Objectives

- Reminders
- Discuss JS and front-end tools

JS as a Functional Language

- Fully functional programming language
  - "First lambda language to go mainstream" *D. Crockford*
- Pass a function wherever we can pass an object, value, array, etc.
- Essential parts of functions
  - Name
  - Arguments
  - Return Value
- Define anonymous functions (without an identifier)

Functions in General

NAZARBAYEV UNIVERSITY
SCHOOL OF SCIENCE AND TECHNOLOGY

$$f(x) = x^2 + 2x + 1$$

- An expression with a free variable like $x^2 + 2x + 1$ can become a function in a process called *abstraction*
- *Application* is the process of calling the function
  - Substitute some value or expression for x
- In the *lambda calculus* abstraction is indicated by $\lambda x.t$ and application is written without any parentheses or commas $f\ a$
  - We say that x is bound in t
- $\lambda x.x^2 + 2x + 1$ has the same meaning as the function above

```
function (x) {return x * x + 2 * x + 1;}
```

this parameter

- Function invocations get an implicit parameter `this`
- *Context* in which the function was called
- `this` is more complicated than in object-oriented languages
  - In Java, for example, instance methods are attached to some object
- Value of this depends on how the function was invoked: function, method, constructor, `apply` and `call`

```
function say() { console.log(this);}
var o1 = { name : "value" };
var o2 = { fn : say };

say();          // prints global object
o2.fn();        // prints o2
say.call(o1);   // prints o1
```

Object constructors

- A function becomes a *constructor* when it is invoked using the keyword `new`
- This is intended to achieve an object-oriented coding style

```
function Point() {
  this.x = 0;
  this.y = 0;

  this.translate = function(tx, ty) {
    this.x = this.x + tx;
    this.y = this.y + ty;
  }
}

var p = new Point();
```

var and let

- Scoping in Javascript works a bit differently than in other common languages
- Declare a variable with a `var` or `let` statement
  - Without a declaration keyword a variable is assumed to be global
  - `let` is a recent addition to the language to allow block scope

```
var i = 12345;

for (var i = 0; i < 10; i++) {}

console.log(i); // '10'

for (let i = 0; i < 100; i++) {}

console.log(i); // '10'
```

functions as objects (memoization)

NAZARBAYEV UNIVERSITY
SCHOOL OF SCIENCE AND TECHNOLOGY

- functions in JS are special kinds of objects
  - → they can have methods and properties attached
  - for example: the `.call` and `.bind` methods from before
- We can attach our own methods and data to functions
- *Memoization*: Caching values of an expensive computation

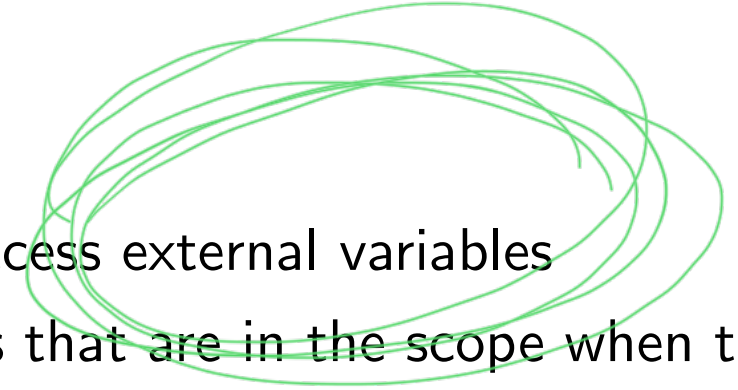
```
function frozen(n) {
  if (!frozen.ans) { frozen.ans = {}; }

  if (frozen.ans[n] !== undefined) {
    return frozen.ans[n];
  }
  c = Math.floor(10 * Math.random());

  frozen.ans[n] = c;
  return c;
}
```

Javascript Closures

- A *closure* or *lexical scoping* simply means that a function has access to the parent scope
- Self-invoked function that returns a function
- Important point is that the scope still exists even after the outer function has returned

```
var prvExample = (function () {
    var somedata = "Private Data";
    return function () {
      somedata = somedata + "!";
      return somedata;}
})();

console.log(prvExample());
console.log(prvExample());
```

