# Programming Language Project Report: JavaScript

Created by:

Shann Neil O. Estabillo 6510099

Joseph Chidiebere Anyalogbu 6520282

Hein Htoo Naing 6440027

## Introduction: Programming Languages and JavaScript

### 1. Importance of Programming Languages in Software Development

Coding languages are the foundation of modern technology. They allow us to communicate with computers, create applications, and build software. With the increasing demand for technology and automation, coding languages have become an essential tool for developers worldwide.

In this report, we will be exploring JavaScript's features, and specifications.

**A. JavaScript: Language Features and Specifications**

**- Syntax**:

- JavaScript Syntax is used to define the set of rules to construct a JavaScript code. You need to follow all these so that you can work with JavaScript.

```
console.log("Basic Print method in JavaScript");
```

- JavaScript syntax refers to the set of rules that determines how JavaScript programs are constructed:

```
// Variable declaration
let c, d, e;

// Assign value to the variable
c = 5;

// Computer value of variables
d = c;
e = c/d;
```

- A JavaScript **variable** is the simple name of the storage location where data is stored. There are two types of variables in JavaScript which are listed below:

- Local variables: Declare a variable inside of a block or function.

Global variables: Declare a variable outside function or with a window object.

Example: This example shows the use of JavaScript variables.

```
// Declare a variable and initialize it
// Global variable declaration
let Name = "Apple";

// Function definition
function MyFunction() {

    // Local variable declaration
    let num = 45;

    // Display the value of Global variable
    console.log(Name);

    // Display the value of local variable
    console.log(num);
}

// Function call
MyFunction();
```

Output:

Apple
45

- JavaScript **operators** are symbols that are used to compute the value or in other words, we can perform operations on operands. Arithmetic operators ( +, -, *, / ) are used to compute the value, and Assignment operators ( =, +=, %= ) are used to assign the values to variables.

- Example: This example shows the use of JavaScript operators.

```
// Variable Declarations
let x, y, sum;

// Assign value to the variables
x = 3;
y = 23;

// Use arithmetic operator to
// add two numbers
sum = x + y;

console.log(sum);
```

Output:

26

- **Expression** is the combination of values, operators, and variables. It is used to compute the values.
- Example: This example shows a JavaScript expression.

```
// Variable Declarations
let x, num, sum;

// Assign value to the variables
x = 20;
y = 30

// Expression to divide a number
num = x / 2;

// Expression to add two numbers
sum = x + y;
```

*console.log(num + "<br>" + sum);*

Output:

10

50

- The **keywords** are the reserved words that have special meanings in JavaScript.

*// let is the keyword used to*

*// define the variable*

*let a, b;*

*// function is the keyword which tells*

*// the browser to create a function*

*function GFG(){};*

- The **comments** are ignored by the JavaScript compiler. It increases the readability of code. It adds suggestions, Information, and warning of code. Anything written after double slashes // (single-line comment) or between /* and */ (multi-line comment) is treated as a comment and ignored by the JavaScript compiler.

- Example: This example shows the use of JavaScript comments.

*// Variable Declarations*

*let x, num, sum;*

*// Assign value to the variables*

*x = 20;*

*y = 30*

```
/* Expression to add two numbers */
sum = x + y;

console.log(sum);
```

Output:

50

- JavaScript provides different datatypes to hold different values on variables. JavaScript is a dynamic programming language, which means do not need to specify the type of variable. There are two types of data types in JavaScript.
- Primitive data type
- Non-primitive (reference) data type

```
// It store string data type
let txt = "GeeksforGeeks";
// It store integer data type
let a = 5;
let b = 5;
// It store Boolean data type
(a == b )
// To check Strictly (i.e. Whether the datatypes
// of both variables are same) === is used
(a === b)---> returns true to the console

// It store array data type
let places= ["GFG", "Computer", "Hello"];
```

```
// It store object data (objects are
// represented in the below way mainly)
let Student = {
firstName:"Johnny",
 lastName:"Diaz",
 age:35,
 mark:"blueEYE"}
```

- JavaScript **functions** are the blocks of code used to perform some operations. JavaScript function is executed when something calls it. It calls many times so the function is reusable.

```
function functionName( par1, par2, ....., parn ) {
    // Function code
}
```

- The JavaScript function can contain zero or more arguments.
- Example: This example shows the use of JavaScript functions.

```
// Function definition
function func() {

    // Declare a variable
    let num = 45;

    // Display the result
    console.log(num);
}
```

*// Function call*

*func();*

Output:

45

**B. JavaScript: Type System**

**- Dynamic Typing**:

- JavaScript is a dynamically typed language. Dynamically-typed languages are those where the interpreter assigns variables a type at runtime based on the variable's value at the time.

