Computer Science

Year 12 into year 13 summer independent work

Deadline – first lesson in September

Part 1 – Website Development

HTML, CSS, JavaScript Programming Task

As there will be a section of coding on your exam that relates to HTML, CSS and JavaScript an exercise has been devised that will require you to develop a website that uses all three languages.

Theme

Produce a website on a topic of your choice. This could be your favourite film, your favourite brand, a computer game or any other suitable topic. Your website will need to have a minimum of four HTML pages in it.

Requirements

1. Your website should be constructed from HTML, CSS and JavaScript that you have written yourself using an editor like VS code, sublime text, notepad++ or something similar. The js examples in part 2 are in **sublime text**. https://www.sublimetext.com/download
2. You cannot use a development tool like Dreamweaver which writes the code for you.
3. Your site must have a minimum of four pages in it.
4. You must have all of your pages linked to a single CSS file.
5. At least one page must feature user interaction using HTML forms and JavaScript to process and output some form of user input. JavaScript will have to process the user input and output it in some way, this is your decision.
6. Optional: Try to include a variety of media types

You need to make use of each of the features for each language shown on pages 2 and 3 below:

What to hand in – Create an evidence document to showcase the following.

1. Print screens of your pages as they appear in a browser
2. Code listing of your webpages, css and js
3. Ensure you have commented your code to annotate what is happening with each section or line

Language Features to be Used

HTML

Learners are expected to have an awareness of the following tags. Any other tags used will be introduced in the question.

<html>
<link> to link to a CSS file
<head>

<title>

<body>

<h1> <h2> <h3>

<img> including the src, alt, height and width attributes.

<a> including the href attribute.

<form>

<input> where the input is a textbox (i.e. has the attribute type="text" and another attribute name to identify it) or a submit button (i.e. has the attribute type="submit")

<p>

<script>

CSS

Learners are expected to be able to use CSS directly inside elements using the style attribute

```
<h1 style="color:blue;">
```

and external style sheets. In the style sheets they should be able to use CSS to define the styling of elements:

```
h1{
    color:blue;
}
```

classes

```
.infoBox{
    background-color: green;
}
```

and Identifiers

```
#menu{
    background-color: #A2441B;
}
```

They are expected to be familiar with the following properties.

background-color
border-color
border-style
border-width
color with named and hex colours
font-family
font-size
height
width

JavaScript

Learners are expected to be able to follow and write basic JavaScript code. It is hoped they will get practical experience of JavaScript in their study of the course. They will not be expected to commit exact details of syntax to memory. Questions in the exam will not penalise learners for minor inaccuracies in syntax. Learners *will* be expected to be familiar with the JavaScript equivalents of the structures listed in the pseudocode section (with the exception of input and output (see below). They will not be expected to use JavaScript for Object Oriented programming or file handling. Questions will not be asked in JavaScript where something is passed to a subroutine by value or reference is relevant.

Input

Input will be taken in by reading values from a form. *NB learners will not be expected to memorise the method for doing this as focus will be on what they do with that input once it is received.*

Output

By changing the contents of an HTML element

```
chosenElement = document.getElementById("example");
chosenElement.innerHTML = "Hello World";
```

By writing directly to the document

```
document.write("Hello World");
```

By using an alert box

```
alert("Hello World");
```

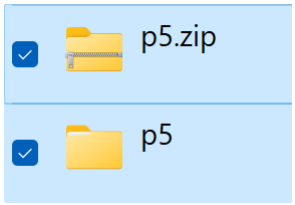
// Help is easily available via the internet, but I recommend w3schools if you do need help.
https://www.w3schools.com/html/default.asp
https://www.w3schools.com/css/default.asp
https://www.w3schools.com/js/default.asp

Part 2 – JavaScript

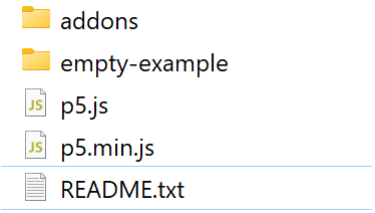
As shown on the previous page, the exam board information about JS is quite vague, so we are going to complete a number of exercises to give you experience of programming a variety of familiar algorithms using the JavaScript syntax.

