Internet Computing
JavaScript (continued)

Arrays

```
var myList = new Array(24, "bread", true)
var myList2 = [24, "bread", true];
var myList3 = new Array(24);
```

- There are two ways to define an array in JavaScript
  - object literal notation
  - use the Array() constructor
- length property used to find the length of an array
  - Only the assigned elements of an array actually occupy space.1.
    - If it is convenient to used the subscript range of 100 to 150 (rather than 0 to 50), an array of length 151 can be created. But if only the elements indexed 100 to 150 are assigned values, the array will require the space of 51 elements, not 15
  - Length property is not necessarily the number of elements in the array.
    - New_list.length = 1002
    - But new_list may have no elements that have values or occupy space
  - Assigning a value to an array element that did not previously exist creates that element.

Arrays methods

- arrays in JavaScript are zero indexed
- [] notation for access
- .length
  - length of the array
- .push()
- .pop()
- concat()
- slice()
  - Does for arrays what the substring method does for strings.
- join(), reverse(), shift(), and sort()

```
Var names = new Array[“Mary”, “Murray”, “Max”]
Var name_string = names.join(“ : “);
Name_string “Mary : Murray : Max”

Var new_name = names.concat(“Moo”, “Meow”)
New_names: [“Mary”, “Murray”, “Max”, “Moo”, “Mewo”]

Var list = [2, 4, 6, 8, 10]
Var list2 = list.slice(1,3); // list2:[4,6]
Var list3 = list.slice(2);
//list3:[6,8,10]
```

Pattern Matching

- JavaScript provides two ways to do pattern matching:
  - Using RegExp objects
  - Using methods on String objects: search, match, split, replace
  - Regular expression used are the same
- Simple patterns
  - search(pattern)
  - Returns the position in the object string of the pattern (position is relative to zero); returns -1 if it fails

```
var str = "Gluckenheimer";
var position = str.search(/n/);
/* position is now 6 */
```

Pattern Matching: Normal and Metacharacters

- normal characters (match themselves)
- metacharacters
  - can have special meanings in patterns--do not match themselves
    - \ | ( ) [ ] { } ^ $ * + ? .
    - A metacharacter is treated as a normal character if it is backslashed
    - period is a special metacharacter - it matches any character except newline

/snow./ matches snowy, smpwe, snowd

Pattern Matching: Character Classes

- Specify classes of characters rather than individual characters
- Put a sequence of characters in brackets, and it defines a set of characters, any one of which matches:
  - [abcd] matches ‘a’, ‘b’, or ‘c’
  - Dashes can be used to specify spans of characters in a class: [a-z]
  - A caret at the left end of a class definition means the opposite [^0-9]
- Character class abbreviations

Abbr.	Equiv. Pattern	Matches
\d	[0-9]	a digit
\D	[^0-9]	not a digit
\w	[A-Za-z_0-9]	a word character
\W	[^A-Za-z_0-9]	not a word character
\s	[\r\t\n\f]	a whitespace character
\S	[^ \r\t\n\f]	not a whitespace character

/\d\.\d\d/ Matches a digit followed by a period, follwed by two digits

/\D\d\D/ Matches a single digit

/\w\w\w/ Matches three adjacent word characters

Pattern Matching: Quantifiers

- To repeat a pattern, a numeric quantifier, delimited by braces, is attached

Quantifier	Meaning
{n}	exactly n repetitions
{m,}	at least m repetitions
{m, n}	at least m but not more than n repetitions

xyyyyz
/xy{4}z/

- Symbolic quantifiers
  - * means zero or more repetitions
    - \d* means zero or more digits
  - + means one or more repetitions
    - \d+ means one or more digits
  - ? Means zero or one
    - \d? means zero or one digit

Pattern Matching: Anchors

- The pattern can be forced to match at a particular position in a string.
  - only at the left end with ^;
  - at the right end with $

```
/^Lee/
matches "Lee Ann" but not "Mary Lee Ann"

/Lee Ann$/
matches "Mary Lee Ann", but not "Mary Lee Ann
is nice"
```

- The anchor operators (^ and $) do not match characters in the string--they match positions, at the beginning or end

Pattern Modifiers

- Can be attached to patterns to change how they are used, thereby increasing their flexibility
  - The i modifier tells the matcher to ignore the case of letters
    - /oak/i matches "OAK" and "Oak" and …
- The x modifier tells the matcher to ignore whitespace in the pattern
  - allows comments in patterns

```
/\d+            #The street number
\s              #The space before the street name
[A-Z][a-z]+     #The street name
/x
```

```
/\d+\s[A-Z] [a-z] +/
```

