CSY1018
Web Development

Tom Butler
thomas.butler@northampton.ac.uk

Topic 2

- Quick recap of the last topic
- Javascript events
- Setting CSS properties with Javascript
- Moving elements
- Detecting which key is pressed

Javascript

- Javascript is included on web pages using the <script> tag
- The <script> tag is placed inside the <head> element
- The src attribute is the name of a file containing your javascript code

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <script src="script.js"></script>
  </head>

  <body>
    <h1>Page heading</h1>
    <p>Page content</p>
  </body>
</html>
```

<script> tag

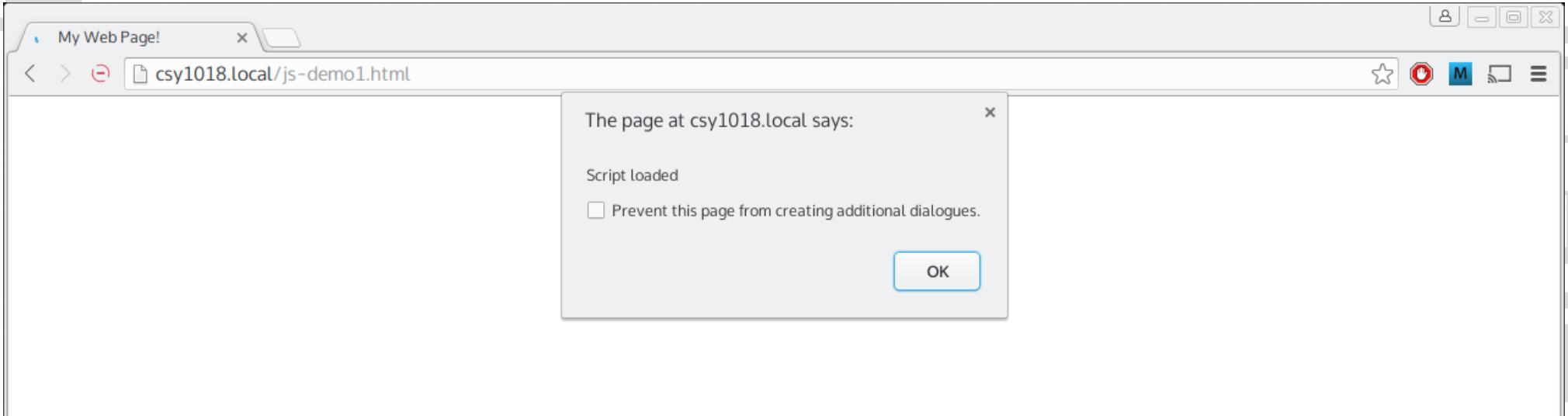
- HTML files are processed from the top to the bottom
- This means the <head> and <script> tags are loaded before the <body> element

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <script src="script.js"></script>
  </head>

  <body>
    <h1>Page heading</h1>
    <p>Page content</p>
  </body>
</html>
```

Javascript

- If you add some code to the javascript file it is run before the elements on the page exist

```
alert('Script loaded');
```



Javascript

- Because of this, if you want to write code that uses the elements in some way (e.g. changing the content) you must run the code to do this **after** the page is loaded

