iOS Dev Accelerator Week2 Day3

- Photos Framework
- Web tools wednesday!!!

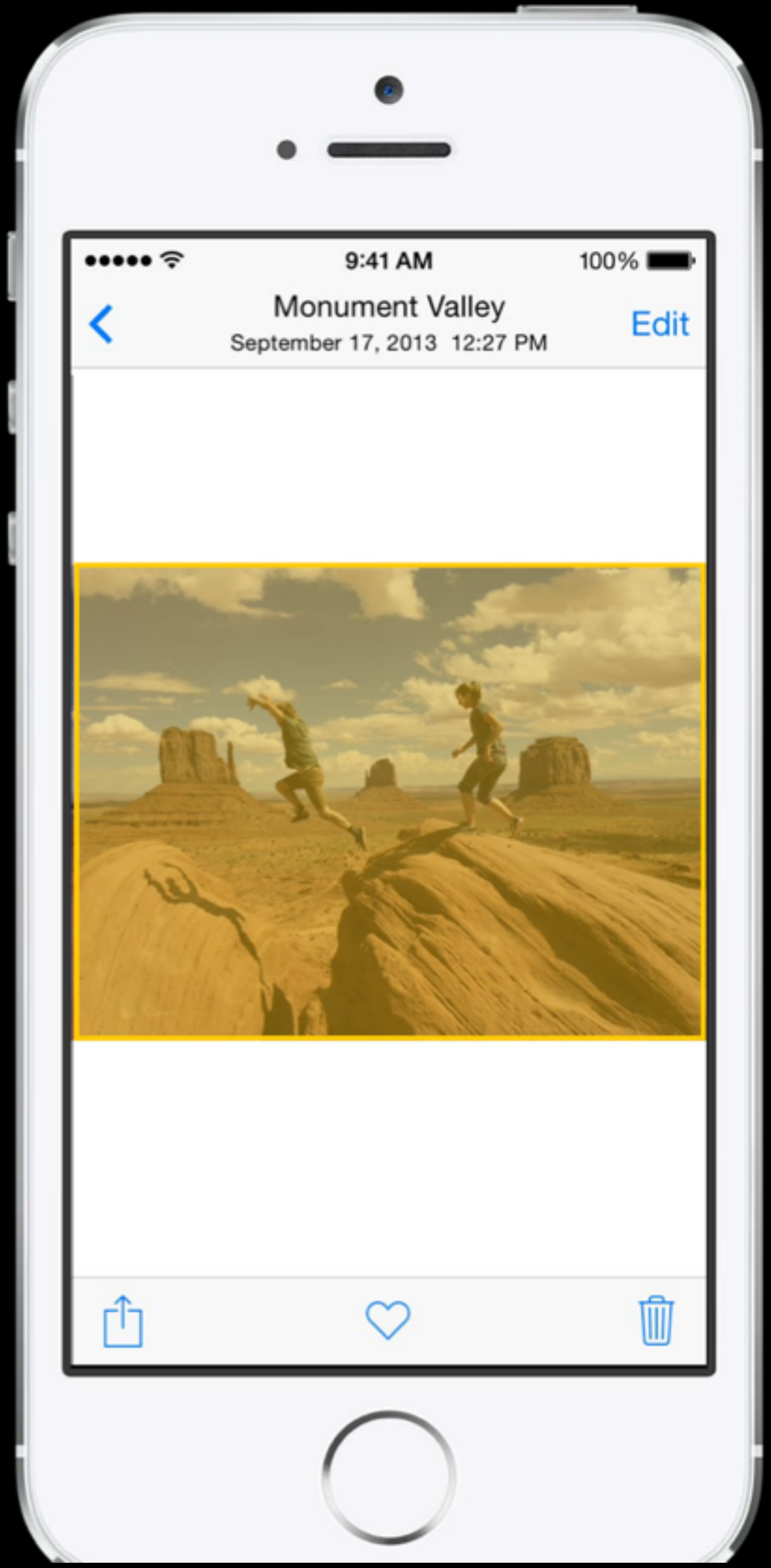
Photos Framework

Photos Framework

- New Apple Framework that allows access to photos and videos from the photo library.
- Also used for creating photo editing app extensions, a new feature with iOS8
- First-class citizen, you can create a full-featured photo library browser and editor on par with Apple’s Photos App.
- Intended to supersede ALAssetsLibrary

