JavaScript

ES6 & Beyond

Lambda Functions

Lambda vs Function Keyword Functions

Lambda Functions

- Lexically bound this
- No arguments array
- Much shorter syntax (Also multiple different syntaxes)

Function Keyword Functions

- Contextually bound this
- Has arguments array
- Longer syntax

This

The this keyword is a special variable that is only evaluated when its enclosing function is called.

It evaluates to the object that the called function is attached to *when it is called*.

Since we can freely move functions around and attach them to any object we please, the behavior of the this keyword can change

This

```
function getThis () {
  return this
}

// returns the global object,
// where all globally-scoped values
// are stored
console.log(getThis())

const receiver = {
  getThis: getThis
}

// returns the object stored in the
// receiver variable.
//
// Just by attaching the function to
// an object, we change the behavior of `this`!
console.log(receiver.getThis())
```

SCOPE

Scope

Scope refers to the variables that are accessible at a particular spot in your code.

An accessible variable is called “in-scope” at that location, while an inaccessible variable is called ”out-of-scope.”

Scope in JS

- Every module has a scope that includes global names (Math, console, process, etc.) and top-level definitions in the module
- Every function also has a scope which includes the parameters to the function and any top-level definitions inside the function body
- When a new scope is opened inside of another one, it inherits the outer scope

```
// module-level scope
// includes top-level definitions: x and addTen
const x = 10

function addTen (n) {
  // function-level scope
  // includes module scope: x and addTen
  // as well as parameter definitions: n
  return n + x
}

// back outside of the function, n is no longer defined
// error!
console.log(n)

```

Closures

Closures

A closure is a property of a function, and it refers to the chain of scopes that function has access to.

A function always has direct access to its closure, even if the function is copied, imported, or otherwise moved around.

A closure is essentially a function’s private state!