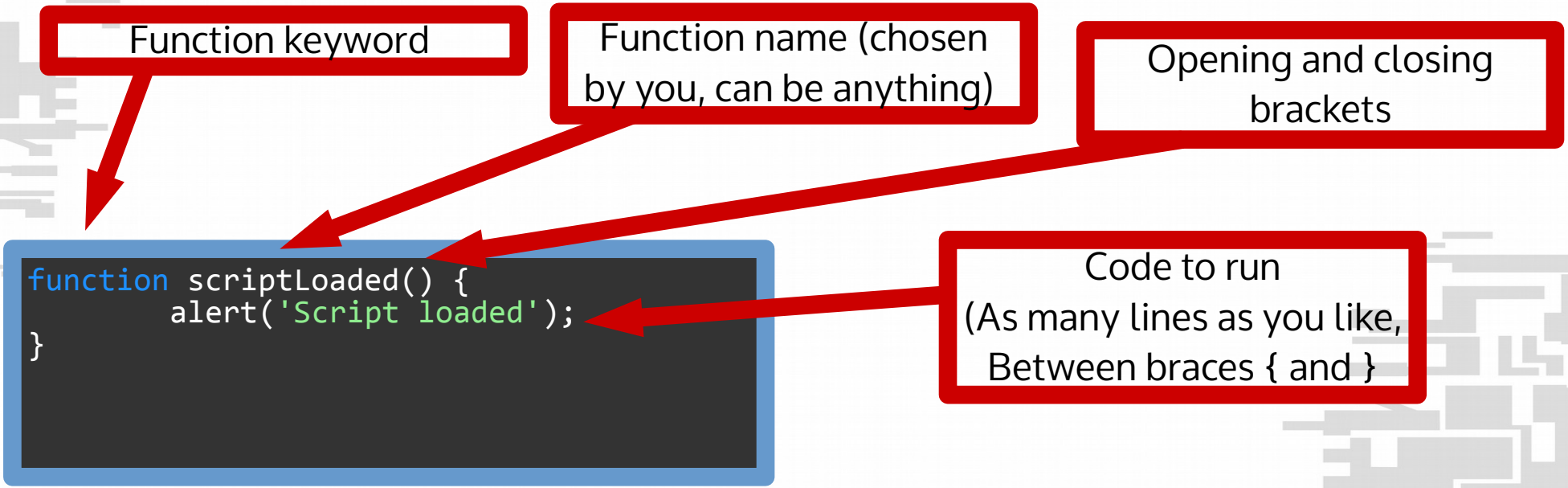
Functions

- You can *label* a block of code using a *function*
- This will store the code for later use where it can be referenced and run
- This allows you to write code out of sequence

```
function scriptLoaded() {
        alert('Script loaded');
}


function addition() {
        var num1 = 5;
        var num2 = 6;

        var num3 = num1 + num2;

        alert(num3);
}
```

Functions

- The syntax for a function looks like this:



Functions

- When code is stored inside a function it is not executed as it's defined
-

```
function scriptLoaded() {
        alert('Script loaded');
}


function addition() {
        var num1 = 5;
        var num2 = 6;

        var num3 = num1 + num2;

        alert(num3);
}
```

- This will not display either alert box!

Selecting elements in Javascript

- Javascript contains functions for selecting HTML elements so you can change properties on the (css, attributes, etc)
- The simplest way is to give an element an ID in the HTML:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <script src="script.js"></script>
  </head>

  <body>
    <h1 id="pageheading">Page heading</h1>
    <p>Page content</p>
  </body>
</html>
```

Selecting elements with Javascript

- Once an element on the page has an ID, you can use the javascript function document.getElementById() to select it and store the *element* in a variable

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Web Page!</title>
    <script src=”script.js”></script>
  </head>

  <body>
    <h1 id=”pageheading”>
         Page heading
    </h1>
    <p>Page content</p>
  </body>
</html>
```

```
var element = document.getElementById('pageheading');
```

Selecting elements

- Once you have an element you can make changes to it
- E.g. to update the content you can use:

```
var element = document.getElementById('pageheading');
element.firstChild.nodeValue = 'New Heading';
```

Javascript

- Because this code requires the elements to exist on the page, it must be run after the page has loaded
- This can be done in two steps:
  - 1) Move the code into a function so it is not run immediately
  - 2) Tell the browser to call the function when the page loads

Javascript

1) Move the code you want to run when the page loads into a function

```
function myLoadFunction() {
        var element = document.getElementById('pageheading');

        element.firstChild.nodeValue = 'New Heading';
}
```

- Note: This function can have any name you like!

