Appendix - The Fysh Programming Language

This is fysh

```
><>
```

Fysh have to be terminated `~`

```
><> ~
```

This is Steven

```
><steven> ~
```

Steven has binary scales. `}` represents 1 and `)` represents 0. Steven is valued at `b101` (5 in decimal)

```
><steven> = ><})}> ~
```

Steven doesn't give a flying fysh about their scale direction. ≈ for variable assignment is cool with them too

```
><steven> ≈ ><}({> ~
```

Steven is blind. You have the power to bless them with sight, but it's completely optional

```
><steven> ≈ ><{({°> ~
><steven> = ><{({o> ~
```

When there is a school of fysh, their collective value is equal to the sum of each individual member. This gives Steven a value of `b101 = b100+ b001`

```
><steven> ≈ ><{((°> ><(({°> ~
```

Sometimes fysh want to be different and swim the other direction. This takes away from the school's value. This gives Steven a value of `b101 = b111 - b010`

```
><steven> ≈ ><{{{°> <°)}})>< ~
```

Fysh often get lonely. This loneliness causes fysh to meet new fysh and proliferate. This gives Steven a value of `b101010 = b110 * b111`

```
><steven> ≈ ><{{(°> ♡ ><{{{°> ~
```

Since the fysh are lonely, they aren't too picky. They'll make do with a lesser form of love `<3`

```
><steven> = ><{{(o> <3 ><{{{o> ~
```

Not every fysh story is a happy one. At times, separation is unavoidable, and their division is symbolized by a heartbreak

```
><steven> = ><{{(({o> </3 ><{({o> ~
```

As life goes on, we learn from our mistakes and improve. Steven's self help journey allowed them to grow an extra tail, incrementing their value by 1

```
>><steven> ~
```

Sometimes we feel like a fyshup, a failure. And that's ok, it's a part of being fysh. However for some fysh, this feeling is too much to handle and is internalized. They haven't received the emotional support they need and have gone on a downward spiral, causing them to feel worthless. They begin to retreat and try to swim away in the opposite direction causing their value to decrement by 1.

```
<steven><< ~
```

Sometimes Steven F*shs up and throws an error. This can be done using two WTF (What The Fysh) encompassing a string

```
><!@#$>
        What The Fysh?!
<!@#$><
```

Not all fysh are created equal. But sometimes they are. We can check this using `≈≈` or `==`

```
><steven> ≈≈ ><theFysh> ~
><steven> == ><theFysh> ~
```

But sometimes we want to ensure that two fysh are different. This can be done using `~≈` or `~=`

```
><steven> ~≈ ><theFysh> ~
><steven> ~= ><theFysh> ~
```

Tadpoles, curious by nature, gravitate towards larger fysh, perhaps seeking guidance or a new destiny. This tadpole ponders if Steven has more than six fins. (Steven > sixFins, Steven >= sixFins)

```
><steven> o~ ><sixFins> ~
><steven> o~≈ ><sixFins> ~
```

Or, with a twist of fate, they explore if Steven is the lesser, a journey of humility and discovery. (Steven < sixFins, Steven <= sixFins)

```
><steven> ~o ><sixFins> ~
><steven> ~o≈ ><sixFins> ~
```

Steven is a curious fysh and often wonders if the world is as it seems. They ponder the meaning of life, the universe, and everything. They seek the truth, and if both steven and theTruth are true, they find it. (Steven && theTruth). Or if steven or theTruth is true, they find it. (Steven || theTruth). or if steven is not real and this is all a simulation, they find it. (!!Steven)

```
><steven> && ><theTruth> ~
><steven> || ><theTruth> ~
!! ><steven> ~
```

Steven is looking for a change and is interested in having their bits rearranged. Given it's 2024, a time when open-mindedness prevails, Steven has a variety of bitwise manipulation techniques at his disposal: `AND (&)`, `OR (|)`, `XOR (^)`, `NOT (!)`. For those moments when he's feeling particularly adventurous, he might even consider the `logical shift left (<<)` or `logical shift right (>>)` operators.

```
><steven> &  ><{((°> ~
><steven> |  ><{((°> ~
><steven> ^  ><{((°> ~
! ><steven> ~
><steven> << ><{((°> ~
><steven> >> ><{((°> ~
```

In the whirlpool of fysh logic, the while loop, symbolized by `><(((@>`, ensnares conditions within `[ ]`, with `><>` and `<><` encapsulating the iterative heart.

```
><(((@> [ ><steven> o~ ><{((°> ]
><>
        <steven><< ~
<><
```

Navigating through decision streams, if and else statements flow naturally. `><(((^>` marks the if, > `<(((*> ><(((^>` for else if, and `><(((*>` for the uncharted else.

```
><(((^> [ ><steven> o~ ><{((°> ]
><>
        <steven><< ~
<><

><(((*> ><(((^> [ ><steven> ~o ><{((°> ]
><>
        >><steven> ~
<><

><(((*>
><>
```

```
        ><steven> ≈ ><(((°> ~
<><
```

For those seeking the fortune of randomness, `><###>` unveils a 32-bit treasure, while `><//>` whispers single-line comment and `></*>`, `<*\><` span multiline scriptures.

```
><###> ~

><//> Comment

></*>
Comments
Comments 2
<*/><
```

A fysh tank `[ ]` is an array of fysh separated by fysh food `-`

```
><steven> ≈ [><({(°> - ><({(°>] ~
```

These fysh tanks can be traversed using a fysh

```
><steven>[><(({°>] ~
```

Factorial Example

```
><//> Comment

><number>    ≈ ><{({°>   ><//> b101 = 5
><factorial> ≈ ><(({°>   ><//> b001 = 1

><//> while number > 1
><(((@> [><number> o~ ><(({°>]
><>
        ><//> factorial = factorial * number
        ><factorial> ≈ ><factorial> ♡ ><number> ~

        ><//> number -= 1
        <number><< ~
<><
```