Closure and Scope

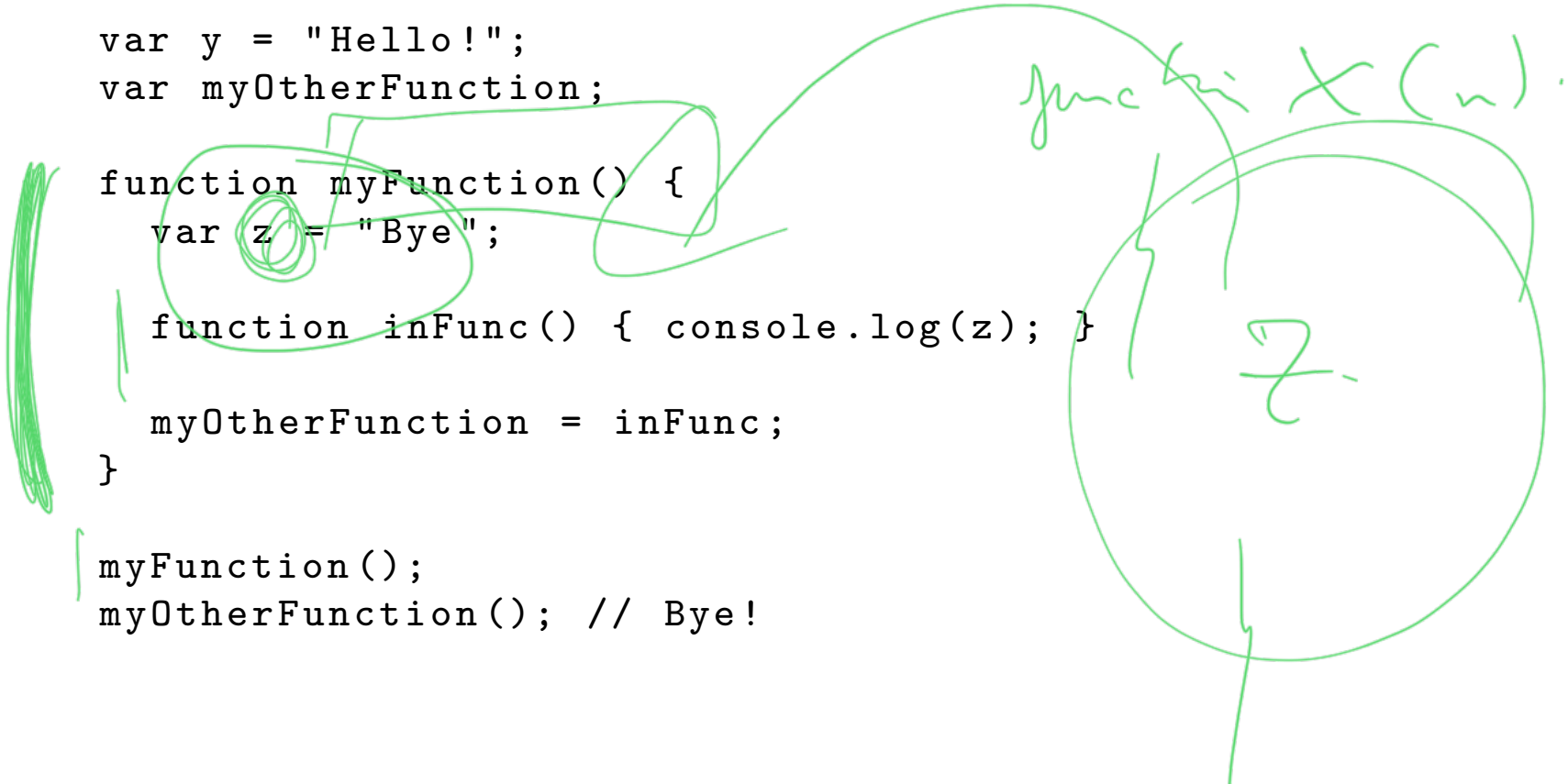


- *Closure* allows a function to access external variables
- A function can *see* all variables that are in the scope when the function is defined
- *Scope*: which identifiers are visible at different parts of a program

