Data types & Variables:

In JavaScript, there are several data types:

- Number: Represents numeric values.
- String: Represents textual data.
- Boolean: Represents either true or false.
- Null: Represents the intentional absence of any object value.
- Undefined: Represents the absence of a defined value.
- Object: Represents a collection of key-value pairs.
- Array: Represents an ordered list of values.
- Symbol: Represents a unique identifier.

Variables are used to store and manipulate data. You can declare variables using the var, let, or const keywords.

Example code:

```
// Number
let age = 25;

// String
let name = "John";

// Boolean
let isStudent = true;

// Null
let myVar = null;

// Undefined
let myVar2;

// Object
let person = { name: "Alice", age: 30 };

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

// Symbol
const id = Symbol();
```

Operators & Control Structures:

JavaScript provides various operators for performing operations on data:

- Arithmetic operators: `+`, `-`, `*`, `/`, `%`.
- Comparison operators: `==`, `===`, `!=`, `!==`, `>`, `<`, `>=`, `<=`.
- Logical operators: `&&`, `||`, `!`.
- Assignment operators: `=`, `+=`, `-=`, `*=`, `/=`.
- Conditional (ternary) operator: `condition ? expression1 : expression2`.
- Increment/Decrement operators: `++`, `--`.
- Bitwise operators: `&`, `|`, `^`, `~`, `<<`, `>>`.

Control structures are used to control the flow of execution in a program. The common control structures in JavaScript are:

- if-else statements: Executes a block of code based on a condition.
- for loop: Repeats a block of code for a specified number of times.
- while loop: Repeats a block of code while a condition is true.
- switch statement: Executes different actions based on different conditions.

Example code:

```
// Arithmetic operators
let sum = 10 + 5;
let product = 3 * 4;

// Comparison operators
let isEqual = 10 === 5;
let isGreater = 10 > 5;

// Logical operators
let isTrue = true && false;
let isFalse = !true;

// Assignment operators
let x = 5;
x += 2; // Equivalent to x = x + 2

// Conditional operator
let result = x > 10 ? "Greater than 10" : "Less than or equal to 10";

// if-else statement
let num = 10;
if (num > 0) {
  console.log("Positive");
} else if (num < 0) {
  console.log("Negative");
} else {
  console.log("Zero");
}

// for loop
for (let i = 0; i < 5; i++) {
  console.log(i);
}

// while loop
let count = 0;
while (count < 5) {
  console.log(count);
  count++;
}

// switch statement
let day = "Monday";
switch (day) {
  case "Monday":
    console.log("It's Monday");
    break;
  case "Tuesday":
    console.log("It's Tuesday");
    break;
  default:
    console.log("It's another day");
}
```

Functions:

Functions in JavaScript are reusable blocks of code that perform a specific task. They allow you to encapsulate logic and execute it whenever needed. Functions can have parameters (inputs) and return a value.

Example code:

```
// Function with parameters and return value
```

```
function add(a, b) {
  return a + b;
}

let result = add(3, 4);
console.log(result); // Output: 7

// Function without parameters and return value
function greet() {
  console.log("Hello!");
}

greet(); // Output: Hello!

// Function expression (assigned to a variable)
const subtract = function(a, b) {
  return a - b;
};

result = subtract(5, 2);
console.log(result); // Output: 3

// Arrow function (shorter syntax for function expression)
const multiply = (a, b) => a * b;

result = multiply(2, 3);
console.log(result); // Output: 6
```

Arrays:

Arrays in JavaScript are used to store multiple values in a single variable. They can contain elements of different types and are zero-based indexed.

Example code:

```
// Creating an array
let fruits = ["apple", "banana", "orange"];

// Accessing array elements
console.log(fruits[0]); // Output: "apple"
```

```
// Modifying array elements
fruits[1] = "grape";
console.log(fruits); // Output: ["apple", "grape", "orange"]

// Array methods
console.log(fruits.length); // Output: 3
fruits.push("pear"); // Adds an element at the end
console.log(fruits); // Output: ["apple", "grape", "orange", "pear"]
fruits.pop(); // Removes the last element
console.log(fruits); // Output: ["apple", "grape", "orange"]
```

Higher Order Functions:

Higher-order functions are functions that can take other functions as arguments or return functions as results. They allow for functional composition and are a powerful feature of JavaScript.

Example code:

```
// forEach: Executes a provided function once for each array element
let numbers = [1, 2, 3];
numbers.forEach(function(number) {
  console.log(number);
});

// map: Creates a new array with the results of calling a function on each element
let squaredNumbers = numbers.map(function(number) {
  return number ** 2;
});
console.log(squaredNumbers); // Output: [1, 4, 9]

// filter: Creates a new array with elements that pass a test
let evenNumbers = numbers.filter(function(number) {
  return number % 2 === 0;
});
console.log(evenNumbers); // Output: [2]

// reduce: Applies a function to reduce the array to a single value
let sum = numbers.reduce(function(acc, number) {
  return acc + number;
}, 0);
console.log(sum); // Output: 6
```

Pure Functions:

Pure functions are functions that always return the same result for the same input and do not have any side effects. They rely only on their arguments and do not modify external state.