Exercise 1 – Visit https://p5js.org/download/ and then click on the button highlighted pink below.

You should get a zip file that you can extract/unzip as shown below:

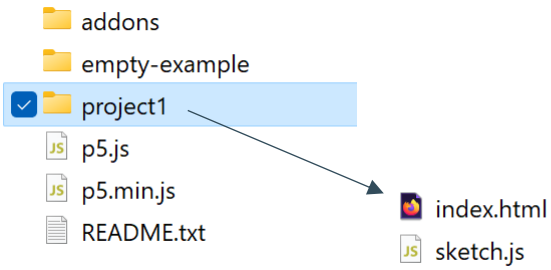


2 – Put the two files into your chosen folder and then navigate to that location, inside the p5 folder you should see:



This is the p5 library plus in the addons folder there is additional functionality for sounds – we won't need this however.

The empty-example folder is the base folder and contains a .html file and a .js file. I would make a copy and rename it (for example) project1, and then every time you want to make a new project, you can copy and paste the empty-example folder to start your next mini project.



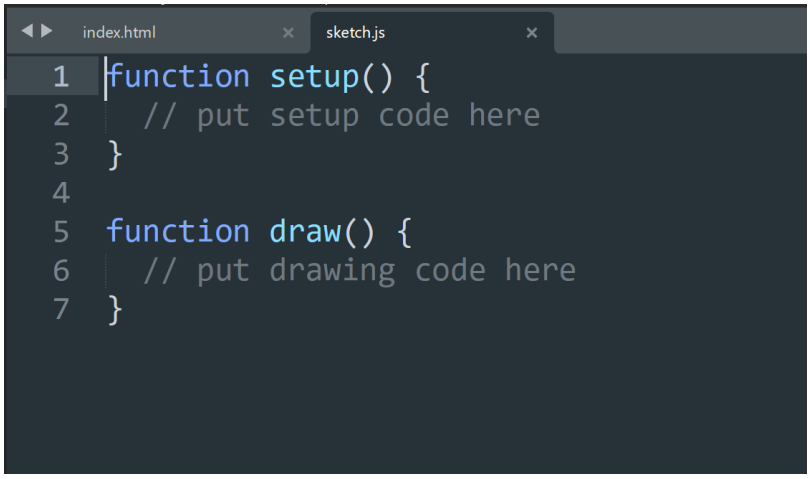
Using an IDE (I am using sublime text – link in part 1 above) go to file > open and open both of the index.html and also sketch.js. They should look like this:

Index.html:

```
<!DOCTYPE html>
<html lang="">

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>p5.js example</title>
  <style>
    body {
      padding: 0;
      margin: 0;
    }
  </style>
  <script src="../p5.js"></script>
  <!-- <script src="../addons/p5.sound.js"></script> -->
  <script src="sketch.js"></script>
</head>

<body>
  <main>
  </main>
</body>

</html>
```

