Version 28/04/2021 14:49

CSC1015F Assignment 6: Functions

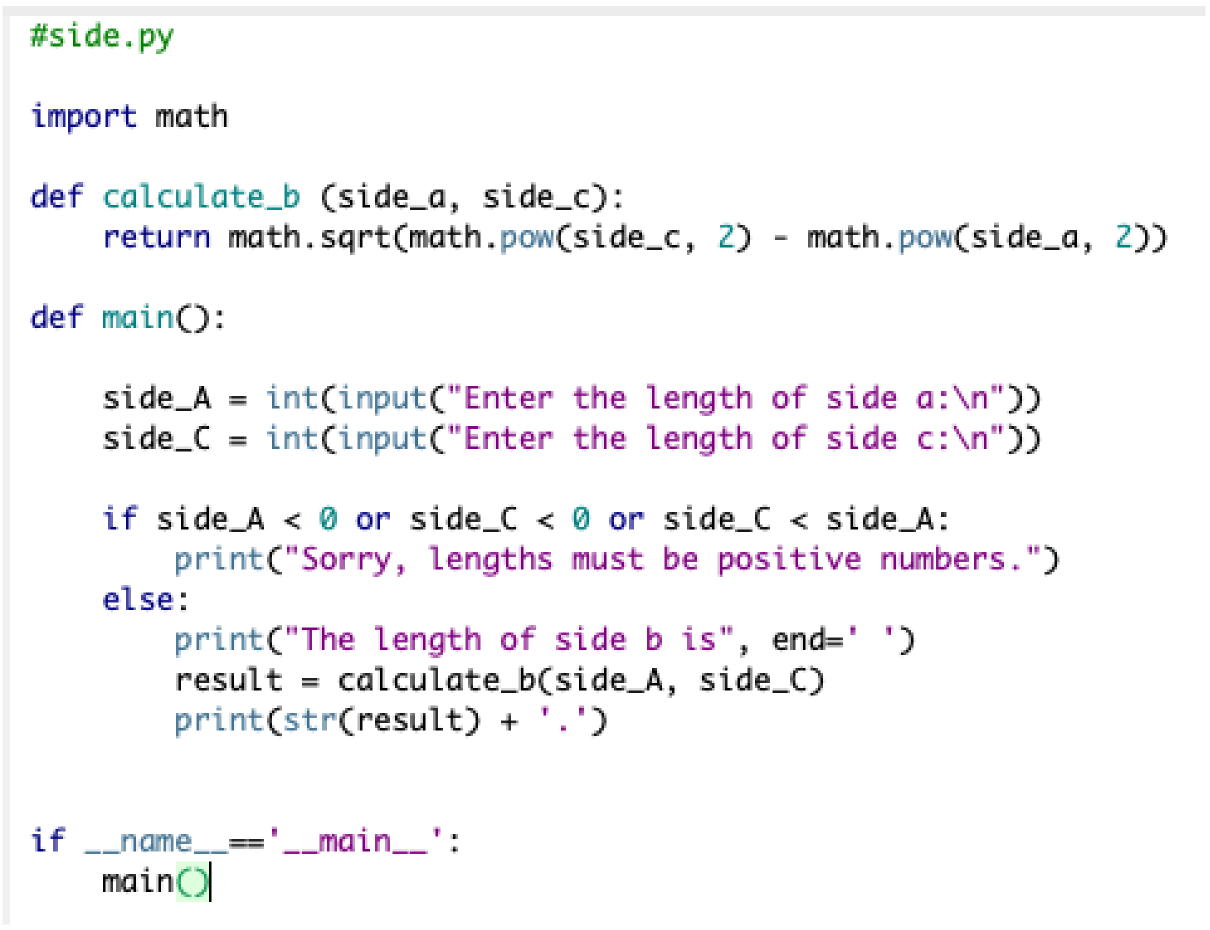
Assignment Instructions

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.

