COMP10001 Foundations of Computing
Semester 1, 2019

Tutorial Questions: Week 6

— VERSION: 1474, DATE: APRIL 2, 2019 —

Discussion

1. Recall that while loops and for loops are the two types of loops we have learned. How are they similar and how are they different? Where would you use them in different situations?

   A: *Both loops allow us to run some code multiple times. A for loop iterates over a sequence, but a while loop allows you the freedom as a programmer to loop as long as a condition will be true.*
   We often use a for loop when we know in advance how many times we want to iterate, whether that's through a sequence or a range of numbers using `range()`. *While loops are slightly more versatile in that they can be used to repeat actions even if you don't know how many times you would like to iterate ie. loop until the user enters a particular input. While loops can also iterate over sequences by incrementing an index, but this requires the creation and update of a counter which is not necessary in a for loop.*

2. Is it always possible to convert a while loop into a for loop and vice versa? How do we do it?

   A: *It is always possible to convert a for loop into a while loop, with the creation of an index to access each value in whatever sequence the for loop is iterating over. Most while loops can be converted into for loops, by modelling values of a loop variable that the while loop may iterate through. In some cases it is not possible, such as where a while loop repeats indefinitely until an action is made by the user.*
   To convert between loops, one method is to identify three things: 1. what the loop variable is initialised to; 2. how the loop variable is incremented during each iteration; and 3. when the loop terminates. You can then put this into a while loop (initialisation, increment and condition for termination) or a for loop (creating a sequence that starts at the initialisation, increments by one for each item and ends at the desired point of termination). The rest of the loop body will remain exactly the same in most cases.

3. Consider the following while loop and two conversions to for loops. Are the two for loops equivalent? Why might you choose one over the other?

```
count = 0
items = ('eggs', 'spam', 'more␣eggs')
while count < len(items):
    print(f"we␣need␣to␣buy␣more␣{items[count]}")
    count += 1
```

```
items = ('eggs', 'spam', 'more␣eggs')
for count in range(len(items)):
    print(f"we␣need␣to␣buy␣more
{items[count]}")
```

```
items = ('eggs', 'spam', 'more␣eggs')
for item in items:
    print(f"we␣need␣to␣buy␣more
{item}")
```

   A: *Both are functionally equivalent and will do the same thing. The first uses range() to get indices which index* `items`*, making it closer to original loop. The second is cleaner since it iterates through list directly.*

   Now try Exercises 1 & 2

4. In what situations would we use a "dictionary". How is it structured, how do we add and delete items?

   A: *A dictionary holds relations between "keys" and "values". It's useful for counting frequencies or storing information about things. Dictionaries are accessed in a similar way to other sequences, by using index notation. Values stored are retrieved by indexing with the associated key (*`d[key]`*). Values are added by indexing with assignment (*`d[key] = value`*) and deleted with the* `.pop(key)` *method, which takes as an argument the key we wish to delete.*

5. What is the difference between using the `.pop()` method on a dictionary and using it on a list?

   A: *On a list:* `.pop()` *called without an index argument removes the last item in the list. Called with an index* `.pop(index)` *deletes the item at that index in the list. Both times it will return the object it has deleted from the list.*
   On a dictionary: `.pop(key)` *deletes the key:value pair associated with that key in the dictionary, returning the value it has removed. Without an argument,* `.pop()` *will not work because unlike lists, dictionaries don't have an inherent ordering so need the key to know which value to delete.*

6. In what situations would we use a "set"? How does it differ to a list and a dictionary?

   A: *A set stores a collection of unique objects. We might use one when storing a set of numbers or other unique objects, or when we want to remove duplicates from some other sequence. The set operations can be very useful too.*

7. What key operations can you perform on sets? How do you add and remove items from them?

   A: *The three main operations are union:* `s1 | s2` *or* `s1.union(s2)`; *intersection:* `s1 & s2` *or* `s1.intersection(s2)`; *and difference:* `s1 - s2` *or* `s1.difference(s2)`. *Adding an item is possible with the* `.add(item)` *method and removing an item is done with the* `.remove(item)` *method. Note that since dictionaries use empty braces* { }, *in order to create an empty set we need to use* `set()`.