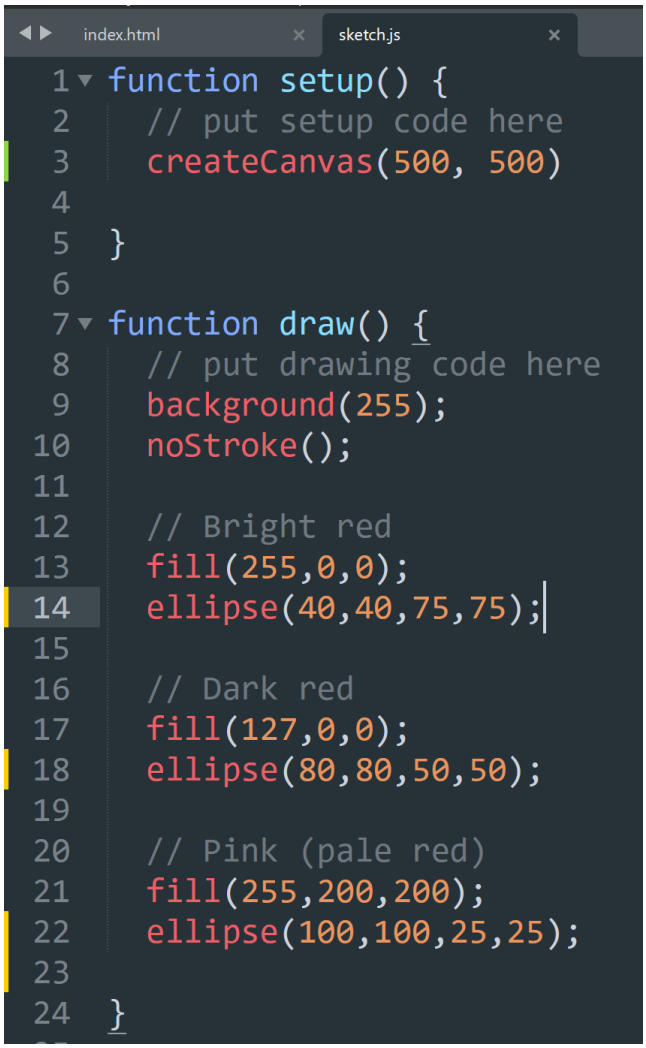
You don't need to do anything with this index file, it already points to the p5 library and the sketch file where you will add your code.

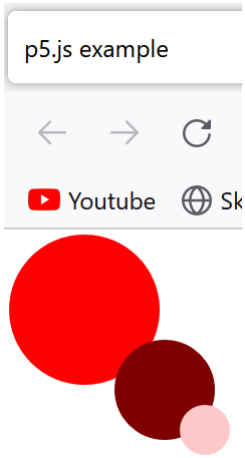
Sketch.js:



This is the file that we will edit to learn some simple js by creating code to draw and then display in our browser.

Type the following code, save it, and then test it by opening the index.html in a browser – I am using FireFox.





This is what you should see in a browser. Paste your code/proof onto your evidence document.

Q – What do the parameters do that we pass to the ellipse function? Show a variety of other ellipses. **Paste your code/proof onto your evidence document.**

Exercise 2 – Start a new project by copy and pasting empty-example and then edit the sketch.js file once more.

index.html sketch.js

```
1  // Variables are usually declared at the top
2  let x = 0;
3
4  // the set up function is called once, used to set the canvas and background
5  function setup() {
6    // put setup code here
7    createCanvas(400, 300)
8
9  }
10
11 // called repeatedly to allow for animation
12 function draw() {
13   ellipse(x, height/2, 20, 20);
14   x = x + 1;
15
16 }
```

Save this file and see what happens. Save the result as a screen print.

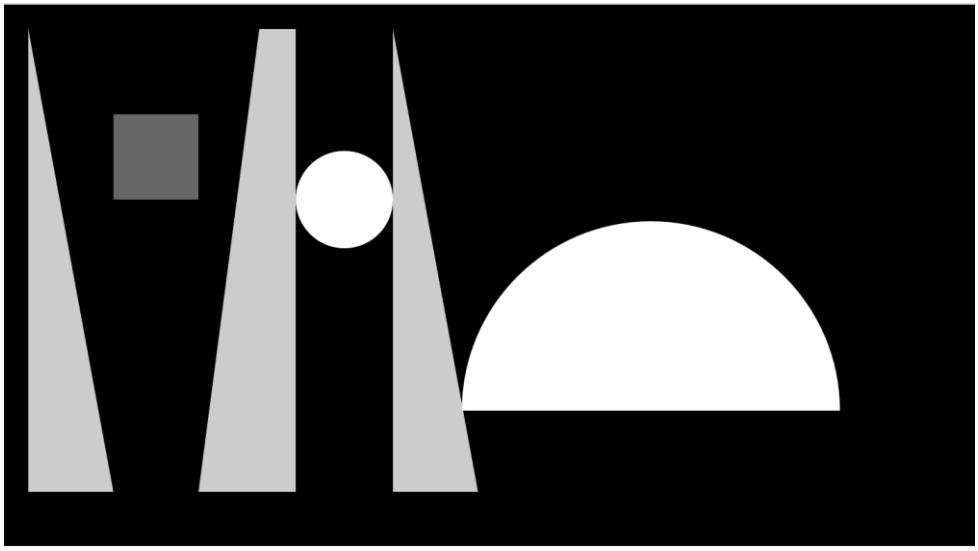
Exercise 3 – Combine the code the code for this example and the previous example to make a multicoloured moving ellipse? **Paste your code/proof onto your evidence document.**

