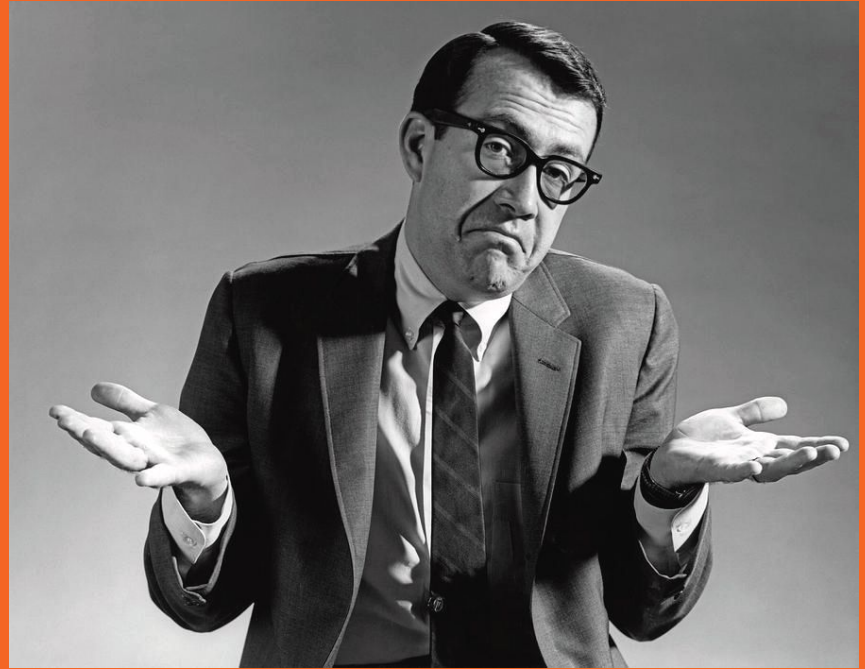


---

---

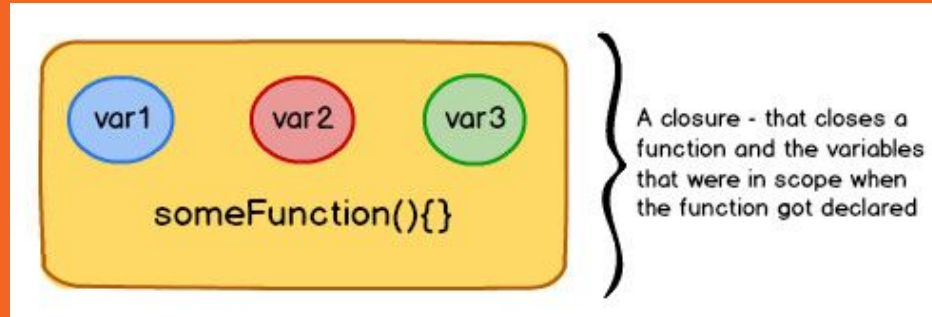
# JavaScript Review

Things you know,  
but probably  
forgot



---

# Functions



# What is a function?

As a baseline, a function is a set of instructions that you can pass around.

It's a recipe.

Imagine you went to a dinner party, and loved the jello centerpiece! You can't live without it. But you aren't the only one. It was such a smash hit, that 10 people would like the recipe too. What would be more efficient?

1. The host goes to every single person and tells them the instructions individually.
2. The host writes down the set of instructions, and hands it out.



The second option, of course. Option one is like rewriting the same code over and over. That is not efficient, or DRY, as developers say:

D - Don't  
R - Repeat  
Y - Yourself

Always write your code in a concise way as possible. Put repeated code into functions.

# Back to the Jello...

## Amazing Jello Fantastico

### Ingredients:

- One package lime jello
- 1 cup sliced cucumber
- 1 cup sliced onions
- 2 cups maraschino cherries
- 3 cups sliced olives

### Instructions:

1. Prepare jello according to package directions in a large bowl.
2. Mix the rest of the ingredients in with the jello.
3. Pour mixture into mould.
4. Let set in fridge overnight.
5. De-mould and enjoy.








