**Variables**

Programmers use variables to write code that is easy to understand and repurpose.

Imagine you're writing a weather app. Your thermometer outside reports the temperature in Celsius, but your goal is to record the temperature in Fahrenheit.

You write a program that takes a temperature of 15 degrees Celsius and calculates the temperature in Fahrenheit.

Once you've done this though, you see the temperature now reads 16 degrees Celsius. To find Fahrenheit again, you'd need to write a whole new program to convert 16degrees Celsius to Fahrenheit.

That's where variables come in. Variables allow us to assign data to a word and use the word to reference the data. If the data changes (like degrees Celsius) we can replace the variable's value instead of re-writing the program.

In this lesson you will learn about two ways to declare *variables*: let and const.

|  |
| --- |
| // This line of code sets the variable location to the string New York City  const location = 'New York City';  // This line of code sets the variable latitude to the number 40.7  let latitude = 40.7;  // This line of code sets the variable inNorthernHemisphere to true  let inNorthernHemisphere = true;  console.log(location);  console.log(latitude);  console.log(inNorthernHemisphere); |
| New York City  40.7  true |

# Create a Variable: const

Let's dive in and see a variable in the wild. Here is how you declare a constant variable:

const myName = 'Arya'; console.log(myName); // Output: Arya

Let's consider the example above:

1. const, short for constant, is a JavaScript keyword that creates a new variable with a value that cannot change.
2. myName is the variable's name. Notice that the word has no spaces, and we capitalized the N. Capitalizing in this way is a standard convention in JavaScript called camelCasing, because the capital letters look like the humps on a camel's back.
3. = is the assignment operator. It assigns the value ('Arya') to the variable (myName).
4. 'Arya' is the value assigned (=) to the variable myName.

After the variable is declared, we can print 'Arya' to the console with: console.log(myName).

You can save any data type in a variable. For example, here we save numbers:

const myAge = 11; console.log(myAge); // Output: 11

In the example above, on line 1 the myAgevariable is set to 11. Below that, console.log() is used to print 11 to the console.

**Instructions**

**1.**

Create a constant variable named entree and set it to equal to the string 'Enchiladas'.

Hint

Use the const keyword to make a variable.

**2.**

Now, under the entree variable, create a constant variable named price that saves the number 12.

Hint

Use const to create a variable named price and save the number 12 to it.

**3.**

Under the price variable, use console.log() to print the value saved to entree.

On the following line, use console.log() to print the value saved to price.

**4.**

Copy and paste the following code to the bottom of your program.

entree = 'Tacos';

This code throws the following error when you run your code:

TypeError: Assignment to constant variable.

We'll discuss what's going on here in the next exercise.

# Create a Variable: let

In the final task of the previous exercise you received the following error:

TypeError: Assignment to constant variable.

JavaScript threw an error because you assigned a new value to a constant variable. Constant variables, as their name implies, are constant — you cannot assign them a different value.

Let variables however, can be reassigned.

let meal = 'Enchiladas'; console.log(meal); meal = 'Tacos'; console.log(meal); // output: Enchiladas // output: Tacos

In the example above, the let keyword is used to create the meal variable with the string 'Enchiladas' saved to it. On line three, the meal variable is changed to store the string 'Tacos'.

You may be wondering, when to use constvs let. In general, only use const if the value saved to a variable does not change in your program.

# Undefined

What happens if you create a variable, but don't assign it a value?

JavaScript creates space for this variable in memory and sets it to undefined. Undefinedis the fifth and final primitive data type. JavaScript assigns the undefined data type to variables that are not assigned a value.

let whatAmI;

In the example above, we created the variable whatAmI without any value assigned to it. JavaScript creates the variable and sets it equal to the value undefined.

**Mathematical Assignment Operators**

In this exercise, let's consider how we can use variables and math operators to calculate new values and assign them to a variable. Check out the example below:

let x = 4; x = x + 1;

In the example above, we created the variable x with the number 4 assigned to it. On the following line, x = x + 1 increases the value of x from 4 to 5.

Notice, on line two in the example above, to increment x by one we had to write the xvariable on the left and right side of the assignment operator (=). Using a variable twice in one expression is redundant and confusing.

