Year 8 Investigation - Mathematical Modelling and Congruent Triangles

September 9, 2022

1 INTRODUCTION

The main goal of this investigation is for you to think carefully and design a set of algorithms (which altogether make a program) that will allow a computer to determine whether or not two triangles are congruent based on lengths and angles entered by the user.

If the triangles are congruent, your program should indicate which congruence test was successful, ideally with the three equality statements that allowed it to reach that conclusion.

After you have thought carefully and designed your algorithms, you are required to implement them in code using the Python programming language. You can use Grok Academy's Python Playground to write your code (keep in mind that Python Playground 11 will be reserved for submission).

If you are familiar with PyCharm EDU or VS Code then you are welcome to use one of these to develop your program (PyCharm EDU is highly recommended). Email isaac.kigodi@education.wa.edu.au or come to the Maths office if you need help with this.

The details of what your program needs to do will be given below, together with the number of marks allocated out of 100. Additional informal guidelines can be found by typing (then running) `import this` into Grok Academy.

2 OBTAIN INPUT AND VALIDATE (30 MARKS TOTAL)

2.1 Requirement: Use the correct input format (5 marks)

Your Python program should prompt the user to provide lengths and angles of two triangles as shown below (we will ignore the units). We will refer to these two triangles as Triangle ABC and Triangle PQR.

```
Enter lengths AB, AC and BC in that order, separated by spaces: 5 5 5
Enter angles BAC, ABC and ACB in that order, separated by spaces: 60 60 60
Enter lengths PQ, PR and QR in that order, separated by spaces: 5 5 5
Enter angles QPR, PQR and PRQ in that order, separated by spaces: 60 60 60
```

See the tutorial on Mathematical Modelling on Connect Notices for examples of how to accept multiple input values in one line.

2.2 Requirement: Check for validity of inputs (10 marks total)

2.2.1 Validating Lengths (3 marks)

Your program should check whether or not the information entered will result in a valid triangle.

For instance, if the user enters lengths as 2 5 9 for triangle ABC, your program should respond as follows:

```
Enter lengths AB, AC and BC in that order, separated by spaces:  2 5 9
ERROR: Those lengths will not result in a valid triangle, try again.

Enter lengths AB, AC and BC in that order, separated by spaces: _
```

Your program should repeat this until the user provides valid inputs.

2.2.2 Validating Angles (7 marks)

Once the user enters lengths that will result in a valid triangle, your program should proceed to ask about angles. Once again, if the angles given will result in an invalid triangle, the program should respond accordingly (think about why this is worth more marks):

```
Enter angles BAC, ABC and ACB in that order, separated by spaces: 60 60 70
ERROR: Those angles will not result in a valid triangle, try again

Enter angles BAC, ABC and ACB in that order, separated by spaces: _
```

Once gain, your program should repeat this until the user provides valid inputs.

2.3 Requirement: Use Functions for Validity Checks (5 marks)

Since checking for valid inputs may need to be done repeatedly, use functions to accomplish this.

A simple example of an input validation algorithm is provided below - make sure to review the Python Functions tutorial on Connect Notices for more details.

```
def check_value(input_value):
    '''Accepts an input_value and checks if it is between 4 and 8 inclusive'''
    if float(input_value) >= 4 and float(input_value) <= 8:
        validity = 'Valid Input'
    else:
        validity = 'Invalid Input'
    return validity

input_value = input("Enter a value between 4 and 8 inclusive: ")
result = check_value(input_value)
while result == 'Invalid Input':
    input_value = input("Enter a value between 4 and 8 inclusive: ")
    result = check_value(input_value)

# if we are here, the input was valid
print(f"{input_value} is a {result}")
```

2.4 Requirement: Allow for missing sides and angles (10 marks)

Your code should allow for partially available information. For example, the user might only have angle BAC and sides AB and AC. In this case, the user should type NA for the missing pieces of information.

```
Enter sides AB, AC and BC in that order, separated by spaces: 3 NA 5
```

```
Enter angles BAC, ABC and ACB in that order, separated by spaces: 60 NA NA
```

Similarly, the user might only have sides PQ and PR and angle QPR

```
Enter sides PQ, PR and QR in that order, separated by spaces: 3 NA 5
```

```
Enter angles QPR, PQR and PRQ in that order, separated by spaces: 60 NA NA
```

Notice that even with missing information, your code should conclude that the two triangles are SAS Congruent.

