

# **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





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;
    }
}
```



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();
```



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

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



To the code editor for a recap of function declarations





Any questions?





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



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



```
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



```
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



To the code editor





Any questions?





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



```
function ageVerification(user, minimumAge = 18)
{
    return user.age >= minimumAge;
}
ageVerification({ age: 20 });  // true
ageVerification({ age: 20 }, 21);  // false
```



To the code editor





Any questions?





Confusing stuff!

A function which is passed to another function

That's it

A function which is passed to another function



setTimeout invokes a function after an amount of time

```
function sayHi(){
    console.log("HI");
}
setTimeout(sayHi, 1000);
```

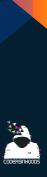


```
function calculate(number, calc1, calc2) {
    const result1 = calc1(number);
    const result2 = calc2(result1);
    return result2;
function double(number) {
    return number * 2;
function decrease(number) {
    return number - 1;
```





```
function calculate(number, calc1, calc2) {
                                           Callbacks
     const result1 = calc1(number);
     const result2 = calc2(result1);
     return result2;
function double(number) {
     return number * 2;
                                 calculate(10, double, decrease); // 19
function decrease(number) {
     return number - 1;
                                 calculate(10, decrease, double); // 18
```



```
function calculate(number, calc1, calc2) {
                                  Callbacks - inline
     const result1 = calc1(number);
     const result2 = calc2(result1);
     return result2;
                               calculate(10, double, decrease); // 19
                               calculate(10, decrease, double); // 18
function double(number) {
     return number * 2;
                               calculate(10, double, function(number) {
function decrease(number) {
                                     return number * 10;
     return number - 1;
                               }); // 200
```



To the code editor





Any questions?

Callbacks are confusing!

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



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!





#### 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;
```





#### 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;
```



=> 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





To the code editor





Any questions?

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



## **Advanced functions**

Any questions?



