JAVASCRIPT

Programming Languages

6510099 Shann Neil O. Estabillo

6520282 Joseph Chidiebere Anyalogbu

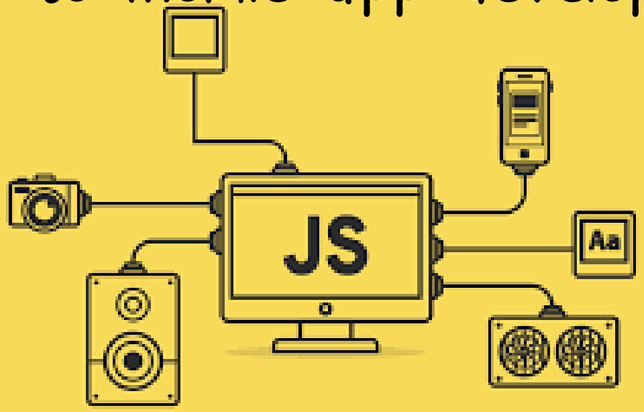
INTRODUCTION

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.

Programming Languages

JAVASCRIPT

JavaScript stands as a ubiquitous language, profoundly shaping the digital landscape. Its versatility spans across web development, enabling interactive user experiences and dynamic content creation. From client-side scripting to server-side applications, JavaScript's influence extends to mobile app development and beyond.



Programming Languages

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");
```

Programming Languages

VARIABLES

```
// 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 in the following slides.

VARIABLE TYPES

There are 2 types of variables in JavaScript: Local variables and Global variables. Local variables declare a variable inside of a block or function. Global variables declare a variable outside a function or with a window object.

VARIABLE TYPES

```
// 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

Programming Languages

OPERATORS

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.

Programming Languages

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

Programming Languages

EXPRESSION

Expression is the combination of values, operators, and variables. It is used to compute the values.

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

Programming Languages

KEYWORDS

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

KEYWORDS

```
// 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(){};
```

Programming Languages

COMMENTS

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.

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

Programming Languages

DATATYPES

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 types and Non-primitive (reference) data types.

Programming Languages

DATATYPES

```
// 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
```

Programming Languages

DATATYPES

```
// 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"}
```

Programming Languages

FUNCTIONS

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.

Programming Languages

FUNCTIONS

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

Programming Languages

FUNCTIONS

-The JavaScript function can contain zero or more arguments.

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

    // Declare a variable
    let num = 45;

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

// Function call
func();
```

Output:

45

Programming Languages

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.

Programming Languages

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.

Programming Languages

OBJECT ORIENTED FEATURES: OBJECTS

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.

Programming Languages

OBJECT ORIENTED FEATURES: OBJECTS

```
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.

Programming Languages

OBJECT ORIENTED FEATURES: OBJECTS

```
// 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.

Programming Languages

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.

OBJECT ORIENTED FEATURES: INSTANCES

```
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.

ACCESING 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.

Programming Languages

ACCESING OBJECT PROPERTIES AND METHODS

```
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.

Programming Languages

ACCESING OBJECT PROPERTIES AND METHODS

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.

Programming Languages

JAVASCRIPT: CLASSES

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.

Programming Languages

JAVASCRIPT: CLASSES

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.

Programming Languages

JAVASCRIPT: CLASSES

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:

Programming Languages

JAVASCRIPT: CLASSES

```
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.

Programming Languages

JAVASCRIPT 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.

Programming Languages

JAVASCRIPT OOP: ENCAPSULATION

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.

Programming Languages

JAVASCRIPT OOP: 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.

Programming Languages

JAVASCRIPT OOP: INHERITANCE

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.

Programming Languages

JAVASCRIPT OOP: POLYMORPHISM

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

Programming Languages

JAVASCRIPT OOP: POLYMORPHISM

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.

Programming Languages

JAVASCRIPT OOP: 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.

Programming Languages

JAVASCRIPT OOP: ABSTRACTION

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.

Programming Languages

JAVASCRIPT OOP: ABSTRACTION

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.

Programming Languages

JAVASCRIPT: 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.

Programming Languages

JAVASCRIPT: ASYNCHRONOUS PROGRAMMING

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

Programming Languages

ASYNCHRONOUS PROGRAMMING: 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.

Programming Languages

ASYNCHRONOUS PROGRAMMING: CALLBACKS

```
  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);
```

Programming Languages

ASYNCHRONOUS PROGRAMMING: CALLBACKS

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

ASYNCHRONOUS PROGRAMMING: CALLBACKS

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.

Programming Languages

ASYNCHRONOUS PROGRAMMING: 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.

Programming Languages

ASYNCHRONOUS PROGRAMMING: PROMISE

A Promise links the producing and the consuming code together. In the example in the next slide, the displayGreeting function is the producing code while the greet function is the consuming code.

ASYNCHRONOUS PROGRAMMING: PROMISE

```
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?`);
}
```

Programming Languages

ASYNCHRONOUS PROGRAMMING: PROMISE

In the example in the following slides, 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.

ASYNCHRONOUS PROGRAMMING: PROMISE

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

Programming Languages

ASYNCHRONOUS PROGRAMMING: PROMISE

```
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)
);
```

Programming Languages

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.

Programming Languages

PROTOTYPE INHERITANCE

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;
```

PROTOTYPE INHERITANCE

```
// 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
```

Programming Languages

PROTOTYPE INHERITANCE

In last 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.

PROTOTYPE INHERITANCE

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.

