OCR A Level Computer Science

Your notes

3.4 Web Technologies

Contents


Your notes

Search Engine Indexing

How do search engines work?

Search engines work in several stages:

- Crawling - think of this as gathering all of the books within a library
- Indexing - think of this as reading the books and making a structured list of the information within the books
- Ranking - think of this as recommending books to the reader

Crawling

- Web pages are discovered by search engines through **software programs called crawlers** (or spiders, bots, or robots)
- Crawlers **follow links from one webpage to another**, systematically visiting pages on the web
- They start from a set of seed URLs and visit other pages linked from those URLs
- Website crawlers **follow rules and guidelines established by website owners**, using mechanisms like the robots.txt file. These guidelines **direct crawlers on which areas of a website to explore or avoid**, respecting website preferences and ensuring privacy
- Once a crawler reaches a webpage, it **fetches the HTML content** of that page
- The crawler **examines the HTML structure and retrieves information**, such as text content, headings, links, and **metadata**
- To understand the structure of the webpage, **the HTML that was retrieved is broken down into individual components**
- This process involves **identifying elements, tags, and attributes** that hold valuable information like titles and headings

Indexing

- The data extracted from the webpage is indexed, which involves storing the collected information in a structured manner within a search engine's database
- Each word in the document is included in the page's index as an entry, along with the word's position on the page
- The **index allows for quick retrieval and ranking of relevant web pages** in response to user queries

Ranking

- When a user enters a query, the search engine searches the index for matching pages and returns the results they believe are the highest quality and most relevant to the user's query

Your notes

Benefits of search engine crawling & indexing

- The process of search engine **indexing** is **essential for search engines** to collect, examine and arrange online content
- It involves **collecting and storing information from web pages** in a searchable index

There are many reasons for search engine crawling and indexing to happen:

- Improved search results
- Efficient retrieval
- Ranking and relevance
- Freshness and updates

Improved search results

- Indexing webpages means search engines can:
  - Provide users with **relevant and up-to-date search results**
  - Match user searches with content which **increases the chances of accurate and valuable results**
- This means the user is **more likely to find what they're looking for quickly**, ideally on the first page of search results, without having to go to additional pages

Efficient retrieval

- Indexing enables **efficient retrieval of information**
- Search engines don't need to scan the entire web for every search query. They can just **search their indexed data to produce search results quickly**

Ranking & relevance

- Indexing enables search engines to **assess the relevance and quality of web pages**
- Search result **rankings are determined by various ranking algorithms** that analyse indexed data. These algorithms consider factors such as **keyword relevance, backlinks, and user engagement**

Freshness & updates

- Search engine crawlers **periodically revisit indexed web pages to detect updates and changes**

- This process guarantees that the search results **display the latest content** that is currently accessible on the Internet
- If a webpage has been updated and not re-crawled, the page may no longer be relevant for the user's search


Your notes

Your notes

PageRank Algorithm

- A crucial element of search ranking algorithms is the Page Rank algorithm
  - The algorithm was developed by Larry Page and Sergey Brin
  - Many search engines rely on it, particularly Google
- **Web pages are evaluated and ranked by the algorithm** based on their perceived relevance and importance

Why is the PageRank algorithm important?

- The PageRank algorithm was created to **tackle the difficulty of determining the importance of web pages** with the **immense amount of information available**
- The purpose of the algorithm is to **provide better search results that are more precise and related** by taking into account various factors beyond just matching keywords

Key elements of the PageRank algorithm

There are 4 key elements to the PageRank algorithm:

- Link analysis
- Link weight distribution
- Iterative calculation
- Damping factor

Link analysis

- The PageRank algorithm **analyses the structure of links between pages** on the web
- **Web pages are given importance** by the algorithm, which considers the **quantity and quality of inbound links from other pages**
- Each **link acts as a "vote" for the target page**, with the voting weight determined by the importance of the linking page
- Websites that have **more high-quality links pointing towards them** are deemed to be more valuable and pertinent and have a higher weight
- Webpages with a **higher weight will score more highly** and have a higher ranking

Link weight distribution

- **The importance of a webpage is calculated by PageRank**, which takes into account the total number of "votes" it has received
- The algorithm distributes the importance of a page to the pages it links to by s**haring a portion of its importance with each outgoing link**
- By following this process, **pages of superior quality are given greater importance** and make a larger impact in determining the ranking of other pages

Iterative calculation

- The PageRank algorithm uses a **repetitive calculation** process. At the beginning, every webpage is given the same value to start with
- In subsequent iterations, **the significance of each page is re-evaluated** by considering the weighted impact of inbound links
- The process continues until the rankings become stable

Damping factor

- In order to avoid infinite loops, an algorithm introduces a **damping factor that ranges between 0 and 1 (usually set at 0.85)**
- The damping factor is the **likelihood of a user clicking on a link at random** rather than following the links on the current page
- The damping factor ensures that the **ranking calculation includes user behaviour** and maintains harmony between discovering new links and staying on the current page

Factors influencing PageRank

Although the initial PageRank algorithm mainly concentrated on link analysis, present-day search engines consider many factors to improve search results rankings. These factors may include:

- Relevance
- User engagement
- Authority and trust
- Content freshness
- Mobile-friendliness

Relevance


Your notes

- The **content of a web page is a crucial factor** in determining its ranking in search results. This is influenced by the **keywords** used, the **quality** of the content, and **how relevant it is** to the search query


Your notes

User engagement

- **The way users interact with a website can be measured through metrics** like click-through rates, time spent on a page (dwell time), and bounce rates. These metrics can reveal the level of user engagement
- Pages that receive **greater engagement from users may be deemed more valuable**

Authority & trust

- The **reputation** and authority of a webpage or website **play a crucial role**
- **Several factors can enhance a website's ranking**, including the **age** of the domain, **quality backlinks** from reputable sources e.g. government website or the BBC, and **trustworthy content**

Content freshness

- Search engines value **fresh and up-to-date content**
- Search queries may give **priority to web pages that are frequently updated** or have up-to-date information

Mobile-friendliness

- As mobile devices became more prominent, **search engines started to factor in the mobile compatibility of web pages** when determining their ranking
- Google primarily **uses the mobile version of a site's content to rank pages** from that site
- Having a responsive design and **optimising the user experience on mobile devices can have a positive impact** on a website's rankings

Limitations & evolving nature

- Although the **PageRank algorithm** is important in search engine rankings, it **is not the only factor** that determines them
- **Search engines use different algorithms** and factors to guarantee that they provide varied, relevant, and top-quality search outcomes
- Over time, the **details of the PageRank algorithm have undergone changes**. Search engines regularly enhance their ranking methods to cater to new challenges and meet user expectations

How does PageRank work?

- To illustrate how PageRank works, let's use **players in a football match** where:
  - Each **player represents a page**

- Each **pass between 2 players represents a link** between 2 pages

PageRank Analogy – a team of football players

- The main things PageRank uses are:
  - The number of links the page gets (or the **number of passes a player receives**)
  - The importance of a page is determined by the number of links pointing towards it or by **how frequently the player who passed the ball is passed to**
- The PageRank of each player gets **updated every time they receive the ball**
- This continues throughout the game

Your notes

Your notes

PageRank Analogy – the players receive a numerical rating based on number and frequency of passes

- As **more passes are made, the PageRank of each player undergoes changes**
- As a result, the PageRank of every player they pass to will be altered
- The **number represents each player's PageRank – the higher the number, the better**
- Once the game concludes, players can be ranked to determine the best performer

PageRank Analogy - the players can now be sorted by their rating

Examiner Tips and Tricks

- In the exam, you won't be asked about the algorithm specifically, just the overall idea of how it works, as detailed above
- You don't need to know exactly how it is calculated
- Although it was created by Google, it's used by many search engines so don't mention Google in the exam

Your notes

Server Side & Client Side Processing

Your notes

Server Side Processing

- Server side processing involves **running code and carrying out operations on the server** instead of on the client's device or browser
- Web development often involves utilising **server side programming languages like PHP, Python, Ruby, or Java** to handle incoming requests, process data, interact with databases, and generate dynamic content

Server side processing with PHP

PHP (Hypertext Preprocessor) is a server side scripting language specifically designed for web development. PHP focuses mainly on **completing tasks on the server**. Here are a few examples of server side processing with PHP:

Data retrieval & manipulation

- When working with PHP, you can retrieve and manipulate data. PHP is capable of **interacting with databases, processing data, and generating dynamic content**

Server operations

- PHP can perform server side operations that are **not accessible to the client**
- PHP is capable of performing server operations such as **retrieving and displaying information from a database**

Form processing

- PHP can handle form submissions, process the submitted data, and perform necessary validations or database operations on the server side

Benefits & drawbacks of server side processing

Benefits of Server Side Processing	Drawbacks of Server Side Processing

Improved security measures can be implemented through server side processing, ensuring the secure management of sensitive data, implementing access control measures, and guarding against common web vulnerabilities.	When multiple requests are made to a server, complex processing tasks can consume server resources and cause a decrease in overall server performance. This is known as **increased server load.**
Server side processing uses the resources of the server to perform advanced calculations, manipulate data, and interact with databases.	Using server side processing may cause latency because it involves communication with the server, which could lead to **slower response times** in comparison to client side processing.
Server side processing ensures **consistent behaviour across different devices and browsers**, as the processing logic is centralised on the server.	Server side processing **relies on the availability and reliability of the server** infrastructure. Downtime or performance issues can affect the functioning of the web application.
Server side processing can be **easily scaled** by adding more servers or optimising the server infrastructure to handle increasing traffic and user demands.	Server side processing typically requires a roundtrip to the server for each user action, **limiting real-time interactivity** and responsiveness.
	Server side processing may require more **complex development** and setup compared to client side processing, potentially increasing development time and effort.


Your notes

Client Side Processing

- Client side processing involves **carrying out code or processing tasks on the user's device**, usually within the web browser, instead of on the **server**
- This feature enables users to have **interactive and dynamic experiences** without constantly requesting data from the server
- Client side processing is **primarily done using JavaScript**, whereas server side processing is commonly carried out using **PHP**

Client side processing with JavaScript

JavaScript is a powerful scripting language that operates mainly on the client side. It provides developers with the ability to **modify web content, manage user interactions, and update the webpage dynamically** without requiring server requests. Here are a few examples of client side processing with JavaScript:


Your notes

Form validation

- With JavaScript, it's possible to **validate user input in real time**, which means that **users can receive instant feedback** without the need for a server roundtrip
- E.g. when completing an online form, check that all required fields are filled out correctly and make sure the input meets the necessary format and length before sending the form to the server. If any areas are blank and need input (e.g. email address) the user will be notified before the form can be submitted

DOM manipulation

- With JavaScript, developers can modify the Document Object Model (DOM) to **make dynamic changes to a web page's content and structure**
- This involves tasks such as adding or removing elements, updating text, modifying styles, and managing events to develop interactive user interfaces e.g. turning on dark mode

AJAX requests

- **Communication with the server happens asynchronously** and retrieves data in the background
- This allows for **dynamically updating content without requiring a full-page reload**

Benefits & drawbacks of client side processing

Benefits of Client Side Processing	Drawbacks of Client Side Processing
Enhanced user experience is made possible through client side processing, creating interactive and dynamic user experiences. This eliminates the need for frequent server requests and page reloads.	There is a **potential security risk** with client side code as it can be seen by users, which may lead to sensitive information and operations being exposed or tampered with.
By offloading processing tasks to the client side, the **server load is reduced**, resulting in improved scalability and resource utilisation.	The **compatibility of devices and browsers** may vary, which can lead to issues with the client side code that depends on their capabilities and support.
User input can be instantly validated and **feedback can be provided in real-time**. This not only improves the user experience but also reduces the need for server roundtrips.	Client side processing can hurt **page load time**, particularly when dealing with large or complex operations that require substantial processing power.

