# CMPT 165

## INTRODUCTION TO THE INTERNET AND THE WORLD WIDE WEB

By Hassan S. Shavarani

## *UNIT7: WORKING WITH JAVASCRIPT*

# TOPICS

1. Variables, Again
2. JavaScript Operators Types
3. Iteration: the for-loop
4. Working with Strings
5. Making Decisions
6. HTML Forms

# VARIABLES, AGAIN

# WHAT VARIABLES ARE FOR

to store some value so we can use it later!

if you don't need it later,
just the function call will be enough!

# VARIABLE VALUES

There are two things you can do with variables:

- store values in them with a variable assignment statement
- retrieve the stored value by referring to the variable by name

# VARIABLES

- help the code be more readable
- aid modification of their values
- just sometimes make the code more wordy!

```
setup = function() {
    $('button').click(do_things)
}
$(document).ready(setup)
/////////Equal to/////////
$(document).ready(function() {
    $('button').click(do_things)
})
```

# JAVASCRIPT OPERATORS TYPES

# JAVASCRIPT ARITHMETIC OPERATORS

| Operator | Description |
|---|---|
| + | Addition |
| - | Subtraction |
| * | Multiplication |
| / | Division |
| % | Modulus (Remainder) |
| ++ | Increment |
| -- | Decrement |

content from https://www.w3schools.com/js/js_operators.asp

# JAVASCRIPT ASSIGNMENT OPERATORS

| Operator | Example | Same As |
|---|---|---|
| = | x = y | x = y |
| += | x += y | x = x + y |
| -= | x -= y | x = x - y |
| *= | x *= y | x = x * y |
| /= | x /= y | x = x / y |
| %= | x %= y | x = x % y |

content from https://www.w3schools.com/js/js_operators.asp

# JAVASCRIPT COMPARISON OPERATORS

| Operator | Description |
|---|---|
| == | equal to |
| === | equal value and equal type |
| != | not equal |
| !== | not equal value or not equal type |
| > | greater than |
| < | less than |
| >= | greater than or equal to |
| <= | less than or equal to |
| ? | ternary operator |

content from https://www.w3schools.com/js/js_operators.asp

# JAVASCRIPT OPERATORS PRECEDENCE

| Value | Operator | Description | Example |
|---|---|---|---|
| 19 | ( ) | Expression grouping | (3 + 4) |
| | | | |
| 18 | . | Member | person.name |
| 18 | [] | Member | person["name"] |
| | | | |
| 17 | () | Function call | myFunction() |
| 17 | new | Create | new Date() |
| | | | |
| 16 | ++ | Postfix Increment | i++ |
| 16 | -- | Postfix Decrement | i-- |

content from https://www.w3schools.com/js/js_arithmetic.asp

# ITERATION: THE `for` LOOP

in most programming languages (including JavaScript)

`for` loop is used when you have

a "definite" number of times to iterate

```
for (step = 1; step <= 10; step += 1) {
    $('#example1').append('<p>One more paragraph.</p>')
}
```

```
for (n = 1; n <= 6; n += 1) {
    $('#example2').append('<p>Paragraph #' + n + '</p>')
}
```

# ITERATION: RAPHAËL EXAMPLE

```
paper = Raphael('container', 400, 200)
circle_attrs = {
    'stroke': '#c50',
    'stroke-width': '2'
}
rect_attrs = {
    'fill': '#292',
    'stroke-width': '1.5'
}
for (count = 0; count <= 11; count += 1) {
    c = paper.circle(10 + count*14, 20 + count*12, 6)
    c.attr(circle_attrs)
    r = paper.rect(250, 100, 40, 40)
    r.attr(rect_attrs)
    r.rotate(count*3)
    r.scale(3 - count*0.25)
}
```

You can find the implementation of this example here !

# BUILDING STRINGS

when the + operator is applied to strings,
it joins them together or **concatenates** them

```
r = 'f'
g = '5'
b = '0'
newstyle = {
    'color': '#' + r + g + b
}
```

# BUILDING STRINGS

when you try to add
a string and a number together in JavaScript

the number is automatically converted to a string,
and the strings are joined

```
a1 = 'abc' + 123
a2 = 'abc' + '123'
a3 = 'abc123'
```

```
a1 === a2 === a3
```

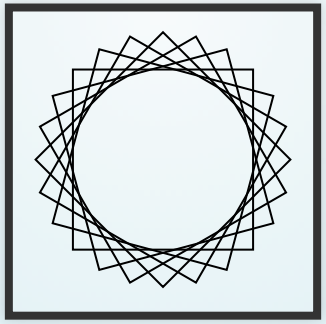
# BUILDING STRINGS: RAPHAËL USE CASE EXAMPLES

`.scale()` and `.rotate()` functions are convenient, but they won't let us animate either scaling or rotation

to do that, we need `.animate()` and the `'transform'` attribute

# BUILDING STRINGS: RAPHAËL USE CASE EXAMPLE 1

```
for (count = 1; count <= 6; count += 1) {
    r = paper.rect(30, 30, 90, 90)
    animation_attrs = {
        'transform': 'r' + (count*15)
    }
    r.attr(animation_attrs)
}
```



* Image from http://www.cs.sfu.ca/CourseCentral/165/common/study-guide/figures/for-build-result.svg

