TMTplus Introduction to Scientific Programming

Mahdi Farnaghi, Mahdi Khodadadzadeh & Robert Ohuru

March 2021

Answers 6

Dictionaries & their expressions

Ex 6.8

```
agenda = {"Mary": "12345678", "Emily": "87654321",
                  "Steve": "56789121", "Patrick": "78901234"}
        # add four friends to the dictionary. Name as key and the
        # phone number as value
print(agenda)  # print the content of the entire dictionary

        #or we can use a diferent approach

agenda = {}
agenda["Mary"]="12345678"
agenda["Emily"]="87654321"
agenda["Steve"]="56789121"
agenda["Patrick"]="78901234"
print(agenda)
```

In the case of a phone number, one could use the integer data type to represent such. However, this is not preferable because of a number of reasons:

- integers allow operations such as addition, subtraction and multiplication; these are meaningless for phone numbers
- many phone numbers starts with a 0 digit; this is not really easy to achieve or even possible with integers
- phone numbers may require non-digit characters for correct representation, characters such as "+" and "-". In the USA, toll-free phone numbers include also the "1–800" numbers that are memorized sometimes by words, as in "1–800-CAR–RENT."

In short, phone numbers are better represented by strings, possibly governed by some format restrictions.

Ex 6.9

https://docs.python.org/3/tutorial/datastructures.html#dictionaries
The keys must be immutable. Types:

1. Number
2. String
3. Tuple

Ex 6.10

```
# with lists
agenda["Mary"] = ["12345678", "mary.brown@gmail.com"]
agenda["Emily"] = ["87654321", "emily.andersen@hotmail.com"]
agenda["Steve"] = ["56789121", "steve.jackson@gmail.com"]
agenda["Patrick"] = ["78901234", "patrick.smith@gmail.com"]
        # print the agenda
print(agenda)

# with tuples
agenda["Mary"] = ("12345678", "mary.brown@gmail.com")
agenda["Emily"] = ("87654321", "emily.andersen@hotmail.com")
agenda["Steve"] = ("56789121", "steve.jackson@gmail.com")
agenda["Patrick"] = ("78901234", "patrick.smith@gmail.com")
        # print the agenda
print(agenda)
```

A list is mutable, therefore it is a better choice if we need to change a single element (phone number or email address) later. A tuple, because it is not mutable, is a better choice if we don't want to allow changes of single elements (the phone number or the email address). In case of tuples, if we need to change one element (phone number or email) we need to re-assign the complete tuple.

Ex 6.11

```
for name in agenda:
    print(name,"->","(",agenda[name][0],agenda[name][1],")")
```

Ex 6.12

```
        # If we used a list
agenda["Mary"][0] = "45678901"
        # If we used a tuple
agenda["Mary"] = ("45678901", "mary.brown@gmail.com")  # change the
        # phone number for one of the friends, because if we use
        # tuples, which are immutable, the entire value has to be
        # changed.  If you use the list then only the single value
        # needs a change.
print(agenda.items()) # print items of the updated agenda

agenda["Emily"] = ("87654321", "emily.andersen@hotmail.com",
                               "89985678")
        # add second telephone number after e-mail
print(agenda.items()) #print items of the updated agenda
```

Ex 6.13

```
dense_matrix=[[0,0,0,1,0,0,0,4,0,4],[0,0,0,0,0,0,0,4,4,0],
              [0,2,0,0,0,0,4,0,3,0],[0,0,0,0,0,0,0,0,0,0],
              [0,0,0,3,0,5,8,2,0,5]]

sparse_matrix = {}

for i in range(5): # we use range because we must loop 5 times
   for j in range(10): #we use range because we must loop 10 times
      if dense_matrix[i][j] != 0:
         key = (i, j)
         value = dense_matrix[i][j]
         sparse_matrix[key] = value

print(sparse_matrix)
```

Ex 6.14

1. Given the purposes described, we could use the length of a found sentence as the key of the dictionary, and the number of sentences of such length as value under such a key.

2. Perhaps something like `sentence_length_counts`?

3. At start we will not have found any sentences, so an empty dictionary `{}` is just appropriate.

4. Suppose `sl` is that new length, then clearly we will find by testing whether it is in `keys` of the dictionary. If indeed it is not, we should register that we now have one such sentence; `sentence_length_counts[sl] = 1`

5. If on the contrary, the length `sl` was previously encountered, one just needs to add one to its entry: `sentence_length_counts[sl] += 1` That is almost identical syntax.

6. Left as an exercise here. Most of the ingredients to the script are sketched above, including the important conditional. What we did not discuss above is how a text file can be split into sentences. A simple (naive, incomplete) way is to read a whole file as a single string and subsequently split that string at all occurrences of `"."` with the string method `.split(".")`. This will give a list of sentences.

   The approach is naive because not all period characters are end-of-sentence stops, and there are end-of-sentence stops that are not period characters, but question marks and so forth. More advanced solutions might work with the regular expressions package `re` or with the natural language toolkit package `nltp`.

   A good sketch of the full complexity of this problem is provided here : stack exchange page. We are not suggesting you follow this thread to the implementation.