COMP10001 Foundations of Computing
Semester 2, 2022

Tutorial Solutions: Week 5

— VERSION: 1479, DATE: AUGUST 22, 2022 —

Discussion

1. What is a method? How do methods differ from functions? How are they the same?

   A: *Both methods and functions run some pre-defined code to achieve a task. Both are called with brackets which can contain arguments. Functions are called anywhere, but methods are "attached" to an object, and are called with a dot after the object name. This means methods can do more interesting things like edit the object they are called on. You won't be expected to write your own methods in this subject, but may learn how to in the advanced lecture on object oriented programming. We will write lots of functions though.*

   Now try Exercise 1

2. What is the difference between a list and a tuple?

   A: *Lists are "mutable", so allow for values to be added and removed at will. Tuples are "immutable" meaning that trying to change a value in a tuple will cause an error. You can, of course, use the values in a tuple to create a new one (like creating new strings with the `+=` operation). Both lists and tuples are data types which can store multiple objects in an ordered sequence. Lists are initialised by surrounding objects with square brackets `[]` or by using the `list()` function. This is achieved by surrounding objects in brackets separated by commas, or calling the `tuple()` function. There is no limit to the amount of objects which can be stored in these sequences other than memory limits of your computer. It is common to see tuples used to group related objects together, such as in a set of coordinates `(x, y)` or to return multiple objects from a function.*

3. How do we add and remove items from a list?

   A: *Items are added to a list using the `.append(item)` method, which takes the item to be added as its argument and returns nothing. `.insert(index, item)` and `.extend(lst)` can also be used to add items to a list. The best way to remove an item is the `.pop(index)` method which takes the index to be removed (defaulting to the last index of the list) and returns the value which was removed. `del`, `.remove(item)` and `.clear()` can also be used to remove items from a list.*

   Now try Exercise 2

4. What is iteration in programming? Why do we need it?

   A: *Iteration is the process of executing a section of code repeatedly, often with a small difference each time. We often need to do the same thing multiple times in programming: iteration allows us to do this without writing the same instructions many times over. Without iteration, code would be tedious to write, hard to read and error-prone.*

5. What are the two main types of loop in python? How do we write them?

   A: *The two types are `for` loops (`for <loop variable> in <sequence>:`) and `while` loops (`while <condition>:`). Both require the loop body to be indented. `while` loops similar to `if` statements: a condition is tested to decide whether loop repeats every time. A `for` loop automates some aspects of the `while` loop, including "loop variable" initialisation and update.*

6. What do we mean by the loop variable in a `for` loop?

   A: *The loop variable is the variable which changes each time the code in a for loop is repeated, taking the value of successive items in the sequence being looped through. This is what allows the for loop to execute slightly different code each time it's repeated.*

7. What are the differences between the two main types of loops? In which situations are they used?

   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 eg. 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.

8. 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 the required amount 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.

Now try Exercises 3 & 4

Exercises

1. Evaluate the following method calls given the assignment `s = "Computing is FUN!"` Think about the input and output of each method. You're not expected to know all methods for all types: if you haven't seen some of these before, your best guess based on the name will probably be right!

   (a) `s.isupper()`
   A: *`False`*

   (b) `s.upper()`
   A: *`'COMPUTING IS FUN!'`*

   (c) `s.endswith("FUN!")`
   A: *`True`*

   (d) `s.count('i')`
   A: *`2`*

   (e) `s.strip('!')`
   A: *`'Computing is FUN'`*

   (f) `s.replace('i', '!')`
   A: *`'Comput!ng !s FUN!'`*

2. Evaluate the following given the assignment `lst = [2, ("green", "eggs", "ham"), False]`

   (a) `lst[2]`
   A: *`False`*

   (b) `lst[1][-2]`
   A: *`"eggs"`*

   (c) `lst[1][-2][:3]`
   A: *`"egg"`*

   (d) `lst.append(5); print(lst)`
   A: *`[2, ("green", "eggs", "ham"), False, 5]`*

