Version: 2025/04/08 13:40

CSC1015F Assignment 7B (70 Marks)

Functions

Learning Objectives

By the end of this assignment, you should be able to:

- Understand what a Function is.
- Be able to create your own Function.
- Understand what parameters are and when/how to use them.
- Create a Function that returns a value.
- Understand what are docstrings, nested functions and the main Function.
- Set default values to parameters.

Assignment Instructions

Previous assignments have involved solving programming problems using input and output statements, '`if`' and '`if-else`' control flow statements, '`while`' statements, ‘`for`’ statements, and statements that perform numerical and string manipulation.

This assignment builds on these technologies and offers practice using functions and modules. Functions are very effective when used in conjunction with a divide-and-conquer approach to problem solving. An example on the use of functions is given in **Appendix A.**

NOTE Your solutions to this assignment will be evaluated for correctness and for the following qualities:

- Documentation
  - Use of comments at the top of your code to identify program purpose, author and date.
  - Use of comments within your code to explain each non-obvious functional unit of code.
- General style/readability
  - The use of meaningful names for variables and functions.
- Algorithmic qualities
  - Efficiency, simplicity

These criteria will be manually assessed by a tutor and commented upon. In this assignment, up to 10 marks will be deducted for deficiencies.

Question 1 – Vehicle Telemetry Black Box Parser [25 marks]

The Vehicle Telemetry Black Box Parser is designed to extract and process specific information from raw data strings emitted by a vehicle's telemetry system. These strings simulate the kind of data you might find in a black box or onboard diagnostics system of a vehicle. On the Amathuba assignment page, you will find a skeleton for a program called '`telemetry.py`'. The intent is that the program be used to extract useful data from a raw data stream obtained from a vehicle black box.

You may not use a List or Dictionary, or their associated functions to answer this question.

The data from the vehicle black box contains a block with the format:

```
"... START speed:3200|90.5 lat=51.5074 long=-0.1278 name=Alex Turner END ..."
```

The program must output the data in the following format:

```
Engine RPM: RPM
Vehicle speed: speed km/h
Coordinates: (latitude, longitude)
Driver's name: name
```

Sample IO *(The input from the user is shown in **bold** font – do not program this)*:

```
Enter the raw data:
adsf ad START speed:4300|112.5 lat=61.5034 long=-0.12908 name=Lesedi Lebaka
END bdfg hsd
Vehicle Black Box Information:
Engine RPM: 4300
Vehicle speed: 112.5km/h
Coordinates: (61.5034, -0.12908)
Driver's name: Lesedi Lebaka
```

To complete the program, you must implement the *rpm*, *speed*, *coordinates*, *driver_name* and *get_data_block* functions. The functions have been identified by applying a divide-and-conquer strategy. Each solves a part of the overall problem:

- `get_data_block(raw_data)`

  Given a string of raw data as a parameter, the *get_data_block* function extracts the sub string starting with 'BEGIN' and ending with 'END'.
- `rpm(data_block)`

  Given a data block string as a parameter, the *rpm* function returns the engine RPM as an integer.
- `speed(data_block)`

Given a data block string as a parameter, the *speed* function returns the vehicle speed as a float.

- `coordinates(data_block)`

  Given a data block string as a parameter, the *coordinates* function returns the latitude and longitude from the telemetry block as a tuple.
- `driver_name(data_block)`

  Given a data block string as a parameter, the *driver_name* function returns the driver's name in title case.
- `main()`

  get an input string of raw data from the user and extract useful information and display it as shown in the sample I/O.

Examples

- `get_data_block("adsf ad START speed:3100|98.0 lat=50.234563 long=-0.432658 name=Thabang Diale END bdfg hsd")` returns the block string `'speed:3100|98.0 lat=50.234563 long=-0.432658 name=Thabang Diale'`.
- `rpm("speed:3100|98.0 lat=50.234563 long=-0.432658 name=Thabang Diale")` returns `3100`.
- `speed("speed:3100|98.0 lat=50.234563 long=-0.432658 name=Thabang Diale")` returns `98.0`.
- `coordinates("speed:3100|98.0 lat=50.234563 long=-0.432658 name=Thabang Diale")` returns `(50.234563, -0.432658)`.
- `driver_name("speed:3100|98.0 lat=50.234563 long=-0.432658 name=Thabang Diale")` returns `'Thabang Diale'`.