Programming Languages

GARBAGE COLLECTION ALGORITHM

In this next section, we will talk about the garbage collection algorithm that JavaScript uses. 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.

GARBAGE COLLECTION ALGORITHM

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.

Programming Languages

GARBAGE COLLECTION ALGORITHM

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.

Programming Languages

LANGUAGE DESIGN DECISIONS

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

LANGUAGE DESIGN DECISIONS

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

LANGUAGE DESIGN DECISIONS

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.

LANGUAGE DESIGN DECISIONS

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

SYNTAX AND TYPE SYSTEM

Syntax (key differences)

JAVASCRIPT:

- It follows curly brace syntax.
- Statements are terminated by semicolon.
- Uses camelCase.
- Function definition by 'function' keyword.

PYTHON:

- It follows an indentation-based syntax.
- Statements are not terminated by semicolon.
- Uses snake case
- Function definition by 'def' keyword.

SYNTAX AND TYPE SYSTEM

Type System (key differences)

JAVASCRIPT:

- JavaScript is loosely typed
- It employs dynamic typing.
- Type coercion is common.
- Primitive data types

PYTHON:

- Python is strongly typed.
- It also employs dynamic typing.
- Type coercion is less common
- Has set of built-in data types

SCOPING RULES AND OTHER LANGUAGE FEATURES.

Scoping Rules: (key differences)

JAVASCRIPT:

- Has function-level scope by default
- Variables with 'var' keyword are function-scoped.
- block-level scoping possible with 'let' and 'const' in ES6
- inner functions have access to variables

PYTHON:

- uses block-level scoping
- Variables declared outside of any function/ block have global scope.
- global' and 'nonlocal' keywords allow modifying variables in the global and enclosing scopes
- nested functions have access to variables

SCOPING RULES AND OTHER LANGUAGE FEATURES.

Asynchronous Programming:

JAVASCRIPT:

- supports asynchronous programming.
- makes use of Callbacks
- Promises provide a cleaner way to handle asynchronous operations
- Async/await syntax makes asynchronous code easier.

PYTHON:

- supports asynchronous programming.
- based on coroutines
- Python's asyncio library provides tools for asynchronous I/O..
- The 'async' keyword defines asynchronous functions, while 'await' pause execution.

SCOPING RULES AND OTHER LANGUAGE FEATURES.

Inheritance Models:

JAVASCRIPT:

- supports prototypal inheritance
- Objects are linked to a prototype object
- Classes are based on prototypes under the hood.

PYTHON:

- supports class-based inheritance
- every class inherits from a base class called 'object'
- model is based on the concept of method resolution order (MRO),
- Supports multiple inheritance

SCOPING RULES AND OTHER LANGUAGE FEATURES.

Type Checking

JAVASCRIPT:

- relies on runtime type checking
- static type checking can be performed using type annotations.

PYTHON:

- relies heavily on runtime type checking
- recent versions of Python (3.5 and later) support type hints,

SCOPING RULES AND OTHER LANGUAGE FEATURES.

Object-Oriented Programming

JAVASCRIPT:

- supports OOP through prototypal inheritance.
- Classes introduced in ECMAScript 2015, provides syntactic sugar for definitions.

PYTHON:

- supports OOP with classes and objects.
- follows a class-based inheritance model.
- has a more traditional class syntax compared to JavaScript

Garbage Collection Algorithms (Comparison)

GARBAGE COLLECTION MECHANISM:

JAVASCRIPT:

- primarily uses a tracing garbage collector.
- garbage collector makes periodic checks.
- Collector marks reachable objects and deletes unreferenced objects.

PYTHON:

- combination of reference counting and a cyclic garbage collector.
- Reference counting tracks the number of references.
- cyclic garbage collector detects and collects cyclic references.

MEMORY MANAGEMENT

JAVASCRIPT:

- manages memory automatically.
- developers have no direct control over the garbage collection process.

PYTHON:

- handles memory management automatically too.
- not as efficient as reference counting, especially in cyclic references.

MEMORY LEAKS

JAVASCRIPT:

- memory leaks occur if developers retain references to objects no longer needed.
- caused by circular references, event listeners that are not properly removed, and long-lived objects.

PYTHON:

- memory leaks may still occur if references to objects are retained unnecessarily.
- garbage collector helps mitigate memory leaks by reclaiming memory.
- creating cyclic references, can lead to memory leaks in python.

CONCLUSION

In conclusion, our exploration into the intricacies of JavaScript and its comparison with Python has illuminated the diverse landscape of programming languages and their implications in software development. Beginning with an acknowledgment of the paramount importance of programming languages in shaping software solutions and fostering innovation, we delved into JavaScript, a widely used language, examining its syntax, dynamic typing system, scoping mechanisms, and key features such as asynchronous programming with Promises, prototypal inheritance, closures, and arrow functions. Additionally, we scrutinized its garbage collection algorithm and explored the rationale behind its design decisions, which significantly impact development practices. Through a meticulous comparative analysis with Python, we highlighted differences in syntax, type systems, and other language features, empowering developers to make informed decisions based on project requirements and preferences.

OUR EXAMINATION

Our examination of garbage collection algorithms underscored the importance of memory management for efficient and reliable software systems. It's crucial to comprehend JavaScript's features given its pervasive presence in web development and beyond, and insights from this comparison drive continuous learning and growth within the developer community, fostering innovation and excellence in software development.

THANK YOU

Presentation by Fayget Group