Intro to Javascript

Curly Braces

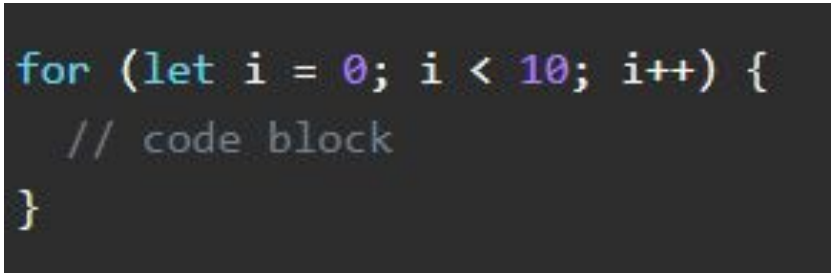
- Curly braces are used to delineate code blocks such as in function definitions, loops, and if blocks.
- Curly braces can also be used to define JavaScript objects.

```
// `if/else` block
if (boolean) {
  // code block...
} else {
  // another code block...
}

// loop
while (condition) {
  // code block...
}

// function definition
function foo() {
  // code block...
}

// JS object
let obj = { key: "value" };
```

LOOPS

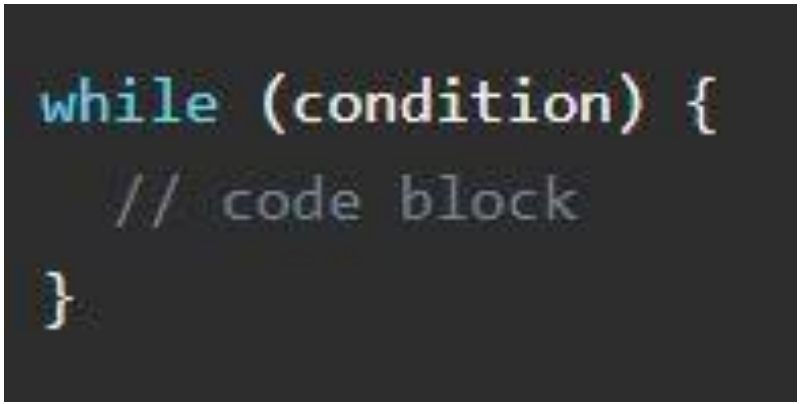
for loops

- There are 3 statements in a for loop:
  - First statement is executed once before executing the code block.
  - Second statement is the condition to continue loop.
  - Third statement is executed every time after the code block.



while loops

- while loops continue to run the code block as long as the condition is true
- To avoid infinite loops, ensure the variable within the condition can change and eventually return false.



Keyword: continue

- This keyword skips the current iteration of a loop.

```
let result = [];
for (let i = 1; i < 10; i++) {
  if (i % 3 === 0) {
    continue;
  }
  result.push(i);
}

console.log(result); // [1, 2, 4, 5, 7, 8]
```

Keyword: break

- This keyword exits the loop.

```
let result = [];
for (let i = 1; i < 10; i++) {
  if (i % 3 === 0) {
    break;
  }
  result.push(i);
}

console.log(result); // [1, 2]
```

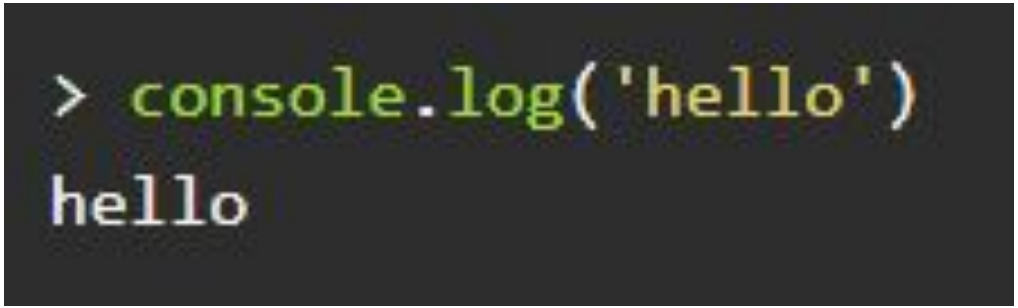
switch statements

- Switch statements are ran once.
- The value of the expression is compared to each case.
  - If there is a match, the block of code is executed.

```
switch (expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
  // code block
}
```

console.log()

- This prints something out onto the console.
- Useful for troubleshooting.