**- Object-oriented Features**: **Objects**

- In JavaScript, objects are crucial for creating complex data structures and modelling real-world concepts. They are collections of key-value pairs, where the keys are known as properties and the values can be of any data type, including other objects or functions.

- JavaScript offers multiple ways to create objects. One common method is using object literals, which allow you to define an object and its properties in a concise manner. For example:

*“const person = {*

*name: "John",*

*age: 25,*

*profession: "Engineer"*

*};”*

- In this example, person is an object with properties such as name, age, and profession. Each property has a corresponding value.

**- Object-oriented Features**: **Objects**

- JavaScript also provides constructors and prototypes as mechanisms for creating objects and defining shared properties and methods. Constructors are functions used to create instances of objects. Prototypes, on the other hand, enable objects to inherit properties and methods from their prototype objects. For example:

*"// Constructor function*

*function Person(name, age) {*

*this.name = name;*

*this.age = age;*

*}*

*// Creating an object instance using the constructor*

*const john = new Person("John", 25);"*

- In this example, the Person function acts as a constructor for creating Person objects. The new keyword is used to instantiate an object from the constructor, passing the necessary arguments.

**- Object-oriented Features**: **Instances**

- Object instances are individual objects created using constructors. Each instance has its own set of properties and values, while also sharing the same methods defined in the prototype. This allows for code reusability and efficient memory usage. For example:

*function Person(name, age) {*

```
  this.name = name;
  this.age = age;
}

Person.prototype.greet = function() {
  console.log("Hello, my name is " + this.name);
};
const john = new Person("John", 25);
const jane = new Person("Jane", 30);
john.greet(); // Output: Hello, my name is John
jane.greet(); // Output: Hello, my name is Jane
```

- In this example, the greet method is defined in the prototype of the Person constructor. Both john and jane instances can access and invoke the greet method, even though it is defined only once in the prototype.

- **Accessing object properties and methods:**

- Accessing object properties and invoking methods can be done using the dot notation or the bracket notation. The dot notation is typically used when the property name is known in advance, while the bracket notation allows for dynamic property access.

```
const person = {
  name: "John",
  age: 25,
  greet: function() {
    console.log("Hello, I'm " + this.name);
  }
```

```
};


console.log(person.name); // Output: John
console.log(person["age"]); // Output: 25
person.greet(); // Output: Hello, I'm John
```

- In this example, the properties name and age are accessed using the dot notation, while the greet method is invoked using parentheses.

- Objects are integral to JavaScript, providing a versatile and powerful way to structure and manipulate data. By understanding how to create objects, utilise constructors and prototypes, and access properties and methods, you can leverage the full potential of objects in JavaScript programming.

- **Classes in JavaScript**

- Now that we know what an object is, we will define a class. In JavaScript, a class is a blueprint for creating objects with shared properties and methods. It provides a structured way to define and create multiple instances of similar objects. Classes in JavaScript follow the syntax introduced in ECMAScript 2015 (ES6) and offer a more organised approach to object-oriented programming.

- When creating a class in JavaScript, you use the class keyword followed by the name of the class. The class can have a constructor method, which is a special method that is called when creating a new instance of the class. The constructor is used to initialise the object's properties.

- Within a class, you can define methods that define the behaviour of the class instances. These methods can be accessed and invoked by the instances of the class. You can also define static methods that are associated with the class itself rather than its instances. Here's an example of a JavaScript class:

**- Other Language Features**

- Within a class, you can define methods that define the behaviour of the class instances. These methods can be accessed and invoked by the instances of the class. You can also define static methods that are associated with the class itself rather than its instances. Here's an example of a JavaScript class:

```
class Car {
  constructor(make, model, year) {
    this.make = make;
    this.model = model;
    this.year = year;
  }

  getAge() {
    const currentYear = new Date().getFullYear();
    return currentYear - this.year;
  }

  static isOld(car) {
    return car.getAge() >= 10;
  }
}

const myCar = new Car("Toyota", "Camry", 2018);
```

*console.log(myCar.getAge()); // Output: 5*

*console.log(Car.isOld(myCar)); // Output: false*

- In this example, the Car class has a constructor that takes the make, model, and year as parameters and initialises the respective properties. It also has a getAge() method that calculates the age of the car based on the current year. The isOld() method is a static method that determines if a car instance is considered old.

