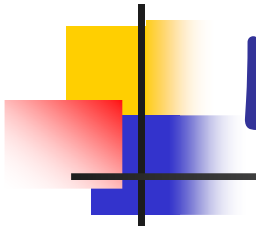
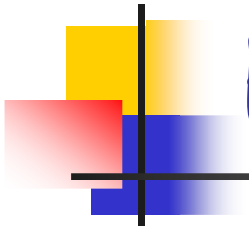


Lecture-3

- Javascript
  - Events
  - OOP concepts of JS

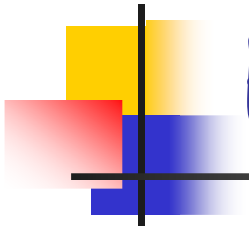


Events – onerror

- Recall our example of error handling

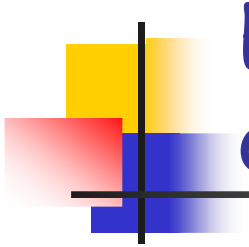
Example:

```
onerror=handleErr // Call handleErr on errors

function handleErr(msg,url,l)
// msg – error msg, url – current URL, l – line #
{
   //Handle the error here
   return true or false
}
```



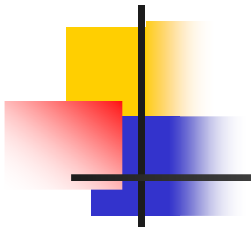
Events – onsubmit, onreset

- onsubmit – Event when a submit button is pressed.
  - Recollect our bank form example.
- onreset – event when a reset button is pressed.
  - Typically, used to cancel/reset the values of all fields.



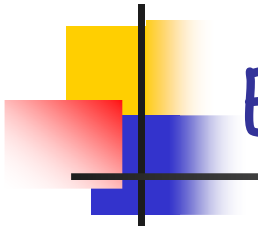
Events – onmouseup, onmousedown, onmouseover, onmouseout

- onmousedown – event when a mouse button is pressed down.
- onmouseup – event when a mouse button is pressed up.
- onmouseover – event when a mouse hovers over (a specific region).
- onmouseout – event when a mouse comes out (of a specific region).



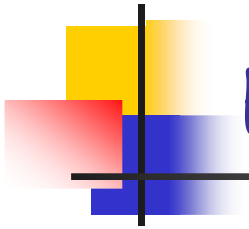
Events – onkeypress, onkeydown, onkeyup

- onkeypress – Event when a key is pressed
- onkeydown – Event when a key is pressed or held down
  - Similar to onkeypress
- onkeyup – Event when a key is released (after being pressed)



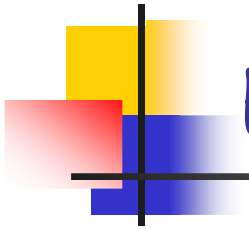
Events – onclick, ondblclick, onchange

- onclick – event when a button is clicked
- ondblclick – event when a button is double clicked
  - Try to avoid onclick when ondblclick is defined



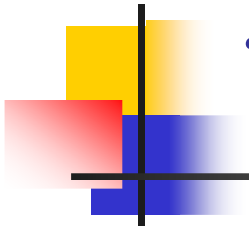
Events – onfocus, onblur, onresize

- onfocus – event when an element gets focus
- onblur – event when an element loses focus
  - Opposite of onfocus



Events - onresize, onchange, onabort

- onresize – event when a browser window is resized (changed).
- onchange – event when the value of a field changes
- onabort – event when loading of an image is interrupted.

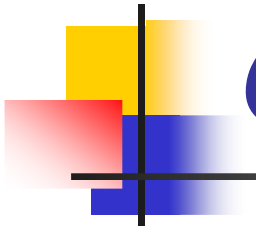


Timing

- setTimeout – similar to "sleep( )"
  - BUT statement following "setTimeout" is executed without any delay.
  - Example

```
t = setTimeout("fn1( )", n);
fn2( );
```

    - fn1( ) is delayed by n millisecs.
    - fn2( ) is NOT, it is executed immediately.
- clearTimeout – stops the timer
  - Opposite of "setTimeout( ).



