# The JavaScript Language

Chapter 4

# Outline

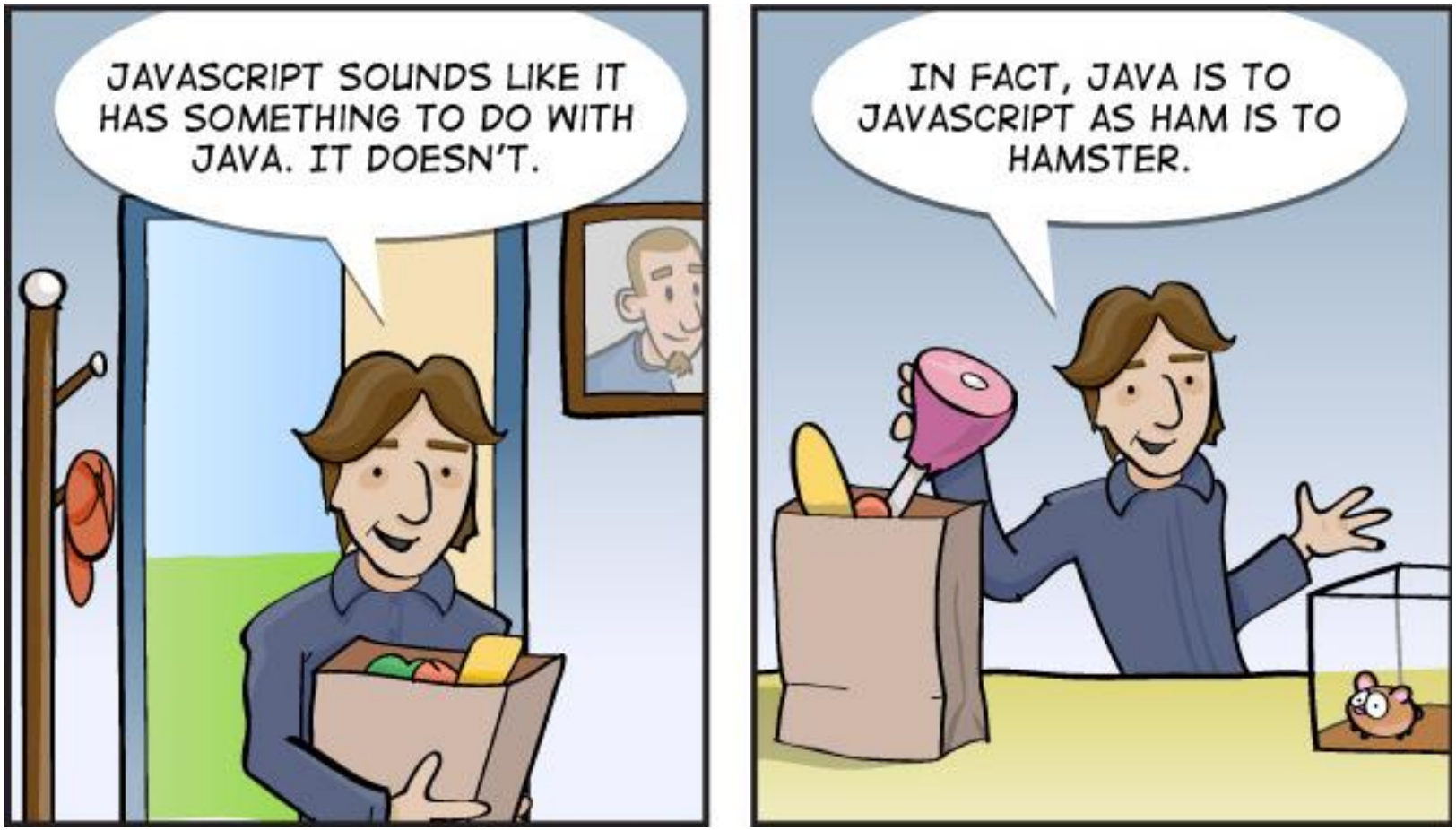


- A. Data structure and control statements
- B. Prototype and inheritance
- C. Function and closure
- D. Regular expression

# JavaScript



- The most widely used programming language on the Web
  - Mainly run in web browsers, add dynamic behavior to web pages
  - No compilation required. Source code of JavaScript included inside HTML pages or separate *.js file
- Looks like Java, but very different internally
  - Standard ECMA-262, also known as ECMAScript


JAVASCRIPT SOUNDS LIKE IT HAS SOMETHING TO DO WITH JAVA. IT DOESN'T.
IN FACT, JAVA IS TO JAVASCRIPT AS HAM IS TO HAMSTER.

# When was the code executed?



- It depends on:
  - The location you put the <script> tags
    - In <head>: Downloaded & executed on page load (**blocks the parser!**)
    - At the end of <body>: Downloaded & executed after the parsing of main DOM elements. Less impact on page load.
  - The *async* and *defer* attribute: modern approach
    - Do NOT block the parsing
    - async: the code will be executed asynchrously with the page, **as soon as they have been downloaded**
    - defer: the code will be executed after the parsing finished, **executed in order**

# Part A. Data structure and control statements



- Topics:
  - Variables and data types
  - Arrays and Objects
  - Control statements
  - Functions (basics)
- Lab:
  - Editing and running JavaScript program

# Variables



- Declare a variable: var a;
- JS uses loose typing. A variable can hold data of different types
- Basic data types
  - Number: var n = 1; n+=2;
  - String: var fname="Peter"; var fullname=fname + " " + 'Chan';
  - Boolean: true or false
  - Array: var P = [2,3,5,7,11]; // P[4] is 11
  - Object: var Dict = { one: 1, two: 2 }; Dict['three']=3; Dict.four = 4; // Dict.one is 1
  - Function
- Special value: null, undefined

# Number

For a list of all methods, see online reference at https://developer.mozilla.org/en/JavaScript/Reference



- JS numbers are 64-bit floating point, similar to Java's double.
  - E.g. 0, 1, 2, 3.1416, 1e2 (i.e. 100), 5e-2 (i.e. 0.05), NaN, Math.PI
  - Numbers are immutable, and not object. However, numbers have some methods.

```
// number.toString(radix) converts a number to string
Math.PI.toString() // "3.141592653589793"
Math.PI.toString(16) // "3.243f6a8885a3"
(202).toString(2) // "11001010"
```

# Math



- The Math object provides some math functions and constants.
  - Math.PI, Math.E
  - Math.sin(), Math.cos(), Math.sqrt(), Math.exp()
  - Math.floor(), Math.ceil(), Math.round()
  - Math.random()

```
var a = Math.random() // a number between 0 and 1

// a number from 1,2,3,4,5 and 6
var b = Math.floor(Math.random()*6)+1

var x = Math.sin(Math.PI/2)  // 1
var y = Math.sqrt(-1) // NaN
```

# String



- A string contains zero or more character
  - Each char is 16bit unicode
  - Single quote, or double quote
  - backslash \ is the escape char, e.g. \\, \", \'
  - The property string.length
  - Concatenation with +

```
var user = "peterchan";
var domain = 'ymail.com';
var addr = user + "@" + domain;
var n = "seven".length;  // 5
```

# String methods



- Strings are immutable, and not object. However, they have some methods.
  - .charAt(pos) – character at a position
  - .indexOf(searchString, pos) – search the searchString from the position of the string
  - .slice(start, end) – copy a portion of the string
  - .split(separater) – split the string into pieces

```
var name = "peter"; var initial = name.charAt(0);
var text = 'Mississippi'; var p = text.indexOf('ss'); // 2
var mesg = "All the best";
p = mesg.slice(0,3); // 'All'
p = mesg.slice(4); // 'the best'
p = mesg.slice(-4); // 'best'
p = mesg.slice(-4,-2); // 'be'
var A = "202.175.3.3".split('.');
```

# Array



- Arrays may change size in run-time
- The length property usually refers to the number of elements

```
var empty = [ ];
var numbers = [ 'zero', 'one', 'two', 'three', 'four' ] ;
empty[1] // undefined
numbers[1] // 'one'
empty.length // 0
numbers.length // 5
```

```
var P = [ 2,3,5,7,11 ] ;  // small primes
var i;
for (i=0; i<P.length; i+=1) {
  console.log(P[i]);
}
```

# Array methods, 1



```
var x = [ 'a', 'b', 'c' ];
var y = [ 1, [ ], 2 ];
var z = x.concat(y); // z is ['a', 'b', 'c', 1, [ ], 2 ]

x.push('d'); // x becomes [ 'a', 'b', 'c', 'd' ]
z = x.pop(); // z is 'd'. x becomes [ 'a', 'b', 'c' ]

x = [ 'a', 'b', 'c' ];
z = x.shift(); // z is 'a'. x becomes [ 'b', 'c' ]
x.unshift('A'); // x becomes [ 'A', 'b', 'c' ]

x = [ 'b', 'a', 't' ];
y = x.reverse(); // y is [ 't', 'a', 'b' ];
x.sort(); // x becomes [ 'a', 'b', 't' ];

x = [ 'a', 'b', 'c']; var s = x.join('-'); // s is 'a-b-c'
```