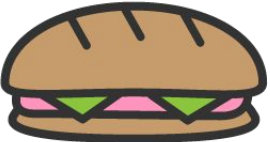
```
function amazingJelloFantastico(jello,
veg1, veg2, fruit, otherIng ){
  let mix = "Stir";
  jello = jello + mix;
  let result = jello + veg1 + veg2 +
fruit + otherIng;
  result = result + mix;
  return result;
}
```

```
amazingJelloFantastico( "lime",
"cucumber", "onions", "maraschino
cherries", "sliced olives" );
```

# Alright, that was a really dumb example.

And the code doesn't even run....

What I'm getting at is that functions are simply a set of instructions. Let's look at some code that will actually run:

```
function makeSandwich(, , ) {  
    let sandwich =  =  +  +  ;  
    return  ;  
}
```

<https://www.codeanalogies.com/javascript-functions-explained>

## In non-visual terms:

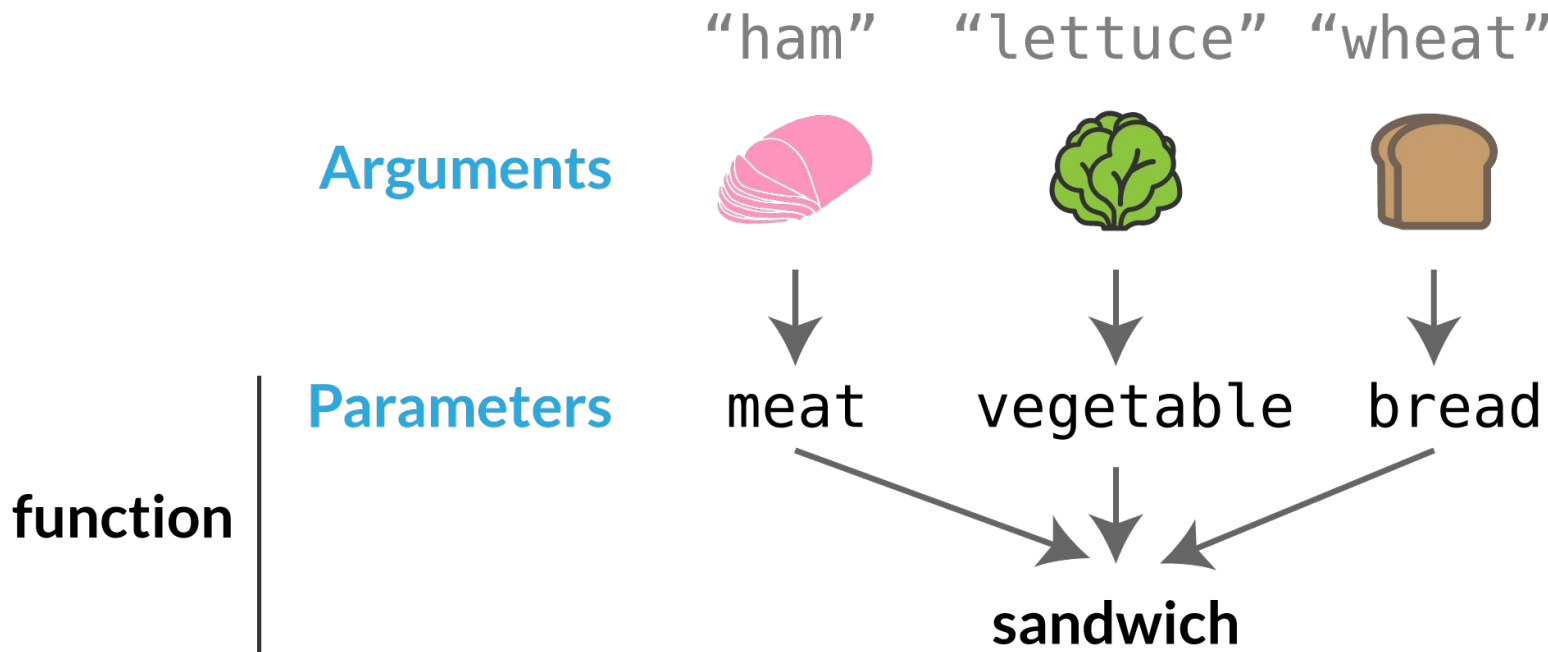
```
function makeSandwich(meat, veg, bread) {  
  let sandwich = meat + veg + bread;  
  return sandwich;  
}
```

```
makeSandwich("ham", "lettuce", "wheat");  
//hamlettucewheat
```



# There are a couple of types of functions:

1. Functions that take in parameters and return a value.
2. Functions that don't take in parameters and don't return a value.