NOTE: The automatic marker will test each of your functions individually. To enable this, you MUST NOT remove the following lines from the skeleton:

```
if __name__=='__main__':
    main()
```

Question 2 *Syllables* [25 marks]

Linguists often like to break up words into their syllables. On the Amathuba assignment page you will find the skeleton of program called `syllables.py` which repeatedly asks the user to enter a word (or 'quit') and then outputs (to the screen) the number of words that it contains.

Sample IO *(The input from the user is shown in* ***bold*** *font – do not program this)*:

```
Enter a word (or 'q' to quit):
magic
The word 'magic' has 2 syllables.

Enter a word (or 'q' to quit):
hair
The word 'hair' has 1 syllable.

Enter a word (or 'q' to quit):
q
```

Mimicking the structure of the program of question one, to complete the program:

1. A function '`count_syllables`' is needed:
   - `count_syllables(word)`
     Given a word, returns the number of syllables it contains.
2. And a main method that uses it.

You should apply a divide-and-conquer strategy to developing the function: identify sub problems and develop further functions for them e.g.

- `is_vowel(letter)`
  Given a letter, returns True if it is a vowel, False otherwise.
- `next_vowel(word, index)`
  Given a word and an index, 0 ≤ index < len(word), return the location of the next vowel in the word, or len(word) if no such letter exists.

Here are the rules you should use to implement the `count_syllables`:

- A syllable can be identified if there is a sequence of vowels {**a**, **e**, **i**, **o**, **u**, **y**} in a word. NB: y is considered a vowel.
- If the letter "e" appears at the end of a word it should be disregarded unless it happens to be the only vowel in the word.
- The lowest number of syllables a word can have is 1.

Examples:

Word	**Number of Syllables**
Harry	2
Hair	1
Hare	1
The	1

NOTE: The automatic marker will test your program AND will test your function independent of the main routine.

Question 3 [20 marks]

This question concerns completing a skeleton for a program called *calendar_month.py* that accepts the name of a month and a year as input and prints out the calendar for that month. Download this program from the Amathuba page for this assignment.

Sample IO *(The input from the user is shown in* ***bold*** *font – do not program this)*:

```
Enter month:
May
Enter year:
2020
May
Mo Tu We Th Fr Sa Su
             1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
```

To complete the program you are required to complete a number of functions, including a *main* function.

- `day_of_week(day, month, year)`
  Given a date consisting of day of month, month number and year, return the day of the week on which it falls. The function returns 1 for Monday, 2 for Tuesday, ..., 7 for Sunday.
- `is_leap(year)`
  Given a year return True if it is a leap year, False otherwise. This function returns a Boolean value.
- `month_num(month_name)`
  Given the name of a month, return the month number i.e. 1 for January, 2 for February, ..., 12 for December. The name can be in UPPER CASE, lower case or Title Case.
- `num_days_in(month_num, year)`
  Given a month number and year, return the number of days in the month.
- `num_weeks(month_num, year)`
  Given a month number and year, return the number of weeks that the month spans. (The first week is the week in which the first of the month falls, the last week is the week in which the last day of the month falls. Counting from first to last gives the number of weeks.)

- `week(week_num, start_day, days_in_month)`
  Given a week number, (1st, 2nd, ...), the day on which the 1st of the month falls (1 for Monday, 2 for Tuesday, ...), and the number of days in the month, return a string consisting of the day of the month for each day in that week, starting with Monday and ending with Sunday.
- `main()`
  Obtain the name of a month and a year from the user and then print the calendar for that month by obtaining the number of weeks and then obtaining the week string for each.

