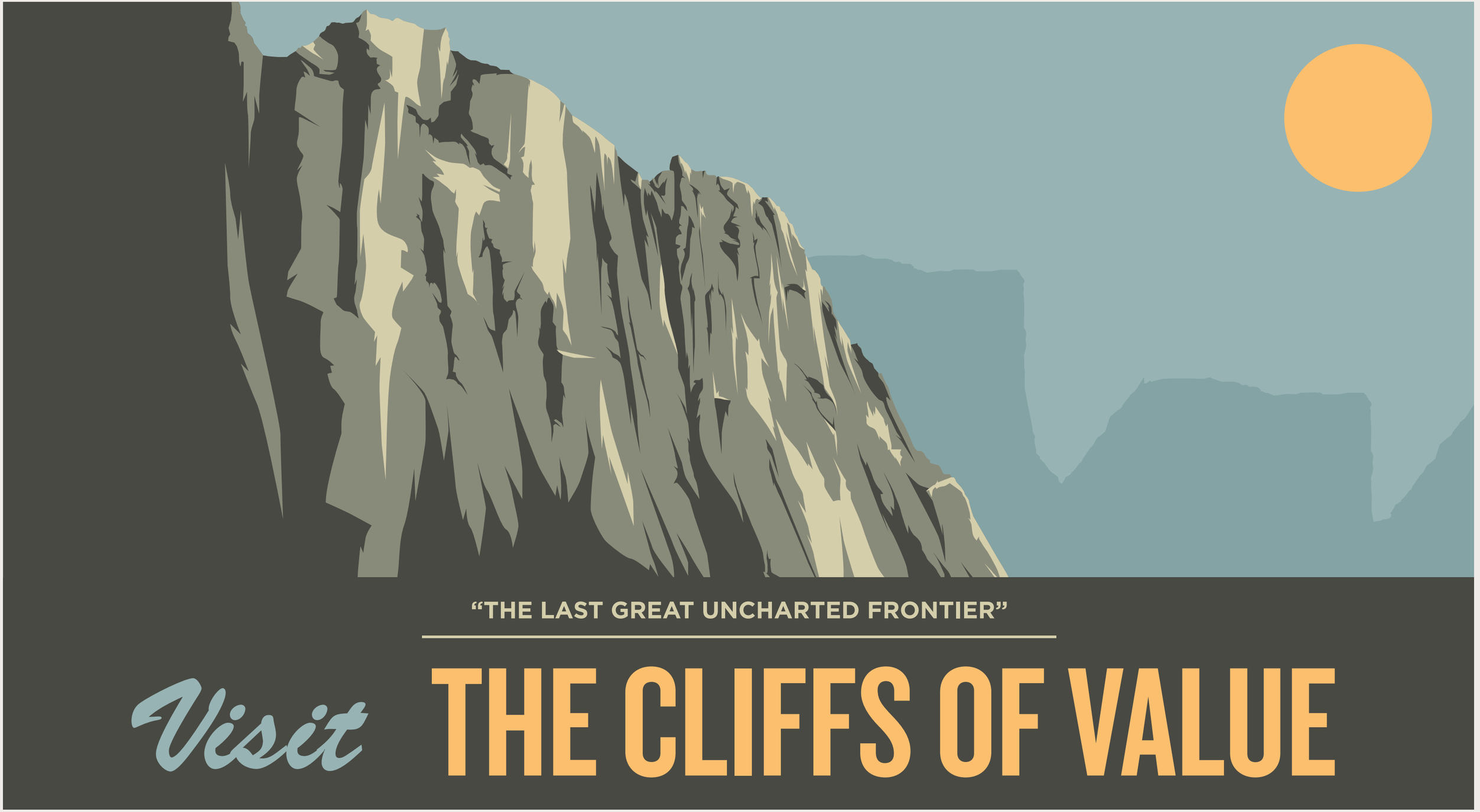

"THE LAST GREAT UNCHARTED FRONTIER"
Visit
THE CLIFFS OF VALUE

LEVEL 1

THE CLIFFS OF VALUE

THE PROMPT & SOME BASIC NUMBERS

`>` The JavaScript Prompt, aka “the Console”

→ What gets returned from the code

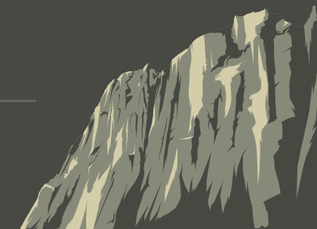
JavaScript automatically recognizes numbers

```
> 24
```

→ `24`

```
> 3.14
```

→ `3.14`



OPERATORS

Common Operators used in JavaScript Syntax:

addition

```
> 6 + 4
→ 10
```

subtraction

```
> 9 - 5
→ 4
```

multiplication

```
> 3 * 4
→ 12
```

division

```
> 12 / 4
→ 3
```

modulus

```
> 43 % 10
→ 3
```

Modulus returns the remainder after division.

