CMPT 165, Spring 2020

Working with JavaScript

1

Aside: Variables

- The basic reason for a variable: keep track of something we need later.
- Remember that variables can be named whatever you want: try to be descriptive, which will make your code more readable.
  - e.g. we called the SVG object paper because the Raphaël docs call it a paper object.
  - e.g. the style information in bigger (a few examples ago) described the transformation the object was going to do: make an SVG element larger.
  - e.g. the paper object gets used many times to draw on it: hat = paper.rect(…).
  - e.g. when creating shapes: we put the element object in a variable so we can style it later: hat.attr(…).

2

Aside: Variables

- e.g. when drawing circles of different size, we needed to keep track of the radius so we could change it and use it again:

```
radius = 50
more = function() {
    circ = paper.circle(50, 50, radius)
    radius = radius - 4
}
```

3

Aside: Variables

- Calling the variable by name is just a way to refer to its value. That is, these are exactly the same, except one uses a variable to hold the attribute information:

```
shape.attr({
    'fill': '#f55',
    'stroke': '#f00'
})
```

```
red_attrs = {
    'fill': '#f55',
    'stroke': '#f00'
}
shape.attr(red_attrs)
```

- The variable name gives a hint what was happening, which makes the code more readable; we could use the value many times if we needed to.

4

Aside: Variables

- The same is true for variables holding functions. These are the same in JavaScript:

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

```
$(document).ready(function() {
    $('button').click(do_things)
})
```

5

The for loop

- We need to be able to repeat the same piece of code several times. Repeating a chunk of code is iteration, repetition, or looping.
- For example, we could add many HTML elements at once, or draw several similar shapes.
- The for loop is used to repeat some code a known number of times. i.e. when you start the loop, you know it's going to run 10 or 20 or n times.

6

The for loop

- This loop runs 10 times:

```
for(i=1;i<10;i+=1){
  $('#container').append("<p> another paragraph</p>")
```

- The body of the loop {…} runs with n=1, n=2, …, n=10.

7

The for loop

```
for(i=1;i<10;i+=1){
  $('#container').append("<p> another paragraph</p>")
```

- for (…) {…}: The basic structure of a for loop.
- (…; …; …): there are three things we have to say to control the loop.
- {…}: the body of the loop, code that runs several times.

8

The for loop

```
for(i=1;i<10;i+=1){
    $('#container').append("<p> another paragraph</p>")
```

- n = 1: starts the loop by setting the variable n to 1, the loop counter.
- n <= 10: continue the loop as long as n is less than or equal to 10. This could equivalently be n < 11.
- n += 1: each time around the loop add one to n. Equivalent to n = n + 1 and n++.

9

The for loop

```
for(i=1;i<10;i+=1){
    $('#container').append("<p> another paragraph</p>")
```

- The result is that the loop uses the variable n to count 1, 2, 3, 4, …, 10, and runs the loop body each time.
- So we add 10 paragraphs to the page
- The structure of our for loops code will be almost exactly like this every time.
- We might change the variable name to something more meaningful. We might change the start/end values. We will change the body to do whatever we need repeated

10

The for loop

- The loop counter can be used like any other variable. That will let us do something different each time we go around the loop.
- For example, we can use it to change the content of the paragraph each time:

```
for (count = 1; count <= 5; count += 1) {
    markup = '<p>Paragraph #' + count + '</p>'
    $('body').append(markup)
}
```

11

The for loop

```
for (count = 1; count <= 5; count += 1) {
    markup = '<p>Paragraph #' + count + '</p>'
    $('body').append(markup)
}
```

- This builds a string in the variable markup. e.g. the first time around the loop it will be:

```
'<p>Paragraph #1</p>'
```

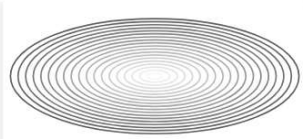
- … then that HTML is appended to the page body. Result: the code adds 5 paragraphs, each slightly different.

12

The for loop

- We can do anything in the loop body we like. The last examples have been jQuery manipulation.
- We can also do Raphaël stuff (or any other JavaScript logic).
- e.g. draw a bunch of ellipses, calculating the radii and some appearance details from the loop counter:

```
paper = Raphael('container', 200, 100)
for (count = 0; count <= 19; count += 1) {
    e = paper.ellipse(100, 50, count*5, count*2)
    e.attr({ 'opacity': count/30 })
}
```



13

The for loop

```
paper = Raphael('container', 200, 100)
for (count = 0; count <= 19; count += 1) {
    e = paper.ellipse(100, 50, count*5, count*2)
    e.attr({ 'opacity': count/30 })
}
```

- This creates the Raphaël paper object before the loop, since we only want one of them. Then, each time around the loop, draw an ellipse and change an attribute.

14

Manipulating Strings

- Review…
  - A string is a sequence of characters.
  - To write a literal string in JavaScript, wrap in quotes: 'string' or "string" .
- Note that these are different. The first is a string; the second is a number:

```
a = '12'
b = 12
```

- We can prove they're different if we add them:

```
c = a + a
d = b + b
e = a + '34'
f = b + 34
```

- The results: c is the string '1212'; d is the number 24; e is '1234'; f is 46.

