

CSC1015F Assignment 5: Functions and Strings

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 3 asks you to write a program called `triangle.py` to calculate the area of a triangle given the length of its sides. Here is a solution that demonstrates the use of functions:

```
# triangle.py
import math

def triangle_area(a, b, c):
    s=(a+b+c)/2
    return math.sqrt(s*(s-a)*(s-b)*(s-c))

def main():
    side_a = int(input('Enter the length of the first side: '))
    side_b = int(input('Enter the length of the second side: '))
    side_c = int(input('Enter the length of the third side: '))

    if side_a+side_b<side_c or side_b+side_c<side_a \
       or side_a+side_c<side_b:
        print('Sorry, those numbers don\'t describe a triangle.')
    else:
        print('The area of the triangle with sides of length ', end='')
        print(side_a, 'and', side_b, 'and', side_c, 'is ', end='')
        result = triangle_area(side_a, side_b, side_c)
        print(str(result)+'.')

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

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

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

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

- It calls *triangle_area*, passing the integer values of *a*, *b* and *c*.
- The *triangle_area* function uses the values to calculate and return the value of the area.
- 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(triangle_area(side_a, side_b, 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 *triangle_area* function can be developed without concern for where *a*, *b* and *c* come from. The *main* function can be developed without concern for exactly what *triangle_area* will do. It's enough to know simply that it requires three values (the sides *a*, *b* and *c*) and that it will calculate the area.

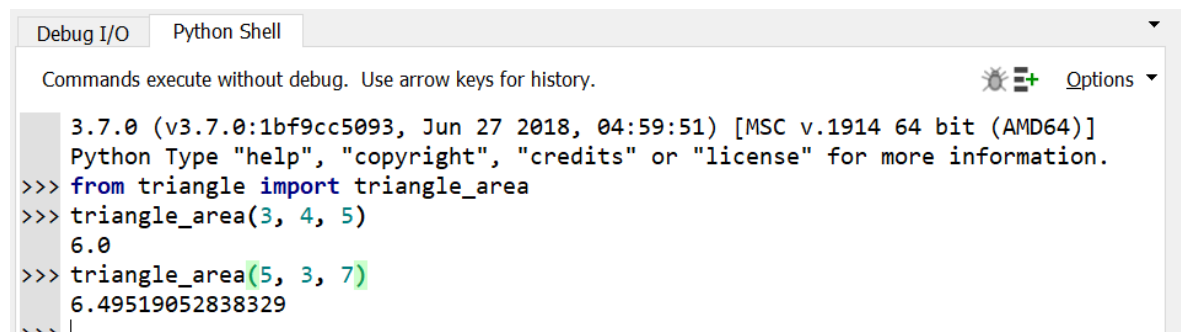
Admittedly, this programming problem is simple, and you probably solved it quite satisfactorily when working on assignment 2, however, it serves to convey the idea.

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 triangle program and you chose to concentrate on the *triangle_area* 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 *triangle_area* e.g.

```
def main():
    print(triangle_area(3, 4, 5))
    print(triangle_area(5, 2, 3))
    # ...
```

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:

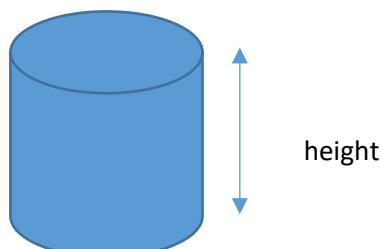


The screenshot illustrates use. Lines entered by the user have a prompt, '>>>' beside them. Lines without are responses from the Python shell.

1. The user enters an import statement for the *triangle_area* function.
2. The user then enters a function call expression, calling *triangle_area* with the values 3, 4 and 5 for *a*, *b* and *c*.
3. The result, 6.0 is printed on the next line.
4. The user enters another function call expression, this time with the values 5, 3, 7,
5. And the result, 6.49519052838329 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 kfkdb fds BEGIN 12.2_1014:18.6E,34.0S NWOT EPAC END fddfds
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]

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.

Sample Input/Output:

```
Enter month:
April
Enter year:
2020
April
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
```

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 used Gauss's formula in assignment 3 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:

$$\left(d + \left\lfloor \frac{13(m+1)}{5} \right\rfloor + y + \left\lfloor \frac{y}{4} \right\rfloor - \left\lfloor \frac{y}{100} \right\rfloor + \left\lfloor \frac{y}{400} \right\rfloor \right) \text{ modulus } 7$$

The result, h , is a number 0 to 6 where 0 is Saturday, 1 is Sunday, ..., and 6 is Friday.

The formula $((h + 5) \text{ modulus } 7) + 1$ 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()
```

(If you enjoy this question, you might be interested in trying the challenge question at the end of this document.)

