Week 1: JavaScript Basics

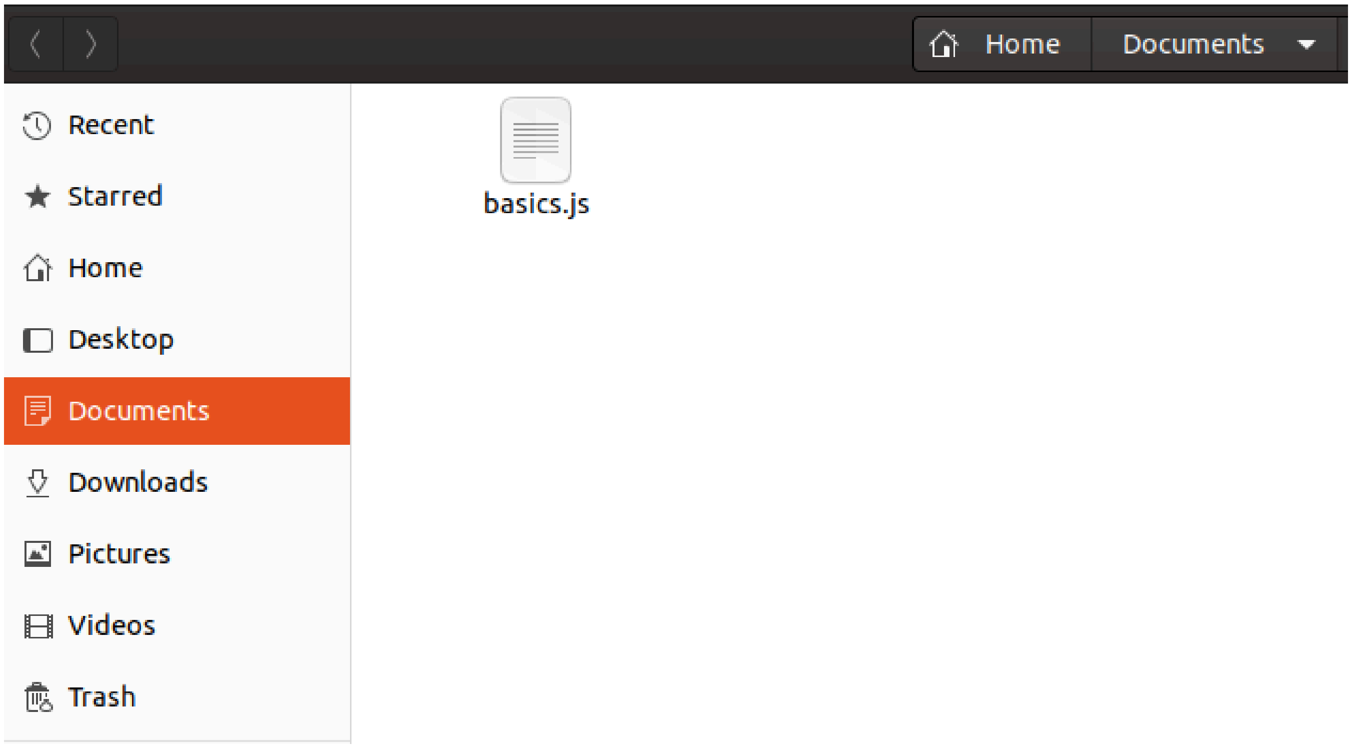
Week 1 is meant to be a lesson on the programming language: JavaScript

By this point, you should have Node.js and Visual Code installed.
It would be easier to follow this lesson plan along with Visual Code.

