## LOGIC

- An if  statement has the following structure. The code will only run if the condition in the parenthesis is true.

```
if (condition) {
// code to run if condition is true
} else {
 // code to run if the condition is false.
 }
```

- The else keyword can be used to add an alternative block of code to run if the condition is false

shorthand:

```
condition ? true-code : false-code
```

- The switch operator can be used to make your code easier to follow when there are lots of conditions to test:

```
switch (condition) {
        case opt1:
         break
        case opt2:
          break
        case opt3:
          break
        default:
         break
       }
```

## Loops

- A while loop will repeatedly run a block of code while a certain condition is true.
- >A do ... while loop is similar to a while loop. The only difference is that the condition comes after the block of code.

```
for (initialization ; condition ; after) { do something}
```

- You can place a loop inside another loop to create *a nested loop*

## Functions

- Defining a function:

```
function hello () {
    console.log ('Hello World!');
    }
```

- Defining a function literal using a function expression (*This assigns an anonymous function to a variable*):

```
const goodbye = function (){
                console.log ('Goodbye World!');
            };
```

- All functions have a read-only property called name:

  **hello.name**

- A function can also be declared using the constructor: `Function ()` .
- The body of the function is entered as a string:

```
const hi = new Function ('console.log("Hi World!");');
```

- It is not recommended to declare functions with the function constructor, there are too many issues when a function is put into a string. Functions created this way are also created in the global scope, regardless of where they are actually declared. This can lead to some strange and unexpected behavior

### *Invoking Function and Return Values*

- Invoking a function tells the computer to run the code inside the functions body. It is easy to invoke a function, type its name followed by parentheses.

  ***hello();***

- Not including the parentheses after the name of the function/variable, will just be a reference to the function/variable, not allowing it to be invoked/executed.
- All functions will return something. If the return statement is omitted, then the function returns *undefined*.
- All other return statements will return the value specified

## Parameters and Arguments

- Parameters and arguments are terms often used interchangeably to represent values provided for the function as an input.
- There is a subtle difference though: any parameters a function needs are set when the function is ***defined***.
- When a function is ***invoked,*** it is provided with arguments.
- To specify a default parameter, simply assign the default value to it in the function definition.
- Arrow functions are always anonymous, so if you want to refer to them, you must assign them to a variable.
- Variable declarations that use the var keyword are automatically moved to the top of the current scope.

## **Callbacks**.

- Since functions are first class objects. This allows us to treat them like any other object and pass them into other functions.
- This is commonly referred to a callback. example from book:

```
function sing(song) {
        console.log(`I'm singing along to ${song}`);
    }
    sing('ABCDEFG')
```

- Adding the callback parameter:

```
function sing(song,callback) {
  console.log(`I'm singing along to ${song}.`);
   callback();
    }
```

- Creating the callback function</li>

```
function dance()  {
   console.log("I'm moving my body to the groove.");
 }
```

- Invoking our function with the new callback
  *sing('Gitchee Gitchee Goo',dance);*

*'I'm singing along to Gitchee Gitchee Goo.'*

*'I'm moving my body to the groove.'*

- Callbacks are passed into the function as a parameter without the parentheses.
- The callback will be executed in the function with the parentheses, generally when some tasks has been executed.