15

Manipulating Strings

- The types of values really matter.
  - Applying + on numbers: arithmetic addition.
  - Applying + on strings: join them together (concatenate).
- We often need strings as arguments to functions.
  - e.g. jQuery's .append() takes a string (representing HTML code of the addition to the page).
  - e.g. all of the CSS property values (for jQuery) and attribute values (for Raphaël) are strings.
- Above, we created a string (containing HTML markup for .append()) with this code:

```
markup = '<p>Paragraph #' + count + '</p>'
```

- But count is a number: in JavaScript, the number is converted to a string, then concatenated.

16

Manipulating Strings

HSL and HSV colour

- Another example: changing colours.
  - We can specify colours (in CSS and SVG) as we have (#ab0 or #aabb00) or some other formats like rgb(100%, 50%, 0%) or hsl(360, 100%, 75%).
  - My goal: produce a bunch of elements having different colours, varying the hue in an HSL colour.
- Values for the hue are 0–360: I want to add 18 elements, with the hues moving 20 steps with each. We can start:

```
for (p=0; p<18; p+=1) {
    hue = p*20
```

17

Manipulating Strings

- Then we can build the style information we want:

```
  colour='hsl('+hue+',100%,50%)'
style={'background-color':colour}
```

- And put stuff on the page to use that style:

```
$('#container').append('<p> another paragraph</p>')
$('#container p').last().css(style)
```

18

Manipulating Strings

- All together.

```
$(function(){

for(p=1;p<18;p+=1){
   hue=p*20;

   colour='hsl('+hue+',100%,50%)'
style={'background-color':colour}


   $('#container').append('<p> another paragraph</p>')
   $('#container p').last().css(style)
}
})
```

19

Manipulating Strings

- Or a more compact version that does the same thing:

```
for (hue=0; hue<360; hue+=20) {
    $('body').append('<p>Hello world</p>')
    $('body p').last().css({
    'background-color': 'hsl(' + hue + ', 100%, 50%)'
    })
}
```

- With a little bit of skill building the strings we need, can get a lot done.
- It just takes a little planning: what values do you need (e.g. calculated numbers), and how should they fit into a string to get the result you want?

20

Manipulating Strings

- e.g. for Raphaël animated transformations, we need strings like 'r30s1.5' to represent 30° rotation and 1.5× scaling.
- We can construct those like:

```
angle = …
scale = …
attribs = {
  'transform': 'r' + angle + 's' + scale
}
shape.animate(attribs)
```

21

Strings as Objects

- Strings are objects in JavaScript (and so are number and everything else we can put in a variable).
- Strings contain some variables like .length:

```
greeting = 'Hello World!'
len = greeting.length
```

- After this, len will be 12.
- They also have some functions like .toLowerCase().

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

- lower_greeting will be 'hello world!'.

22

Strings as Objects

- We can pull characters out of a string with .charAt().
- for a string s, calling s.charAt(2) gives us the character in position 2 (counting from 0, so the third character in the string).

```
letters = 'wxyz'
c = letters.charAt(2)
```

- Here, c will be 'y'.

23

Strings as Objects

- We can use this to extract the right character from a specific string. For example, we'll start with the string '0123456789abcdef' representing the 16 colour code values.
- Then we can use .charAt() to get a value that fits is a colour code and use it to build a colour value.

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

24

Strings as Objects

```
color_values = '0123456789abcdef'
red_component = color_values.charAt(red)
```

- It's a funny little trick, but another good example of how basic pieces of logic can combine to get a lot done.

25

HTML Forms

- The form tags in HTML are used to insert form controls on pages.
- We have seen one: <button>. We used it to create something that the user could click (and that looked clickable).
- In HTML, the content of <button> is the label:

```
<p>Click it: <button id="b">Go</button></p>
```

- And on the JavaScript side:

```
$('#b').click(button_click)
```

- … and define button_click.

26

HTML Forms

- There are many other form controls we can use, mostly with the <input> tag. Its type attribute sets the type of control.
- Using type="text" will produce a one-line text input:

```
<div class="form">What's your name?
<input type="text" id="name" />
</div>
```

<input> tag

27

HTML Forms

- Then we can examine what the user typed in JavaScript. Selecting the control (with jQuery) and using the .val() function will give us the string the user typed. Then we can use that like any other string.

```
handle_text = function() {
    name = $('#name').val()
    markup = '<p>Your name is ' + name + '</p>'
    $('body').append(markup)
}
```

28

HTML Forms

- Once we have a way for the user to give us some input, we can use it to change the result of our code.
- Another example: use the entered text to build some stuff in SVG. We will start with some form elements for the user to interact with:

```
<div id="container"></div>
<div class="form">
Type something:
<input type="text" id="svg-text" />
<button id="add">Add it</button>
</div>
```