The functions have been identified by applying a divide-and-conquer strategy. Each solves a part of the overall problem.

Examples

- `day_of_week(1, 4, 2020)` returns integer 3 (which represents Wednesday).
- `is_leap(2020)` returns the Boolean value True.
- `month_num('April')` returns integer 4.
- `num_days_in(4, 2020)` returns integer 30.
- `num_weeks(4, 2020)` returns integer 5.
- `week(1, 3, 30)` returns the string `'1 2 3 4 5'`
- `week(2, 3, 30)` returns the string `'6 7 8 9 10 11 12'`

HINTS:

- Days of the month.
- We can use Gauss's formula to calculate the day of the week on which the 1st of January of a given year falls. This could be adapted for the *day_of_week* function.

  Alternatively, you could consider Zeller's congruence:
  Given day of the month, *d*, month number, *m*, and year, *y*;
  If the month is January or February, then add 12 to *m* and subtract 1 from y.
  Compute:
  $$\left(d + \left\lfloor\frac{13(m+1)}{5}\right\rfloor + y + \left\lfloor\frac{y}{4}\right\rfloor - \left\lfloor\frac{y}{100}\right\rfloor + \left\lfloor\frac{y}{400}\right\rfloor\right) \mathit{modulus}\ 7$$
  The result, *h*, is a number 0 to 6 where 0 is Saturday, 1 is Sunday, ..., and 6 is Friday.
  The formula **(h + 5) modulus 7) + 1** will give a value from 1 to 7 where 1 is Monday, 2 is Tuesday, ..., and 7 is Sunday.
- The floor of a value, *v*, written $\lfloor v \rfloor$, is the largest integer value that is smaller than *v*. In Python you write '`math.floor(v)`'.

NOTE: The automatic marker will test each of your functions individually. To enable this, you MUST NOT remove the following lines from the skeleton:

```
if __name__=='__main__':
    main()
```

