# CS 416
# Web Programming

## Javascript Functions
## Conditionals and repetition

Dr. Williams
Central Connecticut State University

## Agenda

- Functions
- Conditional execution
- Conditional repetition

## Dynamic images

- just as you can use user-initiated events to change the contents of
- text areas and text boxes, you can also dynamically modify images

```
<img id="faceImg" src="happy.gif" alt="Happy Face" />
```

causes the image stored in the file `happy.gif` to appear in the page

- you can change the image by reassigning its SRC attribute
  – similar to the way that text boxes/areas have their VALUE attribute reassigned

```
document.getElementById('faceImg').src = "sad.gif";
```

replaces `happy.gif` with `sad.gif`

## Accessing text fields

- recall that values entered via text boxes/areas are always returned as strings

```
if (document.getElementById('age').value >= 18) {
    alert("You are old enough to vote.");
}
else {
    alert("Sorry. You are too young to vote.");
}
```

will say that a 2-year old can vote, but a 102-year old can't!

WHY?

if you wish to treat a value obtained from a text box or text area as a number, you must use the `parseFloat` function to convert it

```
age = parseFloat(document.getElementById('age').value);
if (age >= 18) {
    alert("You are old enough to vote.");
}
else {
    alert("Sorry. You are too young to vote.");
}
```

will behave as expected

## Simplifying with functions

- Consider:

```
<input type="button" value="Click for Greeting"
       onclick="firstName = document.getElementById('firstNameBox').value;
                lastName = document.getElementById('lastNameBox').value;
                message = 'Hello ' + firstName + ' ' + lastName +
                          ', or may I just call you ' + firstName +
                          '? You wouldn\'t be related to the ' + lastName +
                          's of Park Avenue, would you?';
                document.getElementById('messageArea').value = message;" />
```

- functions provide a mechanism for simplifying complex functionality such as this
- functions minimize the amount of detail that has to be considered
  - e.g., can use Math.sqrt without worrying about how it works
- functions reduce the length and complexity of code
  - e.g., a single call to Math.sqrt replaces the underlying complex algorithm

## Calling functions

- `<input type="button" value="Calculate differences" onclick="CalculateDiff();" />`

- Handled click event to call user-defined function we define in the HEAD of the page

- To define a function in the HEAD you use

```
<script type="text/javascript"> </script>
```

## User defined functions

- Define new functions in the HEAD section and call them within the page

```
function FUNCTION_NAME()
{
  STATEMENTS TO BE EXECUTED;
}
```

  - a function definition begins with the word `function` followed by its name and `()`
    - a function name should be descriptive of the task being performed
  - Same naming standards that apply to variable naming apply to function naming
  - the statements to be executed when the function is called are placed between the curly braces

## User defined functions cont.

```
function functionName(param1, param2, param3) {
    //code to be executed

    return x;
}
```

- **Do not specify data types for parameters**
- **No type checking on the passed arguments**
- **No check on the number of arguments received**
- **Do not specify in signature if there is a return**