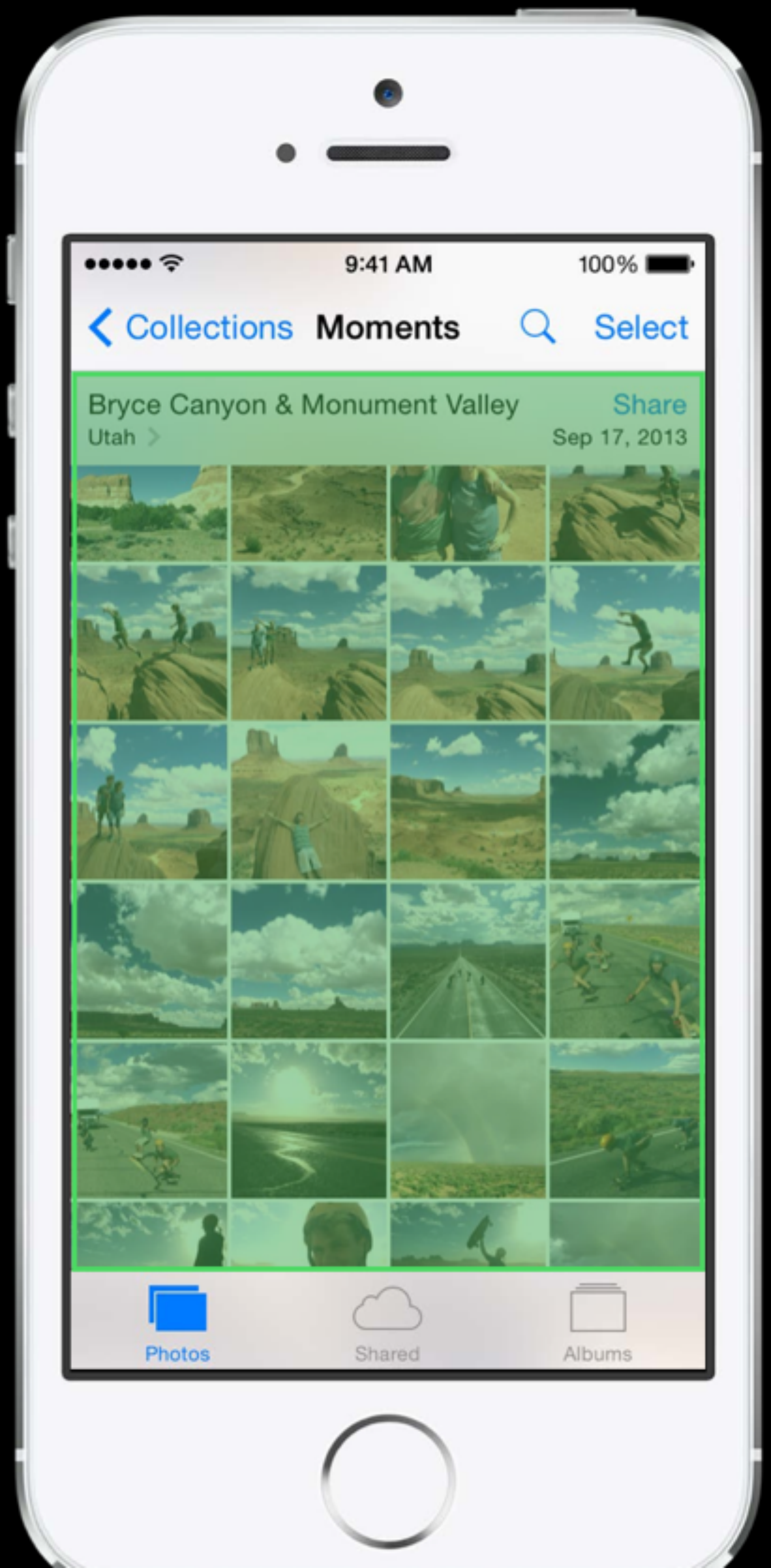
PHAsset

- The Photos framework model object that represents a single photo or video.
- Has properties for Media type, Creation date, Location, and Favorite.



