Week 4 Quiz

**Practice Quiz: Strings**

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.



**def is\_palindrome(input\_string):**

**# We'll create two strings, to compare them**

**new\_string = ""**

**reverse\_string = ""**

**# Traverse through each letter of the input string**

**for letter in input\_string:**

**# Add any non-blank letters to the**

**# end of one string, and to the front**

**# of the other string.**

**if letter != " ":**

**new\_string += letter.lower()**

**reverse\_string = letter.lower() + reverse\_string**

**# Compare the strings**

**if new\_string == reverse\_string:**

**return True**

**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**

**Correct**

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

**1 / 1 point**

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".



**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**

**Correct**

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

**hooray!**

**1 / 1 point**

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"?



**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.**

**1 / 1 point**

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."



**def nametag(first\_name, last\_name):**

**return("{} {}.".format(first\_name, last\_name[0]))**

**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."**

**Correct**

**Great work! You remembered the formatting expression to**

**limit how many characters in a string are displayed.**

**1 / 1 point**

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).



**def replace\_ending(sentence, old, new):**

**# Check if the old string is at the end of the sentence**

**if old[:] == sentence[-len(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 = len(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"**

**Correct**

**Outstanding! Look at all of the things that you can do with**

**these string commands!**

**1 / 1 point**

**Practice Quiz: Lists**

Question 1

Given a list of filenames, we want to rename all the files with extension hpp to the extension h. To do this, we would like to generate a new list called newfilenames, consisting of the new filenames. Fill in the blanks in the code using any of the methods you’ve learned thus far, like a for loop or a list comprehension.



**filenames = ["program.c", "stdio.hpp", "sample.hpp", "a.out", "math.hpp", "hpp**

**.out"]**

**newfilenames = []**

**for file in filenames:**

**if '.hpp' in file:**

**newfilenames.append(file[:-2])**

**else:**

**newfilenames.append(file)**

**print (newfilenames)**

**Correct**

**Great work! You're starting to see the benefits of knowing**

**how to operate with lists and strings.**

**1 / 1 point**

Question 2

The permissions of a file in a Linux system are split into three sets of three permissions: read, write, and execute for the owner, group, and others. Each of the three values can be expressed as an octal number summing each permission, with 4 corresponding to read, 2 to write, and 1 to execute. Or it can be written with a string using the letters r, w, and x or - when the permission is not granted. For example: 640 is read/write for the owner, read for the group, and no permissions for the others; converted to a string, it would be: "rw-r-----" 755 is read/write/execute for the owner, and read/execute for group and others; converted to a string, it would be: "rwxr-xr-x" Fill in the blanks to make the code convert a permission in octal format into a string format.



**def octal\_to\_string(octal):**

**result = ""**

**value\_letters = [(4,"r"),(2,"w"),(1,"x")]**

**# Iterate over each of the digits in octal**

**for digit in [int(n) for n in str(octal)]:**

**# Check for each of the permissions values**

**for value, letter in value\_letters:**

**if digit >= value:**

**result += letter**

**digit -= value**

**else:**

**result += '-'**

**return result**

**print(octal\_to\_string(755)) # Should be rwxr-xr-x**

**print(octal\_to\_string(644)) # Should be rw-r--r--**

**print(octal\_to\_string(750)) # Should be rwxr-x---**

**print(octal\_to\_string(600)) # Should be rw-------**

**Correct**

**You nailed it! This is how we work with lists of tuples, how**

**exciting is that!**

**1 / 1 point**

Question 3

Let's create a function that turns text into pig latin: a simple text transformation that modifies each word moving the first character to the end and appending "ay" to the end. For example, python ends up as ythonpay.



**def pig\_latin(text):**

**say = ""**

**# Separate the text into words**

**words = text.split()**

**for word in words:**

**# Create the pig latin word and add it to the list**

**say += word[1:]+word[0]+'ay'**

**if word != words[len(words)-1]:**

**say +=' '**

**# Turn the list back into a phrase**

**return say**

**print(pig\_latin("hello how are you")) # Should be "ellohay owhay reaay ouyay"**

**print(pig\_latin("programming in python is fun")) # Should be "rogrammingpay niay**

**ythonpay siay unfay"**

**Correct**

**Nice! You're using some of the best string and list**

**functions to make this work. Great job!**

**1 / 1 point**

Question 4

Tuples and lists are very similar types of sequences. What is the main thing that makes a tuple different from a list?



