Enough Python to solve $N = p^2 + q^3$

Andrew Lin

September 2026

My goal is to explain enough Python to solve the programming problem in the OTIS application. **Some statements here are technically false** for the purpose of simplifying the exposition. Sample programs are given for each section.

Anything after a hashtag # is a *comment* and not read by Python.

§1 Basics

§1.1 Variables

A variable is a name assigned to a value. To create or change a variable, put the variable name before an = sign, then put the value (which can include variables already created). Variables can be the following:

- A real number, written in decimal form.
- A comma-separated list of numbers bounded by [ ].

The value of a variable will not change unless you change it directly.

```
x = 3
y = [-1/4, 3/4]
z = x          # z is 3
x = 4          # z is still 3
```

Some words are *keywords* with a special purpose and cannot be used as the name of a variable. When using a keyword for its special purpose, there must be a space between the keyword and any letter or number.

§1.2 Math operations

Python supports the following operations:

Symbol	Binary operation
+	addition
-	subtraction
*	multiplication
/	division
**	exponentiation
%	modulo

The + sign, when placed between two lists of numbers, concatenates them.

```
x = 10
y = 3
a = x + y               # a is 13
b = x - y               # b is 7
c = x * y               # c is 30
d = x / y               # d is 10/3
e = x ** y              # e is 1000
f = x % y               # f is 10 (mod 3) = 1
z = [1, 2, 3] + [4, 5] # z is [1, 2, 3, 4, 5]
```

Use `int(x)` to take the floor of a real number.

```
g = int(d)              # g is int(d) = int(10/3) = 3
```

§1.3 Printing

There are some *built-in* functions. The built-in **print** function takes one input and outputs it to the screen so you can see it.

```
x = 264
print(x ** 2)           # your screen will show 69696
```

§1.4 Conditionals

An if statement allows you to run a block of code only if a condition is satisfied. Use the **if** keyword, followed by a condition and a colon. The following lines should be an indented block of code that runs if and only if the condition is satisfied. Optionally, immediately after this indented block, use the **else** keyword, followed by a colon. Again, the following lines should be an indented block of code. This block of code will run only if the original condition does not hold.

§1.4.1 Conditions

The following symbols can be used to create conditions:

Symbol	Meaning
`==`	equal to
`>`	greater than
`<`	less than
`>=`	greater than or equal to
`<=`	less than or equal to
`!=`	not equal to

For conditions X and Y, the primitives `not X`, `X and Y`, `X or Y` behave exactly as expected.

```
x = 7
if x > 0 and x % 2 == 0:  # does not fire because x is odd
    x = x / 2
else:
    x = 3 * x + 1         # fires, so x is now 22
```

§1.5 Loops

Loops allow you to run the same piece of code multiple times.

§1.5.1 For loops

Use the **for** keyword, followed by a new variable name, the **in** keyword, a list of numbers, and a colon. The following lines should contain an indented block of code. The program iterates over the list of numbers, setting a variable with the specified name equal to the number in the list, and running the code block.

The built-in **range** function takes as input two integer inputs $a < b$ and outputs a list of all integers in order from a to b, including a but not b.

```
a = 0
for i in range(1, 101):
    a = a + i
# range(1, 101) is [1, 2, ..., 100]
# for each such i, we set i equal to the number and add it to a
# a = 1 + 2 + ... + 100 = 5050
```

§1.5.2 While loops

Use the **while** keyword, followed by a condition and a colon. The following lines should contain an indented block of code. The program runs the code in the block until the condition does not hold; it is possible that the code is never run at all.

For example, this would print the smallest power of 2 greater than 10^6:

```
n = 1
while n < 10**6:
    n = n * 2     # doubles n
print(n)          # prints 1048576
```

§2 Optional things that could make life easier

Section 1 is already sufficient for solving C.1, so if you prefer to learn as little as possible, you could stop reading now.

§2.1 Strings

In addition to real numbers and lists, Python supports strings like `"hunter2"`. For the purposes of C.1, the only two primitives you'll care about are:

- You can convert a number to its decimal string using `str(number)`, and
- You can use `x in y` to check whether the string `x` appears in `y`.

For example, we can make a list of powers of 2 up to 2^{100} that contain 67.

```
powers_of_2_with_67 = []
for n in range(0, 101):     # loop over 0 <= n <= 100
    if "67" in str(2**n):
        powers_of_2_with_67 = powers_of_2_with_67 + [2**n]
# Now, powers_of_2_with_67 = [16777216, 67108864, 4294967296, ...]
```

§2.2 The break keyword ends a while or for loop early

If you are in a **for** or **while** loop, you can terminate the loop early by using **break**.

```
# Look for a prime p for which 3^(p-1)-1 is divisible by p^2
guesses = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67]

for p in guesses:
    if (3**(p-1)-1) % (p**2) == 0:
        print(p)  # print the prime we found
        break     # exit right away; no need to waste time on more primes
```

§2.3 Define your own functions

Like in math, functions produce some output given some inputs. Define a function using the **def** keyword, followed by the function name, a comma-separated list of parameters bounded by parentheses, and a colon.

The following lines should be an indented block of code that determines the output given the parameters. To specify the output, use the **return** keyword followed by the data that should be outputted. The function can now be used by using the function name followed by a separated list of values bounded by parentheses (your inputs).

Once the program reads a single return statement, the rest of the function is not read. Any variables defined in a function cannot be accessed by the rest of the program.

```
def double(a):
    b = 2 * a
    c = 1434
    return b      # the program will exit here
    return 665    # the program will never reach this line
z = double(5)     # z is 10
y = c             # INVALID: c cannot be read outside the function body
```

Here is a more serious example that prints https://oeis.org/A006577, the number of Collatz steps for the first twenty values of n.

```
def collatz_time(a):
    count = 0
    while a > 1:
        if a % 2 == 0:
            a = a / 2
        else:
            a = 3 * a + 1
        count = count + 1
    return count
for n in range(1, 20):
    print(collatz(n)) # prints 0, 1, 7, 2, 5, 8, 16, 3, ...
```

(For large inputs a, one should replace `a / 2` with `a // 2` for technical reasons related to integer versus floating point arithmetic.)

§2.4 Speed

As an extremely crude rule of thumb, you can assume Python can perform about a million operations a second.