Pythagorean Triples: a Python Comprehensions Exercise

This doc was produced as an exercise in Python programming. It's based on a similar exercise in Fortress (https://en.wikipedia.org/wiki/Fortress_(programming_language)) programming: Comprehending List Comprehensions (http://hellers.ws/images/geek/sun/ComprehendingComprehensions.fortress.pdf). Starting with a simple specification, a sequence of code snippets are presented, each an optimization of the previous one, each reproducing an identical result. The original Fortress note was intended as a challenge for compiler writers to perform the same sorts of optimiztions automatically.

A Pythagorean triple is three integers a, b, and c representing the sides of a right triangle with hypotenuse c. This means that $a^2 + b^2 = c^2$, which we'll refer to as the Pythagorean equality. We will require that all three sides are less than some specified maximum value m, and for uniqueness we'll also require that $a < b < c \leq m$. Also, triples should be primitive — not a multiple of another triple. Much more information on Pythagorean Triples can be found in this Wikipedia article (https://en.wikipedia.org/wiki/Pythagorean_triple).

In [295]:
```
# Define two helper functions, coprime and even, and a trip object.
import math
import functools

def even(x): return 0 == x%2
print([even(i) for i in range(-3,5)])

def coprime (x,y): return 1 == math.gcd (x,y)
print ([coprime(3,i) for i in range(-1,10)])

class Trip():
    def __init__(self,a,b,c):
        self.a, self.b, self.c = a, b, c
        assert self.c**2 == self.a**2 + self.b**2, f"Bad Trip: <{self.a}
{self.b} {self.c}>"

    def __eq__(self,t): return self.a == t.a and self.b == t.b and self.
c == t.c

    def __repr__(self): return f"Trip: <{self.a} {self.b} {self.c}>"

print(Trip(3,4,5))
Trip(3,4,6) #this invalid trip should generate an AssertionError
```

```
[False, True, False, True, False, True, False, True]
[True, False, True, True, False, True, True, False, True, True, False]
Trip: <3 4 5>

-----------------------------------------------------------------------
----
AssertionError                          Traceback (most recent call l
ast)
<ipython-input-295-de45b4d5c0c9> in <module>
     19
     20 print(Trip(3,4,5))
---> 21 Trip(3,4,6) #this invalid trip should generate an AssertionErro
r

<ipython-input-295-de45b4d5c0c9> in __init__(self, a, b, c)
     12     def __init__(self,a,b,c):
     13         self.a, self.b, self.c = a, b, c
---> 14         assert self.c**2 == self.a**2 + self.b**2, f"Bad Trip:
 <{self.a} {self.b} {self.c}>"
     15
     16     def __eq__(self,t): return self.a == t.a and self.b == t.b
and self.c == t.c

AssertionError: Bad Trip: <3 4 6>
```

In [296]:
```
# We are looking to generate lists like this one, from Wikipedia:
Wikipedia100 = [Trip( 3, 4, 5), Trip( 5,12,13), Trip( 7,24,25), Trip( 8,
15,17),
                Trip( 9,40,41), Trip(11,60,61), Trip(12,35,37), Trip(13,
84,85),
                Trip(16,63,65), Trip(20,21,29), Trip(28,45,53), Trip(33,
56,65),
                Trip(36,77,85), Trip(39,80,89), Trip(48,55,73), Trip(65,
72,97)
                ]

print(Wikipedia100)
```

```
[Trip: <3 4 5>, Trip: <5 12 13>, Trip: <7 24 25>, Trip: <8 15 17>, Tri
p: <9 40 41>, Trip: <11 60 61>, Trip: <12 35 37>, Trip: <13 84 85>, Tri
p: <16 63 65>, Trip: <20 21 29>, Trip: <28 45 53>, Trip: <33 56 65>, Tr
ip: <36 77 85>, Trip: <39 80 89>, Trip: <48 55 73>, Trip: <65 72 97>]
```

In [297]:
```
#set the max leg size
m = 100

def check_trips (trips):
    """Utility to check we match the Wikipedia list of triples"""
    assert(all ([w == t for w,t in zip (Wikipedia100, trips)])), "Trips
 don't match: " + print (trips)
    print("Good Trips")

#Start with a direct enumerate-and-filter specification, parametrized by
m, the maximum side length.
check_trips([Trip(a,b,int(c))
             for a in range(1,m+1)
             for b in range(1,m+1)
             for c in range(1,m+1)
             if a < b < c
             and a**2 + b**2 == c**2
             and coprime(a,b)
            ])
```

```
Good Trips
```

The expression: `for a in range(1,m+1) for b in range(1,m+1) for c in range(1,m+1)` is like a triply nested loop. In this case, the order of the generators is irrelevant (and the compiler might leverage this), but generally the order matters as we'll see. We also specified three filters: an ordering of `a`, `b`, and `c`, the Pythagorean equality, and a guard to ensure that the triple is primitive.

```
In [298]: #Fold the ordering filter into the lower and upper bounds of the generat
          ors.
          check_trips([Trip(a,b,int(c))
                       for a in range(1,m-1)
                       for b in range(a+1,m)
                       for c in range(b+1,m+1)
                       if a**2 + b**2 == c**2
                       and coprime(a,b)
                      ])
```

```
Good Trips
```