With the use of JavaScript, web pages can have their content updated dynamically, resulting in a more seamless and **engaging browsing experience**.	Client side processing is heavily **dependent on JavaScript**. If the user's browser does not support or has disabled JavaScript, the functionality may become inaccessible or break.
Web applications can operate without an active internet connection by using client side technologies to provide **offline functionality.**	The accessibility of client side code to users can put **intellectual property at risk,** as it allows for easier viewing, copying, and modification of the code.

Your notes

Choosing Server Side or Client Side Processing

The **choice** between client side and server side processing **depends on the specific requirements** of a task:

- Client side processing is better for tasks that require **immediate user feedback, real-time interactions, dynamic user interfaces, or data manipulation** within the browser. **JavaScript** is the primary language for such scenarios
- Server side processing is better for tasks that involve **accessing databases, handling sensitive data, complex business logic, or server specific operations**. **PHP** and other server side languages are commonly used in these cases

Worked Example

Big Brains has produced a website that allows students to access revision videos. They want to limit access to those students with a school email account (i.e. one ending .sch.uk). When students sign up JavaScript is used to check that the email address they have entered is from a school account. The address is rechecked when it reaches the server before login details are sent to the address.
Explain why checking the email address with JavaScript and again when it reaches the server is important.

3 marks

How to answer this question:

- You need to know the following from above about client side processing:
  - **Reduced Server Load:** Offloading processing tasks to the client side reduces the burden on servers, improving scalability and resource utilisation

- **Dependency on JavaScript:** Client side processing heavily relies on JavaScript, and if JavaScript is disabled or not supported by the user's browser, the functionality may break or become inaccessible
- You need to apply this knowledge to the scenario in the question

Answer:

Example answer that gets full marks:

The JavaScript check is carried out client side meaning the address can be checked and stopped before reaching the server reducing the unnecessary load on the server.

Acceptable answers you could have given instead:

JavaScript can be amended and circumvented therefore address must be checked at the server to ensure this has not happened.


Your notes

HTML

Your notes

Writing HTML

- **HTML**, or HyperText Markup Language, is the foundational language **used to structure content on the web**
- HTML consists of a series of elements, often referred to as "**tags**," which can be used to structure and format a webpage
- The <html> tag is the root element of an HTML page. The tag includes all other HTML elements used on the page
- Most tags are **opened and closed e.g.** <html> and </html> whereas **some tags are only opened e.g.** <img> and <link>
- The content layer of a web page is made up of HTML elements such as headings (<h1>, <h2>, etc.), paragraphs (<p>), links (<a>), images (<img>), and more

Examiner Tips and Tricks

- An exam question which asks you to write HTML is asking you to write code which is specifically HTML. This means all spelling and syntax MUST be correct for you to get the marks

The Head

- The **head** section contains information about the web page that's **not displayed on the page itself**
  - The head section is enclosed by <head> and </head> Tags
  - **Some of the content** inside the head tag is **shown in the browser tab**

Page title

- The <title> element is used to set the **page title that shows in the browser tab**
- The crown is placed inside the <head> section of the HTML document
- E.g. <title>Sample Webpage</title>

External stylesheet

- **External stylesheets are linked** in the <head> section using the <link> element

- The rel the attribute is set to "stylesheet", and the href the attribute contains the relative file path to the CSS file
- Stylesheets are **loaded in the order they are listed**, so hierarchy is important
- E.g. <link rel="stylesheet" type="text/css" href="styles.css">

Your notes

The Body

- The **body section contains the main content of the web page**, such as text, images, videos, hyperlinks, tables etc.
  - The body section is enclosed by <body> and </body> Tags
  - The content inside the body tag is displayed in the browser window

Headings

- **Headings help users understand how a page is organised and structured**. They help guide the user's eye around the page and help them quickly find the information they're looking for
- Headings are also important for users that use **screen readers**, such as visually-impaired users. Screen reader software uses headings to help the user navigate around the page (e.g. providing a list of all the headings on the page, and allowing the user to jump straight to a particular heading)
- **The tags used for headings are** <h1>, <h2>,<h3> etc.
- <h1> is the highest level heading, with <h2> and <h3> serving as subheadings
- The number in the tag denotes the level of the heading. Headings should be used hierarchically, with <h1> the most prominent, and <h6> the least prominent
- **Headings are also important for Search Engine Optimisation (SEO)**, as they help search engines like Google understand the content and structure of the page

E.g.

```
<h1>Welcome to My Sample Webpage</h1>

<h2>About This Page</h2>
```

Images

- The <img> tag embeds an image into an HTML document
- The img tag includes the src attribute which specifies the URL of the image (the source of the attribute), and the alt attribute which provides **alternative text for screen readers** or in cases where the image cannot be displayed. The tag also helps search engines understand the content of the image
- The height and width attributes specify the dimensions of the image

- E.g. <img src="https://cdn.savemyexams.com/logo/sme-logo-small.svg" alt="Save My Exams Image" height="100" width="200">


Your notes

Links

- The <a> tag, or anchor tag, is **used to create hyperlinks**
- The tag includes the href attribute which **specifies the URL that the link points to**
- This **can be included within another tag** to specify the text which will be displayed
- E.g. Visit the Save My Exams site for more information.
- An **image can also become a hyperlink**
- E.g. <img src="logo.png">

Div

- The tag is a generic container that can be **used to group other HTML elements together**
- The div tag is often used in conjunction with **CSS** to style sections of a webpage

Form

- The <form> tag is used to **create a form for user input**

Input

- The <input> tag is used within a <form> tag to **create interactive controls for web-based forms**
- The type attribute can be set to "text" to create a **textbox**, or "submit" to create a **submit button**
- The name attribute is used to identify the input
- E.g.

```
<form action="/submit" method="post">

<label for="age">Age:</label>

<input type="number" id="age" name="age" min="0">

<label for="email">Email:</label>

<input type="email" id="email" name="email">

<input type="submit" value="Submit">

</form>
```

Paragraph

- The tag is used for a **paragraph of text**
- E.g. This is a sample webpage created to demonstrate the usage of various HTML tags.

Your notes

Lists

- The <li> tag is used to define a list item
- The <ol> and <ul> Tags are used to create ordered and unordered lists
  - An ordered list will look like **numbered bullet points**
  - An unordered list will look like normal **bullet points**
- tags are nested within these to show each item in the list
- E.g.

```
<ul>
  <li>html</li>
  <li>head</li>
  <li>title</li>
  <li>body</li>
  <li>h1, h2, h3</li>
  <li>p</li>
  <li>a</li>
  <li>img</li>
  <li>div</li>
  <li>form</li>
  <li>input</li>
  <li>script</li>
</ul>
```

JavaScript

- The <script> tag is used to **embed or reference JavaScript code** within an HTML document
- E.g. <script src="script.js"></script>

Examiner Tips and Tricks

- It's important to note that the correct usage of relative paths/URLs depends on the file structure and organisation of your web project. Carefully consider the relative relationship between files and directories to ensure accurate referencing of resources

Your notes

Classes & Identifiers in HTML

- Classes and identifiers (IDs) can be used to specify an HTML element
- **A class can have many HTML elements**
- An ID specifies a unique ID for an HTML element
- **Only 1 element can have the ID**
- A class or an ID can have **specific styling applied to it in the CSS**
- E.g.

```
<body>
 <div id="header">
  <h1>Study Guide</h1>
 </div>
 <div class="subject" id="maths">
  <h2>Maths Revision</h2>
  <p>Key topics to revise are algebra, trigonometry, and statistics.</p>
 </div>
 <div class="subject" id="english">
  <h2>English Revision</h2>
  <p>Focus on improving your grammar, vocabulary, and essay writing skills.</p>
 </div>
 <div class="subject" id="science">
  <h2>Science Revision</h2>
  <p>Remember to revise the core concepts in physics, chemistry, and biology.</p>
```

```
 </div>
</body>
```

Styling is covered in more detail in CSS but here's an example to illustrate the use of classes and IDs:

```
/* styles.css */
#header {
 background-color: lightblue;
 padding: 10px;
 text-align: center;
}
.subject {
 border: 1px solid black;
 margin: 10px;
 padding: 10px;
}
#maths {
 background-color: #FFDDDD;
}
#english {
 background-color: #DDFFDD;
}
#science {
 background-color: #DDDDFF;
}
```

Your notes

Example Webpage in HTML

Here's an example webpage and the HTML code written to create it:

Your notes

Welcome to My Sample Webpage

About This Page

This is a sample webpage created to demonstrate the usage of various HTML tags.

HTML Elements Used

- html
- head
- title
- body
- h1, h2, h3
- p
- a
- img
- div
- form
- input
- script

Contact Us

Name:
Submit

Image Example

External Link

Visit the Save My Exams site for more information.

This is the code for this webpage:

```
<!DOCTYPE html>

<html>

<head>

 <title>Sample Webpage</title>

 <link rel="stylesheet" type="text/css" href="styles.css">

</head>

<body>

 <h1>Welcome to My Sample Webpage</h1>
```

```
<h2>About This Page</h2>

<p>This is a sample webpage created to demonstrate the usage of various HTML tags.</p>

<h3>HTML Elements Used</h3>

<ul>
 <li>html</li>
 <li>head</li>
 <li>title</li>
 <li>body</li>
 <li>h1, h2, h3</li>
 <li>p</li>
 <li>a</li>
 <li>img</li>
 <li>div</li>
 <li>form</li>
 <li>input</li>
 <li>script</li>
</ul>

<h2>Contact Us</h2>

<form action="/submit_form" method="post">
 <div>
  <label for="name">Name:</label>
  <input type="text" id="name" name="name">
 </div>
 <div>
  <input type="submit" value="Submit">
 </div>
</form>
```

Your notes

```
<h3>Image Example</h3>

<img src="https://cdn.savemyexams.com/logo/sme-logo-small.svg" alt="Save My Exams Image" height="100" width="200">

<h3>External Link</h3>

<p>Visit the <a href="https://www.savemyexams.co.uk">Save My Exams site</a> for more information.</p>

</body>

<script src="script.js"></script>

</html>
```

Your notes

Worked Example

One page in the website contains a hyperlink on an image. When the image stored as "ticket.png" is clicked, the user is hyperlinked to the page stored as "booking.htm".

Write the HTML code to implement this hyperlink.

