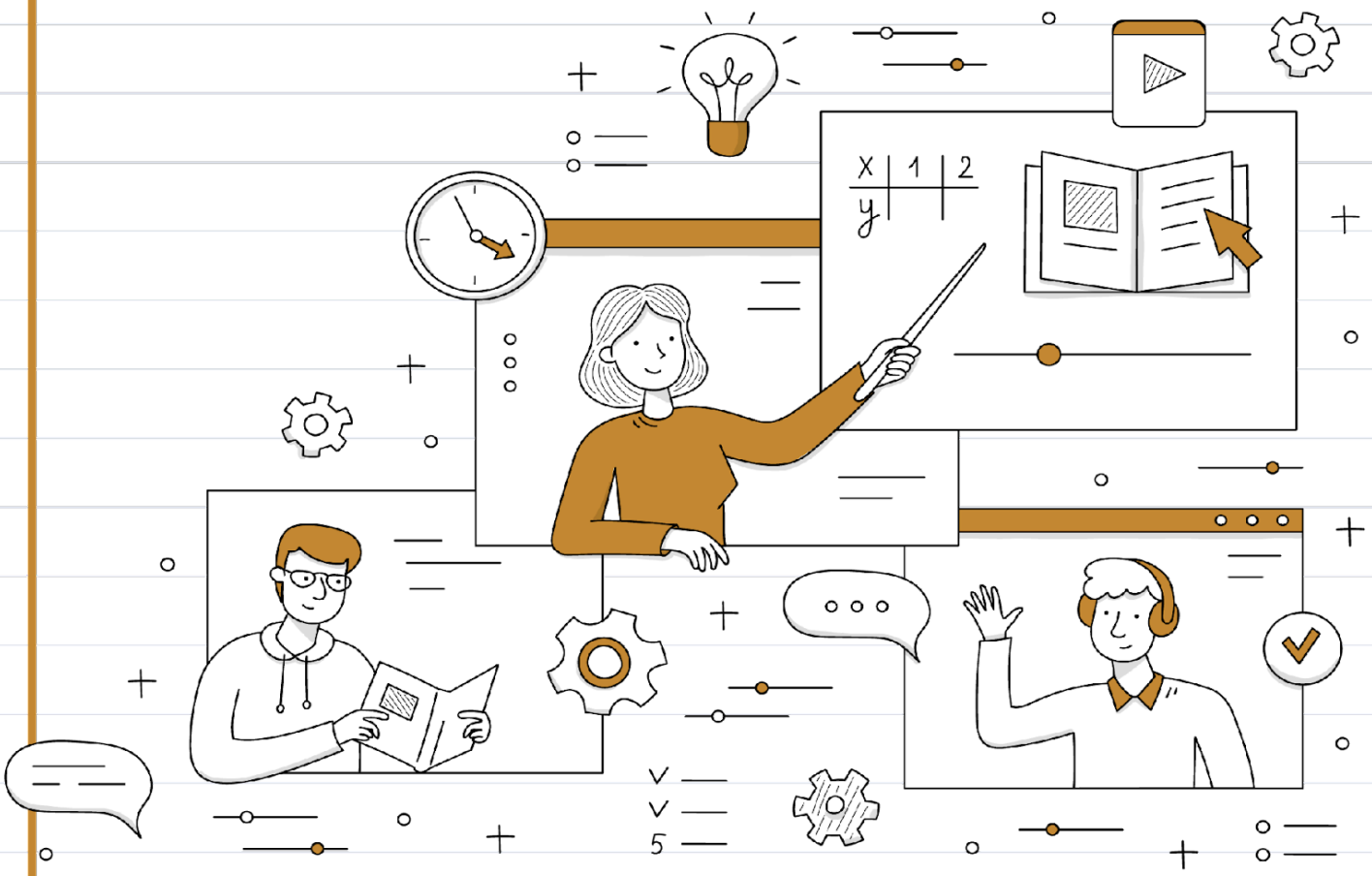
SUNSTONE

JavaScript

LECTURE 2 NOTES

Lexical Structure



- **What is a Character Set?**

  A character set is a defined collection of characters, symbols, and digits that are available for use in textual data. These character sets are typically represented by unique numeric codes assigned to each character. In JavaScript, the character set used is Unicode.

- **What is Unicode?**

  - Unicode is a universal character encoding standard that aims to provide a consistent and unique representation for every character used in written languages worldwide, regardless of the platform, language, or program.
  - It is designed to support the representation and processing of text in different writing systems, including alphabets, ideographs, symbols, and emojis.
  - Unicode uses a unique numeric value, called a code point, to represent each character.

