JavaScript Functions

In design we often use the phrase **form over function**, what does that **imply**?

functions are one of the fundamental **building blocks** in JavaScript

(a procedure in the code, must be defined in scope)

We write a **function** when:

- There is a **block of code** that together performs a **single task**
- We have to **call the code more than once** (don’t repeat code)
- We **define an object** that will act as a **template** for new things (instances)

```
//A function is a piece of code that is called by name
function newSlide() {
  //add new slide to slideshow
}

//We use the function by writing
newSlide();
```

Writing a **function**

```
//number is the parameter passed in
function square(number) {
  result = number * number;
}

//call it by using
square(4);

//the result would be 16 (4x4)
```

It can be **passed data** to operate on (parameters).

```
//More parameters can be passed by adding a comma inside the brackets
function multiplier(number1, number2) {
  //code
}

//Functions can optionally return data (the return value)
function multiplier(number1, number2) {
  return number1 * number2;
}

let x = multiplier(3, 4)
console.log(x);
```

```
function multiplier(number1 = 3, number2 = 4) {
  return number1 * number2;
}

let x = multiplier(2)
console.log(x);
//Notice the default values assigned to each parameter
```

New!
in ES6

Parameters can have a **default value**

```
//this calls the above createDiv and the defaults will be applied
multiplier();

//now this calls again, but with new parameters:
multiplier(5, 2);

//and again, leaving default width and height:
createDiv(undefined, 7);
```

```
var square = function(number) {
  return = number * number;
}

var x = square(3);

/* This initially looks like more code for the
same result, there are times such as in objects
that this makes more sense or as in a later slide: */
```

Functions can also be written as a **function expression**

```
/* This looks a bit odd at first, but the basic rule is,
substitute the word ‘function’ for ‘=>’ (equals and greater
than with no space). Here’s the old way: */

var add = function(x, y) {
  return = x + y;
}
console.log(add(10, 20));

/* The new way, notice the arrow moves to the other side of
the parameters brackets: */
let addUp = (x, y) => {
  return = x + y;
}
console.log(addUp(10, 20));
```

New!
in ES6

Arrow Functions

```javascript
//Here's what was on the last slide
let addUp = (x, y) => {
  return = x + y;
}
console.log(addUp(10, 20));

//No curly braces as it's all on one line and no return keyword:
let addUp = (x, y) => x + y;
console.log(addUp(10, 20));
```

Arrow Functions can **write the same code even smaller**

Let’s look at some examples

Functions as **objects** :

- This is a brief intro, we'll cover this in **more detail next week**
- Functions are **also objects** in their own right and can be used to **make new copies** (instances) of those objects
  - *Sounds confusing?!*
  - *I can't make a new coin from a coin!*
  - *But if I had the mould (and hot metal) I could!*

```
//Consider this:
function Make_person_object(firstname, lastname, age) {
  this.firstname = firstname;
  this.lastname = lastname;
  this.age = age;
}

//We make new people (instances) out of the object
var john = new Make_person_object('John','Smith', 45);
//This is known as the constructor

//We can access individual properties of our John person
console.log(john.age);
//we grabbed John's age by calling the age of object John (we can add more)

var janet = new Make_person_object('Janet','Patel', 22);
console.log(janet.age);
```

```
/* We can also create nested functions that do something useful for us,
add this inside the function: */
function Make_person_object(firstname, lastname, age) {
  this.firstname = firstname;
  this.lastname = lastname;
  this.age = age;

  this.myName = function () {
    var fullname = firstname + ‘ ’ + lastname;
    return fullname;
  }
}

//Now we are using the function expression because it allows it to be called as follows
console.log(janet.myName());
```

```
janet.myName();

/*
  * When accessing a function like this it is known as a method

  * If a function is inside an object we are accessing a method
    of the object, otherwise a function is just a function.

  * Method meaning “a way of using” the object.
*/
```

Practice Quiz!

Give one example of when you write a function?

How do I pass in two parameters to a function?

```
function square(number) {
  result = number1 * number2;
}
```

Can you write an arrow function using function seen here?