ORDER OF OPERATIONS: PEMDAS

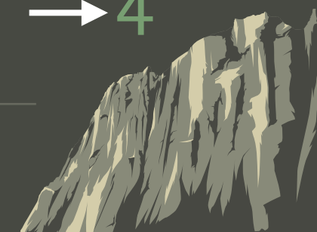
Grouping Expressions in JavaScript

```
> (5 + 7) * 3
    12    * 3
              → 36
```

```
> (3 * 4) + 3 - 12 / 2
    12    + 3 - 12 / 2
    12    + 3 -    6
                  → 9
```

```
> (-5 * 6) - 7 * -2
   -30     - 7 * -2
   -30     -   -14
                 → -16
```

```
> 4 + (8 % (3 + 1))
  4 + (8 %    4   )
  4 +       0
                  → 4
```



COMPARATORS

Common Number Comparators used in JavaScript Syntax:

greater than

```
> 6 > 4
```

→ true

"boolean" value

less than

```
> 9 < 5
```

→ false

greater or equal

```
> 8 >= -2
```

→ true

equals

```
> 3 == 4
```

→ false

two equal signs!

not equals

```
> 12 != 4
```

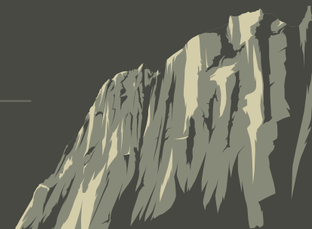
→ true

the "not" symbol

less or equal

```
> 10 <= 10
```

→ true



STRINGS

How JavaScript stores and processes flat text

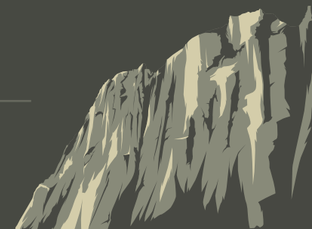
Strings need quotes!

```
> "Raindrops On Roses"
→ "Raindrops On Roses"
```

```
> "Whiskers On Kittens"
→ "Whiskers On Kittens"
```

Plus will glue Strings together

```
> "Raindrops On Roses" + " And " + "Whiskers On Kittens"
→ "Raindrops On Roses And Whiskers On Kittens"
```



THESE ARE A FEW OF MY FAVORITE...STRINGS

Concatenation works with numbers and their expressions, too.

```
>"The meaning of life is" + 42
```

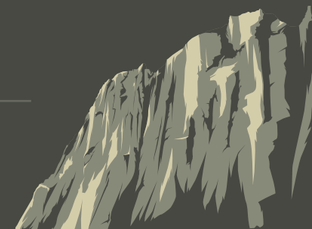
→"The meaning of life is42"

Uh oh...what happened? Concatenation adds no spaces, so we need to add our own.

Notice the extra space!

```
>"The meaning of life is " + 42
```

→"The meaning of life is 42"



THESE ARE A FEW OF MY FAVORITE...STRINGS

Concatenation works with numbers and their expressions, too.

```
> "Platform " + 9 + " and " + 3/4
```

→ `"Platform 9 and 0.75"`

Expressions get evaluated!

```
>"Platform " + 9 + " and 3/4"
```

→ `"Platform 9 and 3/4"`

Make strings out of expressions that you want to see in their original format.