Submission

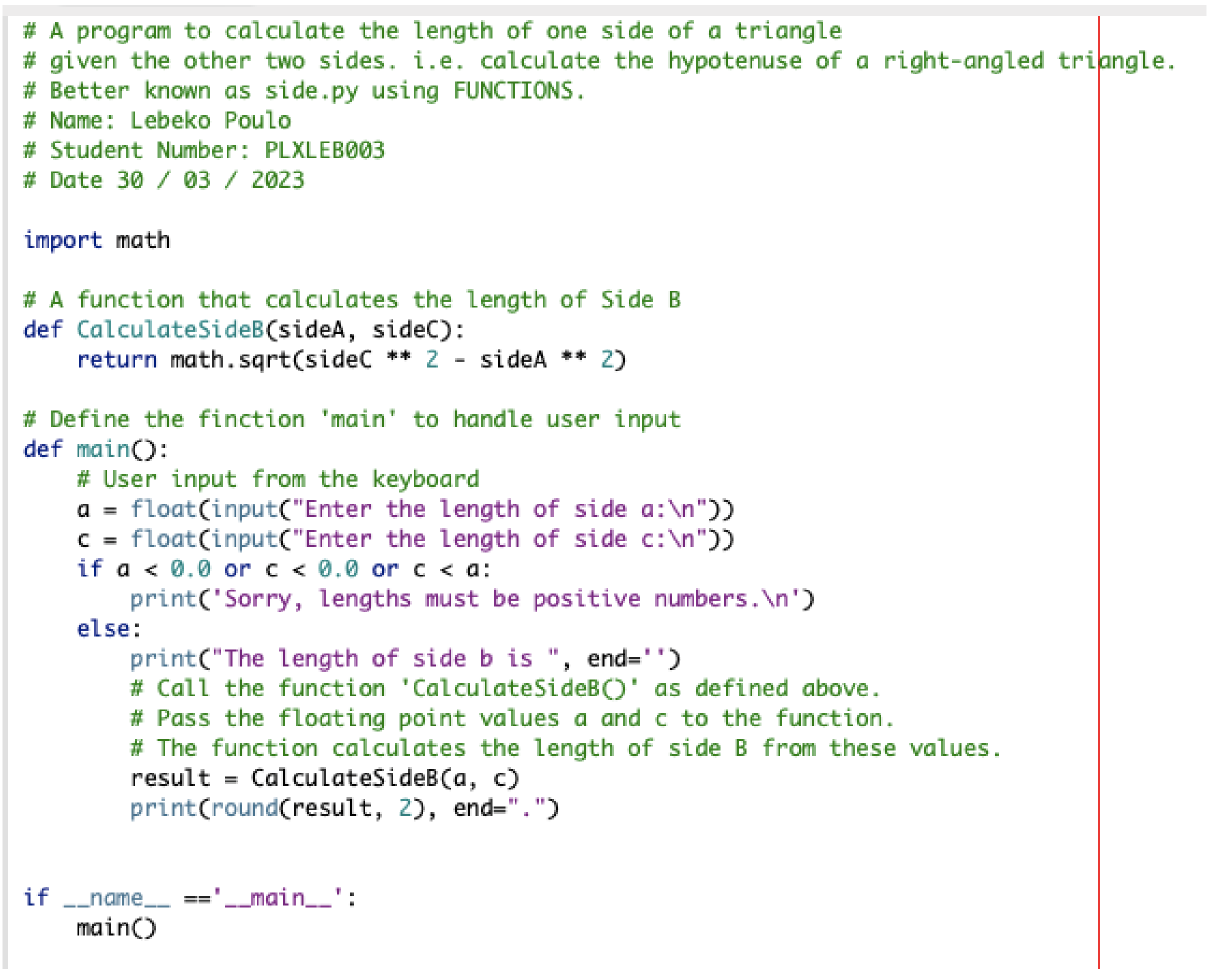
Create and submit a Zip file called '`ABCXYZ123.zip`' (where ABCXYZ123 is YOUR student number) containing `telemetry.py`, `syllables.py` and `calendar_month.py`.

NOTES:

1. FOLDERS ARE NOT ALLOWED IN THE ZIP FILE.
2. As you will submit your assignment to the Automarker, the Assignment tab may say something like "Not Complete". THIS IS COMPLETELY NORMAL. IGNORE IT.

APPENDIX A – A program that calculates the hypotenuse of a right-angled triangle given the other two sides.

To explain by example, consider a scenario where you are asked to write a program called `side.py` to calculate the side, *b,* of a right-angled triangle given the sides *a* and *c* using Pythagoras' Theorem. Here is a solution that demonstrates the use of functions:



```
# A program to calculate the length of one side of a triangle
# given the other two sides. i.e. calculate the hypotenuse of a right-angled triangle.
# Better known as side.py using FUNCTIONS.
# Name: Lebeko Poulo
# Student Number: PLXLEB003
# Date 30 / 03 / 2023

import math

# A function that calculates the length of Side B
def CalculateSideB(sideA, sideC):
    return math.sqrt(sideC ** 2 - sideA ** 2)

# Define the finction 'main' to handle user input
def main():
    # User input from the keyboard
    a = float(input("Enter the length of side a:\n"))
    c = float(input("Enter the length of side c:\n"))
    if a < 0.0 or c < 0.0 or c < a:
        print('Sorry, lengths must be positive numbers.\n')
    else:
        print("The length of side b is ", end='')
        # Call the function 'CalculateSideB()' as defined above.
        # Pass the floating point values a and c to the function.
        # The function calculates the length of side B from these values.
        result = CalculateSideB(a, c)
        print(round(result, 2), end=".")


if __name__ =='__main__':
    main()
```

There are two functions, one called '`main`', and the other '`CalculateSideB`'. Between them, they break the problem into two parts.

- The *main* function is responsible for handling user input and output.
- The *CalculateSideB* function is responsible for calculation.