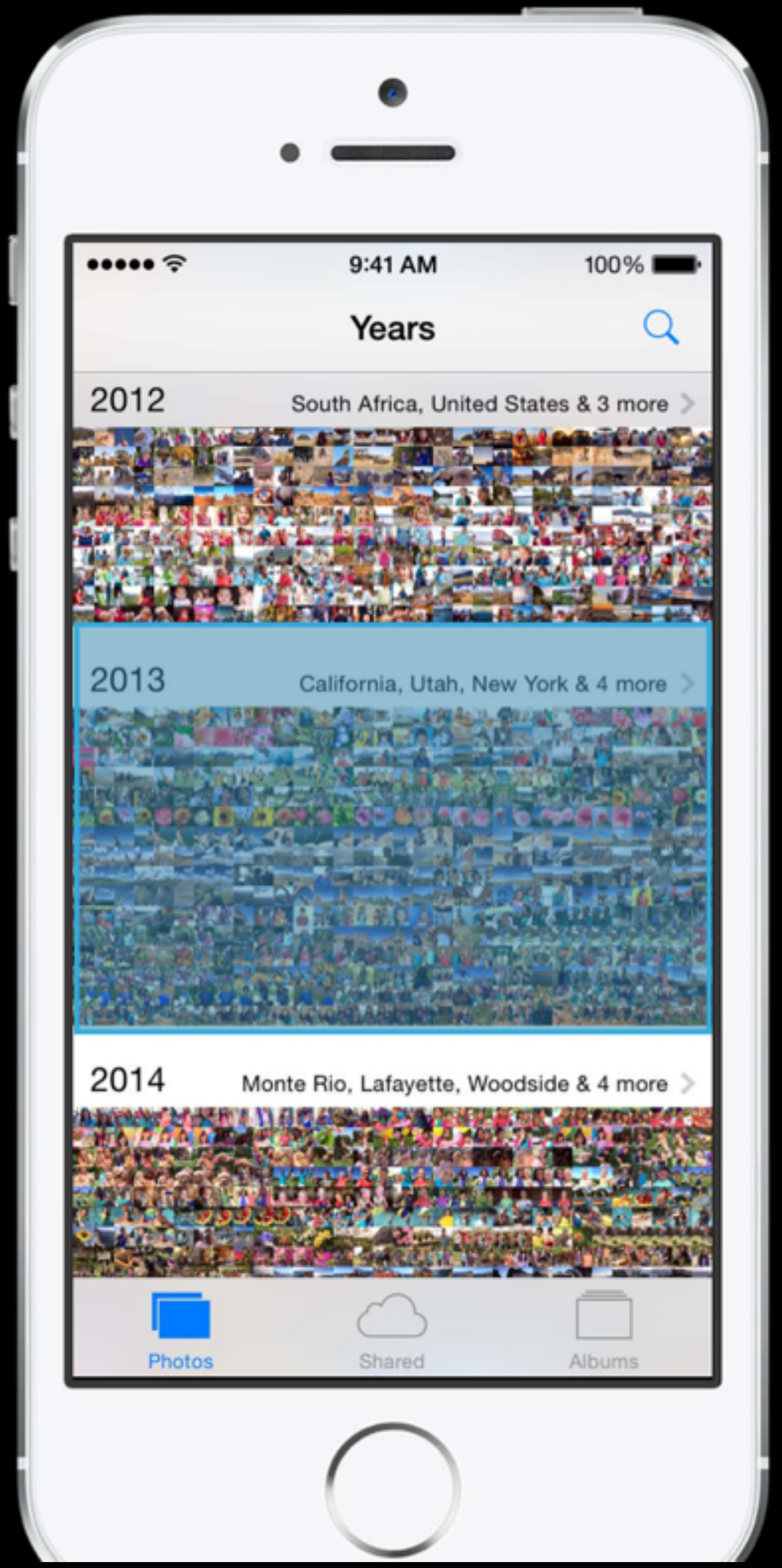
PHAssetCollection

- A Photos framework model object representing an ordered collection of assets.
- Albums, moments, and smart albums.
- Has properties for Type, Title, and Start and End Date.



PHCollectionList

- A Photos framework model object representing an ordered collection of collections.
- This can be a folder, moment, or year
- Has properties for Type, Title, and Start and End Date.


9:41 AM
100%
Years
2012
South Africa, United States & 3 more
2013
California, Utah, New York & 4 more
2014
Monte Rio, Lafayette, Woodside & 4 more
Photos
Shared
Albums

Fetching Model Objects

- You fetch via class methods on the models:
  - PHAsset.fetchAssetsWithMediaType(PHAssetMediaType.Photo, options:nil)
  - PHAssetCollection.fetchMomentsWithOptions(nil)
- Collections do not cache their contents, so you still have to fetch all the assets inside of it. This is because the results of a fetch can be very large, and you dont want all of those objects in memory at once.

PFFetchResult

- Results returned in a PHFetchResult
- Similar to an Array.

assets[n]
Fetch result

Making Changes

- You can favorite a photo and add an asset to an album
- You cannot directly mutate an asset, they are read only (thread safe!)
- To make a change, you have to make a change request.
- Theres a request class for each model class:

```
PHAssetChangeRequest
PHAssetCollectionChangeRequest
PHCollectionListChangeRequest
```

Making Changes

```
func toggleFavorite(asset : PHAsset) {

    PHPhotoLibrary.sharedPhotoLibrary().performChanges({

        //create a change request object for the asset
        var changeRequest = PHAssetChangeRequest(forAsset: asset) as
      PHAssetChangeRequest
        //make your change
        changeRequest.favorite = !changeRequest.favorite

        }, completionHandler: { ( success : Bool,error : NSError!) -> Void in

        //asset change complete
        })

}
```

Making New Objects

Create via creation request

```
var request = PHAssetChangeRequest.creationRequestForAssetFromImage(UIImage())
```

Placeholder objects

```
var placeHolder = request.placeholderForCreatedAsset
```

- Reference to a new, unsaved object
- Add to collections
- Can provide unique, persistent `localIdentifier`

Getting to the actual data

- Many different sizes of an image may be available or different formats of a video
- Use PHImageManager to request images/videos
- Request an image based on target size for displaying
- Request a video based on the usage
- Asynchronous API, because you dont know how long it will take to load the data, it could be very expensive
- Will optionally retrieve the data from the network if its only on iCloud
- Use a PHCachingImageManager when displaying a collection of images for better performance.

Requesting an Image

```
let manager = PHImageManager.defaultManager()

manager.requestImageForAsset(photo,
                            targetSize: cellSize,
                            contentMode: PHImageContentMode.AspectFill,
                            options: nil,
                            resultHandler: {(result : UIImage!, [NSObject : AnyObject]!) -> Void in
                                if result {
                                    imageView.image = result
                                } else {
                                    //tell user something went wrong
                                }
    })
```

