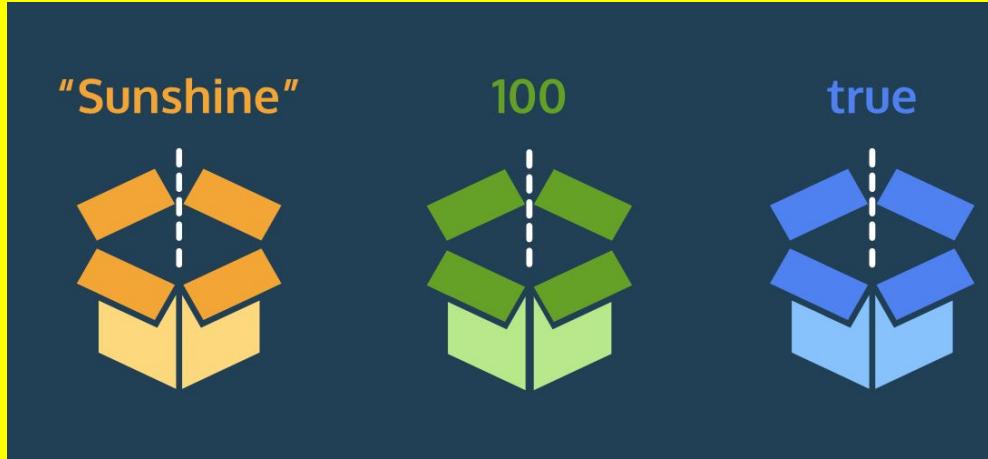


# JavaScript II

---

Variables & Conditionals



# Variables

# Variables - containers for data

In programming, a variable is a container for a value. You can think of variables as little containers for information that live in a computer's memory. Information stored in variables, such as a username, account number, or even personalized greeting can then be found in memory.

In short, variables label and store data in memory.

It is important to distinguish that variables are not values; they contain values and represent them with a name.

# Declaring variables

There are three keywords used to declare variables in JavaScript:

- **var** - the most common, standard in JavaScript and other C-based languages
- **let** - new in ES2015 (ES6). Introduces block scope
- **const** - similar to let, but cannot be reassigned. Also introduced in ES6

# Syntax

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

1. `var`, short for variable, is a JavaScript *keyword* that creates, or *declares*, a new variable.
2. `myName` is the variable's name. Capitalizing in this way is a standard convention in JavaScript called *camel casing*. In camel casing you group words into one, the first word is lowercase, then every word that follows will have its first letter uppercased. (e.g. `camelCaseEverything`).
3. `=` is the *assignment operator*. It assigns the value (`'Arya'`) to the variable (`myName`).
4. `'Arya'` is the *value* assigned (`=`) to the variable `myName`. You can also say that the `myName` variable is *initialized* with a value of `'Arya'`.

# Let's Practice

- Create a new .js file called “variables.js” and save it in a directory called “Class\_5”
- Declare a variable named `favoriteFood` using the `var` keyword and assign to it the string '`pizza`'.
- Declare a variable named `numOfSlices` using the `var` keyword and assign to it the number `8`.
- use `console.log()` to print the value saved to `favoriteFood`.
- use `console.log()` to print the value saved to `numOfSlices`.

# Your Turn!

- Create your own variable and assign it a string value
- console.log the value of this variable
- Create another variable and assign it a float value
- Console.log the value of this new variable

# Browser Practice!

Let's use JavaScript variables to change the style properties of text.

Create a file called js1.html and add the following code:

```
1  <html>
2  <body>
3
4  <p id="p2">JavaScript!</p>
5
6  <script>
7  </script>
8
9  </body>
10 </html>
```

Now let's use JavaScript to change the style properties:

```
1 <html>
2 <body>
3
4 <p id="p2">JavaScript!</p>
5
6 <script>
7
8 document.getElementById("p2").style.color = "red";
9 document.getElementById("p2").style.fontSize = "150px";
10
11 </script>
12
13 </body>
14 </html>
```

Instead of hard-coding in the color “red” and size “150px”, let’s make variables called color and size to hold this data:

```
1  <html>
2  <body>
3
4  <p id="p2">JavaScript!</p>
5
6  <script>
7  var color ="red";
8  var size ="150";
9
10 document.getElementById("p2").style.color = color;
11 document.getElementById("p2").style.fontSize = size+"px";
12
13 </script>
14
15 </body>
16 </html>
```

# Your turn

What happens when we change the data held by the variables size and color?

