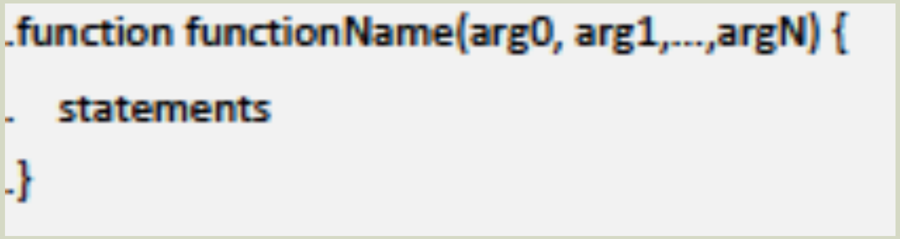
JAVASCRIPT

Unit 3

INDEX

1 – FUNCTIONS

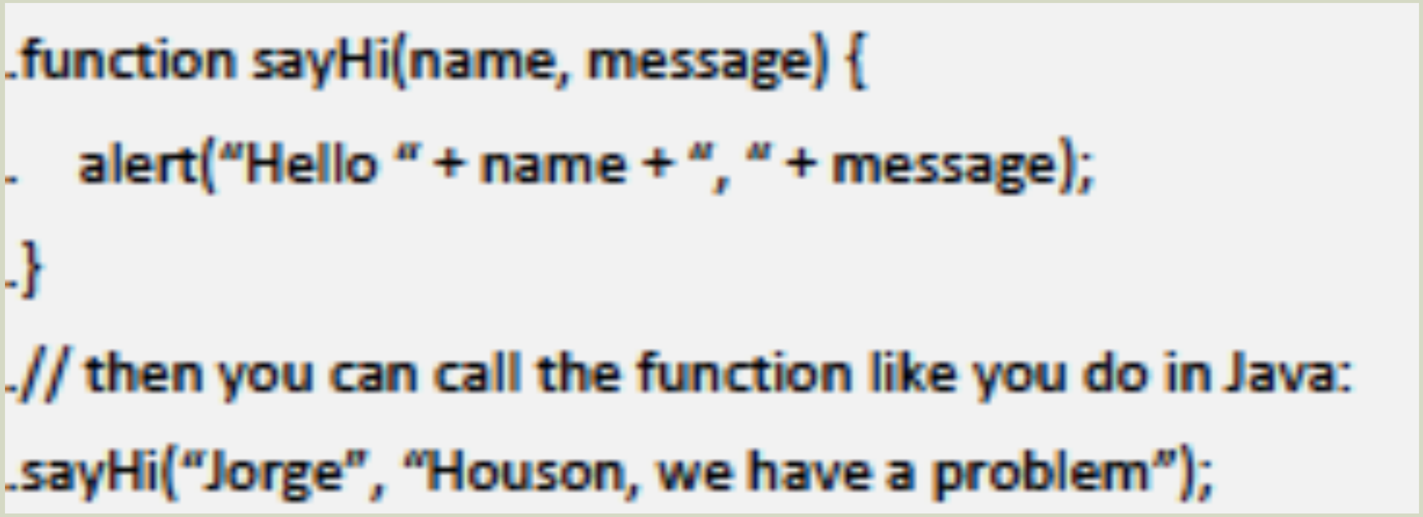
- Are the core of any language because they allow the encapsulation of statements.
- Functions accepts n arguments.
- The basic syntax is as follows.



```
.function functionName(arg0, arg1,...,argN) {
.    statements
.}
```