**All legal/no compile error (too few, too many, different types):**

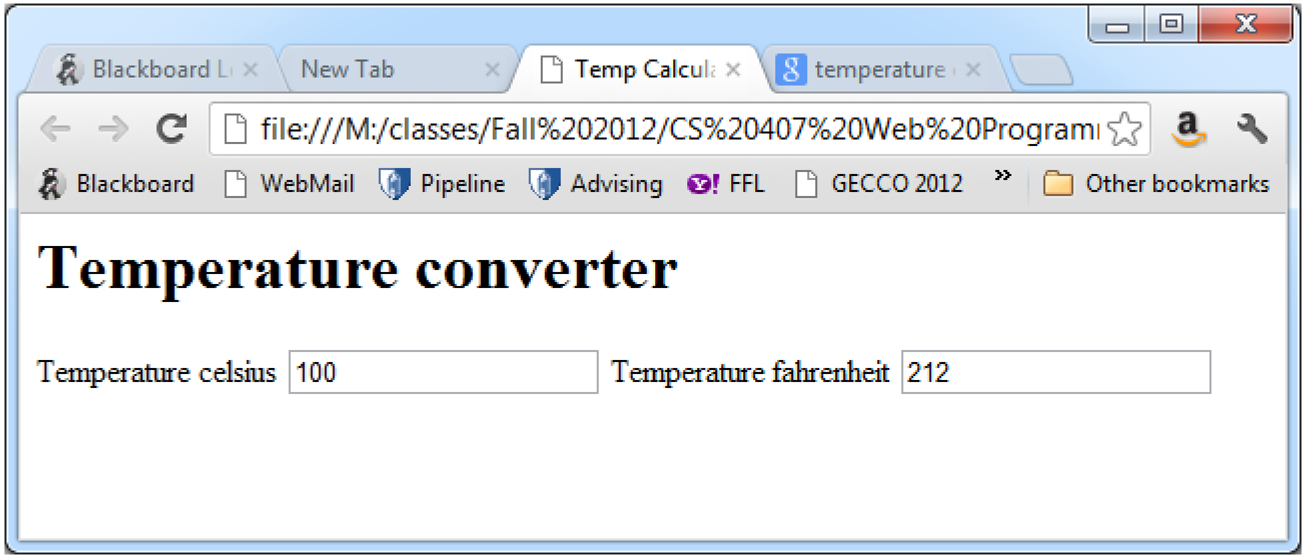
```
functionName(1)  or functionName("a",2,true,"z",4,5)
```

**Can check if argument not passed by testing if parameter is undefined:**

```
if (param2 == undefined){
```

# Your turn

- Write the HTML for the 2 input elements with code to call your function to do the conversion whenever a key is pressed in the Temp Celsius box
- Write a function that reads the value in the first input box and writes out a new value that is that number multiplied by 1.8 + 32



Temperature converter

Temperature celsius 100 Temperature fahrenheit 212

## Condition execution

- Unconditional execution is when the same code is run regardless of any other factors
- Conditional execution allows you to control whether you want some piece of code to run based some test being satisfied
- *Conditional execution* refers to a program’s ability to execute a statement or sequence of statements only if some condition holds true (If statement, while statement)

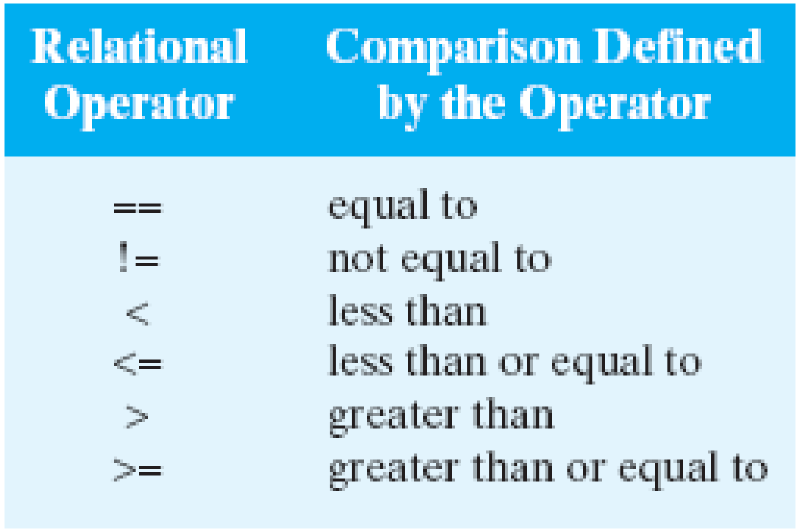
## Example

- The general form of an if statement in Javascript is:

```
if (BOOLEAN_TEST){
     Statements executed if true
}else if (BOOLEAN_TEST){
     Statements executed if first test is false
        and this test is true
}else{
      Statements executed if no other
    conditions were true
}
```

## Boolean tests

- the test that controls an if statement can be any *boolean expression* (i.e., an expression that evaluates to either `true` or `false`)
  - boolean tests are formed using *relational operators* because they test the relationships between values



| Relational Operator | Comparison Defined by the Operator |
|---|---|
| == | equal to |
| != | not equal to |
| < | less than |
| <= | less than or equal to |
| > | greater than |
| >= | greater than or equal to |

