

# Web Programming (CSci 130)

Department of Computer Science  
College of Science and Mathematics  
California State University Fresno  
H. Cecotti

# Learning outcomes

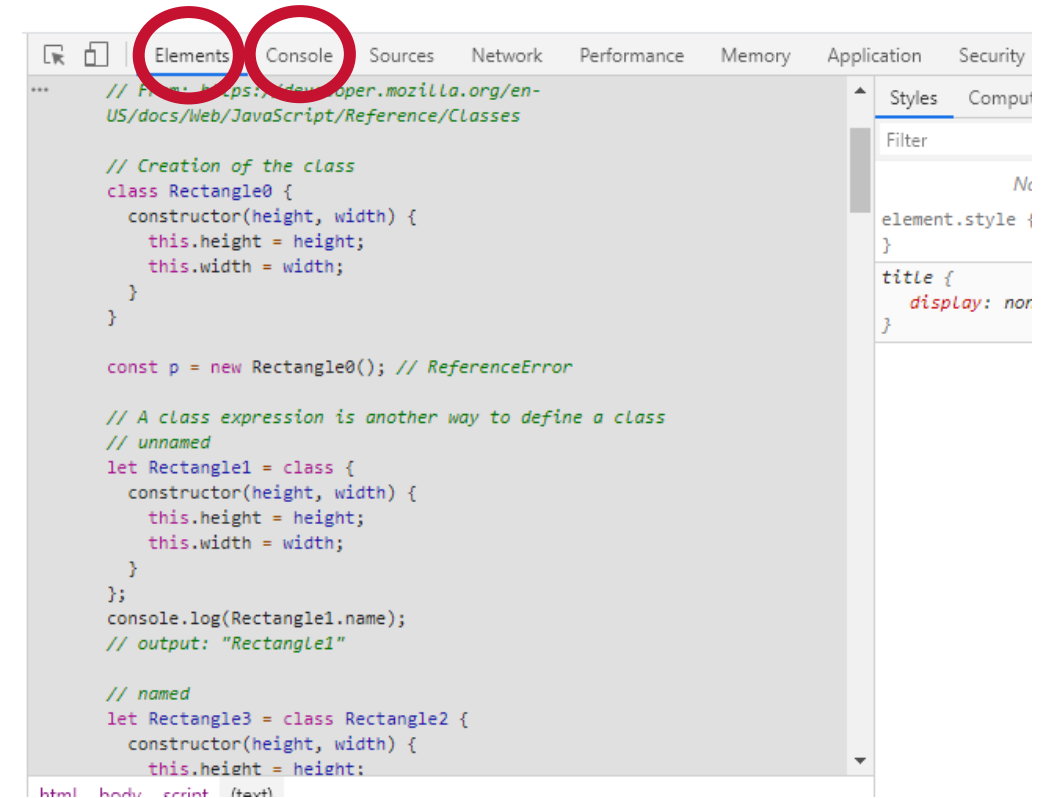
---

- More knowledge about:
  - Special characteristics of Javascript
    - **Objects**
    - **Symbols**
    - **Date and time**
  - In this class
    - Some features of Javascript
      - **NOT COMPLETE** → to read links given on Canvas
        - Our focus: Web Programming as a whole
        - You will use/need Javascript until the end of this course
          - New features/functions will be seen in the future
  - Compared to other courses
    - Several languages (HTML, CSS, Javascript, PHP,...)
      - Keep the pace as you need all the different ingredients to create/manage/understand websites
    - If it goes too fast: office hours + email for extra meetings !!!

# What to do when it does not work?

## ■ Chrome: F12

- Browse in the code in the **Elements** tab
  - HTML5 code
  - → Very useful to visualize the margin/padding
- Error message in the **Console** tab
  - Look at the error message
  - Event driven programming
    - Call of functions!
  - Be careful to the scope of the variables
  - When your script starts
    - Have you created already the HTML elements You want to reach?