Exercise 4 – Code the shapes below.

index.html sketch.js

```
function setup() {
  // Sets the screen to be 720 pixels wide and 400 pixels high
  createCanvas(720, 400);
  background(0);
  noStroke();

  fill(204);
  triangle(18, 18, 18, 360, 81, 360);

  fill(102);
  rect(81, 81, 63, 63);

  fill(204);
  quad(189, 18, 216, 18, 216, 360, 144, 360);

  fill(255);
  ellipse(252, 144, 72, 72);

  fill(204);
  triangle(288, 18, 351, 360, 288, 360);

  fill(255);
  arc(479, 300, 280, 280, PI, TWO_PI);
}
```

Results in:



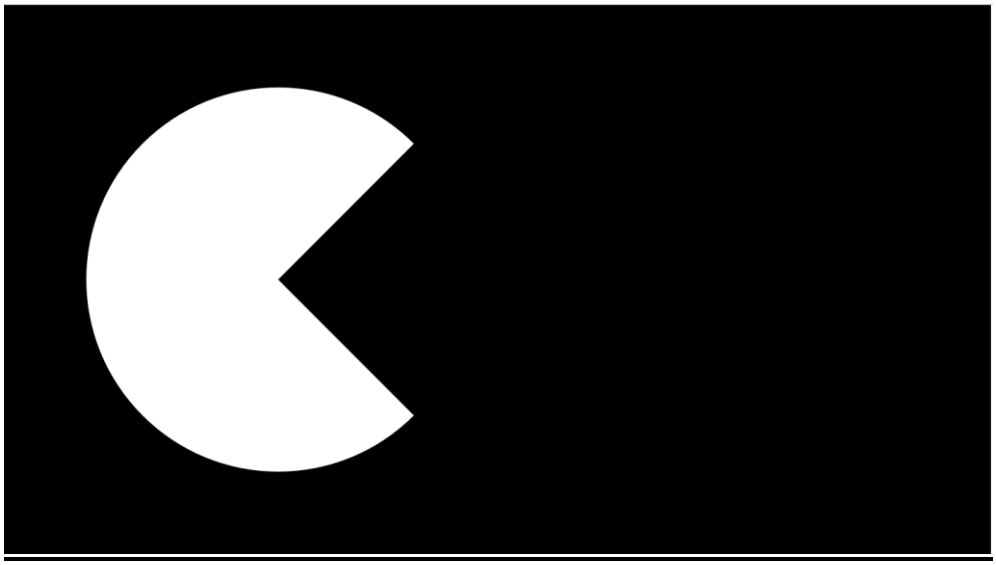
Pay special attention to the **arc function**, the function draws part of an ellipse, the parameters work similar to other shapes, but have start and stop points measured in radians.

Parameter 1 – x location
Parameter 2 – y location
Parameter 3 – width
Parameter 4 – height
Parameter 5 – start (in radians)
Parameter 6 – stop (in radians)

QUARTER_PI = 45 degrees
HALF_PI = 90 degrees
PI = 180 degrees
PI + HALF_PI = 270 degrees
TWO_PI = 360 degrees

```
index.html   sketch.js
1  function setup() {
2    // Sets the screen to be 720 pixels wide and 400 pixels high
3    createCanvas(720, 400);
4    background(0);
5    noStroke();
6
7
8
9    fill(255);
10   arc(200, 200, 280, 280, QUARTER_PI, TWO_PI - QUARTER_PI);
11 }
```

Results in:



Exercise 5 – Create a house scene using shapes, and make it multicoloured rather than monotone as in the example above. **Paste your code/proof onto your evidence document.**

Exercise 6 – Combine the code we have done so far to make a moving PacMan style shape that appears animated and closes its mouth on each movement. **Paste your code/proof onto your evidence document.**

Exercise 7 – Bubble sort

https://p5js.org/examples/simulate-bubble-sort.html

Get the code working in your browser, and read through the annotations to see how it works, and change the background/line colours (colors). **Paste your code/proof onto your evidence document.**

sketch.js

```
let values = [];
let i = 0;
let j = 0;

// The statements in the setup() function
// execute once when the program begins
// The array is filled with random values in setup() function.
function setup() {
  createCanvas(720, 400);
  for(let i = 0;i<width/8;i++){
    values.push(random(height));
  }
}

// The statements in draw() function are executed until the
// program is stopped. Each statement is executed in
// sequence and after the last line is read, the first
// line is executed again.
function draw() {
  background(220);
  bubbleSort();
  simulateSorting();
}

// The bubbleSort() function sorts taking 8 elements of the array
// per frame. The algorithm behind this function is
// bubble sort.
function bubbleSort() {
  for(let k = 0;k<8;k++){
    if(i<values.length){
      let temp = values[j];
      if(values[j] > values[j+1]){
        values[j] = values[j+1];
        values[j+1] = temp;
      }
      j++;

      if(j>=values.length-i-1){
        j = 0;
        i++;
      }
    }
    else{
      noLoop();
    }
  }
}

// The simulateSorting() function helps in animating
// the whole bubble sort algorithm
// by drawing the rectangles using values
// in the array as the length of the rectangle.
function simulateSorting(){
  for(let i = 0;i<values.length;i++){
    stroke(100, 143, 143);
     fill(60);
     rect(i*8 , height, 8, -values[i],20);
   }
}
```

The code below is code for an insertion sort, and is written in python.

insertion sort.py

```python
def insertionSort(alist):
    for index in range(1,len(alist)):

        currentvalue = alist[index]
        position = index

        while position>0 and alist[position-1]>currentvalue:
            alist[position]=alist[position-1]
            position = position-1

        alist[position]=currentvalue
        print (alist)

alist = [9,5,4,15,3,8,11]
insertionSort(alist)
print("\nfinal sorted list ", alist)
```

Exercise 8 – Use the code from the two examples to create a visualisation of the insertion sort in p5.js. **Paste your code/proof onto your evidence document.**

Exercise 9 – Quick sort – use the link or code from scratch using the code below:

https://p5js.org/examples/simulate-quicksort.html

```javascript
// width of each bar is taken as 8.
let values = [];
// The array 'states' helps in identifying the pivot index
// at every step, and also the subarray which is being sorted
// at any given time.
let states = [];

// The setup() function is called once when the program
// starts. Here, we fill the array 'values' with random values
// and the array 'states' with a value of -1 for each position.
function setup() {
  createCanvas(710, 400);
  for(let i = 0; i < width/8; i++) {
    values.push(random(height));
    states.push(-1);
  }
```

```
  quickSort(0, values.length - 1);
}

// The statements in draw() function are executed continuously
// until the program is stopped. Each statement is executed
// sequentially and after the last line is read, the first
// line is executed again.
function draw() {
  background(140);
  for(let i = 0; i < values.length; i++) {
    // color coding
    if (states[i] == 0) {
      // color for the bar at the pivot index
      fill('#E0777D');
    } else if (states[i] == 1) {
      // color for the bars being sorted currently
      fill('#D6FFB7');
    } else {
      fill(255);
    }
    rect(i * 8, height - values[i], 8, values[i]);
   }
}

async function quickSort(start, end) {
  if (start > end) {  // Nothing to sort!
    return;
  }
  // partition() returns the index of the pivot element.
  // Once partition() is executed, all elements to the
  // left of the pivot element are smaller than it and
  // all elements to its right are larger than it.
  let index = await partition(start, end);
```

```
  // restore original state
  states[index] = -1;
  await Promise.all(
    [quickSort(start, index - 1),
     quickSort(index + 1, end)
    ]);
}

// We have chosen the element at the last index as
// the pivot element, but we could've made different
// choices, e.g. take the first element as pivot.
async function partition(start, end) {
  for (let i = start; i < end; i++) {
    // identify the elements being considered currently
    states[i] = 1;
  }
  // Quicksort algorithm
  let pivotIndex = start;
  // make pivot index distinct
  states[pivotIndex] = 0;
  let pivotElement = values[end];
  for (let i = start; i < end; i++) {
    if (values[i] < pivotElement) {
      await swap(i, pivotIndex);
      states[pivotIndex] = -1;
      pivotIndex++;
      states[pivotIndex] = 0;
    }
  }
  await swap(end, pivotIndex);
  for (let i = start; i < end; i++) {
    // restore original state
    if (i != pivotIndex) {
```

```
      states[i] = -1;
    }
  }
  return pivotIndex;
}

// swaps elements of 'values' at indices 'i' and 'j'
async function swap(i, j) {
  // adjust the pace of the simulation by changing the
  // value
  await sleep(25);
  let temp = values[i];
  values[i] = values[j];
  values[j] = temp;
}

// custom helper function to deliberately slow down
// the sorting process and make visualization easy
function sleep(ms) {
  return new Promise(resolve => setTimeout(resolve, ms));
}
```

