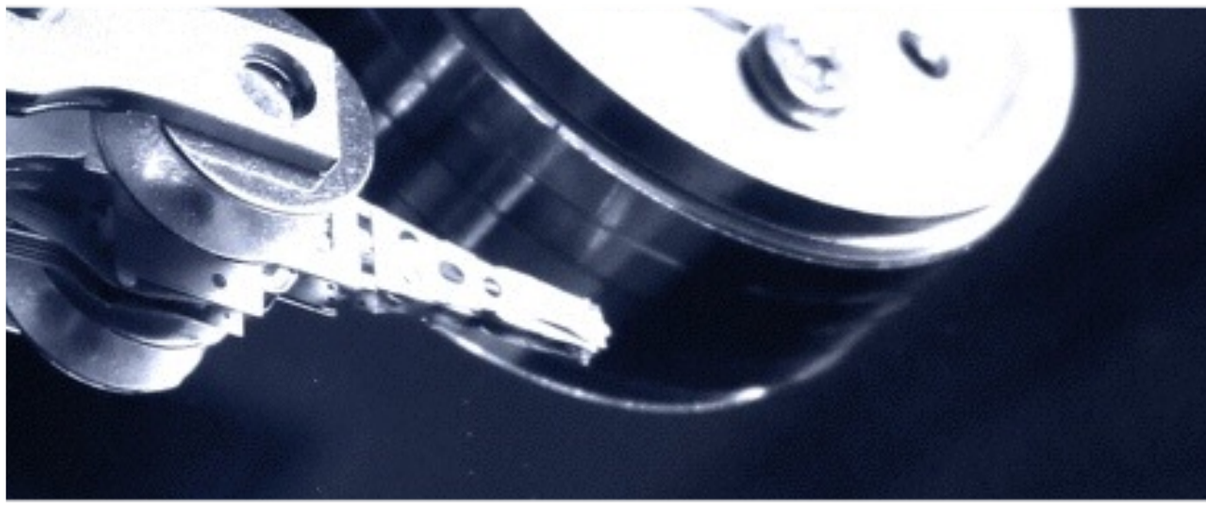
FULL SAIL
UNIVERSITY

web design and development



programming for web applications 1

courseMaterial.5

courseDirector

Fialishia O'Loughlin

foloughlin@fullsail.com

labSpecialist

Eric Silvay

esilvay@fullsail.com

courseMaterial.**5**

goal4.**Recap**

goal4.**Recap**

- review last assignment if already completed
- intro to objects
- intro to the Document Object Model (DOM)
- events and handlers

courseMaterial.**Objective**

- course material
  - *regular expression*
  - *math methods*
  - *date methods*
  - *practice all the new materials*
- assignment
  - *fine tune the concepts from the course materials*

courseMaterial.**5**

regular.**Expression**

regular.Expressions

- what is RegEx?
  - a regular expression is an object that describes a pattern of characters
  - regular expressions is a unique method of ***string*** testing, which checks a string against a **pattern** and "search-and-replace" functions on text
  - ***be warned:*** regular expressions are not for the faint of heart - luckily, once we develop a few, we shouldn't need to remake it
  - here's an example of a complex pattern we would could use for email addresses:

```
var email = /^[\w\.\-]+@([\w\-]+\.)+[a-zA-Z]+$/ ;
```

  - look scary? HA! don't worry, it will make sense soon

regular.Expressions

- RegEx setup
  - we can create our regular expressions using a literal syntax - we use forward-slashes (/) just like quotes around a string
  - or, we can use the **new RegExp( )** constructor by passing a string with the pattern inside.

```
var email = /pattern/ ;                 //a literal
syntax

//using a function
var email = new RegExp("pattern");
```

  - the benefit of the function is that we can inject variables into the pattern, but the literal syntax we cannot

regular.Expressions

- RegEx setup
  - a regular expression *pattern* uses a special set of character rules that uses both ordinary character sets and specialized sets - if we simply wanted to see if a string contained the word “javascript”, the pattern would be:

```
var js = /javascript/;
```

  - any letters and numbers (without special sets around them) are interpreted literally, such as the above example
  - any character that is not a letter or number should be escape with a backslash (\ ) - if you want to use it literally. *note: letters with backslashes have special meaning*

regular.Expressions

- RegExp special rules

character	description
. (dot)	this is a wildcard character - it will match anything except for line breaks
*** (asterisk)**	when an asterisk precedes a character, it must match it 0 or more times
+ (plus)	when a plus precedes a character, it must match it 1 or more times
? (question)	this makes the preceding character optional *(0 or 1 matches only)*
^ (caret)	this matches the *following character* at the start position of a string
$ (dollar)	this matches the *preceding character* at the end position of the string
| (pipe)	this will match the pattern either on the left **or** the right of the pipe

