Quiz #1 Solutions

Question 1

Markup Languages extend content with tags that identify the content for special processing. For example, they are used to give a structure to a document by annotating it and dividing it into human-readable sub-sections. Variables are a foreign concept to Markup Languages since we cannot assign values to memory locations in the hopes of reusing them. Similarly, we cannot, in a pure Markup Language, run programming instructions, and therefore cannot iterate through instructions or selectively run them based on conditions.

Question 2

Problem: the browser will try to render the text content that looks like HTML syntax (with < and >) as actual HTML elements. Solution: Escape the characters, like < and >

Question 3

The standard Web App JavaScript development tool chain includes a transpiring step that maps new JavaScript features to old-style JavaScript allowing it to run in old browsers. That mapping step isn't present for HTML so care needs to be taken to avoid sending down HTML that is too new for the browser.

Question 4

4.1.a) If a element has been given an id or a class and is referred to in the CSS style using that id/class (e.g   p.text { color: blue;}) then the latter style will gain priority over the first one for this element because it's more specific to it.

4.1.b) If a element is contained within a different element (e.g   Text) then any style block that references the element with mention of its parent(s) (eg div p { color: blue;}) will have priority over the first style for this element because it's more specific to it.

4.2) If the element contains other elements (e.g   Text   ), if no style text color is specified for those inner elements, any text they contain will take the color dictated by this rule, i.e their text will be green unless a different CSS rule specifies otherwise.

Question 5

a) `<a href="http://www.example.com/a/b/c.html#d">Link</a>`
   http://www.example.com/a/b/c.html#d

b) `<a href="/a/b/c.html#e">Link</a>`
   http://www.example.com/a/b/c.html#e

c) `<a href="a/b/c.html">Link</a>`
   http://www.example.com/a/b/a/b/c.html

d) Link
http://www.example.com/a/b/c.html#f

Question 6

6.1

```
console.log(o1 === o2);
```
false

```
console.log(o1 === o3);
```
false

```
console.log(o2 === o3);
```
true

```
console.log(JSON.stringify(o1) == JSON.stringify(o2));
```
true

```
console.log(JSON.stringify(o1) == JSON.stringify(o3));
```
true

```
console.log(JSON.stringify(o2) == JSON.stringify(o3));
```
true

6.2

```
console.log(o2 === o3);
```
true

```
console.log(JSON.stringify(o1) == JSON.stringify(o2));
```
false

```
console.log(JSON.stringify(o1) == JSON.stringify(o3));
```
false

```
console.log(JSON.stringify(o2) == JSON.stringify(o3));
```
true

Question 7

7.1

console.log 1
ERROR

console.log 2
ERROR

console.log 3
3

console.log 4
ERROR

console.log 5
ERROR

7.2

console.log 1
undefined

console.log 2
undefined

console.log 3
3

console.log 4
1

console.log 5
2

7.3

Prior to the ES6 extensions, scopes were generated in JavaScript by an Immediately Invoked Function Expression (IIFE). Wrapping the block with a function definition that is called like so:

```
(function( )
{
   var a = 1;
   var b = 2;
   console.log(a + b);
}
)();
```

Question 8

8.1. JavaScript Arrays are also Objects and Objects in JavaScript can accept any string as a key. Any non-integer value will also be coerced to a string when used as a key. This will still work using an Object because the integer keys 1 and 0 can also be coerced to a string.

8.2. Yes objects are polymorphic in the sense that Arrays are polymorphic. Different properties in an object can have different types.

Question 9

The newly allocated instance of Rectangle

height
width

The prototype object of the newly allocated instance of Rectangle

area

The Rectangle class function object

countRects

Question 10

For each of the following primitive JavaScript types, would it be possible for a JavaScript __closure__ to hold a variable of that type.

undefined
number
string
boolean
function
object

Briefly explain your answer:

Closures simply capture all the references of a function so they are available after the function exits. Since a function can access something of any type a closure can hold a variable of any type.

Question 11

Yes, "this" can refer to those other types. You can set the `this` variable of a function call the `.call()` or `.bind()` methods. Using those methods you can cause a function to be called with any JavaScript type. For example: `foo.call(undefined), foo.call(1), foo.call('hello'), foo.call(true), foo.call(()=> true), foo.call({})` would set the 'this' to the different types. Unfortunately, the behavior in the original JavaScript (not 'use strict') is more complicated. Undefined and Null get mapped to the global object (windows) and numbers, strings, and boolean are mapped to the object version of them.