EMAT10007 – Introduction to Computer Programming

Exercises – Week 4. Functions

Reminder: The exercise sheets use `<?>` as a placeholder to represent something missing - this could be an operator, a variable name, function name, etc. Multiple `<?>` in the same question are not necessarily representing the same thing.

Part 1 Functions

Exercise 1 - Functions (Essential)

1. Can you complete the `RetSquared` function below by replacing the blank `<?>` with the correct variable name?

```
def RetSquared(Number):
    # Returns the square
    Squared = <?> ** 2
    return <?>
```

2. Use the `RetSquared` function and a `for` loop to print the squares of all integers from 1 to 10.
3. Can you write a general function for raising numbers to powers? The function should take two arguments `Number` and `Power` and return the number raised to the given power. Remember to give your new function a sensible name.
4. Use your new general function to print the powers of 2 up to 2^{10}.
5. Change this function so that the argument `Power` has the default value `1`.
6. Write a new function which has arguments `Number` and `Powers`. If I have inputs `Number=2` and `Powers=[1, 3, 4]`, the function should return the list `[2, 8, 16]`.
   How else can you write a function when you don't know the exact number of inputs?
7. Write a function called `inverse` that computes the inverse of a number x. That is, the function returns $1/x$ if $x \neq 0$. If $x = 0$ then your function should return the string "undefined". Show that your function works by computing `inverse(2)` and `inverse(0)`.
8. Write a function that swaps the values of two variables. Make sure your function returns the new values. Include a docstring that explains how to call the function. Use the `help` function to print the docstring and read about your function. Then, create two variables `a = ['red', 'blue', 'green']` and `b = {1, 3, 5}` and use your function to swap their values.

Exercise 2 - 99 Bottles of Beer (Essential)

"99 Bottles of Beer" is a traditional song in the United States and Canada. It is popular to sing on long trips, as it has a very repetitive format which is easy to memorise, and can take a long time to sing. The song's simple lyrics are as follows:

```
99 bottles of beer on the wall, 99 bottles of beer.  Take one down, pass it around,
98 bottles of beer on the wall.
```

The same verse is repeated, each time with one fewer bottle. The song is completed when the singer or singers reach zero. Your task here is to write a Python program capable of generating all the verses of the song. This can be done in two steps:

1. Define a function `CreateVerse(CurrentBottle)` that returns the verse of the song based on the value of `CurrentBottle` as a string. For example, if `CurrentBottle = 99`, then the text above should be returned as a string.
2. Write another function `SingSong(TotalBottles)` that prints all of the verses of the song starting from `TotalBottles`. **Hint**: call the function `CreateVerse` in the function `SingSong`.
3. Write an alternate function that takes a number of total bottles e.g. `FillVerses(TotalBottles)`, but instead of printing the verses, returns a list filled with each verse of the song. Use a list comprehension to fill the list, rather than loops. After calling the function, you should then be able to loop through the list and print the verses in the correct order.
   Hint: There are several ways in which to ensure that the verses are printed in order e.g. `reversed(), sorted(...,reverse=...)`

Part 2. Functions arguments and scope

Exercise 3 - Variadic functions (Essential)

Variadic functions can take any number of arguments, just like the print function, which works when used in the following way:
`print("Hello, I am", Age, "years old, born in the month of", Month, ".")`

1. Write a function that takes an unknown amount of numbers (integers or floats) and then:
   - Calculates the sum of the numbers
   - Calculates the product of all of the numbers

   and **returns** these values as opposed to printing them from the function.
2. Write a function that takes an unknown amount of numbers as its arguments and returns the `min`, `max` and `average` of those numbers.

Exercise 4 - Variable scope (Essential)

Variable scope. It is important to understand a little bit about **scope** – an important concept in programming. The following exercises will demonstrate how variables are treated depending on whether they are declared inside (local scope) or outside (global scope) of a function.

1. Examine the following code and predict the value(s) printed to the screen. Then run the code to check your prediction.

   ```
   def F():
       print(S)
   ```

```
S = "I hate spam"
F()
```

2. Examine the following code and predict the value(s) printed to the screen. Then run the code to check your prediction.

```
def F():
    S = "Me too"
    print(S)

S = "I hate spam"
F()
print(S)
```

3. Examine the following code and predict the value(s) printed to the screen. Then run the code to check your prediction.

```
def F():
    global S
    S = "Me too"
    print(S)

S = "I hate spam"
F()
print(S)
```

4. Examine the following code and predict the value(s) printed to the screen. Then run the code to check your prediction.

```
def F():
    T = "I am a variable"
    print(T)

F()
print(T)
```

Part 3. Recursive functions

Exercise 5 - Recursive functions (Essential)

1. Write a recursive function that computes the factorial of a positive integer n. Recall that the factorial is defined as $n! = n \times (n-1) \times \ldots \times 1$. By definition, $0! = 1$. Check your code works by computing $5! = 120$. **Hint:** Notice that $n! = n \times (n-1)!$.
2. Write a recursive function called `Fib(n)` that calculates n-th number in the Fibonacci sequence using a recursive function. The following code

```
for i in range(1,15):
    print(Fib(i), end=" ")
```

should print:

```
1  1  2  3  5  8  13  21  34  55  89  144  233  377
```

If you are struggling, start by researching how to calculate the Fibonacci sequence. There are lots of tutorials online, so be sure to search for implementations of the Fibonacci function using *recursion*. **Hint:** You will need to identify the "special cases" of your program which prevent the function from calling itself infinitely many times.

3. Read about the relative advantages and disadvantages of using *recursive* functions i.e. what happens if `n >= 100`? Is there another version of the Fibonacci function that does not use recursion? Can this be called on large values of `n`?

Part 4. Advanced questions

Exercise 6 - More functions

1. Write a function that computes the median of three distinct numbers that are input in arbitrary order. Recall that the median in the number in the middle; it's neither the largest nor the smallest. For example, the median of 4, 1, and 2 is 2.
2. Write a function called `sum_digits` that computes the sum of the digits of an integer. For example `sum_digits(1234) = 1 + 2 + 3 + 4 = 10`. There are several ways of doing this; however, if you want a challenge, try to do this using only mathematical operations.
3. In Python it's easy to sort a list of numbers `L` using the `sort` method, that is, by calling `L.sort()`. Write your own function that sorts a list with an arbitrary number of numbers.
4. Write a function which returns the prime factorization of a number. **Note:** Learn about prime factorization here: `https://www.mathsisfun.com/prime-factorization.html`