*NOTE:*

*== is for comparisons*

*= is for assignments*

## Logical connectives

- sometimes, simple comparisons between two values may not be adequate to express the conditions under which code should execute
- JavaScript provides operators for expressing multipart tests
  - *logical AND* (&&): represents the conjunction of two things
    - (TEST1 && TEST2) is true if both TEST1 and TEST2 are true

```
if (roll1 == 4 && roll2 == 4) {
    // code to be executed when double fours are rolled
}
```

  - *logical OR* (||): represents the disjunction of two things
    - (TEST1 || TEST2) is true if either TEST1 or TEST2 are true

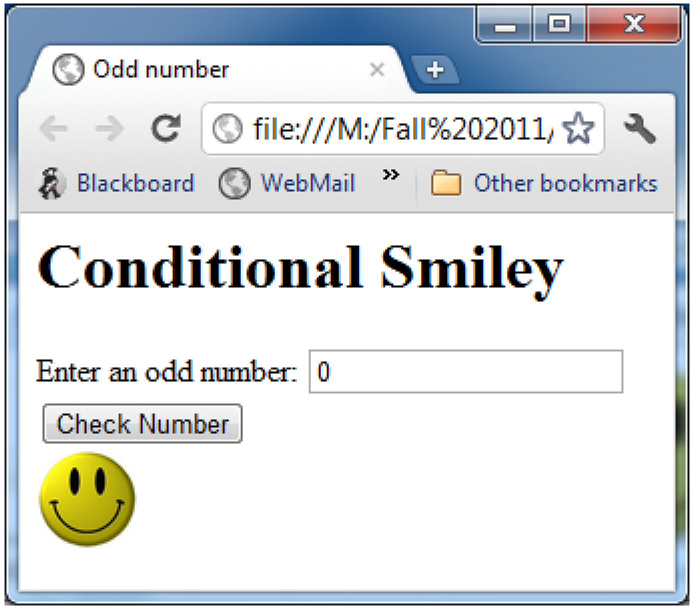
```
if (roll1 == 4 || roll2 == 4) {
    // code to be executed when at least one four is rolled
}
```

  - *logical NOT* (!): represents negation
    - (!TEST1) is true only if TEST1 is false

```
if (!(roll1 == 4 || roll2 == 4)) {
    // code to be executed when neither roll is a four
}
```

# Your turn
# Conditional and images

- Your page should take a number if it is odd show the smile smile.jpg if it is even it should show sad.jpg
- Remember to test if a number is even/odd you can use the modulus operator (“%”) which will return the remainder of the first number divided by the second Ex.
  - 7%2 = 1
  - 6%2 = 0
  - 23%7 = 2



Conditional Smiley

Enter an odd number: 0

Check Number

## Conditional repetition

- an if statement is known as a *control statement*
    - *either do this or don't, based on some condition* (if)
    - *either do this or do that, based on some condition* (if-else)
- closely related to the concept of conditional execution is *conditional repetition*
    - many problems involve repeating some task over and over until a specific condition is met
    - e.g., rolling dice until a 7 is obtained
    - e.g., repeatedly prompting the user for a valid input
    - in JavaScript, *while loops* provide for conditional repetition

## While loops

- a *while loop* resembles an if statement in that its behavior is dependent on a boolean condition.
  - – however, the statements inside a while loop’s curly braces (a.k.a. the *loop body*) are executed *repeatedly* as long as the condition remains true
  - – general form:

```
while (BOOLEAN_TEST) {
    STATEMENTS_EXECUTED_AS_LONG_AS_TRUE
}
```

## While loop cont.

when the browser encounters a while loop, it first evaluates the boolean test

- if the test succeeds, then the statements inside the loop are executed in order, *just like an if statement*
- once all the statements have been executed, program control returns to the beginning of the loop
- the loop test is evaluated again, and if it succeeds, the loop body statements are executed *again*
- this process repeats until the boolean test fails