Advanced Image Request

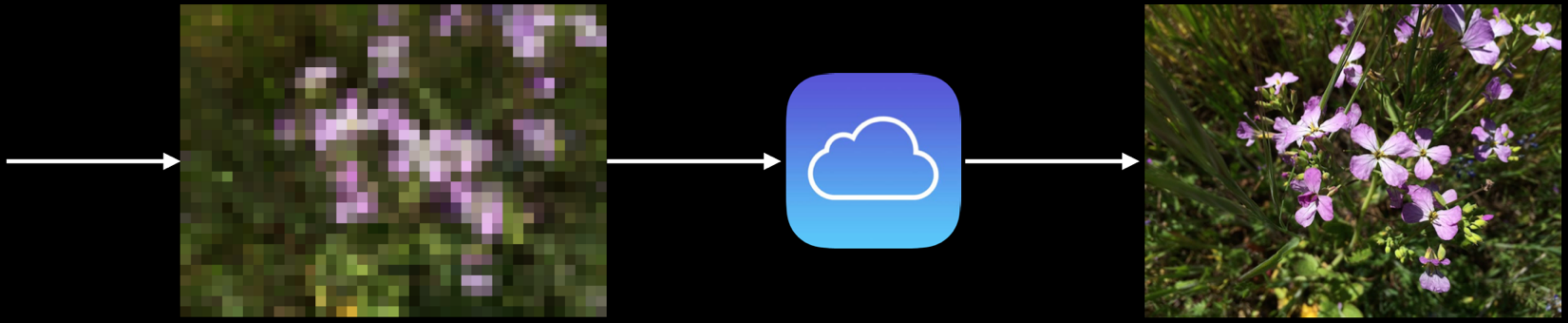
```
var options = PHImageRequestOptions()

options.networkAccessAllowed = true
options.progressHandler = {(progress : Double, error : NSError!, degraded : UnsafePointer<ObjCBool>,
[NSObject : AnyObject]!) in
    //update visible progress UI
    }

//use your options to control the request behavior
manager.requestImageForAsset(photo,
    targetSize: cellSize,
    contentMode: PHImageContentMode.AspectFill,
    options: options,
    resultHandler: {(result : UIImage!, [NSObject : AnyObject]!) -> Void in
```

Advanced Image Request

```
[manager requestImageForAsset: ... ^(UIImage *result, NSDictionary *info) {
    // This block can be called multiple times
}];
```



Demo

Web Tools Wednesday Part 2: JavaScript

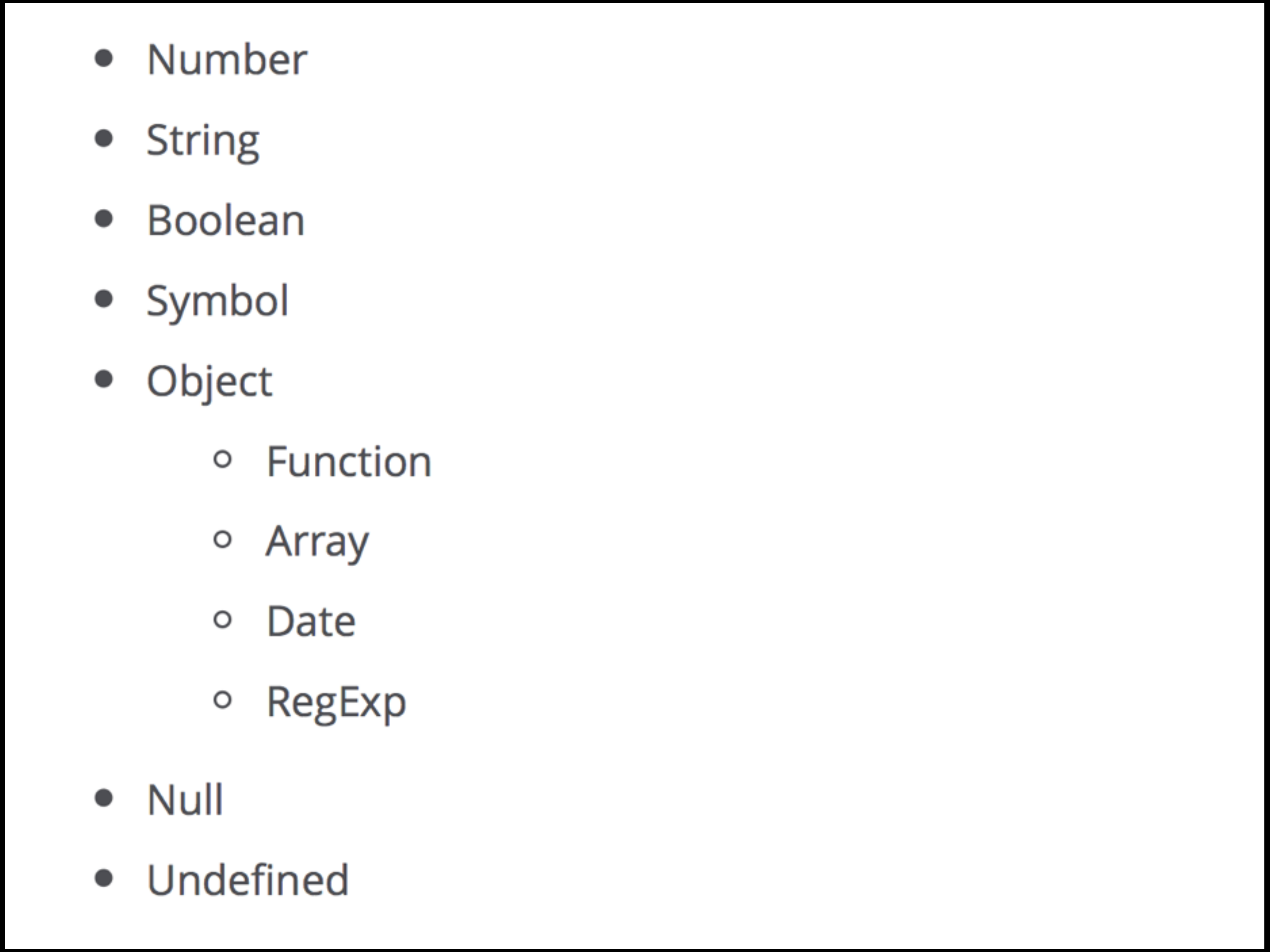
Language History

- JavaScript was invented in 1995 by Brendan Eich, who worked at Netscape.
- It was supposed to be called LiveScript, but they changed it at the last minute to JavaScript to ride off the popularity of Java at the time. Even though the languages are not directly related.
- It is considered a ‘scripting language’, but you don’t really need to care about that.

