Javascript Cheatsheet

DATA TYPES

JavaScript has several data types, including:

- **Number**: numeric values, such as **1, 3.14, -10, etc.**
- **String**: text values, enclosed in quotes, such as **"hello", 'world', "123", etc.**
- **Boolean**: logical values, either **true or false**.
- **Undefined**: a variable that has been **declared but not assigned** a value.
- **Null**: a variable that has been explicitly assigned a value of null.
- **Object**: a collection of **key-value pairs**, such as **{name: "Pugazh", age: 30}**.
- **Array**: a **collection of values**, such as **[1, 2, 3].**

VARIABLES

Variables are used to store data values in JavaScript. The `` `var` ``, `` `let` ``, and `` `const` `` keywords are used to declare variables.

```
var x = 5; // global variable
let y = "hello"; // block-scoped variable
const z = true; // constant (cannot be reassigned)
```

OPERATORS

JavaScript has several operators that can be used to perform operations on values, including:

- **Arithmetic** operators, such as **+, -, *, /, %**
- **Assignment** operators, such as **=, +=, -=, *=, /=**
- **Comparison** operators, such as **==, ===, !=, !==, <, >, <=, >=**
- **Logical** operators, such as **&&, ||, !**

```
let x = 5;
let y = 10;
let sum = x + y; // 15
let product = x * y; // 50

if (x < y) {
  console.log("x is less than y");
}

if (x == 5 && y == 10) {
  console.log("x is 5 and y is 10");
}
```

CONDITIONALS

Conditionals are used to execute different blocks of code based on a condition. JavaScript has several conditional statements, including `if`, `else`, `else if`, and `switch`.

```
let age = 18;
if (age >= 18) {
  console.log("You are an adult.");
} else {
  console.log("You are not yet an adult.");
}

let day = "Monday";
switch (day) {
  case "Monday":
    console.log("Today is Monday.");
    break;
  case "Tuesday":
    console.log("Today is Tuesday.");
    break;
  default:
    console.log("Today is not Monday or Tuesday.");
}
```

LOOPS

Loops are used to execute a block of code multiple times. JavaScript has several loop statements, including `` `for` ``, `` `while` ``, and `` `do-while` ``.

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

let i = 0;
while (i < 5) {
  console.log(i);
  i++;
}

let j = 0;
do {
  console.log(j);
  j++;
} while (j < 5);
```

FUNCTIONS

Functions are used to group a block of code and execute it whenever it is called. Functions can take parameters and return values.

```
function addNumbers(a, b) {
  return a + b;
}
let sum = addNumbers(5, 10); // 15
```

```
function sayHello(name) {
  console.log(`Hello, ${name}!`);
}
sayHello("Pugazh"); // Hello, Pugazh!
```

ARRAYS

Arrays are used to store a collection of values in JavaScript. Arrays can be declared using square brackets `` `[]` `` and values can be accessed using their index number.

```
let fruits = ["apple", "banana", "orange"];
console.log(fruits[0]); // "apple"
console.log(fruits[1]); // "banana"
console.log(fruits[2]); // "orange"
```

Arrays can also be looped over using the `` `for` `` loop or the `` `forEach()` `` method.

```
for (let i = 0; i < fruits.length; i++) {
  console.log(fruits[i]);
}

fruits.forEach(function(fruit) {
  console.log(fruit);
});
```

Arrays can be modified using methods such as `` `push()` ``, `` `pop()` ``, `` `shift()` ``, `` `unshift()` ``, `` `splice()` ``, and `` `slice()` ``.

```
  fruits.push("grape"); // adds "grape" to the end of the
array
```

```
  fruits.pop(); // removes the last element from the
array
  fruits.shift(); // removes the first element from the
array
  fruits.unshift("kiwi"); // adds "kiwi" to the beginning
of the array
  fruits.splice(1, 1); // removes one element from the
array at index 1
  let citrus = fruits.slice(1, 3); // creates a new array
with elements at index 1 and 2
```

OBJECTS

Objects are used to store a collection of key-value pairs in JavaScript. Objects can be declared using curly braces `{}` and values can be accessed using their key.