Javascript

2) Inform the browser you want to run this function when the page loads

- This is done using the inbuilt function document.addEventListner function

```
function myLoadFunction() {
        var element = document.getElementById('pageheading');
        element.firstChild.nodeValue = 'New Heading';
}


document.addEventListener('DOMContentLoaded', myLoadFunction);
```

DOMContentLoaded means when the content on the Page is loaded (the elements exist)

The name of the function

Javascript

- addEventListener is a very useful function
- It allows you to run a function when a specific *event* occurs

```
document.addEventListener('DOMContentLoaded', myLoadFunction);
```

When this happens

Run the function with this name

Click Events

- You can call element.addEventListener to add an event to a specific element
- Last week you ended with a page that allowed you to click on elements to update their content

Last Week's Exercise 6

```
function updateParagraph() {
        var element = document.getElementById('paragraph');
            element.firstChild.nodeValue = 'New Heading';
}

function updateH1() {
        var element = document.getElementById('pageheading');
            element.firstChild.nodeValue = 'New paragraph contents';
}

function myLoadFunction() {
        var element = document.getElementById('pageheading');
        element.addEventListener('click', updateH1);

        var element = document.getElementById('paragraph');
        element.addEventListener('click', updateParagraph);
}


document.addEventListener('DOMContentLoaded', myLoadFunction);
```

Update the contents Of the p element

Update the contents Of the h1 element

When the H1 is clicked, run the function updateH1

When the paragraph is clicked, run the function updateParagraph

When the page loads, run myLoadFunction

Exercise 1 (Last Week recap)

- 5-10 Minutes
- 1) Download the code from GitHub: https://github.com/CSY1018/Topic2 either clone the branch Exercise1-Exercise or download it as a zip
- 2) Add a <script> tag to the page so you can run some javascript
- 3) Add an event listener for DOMContentLoaded and a function that it runs when the page loads
- 4) In the load function you created, add a click event to the element with the ID `circle` so that when the circle is clicked an alert box appears and says "The button was pressed"
- **Note: You will need this for the rest of today's exercises**

Exercise 1 - Solution

```
function myClickEvent() {
  alert('The button was pressed');
}

function myLoadEvent() {
  var element = document.getElementById('circle');
  element.addEventListener('click', myClickEvent);
}

document.addEventListener('DOMContentLoaded', myLoadEvent);
```

When the button is clicked, run the function myClickEvent

When the page loads, run myLoadFunction

Setting CSS Properties

- You can set CSS properties on an element using javascript
- Firstly you have to select the element by using document.getElementById
- Once you have a reference to the element in a variable you can change the CSS on it using

```
element.style.propertyName = 'propertyValue';
```

Setting CSS Properties

- E.g. to set the width and height of an element

```
element.style.width = '50px';
element.style.height = '50px';
```

- Or Text colour:

```
element.style.color = '50px';
```

- Any CSS property can be used and set
- However, the names are slightly different for some properties

CSS Properties

- Some CSS properties contain hyphens, e.g.
  - background-color
  - border-radius
  - font-family
- In Javascript these are written by removing the hyphen and making the first letter of the second word uppercase:

```
element.style.backgroundColor = 'green';
element.style.borderRadius = '50px';
element.style.fontFamily = 'Verdana, Helvetica, Sans-serif';
```

CSS Properties

- The value is placed in quotes as it as a string, however the outcome is the same

```
var element = document.getElementById('paragraph');
element.style.backgroundColor = 'green';
element.style.borderRadius = '50px';
element.style.fontFamily = 'Verdana, Helvetica, Sans-serif';
element.style.height = '50px';
```

- Will have the same effect as the CSS

```
#paragraph {
  background-color: green;
  border-radius: 50px;
  font-family: Verdana, Helvetica, Sans-serif;
  height: 50px;
}
```

CSS in Javascript

- The javascript code is longer but it can be run at **any time.** CSS is applied once, when the page loads
- Javascript can be used to change the look of an element after the page has loaded
- Usually this is useful when an event occurs