Language Details

- Object Oriented
- Dynamically Typed (more on this later)
- JavaScript does not have classes, it has object prototypes.
- Functions are first class citizens in Javascript, just like in Swift. So you can think of functions as objects.

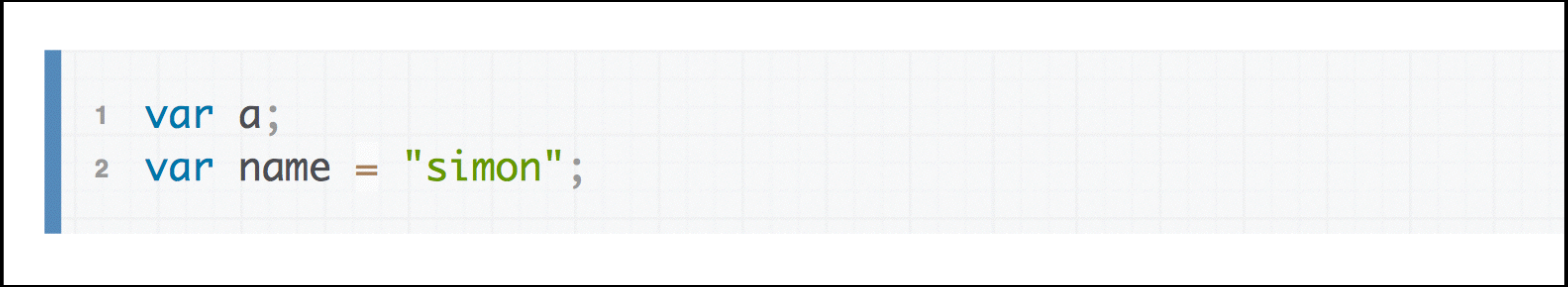
Types in JavaScript



- Number
- String
- Boolean
- Symbol
- Object
    - Function
    - Array
    - Date
    - RegExp
- Null
- Undefined

Variables

- Variables in JavaScript are declared using the var keyword, just like in swift!



```
var a;
var name = "simon";
```

- a variable declared without a value has a type of 'undefined'

Objects in JavaScript

- Objects in JavaScript are just simple collections of name-value pairs. So pretty much a dictionary in Swift/Objective-C.
- In JavaScript, everything is an object except the primitive data types.
- Objects in JavaScripts can have properties and methods (kind of like a class in Swift)
- You can define objects with an object literal:

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

Everything is an Object

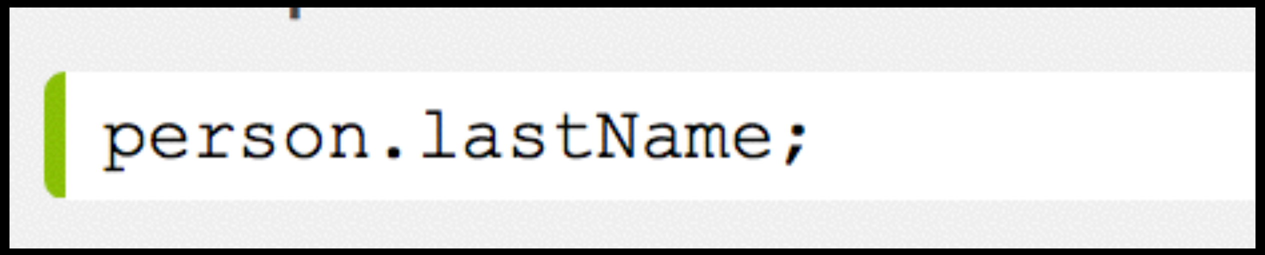
In JavaScript almost everything is an object.

Even primitive data types (except null and undefined) can be treated as objects.

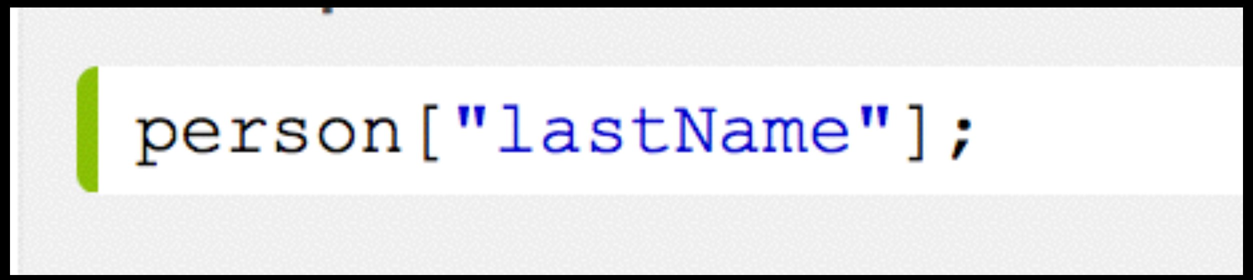
- Booleans can be objects (or primitive data treated as objects)
- Numbers can be objects (or primitive data treated as objects)
- Strings are also objects (or primitive data treated as objects)
- Dates are always objects
- Maths and Regular Expressions are always objects
- Arrays are always objects
- Even functions are always objects