- **Character Sets and Unicode in JavaScript**

  - In JavaScript, character sets and Unicode are handled through the use of strings and the built-in Unicode support in the language. JavaScript strings are sequences of UTF-16 code units, which means they can represent characters from the Unicode character set.
  - Here are some important aspects to consider regarding character sets and Unicode in JavaScript:
  - **String Literals:** JavaScript allows you to represent strings using single quotes (''), double quotes ("") or backticks (``). String literals can include Unicode characters directly by using escape sequences or Unicode code point escapes. For example:

    ```
    let myString = "Hello, \u03A9"; // Greek capital letter omega (Ω)
    console.log(myString);
    ```

  - **Output:**
    ```
    Hello, Ω
    ```

  - **String Manipulation:** JavaScript provides several built-in methods for manipulating strings, such as `charAt()`, `charCodeAt()`, `substring()`, `slice()`, and more. These methods work on the individual characters of the string, including Unicode characters.
  - **String Length:** The length of a JavaScript string is determined by the number of

UTF-16 (Unicode Transformation Format)code units it contains, not necessarily the number of visible characters. Some Unicode characters may require more than one code unit to represent them. To obtain the actual number of characters, you can use the length property or the `Array.from()` method.

```
let myString = "Hello,😊";
console.log(myString.length);
console.log(Array.from(myString).length);
```

- **Output:**
  ```
  8 (5 characters +1 , + 2 code unit for the emoji)
  7 (5 characters +1 , + 1 code unit for the emoji)
  ```

- **String Methods**: JavaScript's string methods, such as `indexOf()`, `replace()`, `toUpperCase()`, `toLowerCase()`, and others, work correctly with Unicode characters.

```
let myString = "Hello,😊";
console.log(myString.indexOf("😊"));
console.log(myString.toUpperCase());
```

- **Output:**
  ```
  6
  HELLO, 😊
  ```

- JavaScript provides robust support for working with character sets and Unicode, allowing you to manipulate and process strings that contain characters from various writing systems.

- **What are Tokens?**

  - Tokens are the smallest individual units of a program, which include keywords, identifiers, operators, punctuation, and literals.

  - They are the smallest individual words, phrases, or characters that JavaScript can understand. When JavaScript is interpreted, the browser parses the script into these tokens while ignoring comments and white space.

  - JavaScript tokens fit in the following categories:

    1) Identifiers
    2) Keywords
    3) Literals

4) Operators
5) Constants
6) Strings
7) Special Characters

- **Identifiers**

  - Identifiers are simply names that represent variables, methods, or objects. They consist of a combination of characters and digits. Some names are already built into the JavaScript language and are therefore reserved; these identifiers are called 'Keywords'. Aside from these keywords, you can define your own creative and meaningful identifiers.

    Some important properties of identifiers in JavaScript that you have to follow:

    1. **Valid Characters:** Identifiers can contain letters (both uppercase and lowercase), digits, underscores (_), and dollar signs ($). However, the first character of an identifier cannot be a digit.

    2. **Case Sensitivity:** JavaScript is case-sensitive, so identifiers like 'myVariable' and 'myvariable' would be treated as two different entities.

    3. **Reserved Words:** You cannot use reserved words (also known as keywords) as identifiers. These are words that have predefined meanings in JavaScript and are used for specific purposes, such as if, else, for, function, etc.

    4. **Unicode Support:** JavaScript allows Unicode characters in identifiers. This means you can use characters from non-English languages, such as Greek, Cyrillic, or Chinese characters.

    5. **Length Limitations:** There is no specific limit on the length of an identifier in JavaScript. However, it is good practice to keep identifiers concise and meaningful.

    6. **Naming Conventions:** While there are no strict rules enforced by the JavaScript language, there are common naming conventions followed by developers. The most widely used convention is camelCase, where the first word starts with a lowercase letter and subsequent words start with uppercase letters (for example, myVariableName).

  - **Valid identifiers:**
    - myVariable
    - _privateVariable

    - $element
    - userName
    - PI
    - first_name
    - ἀπόδοσις

- **Invalid identifiers:**
    - 123abc (starts with a digit)
    - my-variable (contains a hyphen)
    - if (a reserved keyword)
    - function (a reserved keyword)
    - var (a reserved keyword)
    - my variable (contains a space)
    - first-name (contains a hyphen)

- **Keywords**
    - Keywords are predefined identifiers that make up the core of a programming language. In JavaScript, they perform unique functions such as declaring new variables and functions, making decisions based on the present state of the computer, or starting a repetitive loop inside your application. Keywords, which are built into JavaScript, are always available for use by the programmer but must follow the correct syntax.