Change var color="red"; to var color="green";

Change var size="150"; to var size ="300";

Our text changes style properties!

Now, write your own text in a separate <p> tag and assign it another ID.

Use JavaScript variables to change its color and size.

# Variables are dynamic (most of them)

The power and flexibility of variables really lies in their ability to hold dynamic data values. This means, we can change the values they hold for a variety of purposes.

```
1 var meal = 'Enchiladas';
2 console.log(meal); // Output: Enchiladas
3 meal = 'Burrito';
4 console.log(meal); // Output: Burrito|
```

In this example, we initiate the value of the variable meal first as Enchiladas, then we change it to Burrito.

Let's write this code back in our variables.js file to see it in action.

# Your turn!

Create another variable called “calories” and assign it an integer value.

Print the value of calories to the console.

After your console.log statement, re-assign the value of calories to a float number

Print out the new value of calories to the console

# Browser Practice!

We can extend this concept to our browser exercise in js1.html

```
1  <html>
2  <body>
3
4  <p id="p2">JavaScript!</p>
5  <p id="p3">Variables!</p>
6
7  <script>
8  var color ="red";
9  var size ="150";
10
11 document.getElementById("p2").style.color = color;
12 document.getElementById("p2").style.fontSize = size+"px";
13
14 color ="blue";
15 size ="75";
16
17 document.getElementById("p3").style.color = color;
18 document.getElementById("p3").style.fontSize = size+"px";
19 </script>
20
21 </body>
22 </html>
```

Create another `<p>` element and assign it a unique ID. Add the text “JS is Awesome!” to this paragraph.

Change the value of variables color and size a third time to make the text purple and 10px.

Create a new `<p>` element and write in any text you would like.

Now, change the value of the variables you created on your own in the previous exercises to add another color and size.

# **Mathematical Assignment Operators**

# Changing a variable's value

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:

```
9  <script type="text/javascript">
10 var w = 4;
11 w = w + 1;
12
13 console.log(w); // Output: 5
```

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

# Other arithmetic operators

We also have access to other mathematical assignment operators: `-`, `*`, and `/` which work in a similar fashion.

```
15  var x = 20;
16  x = x - 5; // Can be written as x -= 5;
17  console.log(x); // Output: 15
18
19  var y = 50;
20  y = y * 2; // Can be written as y *= 2;
21  console.log(y); // Output: 100
22
23  var z = 8;
24  z /= 2; // Can be written as z = z / 2
25  console.log(z); // Output: 4
```

# Let's practice

Start with the following code:

```
9  <script type="text/javascript">
10 var levelUp = 10;
11 var powerLevel = 9001;
12 var multiplyMe = 32;
13 var quarterMe = 1152;
14
15 console.log('The value of levelUp:', levelUp);
16 console.log('The value of powerLevel:', powerLevel);
17 console.log('The value of multiplyMe:', multiplyMe);
18 console.log('The value of quarterMe:', quarterMe);
19 </script>
```

# Do the following:

- Use the `+` mathematical assignment operator to increase the value stored in `levelUp` by `5`.
- Use the `-` mathematical assignment operator to decrease the value stored in `powerLevel` by `100`.
- Use the `*` mathematical assignment operator to multiply the value stored in `multiplyMe` by `11`.
- Use the `/` mathematical assignment operator to divide the value stored in `quarterMe` by `4`.
- Print out the new values to the console.

# Increment and Decrement Operators

Other mathematical assignment operators include the *increment operator* (++) and *decrement operator* (--).

The increment operator will increase the value of the variable by 1. The decrement operator will decrease the value of the variable by 1. For example:

```
20  var a = 10;  
21  a++;  
22  console.log(a); // Output: 11  
23  
24  var b = 20;  
25  b--;  
26  console.log(b); // Output: 19
```

Just like the previous mathematical assignment operators (+=, -=, \*=, /=), the variable's value is updated *and* assigned as the new value of that variable.

# Your turn again!

Add the following variables to your code:

```
var gainedDollar = 3;  
var lostDollar = 50;
```

Using the increment operator, increase the value of `gainedDollar`.

Using the decrement operator, decrease the value of `lostDollar`.

Print out their new values to the console.

# String Concatenation with Variables

In previous exercises, we assigned strings to variables. Now, let's go over how to connect, or concatenate, strings in variables.