OOP features

- Main OOP concepts
  - Treat real world entities as "objects"
  - Has data and methods
- Importance features of OOP
  - Data encapsulation
  - Inheritance
  - Polymorphism
- JS supports these OOP features
  - But note: JS is a weakly typed language.
  - Implementation of these features
    - Slightly different from strongly typed languages like C++ and JAVA

Creating JS objects

- Create an instance of an object directly

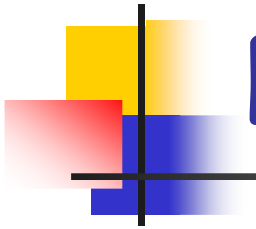
```
p1=new Object( );        // Create an object directly using new
p1.firstname="John";     // Set data variables
p1.lastname="Doe";
p1.age=50;
p1.eyecolor="blue";
p1.incrementAge = changeAge; // Set method
p1.incrementAge( );          // Call method

function changeAge( )        // Function definition
 {
    this.age++;
 }
```

Note: There is NO class keyword, as in C++, JAVA

Creating JS objects … cond.

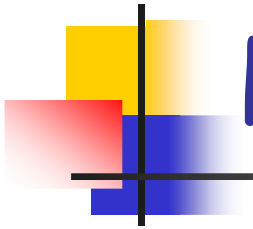
- Crate using a template – use function

```
// Template (class) definition
function person (firstname, lastname, age, eyecolor)
  {
    this.firstname = firstname;
    this.lastname = lastname;
    this.age = age;
    this.eyecolor = eyecolor;
    this.incrementAge = changeAge; // Define a member function
  }

// Function definition
function changeAge( )
  {
    this.age++;
  }

// Creating a new object of person
p1 = new person ("David", "Miller", 50, "brown");
```



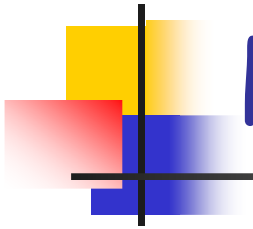
Data encapsulation

- C++, JAVA
  - public, private, protected
- JS
  - public – accessible to class/external members
  - private – accessible to private/privileged members
  - Privileged methods
    - Can access private functions
    - Can access and change private data
    - External methods can access private members of class
    - Something like public access functions of C++, JAVA
  - NO protected data/methods



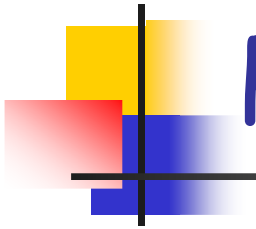
Public members

```
// Public data member definition
function public_Fn_Eg (…)
 {
  this. publicMember = <value>;
 }

// Public function definition
public_Fn_Eg.prototype.pubFn = function (<params>)
  {
      // code
  }
```



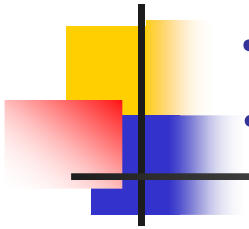
Private members

```
function private_Fn_Eg (…)
{
    // private data members
    var privateMember = <value>;

    //private functions
    function privateFunction_1 (<params>)
     {
         // code
     }

    var privateFunction_2 = function(<params>)
     {
         // code
     }
}
```



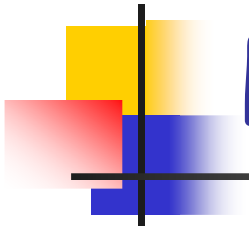
Privileged functions

```
function privileged_fn_Eg
 {
    this.privilegedFn = function(…)
      {
        // CAN access private functions
        // CAN access/change private data
      }
 }
```



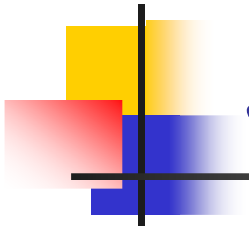
Inheritance

- Define parent and child template functions as before.
- To define the inheritance, use
  - child.prototype = new parent;
- Children do NOT have access to parent's private members.



Polymorphism

- Inherently supported in Javascript
- Any object calls member function in the most specific template class.
- Child objects call member functions
  - From the child class if defined in child objects.
  - From the parent class, otherwise.
- Parent objects calls the function from the parent template class.



Static members

- Static member
  - Specific to the class
    - NOT object specific