1 – FUNCTIONS

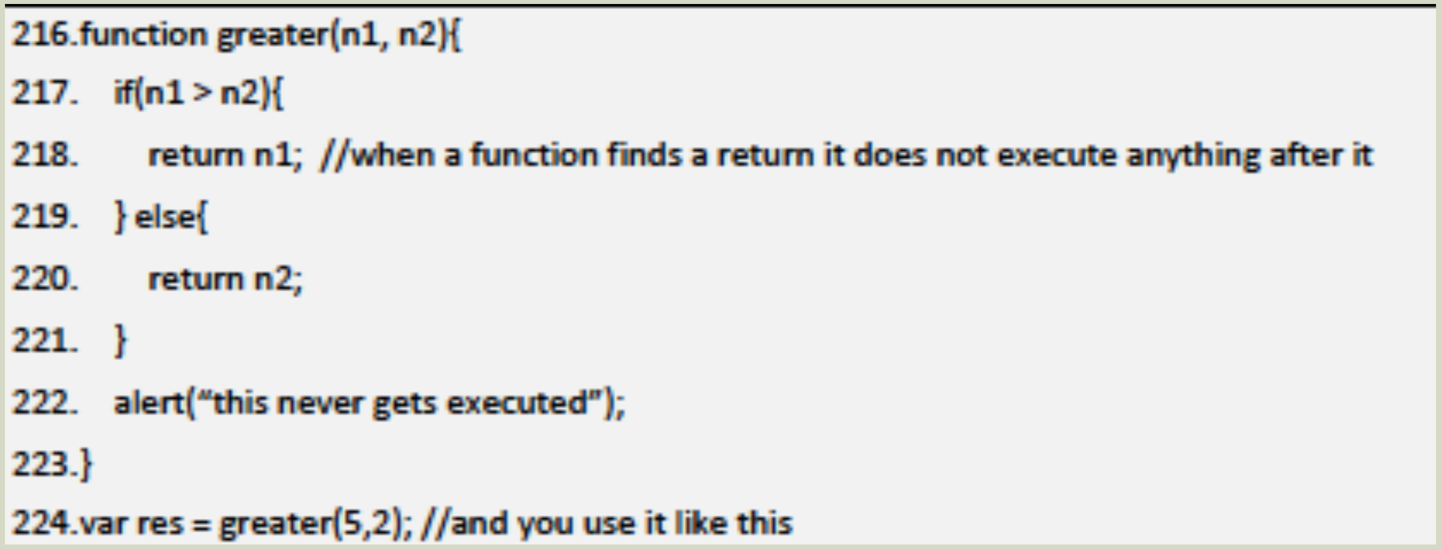
- Example:



```
.function sayHi(name, message) {
.    alert("Hello " + name + ", " + message);
.}
.// then you can call the function like you do in Java:
.sayHi("Jorge", "Houson, we have a problem");
```

1 – FUNCTIONS

- Functions in JS don’t need to specify a return value.
- Any function can return a value at any time.
- You can have multiple returns, but only one will be executed



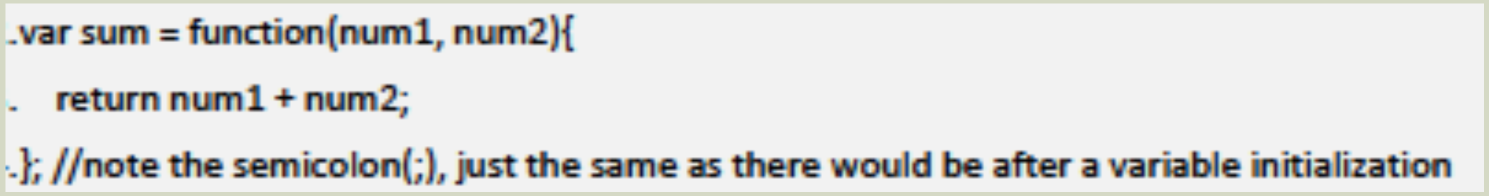
```
function greater(n1, n2){
    if(n1 > n2){
        return n1;  //when a function finds a return it does not execute anything after it
    } else{
        return n2;
    }
    alert(“this never gets executed”);
}
var res = greater(5,2); //and you use it like this
```

1 – FUNCTIONS

- Funcitons args in JS don’t behave in the same way as functions args in most other languages..
- Doesn’t care how many args are passed in because the args are represented as an array internally.
- You can access each arg using arguments[0], arguments[1], etc.
- Thus you can’t overload functions as you do in Java.

1 – FUNCTIONS

- You can also create anonymous functions, also called function expressions.



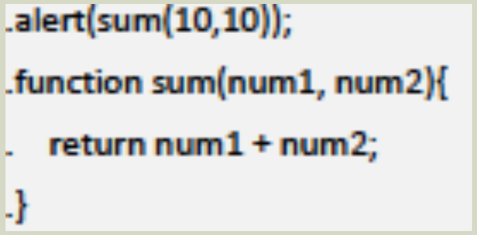
```
var sum = function(num1, num2){
    return num1 + num2;
}; //note the semicolon(;), just the same as there would be after a variable initialization
```

  - Doing this, variable sum is defined and initialized to be a function.
  - There is no name for the function because we reference it using the variable name.