To address this, JavaScript has a collection of built-in *mathematical assignment operators* that make it easy to calculate a new value and assign it to the same variable without writing the variable twice. See examples of these operators below.

let x = 4; x += 2; // x equals 6 let y = 4; y -= 2; // y equals 2 let z = 4; z \*= 2; // z equals 8 let r = 4; r++; // r equals 5 let t = 4; t--; // t equals 3

In the example above, operators are used to calculate a new value and assign it to the same variable. Let's consider the first three and last two operators separately:

1. The first three operators (+=, -=, and \*=) perform the mathematical operation of the first operator (+, -, or \*) using the number on the right, then assign the new value to the variable.
2. The last two operators are the increment (++) and decrement (--) operators. These operators are responsible for increasing and decreasing a number variable by one, respectively.

# String Interpolation

In previous exercises, we assigned strings to variables. Here, you will learn how to insert the content saved to a variable into a string.

The JavaScript term for inserting the data saved to a variable into a string is string interpolation.

The + operator, known until now as the addition operator, is used to interpolate (insert) a string variable into a string, as follows:

let myPet = 'armadillo'; console.log('I own a pet ' + myPet + '.'); // Output: 'I own a pet armadillo.'

In the example above, we saved the value 'armadillo' to the myPet variable. On the second line, the + operator is used to combine three strings: I own a pet, the value saved to myPet, and .. We log the result of this interpolation to the console as:

I own a pet armadillo.

**String Interpolation II**

In the newest version of JavaScript (ES6) we can insert variables into strings with ease, by doing two things:

1. Instead of using quotes around the string, use backticks (this key is usually located on the top of your keyboard, left of the 1 key).
2. Wrap your variable with ${myVariable}, followed by a sentence. No +s necessary.

ES6 string interpolation is easier than the method you used last exercise. With ES6 interpolation we can insert variables directly into our text.

It looks like this:

let myPet = 'armadillo' console.log(`I own a pet ${myPet}.`) // Output: 'I own a pet armadillo.'

In the example above, the backticks (`) wrap the entire string. The variable (myPet) is inserted using ${}. The resulting string is:

I own a pet armadillo.

**Review Variables**

Nice work! This lesson covered a lot of foundational skills that you will often use in your future programming endeavors.

Let's review what we learned:

* Variables hold reusable data in a program.
* JavaScript will throw an error if you try to reassign const variables.
* You can reassign variables that you create with the let keyword.
* Unset variables store the primitive data type undefined.
* Mathematical assignment operators make it easy to calculate a new value and assign it to the same variable.
* The + operator is used to interpolate (combine) multiple strings.
* In JavaScript ES6, backticks (`) and ${}are used to interpolate values into a string.

# Introduction to Control Flow

In this lesson, we'll explore how we can use the building blocks of JavaScript to write programs that make decisions.

Control flow statements enable JavaScript programs to make decisions by executing code based on a condition. If a given condition is true, we execute one block of code. If the statement is false, we execute another block of code. For instance, if we were making a game in which the user had to choose which door to enter, we'd need a way for the program to know what to do once the user was in the next room.

In this lesson, we'll learn how to make decisions with JavaScript and how it can control the program's flow.

|  |
| --- |
| let userName = 'Amy';  let knowsJavaScript = false;  if (knowsJavaScript && userName) {  console.log('Great, ' + userName + '! Get ready to practice your JavaScript!');  } else if (knowsJavaScript) {  console.log('Great! Get ready to practice your JavaScript!');  } else if (userName) {  console.log('Great, ' + userName + '! Get ready to learn something new!');  } else {  console.log('Great! Get ready to learn something new!');  } |

# if/else Statements

The core task of programming is writing lists of instructions for computers, or translating our ideas from human-speak to computer-speak.

Let's learn how we can ask JavaScript to think like us and make decisions the way we do.

We'll start with human-speak. Many decisions we make everyday boil down to this sentence in some form:

"If something is true, let's do option 1, or else, if it is false, let's do option 2."