3 marks

How to answer this question:

You can use the example code from above to write the link:

```
<a href="https://www.savemyexams.co.uk"><img src="logo.png"></a>
```

You'll need to change the link to the one given in the question (booking.htm) and change the image name to the one given in the question too (ticket.png)

The marks are awarded for:

- Correct `<a>` with closing `</a>`
- `href` property to correct page
- correct `<img>` tag with src property to correct file

Answer:

Example answer:

```
<a href="booking.htm"><img src="ticket.png"></a>
```

Examiner Tips and Tricks

- When you're given a link or filename in the question, make sure you copy it exactly as it appears. Your case (capital letters) and spelling need to be spot on to make sure you don't drop marks! E.g. https://www.youtube.com/watch?v=fQBfpaiYlWg is not the same as https://www.youtube.com/watch?v=fqbfpaiyiwg

Your notes

CSS


Your notes

Writing CSS

- Cascading Style Sheets (CSS) **define the style or appearance** of a webpage
- CSS uses selectors such as classes or IDs
- CSS can be **placed within HTML or externally in a file**
- Multiple pieces of CSS can be combined
- Where CSS is used within the HTML, this will be used rather than any external CSS styling and will override the external stylesheet

Classes & Identifiers

Classes

- In CSS, **a class is a type of selector that is used to apply a specific style to one or more HTML elements**
- A class is **denoted by a full stop (.)** followed by the class name. E.g. .myClass refers to any HTML element with the class attribute set to "myClass"
- The class selector is versatile in that it **allows the same styles to be applied to multiple HTML elements across the webpage**

Identifiers

- An identifier, also known as an ID, is another type of selector in CSS
- An identifier is **represented by a hash (#)** followed by the ID name. For example, #myID would select any HTML element with the ID attribute set to "myID"
- IDs are unique within a webpage, meaning that **each ID can only be used once per page**. This makes them useful for styling singular, distinct elements on a webpage
- E.g.

```
<body>
 <div id="header">
  <h1>Study Guide</h1>
 </div>
 <div class="subject" id="maths">
```

```
  <h2>Maths Revision</h2>
  <p>Key topics to revise are algebra, trigonometry, and statistics.</p>
 </div>
 <div class="subject" id="english">
  <h2>English Revision</h2>
  <p>Focus on improving your grammar, vocabulary, and essay writing skills.</p>
 </div>
 <div class="subject" id="science">
  <h2>Science Revision</h2>
  <p>Remember to revise the core concepts in physics, chemistry, and biology.</p>
 </div>
</body>
```

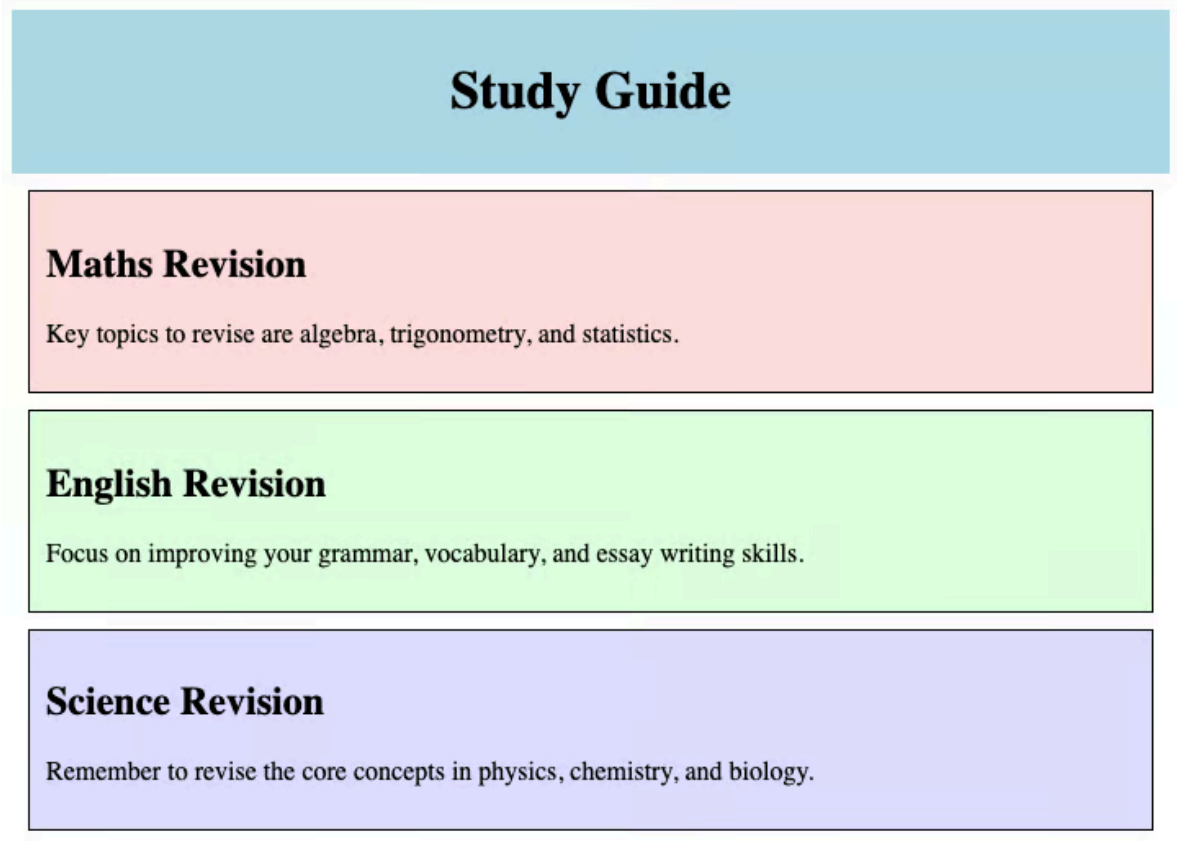
Here's an example to illustrate the styling of classes and IDs:

```
/* styles.css */
#header {
 background-color: lightblue;
 padding: 10px;
 text-align: center;
}
.subject {
 border: 1px solid black;
 margin: 10px;
 padding: 10px;
}
#maths {
 background-color: #FFDDDD;
}
```

Your notes

```
#english {
  background-color: #DDFFDD;
}
#science {
  background-color: #DDDDFF;
}
```

This is what the page looks like:



- All the subjects have the **same border and heading style** but each one has a **different coloured background**

Your notes

CSS: Styling

Your notes

Writing CSS: Styling

- **CSS** can be used to **style individual elements e.g. all of the h1s** or all the paragraphs

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

- CSS can also be used to style classes by adding a . before the class name

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

- Identifiers can be styled by adding a # before the identifier name

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

- Styling can also be done inline (in the HTML). The examples below will cover both

```
<p style="background-color: lightblue;"> Save My Exams </p>
```

- Multiple properties can be included in the styling

```
#header {
 background-color: lightblue;
 padding: 10px;
 text-align: center;
}
```

Examiner Tips and Tricks

- Your code must be exactly what it appears on this page. So color must be spelt the American way as colour doesn't work in CSS. Make sure you know the properties listed below and don't name them something which is incorrect e.g. text-colour instead of color
- Also make sure your syntax is correct so use : instead of =


Your notes

Properties

background-color

Example within HTML

```
<p style="background-color: lightblue;"> Save My Exams </p>
```

Example in external CSS

```
.h1{
  background-color: lightblue;
}
```

border-color

Example within HTML

```
<p style="border-color: black;  > Save My Exams </p>
```

Example in external CSS

```
.h1{
   border-color: black;
}
```

border-style

Example within HTML

```
<p style="border-style: solid; " > Save My Exams </p>
```

Example in external CSS

```
.h1{
  border-style: solid;
}
```

border-width

Example within HTML

```
<p style="border-width: 2px;  "> Save My Exams </p>
```

Example in external CSS

```
.h1{
 border-width: 2px;
}
```

colour

Note that this changes the font colour

Example within HTML

```
<p style="color: #ff0000;  "> Save My Exams </p>
```

Example in external CSS

```
.h1{
 color: #ff0000;
}
```

font-family

Example within HTML

```
<p style="font-family: Arial;" > Save My Exams </p>
```

Example in external CSS

```
.h1{
 font-family: Arial;
}
```

font-size

Example within HTML

```
<p style="font-size: 14px; " > Save My Exams </p>
```

Example in external CSS

```
.h1{
```

Your notes

```
 font-size: 14px;
}
```

height & width

Example within HTML

```
<p style="height: 200px; width: 200px;" > Save My Exams </p>
```

Example in external CSS

```
.h1{
 height: 200px;
 width: 200px;
}
```

Colours

- Colours can be **referred to by name or hex number**
- Colour is **spelt the American way in CSS (color)** - the code won't work if written the British way (colour)
- There are 140 colour names which can be used. Here is a selection:

Black	Blue	Crimson	Cyan
Gold	Grey	Green	Indigo
Orange	Red	White	Yellow

- Colours can also be given **using a hex code e.g. #9ACD32**
- Every 2 characters in the hex colour code **represent either red, green or blue**
  - E.g. Red is #FF0000
  - Green is #00FF00
  - Blue is #0000FF
  - A colour picker can be used to get the hex code for a particular colour

Your notes

Worked Example

The site also contains the following code.

```
<div class="offer">All oranges 50% off.</div>
```

Complete the CSS code that would make any div elements of the class offer have an orange border.

2 marks

```
.....{
    border-style: solid;

    .....

    }
```

How to answer the question:

The styling will apply to the div elements of the class offer. This is done using the .div name so `.offer`

It needs to have an orange border. The property for this is border-color so `border-colour: orange`

Answer:

The correct answer:

```
.offer{

  border-style: solid;

  border-color: orange;

  }
```

Your notes


Your notes

JavaScript

What is JavaScript?

JavaScript is a programming language that **adds interactivity and dynamic functionality to webpages**. It allows a webpage to manipulate HTML elements, perform calculations, handle user input, and more.

Purpose of JavaScript

- JavaScript is primarily used for client-side scripting, meaning **it runs directly in the web browser of the user**
- Its purpose is to enhance webpages by enabling interactive features, dynamic content, form validation, and data manipulation
  - Interactive features include features like **image sliders, drop-down menus, or interactive maps**, like allowing users to zoom in and out on a Google Maps element embedded in the website
  - Dynamic content could be loading more articles or posts when a user reaches the end of a page (**infinite scrolling**), or changing the content based on the user's actions, such as s**howing different product descriptions when a user hovers over product images**
  - Form validation could be **checking that all required fields have been filled in**, that an email address has the correct format, or that p**asswords meet certain strength requirements**
  - Data manipulation could involve **filtering a list of products based on user-selected criteria**, sorting a table of data, or **creating a dynamic chart that updates in real-time as new data comes in**
- JavaScript can respond to user actions, update the content of HTML elements, and communicate with servers to retrieve or send data

Data Types in JavaScript

Data Types in JavaScript

- A data type is a classification of data into groups according to the **kind of data they represent**
- Computers use different data types to represent different types of data in a program
- The basic data types include:
  - **Integer**: used to represent **whole numbers**, either positive or negative
    - Examples: 10, -5, 0

- **Real**: used to represent **numbers with a fractional part**, either positive or negative
  - Examples: 3.14, -2.5, 0.0
- **Char**: used to represent a **single character** such as a letter, digit or symbol
  - Examples: 'a', 'B', '5', '$'
- **String**: used to represent a **sequence of characters**
  - Examples: "Hello World", "1234", "@#$%
- **Boolean**: used to represent **true or false** values
  - Examples: True, False

Your notes

We can declare variables as follows:

Syntax	let variable_name = value;
Example	let x = 5;

It is important to choose the correct data type for a given situation to ensure accuracy and efficiency in the program.


Your notes

Variables & Constants in JavaScript

Variables and **constants** are used to **store a single item of data** in a program. This can be accessed through the identifier. **Variables can be changed during program execution** while **constants remain the same**.