# Array methods, 2



- array.slice(start, end) makes a shallow copy of a portion of an array
- array.splice(start, deleteCount, item…) removes elements from an array, replacing them with new items

```
var x, y;
x = [ 'a', 'b', 'c', 'd' ];
y = x.slice(0, 2); // y is [ 'a', 'b' ]
y = x.slice(-3, -1); // y is [ 'b', 'c' ]

x = [ 'a', 'b', 'c', 'd', 'e'];
y = x.splice(1, 2);
console.log("y is "+ y);
console.log("x is "+ x);
// try to add new items in splice()
```

# Objects



- JavaScript objects are mutable collections of properties
  - Each property is a pair of name and value
  - A method is a property with function value
- An object works like a dictionary (or hash table)
  - Use a property name to look up a value
  - Add a property (i.e. a name-value pair)
  - Remove a property
- A property with function value works like method in Java
  - obj.method(param1, param2)
- A special for .. in .. loop for object

# Two ways to create object



- Constructor function
  - Apply 'new' (will discuss constructor function in Part B)
- Object literal
  - a list of properties inside { }, each property is a name,value pair
  - The order is not important

```
var now = new Date();
var dow = now.getDay();
if (dow==0) {
   /* today is Sunday */
}
```

```
var empty_object = { } ;

var person = {
  firstname: "Peter",
  lastname: "Chan"
};
```

# Object literals



- An object literal is a list of properties
  - If the property name is a valid JS name and is not a reserved word, don't need to quote it
  - Quote the property name otherwise
  - The property value can be any value, incl. array, function, or another object

```
var seasons = {
  spring: '春', summer: '夏',  autumn: '秋', winter: '冬'
};

var contact = {
  login_name: "peter",
  "full name": "Peter Chan",
  age: 20,
  married: false
};
```

# Object as dictionary, 1



- Look up a property with the property name (key) with obj['key'] or obj.key
  - Return undefined if there is no such member
  - Must use [] if the property name is not valid JS names or reserved word
    - See : https://mathiasbynens.be/notes/javascript-identifiers

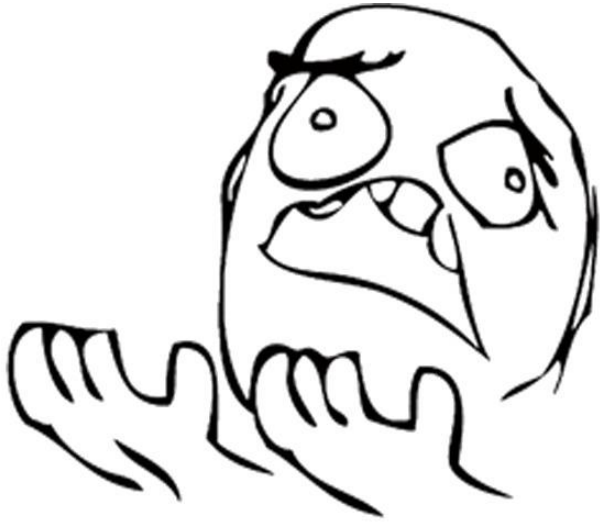
```
var seasons = {
  spring: '春', summer: '夏',  autumn: '秋', winter: '冬'
};

seasons['summer'] // '夏'
seasons.summer //  same as above

seasons['apple'] // undefined
```

```
var cn2en = {
  '春': 'spring', '夏': 'summer',
  '秋': 'autumn', '冬' : 'winter'
};

cn2en['夏'] // 'summer'
cn2en.夏 //  ?
```

# So, valid JS names?



```
// How convenient!
var π = Math.PI;

// Sometimes, you just have to use the Bad Parts of JavaScript:
var ʊ_ʊ = eval;

// Code, Y U NO WORK?!
var ლ_ʊ益ʊ_ლ = 42;

// Cthulhu was here
var HE COMETH = 'Zalgo';
```



Holy JavaScript!

# Object as dictionary, 2



- Add a new member or update an existing member by assignment =
- Delete a member by delete
- for .. in enumerates the **property names** of an object
  - in an **arbitrary** order

```
var N = { zero: 0, three: 3 };

N.two = 2;
N['one'] = 1;

delete N.zero;

console.log(N);

for (var w in N) {
  console.log(w + ' is ' + N[w]);
}
```

# Example: Object literals



```
var flight = {
  airline: "Oceanic",
  number: 815,
  departure: {
    IATA: "SYD",
    time: "2004-09-22 14:55",
    city: "Sydney"
  },
  arrival: {
    IATA: "LAX",
    time: "2004-09-23 10:42",
    city: "Los Angeles"
  }
};
```

```
// add a property which has
another object as value
flight.equipment = {
  model: 'Boeing 777'
};

console.log(
   'This flight goes from ',
   flight.departure.city,
   ' to ',
   flight.arrival.city
);
```

# Objects: reference



- Objects are passed around by reference. They are never copied.
  - Arrays and functions are also objects.
  - Numbers, strings, booleans are not objects. They are passed by value.

```
var person = {  firstname: 'Peter', lastname: 'Chan' };
var x = person;
x.nickname = 'Billy';
var nick = person.nickname; // 'Billy'

var a = { }, b = { }, c = { } ;
// a, b, and c each refer to a different empty object

a = b = c = { } ;
// a, b, and c all refer to the same empty object
// BE CAUTION WITH THIS!
```

# Selection: if



- if statement branches according to an expression

```
// get current time
var d = new Date();
var time = d.getHours();
var greeting;
if (time<10)  {
    greeting = "Good morning";
} else if (time>10 && time<16)  {
    greeting = "Good day";
} else  {
    greeting = "Hello";
}
```

# Selection: switch



- switch statement compares a number of string values with several specified cases.

```
var d = new Date();
var day = d.getDay();
switch (day) {
  case 5:
    comment = "Finally Friday";
    break;
  case 6:
    comment = "Super Saturday";
    break;
  case 0:
    comment = "Sleepy Sunday";
    break;
  default:
    comment = "I'm looking forward to this weekend!";
}
```

# Repetition



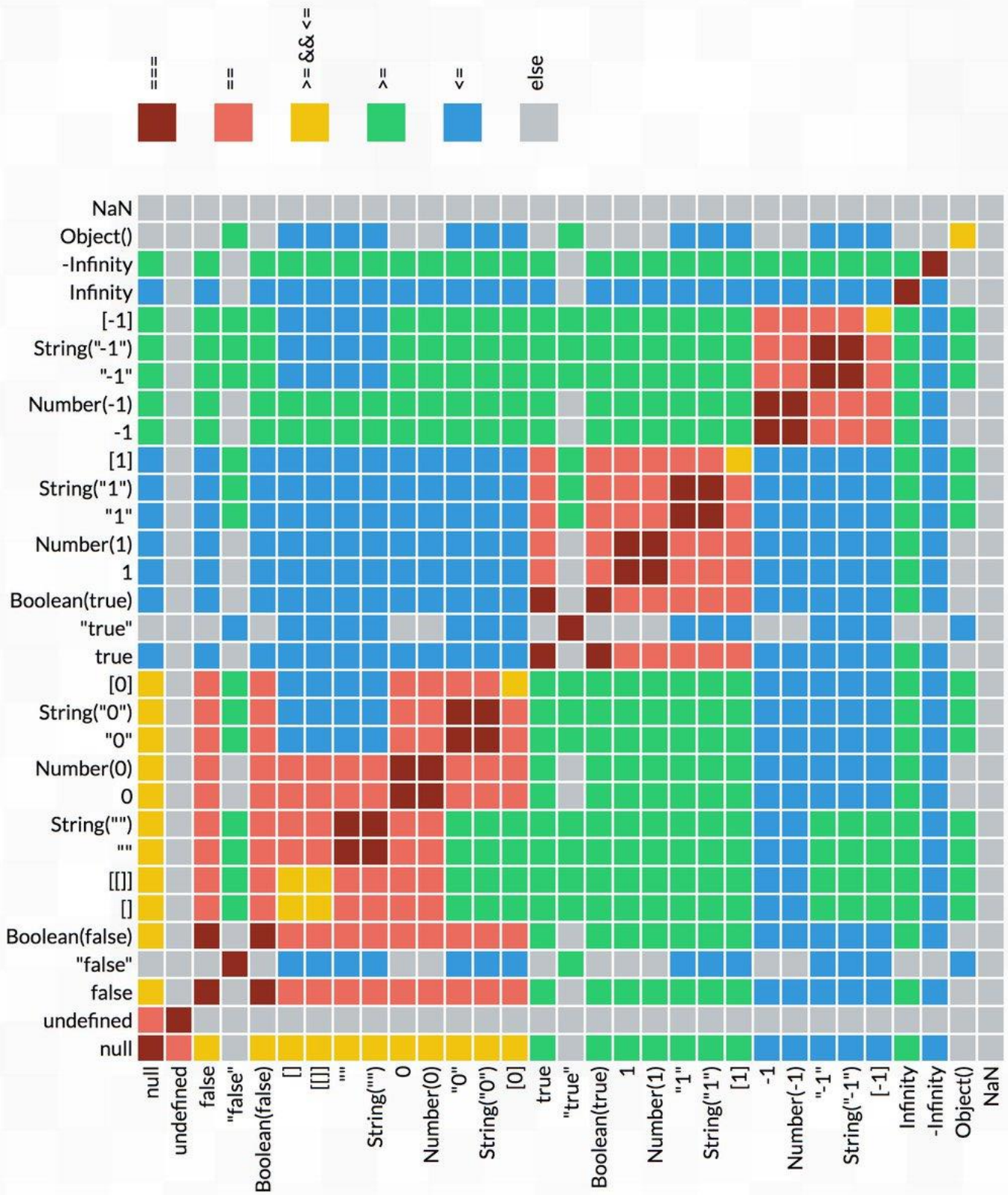
- for, while, and do while loops are similar to the loops in Java

```
var sum=0; var i;
for (i=1; i<10; i++) sum+=i;
// sum=1+2+3+...+9

// sequential search
var A = [ 'mon', 'tue', 'wed', 'thu', 'fri' ];
for (i=0; i<A.length; i++) {
  if (A[i]==='wed') break;
}

// sum of squares less than 100
i = 0; sum = 0;
while (i*i<100) {
   sum += i*i;
   i++;
}
```

