CODE AVENGERS

JavaScript 3 Instructions

1. Learn about mouse event as you build a basic painting app

1.1. introduce **onMouseDown**

The next 5 lessons build a **Painting App** with **Paper.js** as you learn about **mouse event handling**

The code on lines 1-3 define a special function called **onMouseDown** Paper.js runs this function every time a mouse button is pressed **down**. The **event** parameter contains information such as **event.point**—the co-ordinates on the screen where the button was pressed

The code on line 2 prints the co-ordinates of the mouse click in the console

1. Click to run the code
2. Click in each of the 4 corners and work out the size of the canvas
3. Enter the canvas width in pixels:
4. Enter the canvas height in pixels:
5. Click when you are done

1.2. introduce **onMouseUp**

onMouseDown is run when a mouse button is **pressed down onMouseUp** is run when a button is **released**

The code on lines 3 & 4 draw a circle at the location the mouse button is released

1. Click to run the canvas in **auto update mode**
2. Click the canvas to draw red circles
3. Change the code to draw **blue** circles
4. Change the circle radius to **80** pixels
5. Click when you are done

1.3. introduce **onMouseMove**

onMouseMove is run every time the mouse moves

The code `Color.random()` on line 3 gets a random color

1. Click
2. Change line 1 to the **onMouseMove** function
3. Change the circle radius to 10
4. Test the app then click

1.4. introduce **onMouseDrag**

onMouseDrag is run every time the mouse is moved while a button is pressed

The code on line 1 gets a random color that is used for the first brush stroke Lines 9-11 set the `brushColor` to **red** when the mouse is released

1. Click
2. Set the initial `brushColor` to **pink**
3. Choose a random `brushColor` in the **onMouseUp** function
4. Change the **opacity** to `.4`
5. Test the app then click

1.5.

Review Quiz Questions:

1. Which function runs when the mouse moves at the same time as a mouse button is pressed?
2. Which function runs when the right mouse button is pressed down?
3. Which function runs when any mouse button is released?
4. Which of the following functions draws a circle with a diameter of 10

2. Test some apps and figure out how they work

2.1.

For each task in this lesson you will test an app and work out how it is coded. Click and test the **example app** by moving, clicking and dragging the mouse.

Replace the $$ in the code with the correct mouse event functions `onMouseUp` `onMouseDown` `onMouseMove` `onMouseDrag` so that your app works the same as the example one. You only need to change the $$; the rest of the code is correct

Click the button under the canvas to run your code

2.2.

Click and test the **example app** by clicking, dragging and moving the mouse. Replace the $$ in the code with `onMouseUp` `onMouseDown` `onMouseMove` `onMouseDrag` so that your app works the same as the example one.

On line 5, `path.selected = true` makes the path appear **selected**, with a blue border around it On line 11, `path.selected = false` **deselects** the shape

On line 10, `e.event.detail` is 2 if the user **double clicks** the mouse

On line 18, `e.delta` gets the distance the mouse moved since the last time that mouse event was triggered

2.3.

As in the previous task, test the app then complete the code Click then try clicking and dragging

Lines 5-8 create a rectangle with a random color

On line 21, `r.fillColor.hue += e.delta.x` changes the hue (color or shade) of the shape

On line 22, `r.rotation += e.delta.y` rotates the shape

On lines 16, `r.size.width += e.delta.x * 2` increase the width of the shape

`e.delta` is the distance moved by the mouse since the last event

WARNING:something strange happens when you drag to the top left!

2.4.

The app in this task has a circle that follows the cursor. This **brush** is created on lines 1-2. On line 7, `brush.position = e.point` is used to move the brush to the same position as the mouse cursor.

2.5.

Review Quiz Questions:

1. Which line of code makes a **blue** border appear around a shape?
2. Which property sets the color inside a shape?
3. Which line of code turns a shape around?
4. Which line of code moves a shape down?

3. Use the mouse to create, move and delete shapes

3.1. object Event Handlers

In this lesson you will use mouse events to move and delete items.

The function on lines 3-15 creates an **ellipse** when the user double clicks. On line 5, `e.event.detail == 2` is true when the user double clicks.

The function on lines 17-27 re-sizes the ellipse as the user drags the mouse. Line 22 creates a rectangle using the position the user pressed the button down (`e.downPoint`) as one corner and the current mouse position (`e.point`) as the other. Line 25, changes the boundary of the ellipse to match the rectangle object.

Unlike `Shape.Rectangle`, **Rectangle** objects are **not** drawn on the screen; they are for positioning other items

1. On line 7, check if the user **single clicked** the mouse
2. On line 8, create a `Shape.Rectangle` object and store it in the **item** variable. As on line 6, use `e.point` as the top left corner and set the width and height to 1

3.2. mouse Event Buttons

In the next task we will use the right mouse button to delete items.

First we will make the `onMouseDown` event only draw items when the left mouse button is clicked.

The following code exits a mouse event function when the middle mouse button is clicked:

```
if (e.event.button == 1) {
  return;
}
```

The middle **mouse button** is button number 1. The other two buttons are numbered 0 and 2

1. Add code to the start of **onMouseDown** so that the shapes are **not** drawn when the right button is pressed
2. Change **onMouseUp** to remove items that are smaller than 5 pixels in width or height

3.3. item Mouse Events

Events can be attached to individual items.

The code on lines 3-9 define a function called `onMouseDownItem` that removes an item that is clicked with the right button. On line 25, `item.onMouseDown = onMouseDownItem` links that function to each shape the user draws.

When a shape is clicked the onMouseDownItem function runs, followed by onMouseDown.

On line 5, the JavaScript keyword `this` refers to the item that was clicked.

1. Change the code to **remove** items when the **right** button is **double clicked**
2. Change line 6 to check if the user has at least **double clicked** (i.e. clicked at least twice quickly)
3. Change line 7 to increase the **hue** of the shape by 30

3.4. moving Items

onMouseDragItem The function on lines 11-14 will move shapes when they are dragged. **e.delta** gets the distance the mouse has moved since the last drag event was triggered.

1. Add code to attach the onMouseDragItem function to each shape that is drawn
2. Click and see what happens when you use left click to move a shape
3. Change onMouseDragItem to only move items when the **right button** is dragged

3.5.

In mouse event functions, the event parameter includes `point` (the position of the mouse when the event was triggered) and `delta` (the distance moved since the last event was triggered)

This parameter also includes an `event` property that contains additional information such as the `button` that was clicked and the number of clicks (`detail`)

In the following code `e.point` is used to draw a circle at the point the mouse was clicked. `e.event.detail` would be used to check if the user **double clicked** the button.

```
function onMouseDown(e) {
  var shape = new Shape.Circle(e.point, 20);
  shape.fillColor = Color.random();
}
```

Check the reference for **Event.event.button** and **Event.event.detail** for more information

Review Quiz Questions:

1. What shape is drawn when the left mouse button is double clicked?
2. What shape is drawn when the right mouse button is double clicked?
3. What shape is drawn when the middle mouse button is single clicked?
4. What shape is drawn when the right mouse button is single clicked?

4. Find and fix the bugs in a series of broken apps

4.1.

