Functional JavaScript

Putting the "fun" in functions

tl;dr

- Functions can be created in three ways, function statements, function expressions and arrow functions.
- They can always receive any number of parameters; too few or too many
- Default parameters can help with too few
- The rest operator can help with too many

Functions

Functions allow us to reuse code

- Declaring it

```
function func(p1, p2) {
  /* Do things with p1 and p2 here. */
  return anythingYouWant;
}
```

- Calling it

```
x = func(5, "hello");
```

pure vs impure functions

pure functions	impure functions

• Reads nothing outside
  ○ No reading globals
  ○ Predictable
• Changes outside
  ○ No writing globals
  ○ No modifying values passed to them
• Return value depends solely on input parameters
	• May reference globals
	• Re-running it might result in a different return value

There are four ways to declare functions

1. Instantiate a Function
2. Function statement
3. Function expression
4. Arrow function

```
f = new Function('arg1, arg2','body here');
```

1. Instantiate a function

2. Function statement

```
function func(p1, p2) {
  /* Do things with p1 and p2 here. */
  return anythingYouWant;
}
```

- Note: Function statements are always hoisted.

Consider ...

```
var x = 5;
var x = 'a string';
var x = new Date();
var x = ['Walt','Jesse','Skyler'];
var x = {};
```

What do you call the things on the right?

Expressions!!

JavaScript has a *function* expression

```
function (params) {
  body here
}
```

- Keyword function
- Name is optional
- Zero or more parameters
  - Bound by parentheses
  - Separated by commas
- Body
  - Bound by curly braces
  - Zero or more statements

3. Function expression

```
const func = function (p1, p2) {
 /* Do things with p1 and p2 here. */
  return anythingYouWant;
}
```

Functions are objects! You can ...

```
// Assign to a variable
var x = function () { doSomething() };
// Pass them as arguments
doSomethingElse(x);
// Return them from other functions
function foo() {
  return function () { doThings(); };
}
// Put them in arrays
var arrayOfFunctions = [ x, y, z ];
// ... and more!!
```

4. Arrow operator

```
func = (p1, p2) => {
  /* Do things with p1 and p2 here. */
  return anythingYouWant;
}
```

- Parentheses can be omitted if # of parameters is one
- Curly braces can be omitted if # of lines is one
  - If you do, the function implicitly returns the value of your one line

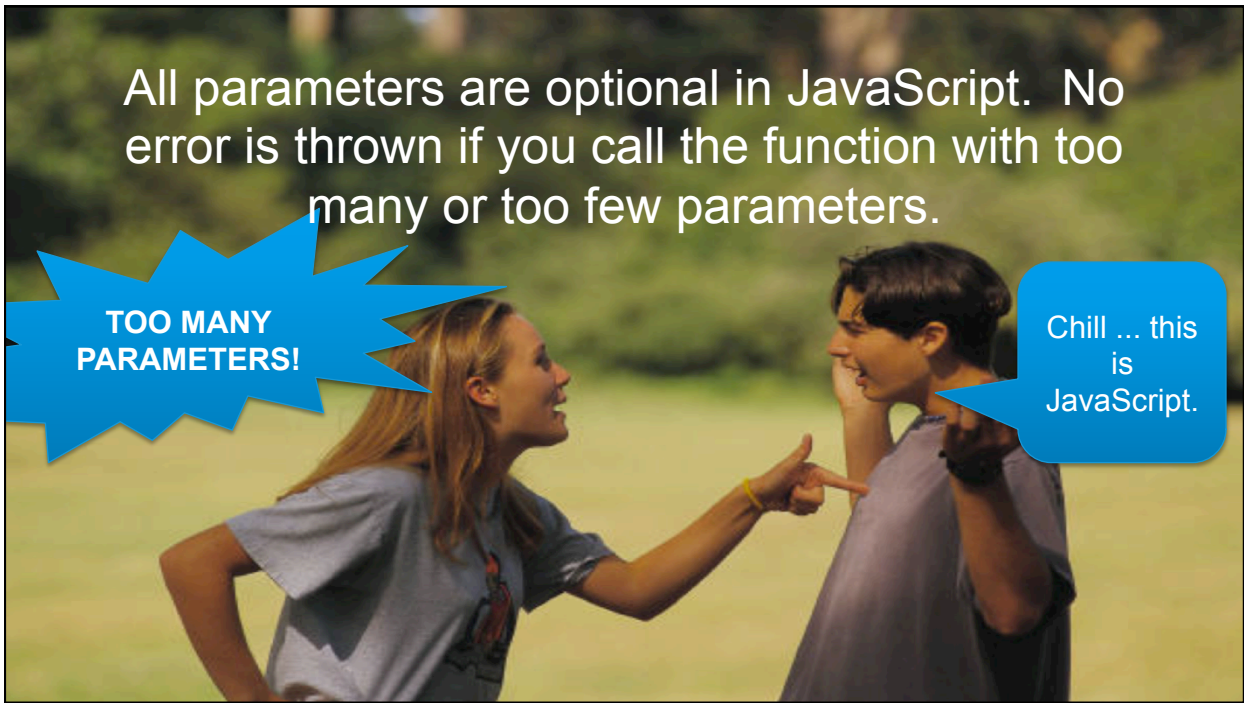
For example ...

```
const square = (x) => {
  return x * x;
};
let y = square(4);
```

- or more succinctly ...

```
const square = x => x * x;
```

Functions are variadic



There is an arguments variable

```
function sum() {
  let total = 0;
  for (var x=0 ; x<arguments.length ; x++) {
    total += arguments[x];
  }
  return total;
}
```

- arguments is array-like. But it isn't a real array.
- It's unexpected by other developers.

Rest parameters may help with too many arguments

```
function sum(...nums) {
  let total = 0;
  nums.forEach(x => total += x);
  return total;
}
```

- In this example, nums is a real array.
- It's right there in the signature so other developers know what to expect.

Default parameters may help with too few

- Just add default values in the function definition with an equal sign
- Syntax:

```
function (a="val1", b="val2" ...) { ... }
```

Traditional way

```
function foo(first, last, age) {
  if (! first)
    first = "John";
  last = last || "Doe";
  age = age || getVotingAge();
  // Do stuff with first, last, and age here
}
```

- Note: if first is falsey in any way, it'll use "John".

New way

```
function foo(first="John",
  last="Doe", age=getVotingAge()) {
  // Do stuff with first, last, and age here
}
```

- If you supply a value it'll be used. If not, the default value is.
- Allows you to pass in null, "", 0, or false as valid values and have them used.

tl;dr

- Functions can be created in three ways, function statements, function expressions and arrow functions.
- They can always receive any number of parameters; too few or too many
- Default parameters can help with too few
- The rest operator can help with too many