3 USE DATA TO BUILD MODELS (10 MARKS TOTAL)

3.1 Requirement: Build a model of each triangle using provided data (7 marks)

Once you have received inputs that constitute a valid triangle, you should package the information in one place, and in a way that makes it easy to capture relationships between angles and sides. We will call this package of information a 'triangle model'.

Dictionaries are able to do a good job of packaging information and thus building models.

Think about your model carefully. If it captures detailed information about the triangle, your coding task will be greatly simplified because you will be able to simply query your triangle model for the information you need.

For instance, you would be able to 'ask' your triangle to give you the length of a particular side, or to give you a side that is opposite a particular angle.

Experiment with the code below to get a better understanding. Notice how there are a lot of comments. You should do the same with your code.

```
# ...
# ...
# ...
# ...
# ...
#
# below is a simple model of a triangle done in two parts,
# angles and sides are kept separate to keep things simple,
# you can  add more parts (dictionaries) as needed in order to ...
# ... capture richer information about a particular triangle.


# for the model below, notice how the values are stored as strings
```

```
# # # # #  START OF TRIANGLE_ABC MODEL

# this dictionary contains the sides of triangle ABC
triangle_ABC_sides = {
    'AB':'3',
    'AC':'5',
    'BC':'NA',
}

# this dictionary contains the angles of triangle ABC
triangle_ABC_angles = {
    'BAC':'NA',
    'ABC':'90',
    'ACB':'60',
}

# # # # #  TRIANGLE ABC_MODEL ABOVE

# Hint: Think about how you would capture information about...
# what side is opposite which angle by creating a dictionary called
# triangle_ABC_opposites = { ... }


# this is how to access a side and an angle
print(triangle_ABC_sides['AB'])
print(triangle_ABC_angles['ABC'])


# here is how to update the value stored for side BC
triangle_ABC_sides['BC'] = '40'


# we can use variables as well
angle = 'ACB'
print(triangle_ABC_angles[angle])


# Here is how to get input from a user in order to update the stored value
angle_name = input("Which angle would you like to update? ")
new_value = input("What value would you like to store for the angle? ")

angle_name = angle_name.upper() # the names of sides and angles are stored as upper case

# store the new value
triangle_ABC_angles[angle_name] = new_value

print(angle_name, value)
print(triangle_ABC_angles)
```

3.2 Optional Requirement: Build a more compact triangle model (3 marks)

The code below is a more compact way to build the above model. It is completely optional for you to do things this way, although you will earn 3 marks if you do so.

```
# this model of a triangle uses dictionaries inside a dictionary
triangle_ABC = {
    'name'  :  'ABC',
    'sides' :  {'AB': '3', 'AC': '5', 'BC': 'na'},
    'angles':  {'BAC': '30', 'ABC': '90', 'ACB': '60'},
}

# hint: You can add another row for relationships between opposites
# so that you can tell which side is opposite what angle

# let's see how our model of the triangle is stored by Python
print(triangle_ABC)

# this is how to access the triangle name
print(triangle_ABC['name'])

# first access sides, then access a side within the sides dictionary
print(triangle_ABC['sides']['AC'])

# do the same for angles
print(triangle_ABC['angles']['ACB'])

# here is how to update the value stored in side BC
triangle_ABC['sides']['BC'] = '40'

# we can use variables as well
query = 'angles'
angle = 'ACB'
print(triangle_ABC[query][angle])

# Here is how to get input from a user in order to update the stored value
what_to_update = input("Would you like to update angles or sides?: ")
its_name = input("Which angle or side would you like to update? ")
new_value = input("What value would you like to store? ")

its_name = its_name.upper() # the names of sides and angles are stored as upper case

triangle_ABC[what_to_update][its_name] = new_value
print(what_to_update, its_name, new_value)
print(triangle_ABC)
```

(Advanced: It would be even better to use classes and objects for modelling - feel free to use those concepts in this investigation if you are familiar with Object Oriented Programming).