This sort of transformation is known as filter promotion. Rather than generate and then filter, we can avoid generating some values in the first place. We can imagine that that a compiler, a run time system, or a clever library, could accomplish simple filter promotion. Now `c` generation must come after `b` generation which in turn must come after `a` generation as `a` is used in the bounds of `b` generation which in turn is used in the bounds of `c` generation --- the generators can no longer be freely interchanged --- `c` generation is "more inner" than `b` generation which in turn is "more inner" than `a` generation.

Shouldn't this sort of transformation be the programmer's responsibility? Perhaps, but sometimes in mathematics we specify sets as generate then filter, and it can be very concise and easy to understand.

What about the next filter, the Pythagorean equality? Once `a` and `b` are selected, this filter tells us that given values for `a` and `b` , there is at most one one possible candidate to consider for `c` (`SQRT(a**2 + b**2)`), and we must check that this candidate is an integer.

We still need to check that `c` satisfys both generator bounds: `b+1 <= c <= m` . The lower bound comes for free, but we have to check the upper bound explicitly with a filter: `c <= m` .

```
In [299]: #Fold the Pythagorean equality filter into c generation, adding an upper
          bound check on c
          check_trips([Trip(a,b,int(c))
                       for a in range(1,m-1)
                       for b in range(a+1,m)
                       for c in [math.sqrt(a**2 + b**2)] if c == int(c) and c <= m
                       and coprime(a,b)
                      ])
```

```
Good Trips
```

Can we promote the filter `c <= m` ? It can be absorbed into b generation. Since `c**2 = a**2 + b**2` and `c <= m` , `a**2 + b**2 <= m**2` , giving us `b <= SQRT(m**2 - a**2)` . This only produces values for `c` at most `m-1` . Are we off by `1` ?

```
In [300]: #promote c <= m into b generation
          check_trips([Trip(a,b,int(c))
                       for a in range(1,m-1)
                       for b in range(a+1,int(math.sqrt(m**2 - a**2)))
                       for c in [math.sqrt(a**2 + b**2)] if c == int(c)
                       and coprime(a,b)
                      ])
```

```
Good Trips
```

It would be pretty impressive if a library or compiler could automatically do the filter promotion we've done by hand:

1. Fold `a < b < c` into `b` and `c` generation. This is the easiest one.
2. Calculate `c` rather than generate and filter. This involves quadratic algebraic manipulation.
3. Fold `c < m` into `b` generation (upper bound): again, algebraic manipulation.

It's tricky, but all of the preceding and the following could conceivably be produced automatically by a clever enough system, perhaps an MS project, or an undergrad capstone project for a very clever undergrad. Just for fun, let's pull common divisibility by `2` out of the `coprime` filter.

```
In [301]: #pull common divisibility by 2 out of the coprime filter
          check_trips([Trip(a,b,int(c))
                       for a in range(1,m-1)
                       for b in range(a+1,int(math.sqrt(m**2 - a**2)))
                       for c in [math.sqrt(a**2 + b**2)] if c == int(c)
                       and not (even(a) and even(b))
                       and coprime(a,b)
                      ])
```

```
Good Trips
```

Is there more fun to be had? It turns out that exactly one of `a` and `b` are even, but proving that is reasonably tricky.

```
In [302]: #strengthen divisibility by 2 to even(a) ^ even(b)
          check_trips([Trip(a,b,int(c))
                       for a in range(1,m-1)
                       for b in range(a+1,int(math.sqrt(m**2 - a**2)))
                       for c in [math.sqrt(a**2 + b**2)] if c == int(c)
                       and even(a) ^ even(b)
                       and coprime(a,b)
                      ])
```

```
Good Trips
```

```
In [303]: #fold divisibility by 2 into b generation using striding.
          check_trips([Trip(a,b,int(c))
                       for a in range(1,m-1)
                       for b in range(a+1,int(math.sqrt(m**2 - a**2)),2)
                       for c in [math.sqrt(a**2 + b**2)] if c == int(c)
                       and coprime(a,b)
                      ])
```

```
Good Trips
```

```
In [304]: #the smallest leg in a triple is 3.
          check_trips([Trip(a,b,int(c))
                       for a in range(3,m-1)
                       for b in range(a+1,int(math.sqrt(m**2 - a**2)),2)
                       for c in [math.sqrt(a**2 + b**2)] if c == int(c)
                       and coprime(a,b)
                      ])
```

```
Good Trips
```

Performing these last two transformations automatically would also be impressive research.

1. Expose common divisibility by `2` and promote into `b` striding
2. Start `a` with `3`. Good luck with this one. :-)