Object properties

- You can access the properties of an object with dot notation or subscript []:



```
person.lastName;
```

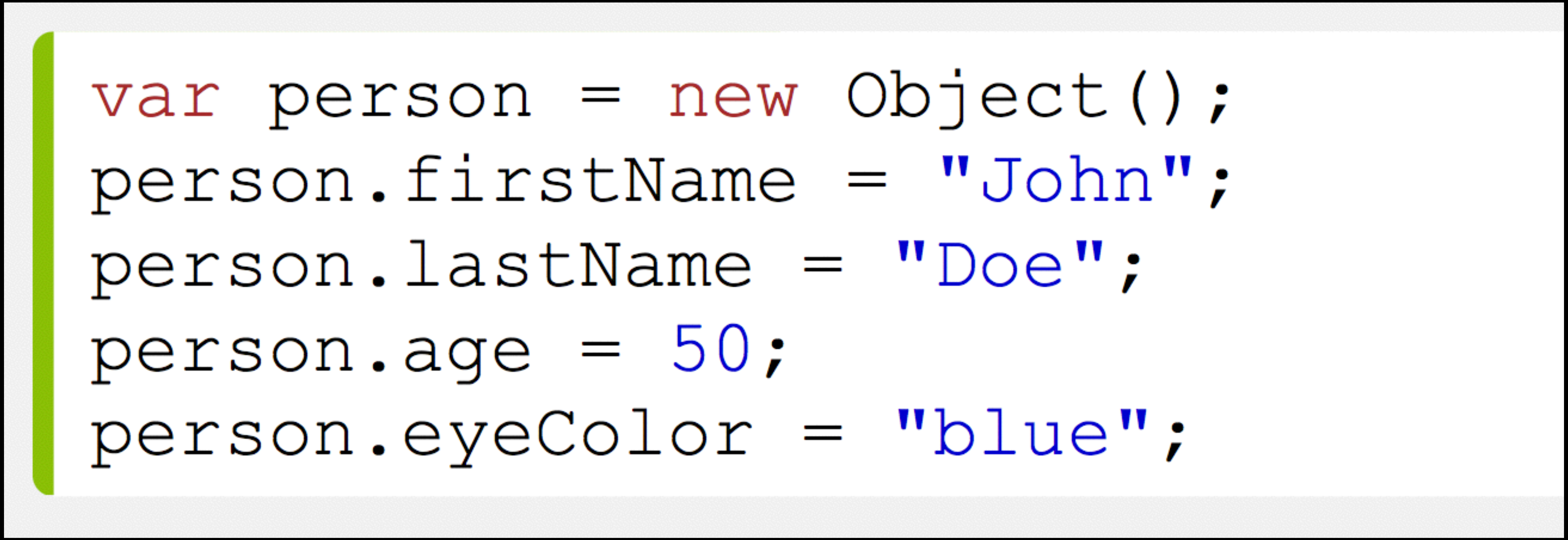


```
person["lastName"];
```

Creating objects

- Three ways to create new objects in JavaScript:
  - Define and create a single object using the object literal {}
  - Define and create a single object using the keyword new
  - Define an object constructor, and then create objects with the constructor.
- Objects are mutable and passed by reference, not copy.

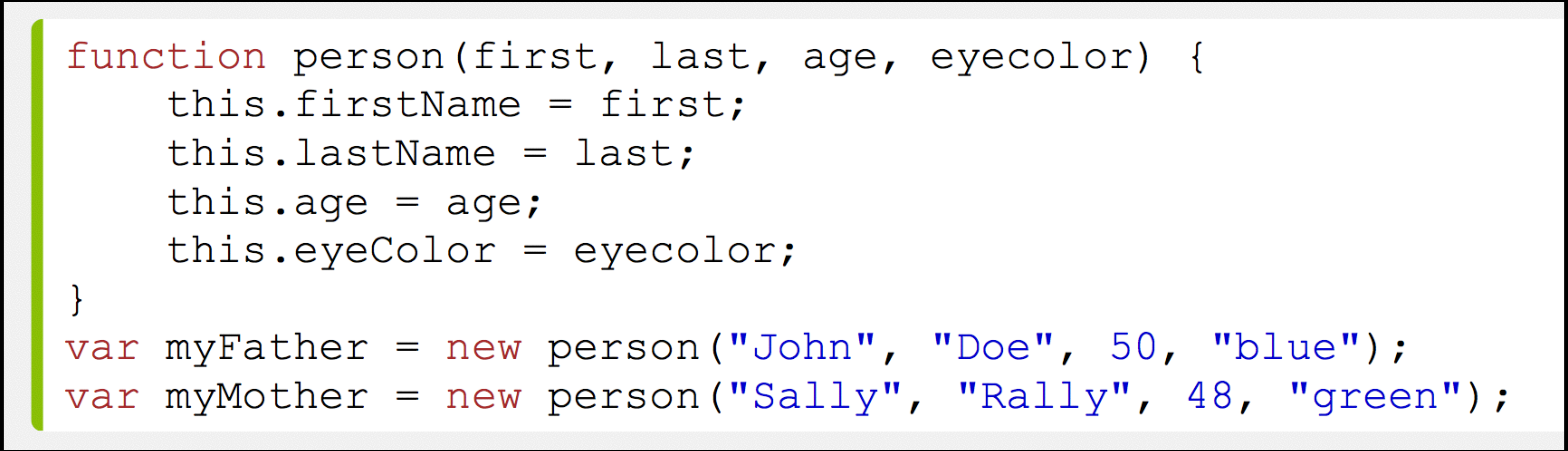
Keyword New



```
var person = new Object();
person.firstName = "John";
person.lastName = "Doe";
person.age = 50;
person.eyeColor = "blue";
```

