> Class 4

## Memory management and scopes

- Global scopes: Initializing a variable without keywords such as let, var or const inside a function.
  - In a webbrowser, top level variables are global however, in node.js this is not the case.
  - The keyword `let` is different from `var` because `var` is a function scope. If you declare variables with an if statement using both `let` and `var`  declarations, the `let` variable wont be defined outside the curly braces.
  - Box scope: The let and const declarations are box scopes meaning they scope to any curly braces and are not defined outside the curly braces.

#### Example 1:

```
for (let i = 0; i < 10; i++) {
  console.log(i); //This will work
}
console.log(i); //This will give you an error

//The syntactic sugar way of seeing this program

{
  let i = 0;
  for (; i < 10; i++) {
    console.log(i);
  }
}
console.log(i);
```

#### Example 2:

```
function outer(a) {
  let b = 20;
  let unused = 50;
  return function inner(c) {
    let d = 40;
    return `${a},${b}, ${c}, ${d}`;
  };
}
let i = outer(10);
let j = i(30);
console.log(j);
//Prints 10,20,30,40

/*
Closures are the combinations of functions and its references/pointers to
other variables. So whats happening is when we return a function, it
returns a closure which are poiners to its inner variables that are needed
```

```
to run properly.
*/
```

| identifier | memory | value |
|---|---|---|
| outer | 0x01 | 0xF1 |

| Memory | value |
|---|---|
| 0XF1 | def of outer |

`Example 3:`

```
function create_adder(x) {
  return (y) => {
    return x + y;
  };
}

const addten = create_adder(10);
const addeight = create_adder(8);

console.log(addten(5));
console.log(addeight(3));
```

`Example 4:`

```
function not(func) {
  return (...input) => {
    return !func(input);
  };
}
const is_even = (x) => {
  return x % 2 === 0;
};
const is_odd = not(is_even);

console.log(`24 is odd ${is_odd(24)}`);
console.log(`25 is odd ${is_odd(25)}`);
```