Other pattern matching methods

- replace(pattern, string)
    - Finds a substring that matches the pattern and replaces it with the string
        - g modifier can be used (replaces all occurrences of pattern)
- match(pattern)
    - The most general pattern-matching method
    - Returns an array of results of the pattern-matching operation
        - If the g modifier, it returns an array of the substrings that matched
        - Without the g modifier, first element of the returned array has the matched substring

```
var str = "My 3 kings beat your 2 aces";
var matches = str.match(/[ab]/g);

matches is set to ["b", "a", "a"]
```

```
var str = "Some rabbits are rabid";
str.replace(/rab/g, "tim");

str is now "Some timbits are timid"
```

demo:forms_check.js

Debugging JavaScript

- FX3+
  - Select Tools, Web Developer, Error Console
  - A small window appears to display script errors
  - Remember to Clear the console after using an error message to avoid confusion
- Chrome
  - Select Tools -> Developers Tools -> JavaScript console
  - Produces an error console window

Functions

- function declaration

```
Function subtotal(price, quantity) {
   return price * quantity;
}
Var result = subtotal(10, 2)
```

- function expression
  - An object whose content is the definition of the function
  - Functions are objects, so variables that reference them can be treated as functions

```
Var sub = function subtotal(price, quantity) {
   return price * quantity;
}
Var result = sub(10, 2)
```

functions

- Anonymous function
  - JavaScript allows creating anonymous function

```
Var calc_sub = function (price, quantity) {
   return price * quantity;
}
Var result = calc_sub(10, 2)
```

- Module pattern
  - Wraps all of your file's code in an anonymous function that is declared and immediately called

```
(function() {
  statements;
})();
```

functions

- return value
  - is the parameter of return
  - If there is no return, or if the end of the function is reached, undefined is returned
  - If return has no parameter, undefined is returned

functions

- function parameters
  - Actual parameters: the parameter values that appear in a call to a function
  - Formal parameters: the parameter names that appear in the header of a function definiiton
- parameters are passed by value, but when a reference variable is passed, the semantics are pass-by-reference
  - DEMP function_params.html and output
- There is no type checking of parameters, nor is the number of parameters checked
  - excess actual parameters are ignored, excess formal parameters are set to undefined)
  - DEMO params.js and output
- All parameters are sent through a property array, arguments, which has the length property

functions

- There is no clean way to send a primitive by reference
  - One way is to put the value in an array and send the array's name
- Sort function
  - To sort something other than strings into alphabetical order, write a function that performs the comparison and send it to the sort method
  - The comparison function must return a negative number, zero, or a positive number to indicate whether the order is ok, equal, or not ok

```
function by10(a)
{
    a[0] *= 10;
}
var x=5;
var listx = new Array(1);
listx[0] = x;
by10(listx);
x = listx[0];
console.log(x);
```