    sum(3,7); //10

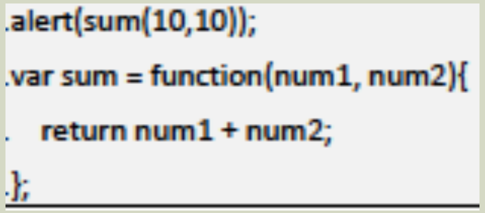
1 – FUNCTIONS

- What’s the difference between function declaration (with name) and function expression (without name)?
  - Function declarations are read and avaliable in an execution context before any code is executed whereas function expressions aren’t complete until the execution reaches that line of code.



```
.alert(sum(10,10));
.function sum(num1, num2){
.    return num1 + num2;
.}
```

✓



```
.alert(sum(10,10));
.var sum = function(num1, num2){
.    return num1 + num2;
.};
```

✗

2 – ARRAYS

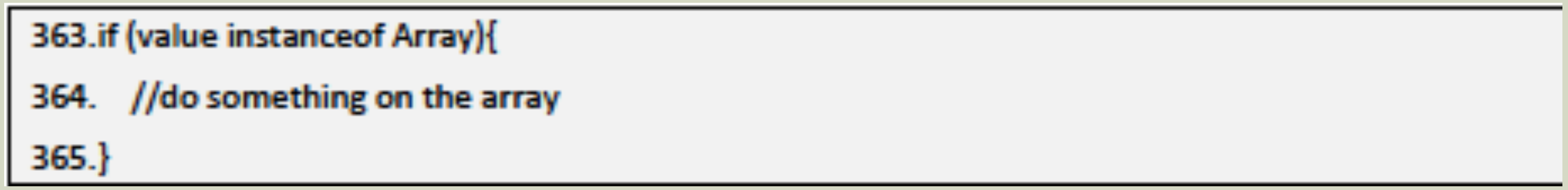
- We already know how to create and manipulate objects basically

  var colors = [“red”,”blue”,”green”]

  alert(colors.length); //3

  alert(colors[2]); //green

- Determining if an object is an array

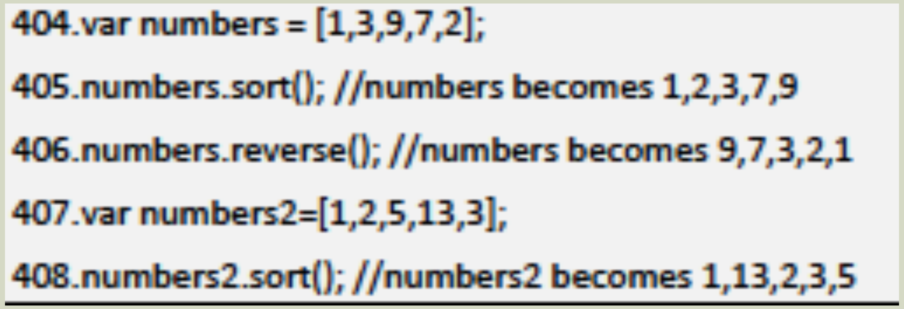


```
if (value instanceof Array){
    //do something on the array
}
```

2 – ARRAYS

- Stack and queue
  - push() → Insert items to the end of an array
  - pop() → removes the last element of an array
  - shift() → the same as pop but the first element
  - unshifht() → insert elements to the front of an array

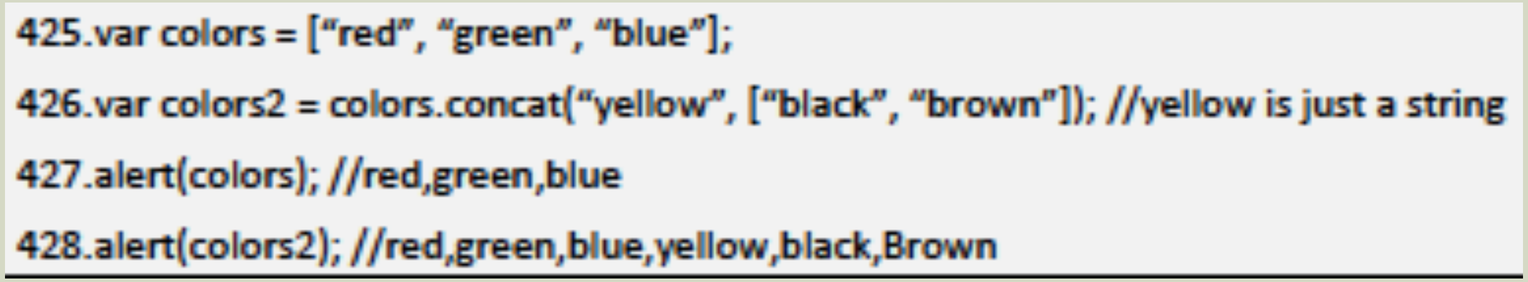
- Reordering
  - sort() and reverse()→ NOT RECOMMENDED



```
var numbers = [1,3,9,7,2];
numbers.sort(); //numbers becomes 1,2,3,7,9
numbers.reverse(); //numbers becomes 9,7,3,2,1
var numbers2=[1,2,5,13,3];
numbers2.sort(); //numbers2 becomes 1,13,2,3,5
```