SPECIAL CHARACTERS INSIDE STRINGS

Some characters need backslash notation in JavaScript Strings

advances to the next "tab stop"

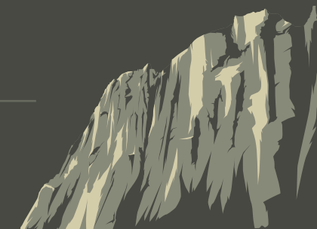
```
> "Flight #:\t921\t\tSeat:\t21C"
```

→ "Flight #: 921   Seat: 21C"

Adds a quotation mark but without ending the string too early.

```
> "Login Password:\t\t\"C3P0R2D2\""
```

→ "Login Password: "C3P0R2D2""



SPECIAL CHARACTERS INSIDE STRINGS

Some characters need backslash notation in JavaScript Strings

Places a backslash itself in the String

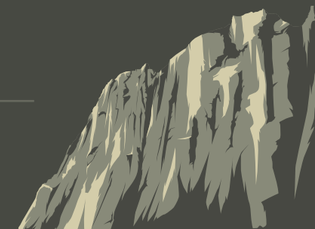
```
> "Origin\\Destination:\tOrlando(MCO)\\London(LHR)"
```

→ "Origin\Destination:	Orlando(MCO)\London(LHR)"

shifts the printout to a "new line"

```
> "Departure:\t09:55A\nArrival:\t14:55P"
```

→ "Departure:	09:55A
→ Arrival:	14:55P"



STRING COMPARISONS

Checking for matching strings and alphabetical ordering

"Double equals" will compare EXACT contents

```
> "The Wright Brothers" == "The Wright Brothers"
```

→ true

```
> "The Wright Brothers" == "Super Mario Brothers"
```

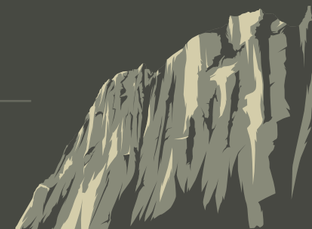
→ false

"Not equals" returns true if there is a mismatch

```
> "The Wright Brothers" != "the wright brothers"
```

→ true

Case counts!



STRING COMPARISONS

The length of strings can be accessed with the .length property

```
> "antidisestablishmentarianism".length
```

