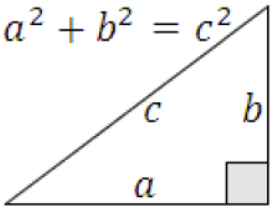
Pythagorean Triples

The problem of finding integers which satisfy certain equations is one for which computers are uniquely well suited. Testing multiple examples and carrying out routine and repetitive tasks is something humans are notoriously bad at – we are prone to make errors, especially when tired, distracted or bored, and we are really slow compared to computers. Humans are much better at creative endeavours, including coming up with interesting problems to solve, solving complex problems, or designing an algorithm - a well-defined procedure - for solving a problem (which can then be out-sourced to a computer to solve).

Pythagoras' Theorem:



Pythagorean triple:

A set of three integers which satisfy the theorem.

A triangle with these side lengths would be right-angled.

Eg: 3, 4, 5

$$a^2 + b^2 = c^2 \qquad 3^2 + 4^2 = 5^2$$

The 3, 4, 5 triangle is the classic example. It is also the smallest possible triangle which satisfies the criteria: right angled, and with integer side lengths. *Can you find another?*

If you're feeling lazy (that is, mathematical) your triple may have simply been a scaled version of the 3, 4, 5 triple. 6, 8, 10 works, since 36+64=100, and so does 30, 40, 50. These are perfectly valid answers, but they all form one 'family', with founder and archetypical triple 3, 4, 5. A triple like 5, 12, 13 is fundamentally different, since it is not simply an enlarged version of the same triangle. Triples like this, where the numbers share no factors in common (other than 1), are known as **primitive triples**. *Can you find any?*

Clarifying the search process

Before sitting down to write code, it pays to think carefully about exactly what you want the computer to do. Writing a program is equivalent to giving instructions to a very pedantic literal-minded person who has no common sense. Someone who, if you ask them to add up two numbers for you, will happily do so, but won't think to tell you the answer unless one of your instructions to them was "… and tell me what you get". Someone who, if you tell them to repeatedly follow a particular set of instructions but don't give them an exit strategy, will dutifully continue to follow that list of instructions forever.

To find the Pythagorean triples up to, say, 100, we could follow these instructions:

- Consider each pair of numbers between 1 and 100.
- Square them, add them together and try square-rooting.
- If you get a whole number, make a note of your two numbers. If not, move on.
- Check the next possible pair.

Turn over to see how to code it up …

Finding Pythagorean Triples *with Python*

Programming Concept: **loop**

```
1    for a in range(1, 20):
2      print(a)
```

In Python, *range(1,20)* gives integer values from 1 up to *but not including* 20. So this code should print out the values from 1 up to 19.

Programming Concept: **nested loop**

```
1    for a in range(1, 10):
2      for b in range(1, 10):
3        print(f"a = {a} and b = {b}")
```

We can nest a loop within another loop. This code will set a equal to 1, and set b equal to 1, then print:

a = 1 and b = 1

Once the main body of the inner code is completed, the value of b is incremented by 1, so the next line is:

a = 1 and b = 2

This will go on until… **a = 1 and b = 9**

Then the inner loop completes, and at that point a is incremented, giving:

a = 2 and b = 1

And this continues till… **a = 9 and b = 9**

Additional Programming Concept: ***f-string***

Python can print things (that is, display their value as a text string to the screen), but it does so by converting to a text string data type. We can use the f-string notation to embed values inside the so-called 'replacement field' with curly braces { } inside a text string.

Programming Concept: **assignment**

```
c = (a**2 + b**2)**0.5
```

We can make a new variable by simply writing its name followed by =. This 'assigns' the value to the right of the = sign to the new variable you write to the left.

In this example, I'm using the fact that x**y is equivalent to x^y. Python is squaring a and b, then square-rooting their sum. The result is stored as a number called c.

Programming Concept: **comparison**

```
if c == int(c):
  print("whole number!")
```

When a single = sign is used, it is treated as an assignment, as above. To see *whether or not* two things are equal, we use ==. This compares the values of c (the length of the hypotenuse) and int(c), which converts the number to an integer (rounding down if needed).

