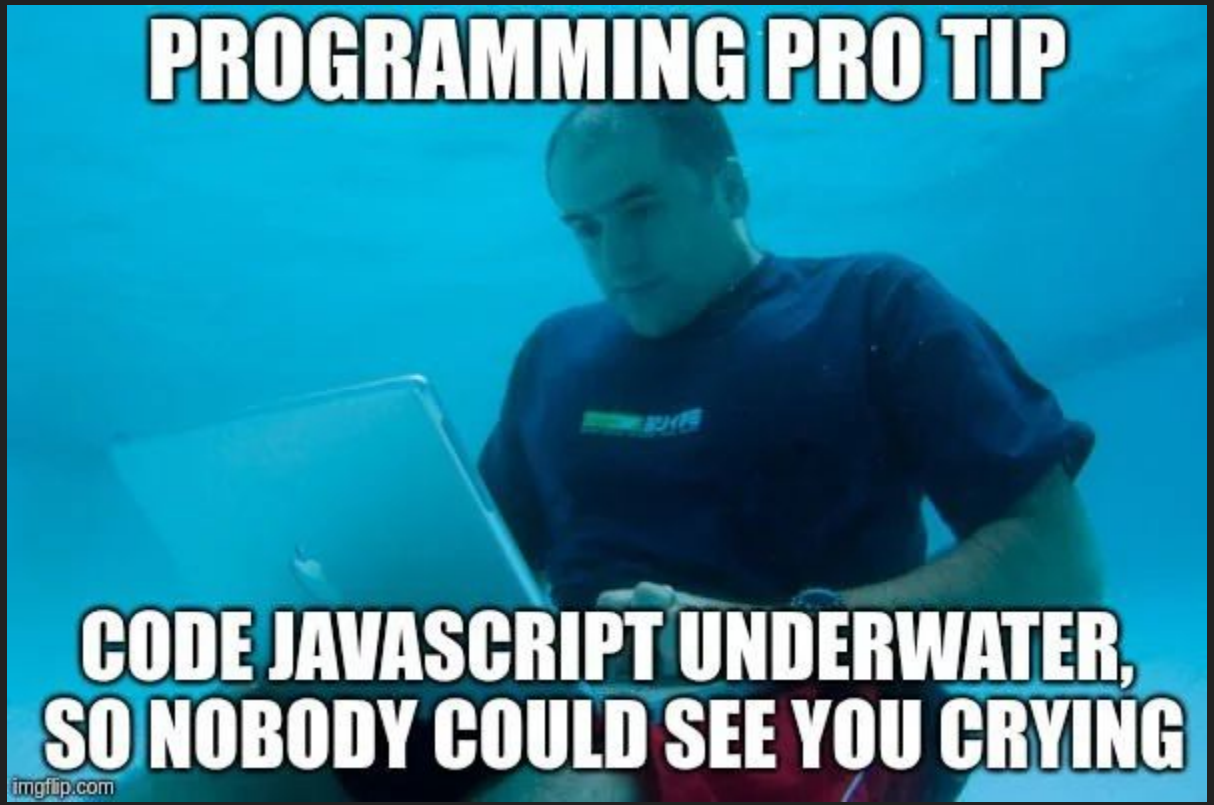
Javascript

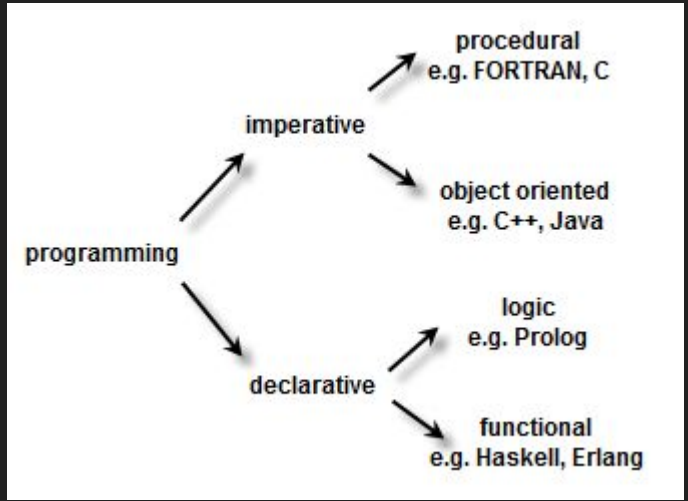
Introduction for Programmers


PROGRAMMING PRO TIP
CODE JAVASCRIPT UNDERWATER,
SO NOBODY COULD SEE YOU CRYING
imgflip.com

Javascript

As a multi-paradigm language, JavaScript supports:

1. event-driven
2. functional,
3. and imperative (including object-oriented and prototype-based) programming styles



Javascript Variables

```
var city = "Copenhagen";
var foundingYear = 1167;
var isAwesome = true;
```

Javascript Variables

```
var city = "Copenhagen";
var foundingYear = 1167;
var isAwesome = true;
```

Variable hoisting

Moving of variable declarations to the top

```
function printInfo(){
    console.log(city + ", " + foundingYear);
}
printInfo();

var city = "Copenhagen";
var foundingYear = 1167;
```

=

```
var city;
var foundingYear;

function printInfo(){
    console.log(city + ", " + foundingYear);
}
printInfo();

city = "Copenhagen";
foundingYear = 1167;
```

Javascript Variables in ES6

`var`: uses variable hoisting, local in a function, but not in a sub-block (eg. if)

`let`: avoids variable hoisting, respects scope of sub-blocks

`const`: immutable, object attributes can be changed

(nothing): creates a global variable using var

```
function varTest() {
  var x = 1;
  if (true) {
    var x = 2;  // same variable!
    console.log(x);  // 2
  }
  console.log(x);  // 2
}

function letTest() {
  let x = 1;
  if (true) {
    let x = 2;  // different variable
    console.log(x);  // 2
  }
  console.log(x);  // 1
}
```

Mini Exercise #1

Fix the following problem

Source is on Canvas:
lecture02/class02-exercise01.html

Don’t reveal the solution!

```
// problem: why is it logging out 0?
// it should be 12 since 4 * 4 is 16,
// 2 * 2 is 4 and 16 - 4 is 12
// fix it to log out 12!
function square(num){
    result = num * num
    return result
}
result = square(4)
result2 = square(2)
var subtracted = result - result2
console.log("result: " + subtracted)
```

