



Advanced Functions

Functions

Functions are so important, they're used everywhere

We've only touched the surface

1. Recap
2. Scope
3. Default arguments
4. Callbacks
5. Arrow functions



Function Declarations

Functions written with the function keyword are function declarations

```
function logIn(user, username, password) {  
    if (user.username === username && user.password === password) {  
        return true;  
    } else {  
        return false;  
    }  
}
```

Function Declarations

Parameters (or arguments) are passed to a function when it is called (or invoked)

```
logIn(user, "HelloKitty99", "ILoveCats446");
```

If no parameters are required, invoke the function with ()

```
logOut();
```



Function Declarations

Functions can return a value back to us when we call them

```
const fullName = createFullName("David", "Beckham");
```

```
fullName === "David Beckham";
```

Function Declarations

To the code editor for a recap of function declarations



Function Declarations

Any questions?



Scope

Variables which are declared *inside* a function are not accessible *outside* a function

Anything declared outside of every function is in the *global scope* and is accessible from anywhere

Scope

```
function createURL(location) {  
    const splitLocation = location.split("/");  
    const resource =  
splitLocation[splitLocation.length - 1];  
  
    return `mywebsite.com/${resource}`;  
}
```

Scope

```
function createURL (location) {  
  const splitLocation = location.split("/");  
  const resource = splitLocation[splitLocation.length - 1];  
  
  return `mywebsite.com/${resource}`;  
}
```

Trying to access `splitLocation` or `resource` here would result in an error

Scope

```
const domain = "mywebsite.com"

function createURL (location) {
  const splitLocation = location.split("/");
  const resource = splitLocation[splitLocation.length - 1];

  return `${domain}/${resource}`;
}
```

We can access `domain` from inside the function



Scope

To the code editor



Scope

Any questions?



Default arguments

Some functions don't need specific, different values every time

We can specify default values for arguments

This allows for a function which could be called with some *optional* arguments

Default arguments

```
function ageVerification(user, minimumAge = 18)
{
    return user.age >= minimumAge;
}
```

```
ageVerification({ age: 20 });           // true
ageVerification({ age: 20 }, 21);       // false
```



Default arguments

To the code editor



Default arguments

Any questions?



Callbacks

Confusing stuff!

A function which is *passed to another function*

That's it

A function which is passed to another function



Callbacks

setTimeout invokes a function after an amount of time

```
function sayHi(){  
  console.log("HI");  
}
```

```
setTimeout(sayHi, 1000);
```

Callbacks

```
function calculate(number, calc1, calc2) {  
  const result1 = calc1(number);  
  const result2 = calc2(result1);  
  
  return result2;  
}
```

```
function double(number) {  
  return number * 2;  
}
```

```
function decrease(number) {  
  return number - 1;  
}
```



Callbacks

```
function calculate(number, calc1, calc2) {  
  const result1 = calc1(number);  
  const result2 = calc2(result1);  
  
  return result2;  
}
```

```
function double(number) {  
  return number * 2;  
}
```

```
function decrease(number) {  
  return number - 1;  
}
```

`calculate(10, double, decrease); // 19`

`calculate(10, decrease, double); // 18`

Callbacks - inline

```
function calculate(number, calc1, calc2) {  
  const result1 = calc1(number);  
  const result2 = calc2(result1);  
  
  return result2;  
}
```

```
function double(number) {  
  return number * 2;  
}
```

```
function decrease(number) {  
  return number - 1;  
}
```

```
calculate(10, double, decrease); // 19
```

```
calculate(10, decrease, double); // 18
```

```
calculate(10, double, function(number) {  
  return number * 10;  
}); // 200
```

Callbacks

To the code editor



Callbacks

Any questions?

Callbacks are confusing!

Inline functions make them look even more confusing: lots of brackets, when am I calling, when am I passing?



Arrow functions

Now it's going to get even more confusing

Function declarations are the *old* way to create functions

Arrow functions are the new way

Less beginner friendly!

Arrow functions

From this:

```
function calculate(number, calc1, calc2) {  
  const result1 = calc1(number);  
  const result2 = calc2(result1);  
  
  return result2;  
}
```

```
function double(number) {  
  return number * 2;  
}
```

```
function decrease(number) {  
  return number - 1;  
}
```



Arrow functions

From this:

```
function calculate(number, calc1, calc2) {  
  const result1 = calc1(number);  
  const result2 = calc2(result1);
```

```
  return result2;  
}
```

```
function double(number) {  
  return number * 2;  
}
```

```
function decrease(number) {  
  return number - 1;  
}
```

To this:

```
const calculate = (number, calc1, calc2) => {  
  const result1 = calc1(number);  
  const result2 = calc2(result1);
```

```
  return result2;  
}
```

```
const double = number => number * 2;
```

```
const decrease = number => number - 1;
```

Arrow functions

=> instead of function

One-liners don't need return or {}

Functions with 1 parameter don't need () around them

It's all about reducing clutter



Arrow functions

To the code editor



Arrow functions

Any questions?

Note: arrow functions bring additional functionality, but this is difficult to explain now without context



Advanced functions

Any questions?