The `+` operator can be used to combine two string values even if those values are being stored in variables:

```
1 var myPet = "armadillo";
2 console.log('I own a pet ' + myPet);
```

In the example above, we assigned 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 concatenation to the console as: `I own a pet armadillo`

Create a variable named `favoriteAnimal` and set it equal to your favorite animal

Use `console.log()` to print `'My favorite animal: ANIMAL'` to the console. Use string concatenation so that `ANIMAL` is replaced with the value in your `favoriteAnimal` variable

# typeof Operator

While writing code, it can be useful to keep track of the data types of the variables in your program. If you need to check the data type of a variable's value, you can use the `typeof` operator.

The `typeof` operator checks the value to its right and *returns*, or passes back, a string of the data type.

```
const unknown1 = 'foo';
console.log(typeof unknown1); // Output:
string

const unknown2 = 10;
console.log(typeof unknown2); // Output:
number

const unknown3 = true;
console.log(typeof unknown3); // Output:
boolean
```

# Practice checking types

Create a new variable:

```
var newVariable = 'Experimenting with typeof';
```

Use `console.log()` to print the `typeof newVariable`.

Below the `console.log()` statement, reassign `newVariable` to 1.

Since you assigned this new value to `newVariable`, it has a new type! On the line below your reassignment, use `console.log()` to print `typeof newVariable` again.

# **Review - your turn!**

Let's review what we've discussed about variables and arithmetic operations:

On the following slide is code that uses button clicks to change the value of variables that affect the properties of size and color.

We'll go over it together, and then you'll need to complete its functionality.

```
variables_review.html ●
```

```
1 <!DOCTYPE html>
2 <html>
3 <body>
4
5 <p id="myP">JavaScript!</p>
6
7 <button type="button" id="increment">Increment</button>
8 <button type="button" id="red">Red</button>
9
10 <script>
11
12 var a;
13 a = 75;
14 var r;
15 r = 0;
16
17
18 var increment = document.getElementById("increment");
19 var red = document.getElementById("red");
20
21 document.getElementById("myP").style.fontSize = a+"px";
22
23 increment.addEventListener("click", function(){
24
25     a = a +1;
26     document.getElementById("myP").style.fontSize = a+"px";
27     console.log(a);
28
29 });
30
31 red.addEventListener("click", function(){
32
33     r = r+10;
34     document.getElementById("myP").style.color = "rgb"+("+"+r+",0,0)");
35
36 });
37
38
39 </script>
40
41 </body>
42 </html>
```

In the html structure of this code we have a paragraph with some text: “JavaScript!” and we create two buttons and give them unique ID’s so we can target them in JS.

In the JavaScript, we create variables for the font-size (a) and red color (r)

Then, we create variables to target each button.

We use JavaScript’s addEventListener to function to bind a function to the appropriate button-click.

In the increment function, we increase the value of by 1, increasing the size.

In the red function, we increase the value of r by 10, increasing the red value of rgb.

We use string concatenation with variables to pass the values to the html structure.

# Your turn!

- Create a button and accompanying function that does the opposite of increment. Clicking this button should shrink the text.
- Create a button for “green” and “blue” to affect the values of those colors. You can refer to the setup for “red” as a reference.

# Conditional Statements



# What are conditional statements?

In life, we make decisions based on circumstances. Think of an everyday decision as mundane as falling asleep— if we are tired, we go to bed, otherwise, we wake up and start our day.

These if-else decisions can be modeled in code by creating *conditional statements*. A conditional statement checks specific condition(s) and performs a task based on the condition(s).

Let's explore how programs make decisions by evaluating conditions and introduce logic into our code!

# Types of Conditionals in JavaScript

In JavaScript we have the following conditional statements:

- Use `if` to specify a block of code to be executed, if a specified condition is true
- Use `else` to specify a block of code to be executed, if the same condition is false
- Use `else if` to specify a new condition to test, if the first condition is false
- Use `switch` to specify many alternative blocks of code to be executed

# If statements

We often perform a task based on a condition. For example, if the weather is nice today, then we will go outside. If the alarm clock rings, then we'll shut it off. If we're tired, then we'll go to sleep. In programming, we can also perform a task based on a condition using an `if` statement:

```
if (true) {  
    console.log('This message will  
print!');  
}  
// Prints "This message will print!"
```

- The `if` keyword followed by a set of parentheses `()` which is followed by a *code block*, or *block statement*, indicated by a set of curly braces `{}`.
- Inside the parentheses `()`, a condition is provided that evaluates to `true` or `false`.
- If the condition evaluates to `true`, the code inside the curly braces `{}` runs, or executes.
- If the condition evaluates to `false`, the block won't execute.