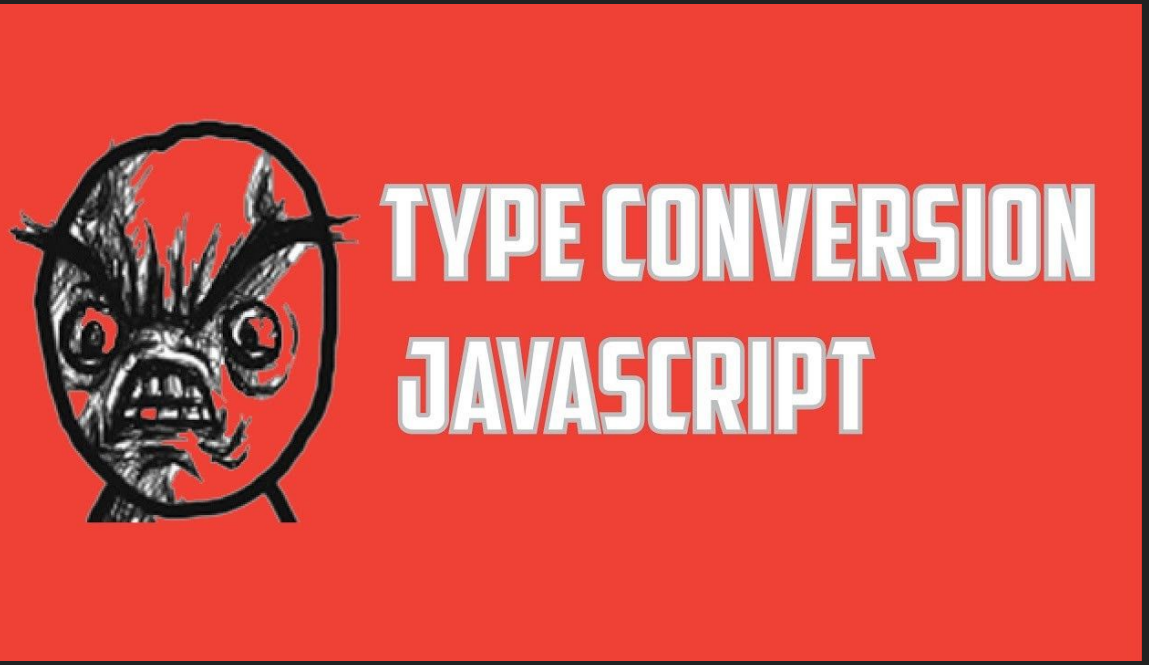
Relational Operators

Operators	Meaning	Example	Result
<	Less than	5<2	False
>	Greater than	5>2	True
<=	Less than or equal to	5<=2	False
>=	Greater than or equal to	5>=2	True
==	Equal to	5==2	False
! =	Not equal to	5! =2	True
===	Equal value and same type	5 === 5	True
		5 === “5”	False
! ==	Not Equal value or Not same type	5 ! == 5	False
		5 ! == “5”	True

Relational Operators

Operators	Meaning	Example	Result
<	Less than	5<2	False
>	Greater than	5>2	True
<=	Less than or equal to	5<=2	False
>=	Greater than or equal to	5>=2	True
==	Equal to	5==2	False
! =	Not equal to	5! =2	True
===	Equal value and same type	5 === 5	True
		5 === “5”	False
! ==	Not Equal value or Not same type	5 ! == 5	False
		5 ! == “5”	True

www.geekyshows.com


TYPE CONVERSION
JAVASCRIPT


0 == "0"
true
JS
0 == []
true
JS
So, if
0 == "0"
and
0 == []
then surely "0" == []
That makes sense to me.
"0" == []
false

The solution:

Aways use ===

Functional Programming

```
let text = "OMG its Javascript!";
function printText(){
    console.log(text);
}
printText();
```

```
let text = "OMG its Javascript!";
printText();
function printText(){
    console.log(text);
}
```

Javascript Objects

Everything in Javascript is an object.

Basic objects can be created and defined by:

```
var city = {
    name: "Copenhagen",
    foundingYear: 1167,
    isAwesome: true
}
```

Objects can own functions

```
var city = {
    name: "Copenhagen",
    foundingYear: 1167,
    isAwesome: true,
    print: function (){
        console.log(this.name + ", " + this.foundingYear);
    }
}
```

`this` in a function always refers to the owner of the function, in a global context **`this`** is the global object.

Attributes and functions can be added on runtime

```
var city = {
    name: "Copenhagen",
    foundingYear: 1167,
    isAwesome: true,
    print: function (){
        console.log(this.name + ", " + this.foundingYear);
    }
}
```

=

```
var city = {
    name: "Copenhagen",
    foundingYear: 1167,
    isAwesome: true
}

city.print = function (){
    console.log(this.name + ", " + this.foundingYear);
}
```

The this keyword

```
var city = {
    name: "Copenhagen",
    foundingYear: 1167,
    isAwesome: true,
    print: function (){
        console.log(this.name + ", " + this.foundingYear);
    }
}

function printThis(){
    console.log(this);
}
```

`this` refers to the city object (owner of **`print()`**)

`this` refers to the global object

What will be printed?

```
var city = {
    name: "Copenhagen",
    foundingYear: 1167,
    isAwesome: true,
    print: function (){
        console.log(this.name + ", " + this.foundingYear);
    }
}

function printThis(){
    console.log(this);
}

city.printThis = printThis;

printThis();
city.printThis();
```

What will be printed?

```
var city = {
    name: "Copenhagen",
    foundingYear: 1167,
    isAwesome: true,
    print: function (){
        console.log(this.name + ", " + this.foundingYear);
    }
}

function printThis(){
    console.log(this);
}

city.printThis = printThis;

printThis();
city.printThis();
```

```
▶ Window {postMessage: ƒ, blur: ƒ, focus: ƒ, close: ƒ, parent: Window, …}
▶ {name: "Copenhagen", foundingYear: 1167, isAwesome: true, print: ƒ, printThis: ƒ}
```

First-class functions

Functions are treated like any other variable.

- can be passed as an argument to other functions
- can be returned by another function
- can be assigned as a value to a variable.