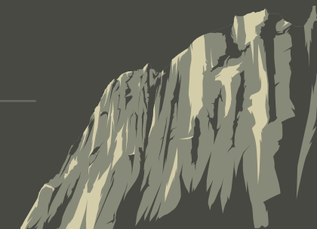
→ 28

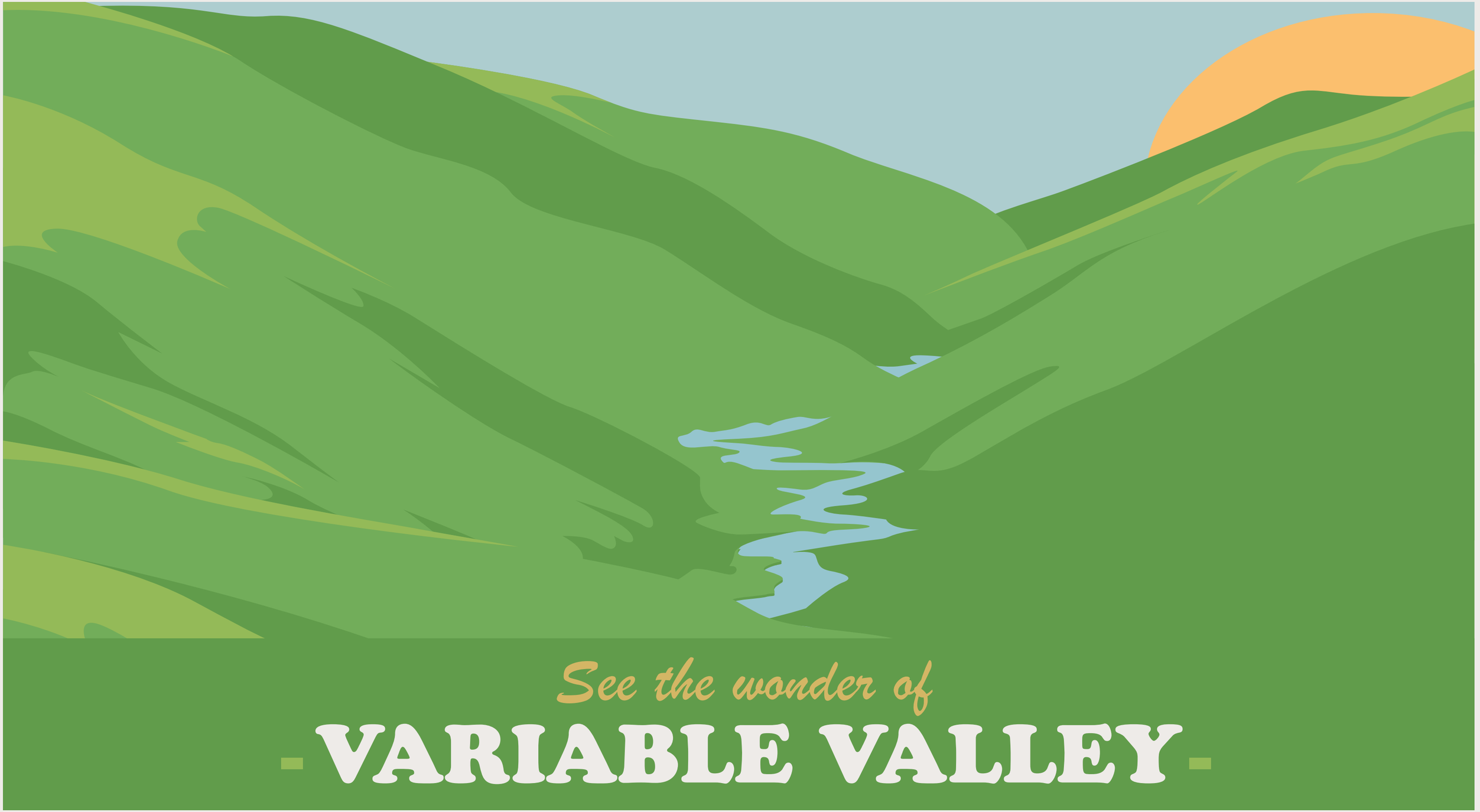
Returns a number value

Spaces and any non-alphabetic characters are counted, too!

```
> "One Fish, Two Fish, Red Fish, Blue Fish".length
```

→ 39



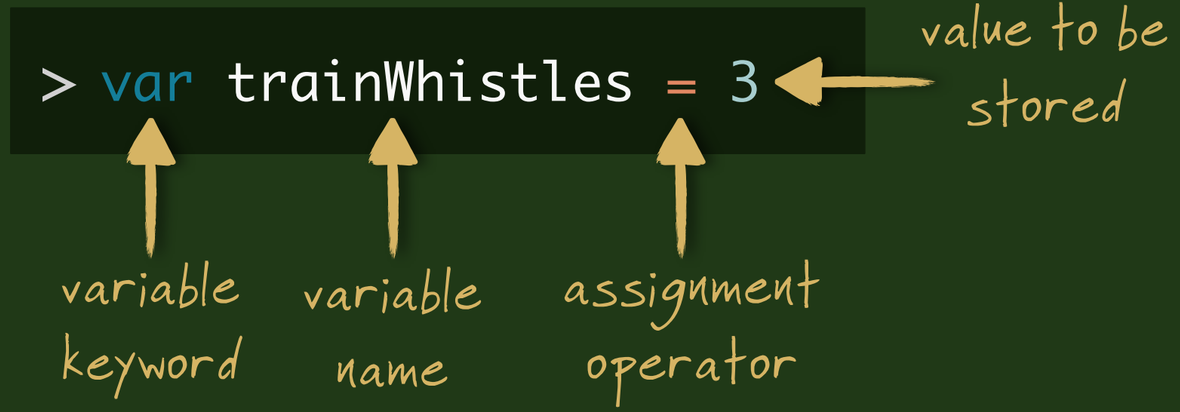

See the wonder of
VARIABLE VALLEY

LEVEL 2

VARIABLE VALLEY

STORING OUR VALUES

JavaScript uses variables to store and manage data



NAMING VARIABLES

Rules and regulations

```
var no spaces
```

no spaces in the name

```
var 3blindmice
```

no digits in front

```
var scored_is_fine
```

underscores are okay, but often irritating

```
var get$
```

dollar signs are also cool...but don't be that person

```
var $_$
```

slightly stupid, but technically legal

```
var goodName
```

begin with lowercase, later words capitalized, "camel case"

```
var mortalKombat2
```

FATALITY!!

CHANGING VARIABLE CONTENTS

Want to change a Variable's value? It's your lucky day.

```
>var trainWhistles = 3
```

```
> trainWhistles
```

→ 3

```
>trainWhistles = 9
```

no 'var' keyword this time, because JavaScript already "knows" about the variable

```
> trainWhistles
```

→ 9

```
>trainWhistles = trainWhistles + 3
```

uses current value to calculate new value

```
> trainWhistles
```

→ 12

CHANGING VARIABLE CONTENTS

Want to change a Variable's value? It's your lucky day.

```
> trainWhistles = trainWhistles + 3
```

same operation, different syntax

```
> trainWhistles += 3
```

```
> trainWhistles
```

→ 12

```
> trainWhistles
```

→ 15

CHANGING VARIABLE CONTENTS

Want to change a Variable's value? It's your lucky day.

```
> trainWhistles += 3
```

```
> trainWhistles
```

→ 15

```
> trainWhistles = trainWhistles * 2
```

same operation, different syntax

```
> trainWhistles
```

→ 30

```
> trainWhistles *= 2
```

```
> trainWhistles
```

→ 60

That's, like, a lot of whistles.