The code in this lesson you'll practice techniques for finding bugs

The 1st technique is to use `console.log` statements To find the mistakes in this code we put `console.log` messages on lines 5 & 7

When the code is run, the log statement on line 7 doesn't run when the mouse is clicked

1. Click and try the example app
2. Fix the mistake on lines 6-13
3. Fix the mistake on lines 15-18

4.2.

Click and try the example app to see how the code should work. This app paints the screen as the user clicks and drags the mouse. The paint brush changes color when they double click.

On line 12, the `brush.clone()` is used to make a copy of the circle attached to the cursor.

This code has 2 mistakes.

On line 10, `console.log(e)` prints out the value of all the properties in the e event object.

1. Study the `console.log` output to help fix the mistake on lines 9-13.
2. Use the reference to help find the mistake on lines 15-19.

4.3.

Click and try the example app to see how the code should work

The mouse event parameter doesn't have to be called **e** or **event** It can be called anything you like For example, the function on lines 6-8 calls the event parameter **x** It is best to call it **e** or **event** because that is easy to remember

This code has 2 more mistakes!

1. Fix the mistake on lines 10-13
2. Fix the mistake on lines 32-35. Hint use the code `console.log(e.event)`

4.4.

With this code the user double clicks and drags to draw a rectangle. This works **OK**!

The user should be able to triple click a shape to remove it. This is **BROKEN**!

The user should be able click and drag shapes to move them. This is also **BROKEN**!

On lines 26 & 27, the functions on lines 4 & 11 are attached to each shape's **onMouseDrag** and `onMouseDown` events The event functions attached to each shape can have any name For example, the function on line 4 could be `function x(e)` as long as it is attached using the code `item.onMouseDown = x`

1. Click and try the example app to see how the code should work
2. Fix the mistake on lines 3-8, so that triple click removes shapes
3. Fix the mistake on lines 41-50, so that click and drag moves a shape

4.5.

5. Make two more apps without any help

5.1. create a Rainbow Brush (part I)

This lesson tests your understanding of mouse event handlers Use the reference to help you complete the tasks

For your 1st task:

1. Draw a **red** circle with a radius of 10, positioned at -10, -10. Store it in a variable called `brush`
2. Make the circle follow the cursor when the mouse moves

5.2. create a Rainbow Brush (part II)

1. Make the brush follow the cursor when the mouse is dragged
2. Use the **clone** method to paint as the user drags the mouse
3. Increase the brush **hue** by 2 for every mouse drag event
4. Increase the brush **hue** by 5 for every mouse move event

5.3. create a Polygon App (part I)

1. Create a path object and store it in a variable called `path`
2. Set the `selected` property of the path to true
3. When a mouse button is pressed down add the current cursor location to the path

5.4. create a Polygon App (part II)

In the mouse down function do the following:

1. Add a point to the path only if the user **single clicks**
2. On double click, set the paths `closed` property to true, `selected` property to false, and give it a random fill color
3. Then set the **path** variable to a new **selected** path object

5.5.

Review Quiz Questions:

1. Which property is used in **onMouseDrag** to draw rainbow colored lines?
2. Which property is used in **onMouseDrag** to draw rectangles that are sized by **clicking** and **dragging**?
3. Which property is used in **onMouseDrag** to draw circles at the location of the mouse cursor?
4. Which property turns a path object into a polygon

6. Learn about frame events by bouncing a ball around the screen

6.1.

The next 5 lessons introduce animation with **Paper.js** using the `onFrame` event handler

The code on lines 4-6 define a special function called **onFrame** Paper.js runs this function **60** times per second

The code on lines 1-2 draws a ball in the top left corner Lines 4-6 make the ball move slowly to the right corner

1. Click to run each task
2. Make the ball start in the top right corner
3. Change the animation to move the ball from right to left

6.2.

The current code moves the ball 1 pixel every frame and 60 pixels per second

1. Animate the ball diagonally from the top right corner to the bottom left, moving 1 pixel down and 1 pixel left per frame
2. Draw a **blue** ball in the top left corner and store it in a variable called `ball2`. Make it the same size as the 1st ball
3. Animate `ball2` from the top left to the bottom right corner

6.3.

To make the ball bounce when it hits the ground we do 2 things:

1. On line 3, `ball.speed = 2` sets the speed property of the ball to 2
2. On line 10, `ball.position.y += ball.speed` moves the ball each frame, based on the balls speed
3. On line 12, `ball.position.y < 50` checks if the ball hits the ceiling and `ball.position.y > 150` checks if the ball hits the **brown** platform created on lines 5-7
4. If it hits the platform the direction of the ball is reversed using `ball.speed *= -1`

Modify the code to do the following:

1. Remove the brown platform
2. Change the speed of the ball to 10
3. Make the ball fall to the bottom of the canvas then bounce back to the top

6.4.

This code is for a ball that bounces around the screen

Lines 3-4 set the speed in the x and y directions

Lines 7-8 move the ball according to the x and y speed

Lines 10-13 reverse the speed in the y direction when the ball hits the top and bottom Lines 15-18 reverse the x speed when the ball hits the left and right walls

1. Change the x speed to 4
2. Change the y speed to 3
3. Change the radius of the ball to 20
4. Adjust line 11 so that the ball hits the walls correctly
5. Adjust line 16 so that the ball hits the walls correctly

6.5.

Review Quiz Questions:

1. What function is used to do animations?
2. How many times is the `onFrame` function run each second?
3. Which code is `true` just as the ball hits the left wall?
4. What code reverses the direction of the ball from up to down?

7. Make a timing game using animation and mouse events

7.1.

In this lesson you'll make a **Timing Game** Click and try the example app

The code on lines 2-4 draws a center line Lines 7-10 create a ball

The `onFrame` method on lines 13-15 moves the ball

1. Click then
2. Double the size of the ball by changing the value of the variable that sets its size
3. Halve the speed of the ball by changing the initial value of its speed property

7.2.

The code on lines 17-19 is supposed to reverse the speed of the ball when it hits a wall

1. On line 17, replace the 2 $$ with the correct symbols
2. On line 18, replace the $$ with the number that reverses the speed of the ball

7.3.

On line 1, `Game.setCursor('/image/game/crosshair.png', 32, 32)` changes the mouse cursor to a 64 by 64 pixel **crosshair** The numbers **32** set its **hot spot** to its center

On line 2, `var score = new Game.Score()` displays the players score in the top right corner `score.add(10)` adds 10 to score

The function on lines 26-36 give the user points when they click the ball Line 27 calculates the distance of the ball from the center line when it is clicked Line 28 gives 100 points if the ball is clicked right on the center and subtracts 1 point for every pixel away from the center If the ball is clicked 200 pixels away from the center, they loose 100 points

1. Complete line 14 to attach the `onMouseDownBall` function to the balls `onMouseDown` event
2. Change line 28 to give 50 points if the ball is clicked right on the center and subtract 1 for every pixel it is off by
3. Lines 31-32 increase the speed of the ball when it's moving left. Complete line 34 to increase the ball speed by 1 when it's moving right

7.4.