Object Constructor

- Defining an object constructor is kind of like defining a class in Swift:

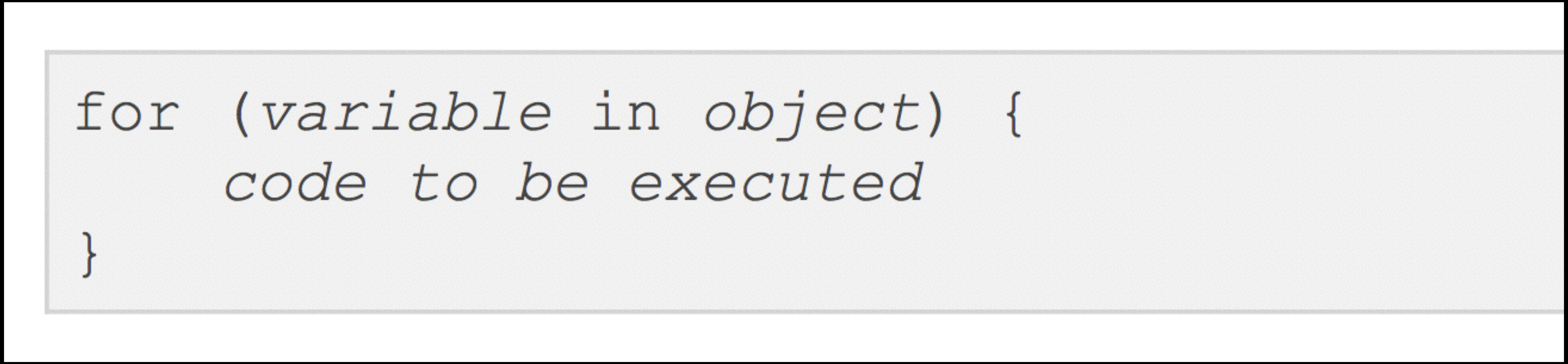


```
function person(first, last, age, eyecolor) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.eyeColor = eyecolor;
}
var myFather = new person("John", "Doe", 50, "blue");
var myMother = new person("Sally", "Rally", 48, "green");
```

- this in JavaScript is just like self in Swift

Object for…in Loop

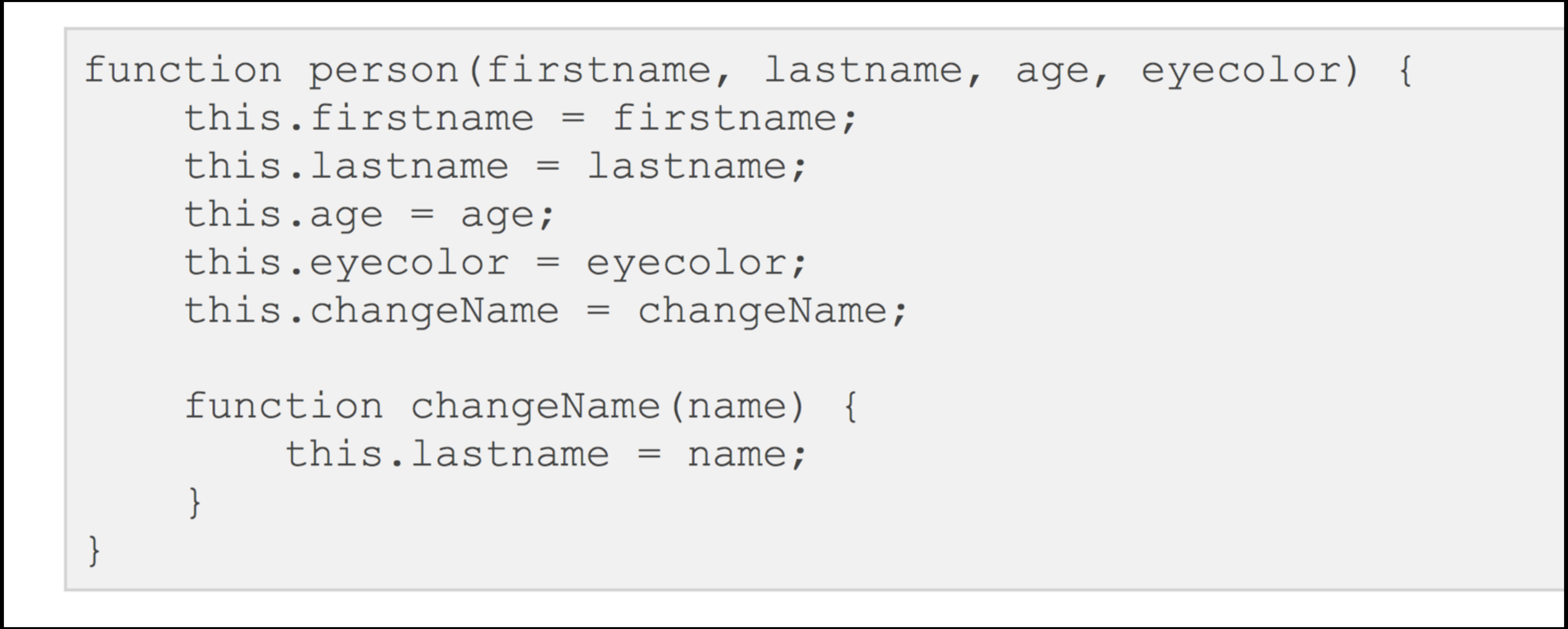
- You can use the for…in statement in JavaScript to loop through all the properties of an object:



```
for (variable in object) {
    code to be executed
}
```

