**Discover the length**

What would the length of your name be?

To discover the length of your name write your name within quotes. Then write a period (full stop) and the word **length** like this:

"yourName".length

Ex: Matt.length would output 4

**Basic math**

Now, let's do some math. You can do math through programming!

Add any two numbers, like this: 3 + 4

The output would be 7

**Numbers and more**

See what happened? You can use the command line to do basic math operations. Try playing around some more.

You can use \* for multiplication and / for division if you want. Enter another valid expression to pass this lesson.

**Script:**

3 \* 8

**Output:** 24

**Error: does not compute!**

There are some things you *can't* do in the console. Computers only speak certain languages, like the one you've been using today: JavaScript!

If you use words that aren't in the JavaScript language, it will get confused and give you an error.

Try to confuse the interpreter by using a word it doesn't know, like eggplant. It will give you a ReferenceError.

Well done. You have confused the computer!

**Script:**

eggplant

**Output:**

ReferenceError: 'eggplant' is undefined

**Editor and comments**

So far we've been writing lines of code in the editor. Now we see two lines that start with //.

The // sign is for **comments**. A comment is a line of text that JavaScript won't try to run as code. It's just for humans to read.

Comments make your program easier to understand. When you look back at your code or others want to collaborate with you, they can read your comments and easily figure out what your code does.

**Script:**

// This is a comment that the computer will ignore.

// It is for your eyes only!

"cake".length

The computer will ignore the code on [lines 1-2](javascript:void(0)), since it is commented out.

On [line 3](javascript:void(0)), find the length of the word "cake".

**What am I learning?**

This is JavaScript (JS), a programming language. There are many languages, but JS has many uses and is easy to learn.

**What can we use JavaScript for?**

* make websites respond to user interaction
* build apps and games (*e.g.* blackjack)
* access information on the Internet (*e.g.* find out the top trending words on Twitter by topic)
* organize and present data (*e.g.* automate spreadsheet work; data visualization)

**Interactive JavaScript**

What we just saw was a fun example of how JavaScript can be interactive. Try it yourself!

**Examples**:

**Script:**

confirm("I feel awesome!");  
confirm("I am ready to go.");

These boxes can be used on websites to *confirm* things with users. You've probably seen them pop up when you try to delete important things or leave a website with unsaved changes.

Write your own message that you want the user to confirm.

**Script:**

// Also try the Q&A forum to get help

// The link is up at the top!

confirm("I feel awesome!");

confirm("I am ready to go.");

**What is programming?**

Programming is like writing a list of instructions to the computer so it can do cool stuff with your information.

Programs can't yet make your bed, but they can do math, keep track of your bank account, or send a message to a friend.

To do any of these actions, the program needs an input. You can ask for input with a **prompt**.

**Examples**:  
  
1. prompt("What is your name?");  
2. prompt("What is Ubuntu?");

**Script:**

prompt("What is your Name?");

prompt("What is Ubuntu?");

**Data Types I & II: Numbers & Strings**

Data comes in various **types**. You have used two already!

**a. numbers** are quantities, just like you're used to. You can do math with them.

**b. strings** are sequences of characters, like the letters a-z, spaces, and even numbers. These are all strings: "Ryan", "4" and "What is your name?" Strings are extremely useful as labels, names, and content for your programs.

To make a *number* in your code, just write a number as numerals *without quotes*: 42, 190.12334.

To write a string, surround words *with quotes*: "What is your name?"

1. Write a string with at least 3 words. Check out the examples of strings above.
2. Find the length of the string by writing a period (full stop) and the word length, like this:
3. "string".length

Length counts every character in the string - including spaces!

To check the length of something, we type "string".length

Remember a string might not always be a word—you can put almost any character in between quotes to make a string.

**Script:**

"Hello there you".length;

**Output:** 15

**Data Type III: Booleans**

Nice job! Next let's look at **booleans**. A boolean is either true or false.

For example, comparing two numbers returns a true or false result:

* 23 > 10 is true
* 5 < 4 is false

Let's compare two numbers to return a true result:

1. First, write the string "I'm coding like a champ"
2. Next, find the length of the string using .length
3. Then, compare the string's length to see if it is greater than 10

If you want to check your code, click "Stuck? Get a hint!" below.

