Spencer Olson **Date Completed**

Lab for Chapter 3 – from Kaefer pp. 52-58: Dictionaries

3.6.Write Python code to define a dictionary named taxi_trip_info with the following key value pairs:

Key	Value
Trip_id	“da7a62fce”
Trip_seconds	360
Trip_miles	1.1
Fare	“$6.25”

Print out the dictionary to show what it looks like. Also print out just the Trip_miles value from the dictionary.
{'trip_id': 'da7a62fce', 'Trip_seconds': 360, 'Trip_miles': 1.1, 'Fare': '$6.25'}

a. Code here:

```
trip_data = {
    "trip_id": "da7a62fce",
    "Trip_seconds": 360,
    "Trip_miles": 1.1,
    "Fare": "$6.25"
}
print(trip_data)
```

```
    print(f"Trip miles are {trip_data['Trip_miles']}")
```

b. Compare and contrast Dictionaries with Lists

Dictionaries are unordered information where lists are ordered. Dictionaries can be used like lists if you use numbers. But the expectation is that a key will always be a string without caring for the order or position among other keys. When you need ordered tasks completed a list is the choice to make. But dictionaries in a sense are labels for information.

3.7. Write Python code to define a **list** of taxi trip durations in miles (use values 1.1, 0.8, 2.5, 2.6). Also define a **tuple** of fares for the same number of trips (use values "$6.25," "$5.25," "$10.50," "$8.05"). Store both the tuple and the list as values in a dictionary called trips, with keys "miles" and "fares." Print out the dictionary to show what it looks like.

a. Code here:

```
trip_durations = [1.1, 0.8, 2.5, 2.6]
trip_fares = ('$6.25', '$5.25', '$10.50', '$8.05')

trips ={
    'miles':trip_durations,
    'fares': trip_fares
}
print(trips)
```

b. Explain how you might use this to keep a record of trips. Print out the duration and cost of the 3nd trip. Is there a purpose for setting the fares as a tuple rather than a list? Why not both tuples or both lists?

You can use as a record to ensure that the fares are lined up with the appropriate miles. Being able to reference them in the future for comparison later. Setting up as both types utilizes them to be referenced as the values we intended to be, either strings and/numeric's. The hindrance of using strings for dollar amount is that we cannot add the strings later on. To the logical thing is to not use strings for numerical values. If we want to have a mutable data set we can use lists for both. Tuples would work for archival data sets that will not be changed. But they should be using the same type unless there is a specific use case that calls for it.

c. Lookup the `zip()` function and use it to create a dictionary where the keys are the durations and the values are the fares. You will need to use `dict()` to create the dictionary from the return value of `zip()`. Print out the duration and cost of the 3rd trip. Explain if this version is more "elegant" than what you did in (a).

Its more elegant because its showing what the end user wants to know in a digestible format for general purpose rather than brackets commas and colins etc. Its explicit in its use case. And it explains itself when ran.

d. Use the output of (c) to create a **list** of dictionaries where each dictionary represents a trip. Use appropriate names for the keys. Do this in VSCode by using "regular expression" find

and replace. You can use `(\d+\.\d+):\s*'(\$\d+\.\d{2})'` in the find and `{'key': $1, 'value': '$2'},` in the replace (change `key` and `value` to appropriate names). Print out the duration and cost of the 3rd trip. Explain if this version is more "elegant" than what you did in (a) and (b). Discuss the use of regular expressions over manually making the changes. Can you think of a way to write code to make the conversion programmatically (i.e. using the output directly rather than VSCode find and replace)?

The elegance compared to a and b is that each trip is now self contained as a dictionary inside a list, and instead of separate lists or value pairs, everything related to a specific trip is together. Its easier to expand and work with, and you can now loop through trips_list easily, and extend the structure. A and B also used strings for the fares which is not ideal for manipulating, tracking and using for later analyzation if needed. So I changed it here in this section.

e. Modify your code to use numbers rather than Strings for the fares (suggestion: use the regex `'\$(.*)'` to find and replace with `$1` rather than manually changing). Note you will have to change how you print the output. Is this better? Why or why not?

Yes because of the similar mutable reasons for the dictionaries, lists etc.

3.8 Go back to exercise 3.3 where you defined a list of survey response values (5, 7, 3, and 8) and a tuple of respondent ID values (1012, 1035, 1021, and 1053). Rather than appending the tuple to the list, create a Dictionary with ID values as the keys and survey responses as the values using `zip()`.

a. Code here:

```
survey_responses = [5 , 7 , 3 , 8]
respondent_ids = (1012, 1035, 1021, 1053)

survey_dict = dict(zip(respondent_ids, survey_responses))

print(survey_dict)
```

b. Discuss why this is a more useable data structure than what was done in 3.3. What might be even more useable?

Its more usable as it connects the ID with the responses rather than having two separate lists. Other words they explicitly match. We area also able to add more responses into the dictionary much easier.

c. Why is a List with indices of respondent ID values and values survey responses e.g. [none, none, <1010 more none>, 5, none, none, ...,] not a useable data structure?

It creates more work to order and retrieve. Impractical for pretty much all scenarios.