      Some keywords:
        - break
        - continue
        - else
        - false
        - for

- function
- if
- in
- int
- new
- null
- return
- this
- true
- var
- while
- with

There are a total of 63 keywords that JavaScript provides to programmers.

Some important properties of keywords in JavaScript:

- **Reserved:** Keywords are reserved for specific purposes and cannot be used as identifiers (variable names, function names, etc.). Attempting to use a keyword as an identifier will result in a syntax error.
- **Fixed meaning:** Keywords have fixed meanings assigned by the JavaScript language. They represent specific language constructs or actions.
- **Case sensitivity:** Keywords are case-sensitive. For example, if and IF are treated as different keywords in JavaScript.
- **Language-specific:** Keywords are specific to the JavaScript language and may differ from keywords in other programming languages. Each programming language has its own set of keywords.
- **Syntax elements:** Keywords are used to define control structures, variable declarations, loops, conditionals, function definitions, and more. They provide the fundamental building blocks for writing JavaScript code.

Keywords are also Reserved Words

- **Literals**

- Literals are data composed of numbers or strings used to represent fixed values in JavaScript. They are values that do not change during the execution of your scripts. In JavaScript, literals represent fixed values that are directly written into your code. They are constant values that are not stored in variables but are used to represent specific data types directly.

- The following are the different types of literals that you can use:

    1. **Numeric literals:** These represent numeric values and can be written in various formats such as decimal, binary, octal, or hexadecimal.

       Examples include `123`, `3.14`, `0b1010` (binary), `0o777` (octal), and `0xFF` (hexadecimal).

    2. **String literals:** These represent textual data enclosed in single quotes (''), double quotes ("") or backticks (``).

       Examples include `'Hello'`, `"World"`, and Template literal ``` `` ```.

    3. **Boolean literals:** These represent boolean values, either true or false.

    4. **Object literals:** These represent key-value pairs enclosed in curly braces ({}), used to define objects.

       Example: `{ name: 'John', age: 25 }`

    5. **Array literals:** These represent lists of values enclosed in square brackets ([]).

       Example: `[1, 2, 3, 4]`

    6. **Regular expression literals:** These represent patterns used for pattern matching. They are enclosed between slashes (/).

       Example: `/[a-z]+/`

    7. **Null literal:** This represents the absence of any object value and is denoted by the keyword null.

    8. **Undefined literal:** This represents a variable or object that has been declared but has not been assigned any value, denoted by the keyword undefined.

- **Operators**

    - Operators in JavaScript are symbols or special keywords that are used to perform operations on values or variables.

- They allow you to manipulate data, perform calculations, compare values, and more.
- JavaScript includes a wide range of operators that can be categorized into several types:

    1. **Arithmetic Operators:** Used for mathematical calculations, such as addition (+), subtraction (-), multiplication (*), division (/), modulus (remainder) (%), increment (++), and decrement (--).

    2. **Assignment Operators:** Used to assign values to variables, such as simple assignment (=), addition assignment (+=), subtraction assignment (-=), multiplication assignment (*=), division assignment (/=), and more.

    3. **Comparison Operators:** Used to compare values and determine their relationship, such as equal to (==), not equal to (!=), strict equal to (===), strict not equal to (!==), greater than (>), less than (<), greater than or equal to (>=), and less than or equal to (<=).

    4. **Logical Operators:** Used to combine or manipulate boolean values, such as logical AND (&&), logical OR (||), and logical NOT (!).

    5. **Bitwise Operators:** Used to manipulate individual bits of numeric values, such as bitwise AND (&), bitwise OR (|), bitwise XOR (^), bitwise NOT (~), left shift (<<), right shift (>>), and zero-fill right shift (>>>).

    6. **Ternary Operator:** A conditional operator that evaluates a condition and returns one of two expressions based on the result.

       It has the form:
       `condition ? expression1 : expression2`

    7. **`typeof` Operator:** Used to determine the data type of a value or variable.

    8. **`instanceof` Operator:** Used to check whether an object belongs to a specific class or constructor.

- **Constants**

  In JavaScript, constants are variables that hold values that cannot be reassigned or changed once they are defined. They provide a way to store fixed values that remain constant throughout the execution of a program. Constants are declared using the `const` keyword.