   (e) `lst.pop(2); print(lst)`
   A: *`False`* *(this is the value removed when .pop() is run)*
   `[2, ("green", "eggs", "ham")]`

3. What is the output of the following snippets of code containing loops?

   (a)
   ```
   i = 2
   while i < 8:
       print(f"The square of {i} is {i * i}")
       i = i + 2
   ```

   A:
   ```
   The square of 2 is 4
   The square of 4 is 16
   The square of 6 is 36
   ```

   (b)
   ```
   for ingredient in ("corn", "pear", "chilli", "fish"):
       if ingredient.startswith('c'):
           print(ingredient, "is delicious!")
       else:
           print(ingredient, "is not!")
   ```

   A:
   ```
   corn is delicious!
   pear is not!
   chilli is delicious!
   fish is not!
   ```

   (c)
   ```
   i = 0
   colours = ("pink", "red", "blue", "gold", "red")
   while i < len(colours):
       if colours[i] == "red":
           print("Found red at index", i)
       i += 1
   ```

   A:
   ```
   Found red at index 1
   Found red at index 4
   ```

(d)
```
MIN_WORD_LEN = 5
long_words = 0
text = "There once lived a princess"
for word in text.split():
    if len(word) >= MIN_WORD_LEN:
        print(word, "is too long!")
        long_words += 1
print(long_words, "words were too long")
```

A:
```
There is too long!
lived is too long!
princess is too long!
3 words were too long
```

4. Rewrite the loops in question 3a and 3b converting `for` loops to `while` loops and vice versa. (We'll include answers for c and d for good measure)

(a) **A:**
```
for i in range(2, 8, 2):
    print(f"The square of {i} is {i*i}")
```

(b) **A:**
```
ingredients = ("corn", "pear", "chilli", "fish")
i = 0
while i < len(ingredients):
    ingredient = ingredients[i]
    if ingredient.startswith('c'):
        print(ingredient, "is delicious!")
    else:
        print(ingredient, "is not!")
    i += 1
```

(c) **A:**
```
colours = ("pink", "red", "blue", "gold", "red")
for i in range(len(colours)):
    if colours[i] == "red":
        print("Found red at index", i)
```

(d) **A:**
```
MIN_WORD_LEN = 5
text = "There once lived a princess"
long_words = 0
words = text.split()
i = 0
while i < len(words):
    if len(words[i]) >= MIN_WORD_LEN:
        print(words[i], "is too long!")
        long_words += 1
    i += 1
print(long_words, "words were too long")
```

An alternative solution stores the split text and iterates through it destructively: removing elements each iteration until it's empty.

```
MIN_WORD_LEN = 5
text = "There once lived a princess"
long_words = 0
words = text.split()
while words:
    word = words.pop()
    if len(word) >= MIN_WORD_LEN:
        print(word, "is too long!")
        long_words += 1
print(long_words, "words were too long")
```

Problems

1. Write a function which takes a positive integer input `n` and prints the thirteen times tables from `1 * 13` until `n * 13`.

   A:

   ```
   def thirteen_table(n):
       for i in range(1, n+1):
           print(f"{i} * 13 = {i * 13}")
   ```

2. Write a function which converts a temperature between degrees Celsius and Fahrenheit. It should take a float, the temperature to convert, and a string, either `'c'` or `'f'` indicating a conversion from degrees Celsius and Fahrenheit respectively. The formulae for conversion are below.

   A:

   ```
   def convert_temp(degrees, unit):
       if unit == 'c':
           result = (degrees * 1.8) + 32
           return result
       elif unit == 'f':
           result = (degrees - 32) / 1.8
           return result
   ```

3. Write a function which takes a tuple of strings and returns a list containing only the strings which contain at least one exclamation mark or asterisk symbol. `words_with_symbols(('hi', 'there!', '*_*'))` should return `['there!', '*_*']`.

   A:

   ```
   def words_with_symbols(words):
       with_symbols = []
       for word in words:
           for letter in word:
               if letter in ('!', '*'):
                   with_symbols.append(word)
                   break
       return with_symbols
   ```