# Let's make an “if” statement

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>conditionals</title>
5  </head>
6  <body>
7  <script type="text/javascript">
8
9
10 </script>
11 </body>
12 </html>
```

In the <script> section of a new .html file:

Declare a variable named `sale`. Assign the value `true` to it.

Now create an `if` statement. Provide the `if` statement a condition of `sale`. Inside the code block of the `if` statement, `console.log()` the string `'Time to buy!'`.

Run the code

Notice that the code inside the `if` statement ran, since `'Time to buy!'` was logged to the console. reassigned `sale` to `false`. Run your code and observe what happens.

# Browser-based example

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>HTML div</title>
5 <style type="text/css">
6   #div1{
7     width: 100px;
8     float:left;
9     height:100px;
10    background:red;
11    margin:10px;
12  }
13 </style>
14 </head>
15 <body>
16
17 <div id="div1">
18 First DIV
19 </div>
20
21 <script type="text/javascript">
22
23
24 </script>
25 </body>
26 </html>
```

Here's some code that draws a <div> as red box onto a browser page.

In the <script> section, create a variable called "color" and assign it the value of true.

Write an if statement that changes the box to green if color is true.

You'll need to use this within the code block section of your statement:

```
document.getElementById("div1").style.background = "green";
```

# if . . . else statements

In the previous exercises, we used an `if` statement that checked a condition to decide whether or not to run a block of code. If the condition evaluated to `false`, the program just didn't run the code.

But, in many cases, we'll have code we want to run if our condition evaluates to `false`. If we wanted to add some default behavior to the `if` statement, we can add an `else` statement to run a block of code when the condition evaluates to `false`. Take a look at the inclusion of an `else` statement:

```
if (false) {  
  console.log('The code in this block  
will not run.');
```

```
} else {  
  console.log('But the code in this block  
will!');
```

```
}
```

```
// Prints "But the code in this block  
will!"
```

An `else` statement must be paired with an `if` statement, and together they are referred to as an `if...else` statement. The code inside the `else` statement code block will execute when the `if` statement's condition evaluates to `false`. `if...else` statements allow us to automate solutions to yes-or-no questions, also known as *binary decisions*.

# Your turn:

Modify your first “if” statement, the one that outputs time to buy if sale=true:

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>conditionals</title>
5  </head>
6  <body>
7
8  <script type="text/javascript">
9  var sale = true;
10
11 if(sale){
12     console.log('Time to buy!');
13 }
14
15 </script>
16
17 </body>
18 </html>
```

Set sale to false.

Then, add an `else` statement to the existing `if` statement. Inside the code block of the `else` statement, `console.log()` the string 'Time to wait for a sale.'

See what happens as sale changes from true to false.

# Browser-based example

Here's our code that changes the background color of a div to green if color = true.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>HTML div</title>
5 <style type="text/css">
6   #div1{
7     width: 100px;
8     float:left;
9     height:100px;
10    background:red;
11    margin:10px;
12  }
13 </style>
14 </head>
15 <body>
16
17 <div id="div1">
18 First DIV
19 </div>
20
21 <script type="text/javascript">
22   var color = true;
23
24   if(color){
25
26     document.getElementById("div1").style.background = "green";
27   }
28
29 </script>
30 </body>
31 </html>
```

Add an “else” statement that changes the color to blue whenever color= false;

Change color to equal false to test your code.

# Operators

# Comparison Operators

When writing conditional statements, sometimes we need to use different types of operators to compare values. These operators are called *comparison operators*.

Here is a list of some handy comparison operators and their syntax:

- Less than: <
- Greater than: >
- Less than or equal to: <=
- Greater than or equal to: >=
- Is equal to: == / === (identity operator)
- Is NOT equal to: !=

Comparison operators compare the value on the left with the value on the right. For instance:

```
10 < 12 // Evaluates to true
```

It can be helpful to think of comparison statements as questions. When the answer is "yes", the statement evaluates to `true`, and when the answer is "no", the statement evaluates to `false`. The code above would be asking: is 10 less than 12? Yes! So `10 < 12` evaluates to `true`.