- **The four pillars of OOP**

- This section of the report will expand on the four pillars of OOP.

- **Encapsulation**

- Encapsulation is defined as the process of bundling data and methods together within a single unit, known as an object. It allows for the organisation of related data and operations, promoting code modularity and reusability. Encapsulation provides two key benefits: data hiding and access control.

- By encapsulating data, an object's internal state is hidden from external entities. This ensures that the data can only be accessed and modified through defined methods, preventing direct manipulation and maintaining data integrity. Access control mechanisms, such as private and public modifiers, allow developers to control the visibility and accessibility of object members. Encapsulation in JavaScript can be achieved using techniques like closures, IIFE (Immediately Invoked Function Expressions), and modules.

- **Inheritance**

- Inheritance is a mechanism that allows objects to acquire properties and methods from a parent object, known as a superclass or base class. It promotes code reuse and hierarchical relationships between objects.

- In JavaScript, inheritance is implemented through prototypal inheritance. Each object has an associated prototype object, and properties and methods not found in the object itself are inherited from the prototype. This enables objects to share common functionality while maintaining the ability to add or override specific behaviours. By leveraging inheritance, developers can reduce code duplication, enhance code organisation, and establish relationships between objects based on their similarities and hierarchies.

- **Polymorphism**

- Polymorphism enables objects of different types to be treated as interchangeable entities. It allows for flexibility in code design and promotes code extensibility.

- In JavaScript, polymorphism is inherent due to the language's dynamic typing nature. This means that the same function or method can be used with different object types, as long as they support the required interface or share a common set of methods. This flexibility enables code to work with objects of multiple types without explicitly checking their specific types. Polymorphism can enhance code readability, simplify code maintenance, and enable the creation of more generic and reusable functions.

- **Abstraction**

- Abstraction focuses on simplifying complex systems by breaking them down into smaller, more manageable modules. It involves defining essential characteristics and behaviours while hiding unnecessary implementation details.

- In JavaScript, abstraction can be achieved through abstract classes and interfaces. Abstract classes provide a blueprint for creating derived classes, defining common methods and properties that must be implemented by subclasses. Interfaces, although not natively supported in JavaScript, can be emulated using object literals or documentation to define expected properties and methods.

- Abstraction allows developers to focus on high-level concepts and functionalities, providing a level of abstraction that hides complex

implementation details. This simplifies code understanding, enhances code maintainability, and facilitates collaboration within development teams.

- **What is Asynchronous Programming?**

- Asynchronous programming is a technique that allows your program to run its tasks concurrently. You can compare asynchronous programming to a chef with multiple cookers, pots, and kitchen utensils. This chef will be able to cook various dishes at a time.

- Asynchronous programming makes your JavaScript programs run faster, and you can perform asynchronous programming with any of these:

- Callbacks
- Promises

- **Callbacks**

- A callback is a function used as an argument in another function. Callbacks allow you to create asynchronous programs in JavaScript by passing the result of a function into another function.

```
function greet(name) {
    console.log(`Hi ${name}, how do you do?`);
}

function displayGreeting(callback) {
    let name = prompt("Please enter your name");
    callback(name);
};

displayGreeting(greet);
```

- In the code above, the greet function is used to log a greeting to the console, and it needs the name of the person to be greeted.

- The displayGreeting function gets the person's name and has a callback that passes the name as an argument to the greet function while calling it. Then the displayGreeting function is called with the greet function passed to it as an argument.

- **Promise**

- Most programs consist of a producing code that performs a time-consuming task and a consuming code that needs the result of the producing code.

- A Promise links the producing and the consuming code together. In the example below, the displayGreeting function is the producing code while the greet function is the consuming code.

```
let name;

// producing code
function displayGreeting(callback) {
    name = prompt("Please enter your name");
}

// consuming code
function greet(name) {
    console.log(`Hi ${name}, how do you do?`);
}
```

- In the example below, the new Promise syntax creates a new Promise, which takes a function that executes the producing code. The function either resolves or rejects its task and assigns the Promise to a variable named promise.

