Novel Defense Against Code Theft Using Properties of Fibonacci Series

Sayan Chaudhry
Carnegie Mellon University
sayanc@andrew.cmu.edu

ABSTRACT

In recent years, the fabled concept of *style* has transferred from fashion parlance to the computer science industry. This plague was first introduced among our fellow soldiers with the original publication of *The Elements of Programming Style* in 1974. This book advocated the notion that code should be written not just for correctness and efficiency, but also for understanding by other humans. This paper highlights the several disadvantages of having well-styled code, apart from being vain and normative, and introduces a novel defence against reverse engineering attacks using properties of the Fibonacci series.

CODE STYLE

Most elements of good code style revolve around improving the visual appearance of the code. Some of these include:

1. Indent coding so you have space to write comments in the margins of the printed copy of the code
2. Excessive use of white space to increase source code file sizes
3. Proper use of capitalization to express how angry you were while writing that piece of code
4. Extremely long and well-defined identifier names that *completely* explain what the variable stores or function does

WHY GOOD STYLE IS BAD

Allegedly, conforming to given style convention makes software maintenance easier and also reduces the chance of making mistakes. This is clearly false, because:

1. Not a single soul wants to read or maintain anyone else's code.
2. If your code doesn't compile, it doesn't compile and beautifying the code won't make it compile.

Yet, dozens of introductory Computer Science courses brainwash their students into believing that code style is important and publish style guides that the students have to follow.

Moreover, in recent years, another serious drawback of having good style and readability has come to light: increased readability. Many focus groups have shown that more readable code is more likely to be stolen because it is easier to understand by the third party.

For example, the following 2 snippets of code do the exact same thing but college students are 42% more likely to copy the code on right because it is more readable.

```
+++++[-]
```

```
*p = +5;
while(*p != 0) {
  *p--;
}
```

Simple `while` loop written in Brainfuck (left) and C (right).

Thus, have good style makes students targets for cheating and puts them at risk of committing academic integrity violations, which may lead to failing the class and even expulsion from the program. For this reason, it is common industry practice to have unreadable code written in assembly so that third parties are unable to reverse engineer the source code and make changes to the software.

INDENT CODING

While good style has absolutely no merits in and of itself, one of its core pillars indent coding can be used to make it harder for adversaries to reverse engineer the code. This technique is called *Fibonacci indent coding*.

Instead of indenting code blocks by a consistent 2/4 spaces, Fibonacci indent coding indents each line according to the Fibonacci series. So, the zeroth lines is indented by 0 spaces, the first line by 1 space, the second line by 1 space, the third line by 2 spaces, the fourth by 3 spaces, the fifth by 5 spaces, and you get the point.

```c
int main() {
  int first, second, tmp;

  scanf("%d", &first);
  scanf("%d", &second);

  tmp = first;
  first = second;
  second = tmp;

  printf("First Number = %d\n", first);
  printf("Second Number = %d\n", second);

  return 0;
}
```

```c
int main() {
 int first, second, tmp;

 scanf("%d", &first);
  scanf("%d", &second);

   tmp = first;
    first = second;
       second = tmp;

            printf("First Number = %d\n", first);
                   printf("Second Number = %d\n", second);

                                  return 0;
                                                  }
```

Standard indenting (top) and Fibonacci indenting (bottom)

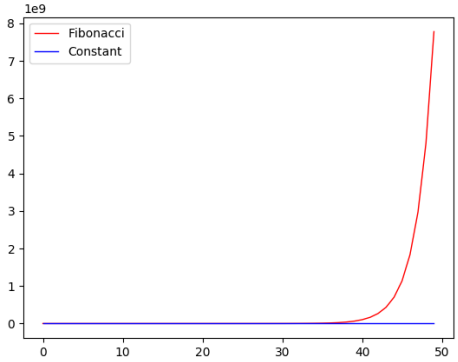
The advantages of Fibonacci indent coding are clear:

1. The increasingly indented lines make it much harder for any third-party to understand and copy the source code, improving *code obfuscation*, as desired.
2. The eleventh Fibonacci number is 89, which is greater than 80, the maximum line width on most editors. Therefore, your code will be scroll past the text editor window starting at the eleventh line, which will confuse most naïve adversaries.
3. Since the Fibonacci sequence grows exponentially, stripping the Fibonacci indenting cannot be done in polynomial time. In other words, even an automated solution to remove the indent coding will take a really long time to do so.
4. Any code snippet that has been copied from StackOverflow or someone else's program will not be Fibonacci indented and obviously distinguishable from the rest of the code. This heuristic can be used to detect software similarity without the use of fancy tools like MOSS, fern, or lichen.
5. It produces a nice staircase pattern in your code.