# Data types

---

- Variable can be one of these 6 categories:
  - Number
  - String
  - Boolean
  - Null value
  - Undefined value
  - Object
  - Symbol
- → to think about the implicit types
  - Based on how the elements are used (combined with operators)

# Data types & conversions

- Think about how Javascript will transform the type of the variables you are manipulating

- Number to String

- num.toString();

- String to Number

- parseInt(x)
  - parseFloat(x)
  - Number(x)

```
var text = '12px';      // 10 for the base
var valinteger = parseInt(text, 10); // returns 12
```

```
var text = '3.14asmurfisdancing';
var valfloat = parseFloat(text); // returns 3.14
```

```
Number('456'); // returns 456
Number('12.3'); // returns 12.3
Number('3.14jinglebell'); // returns NaN
Number('68px'); // returns NaN
```

```
// Example 1
let x=66;
x+=' 5';
x=parseInt(x); // pick the first part that can be an Int
x=x/Math.sin(0) + ' '; // number + string = string
alert(x.charAt(0)); // x must be a string

// Example 2
MyChemicalRomance='';
for (let i=0;i<3;i++)
    MyChemicalRomance+=Math.sqrt(-1) + 'a';
alert(MyChemicalRomance);
```

# Data types & conversions

## ■ Examples

```
<script>
function myFunction() {
    var a = parseInt("10") + "<br>";
    var b = parseInt("10.00") + "<br>";
    var c = parseInt("10.33") + "<br>";
    var d = parseInt("32 42 13") + "<br>";
    var e = parseInt("    60    ") + "<br>";
    var f = parseInt("32 years") + "<br>";
    var g = parseInt("The cat is 12") + "<br>";

    var h = parseInt("10", 10) + "<br>";
    var i = parseInt("010") + "<br>";
    var j = parseInt("10", 8) + "<br>";
    var k = parseInt("0x10") + "<br>";
    var l = parseInt("10", 16) + "<br>";

    var n = a + b + c + d + e + f + g + "<br>" + h + i + j + k + l;
    document.getElementById("demo").innerHTML = n;
}
</script>
```



10  
10  
10  
32  
60  
32  
NaN  
  
10  
10  
8  
16  
16

# Objects

## ■ Declaration

- Let `x=new Object();` // name of the object
- Let `person = { name: "Jim" , age:5 , gender:"M" };`
  - Comma between the fields
- Access a field
  - `person.name`
- Add a field
  - Just name a new field `person.city="Fresno";`
- Remove a field
  - Delete `person.gender;`

## ■ Possibility of interconnections $A \leftrightarrow B$ , $B \leftrightarrow C$ , $A \leftrightarrow C$

- Be careful when you delete/modify objects

# Objects

- **Garbage collection:** automatic memory management
  - Reachability
    - If it is possible to access the data, to use it, it is kept, otherwise it is removed.
  - performed automatically, cannot force, or prevent it.
- There's a base set of inherently **reachable** values, that cannot be deleted for obvious reasons.
  - Example:
    - Local variables and parameters of the current function.
    - Variables and parameters for other functions on the current chain of nested calls.
    - Global variables.
    - ... some other, internal ones as well
  - These values are called roots !!
- Any other value is considered reachable
  - → if it's reachable from a root by a reference or by a chain of references.
  - Example
    - if there is an object in a local variable and that object has a property referencing another object,
    - that object is considered reachable! and those that it references are also reachable.



# Garbage collection

---

## ■ Objects

- Retained in memory while it is possible to reach them.
- Root to all elements
  - Single “Islands” are removed

## ■ /!\ Warning

- Being referenced is not the same as being reachable (from a root)
  - A pack of interlinked objects can become unreachable as a whole.