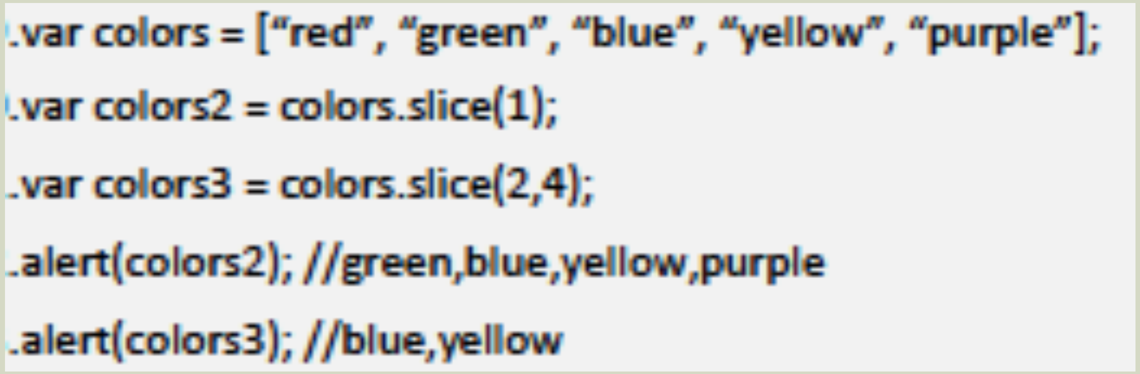
2 – ARRAYS

- Concat



```
var colors = ["red", "green", "blue"];
var colors2 = colors.concat("yellow", ["black", "brown"]); //yellow is just a string
alert(colors); //red,green,blue
alert(colors2); //red,green,blue,yellow,black,Brown
```

- Slice



```
var colors = ["red", "green", "blue", "yellow", "purple"];
var colors2 = colors.slice(1);
var colors3 = colors.slice(2,4);
alert(colors2); //green,blue,yellow,purple
alert(colors3); //blue,yellow
```

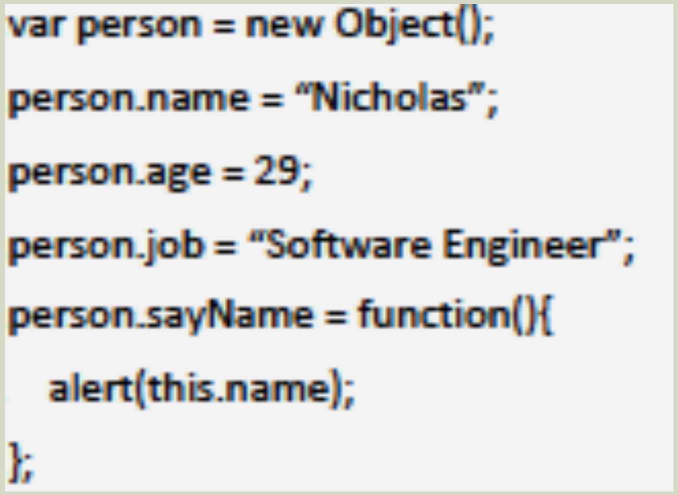
3 – USER-DEFINED OBJECTS (UDO)

- Js is not a OO language that’s why an object is defined as an “unordered collections of properties each of which contains a primitive value, object or function”

3 – USER-DEFINED OBJECTS (UDO)