Declaring variables & constants

- Variables are declared using a data type, a name and a value (optional)
- Constants are declared using the 'const' keyword, a name and a value
- In all programming languages, **variable names should follow certain rules**, such as:
  - Starting with a letter
  - Not containing spaces
  - Can contain letters, numbers, _ or $
  - Not using reserved words (like if, while, for etc.)
- Examples of data types include **integer, float, boolean, and string**

Examples in pseudocode:

Declare a variable called 'score' with a value of 10

```
score = 10
```

Declare a constant called 'PI' with a value of 3.14

```
const PI = 3.14
```

Examples in JavaScript:

```
let score = 10;
```

```
const PI = 3.14;
```

Examiner Tips and Tricks

- You might see code that declares variables like this

```
var variableName = value;
```

- This is bad practice as it utilises global variables but was used until 2015 when let was added to JavaScript


Your notes

Outputs in JavaScript


Your notes

Outputs in Javascript

There are several ways to produce outputs in **JavaScript**:

Changing the contents of an HTML element

- JavaScript can be used to modify the content of HTML elements by accessing them through the **Document Object Model (DOM)**. E.g.:

```
chosenElement = document.getElementById("example");

chosenElement.innerHTML = "Hello World";
```

- The first line of code uses the `document.getElementById()` method to **retrieve the HTML element with the ID "**`example`" from the document's DOM (Document Object Model)
- The returned element is assigned to the variable `chosenElement`, which allows further manipulation of that element
- The second line of code **modifies the content within the** `chosenElement` HTML element
- The `innerHTML` property allows direct access to the HTML content within an element
- In this case, the content of `chosenElement` is changed to "Hello World", effectively replacing any existing content with this new text

Writing directly to the document

- JavaScript can **write directly to the document** using the `document.write()` method. E.g.

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

- This code writes the text "Hello World" directly to the webpage

Using an alert box

- JavaScript provides the `alert()` function to **display a pop-up alert box** with a message. E.g.

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

- This code triggers an alert box with the message "Hello World"

Alert Box with the words 'Hello World'

Your notes

Your notes

Selection in JavaScript

- Programming code is made up of constructs which control the flow of a program
- Constructs tell the computer the order in which to carry out the statements/lines of code
- There are 3 programming constructs:
  - **Sequence**
  - Selection
  - **Iteration**
- **Selection is selecting a line/lines of code to run depending on whether a condition is true or false**
- There are two ways to write selection statements:
  - If statements
  - Switch case statements
- When writing a condition there will need to be a comparison operator. They are listed below:

Operator	Description	Example (where x=5)	Returns
==	equal to	x == 8	false
		x == 5	true
		x == "5"	true
===	equal value and equal type	x === 5	true
		x === "5"	false
!=	not equal	x != 8	true
!==	not equal value or not equal type	x !== 5	false

		x !== "5"	true
		x !== 8	true
>	greater than	x > 8	false
<	less than	x < 8	true
>=	greater than or equal to	x >= 8	false
<=	less than or equal to	x <= 8	true

Your notes

IF Statements in JavaScript

- An if the statement will let you **choose a line/lines of code to run if a condition is true or false**
- Below are three examples of if statements:

1. if
2. if else
3. if else if else

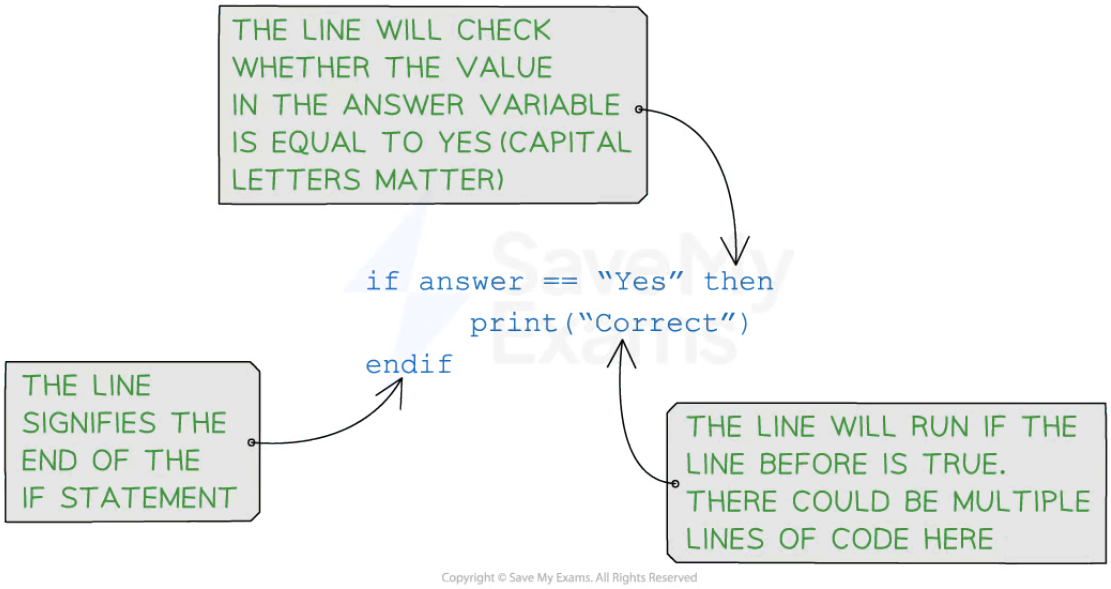
Syntax of an if statement

The syntax of an if the statement consists of the if keyword, followed by **a condition enclosed in brackets**, and **a code block that is executed** if the condition evaluates to true:

```
if (condition) {
  // Code to be executed if the condition is true
}
```

Pseudocode example of an if statement

Your notes



Example in JavaScript: checking if a number is positive

```
const number = 5;

if (number > 0) {
  console.log('The number is positive.');
}
```

- In this example, the `if` statement **checks if the value of the variable** `number` is greater than `0`. If the condition is true, the message `'The number is positive.'` is output to the browser

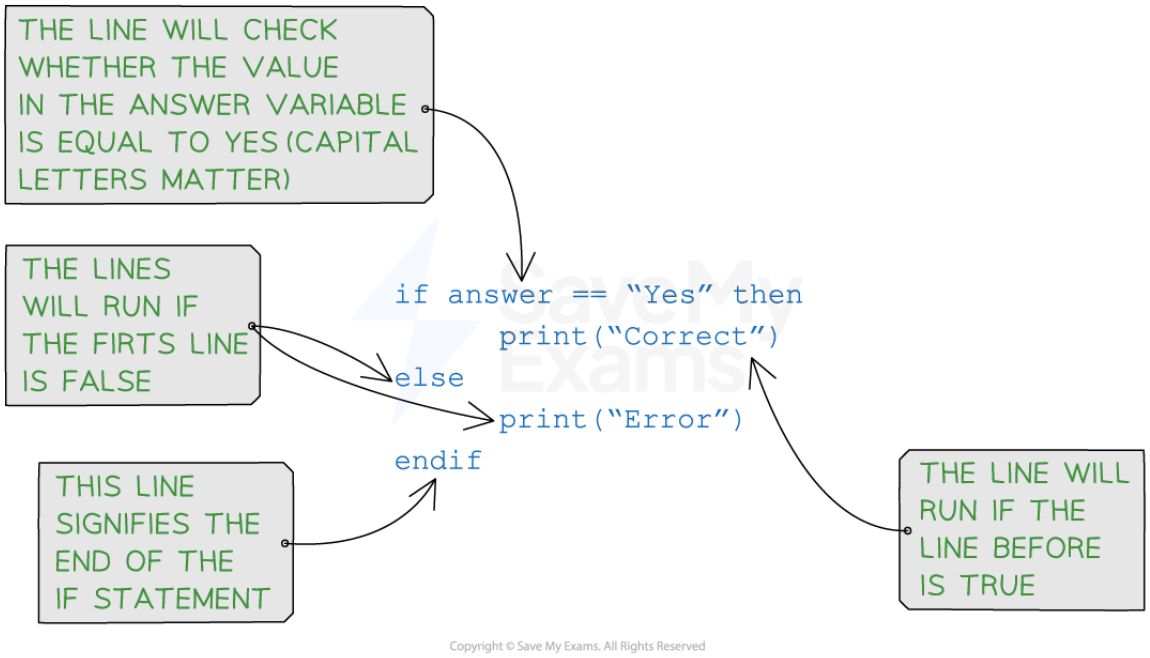
Syntax of an if-else statement

The `if the` statement can be extended with an `else` clause to specify an **alternative block of code that is executed when the condition evaluates to** `false`:

```
if (condition) {
  // Code to be executed if the condition is true
} else {

// Code to be executed if the condition is false

}
```

Pseudocode example of an if-else statement

Your notes



Example in JavaScript: Checking if a number is positive or negative

```
const number = -3;

if (number > 0) {
  console.log('The number is positive.');
} else {
  console.log('The number is not positive.');
}
```

- In this example, **if the value of** number is greater than 0, the message 'The number is positive.' is output. Otherwise, the message 'The number is not positive.' is output

Syntax of an If Else-If Else statement

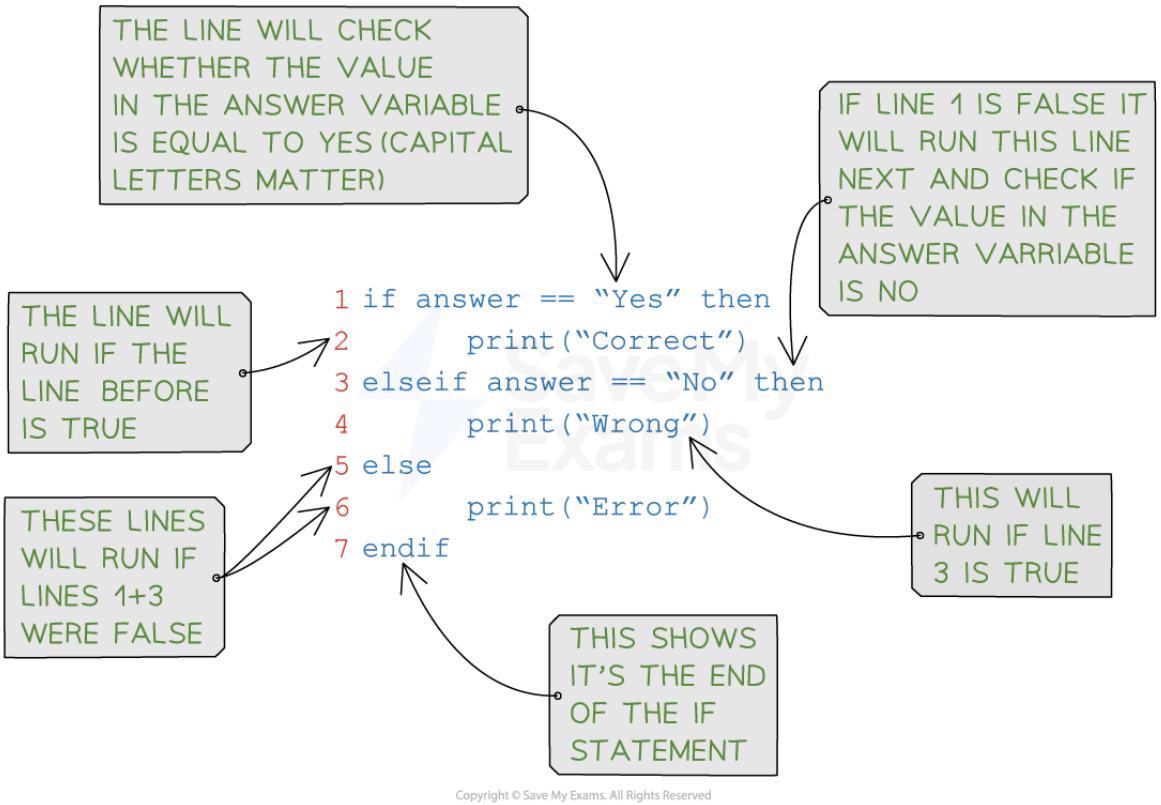
The else if the clause **specifies additional conditions to check if the initial** if **condition is false**. This allows the handling of multiple scenarios in a more complex decision-making process:

```
if (condition) {
  // Code to be executed if the condition is true
} else if (condition) {

  // Code to be executed if the condition is true
} else {
```

```
// Code to be executed if the condition is false

}
```

Your notes

Pseudocode example of an If Else-If Else statement



Example in JavaScript: Grading a score

```
const score = 85;

if (score >= 90) {
  console.log('Excellent!');
} else if (score >= 80) {
  console.log('Good.');
} else if (score >= 70) {
  console.log('Fair.');
} else {
  console.log('Needs improvement.');
}
```

- In this example, the if-else if-else statement **evaluates the value of** score to determine the corresponding message based on the score range

Your notes

Examiner Tips and Tricks

- You can use as many else ifs as you want to within your if statement but it might be clearer to use a switch case statement
- You can have an if else-if statement without an end as the catch-all condition at the end

How do I write my condition?

