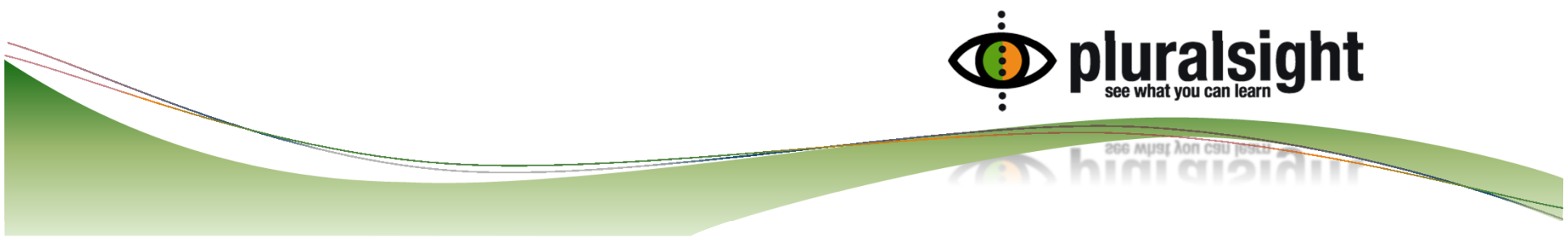
Built-in Types and Libraries

Liam McLennan
www.pluralsight.com



Outline

- **String**
- **Number**
- **Array**
  - Underscore.js
- **Regular expression**
- **Date**
  - Datejs
- **Library functions**
- **Math object**

String

- Primitive type representing an ordered set of characters
- Created using one of two literal notations

```
var string1 = "The quick brown fox's jump";

var string2 = '"The quick brown fox"';
```

- No multiline string syntax

String Escape Sequences

- Common escape sequences begin with \
- New line \n
- String literal delimiter \" or \'
- Unicode symbols \u[code]

EXAMP
LE

String Concatenation

- Join strings with the + operator
- No string interpolation

```
var string1 = "The quick " + "brown" + " fox";
```

String Methods

- charAt(index) – returns the character (as a string) at the specified position
- indexOf(string) – returns the index of the specified string
- replace(from, to) – replaces the first argument with the second argument.
- search (regex) – returns the index of the regex search pattern
- slice – returns a substring of a string
- split(separator) – splits a string on separator
- toLowerCase()
- toUpperCase()

EXAMP
LE

Number

- **All numbers are floating point**
  - Decimal fractions are not exact
- **Standard operators +, -, *, /, %**
- **toFixed(n) – returns the number to n decimal places**

EXAMP
LE

Array

- An indexed collection
- Can store anything
- Declared using the literal syntax [ ]
- Many useful methods

```
var collection = ['a', 1, /3/, {}];
collection[0];      // access the first element
collection.length;  // get the number of elements in the array
```

EXAMP
LE

Underscore.js

- Open source library that adds functional programming support to JavaScript
- Lots of helpful functions for working with arrays
- Can be used in an object-oriented or functional style

```
var numbers = [1, 2, 3];

// functional style
_.each(numbers, function (num) {
    write(num);
});

// object-oriented style
_(numbers).each(function (num) {
    write(num);
});
```

EXAMP
LE

Regular Expression

- **A tool for string pattern matching**
- **Used to search, replace and extract parts of strings**
- **Most characters match themselves**
- **Character classes match classes of characters**
  - \w any word character
  - \d decimal digit
- **Match a set of characters [ ]**
- **. matches any character**
- **{ } quantifies a match ie .{2} matches any two characters**
- **( ) makes a capturing group**
- **\ escapes special characters**
- **Regular expression literal /*pattern*/**

EXAMP LE

Date

- No literal syntax

```
var birthday = new Date(2010, 10, 26);
```

- The month parameter is zero based ie. January is 0
- new Date() is the current date

EXAMP
LE

Datejs

- Adds methods to the date object
- http://www.datejs.com
- Documentation is out of date

```
// adding time spans to dates
Date.today().add(3).days();

// get a date
var secondWednesday = Date.march().first().wednesday();
// test
secondWednesday.is().thursday();

// parse
Date.parse("next tuesday");
```

eval

- Interprets strings of JavaScript code

```
eval("alert('Hello World');");
```

- Slow
- Insecure
- Unnecessary
- Do not use eval

EXAMP
LE

JSON

- JavaScript Object Notation
- Uses JavaScript object literals as a data format
- Lightweight, readable alternative to xml
- increasingly used in AJAX web applications