To explain by example, assignment 2 question 4 asks you 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:



```
#side.py

import math

def calculate_b (side_a, side_c):
    return math.sqrt(math.pow(side_c, 2) - math.pow(side_a, 2))

def main():

    side_A = int(input("Enter the length of side a:\n"))
    side_C = int(input("Enter the length of side c:\n"))

    if side_A < 0 or side_C < 0 or side_C < side_A:
        print("Sorry, lengths must be positive numbers.")
    else:
        print("The length of side b is", end=' ')
        result = calculate_b(side_A, side_C)
        print(str(result) + '.')


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

There are two functions, one called ‘main’, and the other ‘calculate_b’. Between them, they break the problem into two parts.

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

The *main* function contains a ‘function call’.

- It calls *calculate_b*, passing the floating point values of *a* and *c*.

- The *calculate_b* 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*.

(*Actually, the result variable isn't really necessary, and we could just have had a final print statement '*`print(str(calculate_b(side_A, side_C))+'.')`*'. We laid it out that way for clarity.*)

By breaking the programming problem into two parts, each can be concentrated on without concern for the other. The *calculate_b* function can be developed without concern for where a and c come from. The *main* function can be developed without concern for exactly what *calculate_b* 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*.

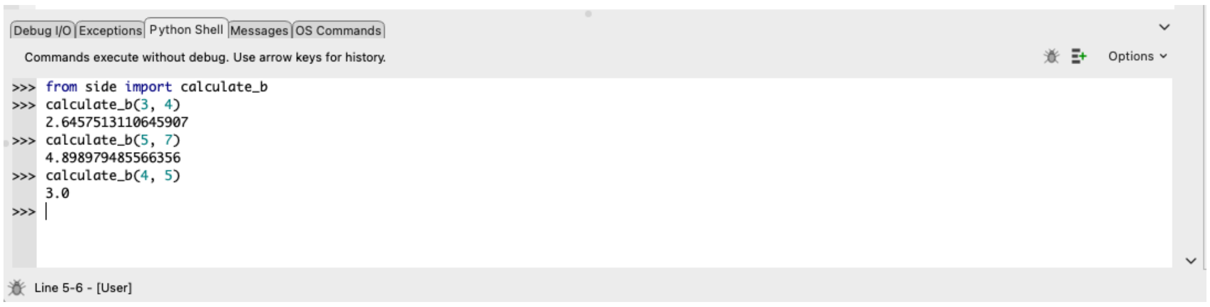
Admittedly, this programming problem is simple, and you probably solved it quite satisfactorily when working on assignment 2, however, it serves to convey the idea.

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 *calculate_b* 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 *calculate_b* e.g.

```
def main():
      print(calculate_b(3, 4))
      print(calculate_b(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:



Commands execute without debug. Use arrow keys for history.

```
>>> from side import calculate_b
>>> calculate_b(3, 4)
    2.6457513110645907
>>> calculate_b(5, 7)
    4.898979485566356
>>> calculate_b(4, 5)
    3.0
>>> |
```

Line 5-6 - [User]

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

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

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

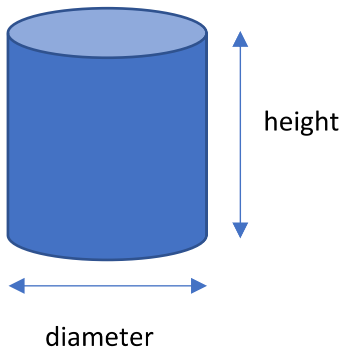
3. The result, 2.6457513110645907 is printed on the next line.

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.

Question 1 [10 marks]

Write a program called `cylinder.py` that calculates the volume of a cylinder given integer values for its diameter and height.



The area, *A*, of a circle of diameter *d* is $\frac{1}{4}\pi d^2$.

The volume of a cylinder of diameter *d* and height *h* is $A \times h$.

Your program should consist of three functions. One called '*circle_area*', another called '*cylinder_volume*', and the third called '*main*'.

Here is a program skeleton:

```
import math
def circle_area(diameter):
       # Your code here
def cylinder_volume(diameter, height):
       # Your code here
def main():
       # Your code here

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

The *circle_area* function has a diameter as a parameter. It will calculate and return the area of the circle with that diameter.

The *cylinder_volume* function has a diameter and a height as parameters. It will calculate and return the volume of the cylinder with that diameter and height. It will call the *circle_area* function to obtain a value for *A*.

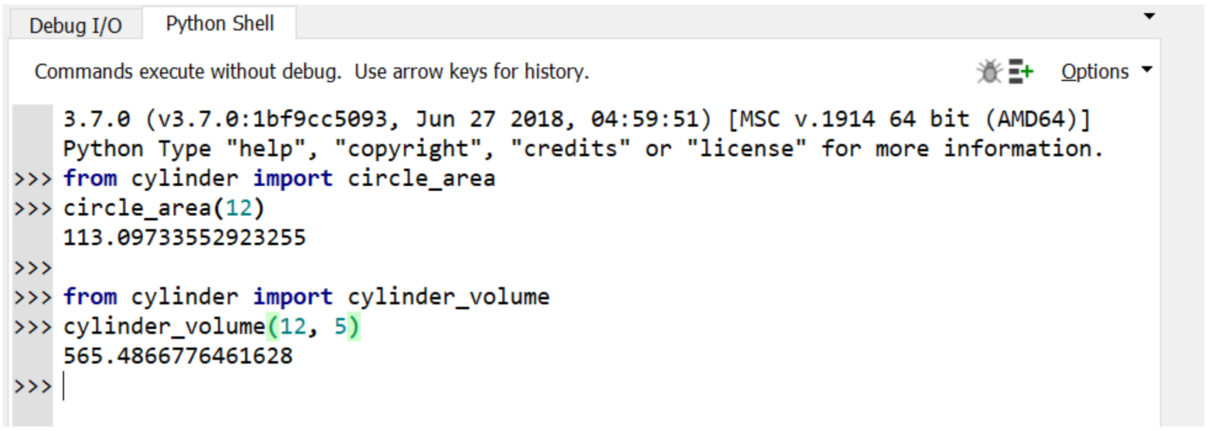
The *main* function will ask the user to input diameter and height and will call *cylinder_volume*, printing out the result.

Sample Input/Output:

```
Enter diameter:
10
Enter height:
5
The volume of the cylinder is 392.70
```

Output is formatted to two decimal places.

Using the Wing IDE Python shell, you should be able to test your functions as illustrated with the cylinder program example.