The *main* function contains a 'function call'.

- It calls *CalculateSideB*, passing the floating point values of *a* and *c*.
- The *CalculateSideB* function uses the values to calculate and return the value of the side *b*.
- The *main* function assigns the value it receives to the variable '*result*'.

- On the next line (and final line) it prints out the value of *result* correct to 2 decimal places.

By breaking the programming problem into two parts, each can be concentrated on without concern for the other. The *CalculateSideB* function can be developed without concern for where a and c come from. The *main* function can be developed without concern for exactly what *CalculateSideB* will do. It's enough to know simply that it requires two values (the sides *a* and *c*) and that it will calculate the length of *b*.

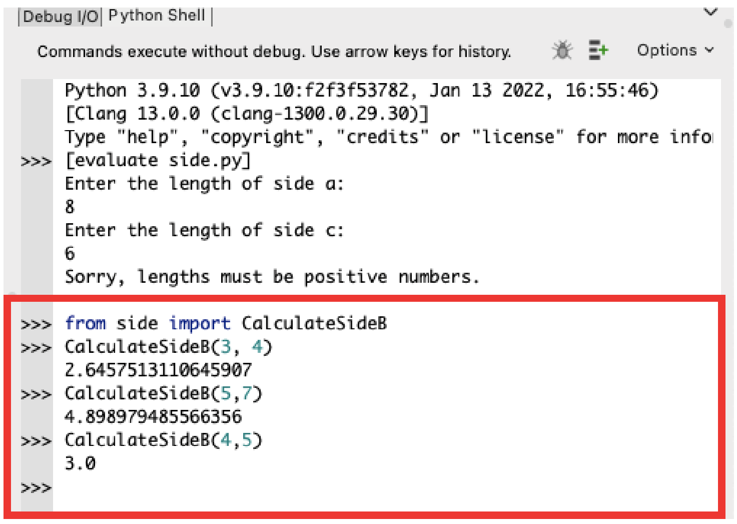
Admittedly, this programming problem is simple and can probably be solved quite satisfactorily without the use of functions, however, it serves to convey the idea.

The idea of divide-and-conquer works best if you have the techniques and technology to fully support working on one part without concern for another. If, say, you were developing the Pythagoras' Theorem program and you chose to concentrate on the *CalculateSideB* function first, you'd probably want to check it worked correctly. But surely that means you need the *main* function so that you can obtain useful inputs?

A technique for dealing with this is to have a '`stub`' *main* function which with which to make test calls to *CalculateSideB* e.g.

```
def main():
      print(CalculateSideB(3, 4))
      print(CalculateSideB(5, 7))
      # ...
```

Alternatively, Wing IDE provides a piece of technology in the form of a Python shell. Within the shell you can import functions that you are working on and then write expressions that use them (see the function calls inside the red rectangle below):



```
    Python 3.9.10 (v3.9.10:f2f3f53782, Jan 13 2022, 16:55:46)
    [Clang 13.0.0 (clang-1300.0.29.30)]
    Type "help", "copyright", "credits" or "license" for more info
>>> [evaluate side.py]
    Enter the length of side a:
    8
    Enter the length of side c:
    6
    Sorry, lengths must be positive numbers.
>>> from side import CalculateSideB
>>> CalculateSideB(3, 4)
    2.6457513110645907
>>> CalculateSideB(5,7)
    4.898979485566356
>>> CalculateSideB(4,5)
    3.0
>>>
```

The screenshot illustrates its use. Lines entered by the user have a prompt, ‘`>>>`’ beside them. Lines without the prompt are responses from the Python shell.

1. The user enters an `import` statement for the *CalculateSideB* function.

2. The user then enters a function call expression, calling *CalculateSideB* with the values 3 and 4 for *a* and *c*.

3. The result, 2.6457513110645907 is printed on the next line.

4. The user enters another function call expression, this time with the values 5, and 7,

5. And the result, 4.898979485566356 is printed on the next line.

etc….