- If the producing code resolves, its result will be passed to the consuming code through the .then handler.

```
let name;
```

```
function displayGreeting() {
    name = prompt("Please enter your name");
}

let promise = new Promise(function(resolve, reject) {
    // the producing code
    displayGreeting();
    resolve(name)
});

function greet(result) {
    console.log(`Hi ${result}, how do you do?`);
}

promise.then(
    // the consuming code
    result => greet(result),
    error => alert(error)
);
```

- **Prototype Inheritance**

- Prototype inheritance in JavaScript is the linking of prototypes of a parent object to a child object to share and utilize the properties of a parent class using a child class.

- Prototypes are hidden objects that are used to share the properties and methods of a parent class with child classes.

- The syntax used for prototype inheritance has the proto property which is used to access the prototype of the child. The syntax to perform a prototype inheritance is as follows:

```
child.__proto__ = parent;
```

- Example:

-

```
// Creating a parent object as a prototype
const parent = {
  greet: function() {
    console.log(`Hello from the parent`);
  }
};

// Creating a child object
const child = {
  name: 'Child Object'
};

// Performing prototype inheritance
child.__proto__ = parent;

// Accessing the method from the parent prototype
child.greet(); // Outputs: Hello from the parent
```

- In this example, parent is the parent object acting as the prototype, and child is the child object that inherits from the parent prototype using the __proto__ property. This allows the child object to access the greet method defined in the parent object directly.

- While the __proto__ property can be used to perform prototype inheritance, it's important to note that directly manipulating __proto__ is not recommended in production code. Instead, the Object.create method or constructor functions with prototype are typically used for setting up prototype chains in a safer and more maintainable way.

- **Garbage Collection Algorithm**

- The JavaScript garbage collection process uses a mark-and-sweep algorithm. Here’s how that works:

- There are two phases in this algorithm: **mark** followed by a **sweep.**

- **Mark Phase:** The garbage collector starts from the root objects and marks all reachable objects. Any object that can be accessed directly from these roots is marked as reachable. This includes objects referenced from other reachable objects.

- **Sweep Phase:** After marking, the garbage collector will then go through the memory and free the space occupied by objects that were not marked as reachable. These are objects that the program can no longer access.

- **Language Design Decisions**

- JavaScript's design decisions are influenced by its role on the web, emphasizing simplicity and flexibility:

- **Web-centric Approach**: JavaScript prioritizes features for web development, such as dynamic typing and prototype-based inheritance.

- **Garbage Collection (GC):** GC automates memory management, relieving developers from manual memory allocation. JavaScript's GC, typically using a mark-and-sweep algorithm, balances automatic memory management with performance considerations.

- **Reasons for Garbage Collection:** The decision to include garbage collection in JavaScript was driven by the need to make memory management easier and less error-prone for developers. By automating memory management, JavaScript relieves developers of the burden of manual memory allocation and deallocation, which can be error-prone and difficult to manage in large and complex applications.

# Comparative Analysis: JavaScript vs. Python

1. **Syntax and Type System (key differences in syntax and type systems between JavaScript and Python.)**

JavaScript and Python are both popular programming languages, each with its own strengths and weaknesses. Let's compare them in terms of syntax and type system:

**A. Syntax**

**- JavaScript**:

- JavaScript syntax is derived from C programming language and shares some similarities with Java. It follows curly brace syntax.
- Statements are terminated by semicolons (;), although they are often optional.
- Uses camelCase for variable and function naming convention.
- Function definition is done using the `function` keyword.

**- Python:**

- Python syntax is known for its simplicity and readability. It follows an indentation-based syntax where blocks of code are defined by their indentation level.
- Statements do not require terminating semicolons.
- Uses snake case for variable and function naming convention.
- Function definition is done using the `def` keyword.

**B. Type System**

**- JavaScript:**

- JavaScript is loosely typed, meaning variables can hold values of any data type without explicit declaration.
- It employs dynamic typing, where variables can change types during runtime.
- Type coercion is common, where JavaScript attempts to convert types implicitly.
- Primitive data types include numbers, strings, Booleans, null, undefined, symbols (newer versions), and BigInt (newer versions).

**- Python:**

- Python is strongly typed, meaning variables are bound to a specific data type and cannot change type unless explicitly converted.
- It also employs dynamic typing, allowing variables to be assigned different types during runtime.
- Type coercion is less common compared to JavaScript.
- Python has a rich set of built-in data types including integers, floats, strings, booleans, None, tuples, lists, dictionaries, and sets.