# Operators



- Operators are similar to Java
  - + - * / %
  - >= <= > <
  - === !==
    - == and != may convert data type before comparision
    - e.g. 2=='2' is true whereas 2==='2' is false
  - && || ?:
    - && produces the value of its first operand if the first operand is false. Otherwise, it produces the value of the second operand.
    - || produces the value of its first operand if the first operand is truth. Otherwise, it produces the value of the second operand.
    - cond? expr1:expr2 is the only **ternary operator** in JS. If the condition is true, expr1 will be executed. Else expr2.


===
==
>= && <=
>=
<=
else
NaN
Object()
-Infinity
Infinity
[-1]
String("-1")
"-1"
Number(-1)
-1
[1]
String("1")
"1"
Number(1)
1
Boolean(true)
"true"
true
[0]
String("0")
"0"
Number(0)
0
String("")
""
[[]]
[]
Boolean(false)
"false"
false
undefined
null
null
undefined
false
"false"
Boolean(false)
[]
[[]]
""
String("")
0
Number(0)
"0"
String("0")
[0]
true
"true"
Boolean(true)
1
Number(1)
"1"
String("1")
[1]
-1
Number(-1)
"-1"
String("-1")
[-1]
Infinity
-Infinity
Object()
NaN

# Function



- A function contains some code that will be executed when the function is called
  - You can pass in some values as parameters
  - A function can return a single value
    - If none specified, return undefined
  - To invoke a function f, use f()

```
function bigger (a,b) {
   if (a>b)
     return a
   else
     return b;
}

var b = bigger(2,3);
```

# Local and global variables



- Local variables are defined inside a function, and are visible only within this function
  - Variables have function scope, and their declarations are hoisted
- Global variables are defined outside any functions, and are visible to all functions

```
var username;

function login(un, pw) {
  /* after verifying the password */
  username = un;
}
function greet() {
   var today = new Date(); var time = today.getHours();
    if (time<10) {
      alert("Good morning " + username);
    } else { … }
}
```

**username** is defined outside any function. It is a global variable.

Definition inside a function makes *today* and *time* local variables.

**username** is not defined locally. So it is treated as global variable.

# Function as a first-class value



- A function is an object
  - assign to a variable, and invoke later
  - save as the value of a property of another object, and later invoke as a method
  - returned by another function

```
// save a function value in a variable and invoke it later
var circle_area = function (r) {
  return Math.PI * r * r;
} ;

// umm.. I want a shorter name
var ca = circle_area;

// call the function
var area = ca(5);
```

# Exceptions



- you can throw an object in case of error
  - Such objects usually have a name and a message

```
var add = function (a,b) {
   if (typeof a !== 'number' || typeof b !== 'number') {
    throw { name: 'TypeError', message: 'add needs numbers' };
   }
   return a + b;
};

try {
  add("seven");
} catch (e) {
   console.log(e.name + ": " + e.message);
}
```

# Part B. Prototypes and inheritance



- Functions as methods
- Defining objects with state and behavior
- Prototype
  - Property lookup along the prototype chain
  - Property inheritance
- Constructor functions
  - Using constructor to define a class
  - Built-in constructor functions
- Defining class
- Defining subclass

# Function as method



- When a property has a function as its value, it can be invoked as a method of the object.
  - Syntax: obj.func()
  - Inside the method, **this** refers to the object on which the method is called.

```
var peter = { name: 'Peter',  age: 12 };
peter.greet = function ( ) {
    console.log("Hello! I am " + this.name);
};

peter.greet(); // print 'Hello! I am Peter'
```

# Defining an object with methods



- JavaScript objects can represent objects in the OOP sense
  - Field (state) – property with non-function value
  - Method (behavior) – property with function value

```
var peter = {
  name: 'Peter',
  age: 12,
  greet: function ( ) {
    console.log("Hello! I am " + this.name);
  }
};

peter.greet();
console.log( peter.name + "'s age is " + peter.age );
```

| | |
|---|---|
| **name** | Peter |
| **age** | 12 |
| **greet** | function () {…} |

# Problem of repeating method definition



- A simplistic approach to define several similar objects is to repeat definition of methods for each object
- Problem: possible memory waste

```
var peter = {
  name: 'Peter',
  age: 12,
  greet: function ( ) {
    console.log("Hello! I am " + this.name);
  }
};
```

| name | Peter |
|---|---|
| age | 12 |
| greet | function () {…} |

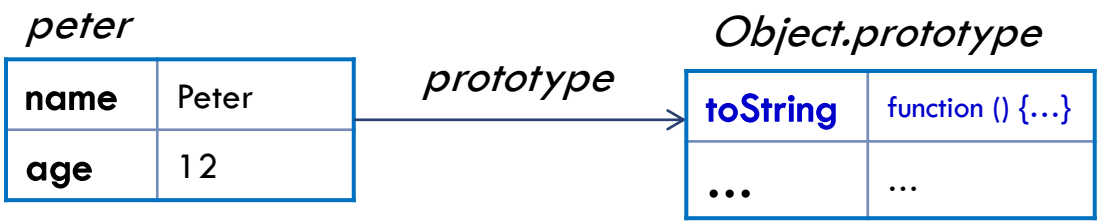
```
var mary = {
  name: 'Mary',
  age: 13,
  greet: function ( ) {
    console.log("Hello! I am " + this.name);
  }
};
```

| name | Mary |
|---|---|
| age | 13 |
| greet | function () {…} |

# Prototype



- Every JavaScript object has an implicit pointer to its prototype object
- When JavaScript cannot find a property in a certain object, it will follow this pointer and continue searching for the property at the prototype object. This provides a form of inheritance.

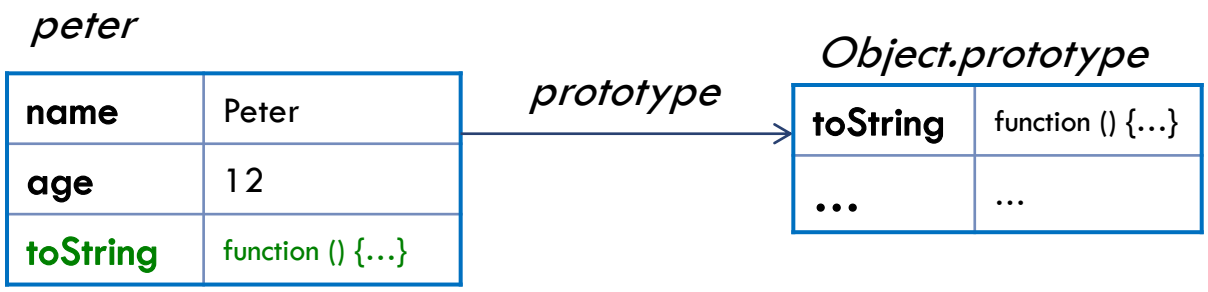



```
var peter = { name: 'Peter', age: 12 };

console.log("The object is  " + peter.toString() );
// the same as console.log("The object is  " + peter );
```

# Redefining property



- An object can redefine a property inherited from its prototype object
- The prototype chain is only searched in property lookup. Adding or updating properties act on the object itself.