# BUILDING STRINGS: RAPHAËL USE CASE EXAMPLE 2

```
paper = Raphael('container2', 140, 140)
rect_attrs = {
    'fill': '#292',
    'stroke-width': '1.5'
}
for (count = 0; count <= 11; count += 1) {
    r = paper.rect(50, 50, 40, 40)
    r.attr(rect_attrs)
    angle = count*3
    scale = 3 - count*0.25
    anim_attrs = {
        'transform': 'r' + angle + 's' + scale
    }
    r.animate(anim_attrs, 2000)
}
```

let's try the result of this code here

# STRINGS AS OBJECTS

Strings we have been using in JavaScript have the fundamental job of holding characters

they also behave as objects:
they hold variables (including functions) that do useful things with the contents of the string

# STRINGS AS OBJECTS: `.toLowerCase()`

gives back a **copy** of itself
converted to entirely lowercase

```
greeting = 'Hello World!'
lower_greeting = greeting.toLowerCase()
```

```
lower_greeting === 'hello world!'
```

# STRINGS AS OBJECTS: `.charAt()`

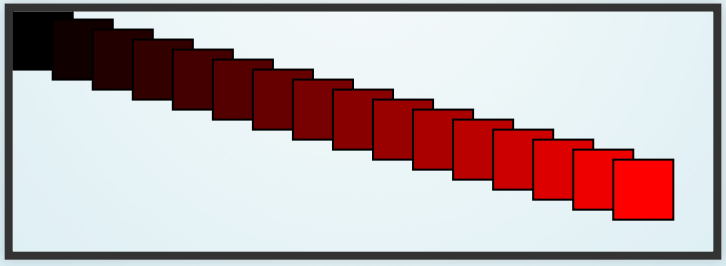
extracts individual characters from a string (counting from zero)

```
letters = 'abcdefg'
third = letters.charAt(2)
```

```
third === 'c'
```

# STRINGS AS OBJECTS: RAPHAËL USE CASE EXAMPLE

```
color_values = '0123456789abcdef'
paper = Raphael('container', 350, 120)
for (red = 0; red <= 15; red += 1) {
    r = paper.rect(red*20, red*5, 30, 30)
    rect_attrs = {
        'fill': '#' + color_values.charAt(red) + '00'
    }
    r.attr(rect_attrs)
}
```



* you can try this example here

* Image from http://www.cs.sfu.ca/CourseCentral/165/common/study-guide/figures/for-build-2-result.svg

# MAKING DECISIONS

sometimes we need to be able to make decisions
based on something the user enters
or some value we calculate, or the time of day, or …

# `if` STATEMENT

we use the `if` statement in JavaScript to make decisions about what code to run

the `if` statement means

*"should this block of code run or not?"*

```
if (count > 100) {
    $('#error').html('That is too many.')
}
```

# REVIEW OVER JAVASCRIPT COMPARISON OPERATORS

the `if` statement condition can be any expression that results in a true or false result

| Comparison | Meaning | Example is… |
|---|---|---|
| `23 == 3` | is equal to | false |
| `23 != 3` | is not equal to | true |
| `4 < 9` | less than | true |
| `4 > 9` | greater than | false |
| `8 <= 6` | less than or equal | false |
| `8 >= 6` | greater than or equal | true |

# `if-else` STATEMENT

*"do this if the condition is true, but that if it isn't"*

the **`else`** which can be added to an **`if`**

lets us do exactly *that*:

code that will run if the condition is false!

```
if (count > 100) {
    $('#error').html('That is too many.')
} else {
    $('#sucess').html('Count is reasonable')
}
```

# MAKING DECISION EXAMPLE

Let's look at Hiding Optional Content Example

# HTML FORMS

In previous versions of HTML,
all inputs had to be inside a `<form>` element
that is not a must anymore in HTML5

here you can see an example use case of what the
`<form>` element looks like

# HTML FORMS

forms can contain different elements (components)
some of which we will review in the next few slides

## `<input>`

The things `<input />` puts on the page are often called controls or widgets;

# DIFFERENT TYPES IN `<input>`

- `text`
- `button`
- `radio` buttons (select one of many options)
- `checkbox` (check on or off)
- `password` input
  (like text, but characters aren't visible on-screen)

# DIFFERENT TYPES IN <input>: EXAMPLE

```
<p>
    Type something:
    <input type="text" id="something" />
</p>
```

```
<p>
    Here is
    <input type="button" id="btn"
        value="something to click" />
</p>
<p>
    Here is
    <button id="btn">something to click</button>
</p>
```

# <textarea>

a text input where the user has space to enter more text

you can specify:

- the number of rows and columns (lines and width)
- initial text as the contents (or leave it empty for none)

```
<textarea id="t-area" cols="30" rows="5">
    This has lots of space for text.
</textarea>
```

# <select> AND <option>

creates a drop-down list with the defined options

```
Choose one:
<select id="seldemo">
    <option value="a">Apple</option>
    <option value="b">Banana</option>
    <option value="c">Cherry</option>
</select>
```

here is an example of this element combined with the decision making syntax

# JQUERY AND FORMS: `.val()` FUNCTION

we can then use jQuery `.val()` function to retrieve the value of an HTML form element

```
<p>Type something: <input type="text" id="something" />
<button id="resultbutton">Go</button></p>
<p id="result"></p>
```

```
show_result = function() {
    typed = $('#something').val()
    $('#result').html('You typed this: ' + typed)
}
setup = function() {
    $('#resultbutton').click(show_result)
}
$(document).ready(setup)
```

let's try this code here

# JQUERY AND FORMS: EXAMPLE

here is another example of using the `<input>` elements to modify a Raphaël paper object

Any Questions?