**Study Guide: Week 4 Graded Quiz**

It is time to prepare for the Week 4 Graded Quiz. Please review the following items from this module before beginning the quiz. If you would like to refresh your memory on these materials, please also revisit the Study Guides located before each practice quiz in Week 4: [Study Guide: Strings](https://www.coursera.org/learn/python-crash-course/supplement/ydyIo/study-guide-strings), [Study Guide: Lists Operations and Methods](https://www.coursera.org/learn/python-crash-course/supplement/sbRdF/study-guide-lists-operations-and-methods), and [Study Guide: Dictionary Methods](https://www.coursera.org/learn/python-crash-course/supplement/Cc19J/study-guide-dictionary-methods).

**Knowledge**

* How to output a list of the keys in a Python dictionary.
* How to determine the output of a string index range used on a string.
* Determine what a list should contain after the .insert() method is used on the list.
* How to replace a specific word in a sentence with the same word in all uppercase letters.
* How to use a dictionary to count the frequency of letters in a string.

**Operations, Methods, and Functions**

* **String Methods, Operations, and Functions**
  + .upper()
  + .lower()
  + .split()
  + .format()
  + .isnumeric()
  + .isalpha()
  + .replace()
  + string index [ ]
  + len()
* **List Operations and Methods**
  + .reverse()
  + .extend()
  + .insert()
  + .append()
  + .remove()
  + .sort()
  + list comprehensions [ ]
  + list comprehensions [ ] with if condition
* **Dictionary Operations and Methods**
  + .items()
  + .update()
  + .keys()
  + .values()
  + .copy()
  + dictionary[key]
  + dictionary[key] = value

**Coding Skills**

**Skill 1: Using string methods**

* Separate numerical values from text values in a string using **.split()**.
* Iterate over the elements in a string.
* Test if the element contains letters with **.isalpha()**.
* Assign the elements of the split string to new variables.
* Trim any extra white space using **.strip()**.
* Format a string using **.format()** and **{ }** variable placeholders.

def sales\_prices(item\_and\_price):

    # Initialize variables "item" and "price" as strings

    item = ""

    price = ""

    # Create a variable "item\_or\_price" to hold the result of the split.

    item\_or\_price = item\_and\_price.split()

    # For each element "x" in the split variable "item\_or\_price"

    for x in item\_or\_price:

        # Check if the element is a number

        if x.isalpha():

            # If true, assign the element to the "item" string variable and add a space

            # for any item names containing multiple words, like "Winter fleece jacket".

            item += x + " "

        # Else, if x is a number (if x.isalpha() is false):

        else:

            # Assign the element to the "price" string variable.

            price = x

    # Strip the extra space to the right of the last "item" word

    item = item.strip()

    # Return the item name and price formatted in a sentence

    return "{} are on sale for ${}".format(item,price)

# Call to the function

print(sales\_prices("Winter fleece jackets 49.99"))

# Should print "Winter fleece jackets are on sale for $49.99"

* Use the len() function to measure a string.

# This function accepts a string variable "data\_field".

def count\_words(data\_field):

    # Splits the string into individual words.

    split\_data = data\_field.split()

    # Then returns the number of words in the string using the len()

    # function.

    return len(split\_data)

    # Note that it is possible to combine the len() function and the

    # .split() method into the same line of code by inserting the

    # data\_field.split() command into the the len() function parameters.

# Call to the function

count\_words("Catalog item 3523: Organic raw pumpkin seeds in shell")

# Should print 9

**Skill 2: Using list methods**

* Reverse the order of a list using the **.reverse()** method.
* Combine two lists using the **.extend()** method.

# This function accepts two variables, each containing a list of years.

# A current "recent\_first" list contains [2022, 2018, 2011, 2006].

# An older "recent\_last" list contains [1989, 1992, 1997, 2001].

# The lists need to be combined with the years in chronological order.

def record\_profit\_years(recent\_first, recent\_last):

    # Reverse the order of the "recent\_first" list so that it is in

    # chronological order.

    recent\_first.reverse()

    # Extend the "recent\_last" list by appending the newly reversed

    # "recent\_first" list.

    recent\_last.extend(recent\_first)

    # Return the "recent\_last", which now contains the two lists

    # combined in chronological order.

    return recent\_last

# Assign the two lists to the two variables to be passed to the

# record\_profit\_years() function.

recent\_first = [2022, 2018, 2011, 2006]

recent\_last = [1989, 1992, 1997, 2001]

# Call the record\_profit\_years() function and pass the two lists as

# parameters.

print(record\_profit\_years(recent\_first, recent\_last))

# Should print [1989, 1992, 1997, 2001, 2006, 2011, 2018, 2022]

**Skill 3: Using a list comprehension**

* Use a list comprehension [ ] as a shortcut for creating a new list from a range.
* Include a calculation with a **for** loop **in** a **range** with 2 parameters (lower, upper+1).