USING VARIABLES

Variable names also act as substitutes for the data they point to

```
> trainWhistles = 3
```

```
> "All of our trains have " + trainWhistles + " whistles!"
```

→ `"All of our trains have 3 whistles!"`

```
> "But the Pollack 9000 has " + (trainWhistles * 3) + "!"
```

→ `"But the Pollack 9000 has 9!"`

USING VARIABLES

Variable names also act as substitutes for the data they point to

```
> trainWhistles = 3
```

```
> "But the Pollack 9000 has " + (trainWhistles * 3) + "!"
```

```
> var pollack9000 = trainWhistles * 3
```

```
> pollack9000
```

→ 9

USING VARIABLES

Variable names also act as substitutes for the data they point to

```
> trainWhistles = 3
```

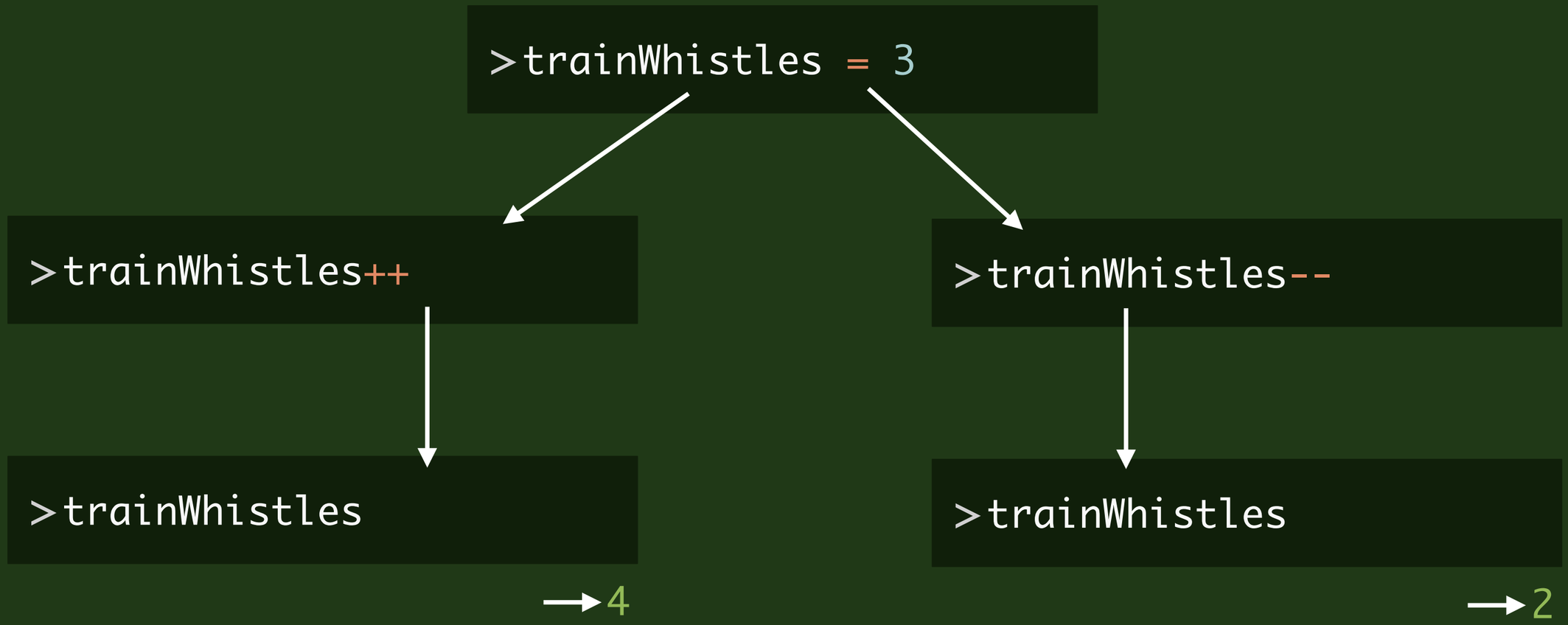
```
> var pollack9000 = trainWhistles * 3
```

```
> "But the Pollack 9000 has " + pollack9000 + "!"
```

→ `"But the Pollack 9000 has 9!"`

INCREMENTING AND DECREMENTING

A simple syntax for increasing or decreasing a variable's value by 1



VARIABLES STORE STRINGS, TOO!