COMPARISON TO OTHER SEQUENCES

We tested the performance of the Fibonacci indent against 5 other sequences to see which one provides the best performance for our indent coding.

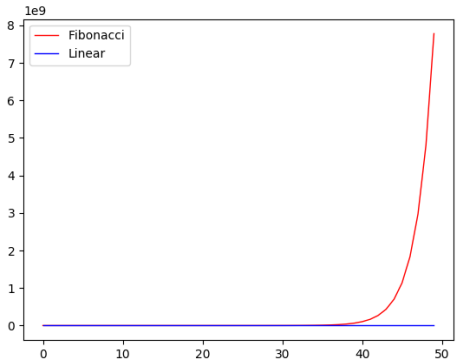
The Constant Series:




```
[ 2 for i in range(50) ]
```

Just using the constant series was equivalent to the naïve 2-space indent coding guidelines we were trying to avoid in the first place.

The Linear Series:



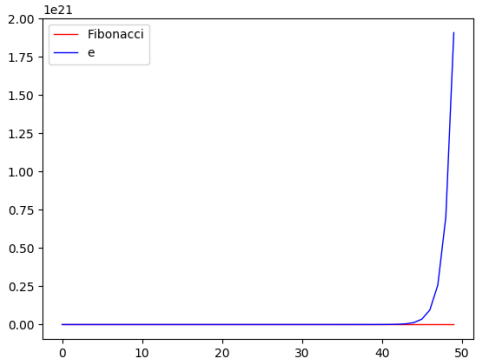

```
[ i for i in range(50) ]
```

The linear series was slightly better, till we were reminded that the Ramanujan summation assigns the sum -1/12 to this series, meaning as we have more and more lines in our code, our

file size will converge to a negative number, which is undesirable.

$$1 + 2 + 3 + 4 + \cdots = -\frac{1}{12},$$

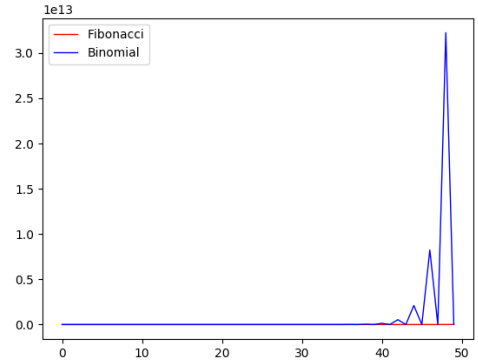
The Exponential Series:




```
[ exp(i) for i in range(50) ]
```

Using e^i gave us the much desired exponential growth, even more so than the Fibonacci series. However, many (all) elements in the series were floating point numbers and it wasn't possible to indent lines by a non-natural number of spaces using the technology available to us today.

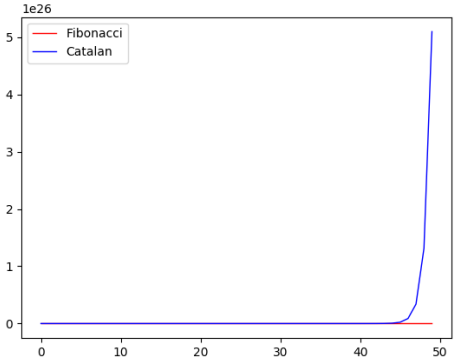
The Binomial Series:




```
[ binom(i, i/2) for i in range(50) ]
```

The binomial series also gave us exponential growth. However, this didn't work for odd numbers because the second term in the binomial coefficient was a decimal (`xx.5`) and gave a `ValueError`, as indicated by the spikes in the graph.

The Catalan Series:




```
[ binom(2*i, i)/(i+1) for i in range(50) ]
```

Using the Catalan numbers for our series avoided the problems we had with the exponential and binomial series. However, we obviously had to reject it because the series name wasn't Italian enough.

PRACTICAL REALIZABILITY

The amazing thing about Fibonacci indent coding is that even though it is an extremely complicated, it is extremely practical. In fact, there already exists an open source Atom package that Fibonacci indents a piece of code.

The package has its limitations though. A couple of them are:

1. It currently supports indenting up to 20 lines, because the Fibonacci numbers grow really large really fast.
2. Python support is still in beta.

CONCLUSION

Overall, we have seen how Fibonacci indent coding can reduce code style and readability and improve code obfuscation. This significantly reduces the risk of academic integrity violations and copyright infringement of your code.