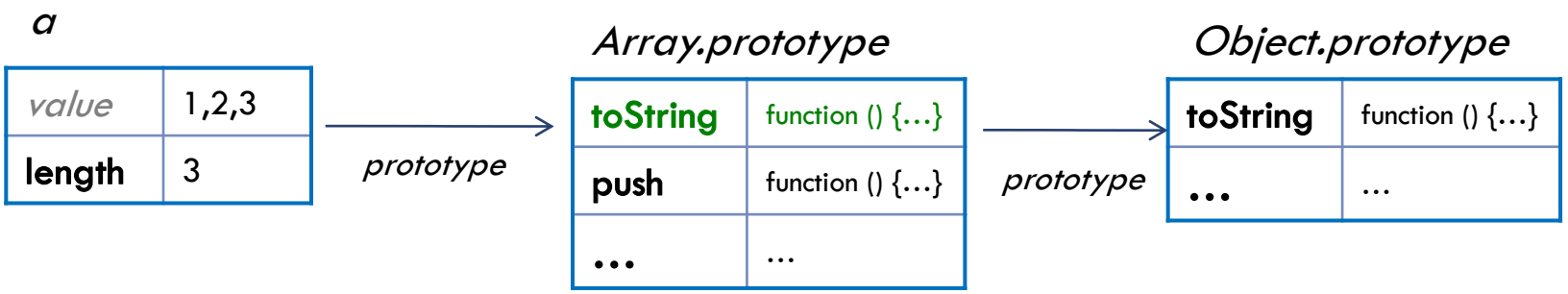



```
var peter = { name: 'Peter', age: 12 };
peter.toString = function () {
   return 'A person called ' + this.name;
}

console.log("The object is  " + peter.toString() );
```

# Prototype chain



- The prototype object of an object also has an implicit prototype pointer
- These pointers link several objects into a prototype chain
- A prototype chain ends in the predefined Object.prototype

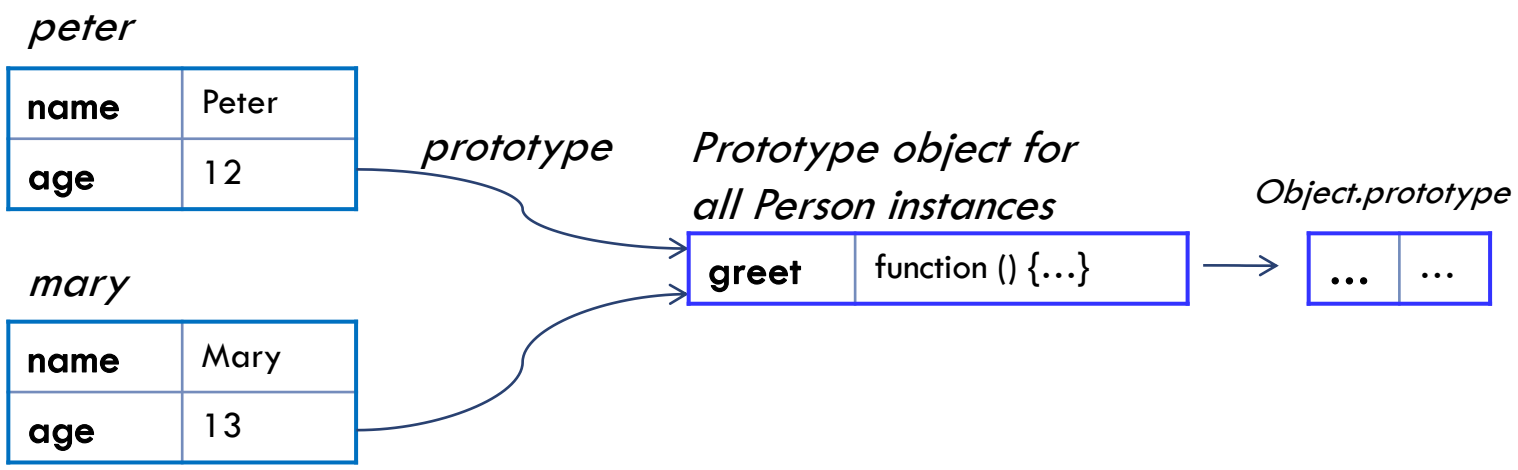



```
var a = [ 1, 2, 3 ];

console.log("The object is  " + a.toString() );
```

# Making a class of objects



- Prototype provides a way to create a class of similar objects in JavaScript
  - Define methods in the prototype object of the class
  - Create instances of the class, and set their prototype pointer to the prototype object
  - But.. How?



# Constructor function



- The standard way to set prototype object is constructor function
  - a constructor function F creates an object when invoked by the new operator
  - This newly created object uses F.prototype as its prototype object
  - Inside the constructor function, this refers to the newly created object
  - By convention, name of constructor functions is capitalized
  - Constructor functions take the role of 'class' in Java
  - 'instance' refers to an object created by a constructor function

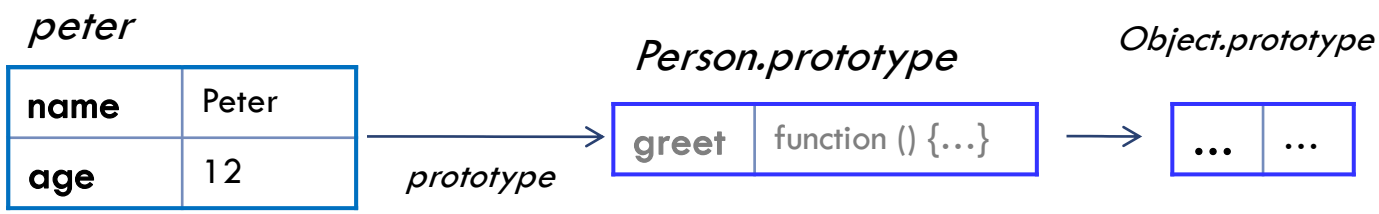
```
function Person (n, a) {
  this.name = n;
  this.age = a;
}
// todo: define method
var peter =
   new Person('Peter', 12);
```

# Calling a constructor



- Steps to create an instance
  - Create an empty object with its prototype pointer set to Person.prototype
  - Initialize properties of the instance
  - Return the instance

```
function Person (n, a) {
  this.name = n;
  this.age = a;
}
// todo: define method
var peter =
   new Person('Peter', 12);
```

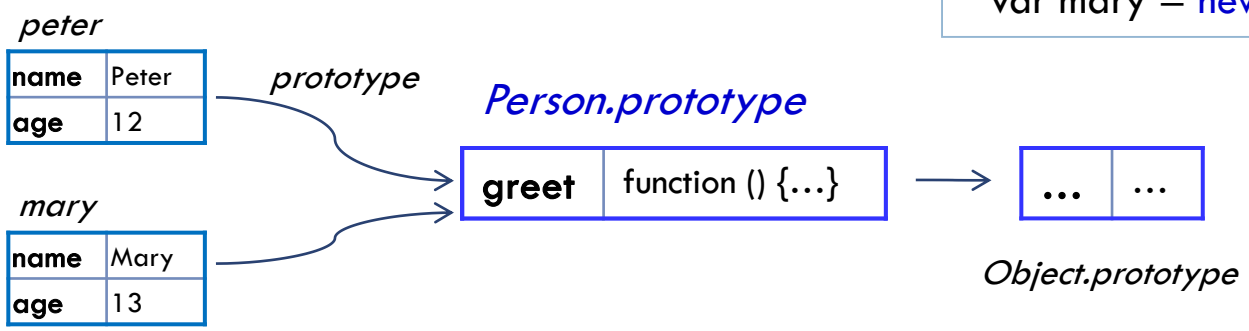


# Adding methods to the prototype



- JavaScript creates an object as the prototype property of a constructor function
- This prototype property is used as the prototype object of instances created by the constructor function
- Methods defined at this prototype object are shared by all the instances

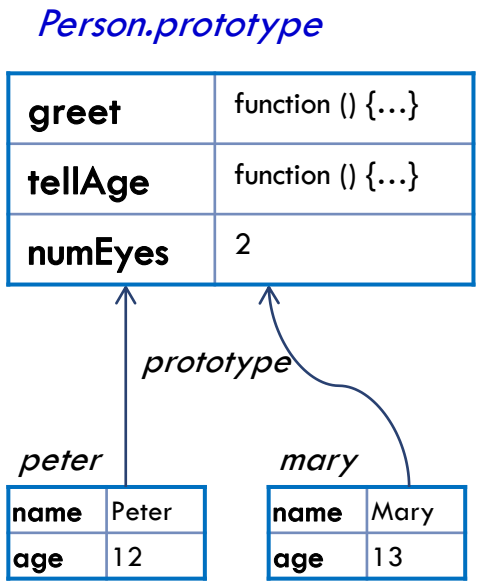
```
function Person (n, a) {
  this.name = n;
  this.age = a;
}

Person.prototype.greet =
   function ( ) {
      console.log("Hello! I am " +
         this.name);
   };

var peter = new Person('Peter', 12);
var mary = new Person('Mary', 13);
```



# Example: defining a class



```
function Person (n, a) {
  this.name = n;
  this.age = a;
}

Person.prototype.greet = function ( ) {
    console.log("Hello! I am " + this.name);
};
Person.prototype.tellAge = function ( ) {
   console.log("My age is " + this.age);
};
Person.prototype.numEyes = 2;

var peter = new Person('Peter', 12);
var mary = new Person('Mary', 13);

peter.greet();  mary.tellAge();
```



# Constructor functions for built-in types



- Values of built-in types can be created by literal or constructor functions
  - Except Date(), it is recommended to use literal to create these values.
  - Use the constructors to look up the reference
  - Common methods defined at the prototype property of the constructor functions

| Constructor | Create by literal | Create by constructor |
|---|---|---|
| Boolean | var b = true; | var b = new Boolean(true) |
| Number | var n = 1.2; | var n = new Number(1.2); |
| String | var s = "peter"; | var s = new String("peter"); |
| Array | var A = [1,2,3]; | var A = new Array(1,2,3); |
| Object | var O = { }; | var O = new Object( ); |
| Function | function add(a,b) { return (a+b); } | var adder = new Function("a", "b", "return a + b"); |
| RegExp | var re = /lo+ng/i; | var re = new RegExp("lo+ng", "i"); |