- Think of it like **writing a yes/no question**
- E.g.
  - Is the number bigger than 10?
  - Is the number between 50 and 100?
  - Is the answer Paris?
- If the question can't be answered with yes/no then it needs to be rewritten in this way

Is it possible to use more than one operator?

- Yes! **Using one operator is most common but sometimes 2 are needed**. More than 2 could be used but it gets more complicated. This involves using boolean operators
- Imagine a program where the user has to enter a number based on the role of a dice. **The number needs to be between 1 and 6**
- The yes/no question could be **is the number between 1 and 6** - but it would be structured slightly differently in the code
- `IF number >=1 AND number <=6 then`
- The first check is **if the number is greater than or equal to 1**
- The second check is **if the number is less than or equal to 6**
- The final check is if **both sides are True**

Examiner Tips and Tricks

- Check whether to use an AND or an OR in the condition - it's easy to get these 2 mixed up. E.g.
  - IF number<0 AND number>100 will check if the number is below 0 and greater than 100 (there are no numbers which fit into this range!)
  - IF number<0 OR number>100 will check if the number is between 1 and 99

Your notes

Worked Example

A website sells tickets for sporting events. The website uses HTML, CSS and JavaScript. The website charges a booking fee of £2.99 on each ticket sold. In addition, if the tickets are purchased from outside of the UK, £4.99 is added to the booking fee. The booking fee is calculated using a JavaScript function named bookingfee().

Complete the definition of the bookingfee() function below.

```
function bookingfee(numtickets, country) {
  var nonUKprice = 4.99;
  var perTicketPrice = .....;
  var total = 0;
  if (country!="UK") {
    total = total + .....;
  }
  total = total + (..... * perTicketPrice);
  return total;
}
```

3 marks

How to answer this question:

- The first blank space is to set the value of the perTicketPrice. The question tells us this is £2.99
- The 2nd blank space is to add something to total if the country is not equal to UK. The question tells us that if the tickets are purchased from outside the UK, £4.99 is added to the booking fee. This is stored in the nonUKprice variable
- The 3rd blank space is something multiplied by perTicketPrice. The question tells us each ticket is £2.99 so we need to multiply the perTicketPrice by the number of tickets (called numtickets)

Answer:

Example answer that gets full marks:

```
function bookingfee(numtickets, country) {
  var nonUKprice = 4.99;
  var perTicketPrice = 2.99;
  var total = 0;
  if (country!="UK") {
```

```
    total = total + nonUKprice;
  }
  total = total + (numtickets * perTicketPrice);
  return total;
}
```

Examiner Tips and Tricks

- When you're given code in a question that you need to refer to or complete, you must spell existing identifier names exactly as they have in the question. E.g. if you write numtikets instead of numtickets you won't get the mark
- Also, make sure you don't include £ in the calculation - this will be added as an output and will cause the calculation to not work as it's included a character that's not an integer

Switch Case in JavaScript

- Switch Case is a type of conditional statement that **provides an alternative way to perform multiple comparisons** based on the value of an expression
- These statements are particularly useful when you have **a single expression that you want to compare against multiple possible values**

Syntax of a Switch Case statement

- The syntax of a switch case statement consists of the switch keyword followed by an expression enclosed in brackets
- This expression is evaluated, and its value is then compared against various case labels. **If a match is found, the corresponding block of code is executed**
- The default keyword is optional and specifies a **block of code to be executed if none of the case labels match** the expression:

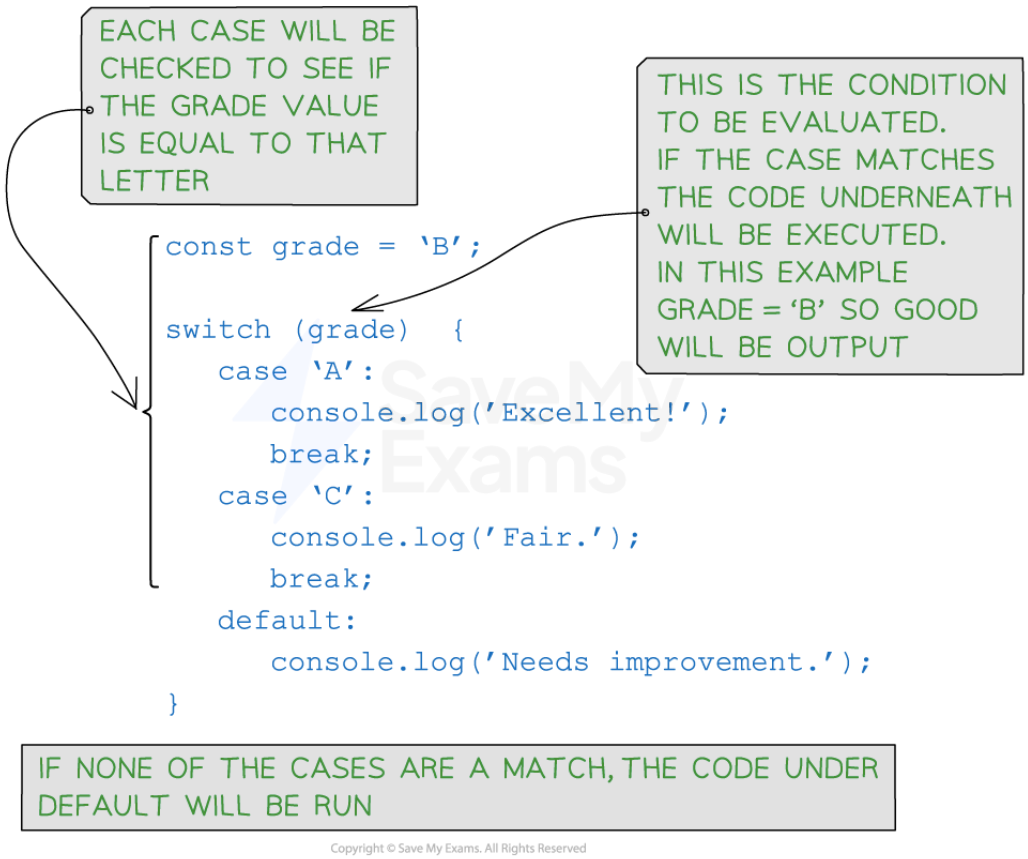
```
switch (expression) {
 case value1:
  // Code to be executed if the expression matches value1
  break;
 case value2:
  // Code to be executed if the expression matches value2
  break;
 // more case statements
 default:
```


Your notes

```
    // Code to be executed if no case matches the expression
}
```

Your notes

Switch case example in JavaScript: Grades evaluation



Fall-Through behaviour

By default, switch case has a fall-through behaviour, meaning that if a case label matches, **the code execution continues to the next case until a** break statement is encountered. This allows you to group multiple cases with the same code:

```
const day = 'Monday';

switch (day) {
  case 'Monday':
  case 'Tuesday':
  case 'Wednesday':
  case 'Thursday':
```

```
  case 'Friday':
    console.log('Weekday');
    break;
  case 'Saturday':
  case 'Sunday':
    console.log('Weekend');
    break;
  default:
    console.log('Invalid day.');
}
```

- In this example, if day is 'Tuesday', 'Weekday' is logged to the console because it falls through the cases of 'Monday' to 'Friday' until a break statement is encountered


Your notes

Your notes

For Loops in JavaScript

A for loop is a control flow statement that **allows a block of code to repeatedly execute for a specified number of iterations**. It provides a concise and structured way to perform repetitive tasks.

Syntax of a for loop

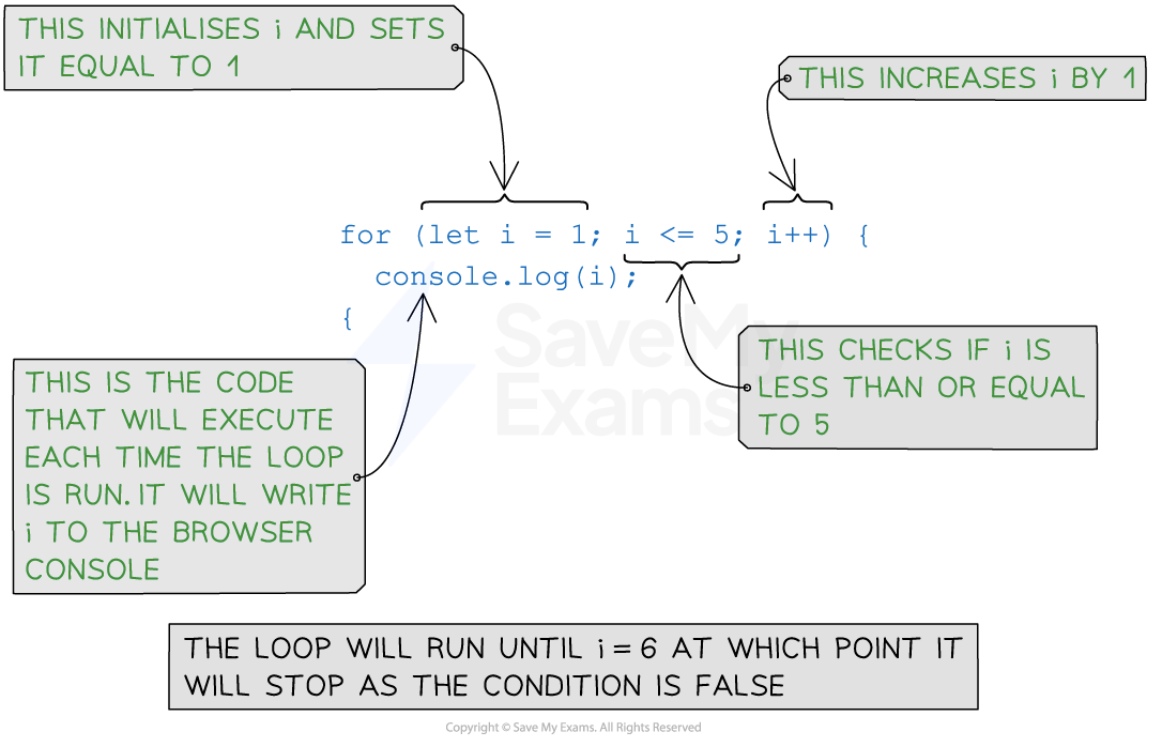
The syntax of a for loop consists of three main parts:

```
for (initialisation; condition; increment/decrement) {
  // Code to be executed in each iteration
}
```

1. **Initialisation**: The initialisation is executed only once at the beginning of the loop. It is used to initialise a counter variable that controls the loop's execution
2. **Condition**: The condition is evaluated before each iteration. If the condition evaluates to true, the loop continues executing the code block. If the condition evaluates to false, the loop terminates
3. **Increment/Decrement**: The increment or decrement statement is executed at the end of each iteration, updating the counter variable to control the loop's progress