1: Creating and getting the development environment ready

a. Please navigate to a folder where you would like to create a new file called `basics.js`

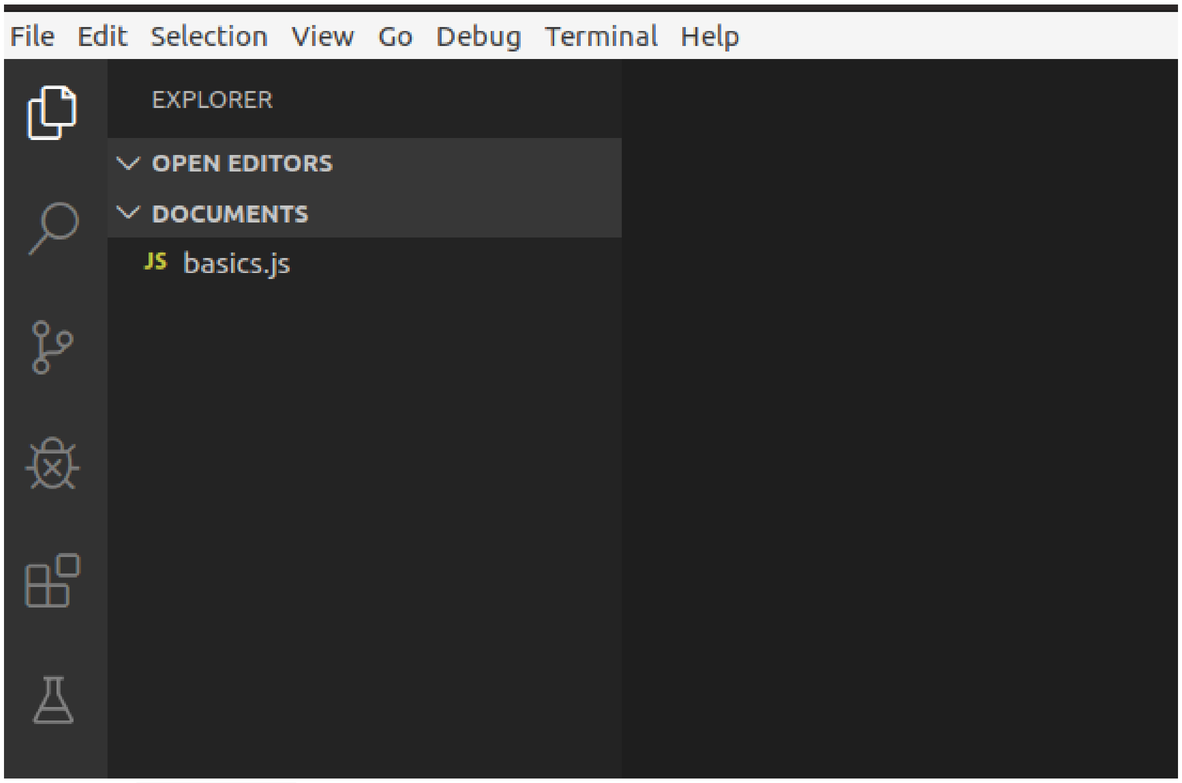
- I will save mine in the `/Documents/` folder, like so:



b. Now that the file is created, let's open it in Visual Code.

1. Open Visual Code
2. Then at the top left corner go to: File -> Open Folder
3. From there, navigate to the folder where you saved `basics.js` file. (In my case `/Documents/`)

