← Back **Practice Quiz: Strings**
Practice Quiz • 10 min • 5 total points

English

Congratulations! You passed!

Grade received 100% **To pass** 80% or higher

Go to next item

1. Fill in the blanks to complete the is_palindrome function. This function checks if a given string is a palindrome. A palindrome is a string that contains the same letters in the same order, whether the word is read from left to right or right to left. Examples of palindromes are words like kayak and radar, and phrases like "Never Odd or Even". The function should ignore blank spaces and capitalization when checking if the given string is a palindrome. Complete this function to return True if the passed string is a palindrome, False if not.

1 / 1 point

```python
def is_palindrome(input_string):
    # Two variables are initialized as string date types using empty
    # quotes: "reverse_string" to hold the "input_string" in reverse
    # order and "new_string" to hold the "input_string" minus the
    # spaces between words, if any are found.
    new_string = ""
    reverse_string = ""

    # Complete the for loop to iterate through each letter of the
    # "input_string"
    for letter in input_string:

        # The if-statement checks if the "letter" is not a space.
        if letter != " ":

            # If True, add the "letter" to the end of "new_string" and
            # to the front of "reverse_string". If False (if a space
            # is detected), no action is needed. Exit the if-block.
            new_string = new_string + letter
            reverse_string = letter + reverse_string

    # Complete the if-statement to compare the "new_string" to the
    # "reverse_string". Remember that Python is case-sensitive when
    # creating the string comparison code.
    if new_string.lower() == reverse_string.lower():

        # If True, the "input_string" contains a palindrome.
        return True

    # Otherwise, return False.
    return False


print(is_palindrome("Never Odd or Even")) # Should be True
print(is_palindrome("abc")) # Should be False
print(is_palindrome("kayak")) # Should be True
```

Run
Reset

Correct

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

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

```python
def convert_distance(miles):
    km = miles * 1.6
    result = "{} miles equals {:.1f} km".format(miles, km)
    return result


print(convert_distance(12)) # Should be: 12 miles equals 19.2 km
print(convert_distance(5.5)) # Should be: 5.5 miles equals 8.8 km
print(convert_distance(11)) # Should be: 11 miles equals 17.6 km
```

Run
Reset

Correct

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

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

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

1 / 1 point

[✓ print(Weather[:4])]

print(Weather[4:])

print(Weather[1:4])

print(Weather[:"f"])

Correct

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

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

```python
def nametag(first_name, last_name):
    return("{} {}.".format(first_name, last_name[0].upper()))


print(nametag("Jane", "Smith"))
# Should display "Jane S."
print(nametag("Francesco", "Rinaldi"))
# Should display "Francesco R."
print(nametag("Jean-Luc", "Grand-Pierre"))
# Should display "Jean-Luc G."

```

Run

Reset

Correct

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

5. The replace_ending function replaces a specified substring at the end of a given sentence with a new substring. If the specified substring does not appear at the end of the given sentence, no action is performed and the original sentence is returned. If there is more than one occurrence of the specified substring in the sentence, only the substring at the end of the sentence is replaced. 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

```python
def replace_ending(sentence, old, new):
    # Check if the old substring is at the end of the sentence
    if sentence.endswith(old):
        # Using i as the slicing index, combine the part
        # of the sentence up to the matched string at the
        # end with the new string
        i = sentence.rindex(old)
        new_sentence = sentence[:i] + new
        return new_sentence


    # Return the original sentence if there is no match
    return sentence

print(replace_ending("It's raining cats and cats", "cats", "dogs"))
# Should display "It's raining cats and dogs"
print(replace_ending("She sells seashells by the seashore", "seashells", "donuts"))
# Should display "She sells seashells by the seashore"
print(replace_ending("The weather is nice in May", "may", "april"))
# Should display "The weather is nice in May"
print(replace_ending("The weather is nice in May", "May", "April"))
# Should display "The weather is nice in April"

```

Run

Reset

```
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!