Exercise 2

- < 5 minutes
- 1) Building on exercise 1, amend the code so that when the button is clicked its background colour is set to blue

Exercise 2 - Solution

```
function myClickEvent() {
  var element = document.getElementById('circle');
  element.style.backgroundColor = 'blue';
}

function myLoadEvent() {
  var element = document.getElementById('circle');
  element.addEventListener('click', myClickEvent);
}

document.addEventListener('DOMContentLoaded', myLoadEvent);
```

Other properties

- There's a CSS property called opacity
- This acts as transparency and takes a value from 0 – 1.
- 1 is completely opaque
- 0 is completely transparent
- 0.5 is half transparent

Exercise 3

- < 5 minutes
- 1) Amend exercise 2 so that in the load event, the circle's opacity is set to 0.5

Exercise 3 - Solution

```
function myLoadEvent() {
  var element = document.getElementById('circle');
  element.style.opacity = '0.5';
}

document.addEventListener('DOMContentLoaded', myLoadEvent);
```

Exercise 4

- 2 minutes
- 1) Amend the page so that when the button is clicked, the opacity is set to 1
  - When the page loads the opacity should be set to 0.5
  - When the button is clicked the button should be completely visible as its opacity is set to 1

Exercise 4 - Solution

```
function myClickEvent() {
  var element = document.getElementById('circle');
  element.style.opacity = '1';
}

function myLoadEvent() {
  var element = document.getElementById('circle');
  element.addEventListener('click', myClickEvent);
  element.style.opacity = '0.5';
}

document.addEventListener('DOMContentLoaded', myLoadEvent);
```

Reading from properties

- You can read a CSS property using the same syntax provided it has been set via javascript originally (this doesn't work if the original property was set via a CSS stylesheet):

```
var element = document.getElementById('circle');
element.addEventListener('click', myClickEvent);
element.style.opacity = '0.5';

var circleOpacity = element.style.opacity;
console.log(circleOpacity);
```

Prints "0.5" to the console

Reading from properties

- Once the value is stored in a variable it can be modified and the original value changed

```
var element = document.getElementById('circle');
var circleOpacity = element.style.opacity;
element.style.opacity = circleOpacity + 0.1;
```

- However, this doesn't quite work!
- This is because CSS values are stored as text
- You need to convert the text into a number before doing any calculations
- The code above will treat both values as *strings* and the opacity value will be the *joined* strings something like '0.50.1' rather than adding them to get 0.6

Converting strings to numbers

- There are two functions for converting strings to numbers
  - parseInt() - converts a string into the closest whole number e.g 123
  - parseFloat() - converts a string into the closest decimal number e.g. 1.23

```
var element = document.getElementById('circle');
var circleOpacity = parseFloat(element.style.opacity);
element.style.opacity = circleOpacity + 0.1;
```

Exercise 5

- 5 minutes
- 1) Change the code for the click event to this

```
var element = document.getElementById('circle');
var circleOpacity = parseFloat(element.style.opacity);
element.style.opacity = circleOpacity + 0.1;
```

- Each time you click on the circle it should get slightly less transparent until it becomes completely opaque
- 2) Change the code so that clicking on the circle makes it become more and more transparent until. If you click enough times it should become invisible!

Exercise 5 - Solution

```
function myClickEvent() {
  var element = document.getElementById('circle');
  var circleOpacity = parseFloat(element.style.opacity);
  element.style.opacity = circleOpacity - 0.1;
}

function myLoadEvent() {
  var element = document.getElementById('circle');
  element.addEventListener('click', myClickEvent);
  element.style.opacity = '0.5';
}

document.addEventListener('DOMContentLoaded', myLoadEvent);
```

Interval

- Along with events that are triggered by the user doing something e.g.
  - click
  - mouseenter
  - mouseleave
  - keyup
  - keydown
- There are also other events that are based on *timers*

