The JavaScript Programming Language
Experiment in a browser console

• Open the console in Chrome
• Go to console in dev tools and experiment
  var x=4;
  x
Numbers

• How many number types are there?
  – One: 64-bit floating point
  – No integers

• Experiment
  x=4
  x+=.33
  x

• How do you get the integer of x?
  – Math.floor(x)  // truncates
  – Math.round(x)  // if you prefer to round
NaN

- Special number: Not a Number
- Result of undefined or erroneous operations
- Toxic: any arithmetic operation with NaN as an input will have NaN as a result
- NaN is not equal to anything, including NaN
- Experiment
  ```javascript
  var r=12
  var t=NaN
  x+r
  x+t
  x+r+t
  ```
Numbers

• How do you convert a string into a number.
  – E.g. "433"?
  – What is "433"+1?
  – What happens when converting "433a" to a number?

• Experiment:
  Number("443")
  Number("443")+x
  "443"+ x
  +"443"+x
String Methods

- `charAt` (or just use array indexing `[ ]`)
- `concat`
- `indexOf`
- `lastIndexOf`
- `length`
- `match`
- `replace`
- `search`
- `slice`
- `split`
- `substring`
- `toLowerCase`
- `toUpperCase`
String function

- String(value)
- Same as value.toString()
- Same as value + ""

• Converts value to a string
• Experiment:
  - String(x)
  - x.toString()
  - x+""
True or False?

- 0
- 4
- "hi"
- `{color: "red"}`
- null
- undefined
- -0
- -1
- "" // the empty string
- [3, 2, 5]
• Equal and not equal
• == and != does type coercion
  – What is type coercion?
• To avoid surprises, it is better to use === and !==, which do not do type coercion.

• Experiment:
  3=="3"
  3==3
  3==="3"
  3===3
  3!="3"
  3!=="3"
null vs undefined vs not defined

- **Experiment**
  ```javascript
  typeof(null)
  g
  var g
  g
  g=null
  g
  ```
- **not defined**
  - A variable that does not yet exist
- **undefined**
  - A different kind of absence of value
  - A variable that has not been initialized
  - Or when querying an array element or object property that does not exist.
- **null**
  - A special keyword
  - Indicates a value of "no value"
  - Is of type object, indicating "no object"
Try ... catch

- Function decide() returns "yes", "no", or throws an exception. Write code to call decide(), and print the result to console.log. If there is an exception, print "Error: " and the caught exception (it will be a string).
  - Hint: you will have a try block (i.e. {}) and a catch block

```javascript
try {
    console.log(decide());
} catch (e) {
    console.log("Error: "+e);
}
```
• Write a function named circumference that returns a circumference when given a diameter
  – Hint: use Math.PI * the diameter
• Define it in two different ways.

```javascript
function circumference(d) {
    return Math.PI * d;
}

var circumference = function(d) {
    return Math.PI * d;
}
```
Arguments

• Extra arguments to a function are ignored.
• If arguments are not supplied, missing values are `undefined`.
• There is no automatic argument type checking.
• All function arguments are accessible via the `arguments` function variable.
  – `arguments[0]` is the first argument
  – `arguments.length` gives the number of arguments
Var, and variable scope

- Always use var to define a new variable
  - global var's are global
  - var's within a function are local
  - using function variables in a function without var is sloppy and dangerous

- Experiment:
  ```javascript
  scope = "global";
  function checkscope() {
    scope = "local";
    scope2 = "local";
    var scope3 = "local";
    return [scope, scope2, scope3];
  }
  checkscope()
  scope
  scope2
  scope3
  ```
Var, and variable scope

- Always use var to define a new variable
  - global var's are global
  - var's within a function are local
  - using function variables in a function without var is sloppy and dangerous

- Experiment:
  ```javascript
  scope = "global";
  function checkscope() {
    scope = "local";
    scope2 = "local";
    var scope3 = "local";
    return [scope, scope2, scope3];
  }
  checkscope() // ["local", "local", "local"]
  scope // "local" - global variable changed
  scope2 // "local" - local variable clutters global namespace
  scope3 // ReferenceError – not defined
  ```
Defining functions

• As a function
  – E.g. function calculator() {... OR var calculator = function() {...