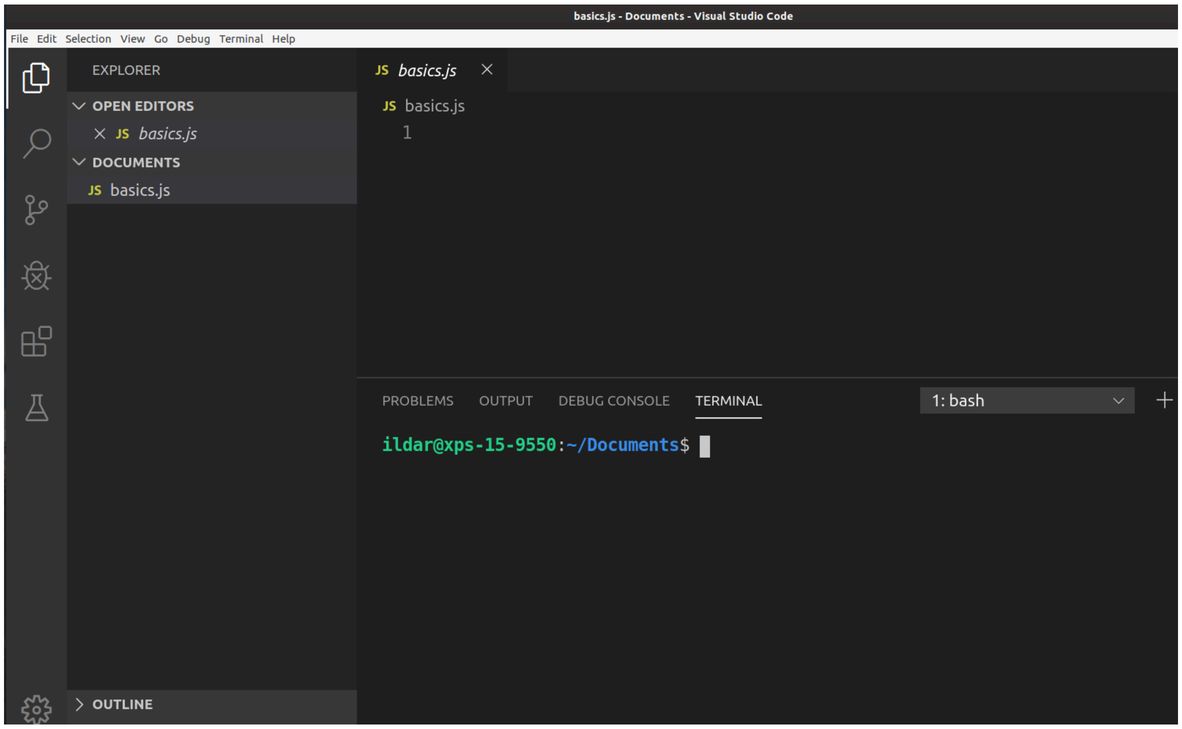
4. Press OK. And then, you should see something similar to this:



5. press on `basics.js` to open the file in the editor. (It should be empty)

c. Now, the file is open, the directory is ready, so the SSonly thing that is left to do is to open the terminal.

- To do that you can press the two keys at the same time `Ctrl ~` if you are on Windows or Linux, or `Control ~` if you are on a Mac. (Or at the top bar of Visual Code there is View -> Terminal)
- Visual Code should look similar to this:


basics.js - Documents - Visual Studio Code
File Edit Selection View Go Debug Terminal Help
EXPLORER
OPEN EDITORS
basics.js
DOCUMENTS
basics.js
OUTLINE
basics.js
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: bash

```
ildar@xps-15-9550:~/Documents$
```

(with the exception that you might not have `1: bash`, but you have something else instead written there)

Notes:

- the left part of the Visual Code screen contains the files in the current directory that you have active.
- the top part of the right side, is where you are able to type in the code, and it currently says that `basics.js` is open.
- the bottom part where it says `/Documents$` on my screen: that is the **terminal**. That is where we will be executing commands to run the code that we write in this project.

Now our environment is setup for Week 1, and we are ready to learn JavaScript.

2: Hello World

In the programming world, it is a big tradition to write a program that will print (output) to the user `hello world` when you are learning a new language.

So let's start with that 😃.