On line 2, `var timer = new Game.Timer(30)` creates a countdown timer. The timer has a method called `isRunning` which returns `true` if there is time remaining and false when time is up. Click to try the game.

1. Change the time limit to 15 seconds.
2. Add the code `if (!timer.isRunning()) return;` to the start of `onFrame` so that the ball stops when the time runs out. This code will exit the functions when time is up.
3. Add code to `onMouseDownBall` to stop the user scoring points when time runs out.
4. **(Optional)** Add a 2nd and 3rd ball.
5. **(Optional)** Instead of changing the speed, decrease the ball **radius** when balls are clicked.
6. **(Optional)** Change the ball color to **green** for a good score, **yellow** for an OK and **red** for bad.
7. **(Optional)** Use `timer.add(2)` to add extra time when the user gets a good score.

7.5.

Review Quiz Questions:

1. How many points does the user have at the start of the game?
2. How many pixels does the ball move each second?
3. How many points does the user get when they double click the ball?
4. What is the value of the ball's speed property when it is moving left?

8. Take down the satellites in your next home made game

8.1.

In this lesson you'll create the Code Avengers satellite shooting game

The code on lines 1-5 set the satellite image, background image, cursor image, and creates score and timer objects

Lines 7-9 create a button that runs the `setup` function when its `onClick` event is triggered

Use the reference to help add the following code to the `setup` function:

1. Use a **for** loop and the **clone** method to create 10 satellites
2. `Game.random()` gets a random number between 0 and 500. Use it to set a random x and y position for each satellite
3. Set the `visible` attribute to false to hide the button
4. Call `timer.start()` to start the timer

8.2.

To animate the satellites do the following:

1. Create a function called `onFrameItem`
2. Inside that function rotate the satellites 1 degree and move them 1 pixel to the right
3. Inside the setup function attach `onFrameItem` to each items `onFrame` event
4. Use the code `Game.random(360)` to get a random number from 0 to 360. Use it to randomly set the initial value of the `rotation` property for each item

8.3.

Do the following, so that players score points when they double click satellites:

1. Create a function called `onMouseDownItem`
2. Inside that function check if the mouse was double clicked
3. If it was, use item's `remove` method, then add 10 points to the **score**
4. Inside the `setup` function, link the `onMouseDownItem` function to the `onMouseDown` event of each item

8.4.

The timer has a method called `isRunning` which checks if the timer is running It also has an event called `onTimeout` which is triggered when time runs out

Complete the game by doing the following:

1. If the x position of an item is more than **530** set the x position to -30 and the y position to a random number between 0 and 500
2. Add the code `if (!timer.isRunning()) return;` as the 1st line of `onFrameItem` and `onMouseDownItem`
3. **(Optional)** Create a function called `gameover` that makes the button visible, and sets its `text` property to **Restart Game**
4. **(Optional)** Inside `setup` change `timer.start()` to `timer.restart()` to restart the timer when the button is clicked
5. **(Optional)** Put the code `timer.onTimeout = gameover` just after the `timer` is created to run the `gameover` function when time is up
6. **(Optional)** Give the items a random speed between 0.5 and 1
7. **(Optional)** Set the `scaling` property to have the same value as the speed
8. **(Optional)** Subtract 10 points for missing a satellite

8.5.

Review Quiz Questions:

1. Which code hides the button?
2. Which code needs to be put on line 3 to animate the ball
3. What properties should replace the $$ to get an item to move and rotate
4. Which code is `true` when item hits the bottom of the screen?

9. Fix four more broken animation based apps

9.1.

In this lesson you will fix the code in 4 simple apps

The code for this task is a **2 ball** version of the **timing** app It has 2 mistakes

1. Click and test the
2. Fix the small mistake on line 33 so that the balls move and bounce of the walls isRunning
3. Fix the mistake on lines 19-23 so that points are scored when the ball is clicked

9.2.

Usually the **event** parameter for the **onFrame** function is not used In this case, it can be left out as on lines 6 & 13

This animation code has 2 mistakes:

1. Click to try the
2. Fix the mistake in onFrameBall to make the balls bounce off the bottom of the screen
3. Fix the mistake in onFrame to make the balls go all the way to the right edge of the screen

9.3.

The code on lines 7-11 create a random polygon with 4 sides Line 8 adds a randomly positioned corner (or vertex) to the polygon Lines 9-10 set a random x and y speeds for the corner

The onFrame function moves each of the corners of the polygon according to their speeds Lines 20-27 are supposed to make the corners bounce when they hit the walls But there are 2 mistakes.

1. Click to try the
2. Fix the mistake on lines 20-22
3. Fix the mistake on lines 24-26
4. **(Optional)** Try increasing the number of sides that are drawn on the polygon
5. **(Optional)** Add the code `p.smooth()` to the bottom of the onFrame function

9.4.

This code is supposed to draw a windmill with a blade that spins when dragged with the mouse

1. Click to try the
2. Fix 2 mistakes on lines 9-14 that made the blade disappear
3. Fix the mistake on lines 18-21 that stops the blade from spinning

9.5.

Review Quiz Questions:

1. This animation is supposed to move a blue ball across the screen. Which line has a mistake?
2. This animation is supposed to rotate a square in the middle of the screen. Which line has the mistake?
3. This animation is supposed to rotate an ellipse in the middle of the screen. Which line has the mistake?
4. This animation is supposed to draw a circle that grows to fill the entire screen. Which line has the mistake?

10. Get artistic as you create another pair of animation apps

10.1.

This lesson tests your understanding of animations Use the reference to help you complete the tasks

1. Click to see what the canvas does
2. Create an `onFrame` function
3. Inside that function create a circle with a radius of 20 and a random x and y position
4. Give the circle a random fill color

10.2.

For this task you will make the balls move diagonally

1. Click to see what the canvas does
2. Create a function called `moveBall`
3. Inside the onFrame function, set `moveBall` as the on frame function for each ball created
4. Inside the onFrame function, set the balls `speed` property to a random speed between -5 and 5
5. Inside moveBall use the ball's `speed` property to move the ball that number pixels in both the x and y direction
6. **(Optional)** Reverse the balls speed when it hits one of the walls

10.3.

Create a comet trail animation by doing the following:

1. Click to see what the canvas does
2. Draw an **orange** circle with a radius of 25 pixels every time the mouse dragged event is triggered
3. Attach a function called `shrinkCircle` to the `onFrame` event of every circle created when the mouse is dragged
4. Create a function called `shrinkCircle` that decreases the radius of the circles by half a pixel each frame
5. At the end of `shrinkCircle` remove the circle when its radius is less than 1

10.4.

To make the effect more interesting do the following:

1. On line 2, create a variable called `brushColor`
2. When the mouse is clicked, set the `brushColor` to a random color with a `lightness` value of **0.3**
3. In `onMouseDrag` use `brushColor.clone()` to set the fill color of the circles
4. Inside `shrinkCircle`, use the code `this.fillColor.lightness *= 1.02` to make the comet tail fade as it shrinks

After you complete the task, try the following:

1. **(Optional)** Make the circles get darker as they shrink
2. **(Optional)** Use `Game.setBackgroundColor` to change the background color to **black**
3. **(Optional)** Make the circles become semi-transparent as they shrink

10.5.

