Java – secure, enterprise, safe, stable, android, planes

C++ - math

Go – google, web

Python – data, excel files

Cobol – old banks

Swift – apple

Javascript – used on most websites, a lot of traction in market

All use American English, can switch in a matter of weeks

Frontend developers – what you see and can interact with. Creative. Always start here.

Backend developers – server side, the cloud, make sure videos fast when play them, give good recommendations, make sure card details safe. Analytical.

Full-stack engineer – do not get paid 2x as much and it takes 2x to learn

HTML, CSS, JS – web understands this

HTML and CSS is not a programming language

Html – headlines, structure of page. First layer (bones)

CSS – add styling, colours, font size, spacing. Second layer (skin – what we see)

JS – invisible layer that makes everything work

Softwares

Coding editors like word, assists us like when make spelling errors on word.

VS code – made by Microsoft, free, works on windows/mac

Chrome is best browser to check code. Used by 65% of population, can see what code was used on websites

GitHub – owned by Microsoft, stores code, is like youtube. Can promote yourself on there by making it open source. Hire engineers by doing this

H1 = heading one

P = paragraph

Img = image

Src = source

Br = line break

A = link

Href = hyperlink reference

Button

Framework:

* Make code more maintainable
* Bullstrap used by twitter – makes look good on phones, tablets, TV. Open sourced

React – javascript, made by FB, open sourced, JS gets messy quick so is good.

Google trends, US is trendsetter, San franscisco esp

Junior react developers well paid.

API = allow us to talk to other applications with one line of code. JS

Udemy, freecodecamp, youtube

**PYTHON**

Trinket IDE used

**Print statement** – display/output information to the user. Print command + one/more arguments (values).

print(“word”)

Statement = command + argument

Syntax = set of rules that govern the structure of sentences, spelling and grammar rules

Debugging = resolving errors in code

**Variables** = store information. Used in calculations to hold values that can be changed. Use descriptive variables that reveal what their functions are and what content they store.

Naming rules – start with a letter or an underscore, remainder of name can consist of letters/ numbers/ underscores, case sensitive

Variable types

* String – combo of characters, used to store surname/ name/ address of a person
* Integer – whole number, positive or negative, used for counting or simple calculations e.g., store no of items want to buy or no of students in a class
* Float – decimal numbers/ fractional component, useful for precision, e.g., storing measurements for building or money
* Boolean – can only store true or false

Data type – in python do not have to declare the data type of the variable, known as “weak-typing”, bc python detects the variables data type by reading how data is assigned to the variable

* Strings – detected by quotation marks
* Integers – detected by lack of quotation marks and presence of whole numbers
* Floats – detected by presence of decimal point numbers

variable\_name = value\_you\_want\_to\_store

e.g., num = 2. Variable named num is assigned 2 so when type num the program retrieves the value 2 that is stored there

Variables to hold user input: name=input(“Enter your name:”)

The variable ‘name’ stores what the user entered as a String.

To display a message that contains data stored in the variable, use the print commant: print(“Hi there” + name)

**Working with numerical data**

Modulus = %

BIDMAS

expression2=(5+6)\*2

print(“expression 2 is “+str(expression2))

<https://docs.python.org/3/library/math.html>

**Casting** = taking data and converting into different types for different purposes, aka type conversion

str() converts an argument to a string

int() converts an argument into an integer

float() converts an argument to a float

Example:

number = int(input("Please enter a number"))

answer = number \* 10

print(str(number) + " x 10 = " + str(answer))

* Int() used to convert the results of the input function from a string to an int value
* The print() function can only work with string arguments
* By default anything entered into an input is a string

**Working with strings**

Strings allow us to communicate with the user.

String = sequence of characters (letters, numerals, symbols, special characters), vast range of values can be stored. Must be written with quotation marks to be correctly identified.

e.g., name = “John Doe”

phone numbers are stored as strings as often start with single leading zero and contain symbols.

