Currying

Currying is a transformation of functions that translates a function from callable as f(a, b, c) into callable as f(a)(b)(c). An American mathematician named Haskell Curry developed this technique, that's why it is called currying.
It splits the arguments of a function and makes it possible to call them sequentially.

Currying using Bind

```
const multiply  = (x, y)=> x * y;

const multiplyByTwo = multiply.bind(this, 2);

const result = multiplyByTwo(50);
console.log(result); //100
```

Currying using Closure

```
const multiply  = (x)=>
{
  return (y)=>
  {
    return x * y;
  }
}
```

We can use shorthand for the above code.

```
const multiply  = (x)=> (y)=> x * y;


const multiplyByTwo = multiply(2);

const result1 = multiplyByTwo(50);

console.log(result1);

const result2 = multiply(4)(5) //20
console.log(result2);
```

Currying using Closure and Callback

```
const curry = (f)=>
{
   return (x)=>
   {
    return (y)=>
    {
      return f(x, y);
    }
   }
}
```

We can use shorthand for the above code.

```
const curry = (f) => (x) => (y)=> f(x, y);

const sum = (x, y)=> x + y;


const currySum = curry(sum);

const addTen = currySum(10);

const result1 = addTen(60);
console.log(result1);


const result2 = currySum(4)(5) //9
console.log(result2);
```

Partially Applied Curried Functions

Partially applied curried functions are very common use of currying. It lets us create functions where we can make custom functions from it.

Partial application and currying are both techniques used in functional programming to create new functions by fixing a certain number of arguments of a given function. While they share similarities, they have some differences in their approach.

In above examples multiplyByTwo and addTen are examples of partial applied currying.

Infinite Curry

```
const curryMultiply = (x) => (y)=> y ? curryMultiply(x * y) : x;
const result = curryMultiply(1)(2)(3)(4)(5)();
console.log(result); //120
```

Advanced Curry

```
const curry = (f)=>
{
  return function curried(...args)
  {
    if(args.length >= f.length)
    {
      return f(...args);
    }
    else
    {
      return function(...next)
      {
        return curried(...args, ...next);
      }
    }
  }
}

const sum  = (a, b, c) => a + b + c

const curriedSum = curry(sum);

console.log(curriedSum(1)(2)(3)) //6
console.log(curriedSum(1, 2)(3)) //6
console.log(curriedSum(1)(2,3)) //6
console.log(curriedSum(1, 2, 3)) //6
```