regular.Expressions

- RegExp examples
  - **/^javascript/** matches “javascript rules”, but not “i love javascript”
  - **/javascript$/** matches “i love javascript” , but not “javascript rules”
  - **/^javascript$/** matches only “javascript” and nothing else.
  - **/yea+h/** matches “yeah”, “yeaaaah”.. but not “yeh” it would need **/yea*h/**
  - **/yea?h/** matches “yeah” and “yeh”.. but not “yeaaaah”
  - **/javascript|JavaScript/** matches “javascript” **or** “JavaScript”

regular.Expressions

- RegExp special rules

expression	description
(..)	▸ round brackets define a group of characters that must occur together ▸ after the closing bracket, you can then apply modifiers such as * + or **?**
[..]	square brackets define a **character class** - a character class matches any ***one*** character inside the brackets - most common to use are:
[aqz]	match one occurrence of “a”, “q”, or “z”, the same as **(a|q|z)**
[a-z]	would match any lower case letter
[A-Z]	would match any upper case letter
[a-zA-Z]	would match any letter
[^..]	any one character not between the brackets - **[^a-zA-Z]** would match any non-letter

regular.Expressions

- RegExp examples
  - **/[Jj]ava[Ss]cript/** - matches “Javascript”, “JavaScript”, “javascript”, or “javaScript”
  - **/^(Java)?Script$/** - matches “JavaScript” or “Script”.. but not “JavaJavaScript”
  - **/^[a-zA-Z\^\-\.]+$/** - matches 1 or more of only letters
  - Combining character sets can create sequences of matches:
    - **/^[a-zA-Z]+[0-9]$/** - matches 1 or more letters at the beginning, and 1 number at the end - would match “mike1”, but not “mike11” or “11mike”
  - most often, validation sets consist of multiple classes like the above

regular.**Expressions**

- RegExp methods

method	description
exec()	RegExp.**exec(**string**)**: apply RegExp to the given string, and returns the matched information
test()	RegExp.**test(**string**)**: tests for a match in its string parameter - returns a boolean
match()	string.**match(**RegExp**)**: match given string with the RegEx
search()	string.**search.(**RegExp**)**: matches RegExp with string and returns the index of the beginning of the match if found, -1 if not
replace()	string.**replace.(**RegExp**)**: matches the given string, and returns the edited string
split()	string.**split.(**RegExp**)**: cuts a string into an array, making cuts at matches

regular.Expressions

- RegExp examples

```
var emailRegEx = /(\w[-._\w]*\w@\w[-._\w]*\w\.\w{2,3})/;
var str = "personal email is jc@google.com work email is
jc@fullsail.com";
```

```
emailRegEx.test(str)        //returns “true”
emailRegEx.search(str)      //returns “true”
str.replace(emailRegEx,"XXX@XXX.com"); //replaces the 1st email
```

```
str.match( emailRegEx );                    //returns array or null
```

The common methods for validation are **test** and **match**

regular.Expressions

- RegExp metaCharacters

metaChar	description
\s	matches any whitespace character, similar to **[ ]**
\S	matches any non-whitespace, similar to **[^]**
\d	matches any digit, similar to **[0-9]**
\D	matches any non-digit, similar to **[^0-9]**
\w	matches any “word” letter, similar to **[a-zA-Z0-9_]**
\W	match any non-”word” letter, similar to **[^a-zA-Z0-9_]**

regular.Expressions

- RegExp variations

items	description
{n, m}	matches at least *n*, but no more than *m*
{n, }	matches at least *n*
{n}	matches exactly *n*

- **/^\d{5}$/ -** could match a 5 digit zipcode (and only 5 digits)
- **/^[a-zA-Z]{2,3}$/ -** would match 2 to 3 letters only

regular.Expressions

- RegExp - live code
  - time to build some common patterns that we could use in our form validator
  - a basic name field *(contain only letters)*

```
/^[a-zA-Z]+$/
```

  - an email field *(someone@domain.com)*

```
/^[\w\.\-]+\@([\w\-\.]+\.)+[a-zA-Z]{2,4}$/
```

```
/^   [\w\.\-]+   \@   ( [\w\-\.]+ \. )+    [a-zA-Z]{2,4}    $/
```

regular.Expressions

- RegExp - resources
  - an excellent resource for finding Regular Expressions is at:
    - http://regexlib.com/
    - as you browse their expression lists, keep in mind that these are general expression syntax, you still have to put in a JavaScript format, such as the RegEx literal:

```
var pattern = / pattern_expression_here /;
```

  - for testing expressions: http://tools.netshiftmedia.com/regexlibrary/

courseMaterial.**5**

method.**Math**

method.**Math**

- what are math methods?
  - the math object allows you to perform mathematical tasks
  - popular math methods:
    - floor( )
    - max( )
    - min( )
    - random( )
    - round( )

method.**Math**

method	description
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 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.

method.**Math**

method	description
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".