Multi-line string = use triple quotes

Concatentation (adding) of strings = to form a sentence/longer word, use +

Name = “Peter”

Surname = “Parker”

FullName = Name + “” + Surname

(“” puts space between the names)

* Cannot add a string and non-string together, must convert the non-string
* When convert number to string, cannot use it in calculations

**How strings are stored (indexing)**

Forward indexing = each character incl blanks is indexed by numbers (0 for first character on left and moves to right)

Backward indexing = characters indexed from right to left using negative numbers (-1 is rightmost)

word = "Hello"

print("Example 2: ")

# Indexing from 0 to 4

char1 = word[0]

char2 = word[1]

char3 = word[2]

char4 = word[3]

char5 = word[4]

print(char1)

print(char2)

print(char3)

print(char4)

print(char5)

print("")

# Indexing from -1 to -5

char1 = word[-1]

char2 = word[-2]

char3 = word[-3]

char4 = word[-4]

char5 = word[-5]

print(char1)

print(char2)

print(char3)

print(char4)

print(char5)

Alternatively:

word = input("Enter a 6 letter word ")

print("Your word backwards is: ")

print(word[-1])

print(word[-2])

print(word[-3])

print(word[-4])

print(word[-5])

print(word[-6])

len() function = gets no. of characters in a string/length of a string. Includes punctuation and spaces

print(len(“Hello World!”)

slicing a string = extracts characters from string based on a starting index and ending index.

e.g., Print(aString[3:7])

This example starts at index 3 and ends at position 6 (Actually think its 4 then 7?)

* Can omit either or both of the indices
* If first index is omitted it defaults to 0, so your chunk starts from the beginning of the original string
* If second index omitted it defaults to the highest index in the stirng, so chunk ends at the end of the original string

Extended slice = can print a string in reverse using [::-1]. The syntax is [begin: end: step]. By not including a beginning and end and specifying a step of -1, the string is reversed.

<https://docs.python.org/2.3/whatsnew/section-slices.html>

\*if print(aString[2:]) it would start at the third letter.

**Code readability**

Comments = add context to your code making it easier to understand, isn’t read by the computer. Start with hash character and extend to the end of the physical line

<https://peps.python.org/pep-0008/#comments>

Naming variables - *myName* and *string\_name* are examples of descriptive variables as they reveal what their functions are and what content they store.

**Useful string functions**

aString = “Hello World”

print(aString.upper()) = HELLO WORLD

print(aString.lower()) = hello world

aSentence = “Hello$World”

print(aSentence.replace(“$”,””)) = Hello World

* Replaces any occurrence of a string with a string of your choice

strHelp = “\*\*Hello\*\*”

print(strHelp.strip(‘\*’)) = Hello

* Removes any unwanted substring from the start and end of a string

**Algorithms**

Express solutions clearly and accurately. Set of instructions showing how to solve a particular problem.

Before implementing code, pseudocode and flowcharts used to design envisioned solution to problem.

Pseudocode = informal description or what algorithm must do, for human reading, does not obey syntax rules so understood by any programmer

*E.g.,*

*Request integer from user*

*If integer is even print “fizz”*

*Else if the integer is odd*

*Print “buzz”*

Flowcharts:A diagram of a diagram

Description automatically generated with medium confidence

Criteria = must have input and output, clarity (never ambiguous), correctness/generality (correctly solve problems), efficient

**If statements (making decisions)**

e.g.,

num=10

if(num < 12):

print(“the variable num is lower than 12”)

>= greater than or equal to

<= less than or equal to

== equals

!= not equals

Even number:

if num % 2 == 0:

print (“fizz”)

else:

print(“buzz”)

**While loops**

Loop statements – used for reducing lengthy code, preventing coding errors, paving way towards code reusability

E.g.,

num = 1

While num < 11:

print(num)

num += 1

* While num <11 the value of num will be printed, so in the first time the loop runs the number 1 will be printed
* Num+=1 is shorthand for num = num + 1
* So after we have printed 1, num then becomes equal to 2 (num = 1+1)

**Lists**

Lists = sequence types, behave like an ordered collection of items, store a collection of any data type. Separated by commas between square brackets. Just like strings, each item can be accessed via its index (0 is first element)

groceries = [“apples”, “milk”, “cheese”]

**.append() and .insert() allows us to add items**

**.pop() and .remove() allows us to delete items**

Example

groceries = ["apples", "milk", "cheese", "bread"]

groceries. append("coffee") # ["apples", "milk", "cheese", "bread", "coffee"]

groceries.insert(0, "carrots") # ["carrots","apples", "milk", "cheese", "bread", "coffee"]

groceries.insert(1, "peas") # ["carrots","peas","apples", "milk", "cheese", "bread", "coffee"]

groceries.pop() # ["carrots", "peas", "apples", "milk", "cheese", "bread"]

groceries.pop(0) # ["peas","apples", "milk", "cheese", "bread"]

groceries.remove("cheese") # ["peas","apples", "milk", "bread"]

**Frequency analysis**

Loops = used to control how many times a statement or group of statements is/are executed.

