to left, omitting blank spaces, and ignoring capitalization. Examples of palindromes are words like kayak and radar, and phrases like "Never Odd or Even". Fill in the blanks in this function to return `True` if the passed string is a palindrome, `False` if not.

1 / 1 point

```
1 def is_palindrome(input_string):
2     # We'll create two strings, to compare them
3     new_string = ""
4     reverse_string = ""
5     # Traverse through each letter of the input string
6     for letter in input_string:
7         # Add any non-blank letters to the
8         # end of one string, and to the front
9         # of the other string.
10        if letter != " ":
11            new_string = new_string + letter
12            reverse_string = letter + reverse_string
13    # Compare the strings
14    if new_string.lower() == reverse_string.lower():
15        return True
16    return False
17
18 print(is_palindrome("Never Odd or Even")) # Should be True
19 print(is_palindrome("abc")) # Should be False
20 print(is_palindrome("kayak")) # Should be True
```

True

False

True

Correct

Woohoo! You're quickly becoming the Python string expert!

Question 2

Using the `format` method, fill in the gaps in the `convert_distance` function so that it returns the phrase "X miles equals Y km", with Y having only 1 decimal place. For example, `convert_distance(12)` should return "12 miles equals 19.2 km".

1 / 1 point

```

1  def convert_distance(miles):
2      km = miles * 1.6
3      result = "{} miles equals {:.1f} km".format(miles, km)
4      return result
5
6  print(convert_distance(12)) # Should be: 12 miles equals 19.2 km
7  print(convert_distance(5.5)) # Should be: 5.5 miles equals 8.8 km
8  print(convert_distance(11)) # Should be: 11 miles equals 17.6 km

```

12 miles equals 19.2 km
 5.5 miles equals 8.8 km
 11 miles equals 17.6 km

Correct

Congrats! You're getting the hang of formatting strings, hooray!

Question 3

If we have a string variable named `Weather = "Rainfall"`, which of the following will print the substring or all characters before the "f"?

1 / 1 point

`print(Weather[:4])`

`print(Weather[4:])`

`print(Weather[1:4])`

`print(Weather[:"f"])`

Correct

Nice job! Formatted this way, the substring preceding the character "f", which is indexed by 4, will be printed.

Question 4

Fill in the gaps in the `nametag` function so that it uses the `format` method to return `first_name` and the first initial of `last_name` followed by a period. For example, `nametag("Jane", "Smith")` should return "Jane S."

1 / 1 point

```

1  def nametag(first_name, last_name):
2      return("{} {}.".format(first_name, last_name[0]))
3
4  print(nametag("Jane", "Smith"))
5  # Should display "Jane S."
6  print(nametag("Francesco", "Rinaldi"))
7  # Should display "Francesco R."
8  print(nametag("Jean-Luc", "Grand-Pierre"))
9  # Should display "Jean-Luc G."

```

Jane S.
Francesco R.
Jean-Luc G.

Correct

Great work! You remembered the formatting expression to limit how many characters in a string are displayed.

Question 5

The `replace_ending` function replaces the old string in a sentence with the new string, but only if the sentence ends with the old string. If there is more than one occurrence of the old string in the sentence, only the one at the end is replaced, not all of them.

For example, `replace_ending("abcabc", "abc", "xyz")` should return `abcxyz`, not `xyzxyz` or `xyzabc`. The string comparison is case-sensitive, so `replace_ending("abcabc", "ABC", "xyz")` should return `abcabc` (no changes made).

1 / 1 point

```
1  def replace_ending(sentence, old, new):
2      # Check if the old string is at the end of the sentence
3      if sentence.endswith(old):
4          # Using i as the slicing index, combine the part
5          # of the sentence up to the matched string at the
6          # end with the new string
7          i = sentence.rindex(old)
8          new_sentence = sentence[:-i] + new
9          return new_sentence
10
11     # Return the original sentence if there is no match
12     return sentence
13 print(replace_ending("It's raining cats and cats", "cats", "dogs"))
14 # Should display "It's raining cats and dogs"
15 print(replace_ending("She sells seashells by the seashore", "seashells",
16     "donuts"))
17 # Should display "She sells seashells by the seashore"
18 print(replace_ending("The weather is nice in May", "may", "april"))
19 # Should display "The weather is nice in May"
20 print(replace_ending("The weather is nice in May", "May", "April"))
21 # Should display "The weather is nice in April"
```

It's raining cats and dogs
She sells seashells by the seashore
The weather is nice in May
The weather is nice in April

Correct

Outstanding! Look at all of the things that you can do with these string commands!