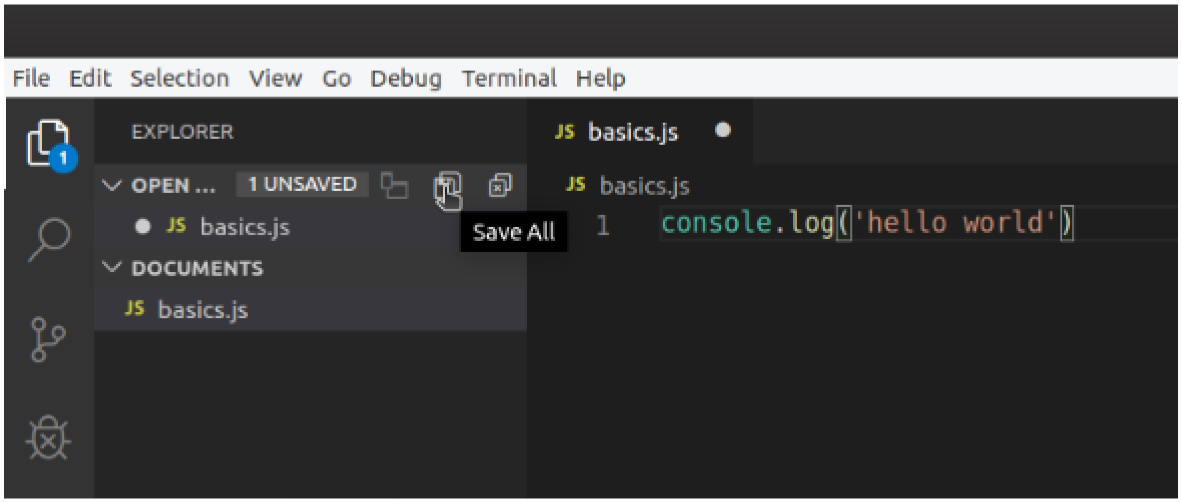
1. Please type in the following code in `basics.js`:

`console.log('hello world')`

And Save the file, since the white circle indicated that the changes to the file are unsaved

(You can do it in at least two ways)

First Way: press the Save All button like in the screenshot



```
console.log('hello world')
```

Second Way: press `Ctrl S` at the same time if you are on Windows or Linux, or `Command S` if you are on a Mac.

Just please make sure the white circle won't be there

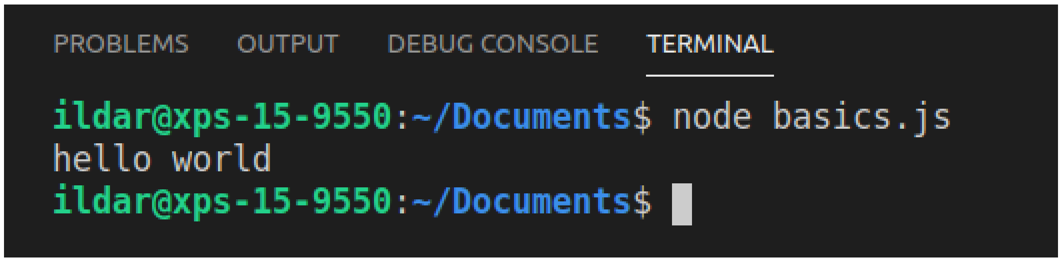
2. Then in the **terminal**, type in:

   `node basics.js`

   and press ENTER while in the terminal.

   This will use Node.js to run the code that you have written in `basics.js`

Now, we would expect the output in the **terminal** to be `hello world`, like the screenshot below



```
ildar@xps-15-9550:~/Documents$ node basics.js
hello world
ildar@xps-15-9550:~/Documents$ 
```

Now we are ready to learn more about the language, and some programming knowledge to prepare us more for the rest of the events

3: Constant values and printing stuff with `console.log` and adding comments

First of all, what is a constant value in programming?

