# Recursion

# Recursion

A definition that refers to itself

A function that calls itself

# Factorial

$5! = 5 * 4 * 3 * 2 * 1 = 120$

$4! = 4 * 3 * 2 * 1 = 24$

$3! = 3 * 2 * 1 = 6$

$2! = 2 * 1 = 2$

$1! = 1 = 1$

$0! = 1$

# Factorial

```
function factorial(n) {
  if (n == 0) {
    return 1;
  } else {
    return n * factorial(n - 1);
  }
}
```

# Recursion

- Recursive calls
- Stop condition

# Factorial

```
function factorial(n) {
  // stop condition
  if (n == 0) {
    return 1;
  }
  // recursive call
  else {
    return n * factorial(n - 1);
  }
}
```

# Factorial

```
factorial(4);
4 * factorial(3);
4 * 3 * factorial(2);
4 * 3 * 2 * factorial(1);
4 * 3 * 2 * 1 * factorial(0);
4 * 3 * 2 * 1 * 1;
// => 24
```

# Fibonacci

$$fib(0) = 0$$

$$fib(1) = 1$$

$$fib(2) = fib(1) + fib(0)$$

$$fib(3) = fib(2) + fib(1)$$

$$fib(4) = fib(3) + fib(2)$$

...

$$fib(n) = fib(n-1) + fib(n-2)$$

# Fibonacci

```
function fib(n) {
  if (n == 0) {
    return 0;
  } else if (n == 1) {
    return 1;
  } else {
    return fib(n-1) + fib(n-2);
  }
}
```

# Fibonacci

```
fib(5);
fib(4) + fib(3);
fib(3) + fib(2) + fib(2) + fib(1);
fib(2) + fib(1) + fib(1) + fib(0) + fib(1) + fib(0) + fib(1)
fib(1) + fib(0) + fib(1) + fib(1) + fib(0) + fib(1) + fib(0) + fib(1);
1 + 0 + 1 + 1 + 0 + 1 + 0 + 1;
// => 5
```

# Iterative vs. Recursive

Alternative solutions
to the same problem

# Recursive

```
function factorial(n) {
  // stop condition
  if (n == 0) {
    return 1;
  }
  // recursive call
  else {
    return n * factorial(n - 1);
  }
}
```

# Iterative

```
function factorial(n) {
  var result = 1;

  while (n != 1) {
    result = result * n;
    n = n - 1;
  }

  return result;
}
```

# Iterative vs. Recursive

Alternative solutions to the same problem

- **Recursion:** Function calling itself to perform instructions repeatedly
- **Iteration:** Using a looping construct to perform instructions repeatedly