4 PROCESS THE STORED DATA (25 MARKS TOTAL)

4.1 Optional Requirement: Calculate and store missing data (15 marks)

If the input data for any of the triangles contains two out of the three angles, calculate and store the third angle into your model (5 marks).

If the triangle is right-angled, calculate and store the unknown side. This side may be the hypotenuse (5 marks), or a side adjacent to the right angle (5 marks).

In order to earn the 15 marks in this section, your program needs to make use of this computed information when determining whether or not the triangles are congruent.

4.2 Requirement: Make your code modular using functions (5 marks)

Whether or not you have filled in the missing angles and/or sides for the two triangles, you can process the information to determine if the two triangles are congruent. Make sure to specify all the congruence test(s) that are successful for a given triangle pair.

It is important that you think carefully about your algorithms and make extensive use of appropriately named functions, each of which should perform a specific task. See section 2.3 for how to properly comment on what a function does.

You should use functions such as those given below and pass them the information they need to process. The functions can return a result to the caller if necessary. The list below is just a suggestion - you can use more functions as needed and organise them as you see fit.

```
validate_lengths(...)

validate_angles(...)

compute_unknown_angle(...) # these two ...
compute_unknown_side(...)  # ... are optional

test_for_SSS(...)

test_for_SAS(...)

test_for_ASA(...)

test_for_RHS(...)
```

Remember to study the tutorial on functions in detail for this part. You can find this on Connect under Notices.

4.3 Requirement: Use code comments extensively (5 marks)

Make sure that every function has a description of what it does as shown above in section 2.3 (within three single quote marks just below the def keyword)

Also make sure you use # comments throughout your code. Comments are like sunscreen, if in doubt, use lots and lots.

5 PROVIDE CORRECT OUTPUT (35 MARKS TOTAL)

5.1 Requirement: State the successful congruence tests (25 marks)

- Correctly state if the triangles are SSS Congruent (5 marks)
- Correctly state if the triangles are SAS Congruent (6 marks)
- Correctly state if the triangles are ASA Congruent (6 marks)
- Correctly state if the triangles are RHS Congruent (8 marks)

For example:

```
The triangles are SAS Congruent
The triangles are SSS Congruent
```

You are required to provide a statement for each congruence test that the triangles pass.

5.2 Requirement: Provide details to support congruence statements (10 marks)

In addition to simply stating the tests for congruence that have been successful, you can provide the three equality statements that allowed you to reach the conclusion (example output shown below).

- Give three statements to support the SSS Congruence statement (1 mark)
- Give three statements to support the SAS Congruence statement (3 marks)
- Give three statements to support the ASA Congruence statement (3 marks)
- Give three statements to support the RHS Congruence statement (3 marks)

For example:

```
The triangles are SAS Congruent:
BAC = QPR
AB = PQ
AC = PR

The triangles are SSS Congruent:
AB = PQ
AC = PR
BC = QR
```

You are required to provide a set of statements to support each successful congruence test.

6 SUBMIT YOUR PROGRAM

Your code should be saved in Grok Academy's Python Playground 11 by 21 October 2022.

During week 3, you will do a 5-minute presentation to a small group with teacher supervision. You will need to demonstrate what your program does and how it works. You will also describe the computational and algorithmic thinking you went through in the process of writing your code.

Email isaac.kigodi@education.wa.edu.au if you need assistance, and remember to `import this`.