## While loop example

- example: roll two dice repeatedly until doubles are obtained

```
roll1 = RandomInt(1, 6);                // SIMULATE THE DICE ROLLS,
roll2 = RandomInt(1, 6);                // STORE IN VARIABLES, AND DISPLAY
document.write("You rolled: " + roll1 + " " +roll2 + "<br />");

while (roll1 != roll2) {                // AS LONG AS YOU DON'T HAVE DOUBLES,
    roll1 = RandomInt(1, 6);            // ROLL AGAIN AND DISPLAY THE ROLLS
    roll2 = RandomInt(1, 6);
    document.write("You rolled: " + roll1 + " " +roll2 + "<br />");
}
document.write("DOUBLES!");             // DISPLAY THE FACT THAT YOU HAVE DOUBLES
```

sample output:

```
You rolled: 6 4
You rolled: 4 1
You rolled: 2 4
You rolled: 6 3
You rolled: 5 2
You rolled: 1 5
You rolled: 5 2
You rolled: 6 5
You rolled: 1 2
You rolled: 5 5
DOUBLES!
```

note: even though while loops and if statements look similar, they are very different control statements

- an *if statement* may execute its code 1 time or not at all
- a *while loop* may execute its code an arbitrary number of times (including not at all)

## For loops

- For loop syntax is identical  to Java syntax

```
for (i=0;i<10;i++){
  //some statements
}
```

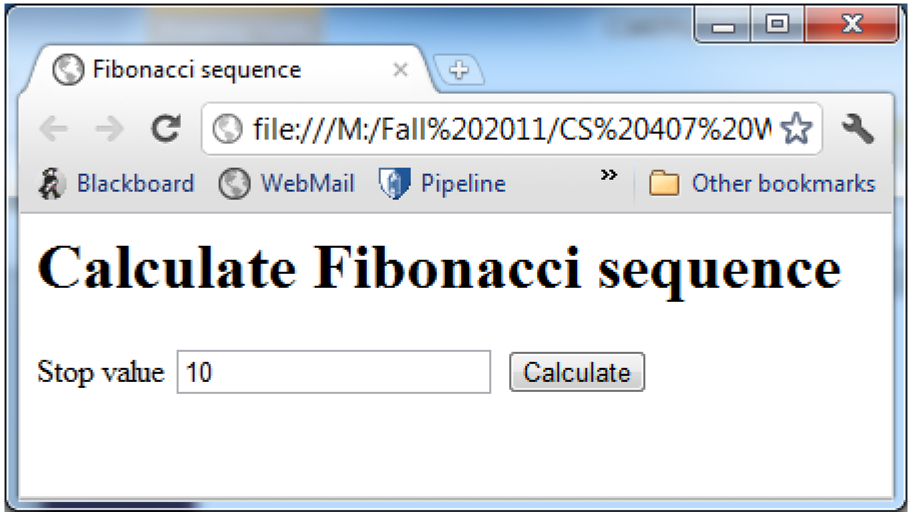
- There is also the concept of looping through elements in arrays (which we’ll cover next lecture)

```
var person=["John", "Doe", 25};

for (x in person)
  {
  txt=txt + person[x];
  }
```

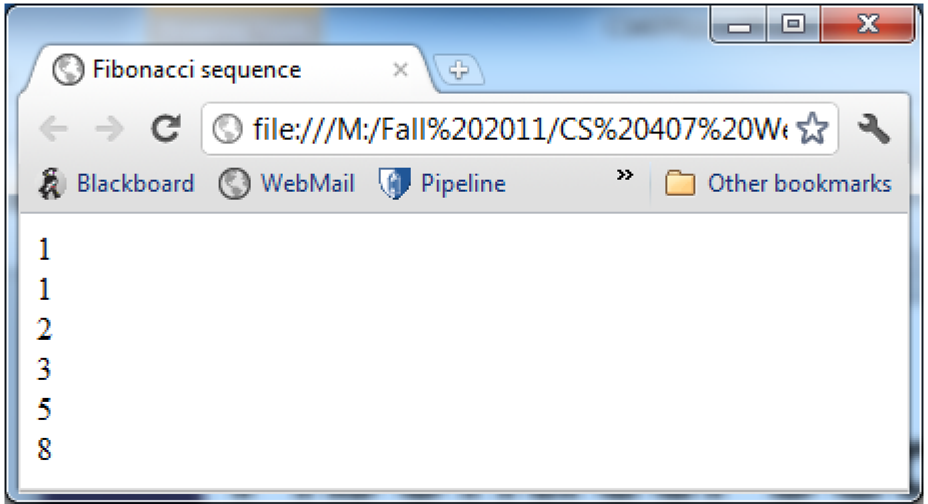
## Your turn
## Fibonacci sequence

- Prompt for stop value
- Calculate the Fibonacci sequence
- (1,1,2,3,5,8,13,...)
- Stop when the sum of the sequence is greater than or equal the stop value