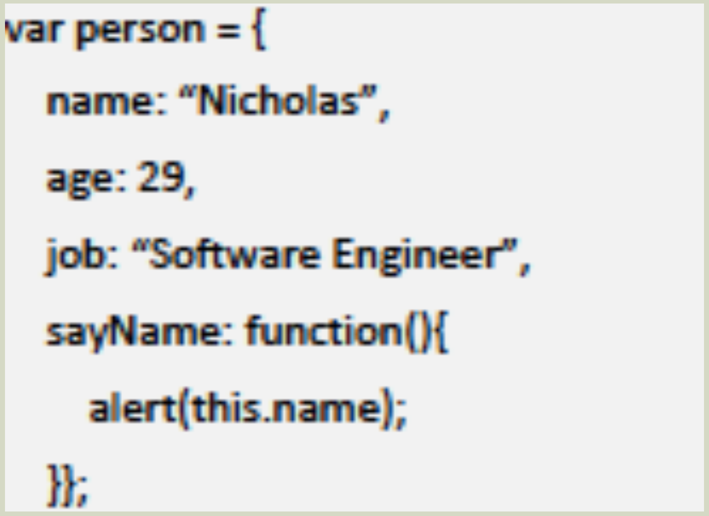
- We can create UDO:

Creating a new instance of Object and adding properties and functions to it



```
var person = new Object();
person.name = “Nicholas”;
person.age = 29;
person.job = “Software Engineer”;
person.sayName = function(){
   alert(this.name);
};
```

Using object literal notation:



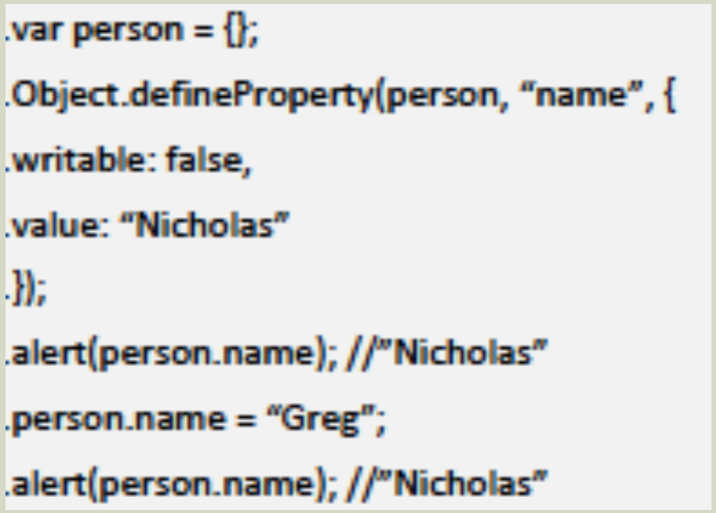
```
var person = {
   name: “Nicholas”,
   age: 29,
   job: “Software Engineer”,
   sayName: function(){
      alert(this.name);
   }};
```

3 – USER-DEFINED OBJECTS (UDO)

- Data properties:
  - Configurable: If the property may be redefined. By default is true.
  - Enumerable: Indicates if the property will be returned in a for-in loop. By default is true.
  - Writable: Indicates if the property’s value can be changed. By default is true.
  - Value: Contains the actual data value for the property.

3 – USER-DEFINED OBJECTS (UDO)

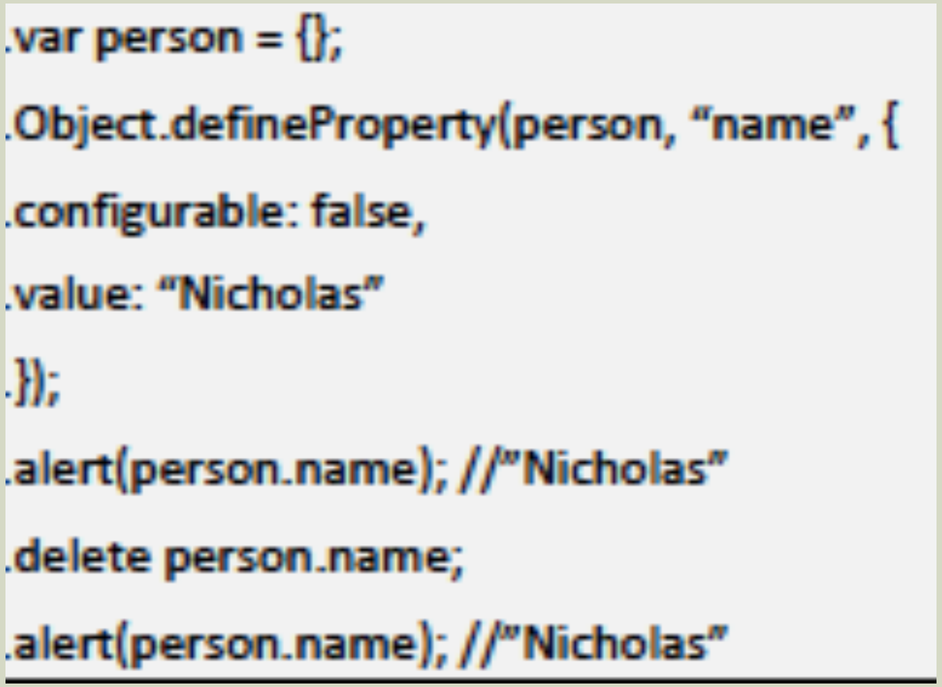
- Object.defineProperty() method
  - Accepts three arguments:
    - 1- the object on which the property should be added or modified
    - 2- the name of the property
    - 3- descriptor object (match the attribute names: configurable, enumerable, writable and value)



```
var person = {};
Object.defineProperty(person, "name", {
writable: false,
value: "Nicholas"
});
alert(person.name); //"Nicholas"
person.name = "Greg";
alert(person.name); //"Nicholas"
```