# The function accepts two parameters: a start year and an end year.

def list\_years(start, end):

    # It returns a list comprehension that creates a list of years in a for

    # loop using a range from the start year to the end year (inclusive of

    # the upper range year, using end+1).

    return [year for year in range(start, end+1)]

# Call the years() function with two parameters.

print(list\_years(1972, 1975))

# Should print [1972, 1973, 1974, 1975]

* Use a list comprehension [ ] with a **for** loop and an **if** condition.

# The function accepts two variable integers through the parameters and

# returns all odd numbers between x and y-1.

def odd\_numbers(x, y):

# This list comprehension uses a for loop to iterate through values

# of n in a range from x to y, with the value of y excluded (meaning

# keep the default range() function behavior to exclude the

# end-of-range value from the range). Since an incremental value is not

# specified, the range function uses the default increment of +1.

# The if condition checks n to test if the number is odd using the

# modulo operator. This condition is written to check if n is divided

# by 2, that the remainder is not 0.

    return [n for n in range(x, y) if n % 2 != 0]

# Call the years() function with two parameters.

print(odd\_numbers(5, 15))

# Should print [5, 7, 9, 11, 13]

**Skill 4: Using dictionary methods**

* Iterate through the keys and values of a dictionary.
* Return the keys and values in a formatted string using the .format() function.

# The network() function accepts a dictionary "servers" as a parameter.

def network(servers):

    # A string variable is initialized to hold the "result".

    result = ""

    # For each "hostname" (key) and "IP address" (value) in the "server" dictionary items...

    for hostname, IP\_address in servers.items():

        # A string identifying the hostname and IP address for each server is added

        # to the "result" variable. The string .format() function and is used to plug

        # the hostname and IP\_address variables into the designated {} placeholders

        # within the string.

        result += "The IP address of the {} server is {}".format(hostname, IP\_address) + "\n"

    # Return the "result" variable string.

    return result

# Call the "network" function with the dictionary.

print(network({"Domain Name Server":"8.8.8.8", "Gateway Server":"192.168.1.1", "Print Server":"192.168.1.33", "Mail Server":"192.168.1.190"}))

# Should print:

# The IP address of the Domain Name Server server is 8.8.8.8

# The IP address of the Gateway Server server is 192.168.1.1

# The IP address of the Print Server server is 192.168.1.33

# The IP address of the Mail Server server is 192.168.1.190

* Create a copy of a dictionary.
* Iterate through the values of the new dictionary.
* Change each value in the new dictionary, while keeping the same keys.

# The scores() function accepts a dictionary "game\_scores" as a parameter.

def reset\_scores(game\_scores):

    # The .copy() dictionary method is used to create a new copy of the "game\_scores".

    new\_game\_scores = game\_scores.copy()

    # The for loop iterates over new\_game\_scores items, with the player as the key

    # and the score as the value.

    for player, score in new\_game\_scores.items():

        # The dictionary operation to assign a new value to a key is used

        # to reset the grade values to 0.

        new\_game\_scores[player] = 0

    return new\_game\_scores

# The dictionary is defined.

game1\_scores = {"Arshi": 3, "Catalina": 7, "Diego": 6}

# Call the "reset\_scores" function with the "game1\_scores" dictionary.

print(reset\_scores(game1\_scores))

# Should print {'Arshi': 0, 'Catalina': 0, 'Diego': 0}

**Reminder: Correct syntax is critical**

Using precise syntax is critical when writing code in any programming language, including Python. Even a small typo can cause a syntax error and the automated Python-coded quiz grader will mark your code as incorrect. This reflects real life coding errors in the sense that a single error in spelling, case, punctuation, etc. can cause your code to fail. Coding problems caused by imprecise syntax will always be an issue whether you are learning a programming language or you are using programming skills on the job. So, it is critical to start the habit of being precise in your code now.

No credit will be given if there are any coding errors on the automated graded quizzes - including minor errors. Fortunately, you have 3 optional retake opportunities on the graded quizzes in this course. Additionally, you have unlimited retakes on practice quizzes and can review the videos and readings as many times as you need to master the concepts in this course.

Now, before starting the graded quiz, please review this list of common syntax errors coders make when writing code.

**Common syntax errors:**

* Misspellings
* Incorrect indentations
* Missing or incorrect key characters:
  + Parenthetical types - ( curved ), [ square ], { curly }
  + Quote types - "straight-double" or 'straight-single', “curly-double” or ‘curly-single’
  + Block introduction characters, like colons - :
* Data type mismatches
* Missing, incorrectly used, or misplaced Python reserved words
* Using the wrong case (uppercase/lowercase) - Python is a case-sensitive language

**Resources**

For additional Python practice, the following links will take you to several popular online interpreters and codepads:

* [Welcome to Python](https://www.python.org/shell/)
* [Online Python Interpreter](https://www.onlinegdb.com/online_python_interpreter)
* [Create a new Repl](https://repl.it/languages/python3)
* [Online Python-3 Compiler (Interpreter)](https://www.tutorialspoint.com/execute_python3_online.php)
* [Compile Python 3 Online](https://rextester.com/l/python3_online_compiler)
* [Your Python Trinket](https://trinket.io/python3)