Intervals

- You can get the browser to repeatedly run a function based on a time interval
  - e.g. run the function every 10 seconds
  - Every second
  - Every 100th of a second
  - Etc
- This is done with the code:

```
setInterval(functionName, interval);
```

Intervals

```
function myInterval() {
    console.log('myInterval called');
}


 setInterval(myInterval, 1000);
```

- This will run the function myInterval every second
- Intervals are measured in 1000ths of a second so 1000 is equal to 1 second

Exercise 6

- 10 - 15 minutes
- 1) Change the load event so that the button starts with an opacity of 1 (completely visible). Hint: you will need to set the initial value or it won't work.
- 2) Change the click event so that when the button is clicked, it starts a timer using setInterval with an interval of 1 second
- 3) Each time the timer is called, reduce the opacity of the button by 0.1 (It should take 10 seconds for the button to disappear completely)
- **Hint: The timer function will contain the exact same code as you used in the click event in Exercise 5, you get the timer to call the function!**
- 4) Adjust the timer from every second (1000) to every hundredth of a second (10)
- 5) Adjust the timer function so that it only removes 0.01 from the opacity each time
- This will add more 'steps' to the transition and the animation will look smoother

Exercise 6 - Solution

```
function myInterval() {
  var element = document.getElementById('circle');
  var circleOpacity = element.style.opacity;

  element.style.opacity = circleOpacity - 0.01;
}

function myClickEvent() {
  setInterval(myInterval, 10);
}

function myLoadEvent() {
  var element = document.getElementById('circle');
  element.addEventListener('click', myClickEvent);
  element.style.opacity = 1;
}

document.addEventListener('DOMContentLoaded', myLoadEvent);
```

Changing the element's position

- There are many different ways of positioning an element:
  - Margin properties
  - Top/Left/Right/Bottom properties
  - By positioning int inside another element with a margin
- Because of this, unless you know exactly how the element has been positioned it's difficult to read the position of an element from a CSS property

Getting an element's current position

- Javascript provides a simple way of retrieving the element's position on the page regardless of how it has been positioned on the page
- This means you don't have to work out which CSS properties have been used to position it
- This can be done with:

```
var element = document.getElementById('circle');
var positionTop = element.offsetTop;
var positionLeft = element.offsetLeft;
```

Getting an element's current position

```
var element = document.getElementById('circle');
var positionTop = element.offsetTop;
var positionLeft = element.offsetLeft;
```

- This will store the position of the element on the page in the variables positionTop and positionLeft
  - positionLeft will store the distance in pixels the element is from the left of the window
  - positionTop will store the distance in pixels the element is from the top of the window

Moving elements

- This allows you to move an element around the screen regardless of its starting position.
- To move an element 10px to the left of where it currently is you can use:

```
var element = document.getElementById('circle');
var positionLeft = element.offsetLeft;
element.style.left = positionLeft - 10 + 'px';
```

- This reads the current offsetLeft, adds 10 to it
- Note that 'px' is also added to create the valid CSS unit

Exercise 7

- 5 minutes
- 1) Make it so when the button is clicked on, it moves 10 px to the left
- 2) Change the event listener from click to keydown on the document (document.addEventListener('keydown', functionName))
- 3) When a key is pressed, move the button 10px to the left
- 4) Each time you press a key it should move further and further left, moving 10px each time

Exercise 7 - Solution

```
function myKeyDown() {
  var element = document.getElementById('circle');
  var positionLeft = element.offsetLeft;

  element.style.left = positionLeft - 10 + 'px';
}

function myLoadEvent() {
  document.addEventListener('keydown', myKeyDown);
}

document.addEventListener('DOMContentLoaded', myLoadEvent);
```

Detecting which key was pressed

- When you press any key on the keyboard, the button will move left!
- It would be better if only a specific key (for example the left arrow key) moved the circle on the page
- To do this you need to work out which key was pressed