You can use .length to find the number of characters in your string, like this:

"string".length;

Your code should look something like this:

**Script:**

"I'm coding like a champ!".length > 10;

"I'm coding like a champ!".length > 10;

**Output:** true

**Using console.log**

You may have noticed that the interpreter doesn't print out every single thing it does. So if we want to know what it's thinking, we sometimes have to ask it to speak to us.

console.log() will take whatever is inside the parentheses and log it to the console below your code—that's why it's called console.log()!

This is commonly called **printing out**.

Please print the following two console.log statements at the same time. Type one on [line 1](javascript:void(0)) and the other on [line 2](javascript:void(0)). Then press Save & Submit Code.

console.log(2 \* 5)  
console.log("Hello")

1. Make sure to include quotes for strings, and no quotes for numbers.
2. Check your parentheses carefully.
3. Make sure you are running two console.log statements **at the same time**.

On [line 1](javascript:void(0)), type out the first statement. Then on [line 2](javascript:void(0)), type out the second statement. Then press "Save & Submit Code"!

console.log(2\*5);

console.log("Hello");

output: true

**Comparisons**

So far we've learned about three data types:

* **strings** (*e.g.* "dogs go woof!")
* **numbers** (*e.g.* 4, 10)
* **booleans** (*e.g.* false, 5 > 4)

Now let's learn more about comparison operators.

**List of comparison operators**:

* > Greater than
* < Less than
* <= Less than or equal to
* >= Greater than or equal to
* === Equal to
* !== **Not** equal to

Try to use each of the operators above.

1. Choose the correct comparison operator to make each of the four statements print out true.

When you use the .length, the value that is returned is a **number**. Therefore, when you enter something like "Jenny".length === 5;, the computer is evaluating whether the number of letters in Jenny is equal to 5.

**Script:**

//After you run the code, you should see true 4 times, and false 2 times.

//By using `console.log` at the start of each line,

//we are able to print 6 lines of output.

console.log(15 > 4);

console.log("Xiao Hui".length < 122);

console.log("Goody Donaldson".length > 8);

console.log(8\*2 === 16);

console.log(4<=1);

console.log("Hello There".length <= 5);

**Output:**

true

true

true

true

false

false

**Decisions, decisions**

Nice work on comparisons! Now let's see how we can use comparisons to ask yes or no questions.

Say we want to write a program that asks whether your name is longer than 7 letters. If the answer is yes, we can respond with "You have a long name!" We can do this with an if statement:

if( "myName".length >= 7 ) {

console.log("You have a long name!");

}

An if statement is made up of the if keyword, a condition like we've seen before, and a pair of curly braces { }. If the answer to the condition is yes, the code inside the curly braces will run.

Check out the if statement in the editor.

1. On [line 1](javascript:void(0)), add a condition inside the parentheses ( ).
2. If the answer to the condition is yes, the code inside the curly braces will run. So on [line 2](javascript:void(0)), use console.log to print out a string.

You can write a condition with a comparison like 10 > 5:

if (10 > 5) {

// This part will run, since 10 > 5

}

**Script:**

if ( 100 > 2 )

{

console.log("You are good at math!");

}

console.log("Just letting you know: your program got to line 6");

**Output:**

You are good at math!

Just letting you know: your program got to line 6

**Computers are smart**

Great! We used an if statement to do something if the answer to the condition was yes, or true as we say in JavaScript.

In addition to doing something when the condition is true, we can do something else if the condition is false. For example, if your name is shorter than 7 letters, we can respond with "You have a short name!" We can do this using an if / else statement:

if( "myName".length >= 7 ) {

console.log("You have a long name!");

}

else {

console.log("You have a short name!");

}

Just like before, *if* the condition is true, then only the code inside the first pair of curly braces will run. *Otherwise*, the condition is false, so only the code inside the second pair of curly braces after the else keyword will run.

In the example above the condition "myName".length >= 7 evaluates to false since "myName" only has 6 letters. Since the condition is false, only the code inside the curly braces after the else keyword runs, and prints out You have a short name!.

1. In [line 1](javascript:void(0)), fill in a condition that will evaluate to false
2. Fill in some code to run in the else portion (this will run if the condition is false). Use console.log for this part.