29

HTML Forms

- We can add a line of text with each click:

```
n = 0
add = function() {
  entered_text = $('#svg-text').val()
  txt = paper.text(100, 5 + 10*n, entered_text)
  n = n + 1
}
setup = function() {
  paper = Raphael('container', 400, 100)
  $('#add').click(add)
}
$(document).ready(setup)
```

30

HTML Forms

- Things happening in that code:
  - The variable n keeps track of how many lines were added: we adjust the text's y coordinate so they appear below the previous.
  - Get the text from the <input type="text"> with .val().
  - Add it to the SVG with the paper's .text() function.
- What if we wanted another input on the form where the use can select the colour (or rotation, etc) of the text?
  - Form controls have other events besides .click().
  - For example, .change() that is triggered every time a form input is modified by the user (after they move focus off the element, e.g. click somewhere else).

31

HTML Forms

- We could have done the previous example without the button, adding text whenever the user types some:

```
setup = function() {
    paper = …
    $('#svg-text').change(add)
}
```

32

Other Controls

- There are several other form controls you can add to pages (and work with from JavaScript).
- The password input works just like type="text" except the user can't see what they have typed.

```
<input type="password" id="pw" />
```

- You can still get the string typed with the jQuery .val() function.

33

Other Controls

- A dropdown select gives the user some options and let them choose.

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

- Here, .val() gives the value from the selected <option> as a string. In this case, one of 'a', 'b', or 'c'.

34

Other Controls

- A multi-line text input is created with <textarea>: its contents are the initial value of the text box.

```
Tell me more:
<textarea id="comments" cols="30" rows="5">This
has lots of space for text.</textarea>
```

- Again, .val() works the same way: gives you whatever the user entered as a string.
- There are several more form tags, but those will be enough for us.

35

Making Decisions

- We need one more way to control the way our code fits together: a way to decide whether or not to run a chunk of code.
- The if statement will let us do this. The idea: check some condition, and only execute a block of code if the condition was true.
- For example, we want to run some code for large value of the variable count.

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

- The code in the {…} will run if count is more than 100, but is skipped otherwise.

36

Making Decisions

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

- The structure of an if:
  - if (…) {…}: The basic structure.
  - (…): the condition that controls the behaviour. It is an expression that evaluates to true or false.
  - {…}: the body of the conditional. The code that runs if the condition evaluated to true.

37

Conditions

- There are several operators we can use to build a condition. A condition is just an expression that will evaluate to true or false.
- All of these make a comparison between two values and ask if it's correct.
  - a < b, a > b: is a less-than/greater-than b?
  - a <= b, a >= b: is a less-than-or-equal-to/greater-than-or-equal-to b?
  - a == b: are a and b the same?
  - a != b: are a and b different?
- We have actually been using a condition to decide if the for loop should continue: keep looping as long as it is true.

```
for (n = 1; n <= 10; n += 1) {
    $('body').append('<p>another paragraph</p>')
}
```

38

Conditions

- We'll probably use the if with forms: that's the most likely case where we don't know what value we'll have while writing the code.

```
<div id="container"></div>
<div class="form">Which way?
    <select id="direction">
        <option value="up">Up</option>
        <option value="down">Down</option>
        <option value="left">Left</option>
        <option value="right">Right</option>
    </select>
    <button id="go">Move</button>
</div>
```

39

Conditions

- We can start with a familiar pattern: create a Raphaël paper and connect the button event to a function.

```
setup = function() {
    paper = Raphael('container', 200, 200)
    shape = paper.circle(100, 100, 40)
    $('#go').click(move_it)
}
$(document).ready(setup)
```

40

Conditions

- When the button is clicked, we can check what the user has selected, and write conditions for the things we want to do.

```
move_it = function() {
    dir = $('#direction').val()
    if ( dir == 'up' ) {
        trns = 't0,-30'
    }
    if ( dir == 'down' ) {
        trns = 't0,30'
    }
    shape.animate({ 'transform': trns }, 1000)
}
```

[Previous slide + this one = complete `.js` file.]

41

Conditions

- The code for move_it isn't really complete: it doesn't deal with the left and right options.
- What would happen if we selected them with the code as it is? How do we complete the code?

42

If … Else

- In the previous examples, we expect that the conditions cover all the cases exactly, but we can easily end up repeating ourselves.

```
if ( x < 10 ) {
    …
}
if ( x >= 10 ) {
    …
}
```

- We're comparing x to 10. Why should we have to say it twice?

43

If … Else

- We can add an else to the if statement. The idea: if the condition is true, run the if body. If it wasn't, run the else body.
- It looks like:

```
if ( x < 10 ) {
    …
} else {
    …
}
```

44

If … Else

- We can clean up the previous example (to deal with the up and down cases, at least) to:

```
move_it2 = function() {
    dir = $('#direction').val()
    if ( dir == 'up' ) {
        trns = 't0,-30'
    } else {
        trns = 't0,30'
    }
    shape.animate({ 'transform': trns }, 1000)
}
```

45