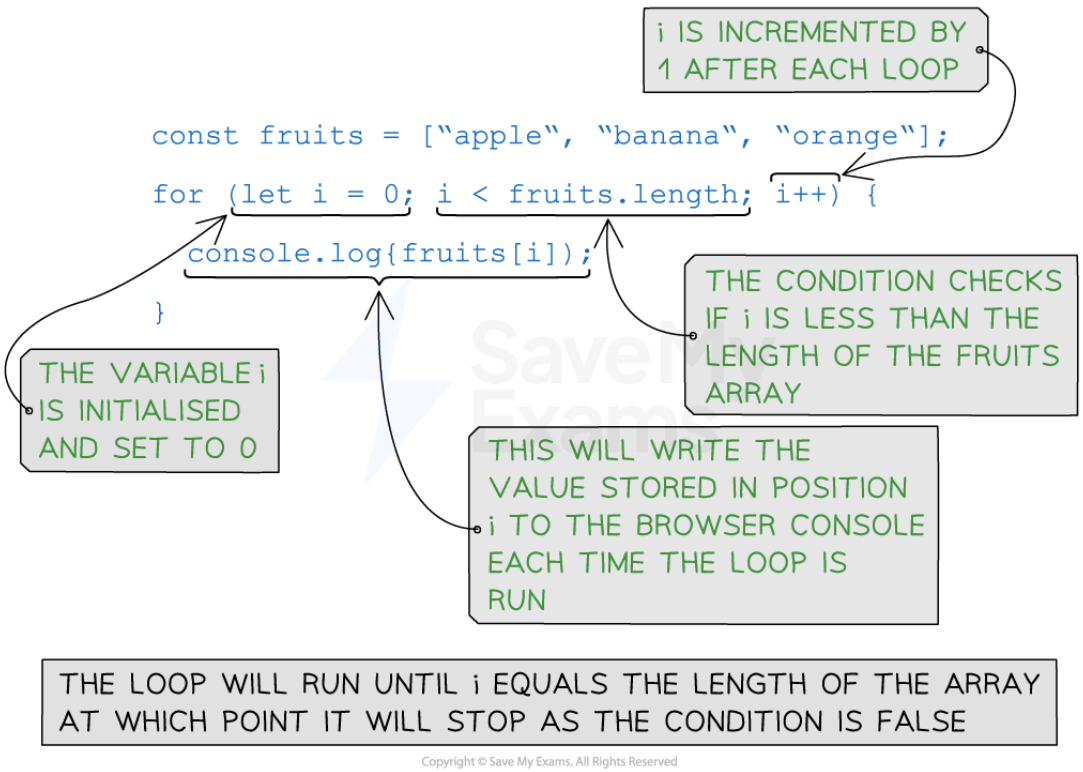
Example 1: Counting from 1 to 5

Your notes



Example 2: Iterating over an array

Your notes



Examiner Tips and Tricks

- You might have seen `i=i+1` or `i+=1` for incrementing by 1. This is the same as `i++` in JavaScript

For In Loops in JavaScript

- The `for in` the loop **iterates through the items in a data structure like a list or array**

```
const fruits = ['apple', 'banana', 'orange', 'grape'];

for (let index in fruits) {
  console.log('Index: ' + index + ', Value: ' + fruits[index]);
}
```

- The list `fruits` contains four items: `'apple'`, `'banana'`, `'orange'`, and `'grape'`
- The `for...in` the loop **iterates over each index of the** `fruits` list
- In each iteration, the `index` the variable is assigned the current index value

- Inside the loop, we use fruits[index] to **access the value associated with the current index**
- The loop executes the code block, which outputs the index and value of each item in the list to the browser


Your notes

While Loops in JavaScript

Your notes

While Loops in JavaScript

- A while loop is a control flow statement that **allows a block of code to repeatedly execute as long as a specified condition remains true**
- While loops provide a flexible and powerful way to handle situations **where the number of iterations is unknown in advance**

Syntax of a while loop

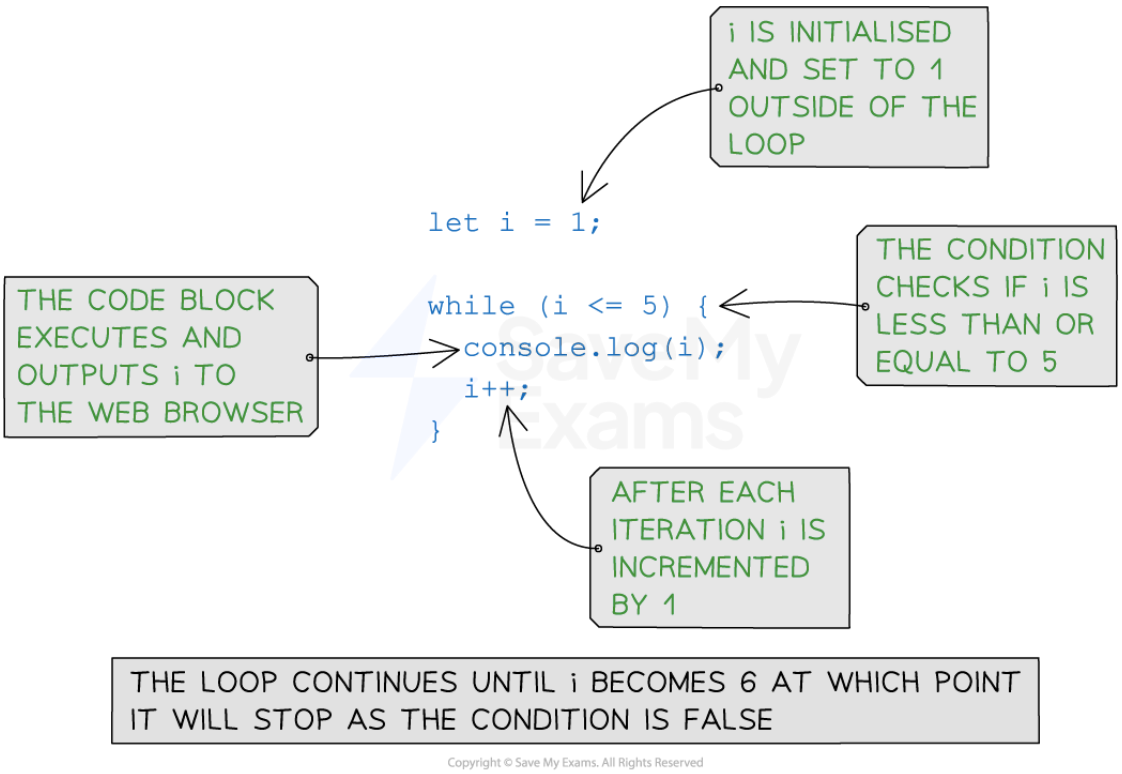
The syntax of a while the loop consists of a condition and a code block:

```
while (condition) {
  // Code to be executed as long as the condition is true
}
```

- The condition is evaluated before each iteration. If the condition evaluates to true, the code block is executed. If the condition evaluates to false, the loop terminates

Example 1: Counting from 1 to 5

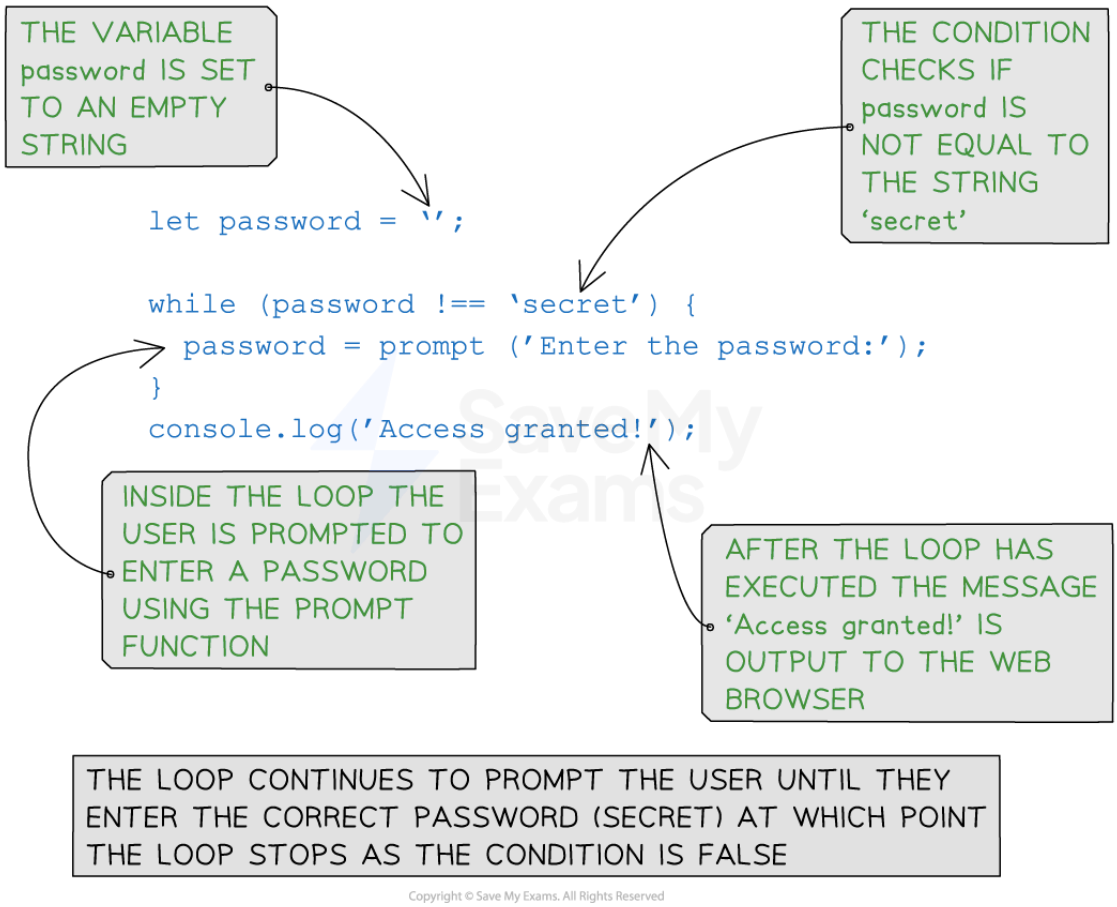
Your notes



While loop in JavaScript counting from 1 to 5

Example 2: Checking if the password is 'secret'

Your notes



While loop in JavaScript checking if the password is correct

Do While Loops in JavaScript

- A do while loop is a control flow statement that **allows a block of code to repeatedly execute at least once, and then continue execution as long as a specified condition remains true**
- Do while loops provide a variation of the while loop with slightly different behaviour

