JS Functions

A Fundamental Building Block

Objectives

- Understand why we use functions
- Define a function without arguments
- Define a function with arguments
- Return a value from a function

Functions

Functions let us wrap bits of code up into REUSABLE packages. They are one of the building blocks of JS.

Declare a function first:

```
function doSomething() {
  console.log("HELLO WORLD");
}
```

Then call it:

```
doSomething();
doSomething();
doSomething();
doSomething();
```

with (): calling it
without () : return the code of the function

Functions

Suppose I want to write code to sing "Twinkle Twinkle Little Star"

```
console.log("Twinkle, twinkle, little star,");
console.log("How I wonder what you are!");
console.log("Up above the world so high,");
console.log("Like a diamond in the sky.");
```

To sing it again, I have to rewrite all the code. This is not DRY!

```
console.log("Twinkle, twinkle, little star,");
console.log("How I wonder what you are!");
console.log("Up above the world so high,");
console.log("Like a diamond in the sky.");
```

Functions

We can write a function to help us out

```
function singSong() {
    console.log("Twinkle, twinkle, little star,");
    console.log("How I wonder what you are!");
    console.log("Up above the world so high,");
    console.log("Like a diamond in the sky.");
}
```

To sing the song, we just need to call *singSong();*

```
//to sing the entire song 4 times
singSong();
singSong();
singSong();
singSong();
```

Arguments

Often we want to write functions that take inputs.

```
function square(num) {
  console.log(num * num);
}
```

Now when we call *square* we need to pass in a value

```
square(10);  //prints 100
square(3);   //prints 9
square(4);   //prints 16
```

Arguments

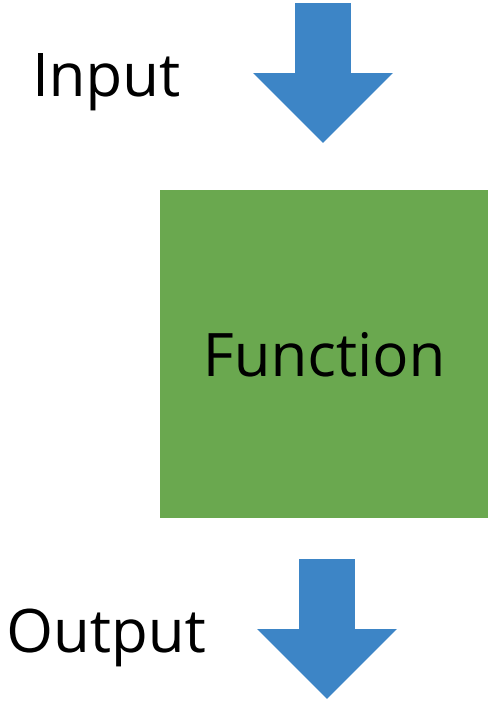
Functions can have as many arguments as needed

```
function area(length, width) {
  console.log(length * width);
}
area(9,2); //18


function greet(person1, person2, person3){
  console.log("hi " + person1);
  console.log("hi " + person2);
  console.log("hi " + person3);
}
greet("Harry", "Ron", "Hermione");
```

The Return Keyword

Often we want a function to send back an output value



The Return Keyword

We use the *return* keyword to output a value from a function

```
//this function capitalizes the first char in a string:

function capitalize(str) {
  return str.charAt(0).toUpperCase() + str.slice(1);
}

var city = "paris";                 //"paris"
var capital = capitalize(city);     //"Paris"

//we can capture the returned value in a variable
```

without return, its returned is undefined

[1,end]

The Return Keyword

The *return* keyword stops execution of a function

```
function capitalize(str) {
  if(typeof str === "number") {
    return "that's not a string!"
  }
  return str.charAt(0).toUpperCase() + str.slice(1);
}

var city = "paris";                //"paris"
var capital = capitalize(city);  //"Paris"

var num = 37;
var capital = capitalize(num);  //"that's not a string!"
```

Another Syntax

Function Declaration vs. Function Expression

```
//function declaration
function capitalize(str) {
  return str.charAt(0).toUpperCase() + str.slice(1);
}

//function expression
var capitalize = function(str) {
  return str.charAt(0).toUpperCase() + str.slice(1);
}
```

if then capitalize is assigned with 15. Then the function is lost.

Exercise

What does the following code return?

```
function test(x,y) {
  return y - x;
}

test(10, 40);
```

Exercise

What does the following code return?

```
function test(x) {
  return x*2;
  console.log(x);
  return x/2;
}

test(40);
```