JavaScript

Data types

What are Data Types

- Data types are categories of values in programming languages that define how the computer or interpreter understands and manipulates the data.
- Data types tell the computer how to store and work with different types of data in our programs.

Why are Data Types Important in JavaScript?

- Data types in JavaScript determine how values are stored and manipulated in memory.
- Understanding data types helps prevent common errors and bugs that can occur when values are used incorrectly.
- Knowing how to use the different data types in JavaScript is essential for writing complex programs that are scalable and maintainable.

JavaScript data types are divided into two categories

- Primitive Data Types: Primitive data types are basic or simple data types that are predefined by the programming language. They are stored directly into the computer memory

- Referenced Data Types: reference data types are not stored directly in memory, but instead are stored as a reference to their location in memory. They are made up of two or more primitive data types

Primitive Data Types

- Number: used to represent numeric values like 42 or 3.14.
- String: used to represent text or characters, like "Hello, world!" or "JavaScript".
- Boolean: used to represent true/false values.
- Undefined: used to represent a variable that has been declared but has not been assigned a value yet.
- Null: used to represent the intentional absence of any object value.
- Symbol: used to create unique identifiers for objects.

Referenced Data Types

- Objects: used to represent a collection of related data, including functions and other objects. Objects in JavaScript are created using the {} notation
- Arrays: used to store a collection of related data, like a list of numbers or names. Arrays in JavaScript are created using the [] notation or the new Array() constructor.
- Functions: used to represent reusable blocks of code that can be called with different arguments. Functions in JavaScript are created using the function keyword, and can be defined as named functions or anonymous functions.

Using console to Identify Data Types

```
let check;
Const  battery = null;
Const num = 56
Const name = “introduction to data types”;
Const sym = Symbol();
Const happy = true;

Const obj = {a:1, b:”bee”, c:3};
Const arr = [1,2,4,’a’,null,5];
Const myFunct = function (){
        console.log(“my function”);
}
```

```
console.log(typeof(check));
console.log(typeof(battery));
console.log(typeof(num));
console.log(typeof(name));
console.log(typeof(sym));
console.log(typeof(happy));
console.log(typeof(obj));
console.log(typeof(arrays));
console.log(typeof(myFunct));
```

Dynamically Typed languages

JavaScript is a dynamically typed programming language, which means that you don't have to specify the data type of a variable when you declare it. Instead, the data type of the variable is determined automatically at runtime based on the value that you assign to it.

Statically Typed languages

Statically typed programming languages require you to declare the data type of a variable before you can use it in your code. This means that you must explicitly specify the data type of each variable when you declare it, and that data type cannot be changed later in the program.

To turn Javascript into a statically typed language, we use Typescript.

Assignment 1

Differences between null and undefined

JS Operators

There are several types of operators in JavaScript, including:

1. Arithmetic operators: used to perform basic mathematical calculations, such as addition, subtraction, multiplication, and division.
2. Comparison operators: used to compare two values and return a boolean value, such as true or false.
3. Logical operators: used to combine or invert boolean values, such as AND (&&), OR (||), and NOT (!).
4. Assignment operators: used to assign a value to a variable, such as the equal sign (=).
5. Bitwise operators: used to perform bit-level operations on numeric values, such as AND (&), OR (|), and XOR (^).
6. Unary operators: used to operate on a single operand, such as the increment operator (++) or the negation operator (-).
7. Conditional (ternary) operator: a shorthand way of writing an if-else statement, such as condition ? value1 : value2.
8. Other operators: concatenation (+), property access (.), rest operators, spread operators, void, delete

Arithmetic Operators

They are used to perform basic mathematical operations on numeric values

1. Addition (+): used to add two or more numbers.
2. Subtraction (-): used to subtract one number from another.
3. Multiplication (*): used to multiply two or more numbers.
4. Division (/): used to divide one number by another.
5. Modulus (%): used to return the remainder of a division operation.
6. Increment (++): used to increase the value of a number by 1.
7. Decrement (--): used to decrease the value of a number by 1.

Open a file on your vs code, for each number, give 2 examples.

Comparison Operators

They are used to compare two values and return a Boolean value (true or false) depending on the comparison result

1. Equal to (==): used to check if two values are equal, regardless of their data type.
2. Not equal to (!=): used to check if two values are not equal, regardless of their data type.
3. Strict equal to (===): used to check if two values are equal and have the same data type.
4. Strict not equal to (!==): used to check if two values are not equal or do not have the same data type.
5. Greater than (>): used to check if one value is greater than another.
6. Greater than or equal to (>=): used to check if one value is greater than or equal to another.
7. Less than (<): used to check if one value is less than another.
8. Less than or equal to (<=): used to check if one value is less than or equal to another.

Logical Operators

They are used to combine two or more Boolean expressions and return a Boolean value (true or false) depending on the result of the combination.

1. Logical AND (&&): used to combine two expressions, and returns true only if both expressions are true.
2. Logical OR (||): used to combine two expressions, and returns true if at least one of the expressions is true.
3. Logical NOT (!): used to invert the value of an expression, returning true if the expression is false, and false if the expression is true.