   Now try Exercises 3 & 4

Exercises

1. Rewrite the following code using a for loop.

```
i = 2
while i < 6:
    print(f"The␣square␣of␣{i}␣is␣{i*i}")
    i = i + 1
```

A:

```
for i in range(2, 6):
    print(f"The␣square␣of␣{i}␣is␣{i*i}")
```

2. Rewrite the following for loops using while loops.

(a)
```
colours = ["yellow", "green", "purple"]
for colour in colours[1:]:
    print(colour, "is␣my␣favourite␣colour")
```

A:

```
colours = ["yellow", "green", "purple"]
i = 1
while i < len(colours):
    print(colours[i], "is␣my␣favourite␣colour")
    i += 1
```

(b)
```
MIN_WORD_LEN = 6
long_words = 0
for word in text.split():
    if len(word) >= MIN_WORD_LEN:
        long_words += 1
```

A:

```
MIN_WORD_LEN = 6
long_words = 0
words = text.split()
i = 0
while i < len(words):
    if len(words[i]) >= MIN_WORD_LEN:
        long_words += 1
    i += 1
```

Alternative solution:

```
MIN_WORD_LEN = 6
long_words = 0
words = text.split()
while words:
    word = words.pop()
    if len(word) >= MIN_WORD_LEN:
        long_words += 1
```

3. Evaluate the following given the assignment `d = {"R": 0, "G": 255, "B": 0, "other": {"opacity": 0.6}}`. Specify whether the value of `d` changes as a result. Assume `d` is reset to its original value after each.

   (a) `"R"in d`

   A: `True`

   (b) `d["R"]`

   A: `0`

   (c) `d["R"] = 255`

   A: *New value of* `d`*:*
   `{'R': 255, 'G': 255, 'B': 0, 'other': {'opacity': 0.6}}`

   (d) `d["A"]`

   A: `KeyError: 'A'`

   (e) `d["A"] = 50`

   A: *New value of* `d`*:*
   `{'R': 0, 'G': 255, 'B': 0, 'other': {'opacity': 0.6}, 'A': 50}`

   (f) `d.pop("G")`

   A: `255`
   New value of `d`*:*
   `{'R': 0, 'B': 0, 'other': {'opacity': 0.6}}`

   (g) `d["other"]["blur"] = 0.1`

   A: *New value of* `d`*:*
   `{'R': 0, 'G': 255, 'B': 0, 'other': {'opacity': 0.6, 'blur': 0.1}}`

   (h) `d.items()`

   A: `dict_items([('R', 0), ('G', 255), ('B', 0), ('other', {'opacity': 0.6})])`

4. Evaluate the following given the assignment `s1 = {1, 2, 4}` and `s2 = {3, 4, 5}`. If `s1` or `s2` change as a result, give their new value. Assume `s1` and `s2` are reset to their original values after each.

   (a) `s1.add(7)`

   A:*New value for* `s1`*:*
   `{1, 2, 4, 7}`

   (b) `s1.add(2)`

   A:`s1` *does not change (2 is already in the set)*

   (c) `s2.remove(5)`

   A:*New value for* `s2`*:*
   `{3, 4}`

   (d) `s1 & s2`

   A:`{4}`

   (e) `s1.union(s2)`

   A:`{1, 2, 3, 4, 5}`

   (f) `s1 - s2`

   A:`{1, 2}`

Problems

1. Write a function which takes a string as input and prints the frequency of each character in the string using a dictionary.

   A:

```python
def freq_count(words):
    freqs = {}
    for letter in words:
        if letter in freqs:
            freqs[letter] += 1
        else:
            freqs[letter] = 1
    for key in freqs:
        print(key, freqs[key])
```

2. Write a function which takes two lists as input and returns a list containing the numbers which they both have in common.

A:

```python
def unique(list_1, list_2):
    set_1 = set(list_1)
    set_2 = set(list_2)
    common = set_1 & set_2
    return list(common)
```