• Within an object (called a method)
  var counter = {
    "increment": function () {this.count++},
    "report": function () {console.log("count: "+this.count)}
  }

• As inner functions
  function multiplyAbsolute(number,factor){
    function multiply(number){
      return number*factor;
    }
    if(number<0)
      return multiply(-number);
    else
      return multiply(number);
  }

Example source: Eloquent JavaScript by Marijn Haverbeke
Variable scope

• As a function
  — E.g. `function calculator() { ... OR var calculator = function() {...

• Within an object (called a method)
  `var counter = {
    "increment": function () {this.count++},
    "report": function () {console.log("count: "+this.count)},
  },`

• As inner functions
  `function multiplyAbsolute(number,factor){
    function multiply(number){
      return number*factor;
    }
    if(number<0)
      return multiply(-number);
    else
      return multiply(number);
  }`

Which `factor` is this?
Variable scope

• As a function
  – E.g. function calculator() {... OR var calculator = function() {...

• Within an object (called a method)
  var counter = {
    "increment": function () {this.count++},
    "report": function () {console.log("count: "+this.count)},

• As inner functions
  function multiplyAbsolute(number,factor){
    function multiply(number){
      return number*factor;
    }
    if(number<0)
      return multiply(-number);
    else
      return multiply(number);
  }

Which factor is this?
The argument in the outer function.
Variable scope

• As a function
  – E.g. function calculator() {... OR var calculator = function() {...

• Within an object (called a method)
  var counter = {
    "increment": function () {this.count++},
    "report": function () {console.log("count: "+this.count)},
  }

• As inner functions
  function multiplyAbsolute(number,factor){
    function multiply(number){
      return number*factor;
    }
    if(number<0)
      return multiply(-number);
    else
      return multiply(number);
  }

Which number is this?
Variable scope

- As a function
  - E.g. `function calculator() {... OR var calculator = function() {...`

- Within an object (called a method)
  
  var counter = {
  "increment": function () {this.count++},
  "report": function () {console.log("count: "+this.count)},
  }

- As inner functions
  
  function multiplyAbsolute(number,factor){
    function multiply(number){
      return number*factor;
    }
    if(number<0)
      return multiply(-number);
    else
      return multiply(number);
  }

Which number is this?
The argument in the inner function.
Closure

• The scope of an inner function continues even after the parent functions have returned.

• This is called closure.
function ping() {
  console.log("Ping");
  var times=0;
  pong = function() {
    console.log("Pong "+ (++times));
  };
}

Experiment:
• Try pong() first
• Then ping()
• pong() again
• pong() again
function ping() {
  console.log("Ping");
  var times=0;

  pong = function() {
    console.log("Pong " + (++times));
  }
}

Experiment:
• Try pong() first
• Then ping()
• pong() again
• pong() again

pong() has access to ping()'s variables, even after ping() has returned.
function makeAdder(amount) {
    return function(number) {
        return number + amount;
    };
}

var addTwo = makeAdder(2);
addTwo(3); // what does this return
var addSix = makeAdder(5);
addSix(8); // what does this return

• Where are the 2 (addTwo) and 5 (addSix) stored?
• CLOSURE: Inner functions have access to the arguments, variables, and declared inner functions of their outer functions.

Example source: Eloquent JavaScript by Marijn Haverbeke
Exercise

Develop a function `countBy(increment)` that returns another function which will return numbers incremented by `increment`.

E.g.

```
> y = countBy(3)
y() 3
y() 6
y() 9
y() 12
y = countBy(8)
y() 8
y() 16
y() 24
```

Example to work from:

```javascript
function makeAdder(amount) {
  return function(number) {
    return number + amount;
  };
}
```
Function definition order – Order matters

```javascript
var f = function () {return " one";}
f() //What does this return?
f = function () {return " two";}
f() //What does this return?

// If you want to extend f()
f = function () {f() + " three";} //Doesn't work, unbounded recursion

// Rather
f = function () {return " two";}
...
var oldf = f;
f = function () {return oldf() + " three";}
f() //What does this return?
```
Object Literals

• What is wrong with these? (Or OK)
  
  var empty = {};
  var two = {x=1, y=4};
  var b = {
  
  kind: book,
  book title: "Cloud",
  "sub-title": "Utility computing",
  for: "Geeks"
  author: {
    "first name": "Fred",
    surname: "Noodle"
  }
  }
  }
Object Augmentation

Adding new properties to an object

E.g. myo={color:"red"}

• How do you add {size:"big"}?
  myo.size="big"

• How do you add {"initial shape":"round"}? 
  myo["initial shape"]="round"