Detecting which key was pressed

- This is done in the event listener
- Each time an event is *triggered* by the browser (clicks, mouse movement, key presses, etc) you can ask the browser to give you some information about what triggered the event
- This is done using the code:

```
function myKeyDown(event) {

}
```

Detecting which key was pressed

```
function myKeyDown(event) {

}
```

- *event* here is a special variable that is created by the browser when the event listener is triggered (e.g. by pressing a key)
- You can read some information about the nature of the event from the variable inside the function

Exercise 8

- < 5 minutes
- 1) Add the the following code to your javascript file

```
function myKeyDown(event) {
  console.log(event.keyCode);
}
document.addEventListener('keydown', myKeyDown);
```

- 2) Make sure you have the console open and press some keys on the keyboard
- You should see different numbers appearing
- 3) Try the same key multiple times. Each key on the keyboard will produce a different number!

event.keyCode

- The keyCode for the left arrow key is 37
- (You can find this out by pressing the left arrow key and looking at the number that appears in the console)
- To make it so that only the left arrow key causes the circle to move on the screen we need to make it so that the code that moves the circle only runs when the keyCode is 37.

If statements

- An *if statement* allows you to test so see if *two values are equal*
- For example, whether or not the keyCode is 37
- If statements take the format:

```
if (condition) {
  //Any code here
  //Will only run
  //If the condition
  //is met
}
```

If statements

- If statements are often used to check if a variable is equal to a specific value
- This is done using the code:

```
if (variablename == value) {
  //Any code here
  //Will only run
  //If the condition
  //is met
}
```

- If the condition evaluates to true, then the code between the braces will run
- If the condition is not met, the code will not run

If statements

- If statements allow you to only run blocks of code in certain circumstances
- This is very useful if you only want some code to run some of the time
- In exercise 7, the code to move the button ran regardless of which key was pressed.
- We only really want the code to run when the keyCode is 37 (the left arrow key is pressed)

```
function myKeyDown() {
  var element = document.getElementById('circle');
  var positionLeft = element.offsetLeft;

  element.style.left = parseInt(positionLeft) - 10 + 'px';
}

function myLoadEvent() {
  document.addEventListener('keydown', myKeyDown);
}

document.addEventListener('DOMContentLoaded', myLoadEvent);
```

If statements

- To check to see if the variable event.keyCode is equal to 37 you can use the code:

```
if (event.keyCode == 37) {
  //Any code here
  //Will only run
  //If the condition
  //is met
}
```

- Note: This is == (Two equals signs!) not =
- In Javascript == is used for comparison, = is only used to set a variable to a specific value

If statements

- One if statement can follow another:

```
if (event.keyCode == 37) {
  //will run when the left arrow is pressed
}
if (event.keyCode == 39) {
  //will run when the left arrow is pressed
}
```

- This will let you perform a different action for each key that's pressed

Exercise 9

- 15-20 minutes
- 1) Add the if statement so that the button moves left when the left arrow key is pressed
- 2) Add an if statement and relevant code to allow the up arrow to move the button up 10px, the down arrow to move the button down 10px and the right arrow to move the button right 10px
  - Hint: You can get the keyCodes for each key in the same way you did for the left arrow using the code from exercise 8 and printing the number in the console
  - Hint: One if statement can directly follow another
- 3) Instead of moving the button 10px, use an interval when the relevant key is pressed and have the button slide off the screen in the direction of the arrow pressed
  - Pressing up should start a timeout that moves the button upwards 1px at a time
  - The animation should look smooth
    - Note: **Do not use a loop for this, use a timeout!**
    - **Hint: Each arrow should trigger a different timer and each timer will need its own function, you'll end up with at least 5 functions! One for the keyPress event and one for each fo the four direction timers**