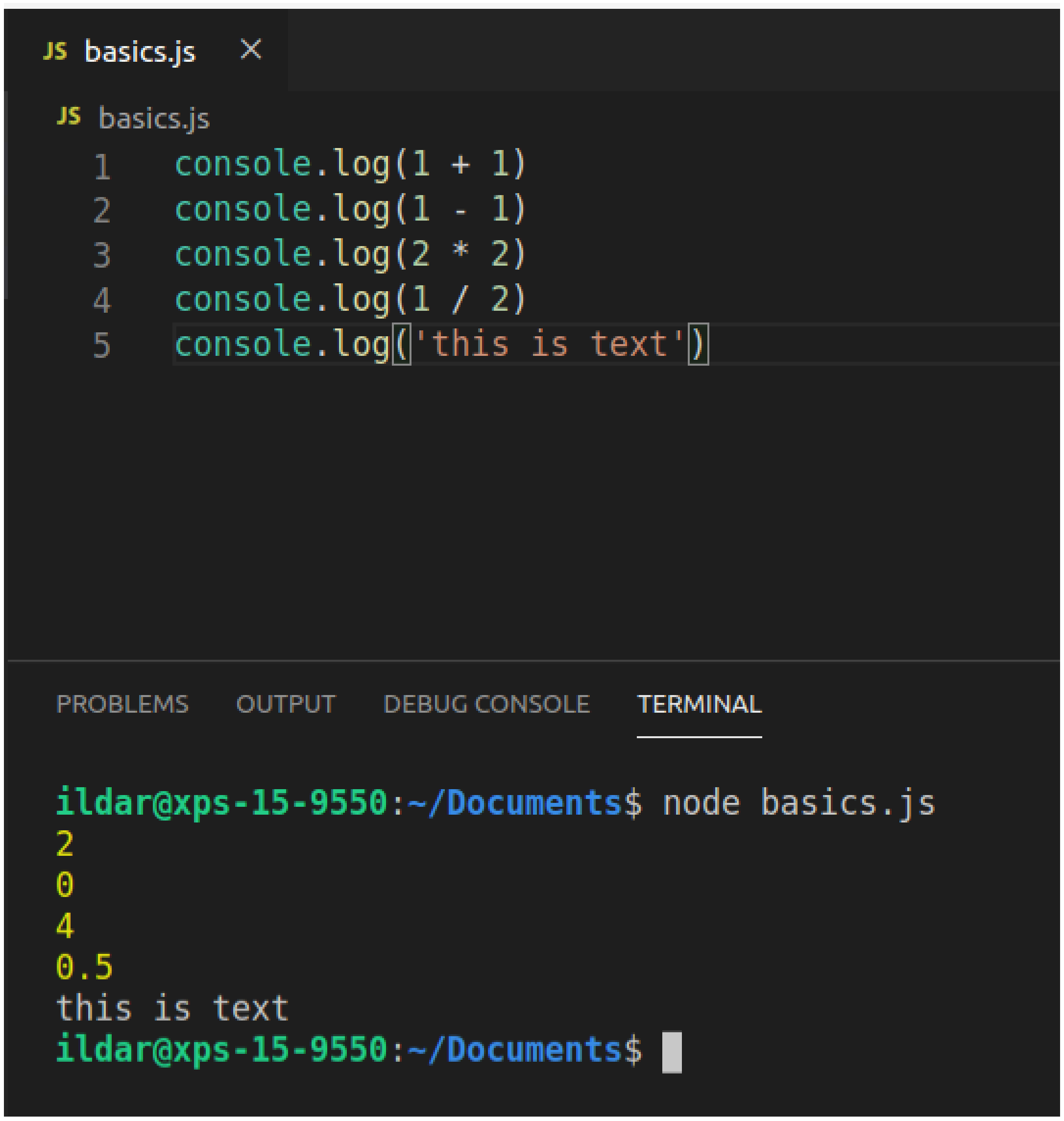
- It is the value that cannot be changed.
  For example, 1+1 will always be 2, 3 will always be 3, and the text 'hello world' will always be the text 'hello world', etc.