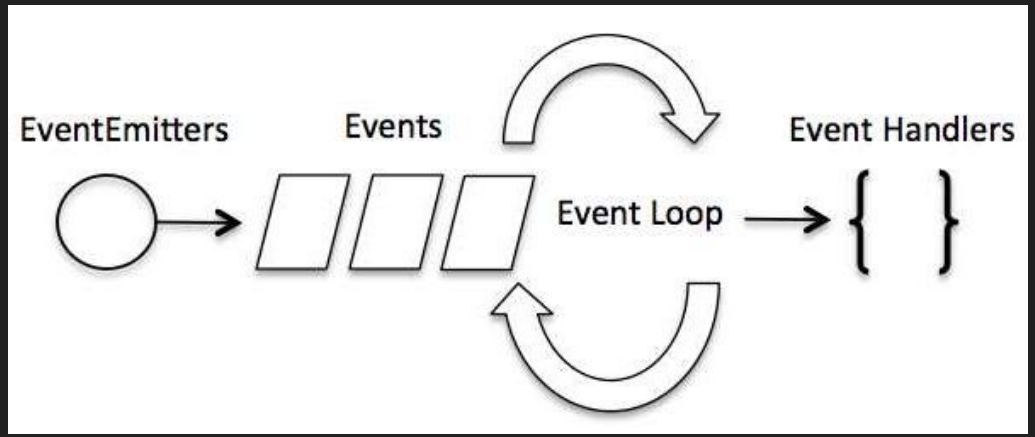
```
var city = {
    name: "Copenhagen",
    foundingYear: 1167,
    isAwesome: true
}

city.print = function (){
    console.log(this.name + ", " + this.foundingYear);
}
```

`print` is a variable that has the value of a function

Passing functions as parameters

```
doIt(theThingToDo);

function doIt(thing){
    thing();
}

function theThingToDo(){
    // some thing to do here
}
```

Event-driven Programming




```
window.addEventListener("keydown", keydown, false);
window.addEventListener("keyup", keyup, false);

function keydown(event) {
    keyMap[event.key] = true;
}

function keyup(event) {
    keyMap[event.key] = false;
}
```

Javascript Arrays

```
let x = new Array(10);
```

Javascript is not typesafe!

```
let x = new Array(10);


x[0] = 42;
x[1] = "This is element 1";
x[3] = {
    name: "Copenhagen",
    foundingYear: 1167
};
```

Javascript Arrays

```
let x = new Array(10);
```

Javascript is not typesafe!

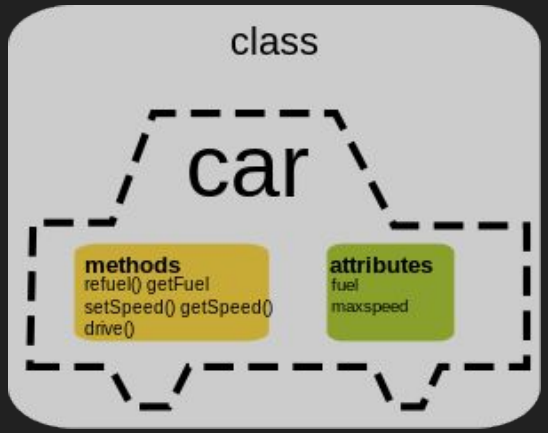
```
let x = new Array(10);


x[0] = 42;
x[1] = "This is element 1";
x[3] = {
    name: "Copenhagen",
    foundingYear: 1167
};
```

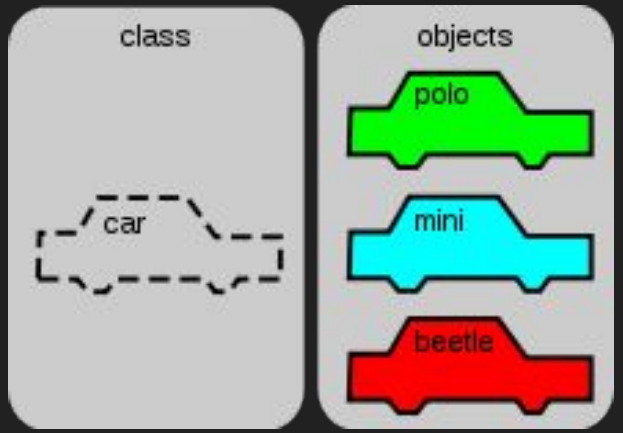
Typed Arrays

Important for later when we start with WebGL

```
var jsArray = new Array();          // Make an array
var ft = new Float32Array(jsArray); // Array of floats
var ct = new Uint16Array(jsArray);  // Array of ints
```