A tuple is mutable



A tuple contains only numeric characters



**A tuple is immutable**



A tuple can contain only one type of data at a time

**Correct**

**Awesome! Unlike lists, tuples are immutable, meaning they can't be changed.**

**1 / 1 point**

**Practice Quiz: Dictionaries**

Question 1

The email\_list function receives a dictionary, which contains domain names as keys, and a list of users as values. Fill in the blanks to generate a list that contains complete email addresses (e.g. diana.prince@gmail.com).



**def email\_list(domains):**

**emails = []**

**for names, users in domains.items():**

**for user in users:**

**emails.append(user+"@"+names)**

**return(emails)**

**print(email\_list({"gmail.com": ["clark.kent", "diana.prince", "peter.parker"],**

**"yahoo.com": ["barbara.gordon", "jean.grey"], "hotmail.com": ["bruce.wayne"]}**

**))**

**Correct**

**Well done! You've created quite an email list!**

**1 / 1 point**

Question 2

The groups\_per\_user function receives a dictionary, which contains group names with the list of users. Users can belong to multiple groups. Fill in the blanks to return a dictionary with the users as keys and a list of their groups as values.



**def groups\_per\_user(group\_dictionary):**

**user\_groups = {}**

**for group, users in group\_dictionary.items():**

**for user in users:**

**if user not in user\_groups:**

**user\_groups[user] = []**

**user\_groups[user].append(group)**

**return user\_groups**

**print(groups\_per\_user({"local": ["admin", "userA"],**

**"public": ["admin", "userB"],**

**"administrator": ["admin"] }))**

**Correct**

**Well done, you! You're now creating dictionaries out of**

**other dictionaries!**

**1 / 1 point**

Question 3

The dict.update method updates one dictionary with the items coming from the other dictionary, so that existing entries are replaced and new entries are added. What is the content of the dictionary “wardrobe“ at the end of the following code?



**wardrobe = {'shirt': ['red', 'blue', 'white'], 'jeans': ['blue', 'black']}**

**new\_items = {'jeans': ['white'], 'scarf': ['yellow'], 'socks': ['black',**

**'brown']}**

**wardrobe.update(new\_items)**



{'jeans': ['white'], 'scarf': ['yellow'], 'socks': ['black', 'brown']}



**{'shirt': ['red', 'blue', 'white'], 'jeans': ['white'], 'scarf': ['yellow'], 'socks': ['black', 'brown']}**



{'shirt': ['red', 'blue', 'white'], 'jeans': ['blue', 'black', 'white'], 'scarf': ['yellow'], 'socks': ['black', 'brown']}



{'shirt': ['red', 'blue', 'white'], 'jeans': ['blue', 'black'], 'jeans': ['white'], 'scarf': ['yellow'], 'socks': ['black', 'brown']}

**Correct**

**Correct! The dict.update method updates the dictionary (wardrobe) with the items coming from the other dictionary (new\_items), adding new entries and replacing existing entries.**

**1 / 1 point**

Question 4

 What’s a major advantage of using dictionaries over lists?



 Dictionaries are ordered sets



 Dictionaries can be accessed by the index number of the element



Elements can be removed and inserted into dictionaries



**It’s quicker and easier to find a specific element in a dictionary**

**Correct**

**Right on! Because of their unordered nature and use of key value pairs, searching a dictionary takes the same amount of time no matter how many elements it contains**

**1 / 1 point**

Question 5

The add\_prices function returns the total price of all of the groceries in the dictionary. Fill in the blanks to complete this function.



1

**def add\_prices(basket):**

**# Initialize the variable that will be used for the calculation**

**total = 0**

**# Iterate through the dictionary items**

**for value in basket.values():**

**# Add each price to the total calculation**

**# Hint: how do you access the values of**

**# dictionary items?**

**total += value**

**# Limit the return value to 2 decimal places**

**return round(total, 2)**

**groceries = {"bananas": 1.56, "apples": 2.50, "oranges": 0.99, "bread": 4.59,**

**"coffee": 6.99, "milk": 3.39, "eggs": 2.98, "cheese": 5.44}**

**print(add\_prices(groceries)) # Should print 28.44**

**Correct**

**Nicely done! Dictionaries are a helpful way to store**

**information, and access it easily when it's needed.**