

In this lecture, we will discuss...

# **Defining Variables, Functions and Scope**



# Variables

```
var message = "hi";
```

- ✧ Variable definition should always start with 'var'
- ✧ No types are declared
  - JS is dynamically typed language
  - Same variable can hold different types during the life of the execution

# Functions

```
function a () {  
    ...  
}
```

Function name

# Functions

```
function a () {...}
```

```
var a = function () {...}
```

Value of function assigned,  
NOT the returned result!

No name defined



# Functions

```
function a () {...}
```

Defines function

```
a();
```

Executes function (aka invokes function)



# Functions

Arguments defined without 'var'

```
function compare (x, y) {  
    return x > y;  
}
```

# Functions

```
function compare (x, y) {...}  
var a = compare(4, 5);  
compare(4, "a");  
compare();
```

ALL LEGAL



# Scope

**Global**

- ✧ **Variables and functions defined here are available everywhere**

**Function  
aka lexical**

- ✧ **Variables and functions defined here are available only within this function**



# Scope Chain

- ✧ Everything is executed in an *Execution Context*
- ✧ Function invocation creates a new *Execution Context*
- ✧ Each *Execution Context* has:
  - Its own *Variable Environment*
  - Special 'this' object
  - Reference to its *Outer Environment*
- ✧ Global scope does not have an *Outer Environment* as it's the most outer there is



# Scope Chain

**Referenced (not defined) variable will be searched for in its current scope first. If not found, the Outer Reference will be searched.**

**If not found, the Outer Reference's Outer Reference will be searched, etc. This will keep going until the Global scope. If not found in Global scope, the variable is undefined.**



# Global

```
var x = 2;
```

```
A();
```

## Function A

```
var x = 5;
```

```
B();
```

Even though  
'B' is called  
within 'A'

## Function B

```
console.log(x);
```

'B' is defined  
within Global

Result: x = 2



# Summary

- ✧ Defining variables – dynamically typed
- ✧ Defining functions – creates its own scope (lexical)
- ✧ JS code runs within an Execution Context
- ✧ Scope chain is used to retrieve variables from Outer Variable Environments