## Return a Value

```
function addTwo(num) {  
  let sum = num + 2;  
  return sum;  
}
```

```
addTwo(8);
```

```
// => 10
```

## Doesn't Return a Value

```
num1 = 8;
```

```
function addTwo() {  
  console.log(num1+2)  
}
```

```
addTwo();
```

```
//10 in console
```

```
// => undefined
```

# What's the difference?

A function that has parameters is much more dynamic and can be passed around in your code more easily, since it does not rely on global variables.

Functions that do not return a value are generally used to manipulate global variables, but it comes down to style and taste.



# Exercise: Functions of FizzBuzz

If you haven't done FizzBuzz yet, lucky you! It is a staple of learning a new computer language. Print out the numbers between 1 and 100. If the number is divisible by 3, print "Fizz" instead. If the number is divisible by 5, print "Buzz" instead. If it is divisible by both 3 and 5, print "FizzBuzz".

1. Create a function `fizzBuzz`, that has no parameters, and accomplishes the instructions.
2. Create a function `fizzBuzzTwo`, that takes in two numbers, and does the FizzBuzz to those values. For example, 8 and 9, instead of 3 and 5.
3. Create a function `fizzBuzzSum`, which has two parameters, and returns the sum of the numbers that are not "Fizz", "Buzz" and "FizzBuzz".

# Why is this important?



“Ah ha!” , you say. “I already know all this stuff. This is really low level review!”

Well, my friend, it is the basis of callbacks, jQuery events and higher level programming. So, if this is fuzzy, really dig deep, have it re-explained and have it solid in your brain.

We are going to jump into callbacks next, so make sure you know this.

# Surprise! Functions are Data Types!

I know, right?!!

Along with Objects, Functions are complex data types in JavaScript.

What does that mean for you?

Well, that simply means that we can look at functions as values, like any other data type. If we think of them that way, callbacks are much simpler than we think they are.



# Callback: what the heck is it?



In simple terms, a callback is a function that is to be executed **after** another function has finished executing — hence the name 'call back'.





# Um...examples please?

Okay, in just a second. I'll finish explaining and then we'll go through some examples together.

As explained before, functions are objects (complex data types). Because of this, **functions can take functions as arguments, and can be returned by other functions.** Functions that do this are called **higher-order functions**. Any function that is passed as an argument is called a **callback function**.

# Why do we need this?

JavaScript is an event driven language, This means that instead of waiting for a response before moving on, JavaScript will keep executing while listening for other events. What happens when you run these?

```
function first(){  
  console.log(1);  
}  
  
function second(){  
  console.log(2);  
}  
first();  
second();
```

```
function first(){  
  setTimeout(function(){  
    console.log(1)}, 500);  
}  
  
function second(){  
  console.log(2);  
}  
first();  
second();
```

# So, JavaScript keeps on running, even with a code delay?

Yup! In the second example, it's not that JavaScript didn't execute our functions in the order we wanted it to, it's instead that **JavaScript didn't wait for a response from `first()` before moving on to execute `second()`**.



# So why show you this?

Because you can't just call one function after another and hope they execute in the right order.

Callbacks are a way to make sure certain code doesn't execute until other code has already finished execution.



## A simple jQuery callback function:

```
$ ("p").on ("click", function () {  
    alert ("The paragraph was  
clicked.");  
}) ;
```

```
$ ("p") .on ( )
```

Don't be scared. We know the selector part, then we use the .on() method. This allows for all kinds of events, like mouse hover, clicks, etc.

```
...("click", function() {  
    alert("The paragraph was clicked.");  
});
```

We then tell the .on() method what kind of event we want. I chose “click”. Next comes the callback. We are telling jQuery what we want to happen when we click the paragraph. We could do anything! Log something to the console, make the paragraph fly around, or make a sound. In this case, an annoying alert pops up and says “The paragraph was clicked.”

# Exercise set timeout

# Who cares? jQuery applications



---

# Objects

---

# What's an object?

# Key value pairs

# Array vs objects

**demo**

**exercise**

# Practical example

# Object constructors



**Why? And who cares?**

# Classes

**demo**

**What is “this”?**

**exercise**

# Practical example