# Garbage collection

- Objects are **associative arrays** with several special features.
  - It stores properties (key-value pairs)
    - Property keys must be strings or symbols (usually strings).
    - Values can be of any type. /!\
- To access a property: 2 possibilities
  - To use the **dot notation**: obj.property.
  - To use **square brackets notation** obj["property"].
    - Square brackets allow to take the key from a variable, like obj[varWithKey].
- Additional operators
  - To delete a property
    - delete obj.prop.
  - To check if a property with the given key exists
    - "key" in obj.
  - To iterate over an object
    - for(let key in obj) loop.

```
// Solution 1
var myProp = 'prop';
if(myObj.hasOwnProperty(myProp)) {
    alert("yes, i have that property");
}

// Solution 2
var myProp = 'prop';
if(myProp in myObj) {
    alert("yes, i have that property");
}

// Solution 3
if('prop' in myObj) {
    alert("yes, i have that property");
}
```

# Objects

## ■ Objects

- Assigned and copied **by reference**.
- A variable stores not the “object value”, but a “reference” (address in memory) for the value.
- Copying such a variable or passing it as a function argument
  - copies that reference, not the object.
  - All operations via copied references (like adding/removing properties) are performed on the same single object.

## ■ To make a “real copy” (a clone)

- use `Object.assign` or `_.cloneDeep(obj)`.
  - `var objects = [{ 'a': 1 }, { 'b': 2 }];`
  - `var deep = _.cloneDeep(objects);`
  - `console.log(deep[0] === objects[0]);`
  - `// => false`

## ■ Object and special objects ...

- There are many other kinds of objects in JavaScript:
  - **Array** to store ordered data collections,
  - **Date** to store the information about the date and time,
  - **Error** to store the information about an error.
  - ...

## ■ “Array type” or “Date type”:

- Not types of their own
  - but belong to a single “object” data type!

## ■ Objects in JavaScript

- Powerful and useful

# Objects

## ■ Example

```
var User = {
  name: ['Javier', 'Castro'],
  age: 23,
  gender: 'male',
  interests: ['music', 'anime', 'hiking', 'Haskell'],
  bio: function() {
    let genderstr= (this.gender=='male') ? 'He' : 'She';
    let list_interests='';
    for (let i=0;i<this.interests.length-2;i++)
      list_interests+=this.interests[i] + ', ';
    list_interests+=this.interests[this.interests.length-2] + ' ,and '; // for the last one
    list_interests+=this.interests[this.interests.length-1];

    alert(this.name[0] + ' ' + this.name[1] + ' is ' + this.age + ' years old.\n' + genderstr +
      ' likes ' + list_interests + '.');
  },
  greeting: function() {
    alert('Hi! My first name is:' + this.name[0] + '.');
  }
};

document.write(User.name + '<br>');
document.writeln(User.name[0]);
document.writeln(User.age);
document.writeln(User.interests[1]);
User.interests[User.interests.length]='eating pizza';
User.bio();
User.greeting();
```

# Objects

- Notation to access elements
  - We will go back into it with **JSON**
- Access values
  - Dot notation
  - Bracket notation
- Example:

```
let person1 = {  
  name: ['Ally', 'Gator'],  
  age: 52  
};  
  
let person2 = {  
  name : {  
    first: 'Donald',  
    last: 'Duck'}  
};  
  
// Dot notation  
document.write(person1.name[0] + '<br>');  
document.write(person1.name[1] + '<br>');  
  
document.write(person2.name.first + '<br>');  
document.write(person2.name.last + '<br>');  
  
// Bracket notation  
document.write(person1['age'] + '<br>');  
document.write(person2['name']['first'] + '<br>');
```

# Object – Math (properties)

---

- E: Euler's constant and the base of natural logarithms; about 2.718.
- LN2: Natural logarithm of 2; about 0.693.
- LN10: Natural logarithm of 10; about 2.302.
- LOG2E: Base 2 logarithm of E; about 1.442.
- LOG10E: Base 10 logarithm of E; about 0.434.
- PI: Ratio of the circumference of a circle to its diameter; about 3.14159.
- SQRT1\_2: Square root of  $1/2$ ; equivalently, 1 over the square root of 2; about 0.707.
- SQRT2: Square root of 2; about 1.414.