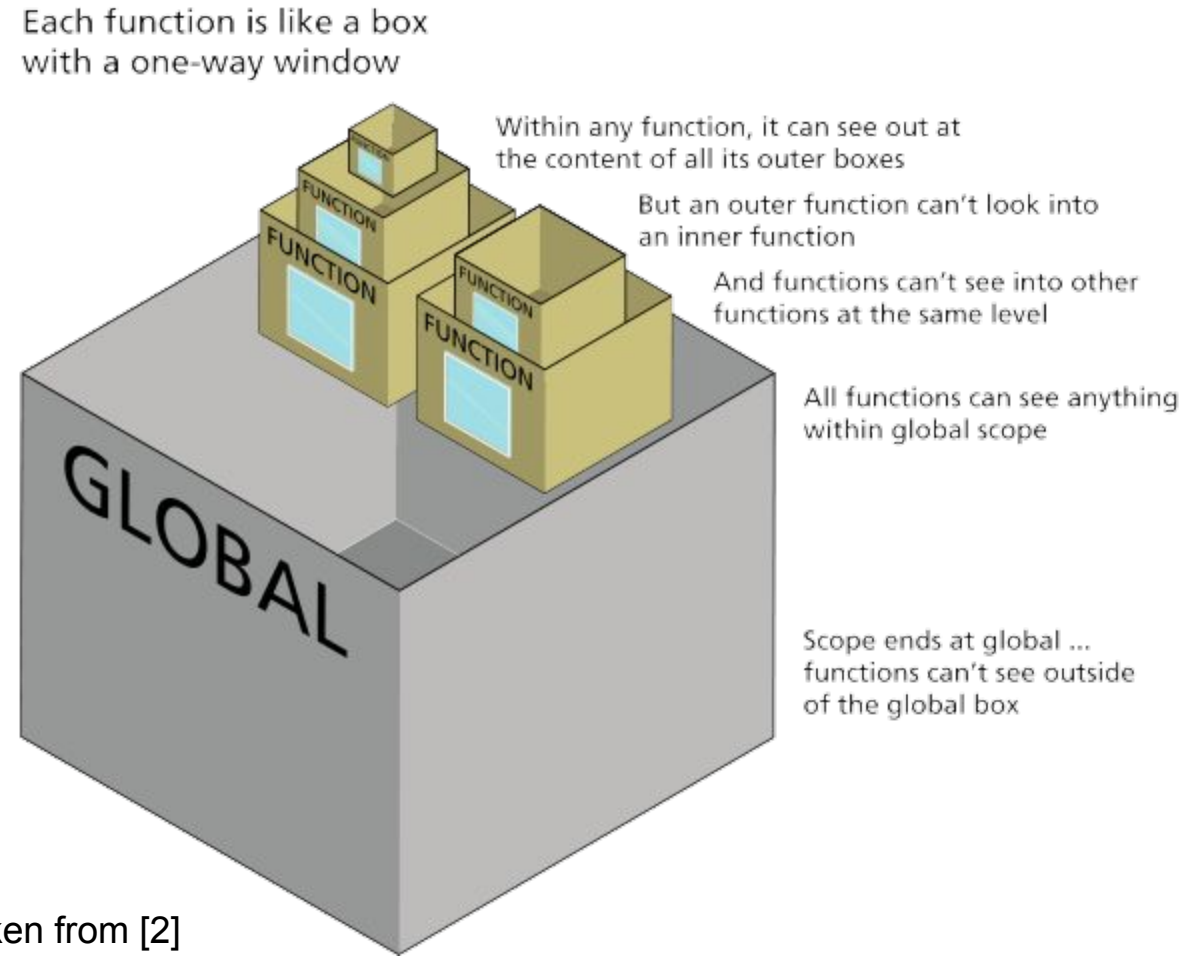
```
function num_order(a, b)
{
return b-a;
}

//sort the array of num_list, into
ascending order

num_list.sort(num_order)
```