# Advanced example



- You can add a method to all array instances

```
var a = [ 1, 2, 3 ];
var b = [ 10, 12, 13, 15 ];

Array.prototype.sum = function() {
  var s = 0;
  for (var i=0; i<this.length; i++) {
     s+= this[i];
  }
  return s;
};

a.sum(); // return 6;
b.sum(); // return 50;
```

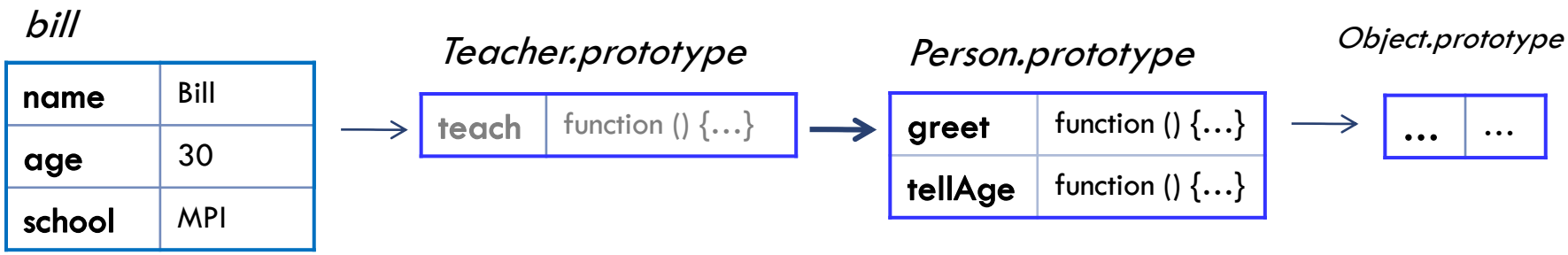
# Defining a subclass



- To define a subclass, set the prototype of the constructor function of the subclass to an *instance* of the superclass
- Properties of superclass are inherited along the prototype chain

```
function Person (n, a) { … }

function Teacher (n, a, sch) { … }
Teacher.prototype = new Person();

var bill = new Teacher ('Bill', 30, 'MPI');
```

Recall that:
var p = new Person();
p.__proto__=Person.prototype;
So,
Teacher.protoype.__proto__
= Person.prototype

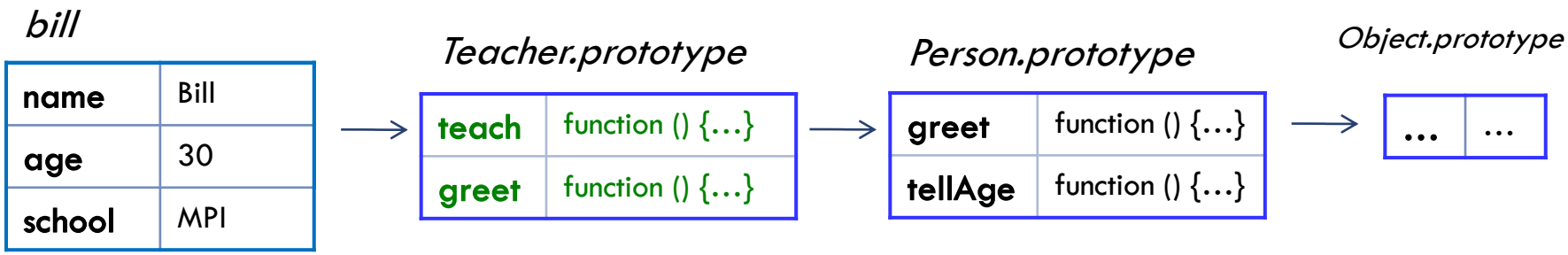


# Defining methods of subclass



- You can define new methods of the subclass, and override methods from the superclass

```
Teacher.prototype.teach = function ( ) { … }

Teacher.prototype.greet = function ( ) {
    console.log("Hello! I am " + this.name +
        ". I teach in " + this.school);
};

var bill = new Teacher ('Bill', 30, 'MPI');
bill.greet(); bill.teach();
```

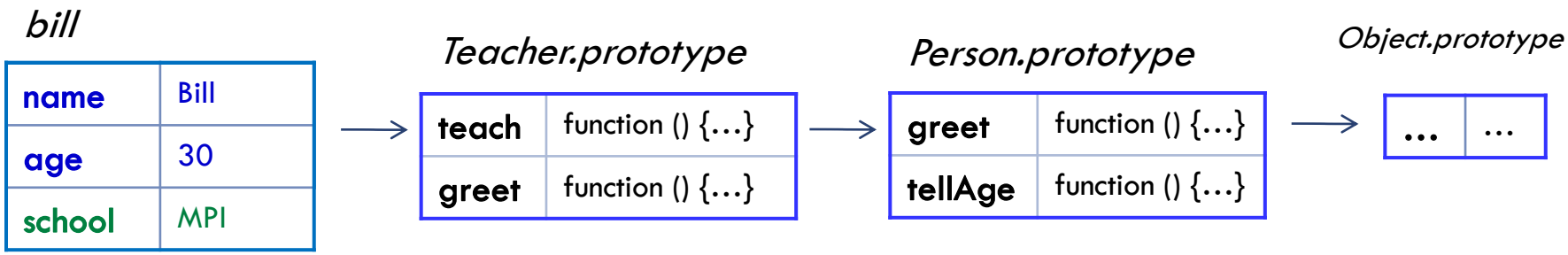


# Calling constructor of superclass



- You can call the constructor of superclass to initialize some properties
- Function.call() allows you to set the this pointer and other parameters when invoking a function

```
function Person (n, a) {
  this.name = n;
  this.age = a;
}
function Teacher (n, a, sch) {
  Person.call(this, n, a);
  this.school = sch;
}
Teacher.prototype = new Person();

var bill = new Teacher ('Bill', 30, 'MPI');
```



# Example: defining a subclass



```
//  define a subclass 'Teacher' of the superclass  'Person'

function Teacher (n, a, sch) {
  Person.call(this, n, a);  // call constructor of Person
  this.school = sch;
}

// allow a teacher to inherit properties of Person
Teacher.prototype = new Person();

// define new properties, or redefine old properties
Teacher.prototype.teach = function ( ) { … }
Teacher.prototype.greet = function ( ) { … }

var bill = new Teacher ('Bill', 30, 'MPI');
bill.teach(); bill.greet(); bill.tellAge();
```

# Determining instance relationships

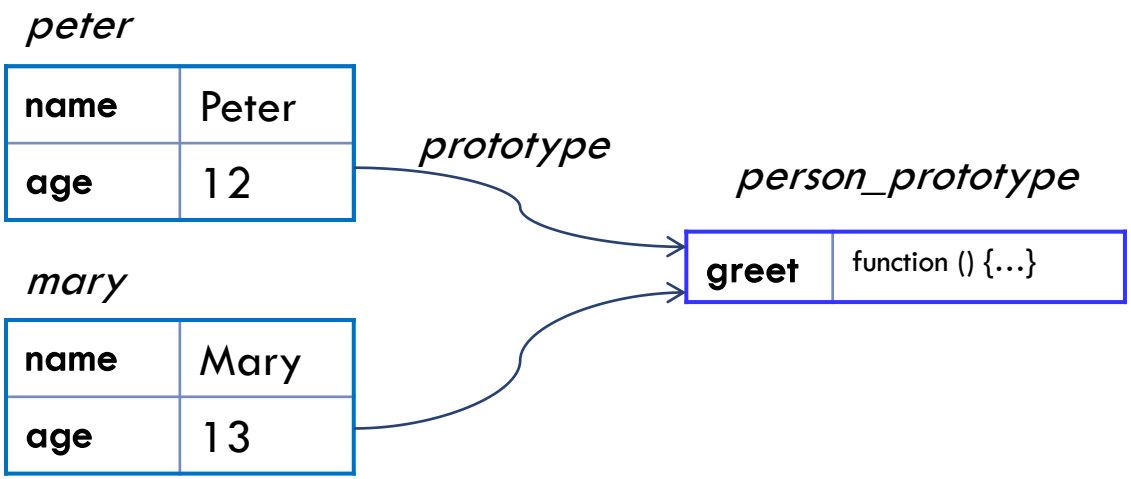


- The special operator instanceof tests whether an object is an instance of a class
  - Compare F.prototype of the constructor function F against each object along the prototype chain of the instance.

```
/* definition of the classes Person and Teacher omitted */

var peter = new Person('Peter', 12);
var bill = new Teacher ('Bill', 30, 'MPI');

peter instanceof Person; // return true;
peter instanceof Teacher; // return false;

bill instanceof Person; // return true;
bill instanceof Teacher; // return true;
```

# Setting up prototype w/o constructor






```
var person_prototype = {
  greet: function ( ) {
    console.log("Hello! I am "
    + this.name);
  }
};
```

```
var peter = Object.create(person_prototype);
peter.name='Peter'; peter.age=12;
var mary = Object.create(person_prototype);
mary.name='Mary'; mary.age=13;

peter.greet();   mary.greet();
```

- This example uses the new standard method Object.create( ) to create an object with a specified prototype object.

