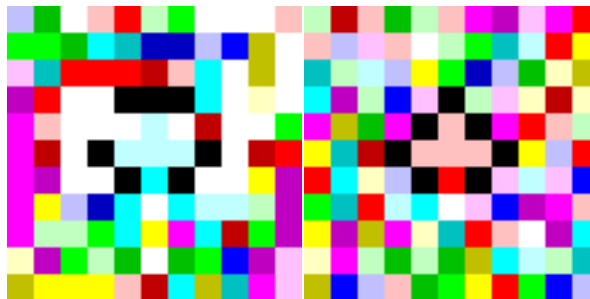


Piet

Brage Isak Keiserås

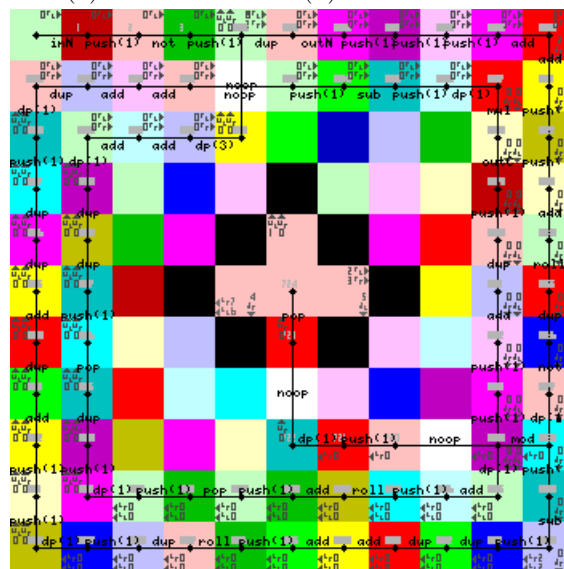
September 10, 2026

1 Piet



(a) Fibonacci

(b) Fibonacci Prettified



(c) The Execution of Fibonacci Prettified

“Anything which can be computed can be computer by a Turing Machine”
 piet is Turing-complete, so everything that can be computed can be
 computed by piet.

2 Kolmogorov Complexity

“Kolmogorov complexity of an object, such as a piece of text, is the length of a shortest computer program (in a predetermined programming language) that produces the object as output.” (https://en.wikipedia.org/wiki/Kolmogorov_complexity)

Consider the strings:

[illegible]

VS

[illegible]

VS

10101110101011101010010101011101010111101011010000111110101010100101111

Which one is more “complex”?

2.1 Only 1s

First consider

[illegible]

Input: n

```

1 out  $\leftarrow$  "";
2 for  $i = 0; i < n; i \leftarrow i + 1$  do
3    $\lfloor$  out  $\leftarrow$  out ++ "1";
4 return out;

```

2.2 Repeating 10s

[illegible]

Input: n

```

1 out  $\leftarrow$  "";
2 for  $i = 0; i < n/2; i \leftarrow i + 1$  do
3    $\lfloor$  out  $\leftarrow$  out ++ "10";
4 return out;

```

2.3 Random String

10101110101011110101001010101101010111101011010000111110101010100101111

```
1 return
```

```
10101110101011110101001010101101010111101011010000111110101010100101111;
```

3 Code Golfing

Most competitive programming optimize either the amount of time used to solve a problem, or the time the solution takes.

For code golfing the winner is the program with the smallest piece of code. See Figure 2.

```

import java.util.Scanner;

public class Fibonacci {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        int a = 0, b = 1;

        while (a <= n) {
            System.out.print(a + " ");

            int c = a + b;
            a = b;
            b = c;
        }

        sc.close();
    }
}

void main(){
    var s=new java.util.Scanner(System.in);
    int n=s.nextInt(),a=0,b=1;
    for(;a<=n;System.out.print(a+" ")){a=b-(b=a);}
}

```

Figure 2: Two equal programs, but one is smaller

4 The Most Beautiful Code for a Problem?

In piet, the code for a program is a $n \times m$ picture. In general, one can argue in piet, that a smaller picture is more “beautiful” than a bigger one. This begs the question “How big does the $n \times m$ input picture have to be able to compute a certain algorithm?”.

Before looking at this, we look at a simpler problem, “How much space, in piet, is used to simply represent a number?”

Notice that any number can be factorized as per the fundamental theorem of arithmetic.

For instance:

$$100 = 50 \cdot 2 = 25 \cdot 4 = 5 \cdot 5 \cdot 4 = 5^2 \cdot 2^2$$

Programs outputting the number 100 can utilize this fact. To represent 100 we can represent 5 twice, and 2 twice, and multiplication 4 times. In piet there is also a notion of duplicating a number, so we can represent 5 once, and duplicate it, saving ~ 5 pixels (actually $5 - \text{size of duplicating}$, which is 1)

Below (Figure 3) are examples of programs all outputting 100.

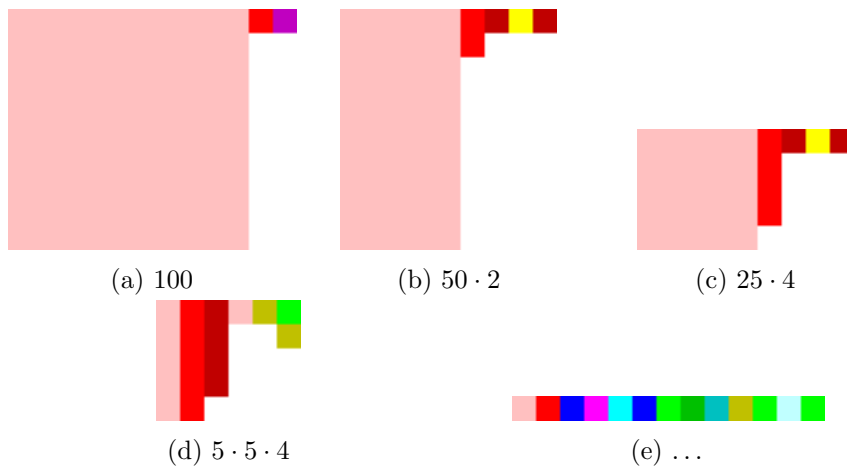


Figure 3: Programs outputting 100

In particular note (e). it reduces printing 100 to a series of 1 by 1 squares outputting 100.



Wouldn't you say this is the most beautiful version? Maybe not. We can also note it can be made in a different way, where instead of being a 1×13 "snake", it is a $n \times n$ square. This is left as an exercise for the reader.

The code it runs can be serialized as follows:

```
push(1)
dup
add
dup
add
dup
push
add
dup
```

```

mul
mul
out

```

This leaves us with a question, “how do we turn a $1 \times x$ “snake” into a $n \times n$ square.”

5 Research Questions

For any given $x \times y$ program how do we turn it into only using 1×1 squares, and can we turn it into a $n \times n$ program?

As an example consider:



this can be turned into:



5.1 Some more unstructured questions:

- Smallest program which outputs the number n ?
- Smallest grid for any given function
- Prettiest program?
 - Piet only cares about difference between previous and current
Shift every colour by n , same program
- Restrictions
 - Only 1 by 1 squares
- Improve current tooling
- Similar models more similar to turing machines/cellular automata?