  Here's an example:

```
const PI = 3.14;
console.log(PI); // Output: 3.14

// Attempting to reassign a constant will result in an error
PI = 3.14159; // Throws an error: Assignment to constant variable
```

In the example above, PI is declared as a constant with the value of 3.14. The const keyword ensures that the value of PI remains constant and cannot be modified later in the program. If you try to reassign a value to a constant, it will result in a runtime error.

Note: Constants are block-scoped, just like variables declared with let. They have block-level scope and are only accessible within the block in which they are defined.

Constants are commonly used to store values that should not be changed, such as mathematical constants, configuration settings, or any other value that should remain constant throughout the execution of a program.
Some Properties of Constants in JavaScript are:

- **Declaration:** Constants are declared using the const keyword followed by the identifier name. For example: const PI = 3.14;
- **Immutability:** The value assigned to a constant cannot be changed or reassigned once it is defined. Any attempt to reassign a value to a constant will result in a runtime error.
- **Block Scope:** Constants are block-scoped, meaning they are only accessible within the block in which they are defined. If a constant is declared within a function or a block (enclosed within curly braces), it is only accessible within that function or block.

- **Strings**

In JavaScript, a string is a data type used to represent a sequence of characters. Strings are enclosed in single quotes (''), double quotes ("") or backticks (``). Here are some important aspects and operations related to strings in JavaScript:

Declaration:

```
let message = 'Hello';  // Using single quotes
let name = "John";      // Using double quotes
let template = `Hello, ${name}!`;    // Using backticks for template literals
```

- **Length**: The length property of a string returns the number of characters in the string.

```
let str = 'Hello';
console.log(str.length);  // Output: 5
```

- **Accessing Characters:** Individual characters within a string can be accessed using bracket notation or `charAt()` method.

```
let str = 'Hello';
console.log(str[0]);      // Output: 'H'
console.log(str.charAt(1));  // Output: 'e'
```

- **Concatenation:** Strings can be concatenated using the concatenation operator (+) or using template literals.

```
let str1 = 'Hello';
let str2 = 'World';
let result = str1 + ' ' + str2;
console.log(result);    // Output: 'Hello World'

let template = `${str1} ${str2}!`;
console.log(template);  // Output: 'Hello World!'
```

- **String Methods:** JavaScript provides various built-in methods for manipulating and working with strings. Some common methods include `toUpperCase()`, `toLowerCase()`, `substring()`, `indexOf()`, `split()`, `trim()`, and many more.

- **Escape Characters:** Certain characters have special meaning in strings, such as quotes or newline characters. To include these characters as part of a string, you can use escape characters preceded by a backslash (\).

```
let message = 'She said, "Hello!"';
console.log(message);  // Output: 'She said, "Hello!"'

let path = 'C:\\Program Files\\App';
console.log(path);     // Output: 'C:\Program Files\App'
```

Strings in JavaScript are immutable, meaning once a string is created, it cannot be changed. However, you can perform operations on strings to create new strings. The various string operations and methods in JavaScript allow you to manipulate, extract, and transform string data to suit your programming needs.

- **Special Characters**

When writing scripts, you might sometimes need to tell the computer to use a special character or keystroke such as a tab or carriage return. To do this, use a backslash in front of one of the special characters as shown in the following list:

- **Backslash (\):** Used as an escape character to represent special characters or sequences.
  Example:
  ```
  let message = "This is a \"quoted\" string.";
  ```

- **Single Quote (') and Double Quote ("):** Used to enclose string literals.
  Example:
  ```
  let str1 = 'Hello';
  let str2 = "World";
  ```

- **Newline (\n)**: Represents a line break or new line character.
  Example:
  ```
  let multiline = "Hello\nWorld";
  ```

- **Carriage Return (\r):** Represents a carriage return character.
  Example:
  ```
  let carriageReturn = "Hello\rWorld";
  ```

- **Tab (\t):** Represents a tab character.
  Example:
  ```
  let indented = "Hello\tWorld";
  ```

- **Backspace (\b):** Represents a backspace character.
  Example:
  ```
  let text = "Hello\bWorld";
  ```

- **Null (\0):** Represents the null character.
  Example:
  ```
  let nullChar = "Hello\0World";
  ```

- **Unicode Escapes:** Special characters can be represented using Unicode escape sequences in the form `\uXXXX`, where `XXXX` is the hexadecimal Unicode value.
  Example:
  ```
  let unicode = "\u03A9"; // Represents the Greek letter Omega
  ```

  These special characters are often used when dealing with string literals, particularly when you need to represent characters that would otherwise be interpreted as part of the JavaScript syntax or when you want to include non-printable characters in your strings.