• How do you remove size from the object? 
  delete myo.size
Arrays v Objects

• Use objects when the names are arbitrary strings.
  – E.g.
    √ {color: "red", size: "big", location: "office"}
    X {1: "Fido", 2:"Shadow", 3:"Clifford"}

• Use arrays when the names are sequential integers.
  – E.g.
    √ ["Fido", "Shadow", "Clifford"]
// Define a constructor function to initialize a new Point object
function Point(x, y) {
    this.x = x;  // this keyword is the new object being initialized
    this.y = y;  // Store function arguments as object properties
}
// By convention, constructors start with capitals
// No return is necessary

// Use a constructor function with the keyword "new" to create instances
var p = new Point(1, 1);  // The geometric point (1,1)

// Define methods for Point objects by assigning them to the prototype
// object associated with the constructor function.
Point.prototype.r = function() {
    return Math.sqrt(this.x * this.x + this.y * this.y);  // Return the square root of x^2 + y^2
    // This is the Point object on which the method...
    // ...is invoked.

};

// Now the Point object p (and all future Point objects) inherits the method r()
p.r()  // => 1.414...
Classes, by example

// Define a constructor function to initialize a new Point object
function Point(x,y) { // By convention, constructors start with capitals
    this.x = x; // this keyword is the new object being initialized
    this.y = y; // Store function arguments as object properties
} // No return is necessary

// Use a constructor function with the keyword "new" to create instances
var p = new Point(1, 1); // The geometric point (1,1)

// Define methods for Point objects by assigning them to the prototype
// object associated with the constructor function.
Point.prototype.r = function() {
    return Math.sqrt(Math.pow(this.x, 2) + Math.pow(this.y, 2)); // Return the square root of x^2 + y^2
    // This is the Point object on which the method...
    // ...is invoked.
};

// Now the Point object p (and all future Point objects) inherits the method r()
p.r() // => 1.414...

Example taken from JavaScript: The Definitive Guide (6th edition)
Classes – How to

• Define a constructor
  – Essentially a function
  – Initializes any instance variables
    • Referred to as this.x = ...
      – var x ... would be method variables.
  – Convention: constructor name starts with capital letter

• Define methods for the class by adding the to the Constructor prototype object

• Instantiate new objects of the class via new and the Constructor name
Objects and Prototypes

• All objects are based on a *prototype*
• Prototypes give objects inherent properties
• E.g. all objects have the method `toString()`
  – Where did it come from?
  – From its prototype (namely, `Object`)
• You can modify an object's prototype
  – e.g. `Point.prototype.r = function() ...`
• Properties of the prototype are shared with objects based on it
  – Changing the prototype changes the objects based on it
  – Changing objects based on it, does NOT change the prototype
Example

> p.r()
1.4142135623730951
> p.r = function(){return "Hello";}
function (){return "Hello";}
> p.r()
"Hello"
> t = new Point(2,2)
Point
> t.r()
2.8284271247461903
Augmenting standard prototypes

String.prototype.rot13 = function () {
  var a = "a".charCodeAt(0); // a is the char code for 'a'
  var z = "z".charCodeAt(0); // z is the char code for 'z'
  s = this.toLowerCase(); // keep it easy, only deal with lower case
  var r = ""; // the encrypted string to be returned
  for (var i = 0; i < s.length; i++) {
    var c = s.charCodeAt(i); // get the next character
    if (c < a || c > z) { // don't encrypt non-letters
      r += s.charAt(i);
      continue;
    }
    c -= a; // subtract 'a' to find the ordinal number in the alphabet
    var rc = ((c+13) % 26); // encrypt by adding 13, mod 26
    rc += a; // ordinal number back into a char code
    r += String.fromCharCode(rc); // add to the encrypted string
  }
  return r;
}

"Hello world".rot13();
Homework for Wednesday (before class)

• Create one class (not involving the DOM)
  – Constructor
  – at least 2 methods
  – demonstrate instantiating 2 objects of that class
  – demonstrate using its methods

• Add one creatively original method to String
  – demonstrate

• For each, turn in to Blackboard
  • code (single file for each)
  • description, with screenshots and narrative, demonstrating it working

• Read Ch 15 on the DOM in JS: Definitive Guide.

• Read Teaching Programming the Way it Works Outside the Classroom
  – This can only be downloaded for free from on campus or by using the CMU VPN.