# C. Function and closure



- Function as objects
  - Constructor: Function. Prototype: Function.prototype
  - Assigned to variables, pass into a function as parameter, returned from a function
- Functional programming. Callback function
- Nested function. Function scope
- Closure

# this and arguments



- A function has access to two special variables
  - this – the current object
  - arguments – an array-like object containing arguments

```
function sum () {
  var i, sum = 0;
  for (i = 0; i < arguments.length; i++) {
    sum += arguments[i];
  }
  return sum;
}

sum(4,8,15,16,23,42); // 108
```

# Invocation pattern



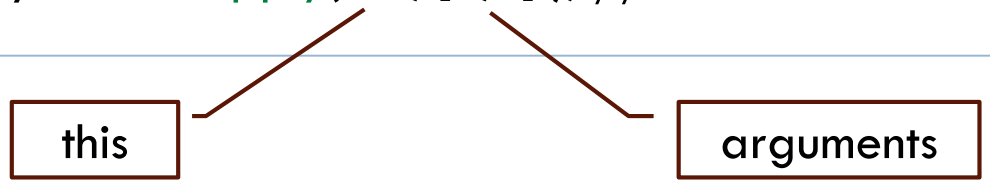
- Value of 'this' depends on how you invoke a function
  - Method, e.g. obj.func(args) :
    this refers to obj
  - Function, e.g. func(args) :
    this refers to the global object window
  - Constructor, e.g. obj = new Func(args) :
    this refers to the newly created object
  - Call/Appy, e.g. func.call(obj, args) :
    this refers to the parameter obj

# Function as an object, 1



- JavaScript function is a kind of objects
  - Constructor function is Function
  - The prototype object is Function.prototype, which defines some methods incl. call, apply and toString

```
function add (a,b) { return a+b; };

console.log("The function source is " + add.toString() );

var x = add.call(null, 1, 2); // similar to add(1,2)
var y = add.apply(null, [1,2] ); // same as above
```



# Example: apply



- Applying a method of the Person class to a non-Person

```
function Person (name, age) {
  this.name = name; this.age = age;
}

Person.prototype.greet = function () {
  console.log("Hello. My name is " + this.name
      + ". I'm " + this.age + " years old.");
}

var peter = new Person("Peter", 20);
peter.greet();

var robot = { name: 'Sonny', age: 2 }; // not a Person!
Person.prototype.greet.apply(robot, [ ]);
```

# Function as an object, 2



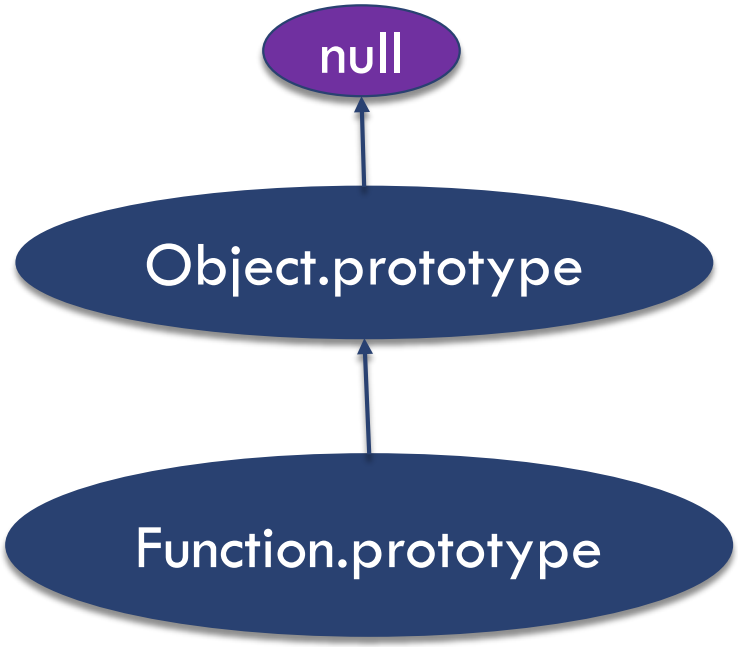
- You can save a function to a variable, pass a function as a parameter, and return a function from another function
- Parameter of function value is sometimes called callback

```
var add = function (a,b) { return a+b; };
function reduce(op, a, b, c) {
   var t = op(a,b);  return op(t,c);
}

reduce( add, 2, 3, 5); // return 10
reduce( function(x,y) { return x*y; }, 2, 3, 5); // return 30
```

```
function makeAdder(a) {
   return function (b) { return a+b; };
}

var add1 = makeAdder(1);
add1(2); // return 3
var add5 = makeAdder(5);
add5(6); // return 11
```

# Prototype of function





# Example, comparator function



- The optional parameter of Array.sort(cmp) is a callback function that determines the sort order of two given elements.

```
var P = [
  { name: 'Peter', age: 10 }, { name: 'Mary', age: 9 },
  { name: 'John', age: 11 } ];

function cmp (p1, p2) {
  if (p1.age<p2.age) return -1;
  if (p1.age==p2.age) return 0;
  return 1;
}
P.sort(cmp); // sort the persons in ascending order of age

// a shorter way to do the same thing
P.sort( function (p1,p2) { return p1.age-p2.age; } );
```

# Example, anonymous function



```
function filter (A, condition) {
   var ans = [ ];
   for (var i=0; i<A.length; i++) {
      if (condition(A[i])) ans.push( A[i] );
   }
   return ans;
}

var N = [ 6,4,8,3,500,0,9,700 ];

var even = filter(N, function (x) { return (x%2===0); } );

// return numbers larger than 100
var big = filter(N, function (x) { return (x>100); } );
```

# Iteration methods of Array



- Array defines several methods that invoke a callback function on each element in the array
  - filter(cb) – creates a new array with all the elements for which the function cb returns true
  - map(cb) – creates a new array with the results of calling the function cb on each element
  - every(cb) – returns true if every element satisfies the condition of the function cb
  - some(cb) – returns true if at least one element satisfies the condition of the function cb
  - reduce(cb, initialValue) – apply the function cb repeatedly on the elements to produce a single result
  - forEach(cb) – execute the function cb on each element

# Example, 1



```
var N = [ 6,4,3,5,0,9,8,7 ];

function isOdd(x) { if (x%2===1) return true; else return false; }
var odd = N.filter(isOdd);  // odd is [3,5,9,7]

var even = N.filter( function (x) { return (x%2===0); } );
// even is [6,4,0,8]

var words = even.map(  function (y) {
    var w = [ 'zero', 'one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine' ];
    return w[y];
} );
// words is ['six', 'four', 'zero', 'eight' ]
console.log(odd, even, words);
```

# Example, 2



```
var N = [ 9,3,5,4,2 ];

if ( N.every( function (x) { return (x<10); } ) ) {
   console.log('The array N only contains number less than 10');
}
if ( N.some( function (y) { return (y==5); } ) ) {
   console.log('The array N contains the number 5');
}

function add(a,b) { return a+b; }
var sum = N.reduce(add, 0); // 23
var product = N.reduce( function (x,y) { return x * y; } , 1 ); // 1080
console.log(sum, product);

N.forEach ( function (elem) { console.log(elem); } );  // log every element
```

# Variables have function scope



- A variable defined within a function is visible in the function
- Variable declarations are hoisted

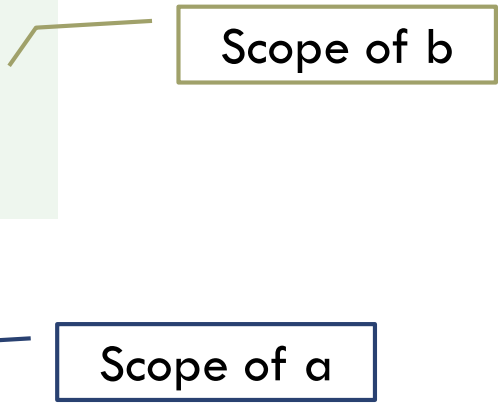
```
var a = 1;
function confusing() {
  a = 10;
  var a = 2;
  b = 20;
  for (var a=1; a<=3; a++) { var b; b+=a; }
  console.log('Local a = ', a, ', b = ', b);
}

confusing();
console.log('Global a = ', a);
```

# Nested functions



- An inner function has access to variables defined in the outer function

```
function outer ( ) {
  var a=10;
  function inner ( ) {
    var b=20;
    console.log('a=',a,', b=',b);
  }
  inner();
  // but b is not accessible here
}

outer();
```



# Closure



- Closure refers to the combination of a function and the environment in which the function is defined
- No matter where and when an inner function executes, it has access to variables defined in the outer function.

```
function translate( ) {
   var numbers = [ 'zero', 'one', 'two',
      'three', 'four', 'five', 'six', 'seven',
      'eight', 'nine' ];
   var words =  [2,3,5,7,9].map(
     function (e) { return numbers[e]; }
   );
  console.log('In English, ', words);
}

translate();
```

```
function compute( ) {
  var sum = 0;
  [2,3,5,7,9].forEach(
    function (e) { sum+=e; }
  );
  console.log('sum is ', sum);
}

compute();
```

# Example



- Question: select names of persons with age >= 10

```
var group = [
  { name: 'Peter', age: 10 }, { name: 'Mary', age: 9 },
  { name: 'John', age: 11 }, { name: 'Ann', age: 8 } ];

function select (g) {
  var ans = [ ];
  g.forEach( function(p) { if (p.age>=10) ans.push(p.name); });
  return ans;
}

select(group);
```

```
// same result
group.filter(
   function(p) { return p.age>=10 }
).map(
   function(p) { return p.name }
);
```

# Example: LCG random number generator



- A function to create a random number generator using Linear Congruential Generators

```
function makeLCG (seed, multiplier, increment, modulus) {
  var xn = seed;
  return function () {
      var xn_plus1 = (multiplier * xn + increment) % modulus;
      xn = xn_plus1;
      return xn_plus1;
  }
}

var random = makeLCG(404, 34, -102, 394);
var nums = [ ];
for (var i=0; i<50; i++) { nums.push( random() ); }
nums;
```

Return an inner function, which has a closure to access local variables and parameters of makeLCG.

