Functional JavaScript

Advanced JavaScript

JS

1

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

3

There are three ways to declare functions

1. Function statement
2. Function expression
3. Arrow function

4

1. Function statement

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

- Note: Function statements are always hoisted.

5

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!!

6

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

7

2. Function expression

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

8

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, foo, foo() ];
// ... and more!!
```

9

3. 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

ES 2015

10

For example ...

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

- or more succinctly ...

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

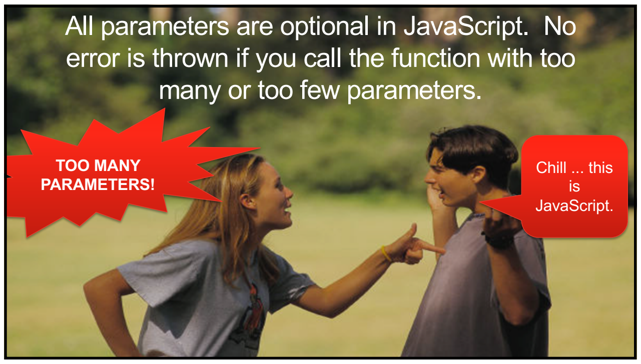
11

- Parameters that are primitives (string, number, bool) are passed by value.
  - There is a copy made.
  - If you change the copy, it doesn't change in the outside world.
- Parameters that are reference variables (objects, arrays) are passed by reference.
  - If you change the copy, it does change in the outside world.

12

Functions are variadic

14


All parameters are optional in JavaScript. No error is thrown if you call the function with too many or too few parameters.
TOO MANY PARAMETERS!
Chill ... this is JavaScript.

15

Rest parameters may help with too many arguments

```
function sum(x, y, ...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.

18

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" ...) { ... }
```

19

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".

20

New way

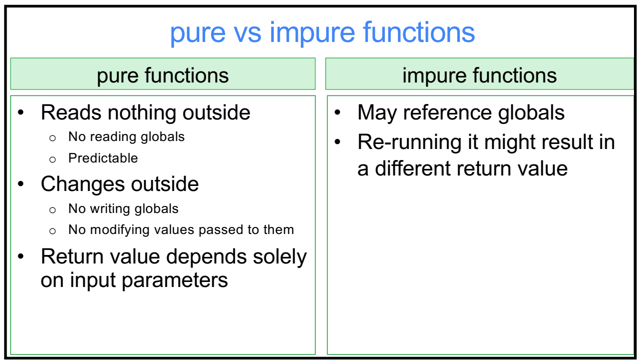
```
function foo(first="John",
  last="Doe", age=getVotingAge()) {
  // Do stuff with first, last, and age here
}
```

- 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.

21

Some function theory

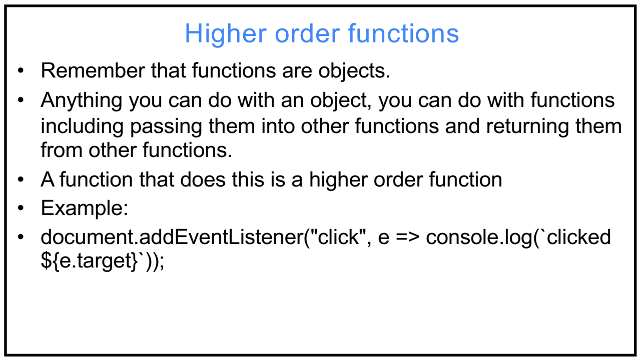
22



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

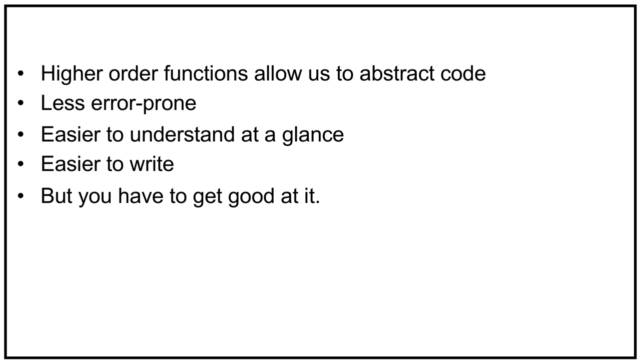
23



Higher order functions

- Remember that functions are objects.
- Anything you can do with an object, you can do with functions including passing them into other functions and returning them from other functions.
- A function that does this is a higher order function
- Example:
- document.addEventListener("click", e => console.log(`clicked ${e.target}`));

24



- Higher order functions allow us to abstract code
- Less error-prone
- Easier to understand at a glance
- Easier to write
- But you have to get good at it.

25

utils.js

```
export getPeopleYoungerThan = age => {
 return fetch('/people')
  .then(res => res.json())
  .then(people => people.filter(p => p.age < age))
}
```

26

store.js

```
import { createStore } from 'redux';

const reducer = (state,action) => ({
 ...mainReducer(state,action),
 person: personReducer(state.person,action),
 pic: picReducer(state.pic,action),
 addy: addyReducer(state.addy,action),
);
export default createStore(reducer, {});
```

27

Currying

- Breaking complex functions into simpler ones that return a function.
- Basically allows you to turn f(a, b, c, d) into f(a)(b)(c)(d)
- The second has four smaller functions that are each simpler

28

getTotal.js

```
export getTotal = cart => {
 let total = 0;
 for (row in cart) {
  total += row.qty * row.price;
 }
 return total;
}
```

getTotal.js

```
export getTotal = cart =>
 cart.reduce(row => row.qty * row.price, 0);
```

29

loggingMiddleware.js

```
import { addEntry } from './logging';

export const loggingMiddleware =
 store => next => action =>
  addEntry(action.message)
```

30

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

31