Review Quiz Questions:

1. Given there are 60 frames per second, what is the diameter of the ball after 1 second?
2. Which direction does the ball move?
3. This ball bounces off 3 of the walls, but goes straight through the other wall. Which wall is missing?
4. What is the **maximum** distance this ball moves in 1 second?

11. Basic text editor

11.1.

Now you'll learn to create apps that require keyboard input

The `onKeyDown` function is run every time a key is pressed down It also runs when the key is held down for a period of time The `onKeyUp` function runs when a key is released

The code on lines 3-8 displays the number of times the `onKeyDown` event is triggered

1. Click and see what happens when you press and hold keys
2. Create a variable called `keyUpCount` to count the number of times the `onKeyUp` event is triggered
3. Add an `onKeyUp` function
4. In that function display a message that says **onKeyUp 1** when the 1st key is released, **onKeyUp 2** for the 2nd etc. Use the existing message box to display these messages.

11.2.

The **KeyEvent** parameter for Key Events contains `key` and `character` properties The `KeyEvent.key` is the key that caused the event If you press the a key this property contains the value **a** If you press `shift` it contains the value **shift**

and `KeyEvent.character` contains the character that would appear if you typed it in a text box If you press a special key like `shift` this property will be empty If you press `shift+A` it will contain the value **A**

1. the code an see what the values of `character` and `key` are when you type uppercase versus lowercase. See what happens when you press the arrow keys.
2. Use `Game.setBackgroundColor` to make the background **blue** when the user presses a special key like `alt`, `ctrl` or the arrow keys
3. Use `Game.setBackgroundColor('')` to set the background back to transparent when the key is released

11.3.

This code creates a basic text editor

The code on lines 2-4 creates a `PointText` object called `text` On line 12, `text.content += e.character` adds the character to the text editor The code on lines 9-10 checks if the `tab` key was pressed and inserts 4 spaces

Add code to do the following:

1. Clear all of the text when **escape** is pressed down
2. Delete the last character when **backspace** is pressed down
3. **(Optional)** Automatically move to the next line when a character is added that goes within 20 pixels of the right edge of the screen. (Hint: use `text.bounds`)

11.4.

The Key Event parameter has another property called `modifiers`, which indicates which special keys (`control`, `option`, `command`, `shift`, `capsLock`, `space`) are pressed down

For example, on line 30, `e.modifiers.option` is true when the **alt** (on Windows) or **option** (on Mac) keys are pressed

The code on lines 32-38 increases the left and right margin when the right arrow is pressed while the `option` key is down The left arrow decreases the margin when `option` is down

Note: `text.point` is the bottom left corner of the text

1. Increase the top and bottom margin when the up arrow is pressed
2. Decrease the top and bottom margins when down arrow is pressed
3. **(Optional)** Change the font size when `up` and `down` are pressed at the same time as `control`

11.5.

Review Quiz Questions:

1. What is the value of `event.character` when the `shift` key is pressed?
2. What Key Event modifiers property is set to true when the **alt** key is pressed on a windows keyboard?
3. What is the value of `event.character` when `shift` then `z` are pressed?
4. What is the value of `event.key` when `shift` then `a` are pressed?

12. Toggle switch

12.1.

This lesson covers the important skill of how to **toggle** a variable value

This code draws a **red** circle on lines 1-2, which is hidden and reshown when the canvas is clicked.

The 3 `toggleVisible` functions do the same thing—they toggle the visibility of an item, i.e. if the item is visible it is hidden, and if its hidden it is made visible again

`toggleVisibleA` hides the item if `item.visible == true` otherwise it shows the item with `item.visible = true`

`toggleVisibleB` uses a shorthand for checking if the item is visible It uses `if (item.visible)`, which checks if `item.visible` is set to any value other than `false`, `null`, `undefined`, `0` or an empty string

1. Create a `Game.Text` item in the middle of the screen that says **STOP**, and store it in a variable called `message`. Make sure the text appears on top of the circle
2. Set its font size to 64
3. Toggle the item's visibility when the screen is clicked

12.2.

`toggleVisibleC` uses a shorthand to toggle a **boolean** variable value: `item.visible = !item.visible`

This function toggles the variable value in 1 line of code, instead of the 5 required in `toggleVisibleB`

The Mouse Event parameter has a property called `item` that contains the item the mouse is currently pointing to or `null` if the mouse is not over any item

The code `if (e.item)` checks if the mouse is pointing at an item; `if (!e.item)` is the same as `if (e.item == null)`, which checks if the mouse is pointing at the background

1. Modify the `onMouseDown` function to toggle the visibility of both items if the mouse clicks the background
2. Remove `toggleVisibleB` and `toggleVisibleA` since `toggleVisibleC` is the most concise. Rename `toggleVisibleC` to `toggleVisible` and update the rest of the code to match

12.3.

Click and try the example When you click the canvas the circle toggles between **red** and **green**, and the message toggles between **STOP** and **GO**

The function on lines 12-18 toggles the color of the circle `if (light.fillColor == new Color('red'))` checks if the current fill color is equal to **red**

Note that `if (light.fillColor == 'red')` is always `false`, because when you do `light.fillColor = 'red'` the fill color is set equal to a **red** `Color` object Check the value printed in the console when you click the canvas

1. Add a function called `toggleText` that toggles the `message` between **STOP** and **GO**
2. Use that function to toggle the message when the canvas is clicked

12.4.

When you use `Game.setBackgroundColor` you get the same value back when you call `Game.getBackgroundColor` Line 2 in the code will print the word **yellow**

Now do the following:

1. Create a function called `toggleBackground` that switches the background color between **yellow** and **black**
2. Toggle the background color when the background is clicked
3. Toggle the circle and text when they are clicked

12.5.

Review Quiz Questions:

1. Which line of code correctly **toggles** the value of the item's `selected` property?
2. Which if statement condition is true when `item` is visible?
3. Which of the following `if` conditions is always `false`?
4. Which statement is `true` if the value of `xxx` is `null`, `undefined`, `false`, `0` or empty string

13. Typing tutor

13.1.

In this lesson you'll create a typing tutor

For your first task do the following:

1. Create the function that is run when a key is pressed down
2. Remove the 1st character from the text when the user types it

13.2.

Click and try the example app Notice that the background color changes when you make a mistake, and complete the sentence

1. Change the background color to **pink** if they type an incorrect character. Be careful not to change the color for `shift`
2. Use `Game.setBackgroundColor('')` to make the background transparent again when the user types a correct character
3. Change the background to **lightGreen** when the sentence is complete

13.3.

Click and try the example app Notice that it displays the users accuracy as they type

1. Store a `Game.Status` object in a variable called `status`. Set the initial text to **Accuracy: 100%**
2. Create variables called `keyCount` and `correctCount` with intial values of **0**
3. Increase `correctCount` when a correct key is pressed
4. Increase `keyCount` when any key is pressed (not including **modifiers**)
5. Display the accuracy rounded to the nearest %

13.4.

