COMP10001 Foundations of Computing
Semester 1, 2019

Tutorial Questions: Week 5

— VERSION: 1474, DATE: MARCH 13, 2019 —

Discussion

1. What is a "function"? How do we call (use) one? How do we define one ourselves?

   A: *A function is a block of reusable code which is defined once and can be reused wherever you would like in a program. A function is called by writing its name and then a pair of brackets. Depending on the function, you can list arguments inside the brackets, separated by commas. To define a function, we use the def keyword followed by the function name, parameter(s) in brackets, a colon and then the function body, indented.*

2. What does it mean to "return" a value from a function and why would we want to? Does a function always need a return value?

   A: *Returning a value from a function makes it available to the line of code which called the function, so that it can be, for example, assigned to a variable. This return value could be the result of a calculation or a status message or something else. A function doesn't need to have a return value: it could just perform an operation such as* `print()`*. You return with the return keyword, followed by the value you want to pass. At this point, the function execution terminates.*

3. Why are functions so useful? Could we live without functions?

   A: *Functions reduce code duplication by allowing us to run the same block of code multiple times. This saves our code from becoming long and repetitive, and makes it easier to edit in the future. It also allows code we write to be used elsewhere, creating more modular code which we can use in more places. We could theoretically live without functions, but code would be very messy and error-prone without them.*

4. Why are brackets important when calling a function? Are they needed even if it takes no arguments?

   A: *Brackets are the difference between a reference to the name of the function and an actual call of the function. Even if there are no arguments, brackets are still needed to call the function (they will simply be empty).*

5. 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. We won't learn how to write our own methods in this subject, though we will write functions.*

   Now try Exercises 1 & 2

6. 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.*

7. What are the two 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 while loop, including "loop variable" initialisation and update.*

8. 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.*

   Now try Exercises 3 - 5

Exercises

1. What's wrong with this code? How can you fix it?

```
def calc(n1, n2):
    answer = n1 + (n1 * n2)
    print(answer)

num = int(input("Enter the second number: "))
result = calc(2, num)
print("The result is:", result)
```

A: *This function prints the answer to the calculation it's performed rather than returning it. This means that the value of* `result` *will be* `None` *and the last line will not work as intended.*

2. 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 the methods available to you: if you haven't seen some of these before, have a guess at what they do and you'll probably be right!

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

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

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

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

3. What is wrong with this code? How would you fix it?

```
def largest_num(nums):
    maxnum = nums[0]
    for num in nums:
        if num > maxnum:
            maxnum = num
            return maxnum

print(largest_num([1, 2, 3]))
```

A: *A* `return` *statement will exit from the function immediately when it's run, returning the current value of the variable back to the calling program. In this case, the first time a number is found which is bigger than the first, the return statement is executed and the function exits. We still want to check the rest of the list for larger values though, so in the case above where a bigger value exists later in the list, the function returns the incorrect value.*
Fix this by indenting the return statement only four spaces (in line with `maxnum = nums[0]`*)*

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

   (a)
```
for i in range(5):
    print(i**2)
```

   A:
```
0
1
4
9
16
```

   (b)
```
for ingredient in ("ham", "cheese", "hollondaise", "lettuce"):
    if ingredient.startswith('h'):
        print(ingredient, "is delicious")
    else:
        print(ingredient, "is tasty")
```

   A:
```
ham is delicious
cheese is tasty
hollondaise is delicious
lettuce is tasty
```

(c)
```
i = 0
colours = ("olive", "red", "violet", "turquoise", "red", "red", "amber")
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
Found red at index 5
```

5. Do the following code snippets do the same thing? What are some advantages and disadvantages of each snippet?

```
print("We␣need␣some␣saws")
print("We␣need␣some␣hammers")
print("We␣need␣some␣cogs")
print("We␣need␣some␣nails")
```

```
def get_str(part):
    return f"We␣need␣some␣{part}"

print(get_str("saws"))
print(get_str("hammers"))
print(get_str("cogs"))
print(get_str("nails"))
```

```
def get_str(part):
    return f"We␣need␣some␣{part}"

parts = ("saws", "hammers", "cogs", "nails")

for part in parts:
    print(get_str(part))
```

A: *These snippets will all result in the same output. The first simply prints the lines which is relatively short but results in a lot of repetition of `"We␣need␣some"`. The second abstracts the creation of the string to be printed in a function which is more elegant, less repetitive and easier to edit. The third removes all repetition by using a for loop to iterate over the parts to be used in the `get_str()` function. This results in code which makes use of all programming structures available to be very clear and efficient.*

Problems

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

   A:

   ```
   def thirteen_table(num):
       for i in range(1, num+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 string, finds the first vowel in it and returns the amount of times that vowel appears in it. If it's empty, return 0.

   A:

   ```
   def count_vowel(word):
       if not word:
           return 0
       for letter in word:
           if letter in 'aeiou':
               return word.count(letter)
   ```