This sentence looks fairly similar when we write it with JavaScript. See for yourself:

let needsCoffee = true; if (needsCoffee === true) { console.log('Finding coffee'); } else { console.log('Keep on keeping on!'); }

1. Lines of code between curly braces are called blocks. if/else statements have two code blocks. If the variable needsCoffee is true, the program will run the first block of code. Otherwise, it will run the other block of code.
2. needsCoffee is the condition we are checking inside the if's parentheses. Since it is equal to true, our program will run the code between the first opening curly brace { (line 2) and the first closing curly brace } (line 4). It will ignore the else { ... } part. In this case, we'd see Finding coffee log to the console.
3. If needsCoffee were false, only the console.log() statement in the elseblock would be executed.

if/else statements are how programs can process yes/no questions programmatically.

# if/else Statements

The core task of programming is writing lists of instructions for computers, or translating our ideas from human-speak to computer-speak.

Let's learn how we can ask JavaScript to think like us and make decisions the way we do.

We'll start with human-speak. Many decisions we make everyday boil down to this sentence in some form:

"If something is true, let's do option 1, or else, if it is false, let's do option 2."

This sentence looks fairly similar when we write it with JavaScript. See for yourself:

let needsCoffee = true; if (needsCoffee === true) { console.log('Finding coffee'); } else { console.log('Keep on keeping on!'); }

1. Lines of code between curly braces are called blocks. if/else statements have two code blocks. If the variable needsCoffee is true, the program will run the first block of code. Otherwise, it will run the other block of code.
2. needsCoffee is the condition we are checking inside the if's parentheses. Since it is equal to true, our program will run the code between the first opening curly brace { (line 2) and the first closing curly brace } (line 4). It will ignore the else { ... } part. In this case, we'd see Finding coffee log to the console.
3. If needsCoffee were false, only the console.log() statement in the elseblock would be executed.

if/else statements are how programs can process yes/no questions programmatically.

**True and False Values**

Even non-boolean data types, like strings and numbers, can still be used like booleans to determine control flow.

In JavaScript, all values have a *truthy* or *falsy*value. This means that they evaluate to true or false when they are used as a part of a control flow condition.

All variables that have been declared and assigned are truthy unless they contain one of the six values listed below:

* false
* 0 and -0
* "" and '' (empty strings)
* null
* undefined
* NaN (Not a Number)

For example:

let variableOne = 'I Exist!'; if (variableOne) { // This code will run because variableOne contains a truthy value. } else { // This code will not run because the first block ran. }

The second line of this program checks a condition if (variableOne). By only writing the name of the variable as the condition, we are checking the *truthiness* of the variableOne. In this case, variableOnebehaves like the value true because a string that is not empty is truthy.

Here is an example with numbers.

let numberOfApples = 0; if(numberOfApples){ console.log('Let us eat!'); // This code will not run because 0 is a falsy value } else { console.log('No food left!'); // This code will run }

Since 0 is one of the six falsy values, the code within the else block runs.

There is an important distinction between a variable's value and its truthiness:

* numberOfApples's value is 0 because that is the data saved to the variable.
* numberOfApples is falsy when used in control flow because it exists and does contain one of the six falsy values listed above.

# True and False Values II

In programming, we often evaluate whether or not an expression is true or truthy. Conveniently, JavaScript provides a shorthand notation for this.

let isRaining = true; if (isRaining) { console.log('Carry an umbrella!'); } else { console.log('Enjoy the sun!'); }

In the example above, the condition is simply if (isRaining). In JavaScript, this is evaluating whether isRaining is truthy. If you read the code out loud to yourself, it sounds like a simple sentence: "If it's raining, carry an umbrella. Else, enjoy the sun!"

JavaScript provides an operator for swapping the truthiness and falsiness of values - the exclamation point (!). We can use this in conditional statements as shorthand to check if the value of a variable evaluates to false rather than true.

let isPhoneCharged = true; if (!isPhoneCharged) { console.log('Plug in your phone!'); } else { console.log('No need to charge!'); }

In the example above, the program checks if isPhoneCharged evaluates to false. Because isPhoneCharged is true, the second block of code will execute.

**Comparison Operators**

In addition to checking whether a variable evaluates to true or false, sometimes we need to compare variables to other values. We can achieve this with *comparison operators*.