The function 'random' produces a random number at each invocation.

# Usage of closure in event handlers



- Closure is especially handy to share data among event handlers

```
<script>
function registerEventHandlers () {
    var n = 0;
    var clockid;
    function tick ( ) {
        document.getElementById('count').innerHTML = n; n++;
    }
    document.getElementById('start').onclick =
        function ( ) {clockid = setInterval(tick, 500);} ;
    document.getElementById('stop').onclick =
        function ( ) {clearInterval(clockid);} ;
    document.getElementById('reset').onclick =
        function ( ) {n=0;} ;
    }
window.onload = registerEventHandlers;
</script>
```

Four inner functions sharing the same environment

# Private variables through closure



- All properties of JavaScript objects are public
- Closure can be used to implement private variables in JavaScript

```
var counter = function ( ) {
  var value = 0;
  return {
    increment: function (inc) {
      // increment the variable value of the outer function
      value += inc;
    },
    getValue: function  ( ) { return value; }
  };
} ( );
counter.getValue(); // return 0;
counter.increment(1);
counter.getValue(); // return 1;
```

The variable 'value' is accessible by the two inner functions.

This anonymous function is invoked immediately. It returns an object with two methods. We use this to create a closure.

# Part D. Regular expressions



- Regular expressions (also known as patterns) are used to search, replace, and extract information from strings
- Create JavaScript RegExp object in two ways
  - RegExp literal – usually preferred
  - RegExp constructor – when you need to dynamically construct the regular expression

```
// a pattern to match Macau tel numbers (8 digits)
var reTel = /^\d{8}$/;

// a pattern to match student ID in IPM, e.g. p0123456
var reStudid = new RegExp("p\\d{7}");
```

# Syntax of Regexp



```
var re = /p\d{7}@ipm\.edu\.mo/i;
```

- A regular expression consists of a combination of simple characters and special characters
  - Simple characters, e.g. numbers and letters, match themselves
  - Special characters have various functions
  - Flags at the end may modify the meaning of the pattern

| Special characters | Description |
|---|---|
| [‥] [^‥] . | Character class |
| * + ? {m,n} | Repetition |
| \| | Alternation |
| (‥) (?‥) | Grouping |
| ^‥$ | Position matching |
| \‥ | Escape character |

# Character classes



- Match one character from a set

| Symbol | Description |
|---|---|
| [xyz] | Match any one character enclosed in the character set. You may use a hyphen to denote range. E.g. /[0-9]/ is the same as /[0123456789]/ |
| [^xyz] | Match any one character *not* enclosed in the character set. |
| . | Match any character except newline or another Unicode line terminator. |
| \w | Match any alphanumeric character including the underscore. Equivalent to [a-zA-Z0-9_]. |
| \W | Match any single non-word character. Equivalent to [^a-zA-Z0-9_]. |
| \d | Match any single digit. Equivalent to [0-9]. |
| \D | Match any non-digit. Equivalent to [^0-9]. |
| \s | Match any single space character. Equivalent to [ \t\r\n\v\f]. |
| \S | Match any single non-space character. Equivalent to [^ \t\r\n\v\f]. |

# Example



| Regexp | matches |
|---|---|
| /java/ | 'java' |
| /javascript/i | 'Javascript', 'JavaScript', 'jaVAscRIPT', … |
| /iP[ao]d/ | 'iPad', 'iPod', but not 'iPxd', 'iPaod' |
| /[a-z][0-9]/ | a lower case letter followed by a digit, e.g. 'a1', 'x4', but not 'aa' |
| /[A-Z][a-z][a-z][a-z]/ | A four letter word that begins with a capital letter |
| /b.t/ | 'bat', 'bit', 'but', … but not 'beat', 'bt' |
| /\d\d\d/ | a 3 digit number, e.g. 123, 001, 888 |
| /\w\w\w/ | 'abc', '123', '1ef', 'xy9', … but not 'a-b', 'b c' |
| /[1-9]\d/ | '10', '11', '12', …'19', '20', '21', .. '90', '91', .. '99' |

# Repetition



- Repeat an item zero, once, or more times

| Symbol | Description | Example |
|---|---|---|
| {x} | Match exactly x occurrences of a regexp clause | /\d{5}/ matches 5 digits. |
| {x,} | Match x or more occurrences of a regexp clause | /\s{2,}/ matches at least 2 whitespace characters. |
| {x,y} | Matches x to y number of occurrences of a regexp clause | /\d{2,4}/ matches at least 2 but no more than 4 digits. |
| ? | Match zero or one occurrences. Equivalent to {0,1}. | /ab?c/ matches 'abc' or 'ac'. |
| * | Match zero or more occurrences. Equivalent to {0,}. | /we*/ matches 'w', 'wee', … |
| + | Match one or more occurrences. Equivalent to {1,}. | /fe+d/ matches 'fed', 'feed', … |

# Example



| Regexp | matches |
|---|---|
| /\d{8}/ | 8 digit number, e.g. Macau tel number |
| /\d+/ | A positive integer, e.g. '0', '2', '35', '650' |
| /lo+ng/ | 'long', 'loong', 'looong', … |
| /apples?/ | 'apple', 'apples' |
| /s[a-z]+s/i | A word that starts and ends with 's' |
| /[A-Z][a-z]*/ | A capital letter, followed by zero or more lower-case letter |
| /[01]+/ | '0', '1', '00', '01', '10', '11', … '1101' … |
| /[1-9]?[0-9]/ | '0', '1', '2', …,'9', '10', '11', … '98', '99' |
| /i[a-z]{0,18}n/ | 'in', 'indian', 'internationalization', … |
| /\d+[+\-*/]\d+/ *or* /\d+[\+\-*\/]\d+/ | '1+2', '20*5', '312/43', … |

# Character



| Symbol | Description | Example |
|---|---|---|
| \xxx | Matches the ASCII character expressed by the octal number xxx. | /\050/ matches left parentheses character "(" |
| \xdd | Matches the ASCII character expressed by the hex number dd. | /\x28/ matches left parentheses character "(" |
| \uxxxx | Matches the ASCII character expressed by the UNICODE xxxx. | /\u00A3/ matches "£" |
| \f \n \r \t \v | Match one control character, e.g. form feed, newline, carriage return, tab, vertical tab | |
| \$ * \\ | Match the special character after the escape character '\' literally | /\$2\.5/ matches '$2.5' |

# Grouping



- Use (‥) to group several characters in the pattern. You may then repeat the group with *, +, ?, {m,n}

| Regexp | matches |
|---|---|
| /peach(es)?/ | 'peach', 'peaches' |
| /(01){2,4}/ | '0101', '010101', '01010101' |
| /\d+(\.\d+)?/ | A decimal number, e.g. 12, 0.5, 12.34 |
| /\d+([+\-*/]\d+)+/ | '1+2*3', '10-2/3+68', '8*8', … |
| /<img\ssrc='[^']*'>/ | "<img src='a.jpg'>" *(file name may differ)* |
| /<img(\s\w+='[^']*')+>/ | "<img src='a.jpg' alt='abc'>", … *(one or more attributes)* |
| /\[(\d+(,\d+)*)?\]/ | *JavaScript array of numbers, e.g.* '[]', '[32]', '[1,2,3]' |

# Alternation



- x|y matches either x or y

| Regexp | matches |
|---|---|
| /apple\|orange/ | 'apple', 'orange' |
| /(Mr\|Mrs\|Miss) \w+/ | 'Mr Chan', 'Mrs Lei', 'Miss Chao' |
| /(comp\|meng\|math)\d\d\d/ | 'comp111', 'meng212', 'math321', … |
| /comp3(1[1234]\|21\|22)/ | 'comp311', 'comp312', 'comp313', 'comp314', 'comp321', 'comp322' |
| /\d\d?:\d\d(am\|pm)/ | '8:00am', '5:30pm', '11:30pm', … |
| /'\w*'\|"\w*"/ | JavaScript string literal that contains alphanumeric characters (and underscore) |
| /a\|b\|c\|d/ | The same as /[abcd]/ |