In summary, while both JavaScript and Python are powerful languages, they differ in syntax, type system, and approach to programming paradigms such as object-oriented programming. JavaScript tends to be more prevalent in web development, while Python is often used in various domains such as data science, machine learning, and backend development. Each language has its own ecosystem and strengths, making them suitable for different tasks and preferences.

## 2. Scoping Rules and Other Language Features.

Let's look at the comparison of JavaScript and Python, focusing on scoping rules, asynchronous programming, inheritance models, and other language features:

### A. Scoping Rules:

**- JavaScript:**

- JavaScript has function-level scope by default, meaning variables declared inside a function are local to that function.
- Variables declared with `var` keyword are function-scoped.
- However, with the introduction of `let` and `const` in ES6, block-level scoping is also possible.
- JavaScript also supports lexical scoping, meaning inner functions have access to variables and parameters of their outer function.

**- Python:**

- Python uses block-level scoping, meaning variables declared inside blocks (e.g., if statements, loops, functions) are local to that block.
- Python also supports lexical scoping, where nested functions have access to variables in the enclosing function's scope.
- Variables declared outside of any function or block have global scope.
- The `global` and `nonlocal` keywords allow modifying variables in the global and enclosing scopes, respectively.

**B. Asynchronous Programming:**

**- JavaScript:**

- JavaScript is inherently asynchronous and supports asynchronous programming through callbacks, promises, and async/await syntax.
- Callbacks are functions passed as arguments to other functions, commonly used in event-driven and asynchronous operations.
- Promises provide a cleaner way to handle asynchronous operations and avoid callback hell.
- Async/await syntax, introduced in ES2017, allows writing asynchronous code in a synchronous-looking manner, making it easier to read and maintain.

**- Python:**

- Python also supports asynchronous programming through async/await syntax introduced in Python 3.5.
- Asynchronous programming in Python is based on coroutines, which are functions that can pause and resume execution asynchronously.
- The `async` keyword is used to define asynchronous functions, and `await` is used to pause execution until an asynchronous operation completes.
- Python's asyncio library provides tools for asynchronous I/O, event loop, and concurrency.

**C. Inheritance Models:**

**- JavaScript:**

- JavaScript supports prototypal inheritance, where objects can inherit properties and methods directly from other objects.
- Objects in JavaScript are linked to a prototype object, and properties not found in an object are looked up in its prototype chain.
- Classes were introduced in ES6, providing syntactic sugar for defining constructor functions and prototypes, but they are still based on prototypes under the hood.

**- Python:**

- Python supports class-based inheritance, where classes define the structure and behaviour of objects.
- In Python, every class inherits from a base class called `object` by default.

- Python's inheritance model is based on the concept of method resolution order (MRO), which determines the order in which methods are resolved in the inheritance hierarchy.
- Multiple inheritance is supported in Python, allowing a class to inherit from multiple parent classes.

**D. Type Checking**

**- JavaScript:**

- JavaScript traditionally relies on runtime type checking, which means type errors may not be caught until the code is executed.
- However, with the introduction of TypeScript, a superset of JavaScript, static type checking can be performed using type annotations.

**- Python:**

- Python also relies heavily on runtime type checking.
- However, recent versions of Python (3.5 and later) support type hints, allowing developers to annotate function signatures and variable types. These hints can be used by static type checkers like Mypy to perform static type checking.

**E. Object-Oriented Programming**

**- JavaScript:**

- JavaScript supports object-oriented programming (OOP) through prototypal inheritance. Objects can inherit properties directly from other objects.
- Classes were introduced in ECMAScript 2015 (ES6), providing syntactic sugar for defining constructor functions and prototypes.

**- Python:**

- Python supports OOP with classes and objects.
- It follows a class-based inheritance model where classes define the structure and behaviour of objects.
- Python has a more traditional class syntax compared to JavaScript

In summary, while both JavaScript and Python offer powerful features and capabilities, they differ in their scoping rules, asynchronous programming models, inheritance mechanisms, and other language features. Understanding these differences can help developers choose the right language for their specific use case and preferences.

## Garbage Collection Algorithms (Comparison)

Here we look at the differences between JavaScript's garbage collection and Python's garbage collection. JavaScript and Python are high-level programming languages, both feature automatic garbage collection to manage memory allocation and deallocation. However, they employ different strategies and mechanisms for garbage collection.

### JavaScript

1. Garbage Collection Mechanism:

- JavaScript primarily uses a tracing garbage collector, specifically a form of garbage collector called "Mark-and-Sweep."