Next, in the previous part we saw that we can print things in JavaScript, when we printed 'hello world`. With the same idea, we can print almost everything in JavaScript: simply by using `console.log` and put the value that we want to be printed in the parentheses ()

For example, consider the following code snippet: (you can try running it too in `basics.js` file you created by **saving** the file and typing `node basics.js` and then ENTER in to the **terminal**)

```
console.log(1 + 1)
console.log(1 - 1)
console.log(2 * 2)
console.log(1 / 2)
console.log('this is text')
```

which would look like this in Visual Code:



```javascript
console.log(1 + 1)
console.log(1 - 1)
console.log(2 * 2)
console.log(1 / 2)
console.log('this is text')
```

```
ildar@xps-15-9550:~/Documents$ node basics.js
2
0
4
0.5
this is text
ildar@xps-15-9550:~/Documents$
```

How this works (in more detail):

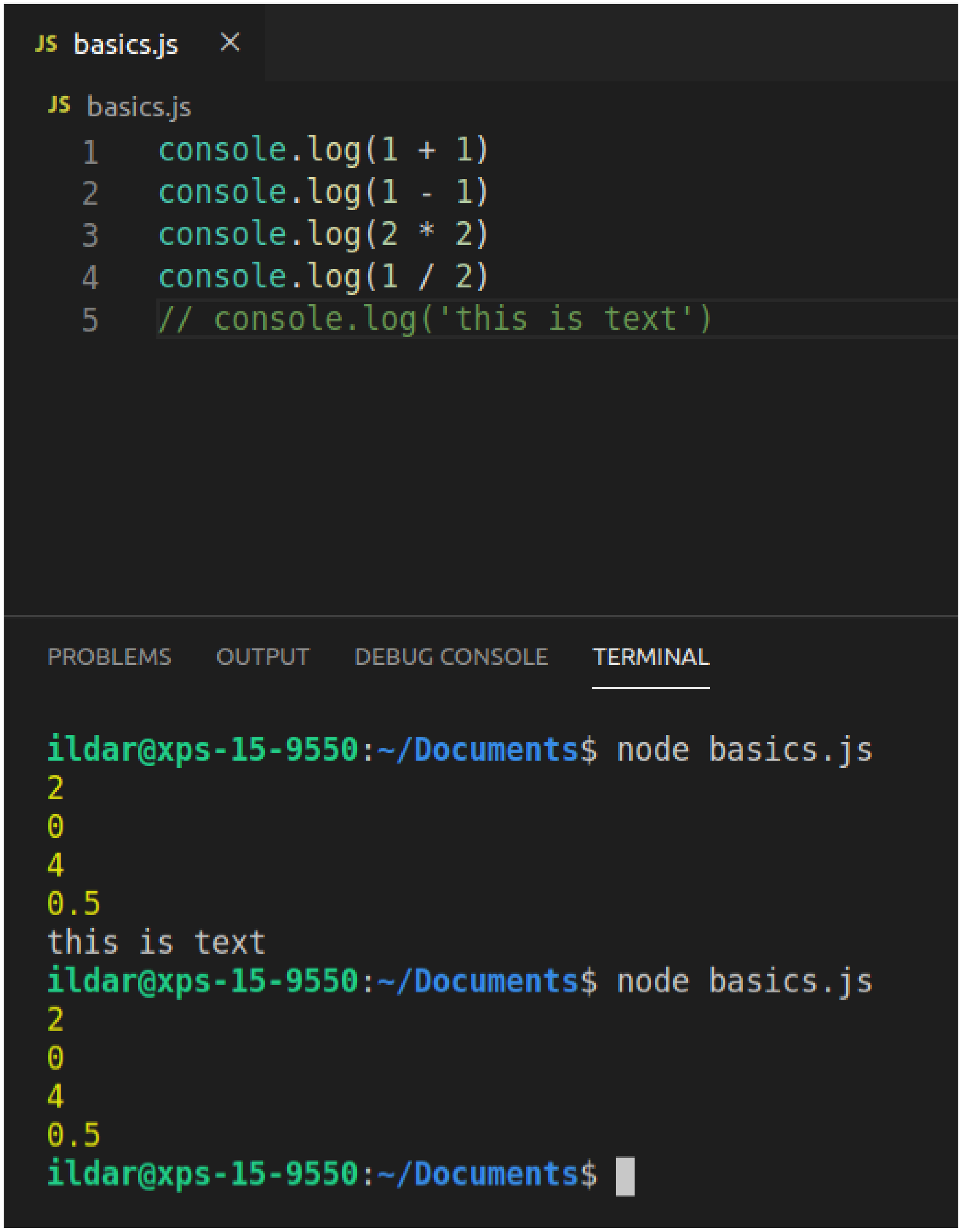
1. when you run `node basics.js`, Node tries to read the file (basics.js), and understand what you have written in JavaScript
2. then, it executes the code line by line: in our case it is printing first: **2**, then **0**, then **4**, then **0.5**, and then **this is text**.
   this execution of lines of code means that this is done in a **synchronous** manner.