Assignment Operators

Assignment operators in JavaScript are used to assign a value to a variable.

1. Simple assignment (=): used to assign a value to a variable, for example x = 10.
2. Add and assign (+=): used to add a value to the variable and assign the result to the same variable, for example x += 5 is equivalent to x = x + 5.
3. Subtract and assign (-=): used to subtract a value from the variable and assign the result to the same variable, for example x -= 2 is equivalent to x = x - 2.
4. Multiply and assign (*=): used to multiply the variable by a value and assign the result to the same variable, for example x *= 3 is equivalent to x = x * 3.
5. Divide and assign (/=): used to divide the variable by a value and assign the result to the same variable, for example x /= 2 is equivalent to x = x / 2.
6. Modulus and assign (%=): used to calculate the remainder of dividing the variable by a value and assign the result to the same variable, for example x %= 4 is equivalent to x = x % 4.

Bitwise Operators

Bitwise operators are often used in low-level programming they are used to manipulate the binary representation of numbers

1. AND (&): performs a bitwise AND operation between two numbers. It returns a number where each bit is set to 1 only if the corresponding bits of both input numbers are also 1.
2. OR (|): performs a bitwise OR operation between two numbers. It returns a number where each bit is set to 1 if either of the corresponding bits of the input numbers is 1.
3. XOR (^): performs a bitwise exclusive OR (XOR) operation between two numbers. It returns a number where each bit is set to 1 if the corresponding bits of the input numbers are different (one is 0 and the other is 1).
4. NOT (~): performs a bitwise NOT operation on a number. It returns a number where each bit is flipped (0 becomes 1 and 1 becomes 0).
5. Left shift (<<): shifts the bits of a number to the left by a specified number of positions. The empty bits on the right are filled with 0.
6. Right shift (>>): shifts the bits of a number to the right by a specified number of positions. The empty bits on the left are filled with the sign bit (the leftmost bit, which indicates whether the number is positive or negative).
7. Zero-fill right shift (>>>): shifts the bits of a number to the right by a specified number of positions. The empty bits on the left are filled with 0.

Unary Operators

Unary operators are operators that operate on a single operand, which means they require only one operand.

1. Unary plus (+): This operator tries to convert its operand to a number.
2. Unary minus (-): This operator negates its operand.
3. Increment (++): This operator adds one to its operand.
4. Decrement (--): This operator subtracts one from its operand.
5. Logical NOT (!): This operator returns the opposite of its operand's truthiness.
6. typeof: This operator returns a string that indicates the type of its operand.
7. delete: This operator deletes an object, a property, or an element at a specified index in an array.

Ternary Operators

Ternary operators are operators that take three operands and are used as a shorthand for conditional statements

condition ? expression1 : expression2

If the condition is true, then the expression1 is executed, otherwise expression2 is executed.

```
let age = 18;
let status = (age >= 18) ? "adult" : "minor";
console.log(status); // "adult"
```

Other Operators

1. Concatenation: This is the process of combining two or more strings into a single string. This is typically done using the "+" operator
2. Rest Parameters: The rest parameter syntax allows us to represent an indefinite number of arguments as an array.
3. Spread Operator: The spread operator can be used to expand an array into individual elements.
4. Destructuring Assignment: The destructuring assignment syntax allows us to unpack values from arrays or objects into distinct variables.

Pitfalls in Javascript

1. Type coercion: JavaScript is a dynamically-typed language, which means that variables can change data types on the fly. This can lead to unexpected results when performing operations that rely on specific data types. For example:
2. NaN: NaN (Not a Number) is a special value in JavaScript that represents an invalid mathematical operation. NaN is a number, but it is not equal to any other number, including itself. This can lead to unexpected behavior when performing operations that result in NaN. For example:
3. Operator precedence: JavaScript has a specific order of operations for operators, which can lead to unexpected results if not used properly. For example:
4. Floating-point precision: JavaScript, like most programming languages, uses floating-point numbers to represent decimal values. However, because of the way floating-point numbers are stored in memory, they can sometimes result in imprecise calculations. For example:
5. behavior of the "==" and "===" comparison operators.
6. behavior of the "+" operator with strings. (the expression 1 + 2 + "3" will result in the string "33")

End

Data types and operations

Assignment

1. Create a program that asks the user for two numbers and then outputs the result of adding, subtracting, multiplying, and dividing those numbers. Make sure to handle any potential errors or unexpected input.
2. Write a program that takes a string input from the user and then outputs the length of the string, as well as whether the string contains any numbers or special characters.
3. Create a program that takes an array of numbers as input and outputs the sum of those numbers. Use a loop to iterate over the array and add up the numbers.
4. Write a program that takes two strings as input and then checks if they are equal using the "===" operator. If they are equal, output a message saying so. If they are not equal, output a message saying that they are not equal.
5. Create a program that takes a number as input and then checks if it is even or odd using the modulus operator (%). Output a message saying whether the number is even or odd.