Day 1 - Basic Terminal Usage, Variables and Data Types, Conditions

Outline

In this session, you are going to learn about these topics:

- Using a terminal
- Types of variables and data
- Strings, numbers, and booleans
- Basic methods and properties
- Conditions
- Falsy and truthy values

Variables

Variables are like containers in the programming world. They can hold values, and you can give them labels or names. Think of it as a lunch box. Lunch boxes can

contain food — although it can also contain other things — and you need to give them proper names or labels so you won't lose them or mistaken them for something else.

There are 3 types of variables in Javascript:

- `const`: to create scoped, constant, and immutable variables
- `let`: to create scoped, mutable variables
- `var`: to create global, mutable variables

Data Types

In the programming world, we have various data types. Each data type will represent a different kind of data such as mathematical numbers, text, and collections.

Here are several basic data types you need to know:

- `string`: represents text
- `number`: represents mathematical numbers
  - `integer`: represents whole numbers
  - `float` / `double` : represents floating or decimal numbers
- `boolean`: represents true or false values

Methods and Properties

Properties

Every data type in Javascript has their own methods and properties. Properties are keys stored in an `object` and can hold values the same way variables do.

> 💡 There are some terminologies that you may not know such as *keys* and *objects*. Don't worry, we'll learn more about them along the way. For now, just follow along.

Let's start with some basics properties. Strings have a property called `length` which returns the string's length and you can access them by using a dot (`.`).

```
const myName = "alex"
myName.length  // returns 4

const yourName = "benjamin"
yourName.length  // returns 8
```

Methods

Methods are a bit different. Methods can also be accessed the same way properties do, however methods contain a different type of value. Properties hold data types, methods contain ***functions***.

> 💡 Again, don't worry about ***functions*** yet. We'll learn about them more later.

Basically functions are a set of instructions that are stored in one place (variable). You can reuse these functions so you won't have to repeat your code over and over again.

Methods store these functions. Which means we have a set of instructions we can use, stored in our data types.

For example, we have a method called `toUpperCase` in `strings`. When called, this method will turn our string into uppercase letters.

```
const doctor = "john"
doctor.toUpperCase()  // returns "JOHN"

const inspector = "lestrade"
inspector.toUpperCase()  // returns "LESTRADE"

"gwen".toUpperCase() // returns "GWEN"
```

Conditions

When creating a program, we have to be able to control the flow of our program, and for that, we have conditions. Conditions help us determine which commands should we execute when given a certain parameter.

If

`if` conditions are used to create a condition for a block of code to execute when given the correct parameters. To know if `if` conditions should execute their part of the program, they need booleans with `true` values.

Here's a basic implementation of `if` conditions.

```
if (true) {
  console.log("hi there!")
}

// output: "hi there!"
```

```
if (false) {
  console.log("hello!")
}

// output:
```

Else

`else` specifies a block of code to execute when `if` is not executed. It is important to note that `else` should **always** be chained with either an `if` or an `else if` (more on `else if` below).

```
if (false) {
  console.log("hi there!")
} else {
  console.log("hello!")
}

// output: "hello!"
```

One more thing to notes is that `else` do not need a condition or boolean. The only thing `else` needs for it to execute is a false condition on its preceding `if` condition.

Else If

`else if` works the same as `if` and `else` combined. `else if` acts similarly to `else` which will execute a block of code if the preceding `if` condition or `else if` condition is false, but what makes it different is that `else if` require a condition to be met.

```
if (false) {
  console.log("hi there!")
} else if (true) {
  console.log("hey!")
} else {
  console.log("hello!")
}

// output: "hey!"
```

```
if (false) {
  console.log("hi there!")
} else if (false) {
  console.log("hey!")
} else {
  console.log("hello!")
}

// output: "hello!"
```

You can also chain `else if` conditions with more `else if` conditions!

```
if (false) {
  console.log("hi there!")
} else if (false) {
  console.log("hey!")
} else if (true) {
  console.log("greetings!")
} else {
  console.log("hello!")
}

// output: "greetings!"
```

Falsy? Truthy?

Falsy and truthy are terms used in programming to determine values within a boolean context. For example, in a boolean context `1` is considered `true` which means `1` is a truthy value. `0` in a boolean context is considered `false` which means it is a falsy value.

At first glance, this seem quite simple but Javascript can sometimes become confusing. Here is a list of some falsy and truthy values that can sometimes be

confusing.

Falsy

- `""` (empty string)
- `0`
- `null`
- `undefined`
- `NaN`

Truthy

- `" "` (blank character string)
- `[]` (empty array)
- `{}` (empty object)
- `1`
- `"1"`
- `"0"`
- `"false"` (string)
- `"true"` (string)

If you're not sure whether a value is truthy or falsy, you can use an if statement and input your value as the condition. If your value is truthy, then surely the if statement should execute the code in the if block. Another way to check if a value is truthy or falsy is by converting it into a boolean like so:

```
console.log(Boolean("1"))     // true
console.log(Boolean(""))      // false
console.log(Boolean(NaN))     // false
console.log(Boolean("false")) // true
```