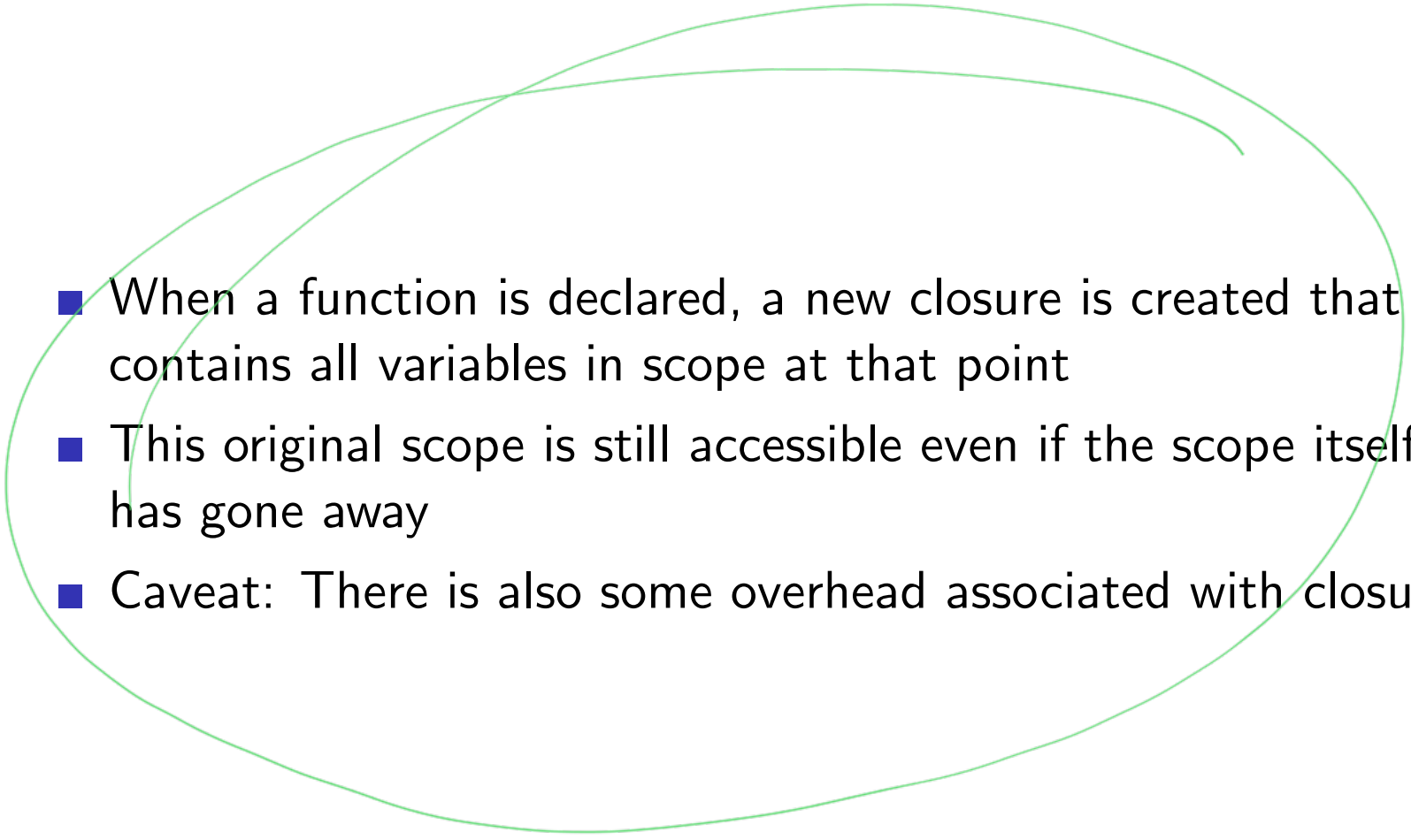
More Interesting Closure

- When `myFunction` executes, a reference to the inner function is copied to outer scope
- On last line this function reference is invoked and output shows that the variable `z` is still accessible

```
var y = "Hello!";
var myOtherFunction;

function myFunction() {
  var z = "Bye";

  function inFunc() { console.log(z); }

  myOtherFunction = inFunc;
}

myFunction();
myOtherFunction(); // Bye!
```



Closure and Scope



- When a function is declared, a new closure is created that contains all variables in scope at that point
- This original scope is still accessible even if the scope itself has gone away
- Caveat: There is also some overhead associated with closures

Code Nesting

NAZARBAYEV UNIVERSITY
SCHOOL OF SCIENCE AND TECHNOLOGY

- Lexical environment: Data structure used to map identifier names to specific variables
- Lexical Environments have to keep track of their parent environment

```
var A = "String 1";
var B = "String 2";

function f1() {
    var B = "String 3";
    var C = "String 4";

    function f2() {
        var D = "String 5";
        sayArguments(A,B,C,D);
    }

    f2();
}

f1();
```

Object Prototypes

- *Prototypes* are an example of delegation
- An object delegates the search for a property or method to its prototype
- In JS prototypes play a role similar to *classes* in object-oriented languages like Java and C++

Reviewing Objects

- Objects in JS are collections of named properties and values
  - Can be created with a literal notation or assigned to

```
var myObject = {
    x: 1,
    y: function () {},
    z: {}
}

myObject.x = 2;
myObject.y = [];
myObject.w = "New Property!";
```