method.**Math**

- math method examples

```
document.getElementById("demo").innerHTML=Math.random();
document.getElementById("demo").innerHTML=Math.max(5,10);
//returns 10

document.getElementById("demo").innerHTML=Math.round(2.5);
//returns 3
```

courseMaterial.5

method.**Date**

method.Date

- what are Date methods?
  - the date object is used to work with dates and times
  - date objects can be created with a “new Date( )”
  - popular date methods:
    - date( )
    - getFullYear( )
    - setFullYear( )
    - getTime( )
    - getDay( )

method.Date

method	description
Date()	Returns today's date and time
getDate()	Returns the day of the month for the specified date according to local time.
getDay()	Returns the day of the week for the specified date according to local time.
getFullYear()	Returns the year of the specified date according to local time.
getHours()	Returns the hour in the specified date according to local time.
getMilliseconds()	Returns the milliseconds in the specified date according to local time.
getMinutes()	Returns the minutes in the specified date according to local time.
getMonth()	Returns the month in the specified date according to local time.
getSeconds()	Returns the seconds in the specified date according to local time.

method.Date

method	description
getTime()	Returns the numeric value of the specified date as the number of milliseconds since January 1, 1970, 00:00:00 UTC.
getTimezoneOffset()	Returns the time-zone offset in minutes for the current locale.
getUTCDate()	Returns the day (date) of the month in the specified date according to universal time.
getUTCDay()	Returns the day of the week in the specified date according to universal time.
getUTCFullYear()	Returns the year in the specified date according to universal time.
getUTCHours()	Returns the hours in the specified date according to universal time.
getUTCMilliseconds()	Returns the milliseconds in the specified date according to universal time.

method.Date

method	description
getUTCMinutes()	Returns the minutes in the specified date according to universal time.
getUTCMonth()	Returns the month in the specified date according to universal time.
getUTCSeconds()	Returns the seconds in the specified date according to universal time.
getYear()	**Deprecated** - Returns the year in the specified date according to local time. Use getFullYear instead.
setDate()	Sets the day of the month for a specified date according to local time.
setFullYear()	Sets the full year for a specified date according to local time.
setHours()	Sets the hours for a specified date according to local time.

method.Date

method	description
setMilliseconds()	Sets the milliseconds for a specified date according to local time.
setMinutes()	Sets the minutes for a specified date according to local time.
setMonth()	Sets the month for a specified date according to local time.
setSeconds()	Sets the seconds for a specified date according to local time.
setTime()	Sets the Date object to the time represented by a number of milliseconds since January 1, 1970, 00:00:00 UTC.
setUTCDate()	Sets the day of the month for a specified date according to universal time.
setUTCFullYear()	Sets the full year for a specified date according to universal time.
setUTCHours()	Sets the hour for a specified date according to universal time.
setUTCMilliseconds()	Sets the milliseconds for a specified date according to universal time.

method.Date

method	description
setUTCMinutes()	Sets the minutes for a specified date according to universal time.
setUTCMonth()	Sets the month for a specified date according to universal time.
setUTCSeconds()	Sets the seconds for a specified date according to universal time.
setYear()	**Deprecated -** Sets the year for a specified date according to local time. Use setFullYear instead.
toDateString()	Returns the "date" portion of the Date as a human-readable string.
toGMTString()	**Deprecated -** Converts a date to a string, using the Internet GMT conventions. Use toUTCString instead.
toLocaleDateString()	Returns the "date" portion of the Date as a string, using the current locale's conventions.

method.Date

method	description
toLocaleFormat()	Converts a date to a string, using a format string.
toLocaleString()	Converts a date to a string, using the current locale's conventions.
toLocaleTimeString()	Returns the "time" portion of the Date as a string, using the current locale's conventions.
toSource()	Returns a string representing the source for an equivalent Date object; you can use this value to create a new object.
toString()	Returns a string representing the specified Date object.
toTimeString()	Returns the "time" portion of the Date as a human-readable string.
toUTCString	Converts a date to a string, using the universal time convention.
valueOf()	Returns the primitive value of a Date object.

method.**Date**: static methods

method	description
Date.parse()	Parses a string representation of a date and time and returns the internal millisecond representation of that date.
Date.UTC()	Returns the millisecond representation of the specified UTC date and time.

method.Date

- date method examples

```
var d = new Date()
console.log(d.getFullYear());       //gets the date's year

var d = new Date()
//the months go from 0 to 11 (i.e January is at 0 index)å
console.log(d.setFullYear(2021,0,01));
document.getElementById("tagbox").innerHTML=d;
```

Assignment / Goal 5

- Goal5: Assignment: Form Validation
  - Log into FSO. This is where all your assignment files will be located as well as Rubrics and assignment instructions.
- Commit your completed work into GitHub
  - As part of your grade you will need at least 6 reasonable GIT commits while working on the assignment.
- In FSO there is an announcement with “Course Schedule & Details” in the title, in that announcement you will see a “Schedule” link which has the due dates for assignments.