JavaScript can store anything in variables.

```
> var welcome = "Welcome to the JavaScript Express Line!"
```

```
> var safetyTip = "Look both ways before crossing the tracks."
```

```
> welcome + "\n" + safetyTip
```

→ "Welcome to the JavaScript Express Line!
→ Look both ways before crossing the tracks."

USING VARIABLE NAMES WITH STRINGS

Variable names can also access the length property

```
> var longString = "I wouldn't want to retype this String every time."
```

```
> longString.length
→ 49
```

If a variable holds a String, we can access the length property directly from the variable name.

MORE COMPARISONS WITH VARIABLES

Comparing String lengths using the length property

```
> var longWordOne = "antidisestablishmentarianism"
```

```
> var longWordTwo = "supercalifragilisticexpialidocious"
```

Compares two numbers returned by the length properties

```
> longWordOne.length > longWordTwo.length
```

→ false

FINDING SPECIFIC CHARACTERS WITHIN STRINGS

Each position in a String has a numbered “index” starting from 0

```
> var sentence = "Antidisestablishmentarianism is fun to say!"
```

0 8 28 42

Think of index numbers as being a "distance from the starting character." Thus, the first character is "zero" away from itself.

Spaces are characters too!

There will always be one less index number than characters present!

```
> sentence.length
```

→ 43

Since the index starts at zero, but the length is counted by number of characters, the length value will always be one more than the last index.

FINDING SPECIFIC CHARACTERS WITHIN STRINGS

Each position in a String has a numbered “index” starting from 0

```
> var sentence = "Antidisestablishmentarianism is fun to say!"
```

```
> sentence.charAt(11)
```

→ `"b"`

```
> sentence.charAt(31)
```

→ `" "`

```
> sentence.charAt(42)
```

→ `"!"`

The charAt( ) method retrieves the character at a specific index.

VARIABLES HELP ORGANIZE DATA

Creating a versatile message out of flexible pieces

```
> var trainsOperational = 8
```

```
> var totalTrains = 12
```

```
> var operatingStatus = " trains are operational today."
```

```
> trainsOperational + " out of " + totalTrains + operatingStatus
```

→ `"8 out of 12 trains are operational today."`

VARIABLES HELP ORGANIZE DATA

Creating a versatile message out of flexible pieces

```
> var trainsOperational = 10
```

```
> var totalTrains = 12
```

```
> var operatingStatus = " trains are operational today."
```

```
> trainsOperational + " out of " + totalTrains + operatingStatus
```

→ `"10 out of 12 trains are operational today."`


Go Over the Edge at
FILES FALLS

LEVEL 3

FILES FALLS

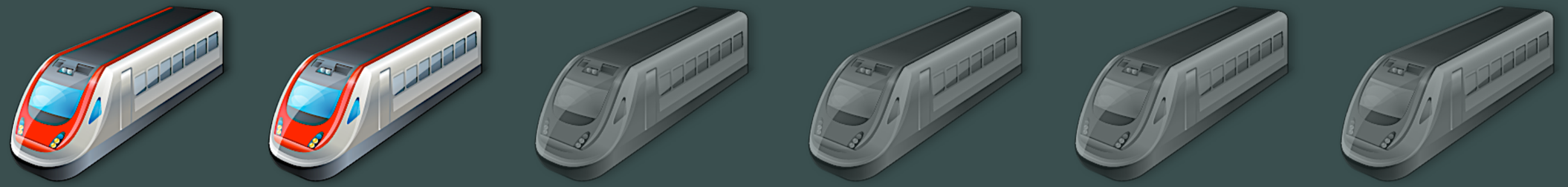
OBJECTIVE: TRAIN SYSTEM STATUS

We want to print a list of running and stopped trains for passengers

1 2 3 4 5 6

✓ ✓ ✓ ✓ ✓ ✓

7 8 9 10 11 12



✓ ✓ ✗ ✗ ✗ ✗

WHICH TRAINS ARE RUNNING?