```
3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)]
Python Type "help", "copyright", "credits" or "license" for more information.
>>> from cylinder import circle_area
>>> circle_area(12)
113.09733552923255
>>>
>>> from cylinder import cylinder_volume
>>> cylinder_volume(12, 5)
565.4866776461628
>>>
```

NOTE that the automatic marker will test each of your functions individually. To enable this, your program MUST, as shown in the skeleton, have the following lines at the end:

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

The way in which functions are tested is like that illustrated with the Wing IDE Python shell (see assignment instructions). A trial consists of trying to execute a code snippet that uses the function under test. If a trial fails, typically, you will see the code snippet – an import statement followed by one or more function call expressions.

Question 2 [20 marks]

Computers often interface with the outside world using external sensors and actuators. For example, weather systems have sensors to detect the temperature and pressure. These sensors usually have limited computational ability, so they produce data in raw form and this data needs to be captured and processed by a program. On the Vula assignment page, you will find a skeleton for a program called 'extract.py'. The intent is that the program be used to extract useful data from a raw data stream obtained from a sensor.

The data from the sensor contains a block with the format:

```
… BEGIN temp_press:Xcoordinate,Ycoordinate emanetiS END …
```

The program must output the data in the following format:

```
Location: Sitename
Coordinates: Ycoordinate Xcoordinate
Temperature: temp C
Pressure: press hPa
```

Sample Input/Output:

```
Enter the raw data:
fbkjf kfjkdb fds BEGIN 12.2_1014:18.6E,34.0S NWOT EPAC END fdfdfds
Site information:
Location: Cape Town
Coordinates: 34.0S 18.6E
Temperature: 12.20 C
Pressure: 1014.00 hPa
```

To complete the program you must implement the *location*, *temperature*, *pressure*, *y_coordinate*, *x_coordinate*, and *get_block* functions. The functions have been identified by applying a divide-and-conquer strategy. Each solves a part of the overall problem:

- `get_block(raw_data)`
  Given a string of raw data as a parameter, the *get_block* function extracts the sub string starting with 'BEGIN' and ending with 'END'.
- `location(block)`
  Given a block string as a parameter, the *location* function returns the location component in title case.
- `pressure(block)`
  Given a block string as a parameter, the *pressure* function returns the pressure component as a real number value.
- `temperature(block)`
  Given a block string as a parameter, the *temperature* function returns the temperature component as a real number value.
- `y_coordinate(block)`
  Given a block string as a parameter, the *y_coordinate* function returns the y coordinate component as a string.
- `x_coordinate(block)`
  Given a block string as a parameter, the *x_coordinate* function returns the x coordinate component as a string.

Examples

- `get_block('fds BEGIN 12.2_1014:18.6E,34.0S NWOT EPAC END fdf fds ')` returns the block string `'BEGIN 12.2_1014:18.6E,34.0S NWOT EPAC END'`.
- `location('BEGIN 12.2_1014:18.6E,34.0S NWOT EPAC END')` returns `'Cape Town'` .
- `pressure('BEGIN 12.2_1014:18.6E,34.0S NWOT EPAC END')` returns 1014.0.
- `temperature('BEGIN 12.2_1014:18.6E,34.0S NWOT EPAC END')` returns 12.2.

- `y_coordinate('BEGIN 12.2_1014:18.6E,34.0S NWOT EPAC END')` returns `'34.0S'`.
- `x_coordinate('BEGIN 12.2_1014:18.6E,34.0S NWOT EPAC END')` returns `'18.6E'`.

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 3 [35 marks]

Mathematical functions map naturally to program functions and modules often are used to group such functions for reuse.

In the Gumatj* language, numbers use only the digits 0-4, such that instead of "tens", the second digit represents multiples of 5. Write a Python module with the following functions for simple Gumatj arithmetic, assuming that all values have at most 2 digits.

(*Reference: http://en.wikipedia.org/wiki/Quinary)

- `gumatj_to_decimal(a)`, that converts a Gumatj number to decimal
- `decimal_to_gumatj(a)`, that converts a decimal number to Gumatj
- `gumatj_add(a, b)`, that adds 2 Gumatj numbers
- `gumatj_multiply(a, b)`, that multiples 2 Gumatj numbers

Sample I/O:

```
Choose test:
d 12
calling function
called function
22
```

Sample I/O:

```
Choose test:
g 22
calling function
called function
12
```

Sample I/O:

```
Choose test:
a 12 14
calling function
```

```
called function
31
```

Sample I/O:

```
Choose test:
m 3 4
calling function
called function
22
```

Save your module as `gumatj.py`. The main program has been supplied as `base5.py` - use this to test your program and do not change this file.

END