<https://docs.python.org/3/reference/compound_stmts.html#for>

For loop = provides a simple way to iterate over a sequence, such as string or list, in order.

The pseudocode syntax for the for loop is:

for iterating\_var in sequence:

statement(s)

iterating\_var = the variable that takes the value of the item inside the sequence on each iteration of the loop

# Given string

given\_string = "You can have data without information, but you cannot have information without data."

# Convert the given string to lowercase

given\_string\_lower = given\_string.lower()

# Create a list containing every lowercase letter of the English alphabet

alphabet\_list = [chr(i) for i in range(ord('a'), ord('z') + 1)]

# Create a dictionary to store letter frequencies

letter\_frequency = {}

# Initialize frequencies to zero for each letter in the alphabet

for letter in alphabet\_list:

letter\_frequency[letter] = 0

# Calculate frequencies in the given string

for char in given\_string\_lower:

if char.isalpha():

letter\_frequency[char] += 1

# Display the letter frequencies

for letter, frequency in letter\_frequency.items():

print(f"The letter '{letter}' occurs {frequency} times in the given string.")

Unicode = character encoding system, unique number (code point) is assigned to every character, symbol and control code

Can use the ord() function in python to get the Unicode code point of a character, and the chr() function to get the character from a Unicode code point.

**alphabet\_list = [chr(i) for i in range(ord('a'), ord('z') + 1)]**

+1 is added to include z in the range

Chr(i) converts an integer ‘i’ representing a Unicode code point into the corresponding character

**for letter in alphabet\_list:**

**letter\_frequency[letter] =** 0

* Uses a for loop to iterate over each letter in alphabet list and assigns initial value of 0 to the corresponding entry in the letter frequency dictionary

**for char in given\_string\_lower:**

**if char.isalpha():**

**letter\_frequency[char] += 1**

* isalpha() checks if the character is an alphabetic letter (true/false)
* char is a loop variable that represents each individual character in the string

**COGRAMMAR WEB DEVELOPMENT COURSE**

**Lecture 1 – HTML**

**Basic terminal commands**

* terminal aka command-line interface/ shell = a text-based interfaces used to communicate with the operating system
* allows user to execute commands by typing in instead of relying on graphical user interface (GUI)
* powerful for automation, scripting and system admin tasks

PWD: print working directory

* shows the current directory/ location you’re in (drive and folder)
* PS E:\practical> pwd

LS: lists the files and directories in the current directory

* Files have extensions but folders do not, will have one name
* Gives size and name of each file

CD: change directory

* Navigate between directories
* cd name\_of\_folder

**Setting up dev environments**

VS code: <https://code.visualstudio.com/> can use web based but depends on internet speed

Browsers: firefox, chrome, edge, safari

**HTML**

Hypertext markup language – used to structure and format the content of websites. The primary building block to create website content, creates skeleton.

Tags:

* <html> - root element of document object model (whole document), contains all other elements in the code.
* <head> - contains metadata about the web page, such as title and any linked CSS or Javascript files
* <body> - contains main content of web page, displayed in the web browser’s window e.g., button, text, heading. Any visual element contained in this.

Document object model (DOM) = programming interface for web documents, represents the page so programs can change the document structure, style and content dynamically. When a web page is loaded, the browser creates a DOM of the page.

DOM tree represents the structure of an HTML document, each element in the document is a node in the tree, forming a parent-child relationship.

Allows you to do button clicks, click on links, hover over headings, cool dynamic effects. Internal representation/ mindmap of the whole document, allowing us to interact with the web page.

Browser does this by itself.

**HTML elements** = consist of several parts incl the opening and closing tags, the content and the attributes.

Opening tag – consists of the elemtn name, wrapped in angled brackets, indicated the start of the element and the point at which the elements effects begin.

<name>

Closing tag – forward slash / before the elemnt name, indicates end of element and point at which the elements effects stop. The content can be text, image or combination.

</name>

Element = opening tag + content + closing tag.