# Object – Math (Methods)

- `abs()`: Returns the absolute value of a number.
- `acos()`: Returns the arccosine (in radians) of a number.
- `asin()`: Returns the arcsine (in radians) of a number.
- `atan()`: Returns the arctangent (in radians) of a number.
- `atan2()`: Returns the arctangent of the quotient of its arguments.
- `ceil()`: Returns the smallest integer greater than or equal to a number.
- `cos()`: Returns the cosine of a number.
- `exp()`: Returns  $E^N$ , where  $N$  is the argument, and  $E$  is Euler's constant, the base of the natural logarithm.
- `floor()`: Returns the largest integer less than or equal to a number.
- `log()`: Returns the natural logarithm (base  $E$ ) of a number.
- `max()`: Returns the largest of zero or more numbers.
- `min()`: Returns the smallest of zero or more numbers.
- `pow()`: Returns base to the exponent power, that is, base exponent.
- `random()`: Returns a pseudo-random number between 0 and 1.
- `round()`: Returns the value of a number rounded to the nearest integer.
- `sin()`: Returns the sine of a number.
- `sqrt()`: Returns the square root of a number.
- `tan()`: Returns the tangent of a number.
- `toSource()`: Returns the string "Math".

# Destructuration

---

- From array to variables

- Removing the structure

- `let v=["A", "B"];`

- `let [val1 , val2]=v;`

- `let [firstName, surname] = "Hubert Cecotti".split(' ');`

- The rest of the elements

- `let [n1, n2, ...rest] = ["I", "like", "pizza", "with mozzarella"];`

- `alert(n1); // I`

- `alert(n2); // like`

- `alert(rest[0]); // pizza`

- `alert(rest[1]); // with mozzarella`

- `alert(rest.length); // 2`



# Symbols

---

## ■ Example 1

- `// s is a new symbol`
- `let s=Symbol();`
- `let s=Symbol("MySymbol"); // description of the symbol`

## ■ Example 2

- `let i1 = Symbol("Myid");`
- `let i2 = Symbol("Myid");`
- `alert(i1 == i2); // false`

## ■ Example 3

- `let sym = Symbol.for("name");`
- `let sym2 = Symbol.for("id");`
- `// get name from symbol`
- `alert( Symbol.keyFor(sym) ); // name`
- `alert( Symbol.keyFor(sym2) ); // id`

# Symbols

- A primitive type for unique identifiers
  - created with `Symbol()` call with an optional description.
- Symbols are always different values, even if they have the same name.
  - If you want same-named symbols to be equal, then you should use the global registry
    - `Symbol.for(key)` returns (creates if needed) a global symbol with `key` as the name.
    - Multiple calls of `Symbol.for` return exactly the same symbol.
- Symbols have **2 main use cases**:
  - **“Hidden” object properties.**
    - If you want to add a property into an object that “belongs” to another script or a library,
      - You can create a symbol and use it as a property key.
      - A symbolic property does not appear in `for..in`, so it won’t be occasionally listed.
      - It won’t be accessed directly, because another script does not have our symbol, so it will not occasionally intervene into its actions.
    - So you can “covertly” hide something into objects that you need, but others should not see, using symbolic properties.
  - Many system symbols used by JavaScript
    - Accessible as `Symbol`.
    - To use them to alter some built-in behaviors.

