Python Challenge: Triangle Calculator

User Specification:

I want to create a program which will allow me to calculate unknown side lengths and angles for triangles. The functions you make should be clearly named, easy and intuitive to use, and should allow me to solve missing side or angle problems for any triangle provided I input valid data. They should also alert me when I enter invalid or incompatible data such as two obtuse angles or sides of 3, 4 and 50.

Getting started:

When you begin designing any program, the first step should always be to think carefully about the specification. You may be the one who wrote the spec, or it may have been provided for you. In either case, getting clarity on exactly what the program will be used for, and how it should run, is vital.

Start with a basic diagram, planning the outline of the program first, and then start to fill in the details:

Input	Processing	Output
What data will your program receive, and in what form? What kind of inputs would you expect any key functions to have, and how will you test to make sure they are valid?	What functions are required by the specification? What additional 'helper' functions might be useful to create? Which values feed into what functions? How will you name your functions to ensure clarity for the user? Are there any third-party modules that might be helpful to import?	What form should the final result take? What kind of outputs would you expect your functions to provide, and how will you communicate any errors that arise along the way?

Tip on testing: You can use GeoGebra to draw triangles, and measure side lengths, angles or area directly.

Turn over for some details on the **math** module that you may find helpful in your project.

The math module:
Most of the basic calculation functions are essential to making any program work, and quite a few commonly used operators are built into Python (including +, -, *, /, **, // and %). However, if you find yourself needing more specialised mathematical functions, such as you might find on a scientific calculator, it is worth checking out the **math** module.

`from math import *`
 	
`print(sin(pi/2))`
`print(sqrt(2))`
`print(degrees(pi/2))`
`print(radians(30))`
`print(atan(sqrt(3)))`
`print(degrees(atan(sqrt(3))))`
	This line tells Python to import all functions and values from the **math** module directly, so that we can call them without using the prefix **math.**
	For instance, if we use **import math**, every time we want π, we would write **math.pi**, but if we use **from math import ***, Python knows what we mean if we just write **pi**.

Hopefully by trying out the code snippet above you have noticed that the default angle unit for the **math** module is radians, not degrees. Fortunately it includes the **degrees** and **radians** functions for converting between them, but if you find you're getting unexpected answers, this may well be the reason.

Notice that the inverse trig functions, commonly represented as $\sin^{-1} x$ on a calculator, appear as **asin**, **acos** and **atan**, which are short for $\arcsin x$, $\arccos x$ and $\arctan x$ respectively, the official names for the functions which produce an *angle* from a given *ratio*.

A note on naming:
When you name functions, try to be as clear as possible. A long name isn't necessarily a bad thing – the key is to be descriptive. Try to consider when it would be used, and how it would read within a piece of code. For instance, if I name a function **sine_rule**, it's not clear whether I'm trying to find an unknown side or an unknown angle. **sine_rule_find_side** would work much better. You'll know if you've named a function clearly enough because the user will know just by reading its name what it is intended to do, and they could probably even guess what input values (arguments) the function may require.

String literals:
A good habit to get into when designing robust code is to include a brief description of your function. Python has a standard way to do this, which is almost like writing a comment, but has the advantage of showing up in help files or when a user starts using your function:

```
def triangle_area(a, b, C):
    """Returns the area of the triangle with side lengths a and b and subtended angle C"""
    if a >= 0 and b >= 0 and C >= 0:
        ret
    else:
        ret

area = triangle_area(
```


Just enter your description between triple speech-marks on the first line of your function.

Example code:

Throughout this code, I have used a function return value of -1 whenever the input values are incompatible.

```
"""A set of functions for finding unknown lengths, angles and area for triangles"""
from math import * # all functions and constants within the math module are imported directly
                   # so we can use things like sqrt(sin(pi/2)) or degrees(acos(x))

def triangle_area(a, b, C):
    """Returns the area of the triangle with side lengths a and b and subtended angle C"""
    if a >= 0 and b >= 0 and C <= 180 and C >= 0:
        return 0.5 * a * b * sin(radians(C))
    else:
        return -1

def sine_rule_find_angle(a, A, b):
    """Returns the unknown angle opposite side b, given two known values: side a and opposite
angle A"""
    if a > 0 and b >= 0 and A <= 180 and A >= 0:
        return degrees(asin(sin(radians(A)) * b / a))
    else:
        return -1

def sine_rule_find_length(a, A, B):
    """Returns the unknown side length opposite angle B, given two known values: side a and
opposite angle A"""
    if A > 0 and A < 180 and B <= 180 and B >= 0 and a >= 0 and A + B <= 180:
        return a * sin(radians(B)) / sin(radians(A))
    else:
        return -1

def cosine_rule_find_length(a, b, C):
    """Returns the third side length, c, opposite angle C, given C and the other two sides"""
    if a >= 0 and b >= 0 and C >= 0 and C <= 180:
        c_squared = a**2 + b**2 - 2 * a * b * cos(radians(C))
        if c_squared > 0:
            return sqrt(c_squared)
        else:
            return -1
    else:
        return -1

def cosine_rule_find_angle(a, b, c):
    """Returns angle C, opposite side c and between sides a and b"""
    if a > 0 and b > 0 and c >= 0 and max(a, b, c) * 2 < (a + b + c):
        ratio = (a**2 + b**2 - c**2) / (2 * a * b)
        if abs(ratio) <= 1:
            return degrees(acos(ratio))
        else:
            return -1
    else:
        return -1

def angle_sum(A=None, B=None, C=None):
    """Returns a tuple of the three angles A, B and C, finding a missing one if possible"""
    if B != None and C != None:
        A = 180 - (B + C)
    elif A != None and C != None:
        B = 180 - (A + C)
    elif A != None and B != None:
        C = 180 - (A + B)
    return (A, B, C)
```

Extension idea:

The last function, **angle_sum**, is an example of a function with *optional keyword arguments* which have default values. If no value is provided for any of A, B and C, the code will treat them as having value **None**, and adapt accordingly. In this case, if it finds two known angles, it will calculate the third. Therefore the return tuple may still contain some **None** values, but whenever it is mathematically possible, it will return three known values for A, B and C. By making similar functions for sine rule and cosine rule, you might be able to create an all-purpose solver: give it any known values of a, b, c, A, B and C and it does the rest.