<name> content </name>

Attributes = provide additional info about the element

* Assigned to HTML element inside ***opening tag***
* Assign name to tag or unique ID to tag
* Assignment operator (name=) then value within quotation marks

<p name=”new\_paragraph”>Hi, I am a paragraph.</p>

Paragraphs - <p> creates paragraphs. Needs to be inside body tag otherwise won’t show up on screen. <p> then </p>

Headings - <h1> to <h6>. Higher the number, the smaller the size of the heading.

Comments - insert notes to web page, not visual, a reminder for yourself. <!—then 🡪

**Javascript**

* Used in both frontend and backend
* Built-in console used for debugging in chrome, safari and firefox
* Can run javascript code directly in the console as well

Variables

* Used to store the data we can use for calculations later on, like a box that holds information
* Declare it first – assign a storage space in memory and give it a name
* ‘let’ or ‘const’ then write the name of the variable after leaving a white space, the assign a value using = then the value, finally end it with a semicolon
* ‘let’ value can be changed at a later time but ‘const’ cannot be changed so used for stuff like value of pi for example

Example:

Let exampleVariable = “value of the variable”;

* Naming: contains only letters, numbers and underscores. Follows consistent convention (such as camelCase), not a reserved keyword like let/ const/ console
* The value assigned to a variable is a data type

Data types:

* Numeric e.g., let gameScore = 100;
* String e.g., let fullName = “Hannah Stanley”;
* Boolean e.g., let gameEnd = true;
* Array can store multiple data types separated by a comma e.g., let playerScores = [100, 200, 400];
* Object related to a real-world entity, a collection of related data, using {name:value} e.g., let playerProfileData = {firstName: “Hannah”, lastName: “Stanley”};

Linking scripts:

* <script> tag in HTML used to add javascript to the document
* Can also link HTML documents with external javascript files using the src attribute
* <script src=”scripts/hello.js”></script>

Print value to console:

Console.log()

**Lecture 2 – CSS and GitHub**

**CSS:**

* Cascading style sheets
* Used to change the presentation and styling of HTML document (document written in markup language)
* Visually appealing and user-friendly
* HTML: structures the content, CSS: controls how the content looks
* Creates style sheets which define appearance and layouts of the elements of the webpage

Styles

Inline style

* One of the attributes of an element is style which we can change by adjusting its properties using CSS rules
* Attributes are adjusted inside the element’s beginning tag
* Semicolon separates property value pairs
* Downside is creates a long line so difficult to understand code. Only use when a few properties to apply to an HTML element. 2-3 max
* Applied to individual elements overwriting the internal or eternal CSS defined for the whole web page

<p style = “font-family:Montserral;color:cornflowerblue;font-size:22px”>

Internal CSS

* CSS rules defined in the head part of the HTML template, inside the style element
* More readable
* The style sheet consists of selectors (indicates which element want to style e.g., p tag, body tag, h1 tag) and declarations (property and a value separated by colon)
* Declaration block – contains 1 or more declarations, separated by semicolons and enclosed in curly brackets
* Overwrite any external styling defined

<style>

p{

font-style: italic;

color: chartreuse;

}

</style>

External CSS

* Style rules in a separate .css file which can be linked to any HTML file to apply the style rules
* Readability – separating CSS code and HTML makes code easier to follow
* Maintainability – updating and debugging styling rules easier since only external CSS files need to change/ be replaced
* Useful when applying the same style rule to multiple HTML files
* Best one to use
* Put in the head part of the HTML in a link element
* Href: define the name and path of your file (relative to the current working directory)
* Rel: describes the type of relation the external file is to the HTML
* Type: tells the browser what sort of file it is (only necessary for older browsers)

<head>

<link href=”externalStyle.css” rel=”stylesheet” types=”text/css” />

</head>

**CSS selectors** – attach to HTML elements on web pages which allows for customised styling

Common ones:

Element selector – the same style is applied to elements with the same tag e.g., paragraph or heading

* Most basic type of selector

ID selector – styles are applied to specific elements using a unique ID

* ID attribute defined at beginning of the HTML tag e.g., <h2 id=”heading2”> Welcome everyone </h2>
* The value assigned to this attribute must be unique
* ID selector called using a hash followed by the ID name e.g., #heading2

Class selector – the same style is applied to elements in the same class e.g., class attribute