Click and try the example app Notice that the typing speed is displayed in words per minute, along with a timer that starts at 60 seconds

1. Store a `Game.Timer` object in a variable called `timer` with an initial time of 60 seconds
2. Start the timer when a key is pressed
3. Use `timer.getTimeElapsed()` to get the number of seconds since the timer started, and calculate the speed in words per minute. Use `correctCount / 5` to estimate the number of words the user types. This assumes the 5 keystrokes (including space and punctuation) are required per word
4. Stop the timer when the user finishes typing the sentence

13.5.

Click and try the example app Notice the score in the top right counts the number of correct characters the user types

1. Add the score to the status
2. Stop accepting input if the time has run out
3. Use `timer.onTimeout` to make the background **red** when time runs out
4. Change the timer length to **5** seconds to make it easy to test
5. **(Optional)** Create an array of sentences and randomly select one for the user

14. Traffic light

14.1.

This **code** creates a traffic light that changes from **red** to **green** to **orange** then back to **red**

1. Change the code to go from **red** to **yellow** to **green** to **orange** to **red**
2. Add a `Game.Text` in the center of the screen with font size **64**. Store it in a variable called `message`.
3. Make the text change from **STOP** to **READY** to **GO** to **SLOW** then back to **STOP** when the light color changes

14.2.

This code uses an array to cycle through the traffic light colors This approach makes it easier to add new colors or change the order

On line 9, `var COLORS = ['red', 'green', 'orange']` creates an array that stores the order of the colors On line 10, `var currentIndex = 0` creates a variable that tracks the index of the color in the array that is currently shown

Lines 13-17 change the light to the next color when the canvas is clicked Lines 14-15 move `currentIndex` back to 0 when it reaches the end of the array

1. Change the code to go from **red** to **yellow** to **green** to **orange** to **red**
2. Uncomment lines 6-7 to add the message
3. Create an array called `TEXT` that stores the messages **STOP**, **READY**, **GO** and **SLOW**
4. Use the array to change the values of the text as the light color changes

14.3.

This code cycles backwards through the values in the arrays On line 14, `currentIndex--` moves to the previous index and lines 15-17 reset `currentIndex` to the end of the array when it is less than 0

1. Add an `onKeyDown` function to handle key presses
2. Make the `right` arrow move forward through array values
3. Make the `left` arrow move backwards through array values

14.4.

This code on lines 13-21 uses **modulus** to cycle through the array values On line 15, `currentIndex = (currentIndex + 1) % COLORS.length` adds 1 to the index but returns to 0 when `currentIndex + 1` is equal to the length of the array

This line is a shortcut for the code on lines 25-28

The code on line 17 is a shortcut for the code on lines 30-33

Use the following initial variable values to answer the questions below:

```
var x = 1, y = 5, z = 3;
```

For each question x starts with a value of **1**

1. What is x when `x = (x + 2) % 4` runs **2** times:
2. What is x when `x = (x + 2) % y` runs **2** times:
3. What is x when `x = (x + 1) % z` runs **4** times:
4. Click when you are done

14.5.

Review Quiz Questions:

1. Which code is the index for the last item in an array called `items`?
2. What code is needed on line 6, in order to correctly cycle through the array values as the mouse is clicked?
3. What code is needed on line 5, in order to correctly cycle through the array values as the mouse is clicked?
4. What value is displayed when the mouse is clicked the 5th time?

15. Digital whiteboard

15.1.

In this lesson you will create an interactive whiteboard app that combines key and mouse events

Click and try the example When you type a character it appears next to the cursor

1. Use an `onMouseMove` function to store the current value of the mouse position in the `cursorPosition` variable
2. When the user types a visible character, create a `PointText` item at the current cursor position. Store the new text item in the `currentText` variable
3. Set the default font size to 24

15.2.

To enable users to type words and sentences you need to store the `currentText`

Click and try the example When you type the text is added to the current text item When you move the mouse and type again, a new text item is created

1. When a character is typed, add it to the current text or create new text if `currentText` is null
2. Remove a character from the current text when backspace is pressed
3. Set `currentText` to `null` when the mouse moves

15.3.

Click and try the example When the mouse is moved over an item the text is surrounded with a **lightBlue** stroke The `delete` key removes text

The `onMouseMove` event parameter has a property called `item` that contains the item the mouse is pointing to

1. Use `currentText = e.item` to set `currentText` to the item the mouse points to
2. Create two functions called `onEnterItem` and `onLeaveItem`
3. Attach them to the `onMouseEnter` and `onMouseLeave` events of every text item that is created
4. In `onEnterItem` set the stroke color to **lightBlue** and stroke width to **1**
5. In `onLeaveItem` hide the stroke
6. Hide the **lightBlue** stroke when a character is added to the current text
7. **(Optional)** Remove text using the `delete` key

15.4.

Click and try the example Press the arrow keys to change the color of the word **PEN** in the top left corner

1. Create a **black** size **24** `PointText` with the text **PEN** in the top left corner and store it in a variable called `pen`
2. Create an array called `COLORS` that contains the strings **black**, **blue**, **red**, **green**
3. Create a variable called `currentColor` and set it to 0
4. Cycle through the pen colors when `left` and `right` are pressed

15.5.

Click and try the example Press the arrow keys to change pen color Triple click to clear the screen

Complete your app by doing the following:

1. Write new text with the current pen color
2. Clear the canvas when the user triple clicks the mouse. Use the `onMouseDown` function.
3. **(Optional)** Change the brightness of the pen with `up` and `down` arrows; the hue with `alt+left` or `alt+right`; the saturation with `alt+up` or `alt+down`
4. **(Optional)** Change the font size with `ctrl+up` or `ctrl+down` and **font family** if the user presses `ctrl+left` and `ctrl+right`
5. **(Experts)** When the arrow keys are pressed while hovering over an item, change its style
6. **(Experts)** Use the `visible` property to only show the `pen` when an arrow key is pressed

16. Spaceship: moving objects smoothly with Key.isDown

16.1.

In text-based games, using `onKeyDown` is fine This lesson introduces a better approach for 2D or 3D games

To begin with let's use the `onKeyDown` approach:

1. Make the ship move 10 pixels forward when the up arrow is pressed
2. Move 5 pixels backwards when down arrow is pressed
3. Move 5 pixels when left is pressed
4. Move 5 pixels when right is pressed

16.2.

Now make the ship rotate as it moves left and right using the `rotation` property

1. Click and try the example
2. Make the robot rotate 20 degrees to the left when it turns left...
3. And 20 degrees to the right when it turns right
4. Return the ship to its regular orientation when it moves forward and backward

16.3.

Did you notice that the movement is not very smooth? Also, the ship could not move diagonally.

To make the movement smooth we check for key presses in the `onFrame` function, using `Key.isDown`

Every frame (~60 times per second) the code on lines 6-16 checks if the left or right keys are down and moves the ship 1 pixel each frame If left and right are pressed at the same time, the ship does not move

Click and try the example. Notice that it is much smoother than before Also, notice that the ship can move diagonal when 2 keys are pressed at the same time

To do this do the following:

1. When **down** is pressed, move the ship backwards at the same speed as it moves left and right
2. When **up** is pressed, move up at **twice** the speed as it moves backwards
3. Change the ship speed to **1.5** pixels per frame

16.4.