There are two comparisons you might be familiar with:

* Less than: <
* Greater than: >

You may also recognize these:

* Less than or equal to: <=
* Greater than or equal to: >=

These comparisons evaluate to true or false.

**Comparison Operators II**

There are two more useful comparisons we can make. Often, we might want to check if two things are equal to each other or if they are not.

1. To check if two things equal each other, we write === (three = signs in a row).
2. To check if two things *do not* equal each other, we write !== (an exclamation with two = signs in a row).

It can be confusing when to use one = sign and when to use three === signs. Use a single = to assign a value to a variable. Use ===to compare the values of two different variables.

# else if Statements

We've explored if/else statements that answer questions that are either yes or no. What can we do if we have a question that has multiple yes conditions, or multiple no conditions?

We can add more conditions to our if/else statement with else if. Check out how this fits into our current knowledge of if/else statements:

let stopLight = 'green'; if (stopLight === 'red') { console.log('Stop'); } else if (stopLight === 'yellow') { console.log('Slow down'); } else if (stopLight === 'green') { console.log('Go!'); } else { console.log('Caution, unknown!'); }

1. We created a variable named stopLightthat is assigned to the string green.

2. Then, there's an if/else statement with multiple conditions, using else if. else ifallows us to check multiple values of the stopLight variable and output different things based on its color.

3. The block ends with the singular else we have seen before. The else is a catch-all for any other situation. For instance, if the stopLight was blinking blue, the last elsewould catch it and return a default message.

**Logical Operators**

We can translate certain thoughts into JavaScript code such as, "Are these things equal?" with ===, or, "Is one thing greater than another thing?" with >.

In English, sometimes we say "both of these things" or "either one of these things." Let's translate those phrases into JavaScript with special operators called *logical operators*.

1. To say "both must be true," we use &&.
2. To say "either can be true," we use ||.

For example:

if (stopLight === 'green' && pedestrians === false) { console.log('Go!'); } else { console.log('Stop'); }

1. In the example above, we make sure that the stopLight is 'green' and (&&) there are no pedestrians before we log Go!.
2. If either of those conditions is false, we log Stop.

Just like the operators we learned previously, these logical operators will return either true or false.

These logical operators are helpful when writing if/else statements since they let us make sure multiple variables are true or false. We can combine these operators with all of the ones we have learned throughout this lesson.

**switch Statements**

Before we move on, let's circle back to else if statements.

Using else if is a great tool for when we have a few different conditions we'd like to consider.

else if is limited, however. If we want to write a program with 25 different conditions, like a JavaScript cash register, we'd have to write *a lot* of code, and it can be difficult to read and understand.

To deal with times when you need many else if conditions, we can turn to a switchstatement to write more concise and readable code.

To a computer, a switch statement and an if/else statement are the same, but a switch statement can be easier for other humans to read. Part of being a good developer is writing code that both computers and other humans can read.

switch statements look like this:

let groceryItem = 'papaya'; switch (groceryItem) { case 'tomato': console.log('Tomatoes are $0.49'); break; case 'lime': console.log('Limes are $1.49'); break; case 'papaya': console.log('Papayas are $1.29'); break; default: console.log('Invalid item'); break; }

1. The switch keyword initiates the statement and is followed by ( ... ), which contains the condition that each case will compare to. In the example, the condition is groceryItem.
2. Inside the block, { ... }, there are cases. case is like the else if part of an if/else if/else statement. The word following the first case is 'tomato'. If groceryItem equalled 'tomato', that case's console.log()would run.
3. groceryItem equals 'papaya', so the first and second case statements are skipped. The third case runs since the case is 'papaya', which matches groceryItem's value. This particular program will log Papayas are $1.29.
4. Then the program stops with the breakkeyword. This keyword will prevent the switch statement from executing any more of its code. Without adding breakat the end of each case, the program will execute the code for all matching cases and the default code as well. This behavior is different from if/elseconditional statements which execute only one block of code.
5. At the end of each switch statement, there is a default condition. If none of the cases are true, then this code will run.