You can write a condition with an inequality like 10 < 5:

**Script:**

if (10 < 5) {

// This won't run, since 10 > 5

}

else {

// This part WILL run!

}

if (5>2)

{

console.log("Let's go down the first road!");

}

else

{

// what should we do if the condition is wrong? fill in here:

console.log("Try again!");

}

**Output:** Let's **go down the first road!**

**More practice with conditionals**

Now let's practice using if/else statements. Do as much as you can by yourself, but if you need a reminder, click the "Stuck? Get a hint!" button below.

1. Write an if/else statement, just like we did in the last exercise. Here's what the outline of the code looked like:

if (condition)

{

// if condition is true

// do this code

}

else // "otherwise"

{

// do this code instead

}

1. If your condition is true, use console.log to print "The condition is true".
2. Otherwise (else) when it is false, use console.log to print "The condition is false".
3. Make sure your condition evaluates to false, so that your program prints out "The condition is false".

Your code could look something like this

if( 10 < 3 )

{

console.log("The condition is true");

}

else

{

console.log("The condition is false");

}

**Script:**

// Remember, the order and punctuation matter.

// If you get an error, check carefully, line by line

// If you're really stuck, see the hint!

if ("Matt".length > 100) {

console.log("The condition is true");

} else {

console.log("The condition is false");

}

**Output:** The condition is false

**Computers aren't that smart**

Well done! Now, computers are very literal. Syntax needs to be in exactly the right place for the computer to understand the code.

As you get started with programming, we will teach you many syntax rules. This is sort of like the grammar of programming languages. Grammar first, then programming poetry!

There are many mistakes in this code. Find them and fix them all.

You are doing what's called "debugging," a term popularized by [Grace Hopper](http://en.wikipedia.org/wiki/Grace_Hopper) when she literally [removed a moth](http://en.wikipedia.org/wiki/File:H96566k.jpg) from her computer.

Remember, if / else statements look like this:

if (condition) {

code;

} else {

code;

}

Make sure the condition is surrounded by parentheses, and the code is surrounded by curly braces!

**Script:**

// The computer doesn't worry about extra spaces between words or brackets

// It just cares about the order of where things are placed

// and that you have used the right characters (){}[]"";

// For extra help, a program called a 'linter' is checking your code

// and will put a red 'x' next to the first line that contains errors

if (10 === 10) {

console.log("You got a true!");

} else {

console.log("You got a false!");

}

**Output:** You got a true!

**Mid-lesson breather**

We've covered a lot of ground so far! So many new terms, so much syntax. Let's take a breather and review. We have learned:

**1. Confirm and prompt**  
  
We can make pop-up boxes appear!   
confirm("I am ok");  
prompt("Are you ok?");

**2. Data types**  
  
a. numbers (*e.g.* 4.3, 134)  
  
b. strings (*e.g.* "dogs go woof!", "JavaScript expert")  
  
c. booleans (*e.g.* false, 5 > 4)

**3. Conditionals**  
  
*If* the first condition is met, execute the first code block. *If* it is not met, execute the code in the else block. See the code on the right for another example.

**Script:**

//This is an example of an if / else statement.

if(12 / 4 === "Ari".length){

confirm("Will this run the first block?");

} else {

confirm("Or the second block?");

}

**Output:** (a confirm prompt) true

**Math**

We saw basic math before. The basic math symbols we learned in school work here. Even the order in which the computer understands the math is the same as in school!

**Code**:  
  
1. ( ): control order of operations  
2. \* and /: multiplication and division  
3. - and +: subtraction and addition

**Examples**:  
  
1. 100/10 evaluates to 10  
2. "Jane".length + 5 evaluates to 9  
3. 5\*(3+1) evaluates to 20

1. Complete the missing bits of code to construct the if / else statement. Make the condition evaluate to true.
2. Finish the else statement by printing out the string "Error Error Error" to the console.

There is a lot of left out syntax. The if keyword and the { } after the else keyword are all left out.

Make sure not to delete any of the provided code. It could cause your code to fail!

**Script:**

if ("Jon".length \* 2 / (2+1) === 2)

{

console.log("The answer makes sense!");

}

else

{

console.log("Error, error, error!");

}

**Output:** The answer makes sense!

**Math and the modulo**

Let's meet an interesting symbol called **modulo**. When % is placed between two numbers, the computer will divide the first number by the second, and then return the **remainder** of that division.

