Technical Overview of JavaScript Language

Monster definition: JavaScript is a **high-level**, **prototype-based, object-oriented**, **multi-paradigm**, **interpreted or just-in-time complied**, **dynamic, single-threaded, garbage-collected** programming language with **first-class functions** and a **non-blocking even loop concurrency model**

1. **High-Level**

* Any programming language needs hardware resources to function eg memory for storage, cpu for processing
* In low level languages (eg C), you have to manually managed these resources and ask the computer for memory to store a variable
* In high level languages, there are abstractions that do this automatically

|  |  |
| --- | --- |
| **High Level** | **Low Level** |
| Auto resource management = Easier to learn | Manual resource management = Hard to learn |
| Cannot optimise as well | Better optimised |
|  |  |
|  |  |
|  |  |

**A picture containing timeline

Description automatically generated**

1. **Garbage Collection**

A picture containing broom

Description automatically generated**Definition:** Garbage Collection is an algorithm inside the JavaScript engine which automatically removes old, unused objects from the computer memory. This prevents memory from becoming clogged up with unnecessary information. Like having a janitor in computer’s memory.

1. **Interpreted or just-in-time compiled**

* The CPU can only understand 1s and 0s (machine code), so ultimately every program has to be written in 0s and 1s.
* Humans can’t easily write machine code, so programming languages were developed as an abstraction over machine code (like a bridge between Humans and CPUs)

**Graphical user interface

Description automatically generated with low confidence**

1. **Multi-paradigm**

**Definition**: An approach and mindset of structuring code, which will direct coding style and technique.

JavaScript is very flexible and versatile because you can use all of the main 3 paradigms

**3 main paradigms:**

* Procedural Programming (FP)
* Object-orientated Programming (FP)
* Functional programming (FP)

**Paradigms can be classed as:**

* Imperative
* Declarative

**Features:**

* Procedural – Organising code in a linear way with some functions in between
* Object-Orientated – *Jonas to explain later*
* Functional Programming - *Jonas to explain later*

Graphical user interface, text, application

Description automatically generated

1. **Prototype-based object orientated**

* Almost everything in JavaScript is an object (eg arrays), except for primitive values (numbers, strings etc).
* It is possible to create an array and use the push method on it immediately because of **prototypal inheritance:**
  + Arrays are created from an array **blueprint** which is called the array prototype
  + The prototype contains all the array methods
  + The array we create in our code then **inherit** all the methods from the prototype
  + **NOTE:** This is an oversimplification

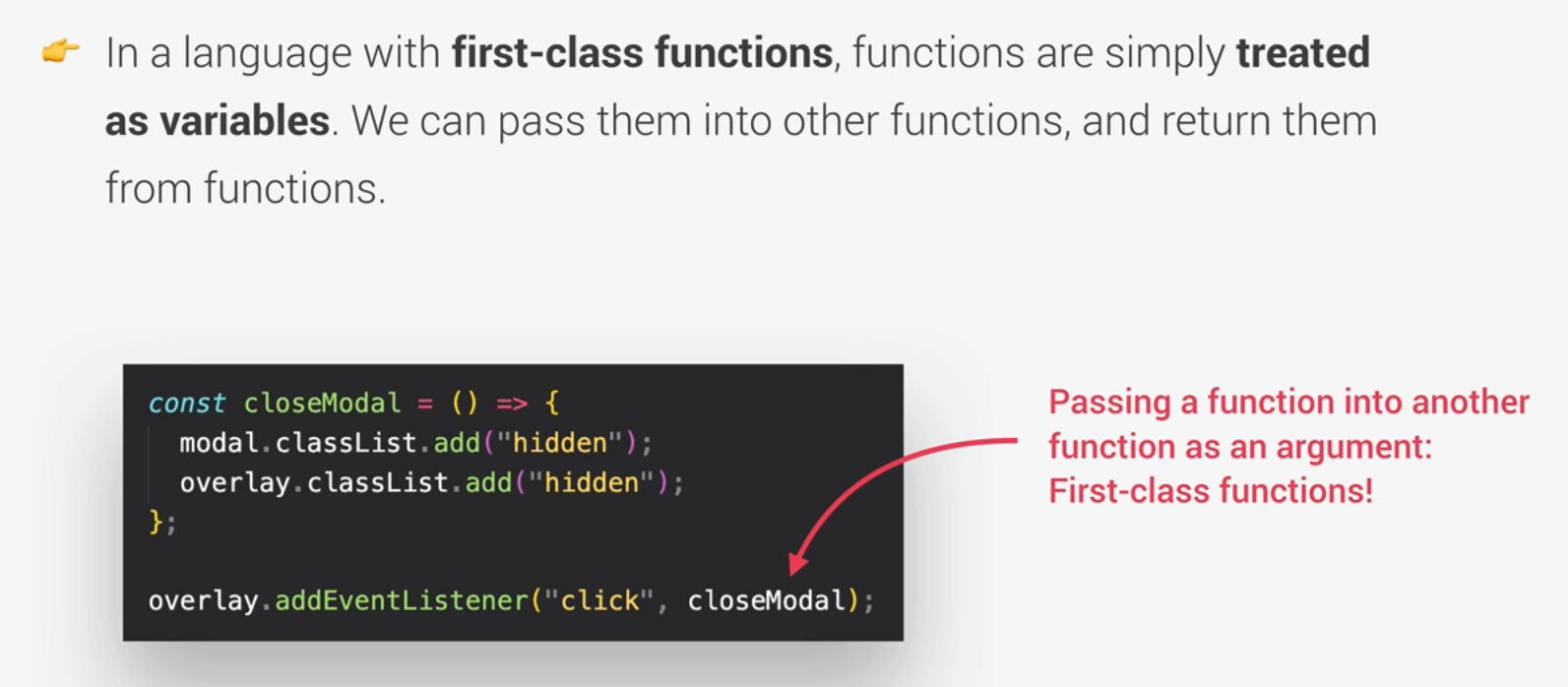
A screenshot of a computer

Description automatically generated with medium confidence

1. **First class functions**

**Definition:** First class functions are functions that are simply treated as variables. They can be passed into other functions as arguments and returned from functions

* This allows for the use of a lot of powerful techniques and for **functional programming**
* First class functions were used when I created modal window



1. **Dynamic (Dynamically typed)**

* In JavaScript, data types are not assigned to variables. The data types only become know when the JavaScript engine executes the code.
* Variable types can easily be changed as variables are re-assigned
* A fair amount of controversy about dynamic programming languages, whether it is good or bad
* **Most** other programming languages require the coder to assign types to the variables:
  + Manual type assignment can help prevent bugs from occurring (and is why people argue that JavaScript should be a strongly-typed language)
  + JavaScript with types is TypeScript

**Diagram

Description automatically generated**

1. Single-threaded / Non-blocking event loop

**Definition:**

**Concurrency Model:** How the JavaScript engine handles multiple tasks happening at the same time

**Thread:** A set of instructions that is executed in the computer’s CPU

**Single Thread:** JavaScript runs in one **single thread,** so it can only do one thing at a time

* If there a long-running task, JavaScript execution could be blocked from executing as it’s single thread. This is undesirable, therefore we need non-blocking behaviour
* An **event loop** is used to execute long-running tasks in the background and then puts them back into the main thread once they are finished
* **NOTE:** This is an oversimplification

Graphical user interface, text, application, email

Description automatically generated