If the ship moves diagonally **1.5**px up and **1.5**px right, it is actually moving a total of **2.25**px per frame This distance needs to be adjusted so that that total distance moved is 1.5px

This is done using the `point.length` property

On line 25, `var delta = new Point(deltaX, deltaY)` creates a **vector** (or point object) that sets the direction the ship moves

The code `delta.length = 2` adjusts the vector length to 2 On line 27, `delta.length = distance` adjusts the length of vector to the ship's speed

On line 26, `if (delta.length)` is a shortcut for `if (delta.length != 0)`

1. Add 1 line of code to point the ship left 20 degrees when the left arrow is pressed...
2. And right 20 degrees for the right arrow
3. Point up when neither left or right are pressed

16.5.

Now for a challenge! You need to animate the ship so that it **turns** smoothly

Click and try the example When the left arrow is pressed the ship turns towards its desired direction **1** pixel at a time

To do this replace line 33 to do the following:

1. Animate the ship left when the left arrow is pressed
2. Animate the ship right when the right arrow is pressed
3. Animate back to the center when neither left or right is pressed

17. Rotate the hands on a clock

17.1.

In this lesson you'll practice changing the length of vectors

The code on lines 1-3 draws a line from the top left corner to the middle of the screen

On line 5, `var point = path.lastSegment.point` gets the end point of the line

The code on lines 8-10 increases the length of the end point when the up key is pressed Lines 12-14 change the angle of the end point, which causes the line to rotate

1. Make the line get shorter when the down key is pressed
2. Rotate the line in a clockwise direction when the right key is pressed
3. Rotate the line in a counterclockwise direction when the left key is pressed

17.2.

This code draws a clock with a **black** line for the minute hand The code on lines 16-20, rotates that line when you press the left and right keys

By default an item rotates about its center This can be changed by setting the `pivot` property

1. Click and try the code
2. Try the example in the reference to see how `pivot` works
3. Add code on line 6 to make the line rotate around the center of the clock

17.3.

Now make the following adjustments:

1. Adjust the minute hand to point to the 12 o'clock position at the start
2. Create an hour hand with a length of **100**px and width of **8**px that points to 12 o'clock. Store it in a variable called `hour`

In the next task you'll rotate the hour hand

17.4.

Now rotate the hour hand by doing the following:

1. Create a constant called `MINUTE_SPEED` and set it to **2** so that it rotates at 2 degrees per frame when the keys are pressed
2. Create a constant called `HOUR_SPEED` and set it to the correct speed
3. Use the constants to rotate the hour and minute hands at the correct speeds when the keys are pressed
4. **(Experts)** Add the numbers 1 to 12 using a for loop and vectors to set the position of the numbers

17.5.

You may want to use a pen and paper for this task!

Review Quiz Questions:

1.
2.
3.
4.

18. Do collision detection using HitTest

18.1.

Let's return to our space ship and add obstacles for the ship to avoid

1. Create a raster object with the image /image/game/bomb.png and store it in avariable called `bomb`. Set the initial position to **250**, **-40**.
2. Animate the bomb so it falls down at **2**px per frame

18.2.

The next step is to detect when the ship hits the bomb This is done using the `hitTest` command

On line 34, `player.hitTest(bomb.position)` is used to test if the center of the bomb is inside the bounds of the ship image If there is a **hit** the `hitTest` command returns a `HitResult` object that contains information such as where the ship was hit

On lines 35, `if (result)` checks that their was a hit, and line 36 displays the HitReset object in the console

1. Click and test the code
2. Try to work out what the problem is with the current approach
3. Click when you think you have discovered the problem

18.3.

The problem is that the ship image is a rectangle and is **hit** even when the bomb hits the transparent parts of the image

You can fix this by using the `color` property of the `HitResult` This property contains the color of the pixel at the position that the image that was hit

Color objects have 4 properties: `red`, `green`, `blue` and `alpha` Alpha specifies the opacity of the color An alpha value of 0 means that color is transparent

1. Use `HitResult.color` to test when the bomb actually hits the ship
2. Remove the bomb and replace the ship image with /image/game/explosion1.png when they collide
3. Don't let the ship move once it explodes

18.4.

This code drops a bomb from the top of the screen Everytime the bomb moves, it checks it has hit the ship

Add the following code:

1. When the bomb goes off the bottom of the screen remove it (do this step first!)...
2. Then create a new bomb with a random **x** position between **0** and **500** and a **y** position of **-40**
3. Increase the bomb speed by 0.5

18.5.

For your final task add scoring.

1. Add a `Game.Score` object and call it `score`
2. Add 1 to the score every time the ship avoids a bomb
3. **(Optional)** After you complete the task, try adding multiple bombs at once

In a later lesson you'll make the robot shoot bullets

19. Etch A Sketch: adding vectors

19.1.

Vectors are very similar to points: they both have x and y coordinates Points specify a position on the canvas Vectors specify distance and direction between 2 points In PaperJS both are represented by `Point` objects

Click The code on lines 3-5 creates 3 vectors: `v1`, `v2` and `v3` The code on lines 7-17 visually display the size and direction of the vectors

Vectors can be added to points to move them

Lines 19-20 draw a dot in the top left corner

Lines 23-24 move a copy of the dot by adding `v1` (the **red** vector) This moves the dot to the end of the red line

Lines 26-28 add the vector represented by the **green** line

1. On line 32, replace v with the correct vector to move a copy of the dot to the next largest dot
2. On line 36, replace v2 to move a copy of the dot to the largest dot

If you want a detailed introduction to vectors try this tutorial at paperjs.org

19.2.

When a vector is multplied by 2 its length is doubled and its direction is unchanged

On line 24, `c.position += v3 / 2` adds a vector half the distance of the **blue** line

1. On line 28, replace v to draw the next largest dot
2. On line 32, replace v to draw the next largest dot
3. On line 36, replace v to draw the largest dot

19.3.

Now let's create an app for the classic toy: "Etch A Sketch"

Click and try the Use the arrows to draw lines

The code on lines 1-25 draws the frame of the Etch A Sketch Lines 31-35 create a path object for the drawing Every time keys are pressed a point is added to the path

The `onFrame` method creates a vector called `delta` on line 39, which stores the direction and distance to move before adding the next point

Lines 41-43 check if the left key is pressed and add a point 1 pixel to the left

1. Add code to handle the right key
2. Add code for the up key
3. Add code for the down key

19.4.

Now add code to stop the app from drawing outside the screen

1. Add code that stops lines going outside edges of the Etch A Sketch screen
2. Clear the screen when the user presses the **escape** key. Use the `removeSegments` method to clear the line

19.5.

Review Quiz Questions:

1.
2.
3.
4.

20. Mini golf: practice vector geometry

20.1.