- When a JavaScript program is running, the garbage collector periodically checks for objects that are no longer reachable (i.e., not referenced by any other part of the program).

- The collector marks reachable objects and sweeps away (deletes) unreferenced objects, reclaiming their memory.

2. Memory Management:

- JavaScript's garbage collector is designed to manage memory automatically, which means developers do not need to manually allocate or deallocate memory.

- JavaScript developers generally don't have direct control over the garbage collection process. However, developers can optimize memory usage and performance by being mindful of object lifecycles and avoiding memory leaks.

3. Memory Leaks:

- While JavaScript's garbage collector is generally effective, memory leaks can still occur if developers unintentionally retain references to objects that are no longer needed.

- Common causes of memory leaks in JavaScript include circular references, event listeners that are not properly removed, and long-lived objects.

**Python**

1. Garbage Collection Mechanism:

- Python also uses automatic garbage collection, employing a combination of reference counting and a cyclic garbage collector.

- Reference counting tracks the number of references to an object. When the count reaches zero, the object is immediately deallocated.

- The cyclic garbage collector detects and collects cyclic references (i.e., reference cycles) where objects reference each other in a circular manner, preventing their deallocation by reference counting alone.

2. Memory Management:

- Like JavaScript, Python's garbage collector handles memory management automatically, alleviating developers from manual memory allocation and deallocation tasks.

- However, Python's garbage collector may not be as efficient as reference counting alone, especially in scenarios involving cyclic references, which require periodic garbage collection cycles.

3. Memory Leaks:

- Python's garbage collector helps mitigate memory leaks by reclaiming memory when objects are no longer referenced.

- However, developers still need to be cautious about creating cyclic references, as they can prevent objects from being deallocated and lead to memory leaks.

- Additionally, Python's garbage collector may not immediately reclaim memory in all cases, which means memory leaks may still occur if developers inadvertently retain references to objects unnecessarily.

## Conclusion

In conclusion, our exploration into the intricacies of JavaScript and its comparison with Python has shed light on the diverse landscape of programming languages and their implications in software development. We began by recognizing the paramount importance of programming languages in shaping software solutions and fostering innovation.

JavaScript, one of the most widely used languages, stood at the forefront of our discussion. We delved into its syntax, dynamic typing system, scoping mechanisms, and various language features such as asynchronous programming with Promises, prototypal inheritance, closures, and arrow functions. Moreover, we scrutinized its garbage collection algorithm and examined the rationale behind its design decisions, which significantly influence development practices.

Through a meticulous comparative analysis, we juxtaposed JavaScript with Python, accentuating their differences in syntax, type systems, scoping rules, and other language features. This comparison provided valuable insights into the strengths and weaknesses of each language, empowering developers to make informed decisions based on their project requirements and preferences.

Furthermore, our examination of garbage collection algorithms in both languages underscored the importance of memory management in ensuring the efficiency and reliability of software systems.

As we wrap up our presentation, it's crucial to reiterate the significance of comprehending JavaScript's features and specifications, given its pervasive presence in web development and beyond. Additionally, the insights gleaned from comparing JavaScript with Python serve as a springboard for further exploration and research, fostering continuous learning and growth within the developer community.

In essence, by understanding the nuances of JavaScript and appreciating its distinctions from Python, developers can navigate the dynamic landscape of

programming languages more adeptly, ultimately driving innovation and excellence in software development.

References:


- https://www.geeksforgeeks.org/javascript-basic-syntax/
- https://developer.mozilla.org/en-US/docs/Glossary/Dynamic_typing
- https://www.theknowledgeacademy.com/blog/oops-concepts-in-javascript/
- https://www.freecodecamp.org/news/asynchronous-programming-in-javascript-examples/#:~:text=async%20%2F%20await%20is%20syntactic%20sugar,the%20producing%20code's%20execution%20call.
- https://www.linkedin.com/pulse/understanding-garbage-collection-javascript-key-memory-tariq-siddiqui-fyocf#:~:text=js)%20primarily%20use%20the%20Mark,roots%20is%20marked%20as%20reachable.
- https://chat.openai.com/c/343d3a3a-765f-4362-8b8b-091870ecf192
- https://www.bacancytechnology.com/blog/python-vs-javascript
- https://stackify.com/python-garbage-collection/
- https://www.calibraint.com/blog/garbage-collection-in-javascript
- https://chat.openai.com/c/b96252f7-2096-4164-ae7a-9f847c37215b