Functions

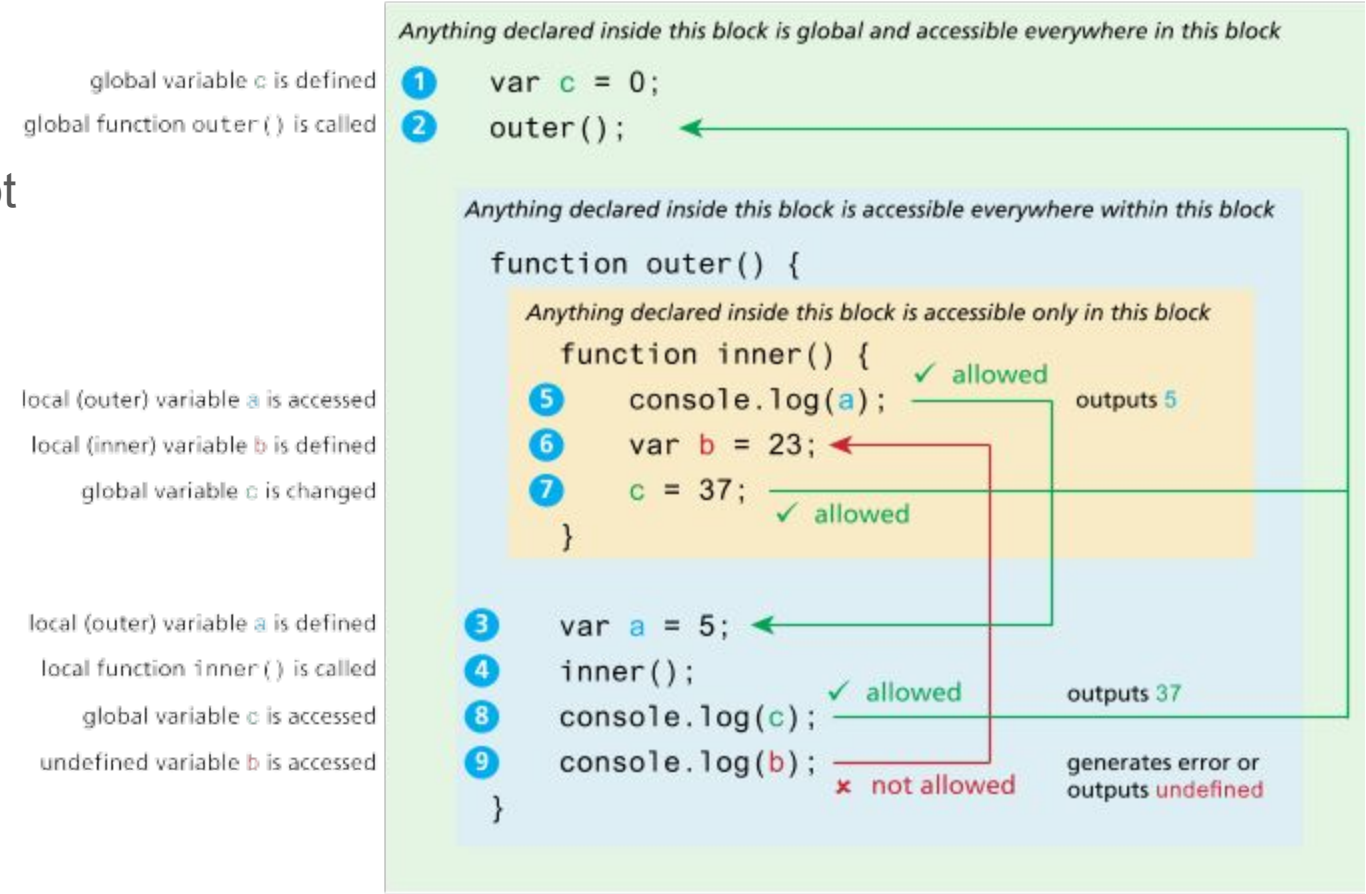
Scope in JavaScript



Taken from [2]

functions

Scope in JavaScript



Taken from [2]

Scoping

Remember that scope is determined at design-time

```
var myGlobal = 55;

function outer() {

    var foo = 66;

    function middle() {

        var bar = 77;

        function inner() {

            var foo = 88;

            bar = foo + myGlobal;
        } 1 looks first within current function

    } 2 then looks within first containing function

} 3 then looks within next containing function

4 then finally looks within global scope
```

Taken from [2]

Function: scoping

A variables that is not declared with var statement is implicilty declared

Variables that are implicitly declared hava global scope.

Object Creation

- Object Literal Notation
- Using constructor (new keyword)

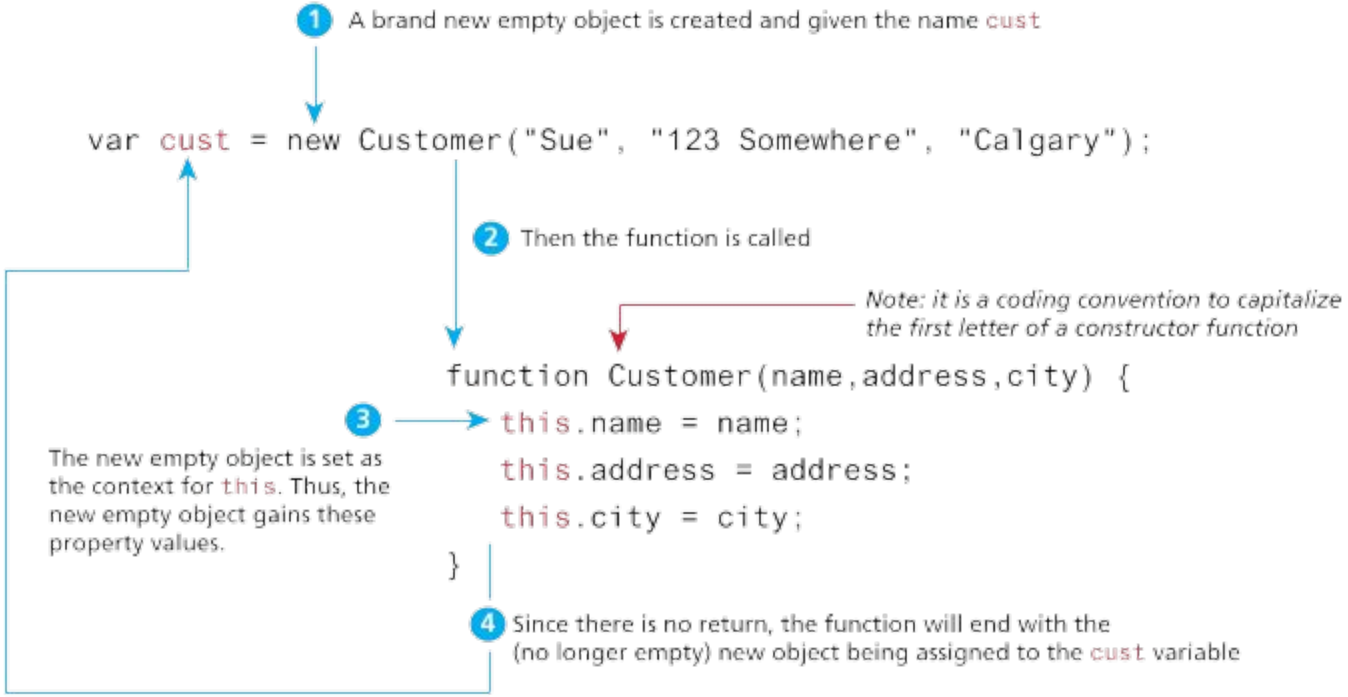
Var my_car = {make: "Ford", model: "Fusion”}

```
var my_car = new Object(); //create an empty object
my_car.make = "Ford";   //create and initialize properties
my_car.model = "Fusion";
var property1 = my_car["model"];
delete my_car.model;
for (var prop in my_car) {
}
```