```
  let person = {
    name: "Pugazh",
    age: 30,
    city: "Chennai"
  };

  console.log(person.name); // "Pugazh"
  console.log(person.age); // 30
  console.log(person.city); // "Chennai"
```

Objects can be modified using dot notation or bracket notation.

```
  person.age = 40; // changes the value of the "age"
property
  person["city"] = "Los Angeles"; // changes the value of
the "city" property
```

```
  person.job = "teacher"; // adds a new property to the object
  delete person.age; // deletes the "age" property from the object
```

Objects can also contain methods, which are functions that are stored as object properties.

```
  let person = {
    name: "Pugazh",
    age: 30,
    city: "Chennai",
    sayHello: function() {
      console.log(`Hello, my name is ${this.name}.`);
    }
  };

  person.sayHello(); // "Hello, my name is Pugazh."
```

DOM MANIPULATION

JavaScript can be used to create, modify, and delete HTML elements in the Document Object Model (DOM).

```
let newElement = document.createElement("div"); // creates a new div element
newElement.textContent = "New Element"; // sets the text content of the new element
document.body.appendChild(newElement); // adds the new element to the end of the body
```

In the above example, a new `***div***` element is created, its text content is set, and it is added to the end of the `***body***` element.

Elements in the DOM can also be selected using methods such as `querySelector()` and `getElementById()`.

```
let element = document.querySelector(".my-class"); // selects the first element with class "my-class"
let element = document.getElementById("my-id"); // selects the element with id "my-id"
```

Once an element is selected, its attributes and properties can be modified using dot notation or bracket notation.

```
element.style.color = "blue"; // changes the color of the element
element.setAttribute("data-name", "Pugazh"); // sets the value of the "data-name" attribute to "Pugazh"
```

EVENTS

Events are actions or occurrences that happen in the browser, such as clicking a button or scrolling the page. JavaScript can be used to listen for and respond to these events.

```
let button = document.querySelector("#my-button"); // selects the button element with id "my-button"
button.addEventListener("click", () => { // adds a click event listener to the button
  console.log("Button clicked");
});
```

In the above example, an event listener is added to a button element that logs a message to the console when the button is clicked.

Events can also be removed using the `removeEventListener()` method.

```
let button = document.querySelector("#my-button");
let handleClick = () => {
  console.log("Button clicked");
};
button.addEventListener("click", handleClick); // adds a
click event listener
button.removeEventListener("click", handleClick); //
removes the click event listener
```

ASYNC PROGRAMMING

Asynchronous programming is a way to execute code that takes a long time to run without blocking the main thread of execution. In JavaScript, this is typically done using callbacks, promises, or async/await.

```
// Callbacks
function fetchData(callback) {
    fetch("https://api.example.com/data")
      .then(response => response.json())
      .then(data => callback(data))
      .catch(error => console.error(error));
  }

  fetchData(data => {
    console.log(data);
  });

  // Promises
  function fetchData() {
    return fetch("https://api.example.com/data")
      .then(response => response.json())
      .catch(error => console.error(error));
  }
  fetchData()
    .then(data => console.log(data))
    .catch(error => console.error(error));

  // Async/await
  async function fetchData() {
    try {
      const response = await
fetch("https://api.example.com/data");
      const data = await response.json();
      return data;
    } catch (error) {
      console.error(error);
    }
  }

  fetchData()
    .then(data => console.log(data))
    .catch(error => console.error(error));
```

In the above examples, `` `fetchData()` `` is a function that fetches data from an API using `` `fetch()` ``. The callbacks, promises, and async/await methods all achieve the same result, but in different ways.

CONCLUSION

The fundamental ideas of JavaScript are covered in this cheat sheet, including variables, data types, operators, control structures, functions, arrays, objects, events, DOM manipulation, and async programming. By learning these ideas, you'll be well on your way to becoming a skilled JavaScript programmer.

PRACTICES

REFERENCES

JavaScript101 - GitHub - reach2arunprakash/javascript-101: Code repository for jQuery Fundamentals training

Medium Blog
- GUVI: Zen Class — Variables Arrays & Objects
- GUVI: Zen Class — Part 1: Find the culprits and nail them — debugging javascript
- https://medium.com/@reach2arunprakash/www-guvi-io-zen-4fa483a7d359
- GUVI: Zen Class — Part 3: Find the culprits and nail them — debugging javascript