We can also use comparison operators on different data types like strings:

```
'apples' === 'oranges' // false
```

In the example above, we're using the *identity operator* (`==`) to check if the string `'apples'` is the same as the string `'oranges'`. Since the two strings are not the same, the comparison statement evaluates to `false`.

All comparison statements evaluate to either `true` or `false` and are made up of:

- Two values that will be compared.
- An operator that separates the values and compares them accordingly (`>`, `<`, `<=`, `>=`, `==`, `!=`).

# Let's Practice

Using `var`, create a variable named `hungerLevel` and set it equal to `7`.

Write an `if...else` statement using a comparison operator. The condition should check if `hungerLevel` is greater than `7`. If so, the conditional statement should log, `'Time to eat!'`. Otherwise, it should log `'We can eat later!'`.

# Browser-based example

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>HTML div</title>
5  <style type="text/css">
6      #div1{
7          width: 100px;
8          float:left;
9          height:100px;
10         margin:10px;
11         border-style: solid;
12     }
13 </style>
14 </head>
15 <body>
16
17 Enter a number: <input type="text" id="myText">
18 <button onclick="myFunction()">submit</button>
19
20
21 <div id="div1">
22 First DIV
23 </div>
24
25 <script type="text/javascript">
26     //var color = 9;
27 var color;
28
29 function myFunction() {
30     color = document.getElementById("myText").value;
31     console.log(color);
32 }
33
34
35 </script>
36 </body>
37 </html>
```

Here's our previous code slightly modified to accept input from an html text field.

Notice when we submit the number entered by the user, we assign that value to our color variable.

Modify this code so that the square will be red if a number  $> 50$  is entered and green if a number  $< 50$  is entered. If the number  $\ == 50$  the square will be blue.

Your if statements should go within the function brackets.

# Logical Operators

Logical operators are often used in conditional statements to add another layer of logic to our code.

# Logical Operators

We can use logical operators to add more sophisticated logic to our conditionals. There are three logical operators:

- the *and* operator (`&&`)
- the *or* operator (`||`)
- the *not* operator, otherwise known as the *bang* operator (`!`)

# && (logical and)

When we use the `&&` operator, we are checking that two things are `true`:

```
if (stopLight === 'green' && pedestrians
    === 0) {
  console.log('Go!');
} else {
  console.log('Stop');
}
```

When using the `&&` operator, both conditions *must* evaluate to `true` for the entire condition to evaluate to `true` and execute. Otherwise, if either condition is `false`, the `&&` condition will evaluate to `false` and the `else` block will execute.

# || (logical or)

When using the `||` operator, only one of the conditions must evaluate to `true` for the overall statement to evaluate to `true`

```
if (day === 'Saturday' || day ===  
    'Sunday') {  
    console.log('Enjoy the weekend!');  
} else {  
    console.log('Do some work.');//  
}
```

In the code example above, if either `day === 'Saturday'` or `day === 'Sunday'` evaluates to `true` the `if`'s condition will evaluate to `true` and its code block will execute. If the first condition in an `||` statement evaluates to `true`, the second condition won't even be checked. Only if `day === 'Saturday'` evaluates to `false` will `day === 'Sunday'` be evaluated. The code in the `else` statement above will execute only if both comparisons evaluate to `false`.

# ! (not!)

The `!` *not operator* reverses, or *negates*, the value of an operator.:

```
let excited = true;  
console.log(!excited); // Prints false
```

```
let sleepy = false;  
console.log(!sleepy); // Prints true
```

Essentially, the `!` operator will either take a `true` value and pass back `false`, or it will take a `false` value and pass back `true`.

# Practice!

Create two variables:

```
var mood = sleepy;
```

```
var tirednessLevel = 6;
```

Let's create an `if...else` statement that checks if `mood` is `'sleepy'` and `tirednessLevel` is greater than `8`.

If both conditions evaluate to `true`, then `console.log()` the string `'time to sleep'`. Otherwise, we should `console.log()` `'not bed time yet'`.

# Browser-based example

Let's modify our box-changing-color code to make the logic more sophisticated by using logical operators.

Modify the if statements so that:

When a number  $\geq 0$  and  $\leq 25$  is entered the box will be red.

When a number  $> 25$  and  $\leq 50$  is entered the box will be green.

When a number  $> 50$  and  $\leq 75$  is entered the box will be blue

When a number  $> 75$  and  $\leq 100$  is entered the box will be yellow

When the string 'purple' or 'Purple' is entered the box will be purple

When the string 'orange' or 'pumpkin' is entered the box will be orange

# Else If Statement

The `else if` statement allows for more than two possible outcomes. You can add as many `else if` statements as you'd like, to make more complex conditionals!

The `else if` statement always comes after the `if` statement and before the `else` statement. The `else if` statement also takes a condition. Let's take a look at the syntax:

```
let stopLight = 'yellow';

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!');
}
```

The `else if` statements allow you to have multiple possible outcomes. `if/else if/else` statements are read from top to bottom, so the first condition that evaluates to `true` from the top to bottom is the block that gets executed.

In the example above, since `stopLight === 'red'` evaluates to `false` and `stopLight === 'yellow'` evaluates to `true`, the code inside the first `else if` statement is executed. The rest of the conditions are not evaluated. If none of the conditions evaluated to `true`, then the code in the `else` statement would have executed.

# Practice!

Create a var called “season” and assign it a starting value of “summer”.

Let’s write an if statement to check if it’s Spring:

```
var season = 'summer';

if (season === 'spring') {
  console.log('It\'s spring! The trees are budding!');
}
```

add an `else if` statement, and a `console.log()` that prints the string `'It's winter! Everything is covered in snow.'`. If `season = "winter"`.

Add another `else if` statement that checks if `season` is equal to `'fall'`.

Inside the code block of the `else if` statement you just created, add a `console.log()` that prints the string `'It's fall! Leaves are falling!'`.

Add an `else if` statement that checks if `season` is equal to `'summer'`.

Inside the code block of the `else if` statement you just created, add a `console.log()` that prints the string `'It's sunny and warm because it's summer!'`.

And finally, add an `else` statement if to `console.log ('Invalid entry')` if `season` isn't spring, summer, winter, or fall.

# Browser Practice!

Take your else if code and modify it so that instead of just a console.log message, a seasonal picture appears when the variable season is changed.

Hint: you'll need to create a div in the body of your html and get the element by ID. You can use the innerHTML method to display the image.

# Switch

A `switch` statement provides an alternative syntax to `else if` that is easier to read and write. A `switch` statement looks 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;
}

// Prints 'Papayas are $1.29'
```

- The `switch` keyword initiates the statement and is followed by `( ... )`, which contains the value that each `case` will compare. In the example, the value or expression of the `switch` statement is `groceryItem`.
- Inside the block, `{ ... }`, there are multiple `cases`. The `case` keyword checks if the expression matches the specified value that comes after it. The value following the first `case` is `'tomato'`. If the value of `groceryItem` equalled `'tomato'`, that `case`'s `console.log()` would run.
- The value of `groceryItem` is `'papaya'`, so the third `case` runs—`Papayas are $1.29`is logged to the console.
- The `break` keyword tells the computer to exit the block and not execute any more code or check any other cases inside the code block. Note: Without the `break` keyword at the end of each `case`, the program would execute the code for all matching cases and the `default` code as well. This behavior is different from `if/else` conditional statements which execute only one block of code.
- At the end of each `switch` statement, there is a `default` statement. If none of the `cases` are true, then the code in the `default` statement will run.

# Practice!

Let's write a `switch` statement to decide what medal to award an athlete.

Create a `var athleteFinalPosition;`

start by writing a `switch` statement with `athleteFinalPosition` as its expression.

Inside the `switch` statement, add three `cases`:

- The first `case` checks for the value '`first place`'
  - If the expression's value matches the value of the `case` then `console.log()` the string '`You get the gold medal!`'
- The second `case` checks for the value '`second place`'
  - If the expression's value matches the value of the `case` then `console.log()` the string '`You get the silver medal!`'
- The third `case` checks for the value '`third place`'
  - If the expression's value matches the value of the `case` then `console.log()` the string '`You get the bronze medal!`'

Remember to add a `break` after each `console.log()`.

Now, add a `default` statement at the end of the `switch` that uses `console.log()` to print '`No medal awarded.`'. If `athleteFinalPosition` does not equal any value of our `cases`, then the string '`No medal awarded.`' is logged to the console.

Remember to add the `break` keyword at the end of the `default` case.

# Browser Practice

Write another version of your seasons code that uses a switch statement instead of else if.