# Date and time

---

- Creation of Objects
- Several constructors
  - `let now= new Date();`
  - `alert(now);`
- `let t=new Date(T);` with T in ms since 1/1/1970.
- `let t= new Date(0);` → 1/1/1970
- `new Date(year,month,day,hours,minutes, seconds, ms)`

# Date and time

---

## ■ Access

- `getFullYear();` // 4 digits year
- `getMonth();` // number between 0 and 11
- `getDate();` // day of the month
- `getDay();` // 0 = Sunday to 6 = Saturday
- `getHours();` `getMinutes();` `getSeconds();` `getMilliseconds();`
- `getTime();`
- `getTimezoneOffset();`

# Date and time

---

## ■ Modifications

- `setFullYear(year [, month, date])`
- `setMonth(month [, date])`
- `setDate(date)`
- `setHours(hour [, min, sec, ms])`
- `setMinutes(min [, sec, ms])`
- `setSeconds(sec [, ms])`
- `setMilliseconds(ms)`
- `setTime(milliseconds)`
  - (sets the whole date by milliseconds since 01.01.1970 UTC)

## ■ Difference of dates to obtain durations

# Date and time

---

- Date and time
  - Date object
  - Date objects always carry both date AND time.
- Months are counted from 0
  - January is a zero month, be careful for conversion!
- Days of week in `getDay()`
  - counted from 0 (that's Sunday).
- Date auto-corrects itself
  - when out-of-range components are set.
  - Convenient for adding/subtracting days/months/hours.
- Dates can be
  - subtracted,
    - difference in milliseconds., because a Date becomes the timestamp if converted to a number.
- Use `Date.now()`
  - to get the current timestamp fast.

# Autoboxing and Unboxing

---

- **Boxing:**

- It is the process of converting a **value type** to the **type object** or to any interface type implemented by this value type.

- **Autoboxing:**

- It is process whereby the JS compiler will convert primitive data types to their corresponding object wrapper classes.

- **Unboxing:**

- It happens when an instance of an **object wrapper** class is converted to its corresponding **primitive data type**.

# Autoboxing and Unboxing

---

- Example

- `const name = "Smurf"`
- `const age = 8`
- `console.log(typeof name) // string`
- `console.log(typeof age) // number`
- `console.log(name.length) // 5` How can you use length if it's a string?
- `console.log(age.toString()) // "8"` How can you use toString if it's just a number?

- Whenever we try to access a method or property on a primitive

- → the primitive is wrapped into an object



# Classes with Javascript

---

- Class declarations execute in strict mode
- See files on Canvas
  - `Class_javascript_class_01.html`

# String vs string

- Do not be confused between **string** and **String**
  - JavaScript has 2 main type categories, primitives and objects.
    - `var s = 'test'; // (typeof string)`
    - `var ss = new String('test'); // (typeof Object)`
  - Single quote/double quote patterns are identical in terms of functionality.
  - The behavior you are trying to name is called auto-boxing. So what actually happens is that a primitive is converted to its wrapper type when a method of the wrapper type is invoked. Put simple:
  - `var s = 'test';`
    - A primitive data type, no methods, nothing more than a pointer to a raw data memory reference → faster access speed.
  - `s.charAt(i) ...` because `s` is **not** an instance of `String`, JavaScript will autobox `s` (`typeof string`) to its wrapper type `String` (`typeof object`)
    - More precisely `s.valueOf(s).prototype.toString.call = [object String]`.
  - The autoboxing behavior casts `s` back and forth to its wrapper type as needed but the standard operations are incredibly fast since you are dealing with a simpler data type.

# Some words about Strings

- Strings: immutable
  - You cannot edit a character directly within the string
- Functions to get substrings, uppercase, lowercase... don't reinvent the wheel
  - Alternative solutions to CSS when you get into the DOM
- Special characters
  - \b: Backspace
  - \f Form feed
  - \n: New line
  - \r: Carriage return
  - \t: Tab
  - \uNNNN A unicode symbol with the hex code NNNN
    - \u00A9 – a unicode for the copyright symbol ©.
    - /!\ It must be exactly 4 hex digits.
  - \u{NNNNNNNN} rare characters are encoded with 2 unicode symbols
    - taking up to 4 bytes.
    - /!\ This long unicode requires braces around it.