* Change all HTML elements associated with a specific class
* Class is also an attribute, defined like an ID but it is not unique e.g., <h2 class=endingMessage>Thank you for joining us </h2>
* Called using a dot followed by class name e.g., .endingMessage
* The element tag belonging to that class can be referenced as well e.g., if just want paragraphs containing that class will do p.endingMessage

**The Box Model**

* A rectangle is created for each element in the HTML document
* The padding, border and margin are added to the content to create the rectangle
* Each area is surrounded by a perimeter called an edge

A screen shot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

**CSS validator:**

* Using tools like VSCode identifies errors in syntax and formatting but some errors unnoticed
* Can use CSS validation service to check CSS code when doesn’t behave as expected

**Version control**

* VC systems record modifications to a file or set of files so you can recall specific versions of it later on

Benefits:

* Collaboration – team projects, changes merged to a common version and stored in a central place
* Storing versions – current version of a project worked on and stored locally and all prev versions stored and managed by the VC system
* Restoring previous versions – if errors/unintended results
* Understanding what happened – short description needs to be provided which describes the changes (comments) when changes made resulting in new version of project. Why made, who made them.
* Backup – every member of the team has complete version of project on their disk incl complete history, if central server breaks down or your backup fails a team members local version can be used instead

Git Version Control

* Git is most widely used modern version control system, free, open-source and scalable with project size
* Has a distributed architecture and is an example of a distributed version control system (DVCS)
* Every developers working copy of the code is also a repository that contains the full history of all changes

The files can have three main states:

Committed – files stored in local database (in your PC)

Modified – files have changed but are yet to be committed to your local database

Staged – modified files have been marked to go int0 the next commit in its current version

There are 3 main sections of a git project:

* Working directory
* Staging area
* Git repository (can then be pushed to server)

Git add – working directory to staging area

Git commit – staging area to repository

**GitHub**

* Online Git repository hosting service (remote repository), providing a web-based graphical interface
* GitHub Desktop allows to access on local desktop
* Also a large social networking site for developers and programmers
* Create a technical portfolio to showcase work and technical capabilities

**Repositories**

* A repository in Git is a hidden folder called ‘.git’ which is located in the root directory of your project (local repository)
* This is where your version control system stores all the files for a particular project
* Local repository is located on computer as the ‘.git’ folder inside the projects root folder
* Remote repository located on a remote server on the internet or in your local network
* Local repositories are created by either initialising a new repository or cloning

Common Git commands (all ran in the terminal)

* Initialise a new repository in the current directory – git init
* Add a new file to the repository staging area – git add fileName.js
* Check the status of the files in your working directory – git status
* Committing staged changes with a meaningful message – git commit -m “Added new file fileName.js”
* Reviewing change history – git log

Using remote repositories:

* To synchronise a local repository to a remote repository hosted by Github, authenticated access to your Github account is needed
* For this, GitHub Command Line Interface needs to be used
* Once installed, login using the following command and follow the prompts

Gh auth login

Choose github.com for the account and HTTPS for the preferred protocol

* Sync your local repository to a remote GitHub repository:

Git remote add origin [https://github.com/[REPO-OWNER]/[REPO-NAME](https://github.com/%5bREPO-OWNER%5d/%5bREPO-NAME)]

Git branch -M main

Git push -u origin main

To clone a repository hosted on GitHub

Git clone [https://github.com/[REPO-OWNER]/[REPO-NAME](https://github.com/%5bREPO-OWNER%5d/%5bREPO-NAME)]

To fetch the newest version of the repository:

Git fetch origin

**Branches (Git)**

* Branch represents an independent line of developing
* Allows each developer to branch out from the original codebase and isolate their work from others
* Developers can continue to work without messing up or disrupting the main line
* Ensures that unstable code is not committed to the main codebase

Creating a branch:

* Git branch branch-name

Switching branches:

* Git checkout branch-name
* Git checkout -b new-branch-name

Merging branches:

* Git merge branch-name

Git add .

* Makes all modified documents staged

**TUTORIAL 1**

**Navigating the terminal:**

* Create a new file: vi filename.txt, press i to insert content, then save by pressing escape and shift and zz
* Cat command: allows us to read the contents of a file in the terminal, type cat filename (remember extension such as .txt)
* CP command: use to copy contents, create copies, the original file comes first then write the name of the copy. Cp filename.txt filename\_copy.txt

Can also use it to make a copy of a folder. Cp -r directoryname1 directorynamecopy (no extensions as folders don’t have extensions)

* MV command: renaming files or directories, original name first then new name second. mv filename1 filename2. Mv directory1 directory2
* RM command: deleting files. rm filename.txt. if want to remove a directory, use rmdir directoryname.