Let's do some more vector Math by building a mini golf game

The code in the editor draws the layout of the hole Click and try out the Click the mouse to move the golf ball

The code on lines 36-39 calculates a vector for the **velocity** (speed and direction) of the ball `velocity = e.point - ball.position` creates a vector with correct angle and `velocity.length = 1` sets the speed the ball travels to 1px per frame

1. Add an `onFrame` method that adds the velocity vector to the ball position
2. Adjust the speed of the ball to **2**px per frame
3. When the mouse is clicked, hide the `message` in the middle of the screen...
4. And add **1** to the stroke score
5. **(Optional)** Make the ball bounce off the top, bottom, left and right edges of the canvas. In the next task you'll make it bounce off the concrete walls

20.2.

The border variable defines the concrete boundary Now let's make the ball bounce off the walls using the border's `hitTest` command

On line 45, `var newPosition = ball.position + velocity` calculates the position the ball should move to On line 47, `if (border.hitTest(newPosition))` checks if the ball hits a wall

The code on lines 48-53 flips the **x** velocity if the ball hits a vertical wall

1. Modify the code on line 52 so that the ball looses 30% of its energy each time it hits the wall
2. Add code to make the ball bounce off horizontal walls

20.3.

Now let's add friction to the ball so that it gradually slows down as it rolls

The following code will reduce the length of a vector by 10%:
`vector.length *= .9`

1. Click and try the
2. Add code to reduce the speed of the ball by **0.8%** each frame
3. Stop the ball completely when its speed falls below **0.3**
4. Make the ball lose **5%** of its energy when it hits the walls
5. Change the length of the initial velocity to be **5%** of the distance between the ball and the position the mouse is clicked

20.4.

In order for the player to get the ball in the hole it must hit the hole with a speed of less than **2px** per frame If it is going too fast, the ball rolls over the hole

1. Click and try the
2. Check if the the ball hits the hole at less than **2px** per frame
3. If it does, set its speed to **0**...
4. And move the ball to the center of the hole...
5. And display a message that says: "Hole complete!"
6. **(Optional)** Change the message to "Hole in One!" if they get it with 1 stroke. Hint: use `score.get()`

20.5.

To complete the game do the following:

1. When the ball goes in the hole, set the `complete` variable to `true` to inicate the hole is complete
2. Use the `complete` variable to make the game reset when the mouse is clicked and the ball is in the hole. Use `score.reset()` to reset the score
3. **(Optional)** After you complete the task, try creating your own hole layout.
4. **(Experts)** Add additional obstacles to the hole

21. Intro to Classes

21.1.

The next few lessons introduce the basics of **Classes** and **Object Oriented Programming** (OOP) First, let's take a look at **why** classes are important

In the level 1 course you created programs with lists of commands, **if statements** and **loops** When you started, this may have seemed complex, but probably seems quite simple now!

For example remember programs like this?

1. Click to run the code and see if you can remember the answers to the questions. (No peeking at the answers!!)
2. Try adding a question no.6.
3. Make sure you update all the places that need to be updated!

21.2.

As programs get larger and more complicated it is important to divide the code into **functions**: named reusable sections of code that do a specific task and may return a value

Using functions makes it easier to read, write and modify larger programs

For example, as shown in the editor the code from the previous task is greatly simplified using functions

1. See how easy it is to add another question!
2. Don't forget to update the total questions

21.3.

As programs grow in size they get more and more complex Keeping track of 1000+ functions and variables is difficult That is where **classes** come in

Just as **functions** group variables and statements together into a reusable sections, **classes** group functions and variables together Using classes makes it easier to create and maintain complex programs

So what is a class and how do you create and use them? Actually you've been using them all along!

The intro to game development project used 6 classes that we created: Player, GoodItem, BadItem, Enemy, Obstacle and Scoreboard 250 lines of code is used to create the 6 classes But you don't have to understand how that code works to use the classes

You simple write code like `new Game.GoodItem(225, 225)` to create a `GoodItem` object

When working in teams, classes hide complex parts of a program from those who only need to use the classes, but not understand what happens in the background

In the next few lessons you'll learn how to create your own classes

After all that reading you deserve a freebee. Click check to complete the task.

21.4.

In JavaScript classes are created by putting functions, inside other functions

Before you start creating your own classes, let's introduce a 2 important terms: **instance** and **method**

When you create an object using the `new` keyword we say you are creating an **instance** of a **class**

For example, on line 1 `var scoreboard = new Game.Score()` creates an **instance** of the `Game.Score` **class** and stores it in a variable called `scoreboard`

Functions that are attached to classes are called **methods**

For example, on line 2, `scoreboard.add(5)` uses the `add` method for the `Game.Scoreboard` class to set the initial score to 5 points

1. Click to try the
2. Create an instance of the `Game.Button` class with the text `+` and store it in a variable called `addButton`
3. Make the button use the `add` method in the `Game.Scoreboard` class to add 1 point to the score each time it is clicked
4. Create an instance of the `Game.Button` class with the text `-` and store it in a variable called `minusButton`
5. Make that button subtract a point each time it is clicked

21.5.

Review Quiz Questions:

1. Which keyword is used to created an object from a class?
2. Classes are created by grouping functions and variables inside a:
3. Which is **NOT** a correct reason for using classes?
4. What is the name of an object created using the `new` keyword?

22. Tank Duel I: Create a tank class

22.1.

In JavaScript there are several ways to create Classes

Lines 1-4 define the Tank class, which creates and returns a `Raster` object

On line 6, `var tank = new Tank()` creates an instance of the `Tank` class Since our tank object is based on a `Raster` it has all the raster properties and methods Line 7 moves the tank to the top right corner by setting the position of the rastter

1. On line 9, create a 2nd tank object
2. Move the 2nd tank to the bottom left corner

22.2.

The `Tank` function is called a class `constructor` It is a function that creates a Tank object

The constructor function can have 0 or more parameters

On line 1, `function Tank(no)` defines a class constructor with 1 parameter that sets the player number If player number is 1, it uses the image [/image/game/tank1.png]() and if the player number is 2 it uses [/image/game/tank2.png]()

Move lines 7 & 10 inside the function:

1. Make player 1 always start in the top right corner and face left...
2. And player 2 start in the bottom left corner and face right

22.3.

Now we have added a **variable** and **method** to our class Our class constructor has 3 parts:

Lines 3-10 create the raster obejct and set the starting position and rotation

On line 13, `var speed = 2` creates a variable that can only be accessed inside the `Tank` class

Lines 16-22 create a method that moves the robot

The code on lines 32-44 calls `tank1.move` when the left and right keys are pressed

1. Inside the Tank class, complete the `move` method so that `tank1` moves left and right with the direction specied by the `vector` parameter and the distance specified by the `speed` variable
2. Inside onFrame add code to move `tank1` left, right, up, down, and diagonally

22.4.

There are actually 2 ways to define **functions** in JavaScript This code gives examples of both ways

Lines 16-23 use the way we have used so far in this course

The 2nd approach is to do the following:

```
var coolFunction = function() {
  //do stuff
};
```

This code creates an **anonymous** function (one without a name) and stores it in a variable You can then use `coolFunction()` to call the function

This 2nd approach is used on lines 25-28 to define a method that teleports the tank to a random position on the screen

1. Change the `move` method to use the 2nd more compact approach. Notice that with the 2nd approach a `;` is put after the closing `}`
2. Use the `onKeyDown` method to teleport `tank1` when the `backspace` key is pressed. Don't do it in `onFrame` otherwise the tank will flash all over the screen if you press the backspace key for a normal length of time

22.5.