Now, what if you want to write something that you don't want to be "seen" by Node.

This brings us to what is called "commenting" in the code
This allows you to make the code "invisible to Node" but still very visible to the developer

To do that, you just put `//` on the left of (before) the code you want to make "invisible".

For example, say I want to comment out the line `console.log('this is text')`
It would look like this:



```
console.log(1 + 1)
console.log(1 - 1)
console.log(2 * 2)
console.log(1 / 2)
// console.log('this is text')
```

```
ildar@xps-15-9550:~/Documents$ node basics.js
2
0
4
0.5
this is text
ildar@xps-15-9550:~/Documents$ node basics.js
2
0
4
0.5
ildar@xps-15-9550:~/Documents$
```

Note that 'this is text' is not printed the second time I run `node basics.js` since the line responsible for that is commented out.

4: Variables and Data Types

Now, it is quite difficult to only use constant values in programming as the tasks become more and more difficult, since sometimes you want to store a value somewhere, to be able to retrieve it later. Or, say you want to update the value to something else, and be able to retrieve that for later use.

This is where **variables** come in, and why they are so important.

Variables
In JavaScript the original way to define a variable was:
`var name = 1234`
Which means it creates a new variable (what `var` means), assigns the constant value of `1234` to the variable named `name`

and now you can use it whenever, for example:

```
console.log(name) // this will print 1234
```

But, if you can define a variable, and you can access it, then you should be able to change the value of the variable. You can do it like so:

That is called an assignment.

```
name = 12345
```

However, in more recent years (2015 and newer), the new way to define variables became more prominent, as the original way had issues such as the ability to do the following:

```
var a = "1234"
var a = "12345"
```

the code snippet above means create a new variable named `a` with the value "1234", and then on the next line: create a new variable named `a` with the value "12345".

So now, there are two variables with the name `a` at different points of our code. This could be very problematic, and hard to debug, since the way the language is made, this would make the first variable get overriden by the new one. So, to circumvent this problem there were introduced two keywords that replace `var` which are `let` and `const`.

`let` **vs** `const`

Ultimately, `let` and `const` are similar, at least in the way that they prevent the problem that was mentioned earlier.

this will not work any more:

```
let a = 1234
let a = 12345

// or

const a = 1234
const a = 12345
```

However, the caveat is that if you use `let`, value assignment is possible:

```
let a = 'some text'
a = 'some new text'
```

But, with `const` it is not possible to reassign the value of the variable. For example, **this will not work**:

```
const a = 'some text'
a = 'some other text'
```

Aside:
These statements are equivalent:

```
i = i + 1
i++
```

Data Types
In JavaScript, there are the following data types:

`String`: a fancy word for text (sequence of characters)
You can define a String in JavaScript using the following ways:

```
    let a = 'String'
    let b = "String"
    let c = `String`
```

`Number`: any type of number (with decimals, or without)
You can define a Number in JavaScript using the following way:

```
    let a = 1
    let b = 2 / 3 // b will have a value of something close to 0.6666666666
```