We need a printout for each train since each train number is unique

```
> "Train #" + 1 + " is running."
```

→"Train #1 is running."

```
> "Train #" + 2 + " is running."
```

→"Train #2 is running."

```
> "Train #" + 3 + " is running."
```

→"Train #3 is running."

And on and on and on . . . wow,
this sucks with significance.

RUNNING JAVASCRIPT IN AN HTML FILE

Embedding code that signals which JavaScript file to use

The src signals which file our runnable JS code is in. In this case, let's call our file trains.js.

index.html

```
<html>
<head>
<script src="trains.js"></script>
</head>
<body>
<h1>JAVASCRIPT EXPRESS!</h1>
...
</body>
</html>
```

The <script> tag says "YO, we want some JavaScript code executed!"

These dots just mean that a bunch of html code exists here to handle other parts of the website.

Remember to turn off your 'script' tag!

SO NOW, WHAT IS THAT TRAINS.JS FILE?

Building a file of JavaScript code that can be used repeatedly by our site

You can use your favorite simple text-based editor to make any .js file.

trains.js

```
Our JavaScript code
for printing the
running trains will
go in here!
```

Inside the file, we write the JavaScript code that we want to be executed when the index.html file reaches our script tag.

WHERE DO THESE FILES GO?

Adding our files to an appropriate location

root/

index.html

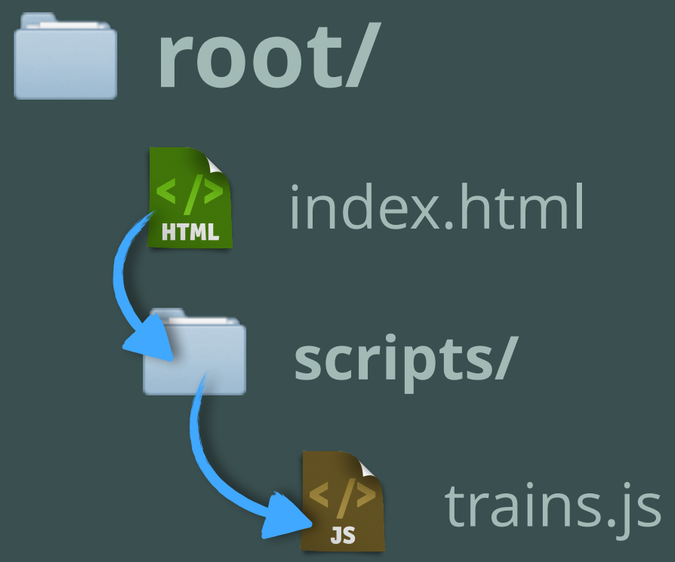
trains.js

So that our `src = "trains.js"` code works correctly, we'd need to place our trains.js file in the same directory as the index.html file.

```
<html>
<head>
<script src="trains.js"></script>
</head>
<body>
<h1>JAVASCRIPT EXPRESS!</h1>
...
</body>
</html>
```


STAYING ORGANIZED AS YOU CODE

Many websites will keep all scripts in descriptive locations

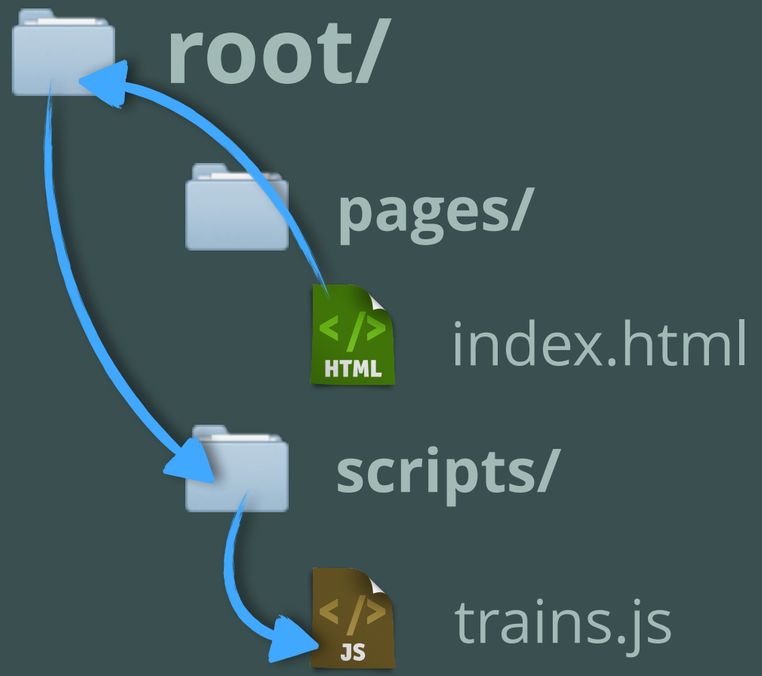



```
<html>
<head>
<script src="scripts/trains.js"></script>
</head>
<body>
<h1>JAVASCRIPT EXPRESS!</h1>
...
</body>
</html>
```

This syntax says to `index.html`, first go down to the `scripts` folder, and there you'll find the train.js file.