Syntax of a do while loop

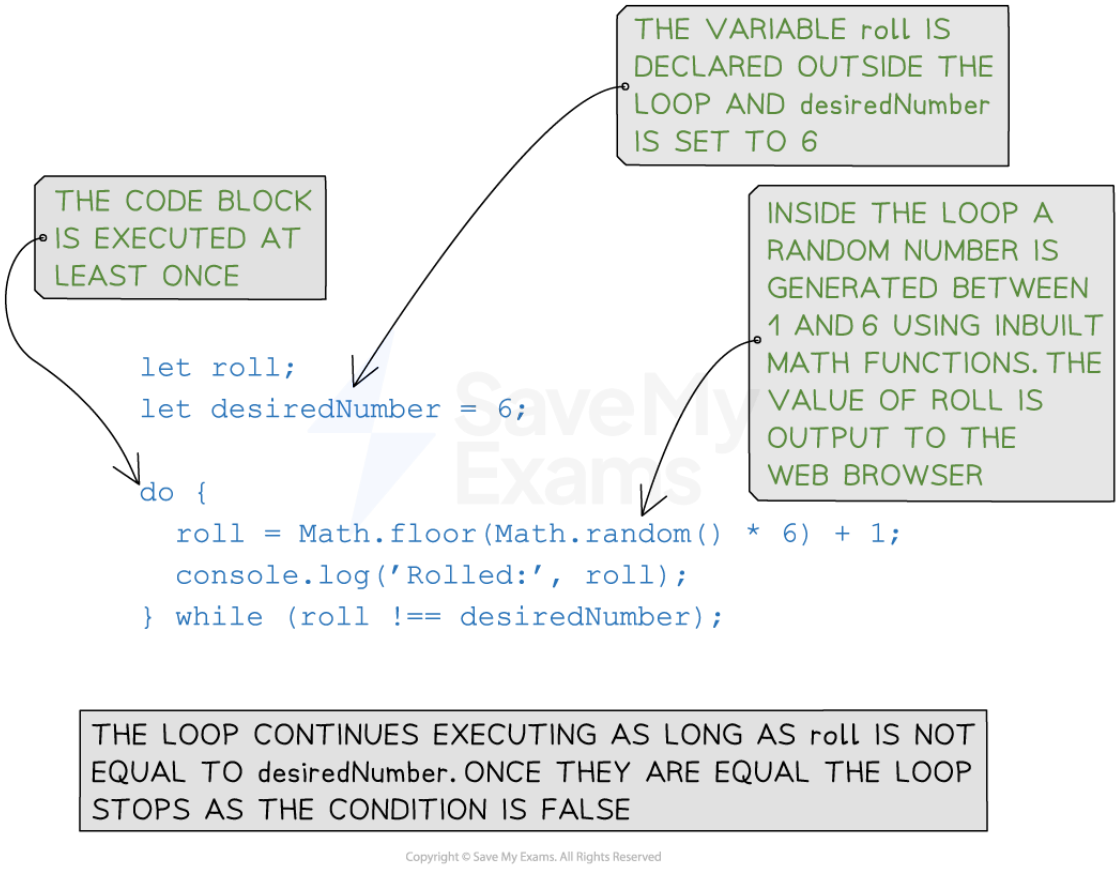
The syntax of a do while the loop consists of a code block and a condition:

```
do {
  // Code to be executed at least once
} while (condition);
```

- **Code Block**: The code block is executed first before evaluating the condition
- **Condition**: The condition is evaluated after executing the code block. If the condition evaluates to true, the loop continues executing. If the condition evaluates to false, the loop terminates

Your notes

Example: Rolling a die until a desired number is obtained



Do while loop in JavaScript - a dice roll repeats until the target number is met

Examiner Tips and Tricks

- You can use either a while loop or a do while loop but don't forget that a do while loop will always run once before checking if the condition is true


Your notes

Strings in JavaScript

A string is a sequence of characters enclosed in single quotes (') or double quotes ("). Strings are widely used for representing and manipulating text-based data.

Creating strings

- Strings can be created in JavaScript by enclosing text within quotes:

```
const message = 'Hello, world!';
const name = "John Doe";
```

String length

- To **find the length of a string**, use the `length` property
- The `length` property **returns the number of characters** in a string:

```
const message = 'Hello, world!';
console.log(message.length);

// Output: 13
```

- In this example, the `length` property is used to determine the number of characters in the `message` string

Substring extraction

- To extract a portion of a string, use the `substring()` method
- The `substring()` method takes **two parameters: the starting index and the ending index (optional)**
- A new string is returned containing the characters within the specified range:

```
const message = 'Hello, world!';
const substring = message.substring(0, 5);
console.log(substring);

// Output: Hello
```

- In this example, the `substring()` method is used to extract the characters from index 0 to index 5 (exclusive) from the `message` string, resulting in the substring `'Hello'`
- By **omitting the second parameter**, the `substring()` method will **extract the characters from the starting index to the end of the string:**

```
const message = 'Hello, world!';
const substring = message.substring(7);
console.log(substring);

// Output: world!
```

- Here, the `substring()` method extracts the characters from index 7 to the end of the `message` string, resulting in the substring `'world!'`

Examiner Tips and Tricks

- JavaScript allows flexibility in choosing single or double quotes to define strings, as long as you maintain consistency within a string
- There are many other things you can do with strings in JavaScript but these are the only ones you need to know before the exam
- The exam might give you additional string functionality but if it does it will provide the code and explain it

Your notes

Operators in JavaScript


Your notes

Arithmetic Operators in JavaScript

Arithmetic operators are symbols or keywords used to **perform mathematical calculations** and operations on numerical values. They allow the computer to perform **addition, subtraction, multiplication, division**, and more.

Addition operator (+)

The addition operator (+) is used to **add numerical values together** or **concatenate** strings:

```
const sum = 5 + 3;

const fullName = 'John' + ' ' + 'Doe';
```

- In this example, the addition operator adds the values 5 and 3, resulting in 8
- It also concatenates the strings 'John', ' ', and 'Doe' to form the full name 'John Doe'

Subtraction operator (-)

The subtraction operator (-) is used to **subtract one numerical value from another**:

```
const difference = 10 - 4;
```

- In this example, the subtraction operator subtracts the value 4 from 10, resulting in 6

Multiplication operator (*)

The multiplication operator (*) is used to **multiply numerical values together**:

```
const product = 3 * 5;
```

- In this example, the multiplication operator multiplies the values 3 and 5, resulting in 15

Exponentiation operator (**)

The exponentiation operator (**) is used to **multiply numerical values by the power of another number**:

```
const product = 3 ** 3;
```

- In this example, the exponentiation operator multiplies the value 3 to the power 3, resulting in 27

The division operator (/)

The division operator (/) is used to **divide one numerical value by another**:

```
const quotient = 10 / 2;
```

- In this example, the division operator divides the value 10 by 2, resulting in 5


Your notes

The modulus operator (%)

The modulus operator (%) **returns the remainder when one numerical value is divided by another**:

```
const remainder = 10 % 3;
```

- In this example, the modulus operator divides the value 10 by 3 and returns the remainder, which is 1

Increment operator (++) & decrement operator (--)

The increment operator (++) and decrement operator (--) are used to **increase or decrease the value of a numerical variable by one**:

```
let counter = 5;
counter++;
counter--;
```

- In this example, the increment operator increases the value of counter by one, resulting in 6
- The decrement operator decreases it back to 5

Operator precedence (BIDMAS / BODMAS)

Arithmetic operators follow the rules of operator precedence, which determine **the order in which operators are evaluated**. Parentheses () can be used to control the order of evaluation:

```
const result = 2 + 3 * 4;
const adjustedResult = (2 + 3) * 4;
```

- In the first example, without parentheses, the multiplication (*) is performed before the addition (+), resulting in 14
- In the second example, with parentheses, the addition is evaluated first, resulting in 5, which is then multiplied by 4, resulting in 20

Logical Operators in JavaScript

Logical or comparison operators are symbols or keywords used to **compare values and return a boolean result**. They allow a comparison of variables, or expressions to determine relationships, equality, or inequality between them.

Equal to (==) operator

The equal to the operator (==) **compares two values and returns** true if they are equal, and false otherwise. It performs type coercion, meaning it converts the operands to a common type before comparing them:

```
const x = 5;
const y = '5';

console.log(x == y);
```

- In this example, the equal to operator compares the value `5` (a number) with the value `'5'` (a string). Since JavaScript performs type coercion, the operands are considered equal, resulting in `true`

Not equal to (!=) operator

The not equal to the operator (`!=`) **compares two values and returns** `true` if they are not equal, and `false` if they are equal. Like the equal to operator, it performs type coercion:

```
const x = 5;
const y = '5';

console.log(x != y);
```

- In this example, the not equal to operator compares the value `5` with the value `'5'`. Since JavaScript performs type coercion and considers the operands equal, the result is `false`

Strict equal to (===) operator

The strict equal to the operator (`===`) **compares two values and returns** `true` if they are equal in both value and type. Unlike the equal to operator, it does not perform type coercion:

```
const x = 5;
const y = '5';

console.log(x === y);
```

- In this example, the strict equal to operator compares the value `5` with the value `'5'`. Since the operands have different types, the result is `false`

Strict not equal to (!==) operator

The strict is not equal to the operator (`!==`) **compares two values and returns** `true` if they are not equal in either value or type. It does not perform type coercion:

```
const x = 5;
const y = '5';

console.log(x !== y);
```

- In this example, the strict not equal to the operator compares the value `5` with the value `'5'`. Since the operands have different types, the result is `true`

Greater than (>) & less than (<) operators


Your notes


Your notes

The greater than the operator (>) **compares two values and returns** true if the left operand is greater than the right operand. The less than operator (<) **returns** true if the left operand is less than the right operand:

```
const x = 5;
const y = 10;

console.log(x > y);
console.log(x < y);
```

- In this example, the greater than operator compares the value 5 with the value 10, resulting in false
- The less than operator compares 5 with 10, resulting in true

Greater than or equal to (>=) & less than or equal to (<=) operators

The greater than or equal to the operator (>=) **compares two values and returns** true if the left operand is greater than or equal to the right operand. The less than or equal to the operator (<=) **returns** true if the left operand is less than or equal to the right operand:

```
const x = 5;
const y = 10;

console.log(x >= y);
console.log(x <= y);
```

- In this example, the greater than or equal operator compares 5 with 10, resulting in false
- The less than or equal to operator compares 5 with 10, resulting in true

Boolean Operators in JavaScript

Boolean operators are symbols or keywords used to **combine and manipulate boolean values**. They allow the computer to **perform logical operations, such as combining conditions, negating values, and determining whether an expression is true or false**.

AND operator (&&)

The AND operator (&&) **returns** true if both operands are true, otherwise returns false:

```
const x = 5;
const y = 10;
const z = 15;

const result = (x < y) && (y < z);
```

- In this example, the expression (x < y) && (y < z) evaluates to true because both conditions are true. If any of the conditions were false, the result would be false

OR operator (||)

The OR operator (||) **returns** true if at least one of the operands is true, and false if both operands are false:

```
const a = 5;
const b = 10;
const c = 15;

const result = (a > b) || (b < c);
```

- In this example, the expression (a > b) || (b < c) evaluates to true because the second condition is true, even though the first condition is false. If both conditions were false, the result would be false

NOT operator (!)