# Position matching



- match a substring that occurs at a specified location within the larger string

| Symbol | Description | Example |
|---|---|---|
| ^ | Only matches the beginning of a string | /^The/ matches 'The' in "The night" but not "In The Night" |
| $ | Only matches the end of a string | /and$/ matches 'and' in "Land" but not "landing" |
| \b | Matches any word boundary (test characters must exist at the beginning or end of a word within the string) | /\bnot\b/ matches the word 'not' in "This note about knot is not noteworthy." |
| \B | Matches any non-word boundary | /\Bor/ matches “or” in "normal" but not "origami." |

# Pattern flags



- Modifies the behavior of the regexp

| Symbol | Description | Example |
|---|---|---|
| i | Ignore the case of characters. | /The/i matches "the" and "The" and "tHe" |
| g | Global search for all occurrences of a pattern | /ain/g matches both "ain"s in "No pain no gain", instead of just the first.<br>Supported by string.match( ), string.replace( ), and regexp.exec( ) |
| m | Multiline (^ and $ can match line-ending characters) | /^The/m matches "The" at the beginning of each line. |
| | (may combine the flags) | |

# Using RegExp in JavaScript



- String and RegExp objects work with regular expression
  - regexp.test(string) : validation by pattern
  - string.search(regexp) : searching substring matching a pattern
  - string.match(regexp) : extracting matches to a pattern
  - string.split(regexp) : split a string by a delimiter specified as a pattern
  - regexp.exec(string) : similar to string.match(regexp), but may be executed repeatedly to find multiple matches
  - string.replace(searchValue, replaceValue) : replace a match with new value

# regexp.test(string)



- The test method returns true if the regexp matches the string; otherwise, it returns false.
  - often used to perform data validation

```
var s = prompt ("Your tel please");
// a pattern to match Macau tel numbers (8 digits)
var re = /^\d{8}$/;
if (re.test(s)) {
   alert("Thanks.");
} else {
   alert("This is not a tel no.");
}
```

```
if (/^\d{8}$/.test(s)) {
   …
}
```

This regexp matches exactly 8 digits. Notice the meaning of ^ and $.

# string.search(regexp)



- The search method returns the position of the first character of the first match of the regexp in the string, or -1 if the search fails.

```
var text = "Web apps use JavaScript, HTML and CSS.";
var re = /javascript/i;
var pos = text.search(re);
alert("javascript found at position " + pos);
```

Case-insensitive search of 'javascript' within the sentence.

# string.match(regexp)



- The match method returns substrings in the string that match the regexp.
  - if the regexp has the global flag (g), the method returns an array of all matches
  - if the regexp does not have the global flag, the method returns only the first match.

```
// find all prices in the string
var s = "$60 + $70 = $130";
var A = s.match(/\$\d+/g);
// A is ["$60", "$70", "$130"];
console.log(A);
```

Find out all prices in the string.

# string.split( )



- string.split(regexp) method splits the string into an array of substring. The parts were separated by delimiters matched by the regexp.

```
var s = "1+2*3-4";
var nums = s.split(/[+\-*/]/);
// nums is ["1", "2", "3", "4"]
```

Exercise: modify the pattern to handle spaces in math expression correctly. E.g. "1 + 2 * 3 – 4" should be broken as the same result.

# Matching sub-pattern



```
// a regexp to match a date like '1 Oct 2011'
// it has three capturing groups for day, month and year
var re = /(\d\d?) ([A-Z][a-z][a-z]) (\d{4})/;
```

- In addition to matching a pattern, some functions can also return parts of a match that correspond to *capturing groups*
- extracting data from string
  - string.match(regexp) without global flag : return first match and parts matching capturing groups
  - regexp.exec(string) : repeatedly match the pattern and sub-pattern

# string.match(regexp) without g flag



- If the regexp in string.match(regexp) does not have the global flag, the function returns a match and some parts in the match corresponding to capturing groups
- Simple parenthesis serves as capturing group

```
var addr = 'peter@abc.com';
var re = /\w+@\w+\.\w+/;
var m = addr.match(re);
// m is [ 'peter@abc.com' ]
```

```
var addr = 'peter@abc.com';
var re = /(\w+)@(\w+\.\w+)/;
var m = addr.match(re);
// m is [ 'peter@abc.com', 'peter', 'abc.com' ]
```

Two capturing groups: user name and domain name in an email address

# Capturing vs. non-capturing groups

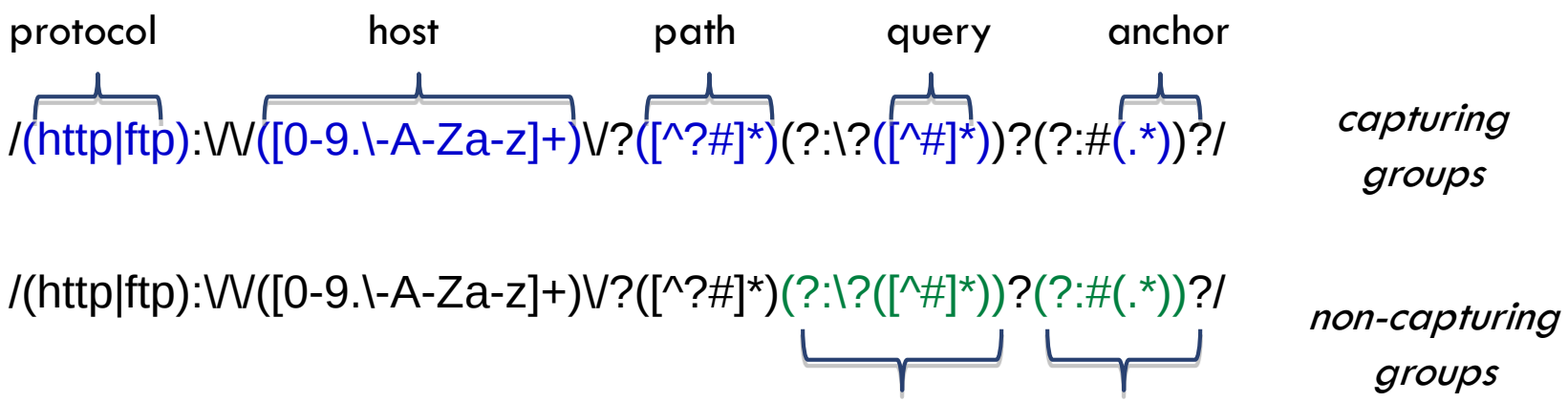


- Simple parenthesis (‥) has the side-effect of capturing substring matching the sub-pattern inside the parenthesis. If capturing is not desirable, you can use (?:‥) for grouping.

```
var s="1.2 + 3.4 = 3.6";
s.match(/\d+(\.\d+)?/); // return [ '1.2', '.2' ]
s.match(/\d+(?:\.\d+)?/); // return [ '1.2' ]
```

# Example






```
var re_url = /(http|ftp):\/\/([0-9.\-A-Za-z]+)\/?([^?#]*)(?:\?([^#]*))?(?:#(.*))?/;

var url = "http://www.example.com/home?a=1#top";
var result = url.match(re_url);
// result is [ "http://www.example.com/home?a=1#top",
  "http", "www.example.com", "home", "a=1", "top" ];
```

# regexp.exec(string)



- The exec method returns an array of substrings in the string that match the regexp and its capturing groups
  - This method works similar to string.match(regexp) if the regexp does not have the global flag.
  - if the regexp has the global flag (g), the method can be invoked repeatedly to find successive matches.
  - It returns null if no more matches

Important: missing the global flag will make an infinite loop!

```
// find tags in HTML source code
var text = "<p id='abc'>This is a <em>test</em></p>";
var tags = /<(\/?)([A-Za-z]+)([^<>]*)>/g;
var m;
while ((m=tags.exec(text)) !== null) {
   console.log(m[0] + ' :: ' + m[1] + ' # ' + m[2] + ' # ' + m[3]);
}
```

# string.replace( ), 1



- string.replace(searchValue, replaceValue) method does a search and replace operation on a string.
  - use a regexp with global flag to replace all occurrences
  - in replaceValue, $1,$2 .. refer to value of capturing groups
    - $$ - to escape $

```
var re = /\b(teh|tghe)\b/g;
var s = 'teh man is going to tghe station';
s.replace(re, 'the');  // return "the man is going to the station"

var re = /(\d\d?) ([A-Z][a-z][a-z]) (\d{4})/;
s = 'Today is 10 Sep 2011';
s.replace(re, '$2 $1, $3');  // return "Today is Sep 10, 2011"
```

# string.replace( ), 2



- In string.replace(searchValue, replaceValue) method, replaceValue can be a function. The function parameters are the matched string and the capturing groups.

```
var characters = {
  '<' : '&lt;',       '>' : '&gt;',
  '&' : '&amp;',    '"' : '&quot'
};

var s = "< & >";
var b = s.replace(/[<>&"]/g,
        function (c) { return characters[c]; } );
```