Inheritance

- Code reuse is a central tenet of good software development
  - Duplication is a sign that you need to abstract some piece of functionality into its own method, class, or function
- *Inheritance* is one form of code reuse where the features of one object are extended with another
- In JS, inheritance has been achieved with prototyping

Prototyping

- Each object may have a reference to its own prototype
- If an object does not have a property a search is made on the prototype of the object

Prototype Chains

- The `prototype` property of an object is not directly accessible
- Can be set with the `Object.setPrototypeOf` method
- If `alice` does not have the requested property the search will be delegated to the prototype

```
var alice = {age : 20};
var bob = {rank : "corporal"};
var jim = {place : "new york"};

Object.setPrototypeOf(alice, bob);
Object.setPrototypeOf(bob, jim);

console.log(alice.rank);
console.log(alice.place);
```

Object Creation and Prototyping

- New objects can easily be created with the empty literal notation
- Some programmers, however, may be used to using class constructor
  - Initialize an object with some desired initial state
- If we will have multiple instances of similar objects then we don't want to assign everything manually
- `new` calls a function as a `constructor`

Constructors and Prototypes

- All functions have a prototype by default
- When new objects are created with the function the prototype of the objects is set to the prototype of the function
- The method `greet()` is a property of the prototype not the instances

```
function Person() {}
Person.prototype.greet = function () { return "Hello!";};

var person = new Person();
person.greet();
```

Constructors and Prototypes (Cont.)

- `this` references the objects that will be created
- Can set instance properties as below
- What should happen in case of a name clash?

```
function Person() {
    this.greet = function () { return "Personal Greeting";};
}
Person.prototype.greet = function () { return "Hello!";};

var person = new Person();
person.greet();
```

Run-Time Changes

- Recall that JS is a dynamic language
- We can change and add properties easily
- In the code below, changes can take effect even after the object is created

```
function Person() {}
Person.prototype.greet = function () { return "Hello!";};

var person = new Person();

Person.prototype.greet = function () { return "No thank you!";};

person.greet();
```

Code Linters

- Checks syntax and validates JS programs
- Common pitfalls that are not language errors
  - Automatic semicolon insertion
  - Switch statements
- Enforce good styling rules
  - Location of braces, indentation, space between a keyword or variable and parentheses
  - Compare this with the Python PEP 8 Style Guide

Linters for JS

- There are two common linters for Javascript: `jslint.com` and `jshint.com`
  - Both allow code to be pasted directly via their respective webpages
- Integrated with some of the task automation tools in node
- Also called *static analysis*

Linter Examples (switch)

- Switch statements can be confusing depending on how they are written

```
function test(v) {
    var result = "";
    switch (v) {
        case 0:
            result += "zero ";
            break;
        case 1:
            result += "one ";
        case 2:
            result += "two ";
            break;
        default:
            result += "default";
    }
    return result;
}

let a = [0, 1, 2, 3, 4];
console.log(a.map(test));
```

Linter Examples (`switch`) cont.

- Cases in a switch can share code blocks (fallthrough) if explicit `break` statements are not provided
- One example of the utility of a linter

```
let a = [0, 1, 2, 3, 4];
console.log(a.map(test));
// prints
// [ 'zero ', 'one two ', 'two ', 'default', 'default' ]
```

Linter Examples (semicolon insertion)

- JS will automatically insert some missing semicolons
- Intention of code below is to return the object literal but instead returns `undefined`
  - whitespace becomes important because the language will insert a semicolon on the same line as the return statement

```
function getPerson() {
    return
    {
        name: "Bob"
    };
}

getPerson(); // returns undefined
```

Node Package Manager

- `npm` is the package manager for node
- Npm, Inc. also maintains a large public registry for packages
  - Similar to maven for Java
- Facebook recently released its own Javascript package manager (yarn)

Basic usage of npm

- Once installed you may install and update modules, search the registry, and create your own packages via the command line tool
- `npm install <name>`: download the named package from the registry and install to your project
  - may install a package locally to your project (in the *node_modules* directory) or globally
- `npm init`: Initialize a new npm project, will open a dialogue for creating a new *package.json* file
  - navigate to new project directory and initialize
  - will step through a dialogue for setting package names, keywords, version numbers, repo, etc

Basic usage of `npm` cont.

- `npm search <name>`: search registry for modules that match the name
- `npm update <name>`: update an already installed package to a newer version
- For searching packages you can also visit the *npmjs.com* website