Adding methods to Object

- Defining methods for an object is done inside the constructor:

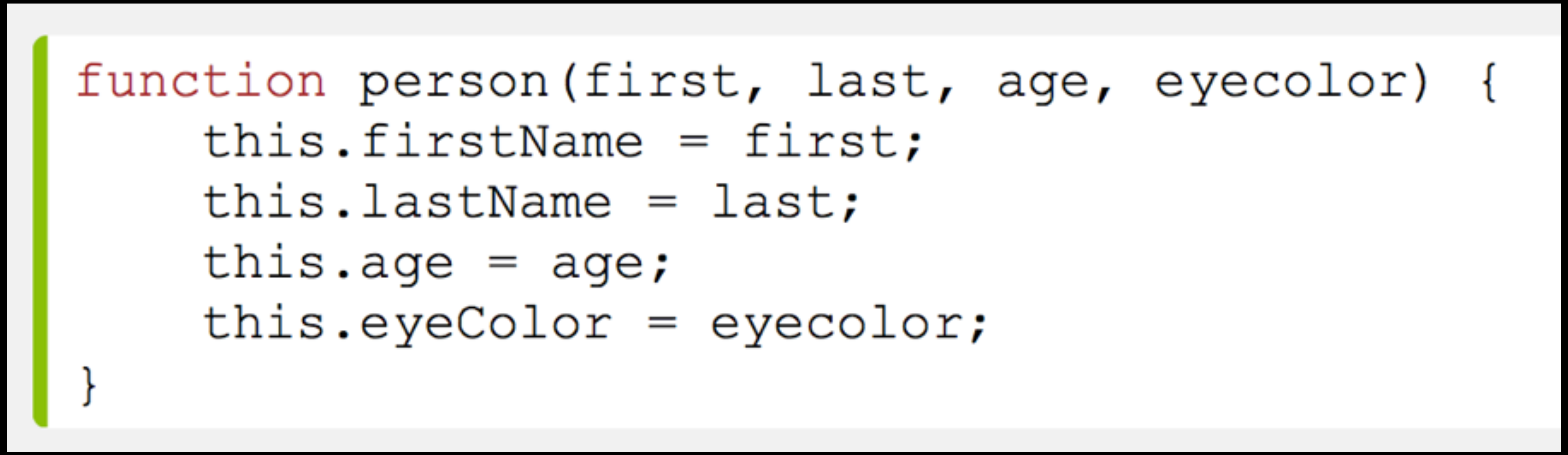


```
function person(firstname, lastname, age, eyecolor) {
    this.firstname = firstname;
    this.lastname = lastname;
    this.age = age;
    this.eyecolor = eyecolor;
    this.changeName = changeName;

    function changeName(name) {
        this.lastname = name;
    }
}
```

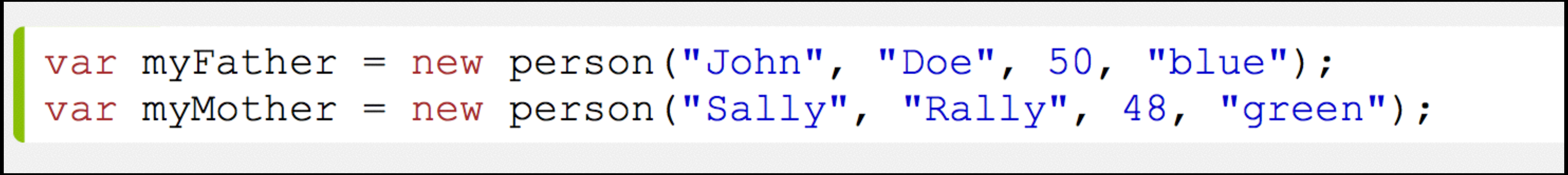
Object Prototypes

- All JavaScript objects have a prototype. The prototype is also an object.
- All JavaScript objects inherit their methods and properties from their prototype
- A prototype is created when you create an object constructor function:



```
function person(first, last, age, eyecolor) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.eyeColor = eyecolor;
}
```

- Now you can use the new keyword to create new objects from the same prototype person:



```
var myFather = new person("John", "Doe", 50, "blue");
var myMother = new person("Sally", "Rally", 48, "green");
```