USEFUL METHODS

Mathematical Operators

- **+ :** Add
- **- :** Subtract
- *** :** Multiply
- **/ :** Divide
- **% :** Modulo (remainder from division)

Commenting

```
// this is a javascript in-line comment

/*
  this is
  a javascript
  comment block
*/
```

Comparison Operators

- **>** : greater than
- **<** : less than
- **>=** : greater or equal to
- **<=** : less than or equal to
- **===** : equal to
- **!==** : not equal to

- Note there this is a “==” double equals operator. This operator does some type conversion that may lead to confusing results. It’s usually best to stick with “===”

Logical Operators

- **&&** : and
- **||** : or
- **!** : not

String Methods

- **String.prototype.toLowerCase**
- **String.prototype.toUpperCase**
- **String.prototype.indexOf**
- **+ :** concatenation
- https://www.w3schools.com/js/js_array_methods.asp

Array Methods

- **Array.prototype.length() :** returns length of an array
- **Array.prototype.pop() :** removes last element and returns that element
- **Array.prototype.push() :** adds element(s) to the end of array
- **Array.prototype.unshift() :** adds elements(s) to the beginning of an array
- **Array.prototype.shift() :** removes first element and returns that element
- **Array.prototype.indexOf() :** returns first index where a given element is found
- **Array.prototype.slice([start, [end]]) :**
  - Makes a copy of an array from the start index up to but not including the end index. Both arguments are optional (the first and last elements are used by default).
- **Array.includes() (ES6+) :** returns boolean for whether an element is in an array

Data Types

Data Types

- In JavaScript, there are 6 data types
  - number
  - string (text)
  - boolean (true/ false)
  - undefined
  - null
  - object

JavaScript Objects

- Objects are VERY important
- They store properties, which can also include functions

```
var cat = {
  name: "Breakfast",
  age: 8,
  purr: function () {
    console.log("meow!");
  }
};
```

```
// using Bracket-Notation
console.log(cat['name']); // => Breakfast
// using Dot-Notation
console.log(cat.age); // => 8

// calling a method
cat.purr(); // => 'meow!'

// reassigning properties
cat.name = "Earl";
cat['age'] += 1;
```

Variables

- **var :** declaring with this makes a **functionally-scoped** variable
- **let (ES6+) :** declaring with this makes a **block-scoped variable**
- **const (ES6+) :** declaring with this creates an **immutable constant**

- Leaving off a declaration creates a global variable. **NEVER DO THIS**
- It is almost always preferred to use **let** and **const** over **var**.

Functions

Declaring Functions

Function-style:

```
function functionName(arg1, arg2, arg3, argN) {
  // code block...
}
```

Expression-style:

```
const functionName = function(arg1, arg2, arg3, argN) {
  // code block...
};
```

Fat Arrow-style (ES6+):

```
const functionName = (arg1, arg2, arg3, argN) => {
  // code block...
};
```

Invoking functions with ()

```
// function with 0 arguments
function retHello() {
  return "hello";
}

retHello; //=> [Function: retHello]
retHello(); //=> "hello"
```

```
// function with 2 arguments
function sum(n1, n2) {
  return n1 + n2;
}

sum; //=> [Function: sum]
sum(10, 20); //=> 30
```

Assigning properties to functions

- Functions that are passed as an argument to another function are called **callbacks**.

```
function logIfEven(num) {
  if (num % 2 === 0) {
    console.log(`${num} is an even number!`);
  }
}

[1, 2, 3].forEach(logIfEven);
```