Now add code to make the 2nd tank move

One way to do this is to move the `onFrame` event inside the Tank class and attach it to the Raster

1. Move `onFrame` inside the Tank class and attach to the Raster object for the tank
2. Create variables `upKey`, `downKey`, `leftKey`, `rightKey` that store the correct keys for both player 1 and player 2
3. Make the 2nd tank move when the keys `WASD` are pressed.

23. Task Duel II: Using the debugger

23.1.

As the code in a program gets longer it gets more difficult to find mistakes in your code This lesson helps you practice finding bugs in classes

This code has several mistakes that stop the tanks from moving

To find the bug we have added `console.log` statements on lines 41 & 44 The log statement on lines 41 ensures that the `onFrame` method is attached The code on line 44 checks if the method is ever being run

1. Fix the mistake that is stopping `onFrame` from running
2. Fix the mistakes inside the `onFrame` method that stop the tank from moving

23.2.

This code adds a `Bullet` class that takes the position and direction of the tank as parameters The bullet travels at 5 pixels per second

Every time the players press `tab` or `enter` the tank calls its `shoot` method which creates an instance of the bullet class

This code has several mistakes that should be easy to find The 1st mistake is on line 64, where the variable `shootKey` is mispelled The computer knows that `shotKey` does not exist and prints an error message that includes the line number

Errors caused by mispelled variable or method names and missing symbols are called **syntax errors**; these are usually easy to find and fix

1. Fix the 1st mistake
2. Fix the 2nd mistake
3. Fix the 3rd mistake so that the tank shoots bullets when the enter key is pressed

23.3.

Some mispellings are more difficult to find Using log statements can help narrow down the line that has the error

The `Bullet` class has 2 errors

1. Find the error inside the Bullet's `onFrame`
2. Find the error in the Bullet's variable declaration section (where the variables are defined)

23.4.

On line 28, `result = project.activeLayer.hitTest(this.position)` checks if the bullet position hits any item on the screen

On line 31, `if (result && result.item != this)` checks that the `hitResult` was an item other than the bullet itself Line 33, checks that the bullet hit a non-transparent pixel

Another good way to find errors is using a **debugger** statement as shown on line 1 This pauses the code on that line and allow you step through your code 1 line at a time using the web browsers **debugging tools**

Click the following links for help on the debugging tools in your browser: [Chrome]() | [IE11]() | [Firefox]()

1. Press F12 to show the developer tools for your web browser
2. Click to run your code and it should pause on line 1. Notice that PaperJS modifies some of your code before it runs, but you should still be able to understand it
3. Click the arrows in the developer tools to experiment with stepping through your code
4. Try click a line number on the left edge of the developer tools. This adds a **break point**. The computer pauses when it hits that line
5. Hover over variables in the code and their values are displayed

23.5.

This code has a little mistake

1. Press F12 to open the developer tools
2. the code
3. Add a breakpoint at line 101, then click the play button in the developer tools
4. Step through the code to determine the error
5. Fix the error

24. Banking & scoring: understanding classes

24.1.

The function that creates a class object is called a class **constructor**

As shown in this example, the class constructor doesn't have to have a **return** statement In this case the computer does 3 things when it runs the code `var accountA = new Account(0)`

1. It creates an empty object
2. It runs the `Account` constructor function with `this` set to the new empty object
3. when the function ends, it automatically returns the new object and stores that object in the `accountA` variable

24.2.

This example adds a function for earning and paying interest

Note that their is no way to directly access the `balance` variable from outside the `Account` class This type of variable is called a **private variable**

It is good practice to make the data inside your class private, that is, only accessible by the methods inside that class Doing this makes it easier to modify your class in the future

24.3.

This class keeps a players score in a tennis match.

Notice that a semi-colon is put at the end when annonymous functions are stored in variables, e.g.:

```
this.setPoints = function(p) {
  points = 0;
};
```

When you create a **named** function you do not put a semi-colon after it, e.g.:

```
function xyz() {
  alert('Hello!');
}
```

As with other places the semi-colons are almost always optional, but the style above is standard

24.4.

This task adds a tennis match class that contains 2 player objects

Notice that Class constructor parameters can be used inside class methods For example, the `Player` constructor has a parameter `name` that is returned from the `getName` method. It is used in the same way as the **private variables** `points` and `games`.

24.5.

Review Quiz Questions:

1. Which line is the start of a constructor?
2. Which contains an anonymous function with parameters?
3. Which line creates an instance of a class?
4. Which line defines a private variable?

25. Create classes for a digital scoreboard

25.1.

In this lesson you'll make a controller for an LED sports scoreboard

Task 1 is to create a **seven segment display** (SSD for short) class

The code on lines 1-19 defines the `SSD` class and line 21 creates an instance of the class

The code on lines 6-12 creates rectangles for 6 of the 7 segments The variables `l` and `w` contain the length and width of each segment This makes it easy to change the size of the numbers on the scoreboard

1. Click and check the
2. Complete the **for loop** on lines 16-18 to set the color of each segment to **red**
3. Add the 7th segment
4. Construct a 2nd `SSD` object with a **15px** gap between the digits

25.2.

We have added a method that sets the number displayed by the `SSD` class The method currently works for the numbers 0 to 4

The code uses a loop on lines 23-25 to turn on all 7 segments It then turns off the correct segments for each number

The code `ss[4].visible = ss[5].visible = false` turns off the 2 segments in a single line It is a shortcut for:
`ss[4].visible = false;`
`ss[5].visible = false;`

The last 10 lines of code create and display 10 `SSD` objects and call the set methods to set them to the numbers 0-9

1. the to see what the 10 digits look like
2. Complete the set method to work for the number 5,
3. number 6,
4. number 7,
5. number 9

25.3.

Now use the `SSD` class to make a scoreboard

1. the to see what happens when you click the plus and minus buttons
2. Inside the `SSD` class use the code `this.set(0)` to set the initial number displayed to 0
3. Create a method called `add` that has a parameter `n`, which is added to the number currently displayed. Make the number go back to 0 after it reaches 9 (Hint use **modulus** `%`)
4. Add `onMouseDown` event handlers to the buttons that add 1 and minus 1 from the score

@TODO: Add modulus to the reference material to replace the existing text gloss

25.4.

Now create another class called `Score` that can display 2 digit numbers from 0 to 99

1. Create a class called `Score` that has a parameter `origin` that sets the position of the top left corner of the score
2. Inside the class create 2 `SSD` instances and store them in variables called `digit1` (the one the left) and `digit2` (the one on the right)

25.5.

Complete the `Score` class by adding the following methods:

1. The `set` method sets the score that is displayed
2. The `add` method adds and subtracts the number of points specified by a parameter
3. Make the score go back to 0 when it reaches 99
4. Don't subtract points if the score is 0