JSON vs XML

```
{
    books:[
        {
            title: "Frankenstein",
            author: "Mary Shelley",
            genres: ["horror", "gothic”]
        },
        {
            title: "Moby Dick",
            author: "Herman Melville",
            genres: ["adventure", "sea"]
        }
    ]
}
```

```
<books>
  <book>
    <title>Frankenstein</title>
    <author>Mary Shelley</author>
    <genres>
      <genre>horror</genre>
      <genre>gothic</genre>
    </genres>
  </book>
  <book>
    <title>Moby Dick</title>
    <author>Herman Melville</author>
    <genres>
      <genre>horror</genre>
    </genres>
  </book>
</books>
```

Parsing JSON

- **Use json2.js (http://www.json.org/js.html)**
- **Two important functions**
  - parse – converts JSON to JavaScript objects
  - stringify – converts JavaScript objects to JSON
- **Parsing with json2 is faster, more robust and more secure than parsing with eval**

EXAMP
LE

isNaN

- Top-level function
- Simple way to test for numbers and failed mathematical expressions

EXAMP
LE

parseFloat

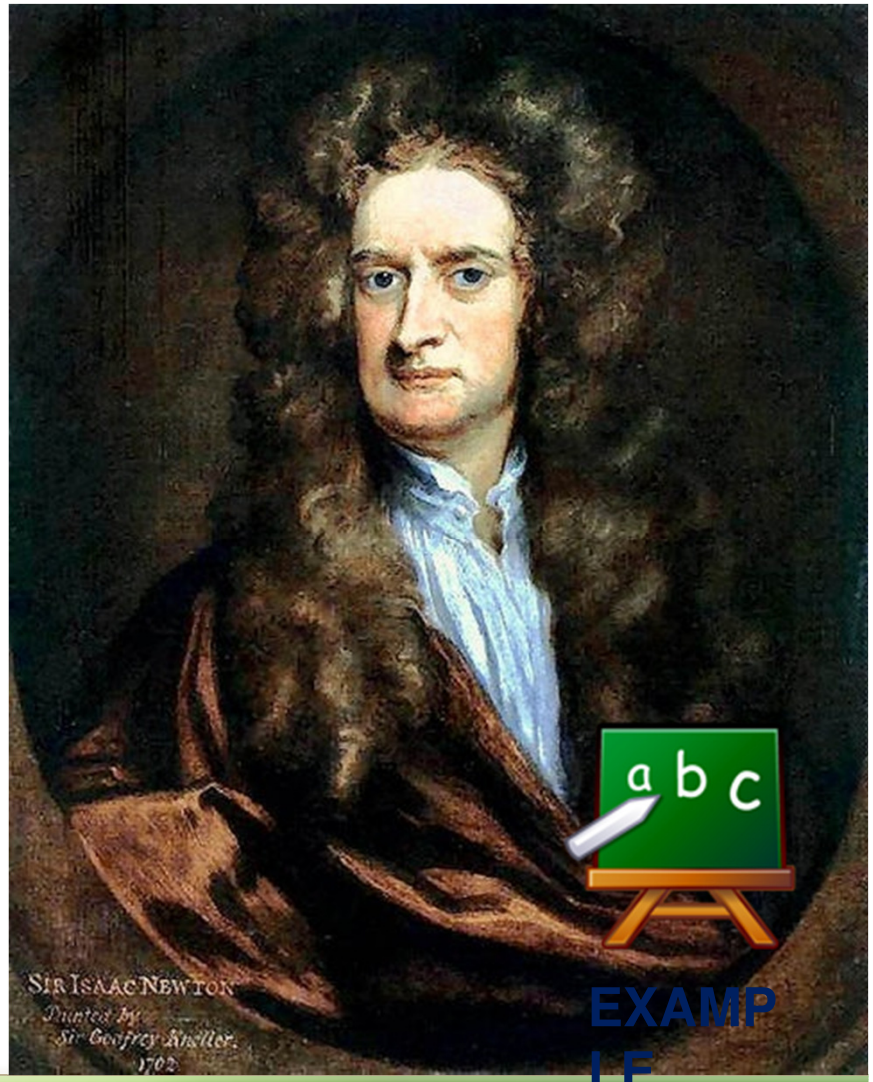
- Converts a string to a number

```
var age = parseFloat("48");

var PIApprox = parseFloat("0.0314E+2");
```

The Math Object

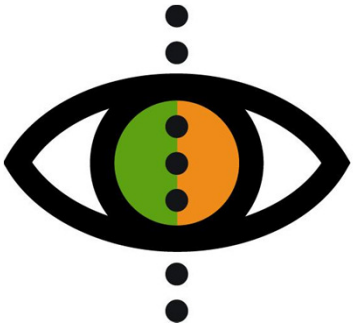
- abs
- floor
- ceil
- pow
- random
- round



Summary

- **Data Types**
  - String, number, date, regular expression
- **Collection Types**
  - Object and array
- **Underscore.js**
- **JSON**
- **isNaN, parseFloat and Math**

For more in-depth online developer training visit



pluralsight

see what you can learn

on-demand content from authors you trust