- Properties can be accessed by **dot notation** or in **array notation,** as in

functions

Function constructors



Taken from [2]

Functions

Objects and function together

```
var order = {
    salesDate : "May 5, 2017",
    product : {
        type: "laptop",
        price: 500.00,
        output: function () {
            return this.type + ' $' + this.price;
        }
    },
    customer : {
        name: "Sue Smith",
        address: "123 Somewhere St",
        output: function () {
            return this.name + ', ' + this.address;
        }
    },
    output: function () {
            return 'Date' + this.salesDate;
    }
};
```

Taken from [2]

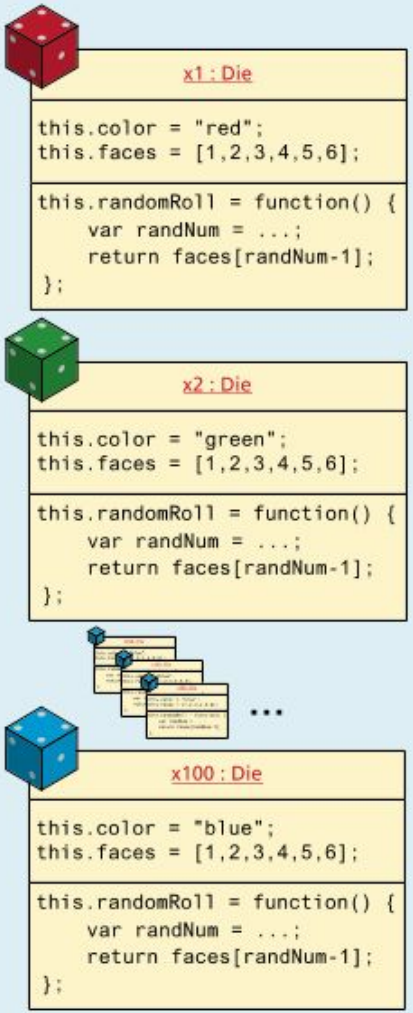
Object Prototypes

- "All JavaScript objects inherit properties and methods from a prototype:
- The JavaScript `prototype` property allows you to add new properties and method to object constructors:
  - `Date` objects inherit from `Date.prototype`
  - `Array` objects inherit from `Array.prototype`
  - `Person` objects inherit from `Person.prototype`
- The `Object.prototype` is on the top of the prototype inheritance chain:
- `Date` objects, `Array` objects, and `Person` objects inherit from `Object.prototype`."[3]

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

Person.prototype.nationality = "English";


