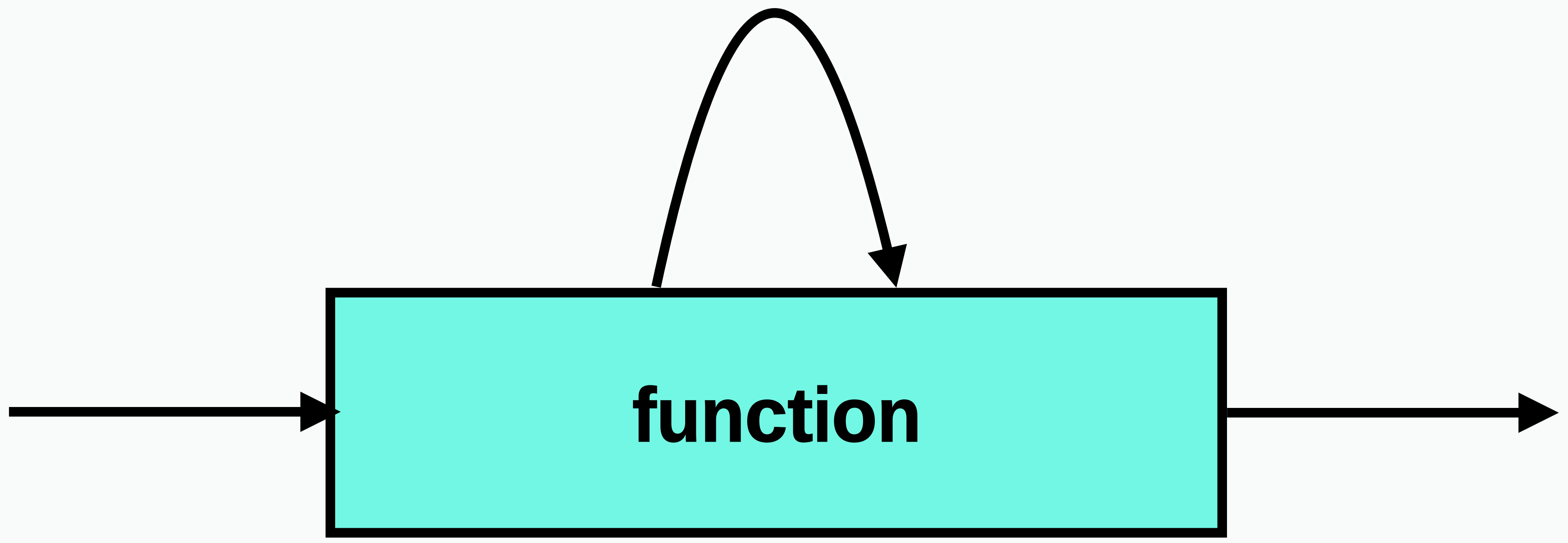
?

You may think of as a programming structure where a function calls itself. We call such a function a .

Many algorithms can be implemented using recursion.



?

- Recursion can provide an elegant solution which breaks a problem down into smaller parts.
- Recursion is used in numeric calculations, tree traversals, and many other applications.
- Recursion can solve problems without requiring an explicit loop.

A classic example of a recursive function is the where we calculate the value of

!

```
int factorial(int n) {
    if ((n == 0) || (n == 1)) {
        // recall 0! is defined to be 1
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

!

```
int factorial(int n) {
    if ((n == 0) || (n == 1)) {
        // recall 0! is defined to be 1
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

Here's the recursion

!

```
int factorial(int n) {
    if ((n == 0) || (n == 1)) {
        // recall 0! is defined to be 1
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

What's all this?

!(   )

```
int factorial(int n) {
    return n * factorial(n - 1);
}
```

What happens if we call this function?

!(   )

```
int factorial(int n) {
    return n * factorial(n - 1);
}
```

Let's say we call this function, providing the value 5 as an input parameter.

This will calculate $5 \times 4 \times 3 \times 2 \times 1 \times 0 \times -1 \times -2 \times -3 \times -4$... etc.

!(   )

```
int factorial(int n) {
    return n * factorial(n - 1);
}
```

Let's say we call this function, providing the value 5 as an input parameter.

This will calculate $5 \times 4 \times 3 \times 2 \times 1 \times 0 \times -1 \times -2 \times -3 \times -4$... etc.

This will   !

!

```
int factorial(int n) {
    if ((n == 0) || (n == 1)) {
        // recall 0! is defined to be 1
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}
```

Base cases

In addition to the recursive case, a recursive function must have one (or more) , that provide termination criteria for the function.

A recursive function must have

- a   , and
- one or more   (without recursion)

The recursive case breaks the problem down into smaller instances. The base cases provide termination criteria, breaking the chain of recursion and preventing infinite regress.

Another classic example of recursion is the Fibonacci function.

The Fibonacci sequence starts with 0, 1, and then calculates subsequent terms by taking the sum of the previous two terms in the sequence. So a Fibonacci number, denoted where is an index, is calculated thus

$$
\begin{cases}
F_n = 0 & \text{if } n = 0 \\
F_n = 1 & \text{if } n = 1 \\
F_n = F_{n-2} + F_{n-1} & \text{otherwise}
\end{cases}
$$

Another classic example of recursion is the Fibonacci function.

A Fibonacci sequence starts with 0, 1, and then calculates subsequent terms by taking the sum of the previous two terms in the sequence. So a Fibonacci number, denoted where is an index, is calculated thus

$$\begin{cases} F_n = 0 & n = 0 \\ F_n = 1 & n = 1 \\ F_n = F_{n-2} + F_{n-1} & \text{otherwise} \end{cases}$$

← Base cases

C++

Here's the Fibonacci function to return the th Fibonacci number, implemented in C++:

```
int fibonacci(int x) {
   if (x == 0) || (x == 1) {
      return(x);
   } else {
      return(fibonacci(x - 2) + fibonacci(x - 1));
   }
}
```

C++

Here's the Fibonacci function to return the th Fibonacci number, implemented in C++:

```
int fibonacci(int x) {
   if (x == 0) || (x == 1) {
      return(x);
   } else {
      return(fibonacci(x - 2) + fibonacci(x - 1));
   }
}
```

C++

Here's the Fibonacci function to return the th Fibonacci number, implemented in C++:

```
int fibonacci(int x) {
   if (x == 0) || (x == 1) {
      return(x);
   } else {
      return(fibonacci(x - 2) + fibonacci(x - 1));
   }
}
```

' ?

- Recursion is an elegant approach to break a problem down into smaller instances and solve by recurring calculation.
- Recursive functions are functions that call themselves.
- Recursive functions require a recursive case and at least one base case.
- While elegant, they may not be the most efficient approach, so use with caution.