3 – USER-DEFINED OBJECTS (UDO)

- What happens here?

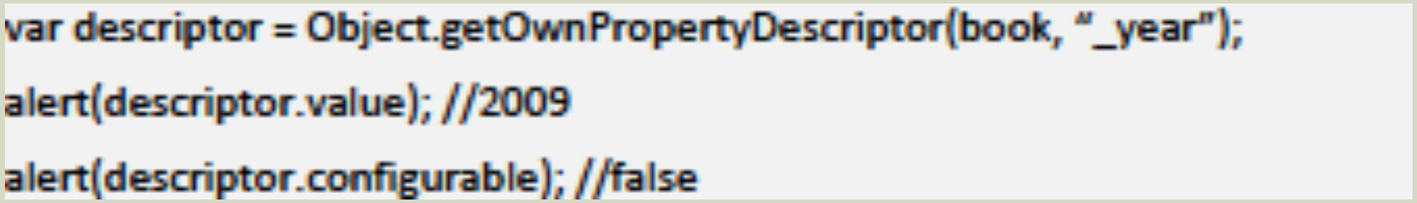


```
.var person = {};
.Object.defineProperty(person, “name”, {
.configurable: false,
.value: “Nicholas”
.});
.alert(person.name); //”Nicholas”
.delete person.name;
.alert(person.name); //”Nicholas”
```

Setting configurable to false means that the property cannot be removed from the object.

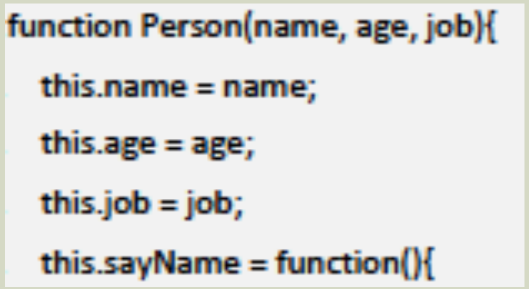
3 – USER-DEFINED OBJECTS (UDO)

- We can also define Multiple Properties at once.
- We can read property attributes.

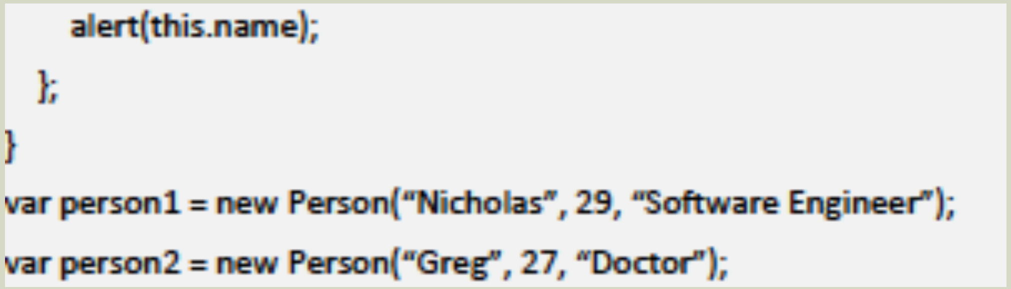


```
var descriptor = Object.getOwnPropertyDescriptor(book, "_year");
alert(descriptor.value); //2009
alert(descriptor.configurable); //false
```