This is one implementation of the quick sort, get it working in another p5.js project, and change the background/line colours (colors). **Paste your code/proof onto your evidence document.**

Exercise 10 – Merge sort

The code below is an implementation of a merge sort written in Python.

```
# merge sort
def mergeSort(alist):
    if len(alist) > 1:
        # print("Splitting ",alist)
        mid = len(alist) // 2  # performs integer division
        lefthalf = alist[:mid]  # left half of alist put into lefthalf
        print("left half ", lefthalf)
        righthalf = alist[mid:]  # right half of alist put into righthalf
        print("right half ", righthalf)
        mergeSort(lefthalf)
        mergeSort(righthalf)
        i = 0
        j = 0
        k = 0
        while i < len(lefthalf) and j < len(righthalf):
            if lefthalf[i] < righthalf[j]:
                alist[k] = lefthalf[i]
                i = i + 1
            else:
                alist[k] = righthalf[j]
                j = j + 1
            k = k + 1
        # endwhile
    # endif
    # check if the left half still has elements not merged
    # if so, add them to alist

        while i < len(lefthalf):
            alist[k] = lefthalf[i]
            i = i + 1
            k = k + 1
        # endwhile
        # check if the right half still has elements not merged
        # if so, add them to alist
        while j < len(righthalf):
            alist[k] = righthalf[j]
            j = j + 1
            k = k + 1
        # endwhile
        print("Merged sublist ", alist)
     # endif

# ENDSUB
# ************ MAIN PROGRAM ***************
alist = [5, 3, 2, 7, 9, 1, 3, 8]
print("\nUnsorted list: ", alist)
mergeSort(alist)
print("\nSorted list: ", alist)
```

Use the code from the two examples to create a visualisation of the merge sort in p5.js. **Paste your code/proof onto your evidence document.**

Part 3 – Practical Project – Development

At this stage, the analysis and the problem decomposition and GUI/Usability sections of the design of your project should be complete and although it may be necessary to revisit these two sections as you come across unexpected issues etc, it is now time to complete the design and then move onto the development of your system.

Outline to stages to complete

1 – The analysis should be completed – the formal deadline was 26th June 2025.

2 – The design

- Over summer you should complete:
  - D1 – Decomposition
  - D2 – User interface and usability

Complete at least one, preferably two tutorials that are of similar projects

3 – To help you with writing your design algorithms, thinking about what you will need to test, and also help with your implementation, you will need to:

Find one or two example projects that are of a similar type to yours, for examples if you are doing phone app, complete something similar, if you are making a game, find a similar resource. This will help with the design and development sections of the NEA. Add these to your NEA doc to share with me.