So if we do 23 % 10, we divide 23 by 10 which equals 2 with 3 left over. So 23 % 10 evaluates to 3.

**More examples**:  
  
17 % 5 evaluates to 2  
  
13 % 7 evaluates to 6

Use console.log and modulo three times to print the remainder of the following equations:

a. 14 / 3  
b. 99 / 8  
c. 11 / 3

Please type out console.log three times on three lines, filling in the parentheses each time with a new statement. Don't forget to use % and not / to get the remainder!

**Script:**

// Below is an example of printing the remainder of 18/4 using modulo

// console.log(18 % 4);

console.log(14%3);

console.log(99%8);

console.log(11%3);

console.log(8%2);

**Output:**

2

3

2

0

**Modulo and if / else**

So why learn modulo? For one thing, it's good at testing divisibility. Consider 30 % 10. What does it return? There is nothing left over, so 0.

How about 9 % 3? Also 0.

We can use modulos in comparisons, like this:

* 10 % 2 === 0 evaluates to true
* 7 % 3 === 0 evaluates to false because there is 1 left over.

Let's get the if/else" statement to display "The first number is even".

1. Edit [line 5](javascript:void(0)) by adding a comparison that evaluates to true.
2. In the comparison, use a modulo and an even number, like we did in the example above.

Even numbers are evenly divisible by 2. So, for example, 10 % 2 === 0 should work!

Make sure not to add any semicolons to your code. There should **not** be a semicolon between the ) and { in your if statement!

**Script:**

//An example of an if/else statement with modulo in the condition

if( 10 % 3 ) {

console.log("The first number is even!");

console.log(10%3);

} else {

console.log("The first number is odd!");

}

**Output:**

The first number is even!

1

**Substrings**

We've learned a few ways to manipulate numbers. What about manipulating strings?

Sometimes you don't want to display the *entire* string, just a part of it. For example, in your Gmail inbox, you can set it to display the first 50 or so characters of each message so you can preview them. This preview is a *substring* of the original string (the entire message).

**Code**:  
  
"some word".substring(x, y) where x is where you start chopping and y is where you finish chopping the original string.

The number part is a little strange. To select for the "he" in "hello", you would write this:

"hello". substring(0, 2);

Each character in a string is numbered starting from 0, like this:

0 1 2 3 4

| | | | |

h e l l o

The letter h is in position 0, the letter e is in position 1, and so on.

Therefore if you start at position 0, and slice right up till position 2, you are left with just he

**More examples**:  
  
1. First 3 letters of "Batman"  
"Batman".substring(0,3);  
  
2. From 4th to 6th letter of "laptop"  
"laptop".substring(3,6);

Find the 4th up to and including the 7th letter of the string "wonderful day".

Remember to start counting from 0. In other words, the 1st letter starts at position 0, the 2nd letter starts at position 1, and so on.

There is no need to use console.log as you are not asked to print it out!

**Script:**

// Be careful with the substring's letter positions

"wonderful day".substring(3,7);

**Output:** "derf"

**More substring practice**

Getting the positioning of substring letter positions is tricky! Let's make sure we really have it nailed down.

Remember that each character in a string is numbered starting from 0. So for the word "hello", The letter h is in position 0, the letter e is in position 1, and so on.

Using console.log, on three separate lines, print out the substrings for the following strings.

a. "Jan" in "January"  
b. "Melbourne is" in "Melbourne is great" (note the space!)  
c. "burgers" in "Hamburgers"

Here's how to print the first 2 letters of the world "hello":

console.log( "hello".substring(0, 2) );

**Script:**

// Use console.log( ) to print out the substrings.

// Here is an example of the 1st to 4th letter of "JavaScript"

// console.log("Javascript".substring(0,4));

console.log("January".substring(0,3));

console.log("Melbourne is great".substring(0,12));

console.log("Hamburgers".substring(3,10));

**Output:**

Jan

Melbourne is

burgers

**Variables**

We have learned how to do a few things now: make strings, find the length of strings, find what character is in the nth position, do basic math. Not bad for a day's work!