# Closure

---

- Javascript:
  - “Function oriented language”
- Function can access a variable outside of it
- Nested functions
  - Function created inside a function
    - Useful for the creation of Objects
- Function declared in a block
  - Example: if, for, ... → they just live within this block
- Advice from lots of instructors
  - Avoid global variables 😊

# Javascript + DOM

## ■ Mouse Events

- Onclick: The event occurs when the user clicks on an element
  - Syntax: `<button onclick="myFunction()">Click me</button>`
- Oncontextmenu: The event occurs when the user right-clicks on an element to open a context menu (css 3)
- Ondbclick: The event occurs when the user double-clicks on an element
- Onmousedown: The event occurs when the user presses a mouse button over an element
- Onmouseenter: The event occurs when the pointer is moved onto an element
- Onmouseleave: The event occurs when the pointer is moved out of an element
- Onmousemove: The event occurs when the pointer is moving while it is over an element
- Onmouseover: The event occurs when the pointer is moved onto an element, or onto one of its children
- onmouseout The event occurs when a user moves the mouse pointer out of an element, or out of one of its children (2)
- onmouseup The event occurs when a user releases a mouse button over an element

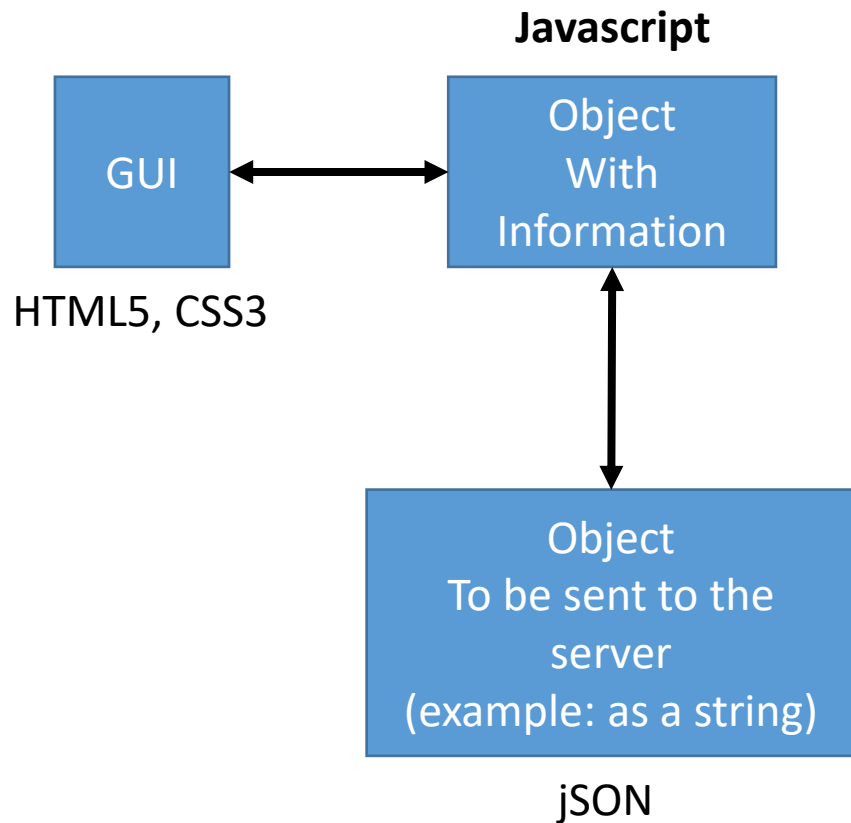
## ■ Keyboard Events

- Onkeydown: The event occurs when the user is pressing a key
- Onkeypress: The event occurs when the user presses a key
- Onkeyup: The event occurs when the user releases a key