The NOT operator (!) is **used to negate a Boolean value**. It returns true if the operand is false, and false if the operand is true:

```
const value = false;

const result = !value;
```

- In this example, the ! operator negates the value of false, resulting in true


Your notes

Nested Statements in JavaScript

Your notes

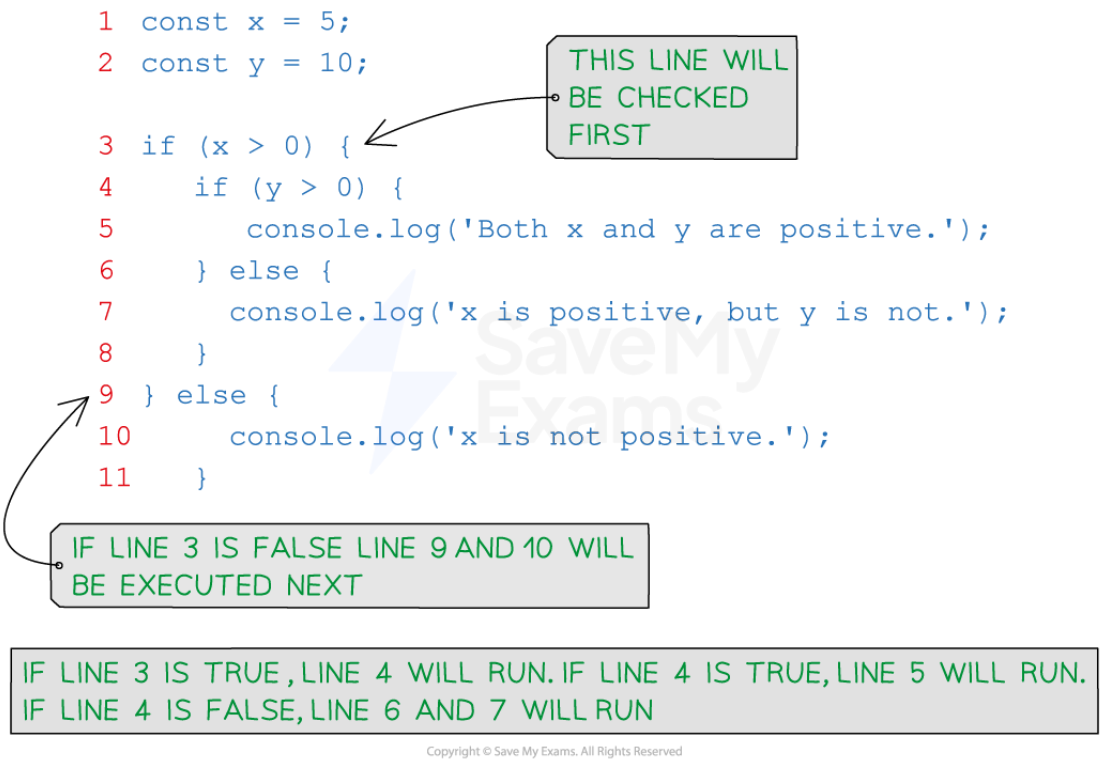
Nested Statements in JavaScript

Nesting is **putting one block of code within another.** This is commonly done with if statements and loops.

- Nested if statements refer to the practice of including one or more conditional statements inside the block of another conditional statement. These nested structures allow you to **create more complex decision-making logic by evaluating multiple conditions** and handling various scenarios within your code
- Nested loops refer to the practice of including one or more loops inside the block of another loop. These nested structures allow you to **iterate over multiple sets of data** and create more complex patterns of repetition within your code

Nested If Statements in JavaScript

If statements can be nested within each other to handle more intricate conditions and decision-making:



Nested if statement in JavaScript

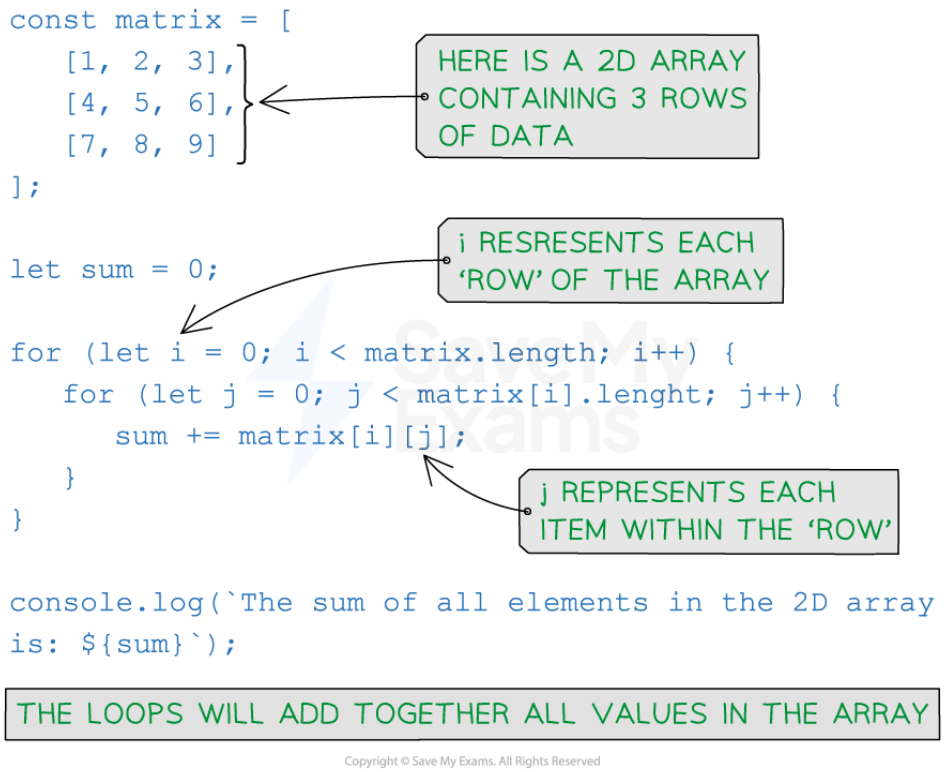
Nested Loops in JavaScript

Your notes

The syntax of nested loops involves placing one loop (for loop, while loop, etc.) inside the block of another loop:

```
for (let i = 0; i < outerArray.length; i++) {
 // Code block for outer loop

 for (let j = 0; j < innerArray.length; j++) {
  // Code block for inner loop

  // More nested loops if needed
 }

 // More code after inner loop
}
```

Example: Nested for loops



Nested for loop in JavaScript showing a 2D array with each item in the array being added together to find the sum of the whole array

- After the first iteration sum=1 as it's only added the 1st element which is 1
- After the 2nd iteration sum=3 as it's added the 2nd element (2) to 1
- After the 3rd iteration sum=6 as it's added the 3rd element (3) to the 1 and 2
- After the 4th iteration sum=10 as it's added the 4th element (4) to the 1, 2 and 3
- The algorithm continues working its way through the rest of the 2nd row and then moves on to the 3rd


Your notes

Example: Nested while loops

This code below does the same as the code above but utilises while loops rather than for loops:

```
const matrix = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]
];

let i = 0;
while (i < matrix.length) {
  let j = 0;
  while (j < matrix[i].length) {
    console.log(matrix[i][j]);
    j++;
  }
  i++;
}
```

Examiner Tips and Tricks

- If you need to check or work out how many times an inner loop is run, multiply the number of outer loop iterations by the number of inner loop iterations. E.g.

```
for (let i = 0; i < 3; i++) {
  for (let j = 0; j < 5; j++) {
    console.log(i * j);
  }
}
```

- The outer loop runs 3 times and the inner loop will run 5 times. To find out the total number of times the inner loop will run, multiply 3 by 5 = 15

Your notes

Functions & Procedures in JavaScript

- Functions and procedures are essential building blocks that allow you to **enclose blocks of code and execute them as needed**
- They promote code **reusability, modularity, and organisation**, enabling a programmer to write efficient and maintainable programs

Considerations and Best Practices

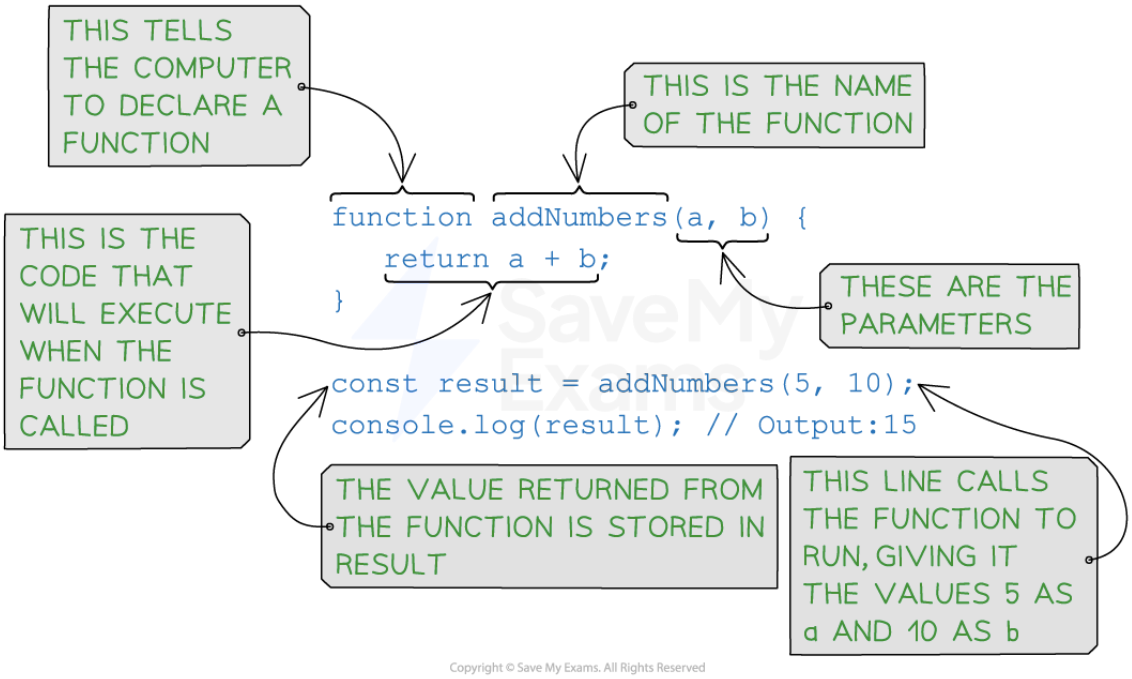
- **Naming**: Choose descriptive and meaningful names for your functions and procedures that indicate their purpose
- **Parameter Names**: Use clear and meaningful parameter names to improve code readability
- **Function Length**: Aim for functions and procedures that are short and focused
- **Return Values**: Functions should have explicit return statements with meaningful return values, while procedures should not have return statements

Functions in JavaScript

A function is a **reusable block of code that performs a specific task or calculation** and can be **called** from anywhere in the code. Functions can take input parameters (arguments) and **return a value**.

The syntax for defining a function is as follows:

```
function functionName(parameter1, parameter2) {
  // Code block to perform the task
  // Return value;
}
```



Function in JavaScript

Your notes

Worked Example

A website sells tickets for sporting events. The website uses HTML, CSS and JavaScript. The website charges a booking fee of £2.99 on each ticket sold. In addition, if the tickets are purchased from outside of the UK, £4.99 is added to the booking fee. The booking fee is calculated using a JavaScript function named bookingfee().

Complete the definition of the bookingfee() **function below.**

```
function bookingfee(numtickets, country) {
  var nonUKprice = 4.99;
  var perTicketPrice = .....;
  var total = 0;
  if (country!="UK") {
    total = total + .....;
  }
  total = total + (..... * perTicketPrice);
  ..... total;
}
```

4 marks

How to answer this question:

- The first blank space is to set the value of the perTicketPrice. The question tells us this is £2.99
- The 2nd blank space is to add something to total if the country is not equal to UK. The question tells us that if the tickets are purchased from outside the UK, £4.99 is added to the booking fee. This is stored in the nonUKprice variable
- The 3rd blank space is something multiplied by perTicketPrice. The question tells us each ticket is £2.99 so we need to multiply the perTicketPrice by the number of tickets (called numtickets)
- The 4th blank space is the last line of the function. As it's a function, a value must be returned. A value hasn't yet been returned in this function so total must be the value returned

Answer:

Example answer that gets full marks:

```
function bookingfee(numtickets, country) {
  var nonUKprice = 4.99;
  var perTicketPrice = 2.99;
  var total = 0;
  if (country!="UK") {
    total = total + nonUKprice;
  }
  total = total + (numtickets * perTicketPrice);
  return total;
}
```

Your notes

Procedures in JavaScript

- A procedure is similar to a function but **does not return a value**. Instead, it **performs a series of actions or operations** which could be anything the programmer wants the procedure to execute
- **A procedure is essentially a function without a** return **statement** or with a return statement that has no value to return

The syntax is the same as for functions:

```
function procedureName(parameter1, parameter2) {
  // Code block to perform actions
  // No return statement or return with no value;
}
```