Person.prototype.name = function() {
  return this.firstName + " " +
this.lastName;
};
```

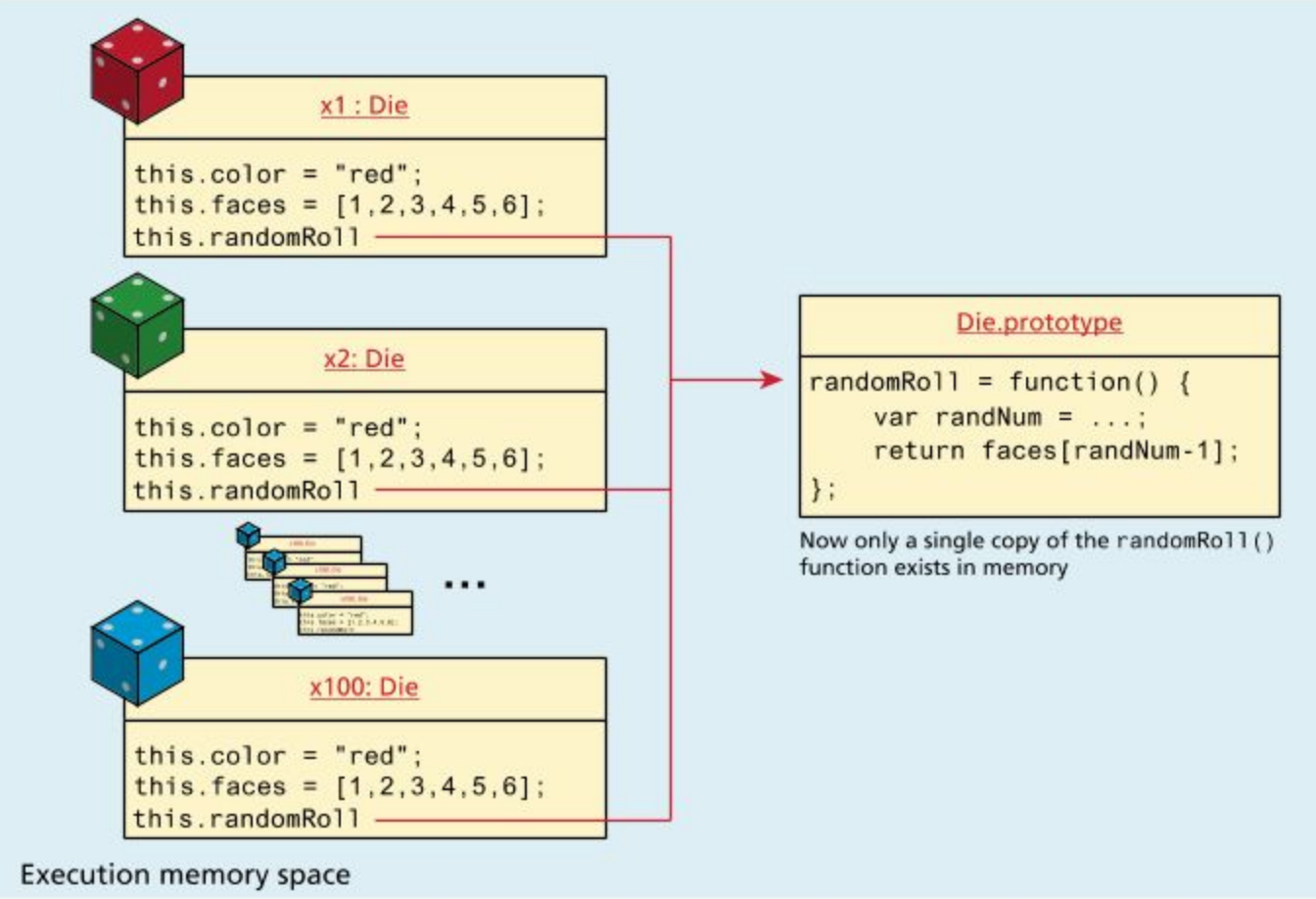


A function expression is an object whose content is the definition of the function ...

so each instance will contain that same content ...

which is incredibly memory inefficient when there are many instances of that object

Execution memory space


x1 : Die
this.color = "red";
this.faces = [1,2,3,4,5,6];
this.randomRoll
x2: Die
this.color = "red";
this.faces = [1,2,3,4,5,6];
this.randomRoll
...
x100: Die
this.color = "red";
this.faces = [1,2,3,4,5,6];
this.randomRoll
Die.prototype
randomRoll = function() {
var randNum = ...;
return faces[randNum-1];
};
Now only a single copy of the randomRoll()
function exists in memory
Execution memory space

References

1. Programming the World Wide Web, 8th edition
2. Fundamentals of Web Development, 2nd edition
3. https://www.w3schools.com/