Question 4 [35 marks]

This question concerns constructing a Python module. A module is typically a collection of related functions and can also be used to support a divide-and-conquer approach to problem solving.

Modules are also often used to support reuse. You may have observed that some of the functions required for the previous question could easily be used in different circumstances to solve other calendar-related programming problems i.e. they're eminently reusable. It would make sense to collect them in a calendar utility module.

Write a Python module called `piglatin.py` that contains functions for translating sentences between English and a variant of Pig Latin (see: http://en.wikipedia.org/wiki/Pig_Latin).

To convert from English to Pig Latin, each word must be transformed as follows:

- if the word begins with a vowel, 'way' should be appended (example: 'apple' becomes 'appleway')

- if the word begins with a sequence of consonants, this sequence should be moved to the end, prefixed with 'a' and followed by 'ay' (example: 'please' becomes 'easeaplay')

NB: Assume, when reverting Pig Latin to English that the original English text does not contain the letter "w".

The Python module will contain the following functions:

- `to_pig_latin(sentence)`
Return the Pig Latin sentence for a given English sentence.
- `to_english(sentence)`
Return the English sentence for a given Pig Latin sentence.

A main program called 'plmodtest.py' has been provided. Use this to test your program. (Note plmodtest must not be modified.)

Sample I/O:

```
(E)nglish or (P)ig Latin?  
E  
Enter an English sentence:  
the quick black fox jumps over the lazy apple  
Pig-Latin:  
eathay uickaay ackablay oxafay umpsajay overway eathay azyalay  
appleway
```

Sample I/O:

```
(E)nglish or (P)ig Latin?  
P  
Enter a Pig Latin sentence:  
eathay uickaay ackablay oxafay umpsajay overway eathay azyalay  
appleway  
English:  
the quick black fox jumps over the lazy apple
```

Submission

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

Challenge

No marks, but given your solution to question 3, can you develop a program called `calendar.py` that accepts a year as input and prints the calendar for that year?

Sample I/O:

Enter the year:

2015

2015

January							February							March						
Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su
			1	2	3	4							1							1
5	6	7	8	9	10	11	2	3	4	5	6	7	8	2	3	4	5	6	7	8
12	13	14	15	16	17	18	9	10	11	12	13	14	15	9	10	11	12	13	14	15
19	20	21	22	23	24	25	16	17	18	19	20	21	22	16	17	18	19	20	21	22
26	27	28	29	30	31		23	24	25	26	27	28		23	24	25	26	27	28	29
														30	31					

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

July							August							September							
Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	
			1	2	3	4	5						1	2		1	2	3	4	5	6
6	7	8	9	10	11	12	3	4	5	6	7	8	9	7	8	9	10	11	12	13	
13	14	15	16	17	18	19	10	11	12	13	14	15	16	14	15	16	17	18	19	20	
20	21	22	23	24	25	26	17	18	19	20	21	22	23	21	22	23	24	25	26	27	
27	28	29	30	31			24	25	26	27	28	29	30	28	29	30					
							31														

October							November							December						
Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su
			1	2	3	4							1		1	2	3	4	5	6
5	6	7	8	9	10	11	2	3	4	5	6	7	8	7	8	9	10	11	12	13
12	13	14	15	16	17	18	9	10	11	12	13	14	15	14	15	16	17	18	19	20
19	20	21	22	23	24	25	16	17	18	19	20	21	22	21	22	23	24	25	26	27
26	27	28	29	30	31		23	24	25	26	27	28	29	28	29	30	31			
							30													

END

CONTINUED