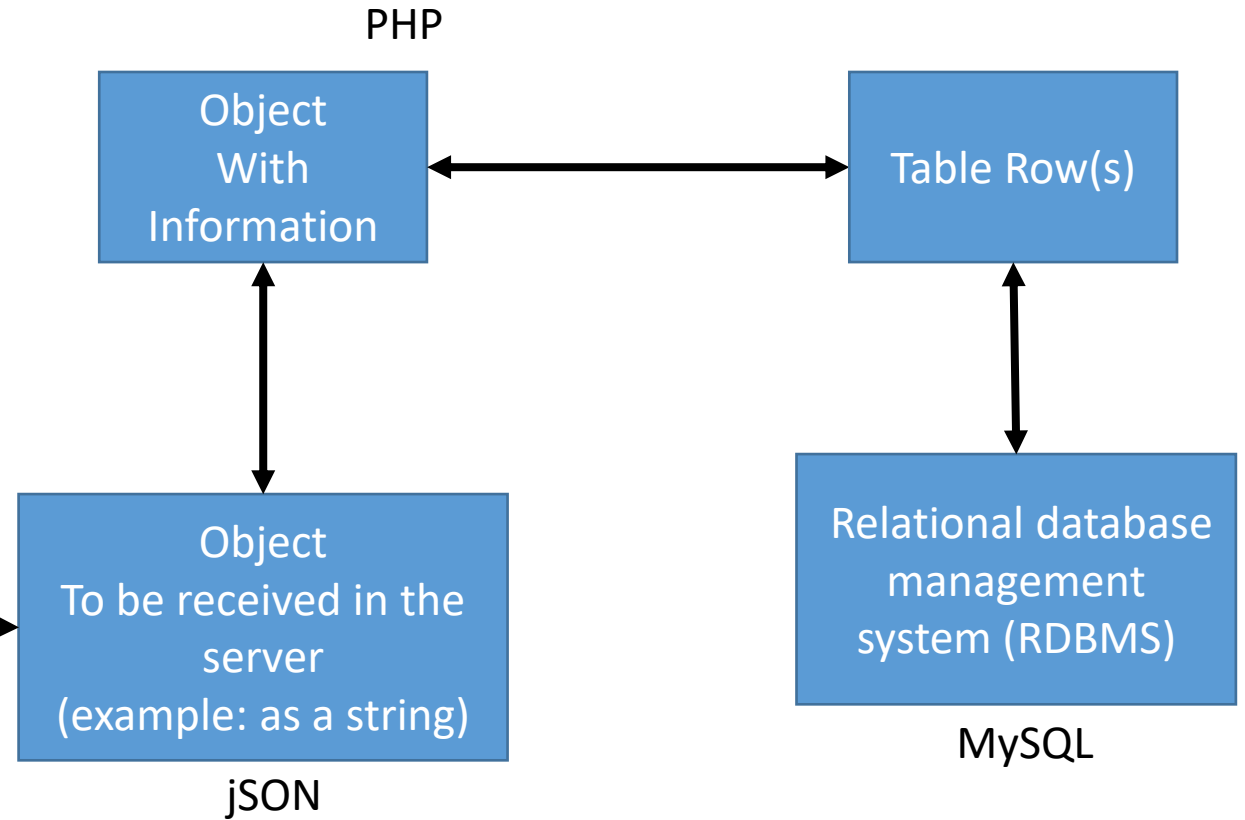
## ■ + Frame/Object/Form/Clipboard (copy/paste/cut)/Media events ...

# CSci 130: the main story

## ■ Client



## Server



# Conclusion

---

## ➤ Javascript

- Very important to **practice** (debugging is not simple)
  - To work on personal projects
  - To aim at well done work with rich functionalities
  - To go beyond what other website exist
- Next session
  - The canvas for drawing elements
  - More events (mouse) to interact with graphical elements

## ➤ Reading

- [https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object-oriented\\_JS](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object-oriented_JS)

## ➤ Lab session

- Calculator (group project)