|  |
| --- |
| let moonPhase = 'full'  switch(moonPhase){  case 'full':  console.log('How1');  break;  case 'mostly full':  console.log('Arms and legs are getting hairier.');  break;  case 'mostly new':  console.log('Back on two feet.');  break;  default:  console.log('Invalid moon phase');  break;  }  if(moonPhase === 'full' || isFoggyNight){  console.log('How1!');  }else if(moonPhase === 'mostly full'){  console.log('Arms and legs are getting hairier.');  }else if (moonPhase === 'mostly new'){  console.log('Back on two feet.');  }else{  console.log('Invalid moon phase');  } |

**Ternary Operator**

In the previous exercise, we learned shorthand for writing multiple if/else if/else statements to make them easier to read. JavaScript also provides a way to shorten simple if/else statements called the *ternary operator*.

let isNightTime = true; if (isNightTime) { console.log('Turn on the lights!'); } else { console.log('Turn off the lights!'); }

In the example above, we see a very familiar pattern. See the example below for an equivalent way to express this.

isNightTime ? console.log('Turn on the lights!') : console.log('Turn off the lights!');

The code in the example above will operate exactly as the code from the previous example. Let's break this example into its parts:

1. isNightTime ? — the conditional statement followed by a question mark. This checks if isNightTime is truthy.
2. console.log ('Turn on the lights!') — this code will be executed if the condition is truthy.
3. : — a colon separates the two different blocks of code that can be executed.
4. console.log('Turn off the lights!'); — this code will be executed if the condition is falsy

In this example, we checked if the value of a variable was true or false. The ternary operator can be used for any condition that can be evaluated to true or false, such as those with comparison operators.

age >= 16 ? console.log('You are old enough to drive in the United States!') : console.log('You are not old enough to drive in the United States!');

In the example above, the conditional statement is checking whether the value of the variable age is greater than or equal to 16. If so, a message that states the user is old enough to drive will be logged to the console. Otherwise, a message that states the user is not old enough to drive will be logged.

**Instructions**

**1.**

In **main.js**, refactor the first if/elseblock to use the ternary operator.

Hint

First, check the truthiness of isLocked using the ? operator. The beginning of the ternary operator looks like:

isLocked ?

Then you have to include your truthy expression. The code will look like:

isLocked ? console.log('You will need a key to open the door.')

Afterward, separate the truthy expression from the falsy expression using :. Fully pieced together, the code will be:

isLocked ? console.log('You will need a key to open the door.') : console.log('You will not need a key to open the door.')

**2.**

In **main.js**, refactor the second if/else block to use the ternary operator.

**3.**

In **main.js**, refactor the third if/elseblock to use the ternary operator.

Hint

If you're using single quotes in the falsy expression, make sure you use the escape character \. Otherwise, JavaScript will not interpret the string correctly. For example:

'Look we\'re using the escape character!'

You can also avoid using \ altogether by using double quotes:

"Look we're not using the escape character!"

|  |
| --- |
| let isLocked = false;  isLocked ? console.log('You will need a key to open the door.') : console.log('You will not need a key to open the door.');  let isCorrect = true;  isCorrect ? console.log('Correct!') : console.log('Incorrect!');  let favoritePhrase = 'Love That!';  favoritePhrase === 'Love That!' ? console.log('I love that!') : console.log("I don't love that!"); |

**Review: Control Flow**

Way to go! We just learned a lot of control flow concepts:

* if/else statements make binary decisions and execute different code based on conditions.
* All conditions are evaluated to be truthy or falsy.
* We can add more conditional statements to if/else statements with else if.
* switch statements make complicated if/else statements easier to read and achieve the same result.
* The ternary operator (?) and a colon (:) allow us to refactor simple if/elsestatements.
* Comparison operators, including <, >, <=, and >= can compare two variables or values.
* After two values are compared, the conditional statement evaluates to trueor false.
* The logical operator && checks if both sides of a condition are truthy.
* The logical operator || checks if either side is truthy.
* The logical operator !== checks if the two sides are not equal.
* An exclamation mark (!) switches the truthiness / falsiness of the value of a variable.
* One equals symbol (=) is used to assign a value to a variable.
* Three equals symbols (===) are used to check if two variables are equal to each other.