- Constructor Pattern



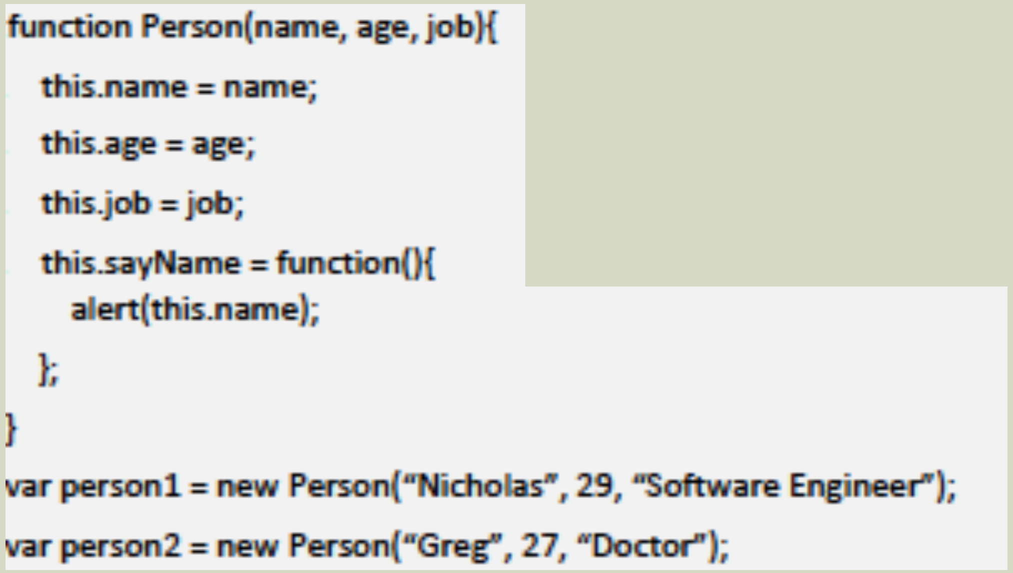
```
function Person(name, age, job){
    this.name = name;
    this.age = age;
    this.job = job;
    this.sayName = function(){
```



```
        alert(this.name);
    };
}
var person1 = new Person("Nicholas", 29, "Software Engineer");
var person2 = new Person("Greg", 27, "Doctor");
```

3 – USER-DEFINED OBJECTS (UDO)

- Constructor Pattern



```
function Person(name, age, job){
  this.name = name;
  this.age = age;
  this.job = job;
  this.sayName = function(){
    alert(this.name);
  };
}
var person1 = new Person("Nicholas", 29, "Software Engineer");
var person2 = new Person("Greg", 27, "Doctor");
```

4.- FORMS AND CANVAS

- As the other HTML elements, form and canvas have their own methods and properties in JavaScript.
  - Forms
    - We can change the method (POST, GET)
    - reset();
    - submit();
    - We can change the action
    - And much more
  - Canvas
    - As with other elements, widht and height attributes are also avaliable as properties in JavaScript.

5.- EVENTS

- JavaScript applications commonly respond to user actions like clicking on a button. These actions are called events, and the anonymous functions that handle the events are called event handlers.
- To make that happen, you have to attach the function to the events.

5.- EVENTS

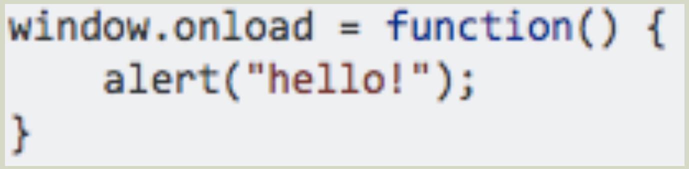
- There are several numerous categories of events that can occur in a web browser.
- Event groups:
  - User interface events: browser event.
  - Focus events: when gains or loses focus.
  - Mouse events: fired by the mouse
  - Wheel events: fired by the mouse wheel
  - Text event: fired when text is input into the document
  - Keyboard events: when the keyboard is used.
  - Etc…

5.- EVENTS

- There are several numerous categories of events that can occur in a web browser.
- Event groups:
  - User interface events: browser event.
  - Focus events: when gains or loses focus.
  - Mouse events: fired by the mouse
  - Wheel events: fired by the mouse wheel
  - Text event: fired when text is input into the document
  - Keyboard events: when the keyboard is used.
  - Etc…

5.- EVENTS

- Window Event attributes: events fired by the window object.
  - onload: fires after the page is finished loading.



```
window.onload = function() {
    alert("hello!");
}
```