Example code:

```
// Impure function (with side effect)
let totalPrice = 0;

function calculateTotal(price) {
  totalPrice += price; // Modifies external state
  return totalPrice;
}

console.log(calculateTotal(10)); // Output: 10
console.log(calculateTotal(5)); // Output: 15

// Pure function
function add(a, b) {
  return a + b; // Always returns the same result for the same inputs
}

console.log(add(2, 3)); // Output: 5
console.log(add(2, 3)); // Output: 5
```

Objects:

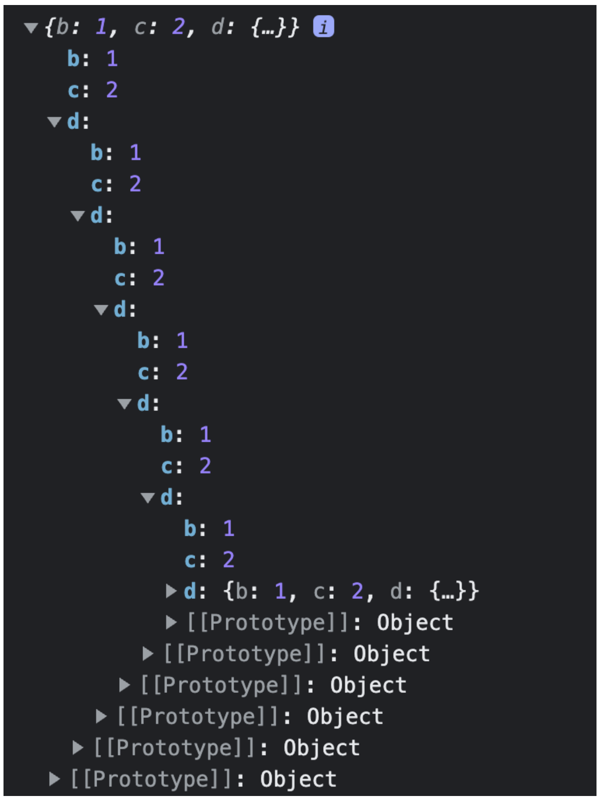
Objects in JavaScript are collections of key-value pairs. They can represent real-world entities and have properties (values associated with a key) and methods (functions associated with an object).

Example code:

```
// Creating an object
let person = {
  name: "John",
  age: 25,
  greet: function() {
    console.log("Hello!");
  }
};

// Accessing object properties
console.log(person.name); // Output: "John"

// Modifying object properties
person.age = 30;
console.log(person.age); // Output: 30

// Calling object methods
person.greet(); // Output: "Hello!"

// Adding new properties
person.location = "New York";
console.log(person.location); // Output: "New York"
```

Infinitely Nesting Object

```
const a = {b : 1, c: 2}
a.d = a;
console.log(a);
```



```
▼{b: 1, c: 2, d: {…}} i
   b: 1
   c: 2
  ▼d:
     b: 1
     c: 2
    ▼d:
       b: 1
       c: 2
      ▼d:
         b: 1
         c: 2
        ▼d:
           b: 1
           c: 2
          ▼d:
             b: 1
             c: 2
            ▶d: {b: 1, c: 2, d: {…}}
            ▶[[Prototype]]: Object
          ▶[[Prototype]]: Object
        ▶[[Prototype]]: Object
      ▶[[Prototype]]: Object
    ▶[[Prototype]]: Object
  ▶[[Prototype]]: Object
```

Q) Why is

    console.log(2 < 5 < 8) true,
    console.log(8 > 5 > 2) false,
    console.log(2 < 5 > 8) false

in Javascript. Explain

Ans) In JavaScript, comparison operators evaluate expressions from left to right, and they have left-to-right associativity. This means that when you have multiple comparison operators in a single expression, they are evaluated one after the other, following the order in which they appear.

1. console.log(2 < 5 < 8): This expression is evaluated as follows:
    - 2 < 5 is true because 2 is indeed less than 5.
    - The result of the above comparison is then compared to 8, so the expression becomes true < 8.
    - In JavaScript, when comparing a boolean (true) to a number (8), the boolean is automatically coerced into a number. true is converted to 1, and the expression becomes 1 < 8.
    - Since 1 is indeed less than 8, the final result is true. Therefore, console.log(2 < 5 < 8) outputs true.
2. console.log(8 > 5 > 2): This expression is evaluated as follows:
    - 8 > 5 is true because 8 is greater than 5.
    - The result of the above comparison is then compared to 2, so the expression becomes true > 2.
    - Again, JavaScript coerces the boolean (true) into a number (1), so the expression becomes 1 > 2.
    - Since 1 is not greater than 2, the final result is false. Therefore, console.log(8 > 5 > 2) outputs false.
3. console.log(2 < 5 > 8): This expression is evaluated as follows:
    - 2 < 5 is true because 2 is less than 5.
    - The result of the above comparison is then compared to 8, so the expression becomes true > 8.
    - Again, JavaScript coerces the boolean (true) into a number (1), so the expression becomes 1 > 8.
    - Since 1 is not greater than 8, the final result is false. Therefore, console.log(2 < 5 > 8) outputs false.

Prepared By :
Divyansh Singh (LinkedIn : rgndunes)