Remove subdirectory by doing rmdir directory/subdirectory

* Create new folder: mkdir new\_directory\_name

Can also make subdirectories in a directory using a slash: mkdir folder\_name/subfolder\_name

* Listing all files in the terminal: ls

If want to only find directories with lecture in for example can do ls lec\*

Ls-l will show the metadata for the files

Wildcard: ls \*commands.txt (lists every file that has commands.txt at the end of it)

Ls g?\* – all the files that have g at the beginning

The question mark checks for a single character, \* checks for multiple characters

* Find any hidden contents in files or directories: ls – a
* Check contents of a specific directory: ls directory name
* Change directory/ enter directory: cd directory name
* To go back to parent directory (reverse the above): cd ..
* Ask the terminal current location, print working directory: pwd
* Word count: wc filename. Shows three numbers – first one is number of lines, second number is number of words, third number is the number of bites

**Git commands (in the terminal):**

* Check if you have git: git – version
* Make a new repository: git init
* Make username/ change username: git config user.name your\_name\_here
* Show username: git config user.name
* Show email git config user.email (put at end if wanna change)
* Current status of all files in repository: git status

Untracked files mean hasn’t been added to the stage yet

* Stage all files: git add . (use filename not dot if just want one)
* Commit changes to repository: git commit -m “message is compulsory”

Record of different changes made to repository

* Track commits made to repository: git log, press q to exit (like opening history in browser)
* The .git is the child folder of the terminal-git
* Can reverse staging by: git restore – staged . (drags back to modified status)
* Git restore . will reverse changes made to file

Sync local repository to Github repository

* Git remote add origin <https://github.com> (link from new repository made on github.com)
* Git branch -M main (creates main branch instead of master in local repository, Github only works with main not master)
* Upload to remote repository: git push -u origin main

Link remote repository to an empty folder on your laptop:

* Using the terminal, go to the folder via: cd folder\_name
* Then git clone + link from remote repository (https version)

**Branches:**

* Git branch – to list all branches, also marks the current one you are in
* Git branch new-branch-name – make a new branch
* Git checkout branch-name – move to another branch
* Git merge branch-name – go into main branch then branch-name will be the branch you want to merge to it
* Git fetch origin – gets the metadata from the remote repository to the local repository, so any new branch created in remote repository gets updated to local repository
* Git remote show origin –

**HTML**

* Semantic tags – accurately describe their purpose and describe the type of their content e.g., img, p, h1, h2
* Non-semantic tags – do not have a specific meaning or purpose, used to create general-purpose containers for content without providing any additional meaning or context e.g., div, header, footer

**CSS**

* The selectors after the : and :: are pseudo-class selectors, which allow us to select elements on the basis of their position and state within the document
* This example: button tag that has been hovered on

Button:hover {

Color: white;

}

Can check on the browser the padding under inspect then styles

Line height – further negative increases line height

Font-weight – thickness of the text

**Javascript:**

This is how you make a **variable** in javascript

This is a numerical data type, can also be decimal

let numberOfCats = 9;

Print values to browser console: console.log("Number of cats: ", numberOfCats);

This is a **string data type**

let fullName = "John Doe";

console.log("Name: ", fullName);

This is how you make a **boolean data type**

Remember: A boolean data type can be either true or false

let doorOpen = true;

console.log("doorOpen is a boolean data type: ", doorOpen);

**Arrays:**

let numbers = [1, 2, 3, 4, 5, 6];

console.log("Numbers array: ", numbers);

let items = ["Charlie", 6, true, 9, 4, 6, 5, 3];

console.log("Items array: ", items);

**Objects** used to represent a real-world entity:

let car = {

numberofWheels: 4,

modelName: "Model T",

manufacturerName: "Tesla",

};

* Contains property value pairs

console.log("Car object: ", car);

**Constants:**

const PI = 3.14;

PI = 2.2 (will cause error)

* Allows the developer to create constant variables in JS
* Can only initialise them once, then cannot change them later on

Check html syntax errors: <https://validator.w3.org/#validate_by_input>

Check CSS: <https://jigsaw.w3.org/css-validator/#validate_by_input>