- onresize: fires when the browser window is resized

5.- EVENTS

- Form Events: Events fired by actions inside a form (also applies to almost all HTML elements, but is most used in forms).
  - onfocus
  - onblur
  - oninput
  - onsubmit

5.- EVENTS

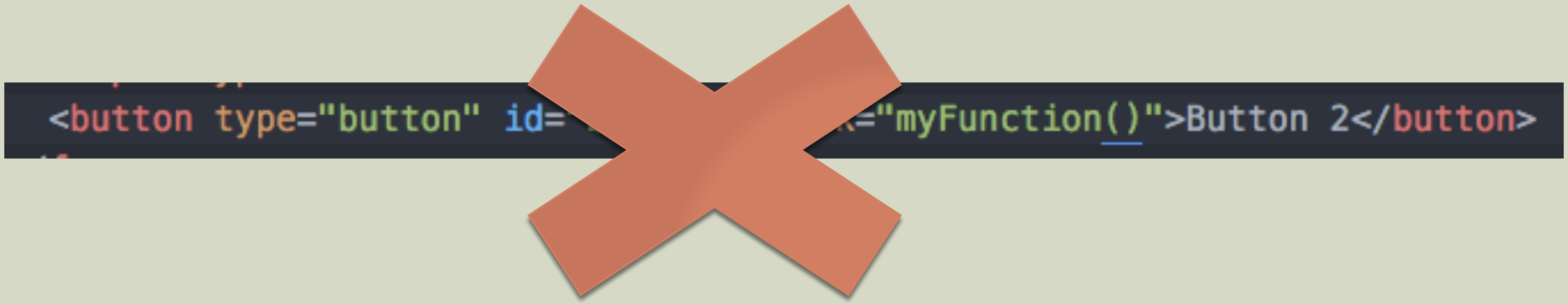
- Keyboard Events
  - onkeydown
  - onkeypress
  - onkeyup

- Mouse Events
  - onclick
  - ondrag
  - ondrop
  - onmouseover

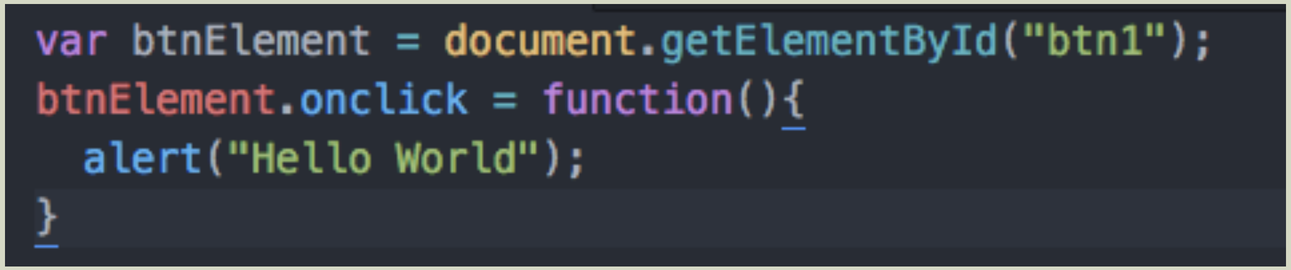
- http://www.w3schools.com/jsref/dom_obj_event.asp

5.- EVENTS

- Event Handlers



```
<button type="button" id= [illegible] ="myFunction()">Button 2</button>
```

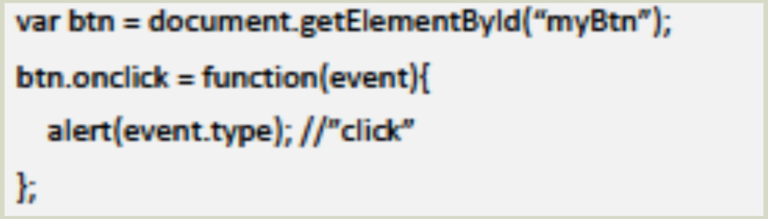


```
var btnElement = document.getElementById("btn1");
btnElement.onclick = function(){
  alert("Hello World");
}
```

5.- EVENTS

- Event Handlers
  - Can be removed setting the property to null.
    - btnElement.onClick = null;

- There is an object called Event:
- You can get it retrieving it as an arg in your function.



```
var btn = document.getElementById("myBtn");
btn.onclick = function(event){
   alert(event.type); //"click"
};
```