`Boolean`: true or false
You can define a Boolean in JavaScript using the following way:

```
    let a = true
    let b = false
    let c = 2 == 5 // == is used to check the equality
    let d = !true // ! is used to negate the boolean after it
    let e = 2 < 5 // used for inequality checking
    let f = 2 == 5 || 2 < 5 // || is used as an 'or' in logic
    let g = 2 == 5 && 2 < 5 // && is used as an 'and' in logic
```

`Undefined`: undefined type, usually happens when the variable has no value assigned to it

`Arrays`: an array, discussed in part 6 of this week

`Function`: a function, discussed in part 8 of this week

5: Conditionals (if-statements)

Conditionals are very important when we have to deal with decisions in our code.

To do that, we can do it by using an `if-statement` in JavaScript.

```
if (some condition) {
    // execute this code
} else if (some other condition) {
    // execute this code
} else {
    // all conditions before this failed
    // execute this code
}
```

`condition`: must be a boolean-like expression.

```
const a = 5
if (a % 2 == 0) {
    console.log('a is even')
} else {
    console.log('a is odd')
}
```

Aside

- you can have as many `else if` statements as you potentially need. Also, an `else` statement is not necessary.
- `%` operator is called 'modulo operator' that is equivalenet to getting the remainder of the division, rather than the quotient.

6: Arrays (lists of values)

Consider the following code snippet:

```
let a = 1
let b = 2
let c = 3
// ...
```

This will be potentially a lot of variables, so a better way to store that would be some sort of list, since those variables are related.

To create an array in JavaScript:

First approach:

```
let arr = [1, 2, 3, ...] // creates a variable named arr of type array with values 1, 2, 3,...
```

Second approach:

```
let arr = [] // creates a variable named arr of type array
arr.push(1, 2, 3, ...) // adds values 1, 2, 3, ... to arr
```

Then to access a value in the list, in JavaScript (and many other programming languages you don't get the first element, but the zero-th element first), you can get the `i-th` element by doing so:

```
console.log(arr[0]) // this will print 1
console.log(arr[1]) // this will print 2
```

Aside:

- you can add any data type to an array, even another array. And, you can even add the array to itself as an element.
- position of an element in an array is also called its index

7: Loops

Sometimes, the code can be very repetitive, for example you want to print each element in the array individually.
For example:

```
const arr = [1, 2, 3, 4]

console.log(arr[0])
console.log(arr[1])
console.log(arr[2])
console.log(arr[3])
```

to make it less repetitive, as the only thing that changes is the index, we can use a `for-loop`

```
for (statement 1; condition; statement 3) {
    // code block to be executed
}
```

`statement 1` : a bit of code that will run before the loop begins
`condition` : the for loop will run while `condition` is true

`statement 3` : a bit of code that will get executed after each iteration in the loop

```
// equivalent to the non-loop code from above
const arr = [1, 2, 3, 4]
for (let i = 0; i < 4; i = i + 1) {
    console.log(arr[i])
}
```

8: Functions

It's often helpful to organize code in snippets, and make it reusable.

Key idea: have functions do a single task.

Here is how you would define a function in JavaScript:

```
function f (variables) {
    // code to be run when a function is called
}
```

`f` is the function name
`variables` is a list of variables that the function can use when called. Referred to as: parameters of a function

and to call a function `f` with the value `1` for `variables` parameter(make it run)

```
f(1)
```

returning values

lastly, functions can be written to return values.
consider the following example:

```
function addFive (x) {
    return x + 5
}

const ten = addFive(5) // should assign a value of 10
```

functions are also data types, so we can define them as variables

another way to define a function, like more so a variable is to do it in this manner:

```
const addFive = function (x) {
    return x + 5
}
```

motivation: this allows us to pass the function as a parameter, or use it as a variable in another way.

Example:

```
const addFive = function (x) {
    return x + 5
}
const test = function (func) {
    return func(10)
}

console.log(test(addFive))
```

This will print out `15`