Calculate Fibonacci sequence

Stop value 10

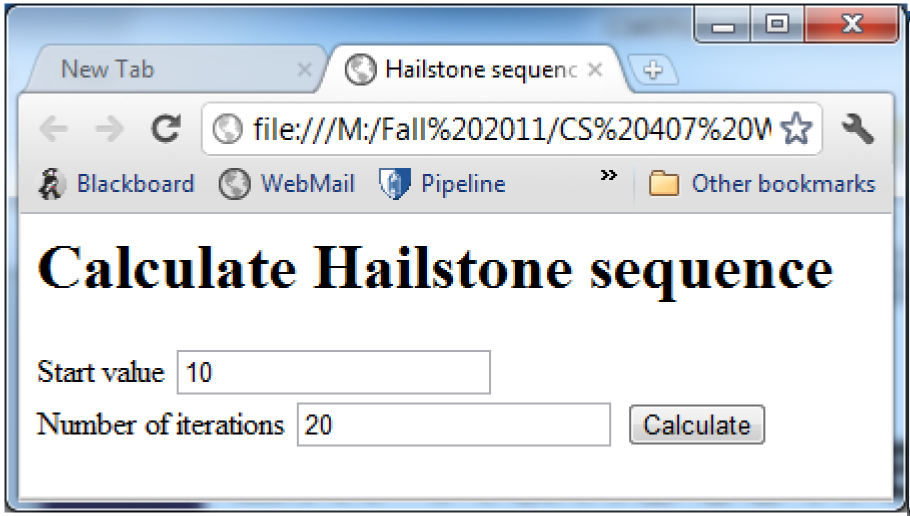


```
1
1
2
3
5
8
```

# Your turn
# Hailstone sequence

Interesting fact: It hasn't been mathematically proven that all starting values converge to 4,2,1

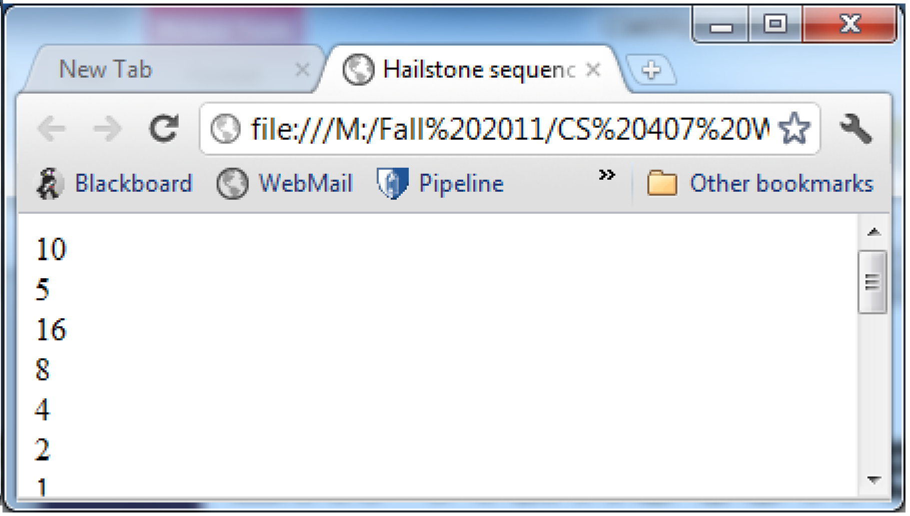
- Hailstone sequence
  1. start with any positive integer
  2. if the number is odd, then multiply the number by three and add one; otherwise, divide it by two
  3. repeat as many times as desired
  - for example: 5, 16, 8, 4, 2, 1, 4, 2, 1, 4, 2, 1, …
  - To test if a number is even use modulus operator "%" which returns the remainder of the first number divided by the second number so:
  - `(a%2 == 0)` means *a* is even, `(a%2 != 0 )` means *a* is odd



**Calculate Hailstone sequence**

Start value 10

Number of iterations 20 Calculate



```
10
5
16
8
4
2
1
```