Exercise 9 - Solution

```
function myKeyDown(event) {
  if (event.keyCode == 37) {
    setInterval(moveLeft, 10);
  }
  if (event.keyCode == 38) {
    setInterval(moveUp, 10);
  }
  if (event.keyCode == 40) {
    setInterval(moveDown, 10);
  }
  if (event.keyCode == 39) {
    setInterval(moveRight, 10);
  }
}

function myLoadEvent() {
  document.addEventListener('keydown', myKeyDown);
}

document.addEventListener('DOMContentLoaded', myLoadEvent);
```

```
function moveUp() {
  var element = document.getElementById('circle');
  var positionTop = element.offsetTop;
  element.style.top = positionTop - 1 + 'px';
}

function moveDown() {
  var element = document.getElementById('circle');
  var positionTop = element.offsetTop;
  element.style.top = positionTop + 1 + 'px';
}

function moveLeft() {
  var element = document.getElementById('circle');
  var positionLeft = element.offsetLeft;
  element.style.left = positionLeft - 1 + 'px';
}

function moveRight() {
  var element = document.getElementById('circle');
  var positionLeft = element.offsetLeft;
  element.style.left = positionLeft + 1 + 'px';
}
```

intervals

- Run the code from exercise 9 and press one of the keys more than once
- It will speed up the more times you press the key!
- This is because each time you press a key it creates a new timer that runs every 10th of second
- Each time one of the timers runs it moves the element

Cancelling timers

- You can store a timer inside a variable
- This is done using the code:

```
var timer =  setInterval(moveLeft, 10);
```

- Once a timer has been stored in a variable it can be stopped using the code:

```
var timer =  setInterval(moveLeft, 10);
clearInterval(timer);
```

Variable Scope

- When a variable is created inside a function it is only accessible inside the function it is created in, and it is recreated each time the function is called
- However, you can create a variable that is available in every function and retains its value when the function is called again
- This is done by declaring the variable outside of any functions:

Variable scope

- The variable myVariable is declared outside the function
- Each time myClickEvent is called the value of the variable is retained from last time (If the variable was declared inside the function this would not happen!)
- This is called a “global variable”

Note: When global Variables are used They have already been Defined so don't need the var keyword

```
var myVariable = 0;

function myClickEvent() {
  myVariable = myVariable + 1;
  console.log(myVariable);
}

function myLoadEvent() {
  document.addEventListener('click', myClickEvent);
}

document.addEventListener('DOMContentLoaded', myLoadEvent);
```

Exercise 10

- 1) Test the code from slide 63 to see how the variable retains its value
- 2) Amend your code from exercise 9 to store the timer in a variable
- 3) When keyDown() is pressed clear the last timer
- 4) Now, when you press a key more than once it should not affect the speed (and pressing a different key will change the direction!)
- 5) Add another key press, when space is pressed stop the circle from moving by clearing the timer
  - Hint: You should only have one timer variable

Exercise 10 - Solution

```
var interval = 0;
function myKeyDown(event) {

  clearInterval(interval);

  if (event.keyCode == 37) {
    interval = setInterval(moveLeft, 10);
  }
  if (event.keyCode == 38) {
    interval =setInterval(moveUp, 10);
  }
  if (event.keyCode == 40) {
    interval =setInterval(moveDown, 10);
  }
  if (event.keyCode == 39) {
    interval =setInterval(moveRight, 10);
  }
}

function myLoadEvent() {
  document.addEventListener('keydown', myKeyDown);
}

document.addEventListener('DOMContentLoaded', myLoadEvent);
```

```
function moveUp() {
  var element = document.getElementById('circle');
  var positionTop = element.offsetTop;
  element.style.top = positionTop - 1 + 'px';
}

function moveDown() {
  var element = document.getElementById('circle');
  var positionTop = element.offsetTop;
  element.style.top = positionTop + 1 + 'px';
}

function moveLeft() {
  var element = document.getElementById('circle');
  var positionLeft = element.offsetLeft;
  element.style.left = positionLeft - 1 + 'px';
}

function moveRight() {
  var element = document.getElementById('circle');
  var positionLeft = element.offsetLeft;
  element.style.left = positionLeft + 1 + 'px';
}
```