- **Variable**

  In JavaScript, variables are used to store and manipulate data. They provide a way to hold values that can be referenced and modified throughout your program. Here's an overview of variables in JavaScript:

1. **Variable Declaration:** Variables in JavaScript can be declared using the `var`, `let`, or `const` keywords.

    a) `var`: Variables declared with var have function scope or global scope. They can be accessed and modified throughout the function or the global scope.

    Example:
    ```
    var name = "John";
    ```

    b) `let`: Variables declared with let will have block scope. They are limited in scope to the block (enclosed within curly braces) in which they are defined.

    Example:
    ```
    let age = 25;
    ```

    c) `const`: Constants are variables that cannot be reassigned after they are declared. They have block scope and behave like let variables, but their value remains constant.

    Example:
    ```
    const PI = 3.14;
    ```

2. **Variable Assignment:** Variables can be assigned values using the assignment operator (=).

    Example:
    ```
    let score = 100;
    ```

3. **Variable Naming Rules:**

    - Variables must start with a letter, underscore (_), or dollar sign ($).
    - They can contain letters, digits, underscores, or dollar signs.
    - Variables are case-sensitive.
    - It's recommended to use descriptive names that convey the purpose of the variable.

4. **Variable Scope:** The scope of a variable determines its visibility and lifetime within a program.

    - **Function Scope:** Variables declared with var have function scope. They are accessible within the function they are defined in.
    - **Block Scope:** Variables declared with `let` and `const` have block scope.

They are limited to the block they are defined in, such as within loops or if statements.

- **Global Scope:** Variables declared outside of any function have global scope. They can be accessed and modified from anywhere in the program.

5. **Variable Reassignment:** Variables declared with var or let can be reassigned with a new value.

   Example:
   ```
   let count = 5;
   count = count + 1;
   ```

6. **Variable Types:** JavaScript is dynamically typed, meaning variables can hold values of any data type. The type of a variable is determined by the value it currently holds.

- **Comments and White Spaces**

  ➢ **Comments**

  Comments are used to add explanatory notes or annotations within your code. They are ignored by the JavaScript interpreter and are solely meant for human readers to understand the code better.

  There are two types of comments in JavaScript:

  1) Single-line comments

  2) Multi-line comments

  - **Single-line comments** start with two forward slashes (//) and continue until the end of the line.

    For example:

    ```
    // This is a single-line comment
    var name = "John"; // Assigning a value to the variable "name"
    ```

  - **Multi-line comments**, also known as block comments, begin with a forward slash followed by an asterisk (/*) and end with an asterisk followed by a forward slash (*/). They can span across multiple lines.

    For example:

```
/*
This is a multi-line comment.
It can contain multiple lines of text.
*/
var age = 25; // Assigning a value to the variable "age"
```

Comments are useful for explaining code functionality, providing context, or temporarily disabling specific lines of code without deleting them.

➢ White Spaces

Whitespace in JavaScript refers to spaces, tabs, and line breaks that are used for formatting and readability purposes. While JavaScript ignores most whitespace, it plays a vital role in making code more legible and organized.

Here are a few common use cases for whitespace in JavaScript:

1) **Separating Tokens:** Whitespace is used to separate different elements in the code, such as keywords, variables, operators, and punctuation. For example:

   ```
   var x = 10 + y * 2;
   ```

   In the above code, spaces are used to separate the assignment operator (=) and the arithmetic operators (+ and *), making the code easier to read.

2) **Indentation:** Proper indentation using whitespace helps in visually organizing blocks of code, such as loops, conditionals, and function definitions. It improves code readability and makes it easier to understand the code's structure.

   For example:

   ```
   if (condition) {
       // Code block
       statement1;
       statement2;
   } else {
       // Code block
       statement3;
       statement4;
   }
   ```

   In the above code, the indentation with spaces or tabs helps to identify the scope and nesting of the if statement and the associated code blocks.

3) **Line Breaks:** Line breaks, often represented as newline characters, are used to separate lines of code. They are especially useful when code becomes lengthy or when it is desired to improve readability. For example:

```
var message = "Hello, " +
              "world!";
```

In the above code, the line break after the concatenation operator (+) enhances readability by clearly separating the two parts of the concatenated string.

Note: JavaScript interprets excessive whitespace between tokens as a single space. This means that multiple consecutive spaces or tabs are treated the same as a single space.