Object-oriented Programming and Classes





```
Car = class {
    constructor(name, fuel, speed){
        this.name = name;
        this.fuel = fuel;
        this.speed = speed;
        this.distance = 0;
    }
    refuel(fuel){
        this.fuel += fuel;
    }
    drive(distance){
        // use up 6 liters per 100km
        this.fuel -= distance/100 * 6;
        this.distance += distance;
    }
    print(){
        document.write(
            "<br/>" + this.name +
            ", fuel: " + this.fuel +
            ", speed: " + this.speed +
            ", distance: " + this.distance);
    }
};
```

```
let myCar = new Car("VW", 100, 180);
myCar.print();
myCar.drive(558); // drive from Copenhagen to Osnabrück
myCar.print();
myCar.refuel(40);
myCar.print();
```

VW, fuel: 100, speed: 180, distance: 0
VW, fuel: 66.52, speed: 180, distance: 558
VW, fuel: 106.52, speed: 180, distance: 558

Mini Exercise #2

Fix the following problem:

Tipp: Look up the documentation of setTimeout!

Source is on Canvas:
lecture02/class02-exercise02.html

Don’t reveal the solution!

```
"use strict";
// problem: why does Ralph only bark once?
// fix it to make Ralph bark twice!
class Dog {
    constructor(name){
        this.name = name;
    }
    bark() {
        console.log(this.name);
    }
}
// make a new instance of a Dog
var ralph = new Dog('Ralph');
// make Ralph bark once immediately
ralph.bark();
// in 1 second we want Ralph to bark again
setTimeout(ralph.bark, 1000);
```

Class declarations are not hoisted!

```
1  const p = new Rectangle(); // ReferenceError
2
3  class Rectangle {}
```

Fields and access modifiers

None (public)

```
Car = class {
    constructor(name, fuel, speed){
        this.name = name;
        this.fuel = fuel;
        this.speed = speed;
        this.distance = 0;
    }
```

public

```
Car = class {
    name;
    fuel;
    speed;
    distance;
    constructor(name, fuel, speed){
        this.name = name;
        this.fuel = fuel;
        this.speed = speed;
        this.distance = 0;
    }
```

for private

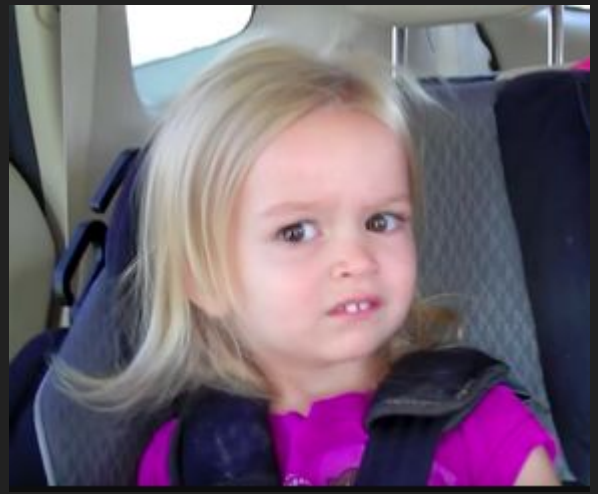
```
Car = class {
    #name;
    #fuel;
    #speed;
    #distance;
    constructor(name, fuel, speed){
        this.#name = name;
        this.#fuel = fuel;
        this.#speed = speed;
        this.#distance = 0;
    }
```

ES5 “classes” - prototype based object-orientation

```
function PrototypeCar(name, fuel, speed){
    this.name = name;
    this.fuel = fuel;
    this.speed = speed;
}

PrototypeCar.prototype.drive = function (distance) {
    this.fuel -= distance/100 * 6;
    this.distance += distance;
};

var myCar = new PrototypeCar("VW", 100, 180);
myCar.drive(558);
```

ES5 “classes” - prototype based object-orientation

```
function PrototypeCar(name, fuel, speed){
    this.name = name;
    this.fuel = fuel;
    this.speed = speed;
}

PrototypeCar.prototype.drive = function (distance) {
    this.fuel -= distance/100 * 6;
    this.distance += distance;
};

var myCar = new PrototypeCar("VW", 100, 180);
myCar.drive(558);
```



https://www.destroyallsoftware.com/talks/wat