LINK SYNTAX FOR DISTANT FILES

Staying organized means we'd have to be more detailed on our src links.




```
<html>
<head>
<script src="../scripts/trains.js"></script>
</head>
<body>
<h1>JAVASCRIPT EXPRESS!</h1>
...
</body>
</html>
```

The double-dot syntax indicates to "go up a folder." This first takes the path back to the root folder, and then takes it down to the scripts folder, where our train.js is found.

TRYING SOME CODE IN A FILE

Let's try executing a few console-style expressions.

trains.js

```
"Train #" + 1 + " is running."
"Train #" + 2 + " is running."
"Train #" + 3 + " is running."
```

Something's up! The compiler doesn't understand what we've placed in our file in the same way that the console does.

→ ERROR

BUILDING STATEMENTS IN FILES

We need a way to differentiate between executable statements

So here's our Console Entry:

```
> var trainsOperational = 8
```

But guess what? If we enter multiple console statements to execute in one file...

```
var trainsOperational = 8
trainsOperational = trainsOperational + 4
"There are " + trainsOperational + " running trains."
```

...it is interpreted by the compiler pretty much like this utter nonsense:

```
var trainsOperational = 8trainsOperational = trainsOperational +
4"There are " trainsOperational + " running trains."
```


ENTER SEMICOLONS!

We like semicolons; they do not suck.

Console Entry

```
> var trainsOperational = 8
```

File Entry

```
var trainsOperational = 8;
```

Add a semicolon in files to tell the compiler where statements end.

Multiple executable statements are all separated by semicolons

wootForSemicolons.js

```
var trainsOperational = 8;
trainsOperational = trainsOperational + 4;
"There are " + trainsOperational + " running trains.";
```

No error, but now our output is blank? No printed string? What's up?

→ (blank) ✖

PRINTING FROM A FILE TO THE CONSOLE

The console.log() method outputs results of code operations in files

```
var totalTrains = 12;
var trainsOperational = 8;
```

Place expression inside the enclosing parentheses of the console.log() method

```
console.log("There are " + trainsOperational + " running trains.");
```

```
There are 8 running trains.
```

Notice now that we get the actual String contents, with no quote notation.

```
console.log(trainsOperational == totalTrains);
```

The console.log() returns the results of any expression we want printed.

```
false
```

USING CONSOLE.LOG( ) IN TRAINS.JS

Now we can print out every running train.

trains.js

```
console.log("Train #" + 1 + " is running.");
console.log("Train #" + 2 + " is running.");
console.log("Train #" + 3 + " is running.");
console.log("Train #" + 4 + " is running.");
console.log("Train #" + 5 + " is running.");
console.log("Train #" + 6 + " is running.");
console.log("Train #" + 7 + " is running.");
console.log("Train #" + 8 + " is running.");
```

```
Train #1 is running.
Train #2 is running.
Train #3 is running.
Train #4 is running.
Train #5 is running.
Train #6 is running.
Train #7 is running.
Train #8 is running.
```

Combine semicolons to separate executable statements, and the console.log() method to get results printed to the console, and we have a winner!...sort of.

OUR CURRENT TRAIN STATUS SYSTEM

index.html

```
<html>
<head>
<script src="trains.js" />
</head>
<body>
<h1>JAVASCRIPT EXPRESS!</h1>
</body>
</html>
```

trains.js

```
console.log("Train #" + 1 + " is running.");
console.log("Train #" + 2 + " is running.");
console.log("Train #" + 3 + " is running.");
console.log("Train #" + 4 + " is running.");
console.log("Train #" + 5 + " is running.");
console.log("Train #" + 6 + " is running.");
console.log("Train #" + 7 + " is running.");
console.log("Train #" + 8 + " is running.");
```

We still haven't solved that repetitive code issue!