Programming Concept: **conditionals (if)**

```
if c == int(c):
  print("whole number!")
```

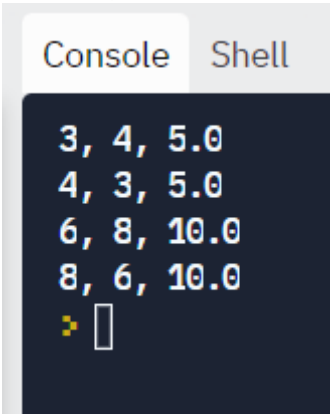
If the 'condition' written after **if** is *true*, the following block of code is run. If not, not.

Putting it all together

```
1   for a in range(1, 10):
2     for b in range(1, 10):
3       c = (a**2 + b**2)**0.5
4       if c == int(c):
5         print(f"{a}, {b}, {c}")
```

This loops through all the possible pairs of values a and b between 1, 1 and 9, 9. For each combination, the hypotenuse c is calculated, and then if c is a whole number Python prints out the values of a, b and c.

The output



The output from the code above should appear to the right of your program, in the 'console'.

A few things you might notice:

- Because our code considers 3, 4 separately to 4, 3, it's given us the 3, 4, 5 triple twice (and the 6, 8, 10 triple twice).
- Because c is calculated using square-roots, it is stored as a **float** rather than an **int** (a floating point number is stored differently to integers because Python may need to take into account digits after the decimal point).

Extending / improving the code

There are some easy changes we can make straight away. The most obvious is to find more triples by changing the range used for a and b:

```
1   for a in range(1, 100):
2     for b in range(1, 100):
```

We could also deal with the duplication issue by making the inner loop start from a instead of from 1. That way, b will never be lower than a:

```
1   for a in range(1, 100):
2     for b in range(a, 100):
```

And we can force c to be an integer type, too:

```
1   for a in range(1, 100):
2     for b in range(a, 100):
3       c = (a**2 + b**2)**0.5
4       if c == int(c):
5         c = int(c)
6         print(f"{a}, {b}, {c}")
```

Next steps

Further improvements we could make include:

- Instead of just displaying on screen, store results in a list, or write to a text file.
- Find a way to check for common factors, and only return primitive triples.
- Find Pythagorean 'quadruples': integer length width and height for a cuboid such that the diagonal is also an integer.

Bonus: examples of modifications you can make to your code

Using a list to store values

```
1   results = []
2
3   for a in range(1, 100):
4     for b in range(a, 100):
5       c = (a**2 + b**2)**0.5
6       if c == int(c):
7         c = int(c)
8         results.append([a, b, c])
9
10  for r in results:
11    print(r)
12
13  print(f"Found {len(results)} triples")
```

We start by defining a new variable which is of type **list**, instead of an **int** (integer) or **float** (floating point number) or **str** (text string).

We use the *list method* **append** to add things to the list. You can put any objects into a Python list, including (as in this case) other lists. [a, b, c] is a list containing the values of a, b and c.

Notice that for loops can be more flexible than just looping through values. The loop on line 10 cycles through all the entries found in the results list.
The replacement field on the final line contains len(results), which gives the length of the results list (that is, the number of elements within the list).

Writing to a text file:

```
with open("results.txt", "w") as out_file:
  for r in results:
    out_file.write(f"{r}\n")
```

The **with** block takes care of opening (and if necessary creating) a document, then saving and closing when done.

We loop through the list of results, and for each one we write a line into the text file. Notice that we include \n, which is the 'new line' character: this is how Python knows to 'press enter' after writing the result.

Pythagorean Quadruples:

```
for a in range(1, 100):
  for b in range(a, 100):
    for c in range(b, 100):
      d = (a**2 + b**2 + c**2)**0.5
      if d == int(d):
        d = int(d)
        results.append([a, b, c, d])
```

This simply extends the idea to a third nested loop, so we can test all possible sets of 3 integers from 1 to 99.
Note that this program will test around a million sets of numbers, since there are about 100 possible values for c for each value of b, and 100 possible values of b for each value of a.

Challenge:

Can you find a way to determine whether a given Pythagorean triple is primitive? You may need to think about methods for finding the highest common factor (sometimes known as the greatest common divisor) of your numbers.