To do more complex coding, we need a way to 'save' the values from our coding. We do this by defining a variable with a specific, case-sensitive name. Once you create (or **declare**) a variable as having a particular name, you can then call up that value by typing the variable name.

**Code**:  
  
var varName = data;

**Example**:  
  
a. var myName = "Leng";  
b. var myAge = 30;  
c. var isOdd = true;

Create a variable called myAge and type in your age.

Remember to not use quotes, or your age will turn into a string. To declare a variable, you only need to type:

var variableName = /\* some value \*/

**Script:**

// To create a variable, we use only one equals sign

// But to check if two values are equal, we use 3 equal signs.

// declare your variable here:

var myAge=42;

console.log(myAge);

**Output:** 42

**More Variable Practice**

We have seen how to create a variable. But how do we use it? It is useful to think that any time you type the variable's name, you are asking the computer to swap out the variable name and swap in the *value* of the variable.

**For example:**  
  
var myName = "Steve Jobs";  
  
myName.substring(0,5)

Look at the second line above. You have asked the computer to swap out myName and swap in Steve Jobs, so

myName.substring(0,5)

becomes

"Steve Jobs".substring(0,5)

which evaluates to Steve.

**Another example**

var myAge = 120;

What is

myAge % 12 ? See the hint to check your answer.

So the variable stores the *value* of the variable, whether that is a number or a string. As you will see soon, this makes writing long programs much easier!

Note that "New York" and "new york" are seen by the computer to be different strings. (i.e. "New York" is not the same as "new york")

And the answer from above is that myAge % 12 evaluates to 0.

**Script:**

// Declare a variable on line 2 called myCountry and give it a string value.

var myCountry = "America";

// Use console.log to print out the length of the variable myCountry

console.log(myCountry.length);

// Use console.log t print out the first three letters of myCountry

console.log(myCountry.substring(0,3));

**Output:**

7

Ame

**Change variable values**

So far, we've seen  
a. how to create a variable  
b. how to use a variable

Let's now see how to change a variable's value. A variable's value is easily changed. Just pretend you are creating a new variable while using the same name of the existing variable!

**Example**:  
  
var myAge = "Thirty";  
Say I had a birthday and I want to change my age.  
myAge = "Thirty-one";

Now the value of myAge is "Thirty-one"!

Follow the instructions on [line 1](javascript:void(0)), [line 3](javascript:void(0)), [line 5](javascript:void(0)) and [line 8](javascript:void(0)). We're using this method to show you the order in which you tell the computer what to do is very important.

To print out the myName variable, put myName between the parentheses after console.log.

**Script:**

//On line 2, declare a variable myName and give it your name.

var myName="Matt";

//On line 4, use console.log to print out the myName variable.

console.log(myName);

//On line 7, change the value of myName to be just the first 2

//letters of your name.

myName="Ma";

//On line 9, use console.log to print out the myName variable.

console.log(myName);

**Output:**

Matt

Ma

**Conclusion: Part 1**

Let's do a quick review!

**Data types**

* strings (e.g. "dogs go woof!")
* numbers (e.g. 4, 10)
* booleans (e.g. false, 5 > 4)

**Variables**  
We store data values in variables. We can bring back the values of these variables by typing the variable name.

**Manipulating numbers & strings**

* comparisons (e.g. >, <=)
* modulo (e.g. %)
* string length (e.g. "Emily".length;)
* substrings (e.g. "hi".substring(0, 1);)

**console.log( )**   
Prints into the console whatever we put in the parentheses.

1. On [line 1](javascript:void(0)), create a variable myColor and give it a string value.
2. On [line 2](javascript:void(0)), print the length of myColor to the console.

**Script:**

//On line 2, write your first comment! It can be anything!

//this is my comment

var myColor = "red";

console.log(myColor.length);

**Output:** 3

**Conclusion: Part 2**

Congratulations on making it this far. You have learned a lot! Just one more exercise before a big pat on the back!

The last tricky thing we learned was about **if / else** statements.

If / else statements are conditional statements. Under different conditions, the computer will output different things.

1. Write your own if / else statement.
2. The only instruction is that the result of evaluating the statement is a log to the console of "I finished my first course!".

There are three steps to an if / else statement.   
1. Work out what condition you want  
2. If the condition is true, work out what code you want to run.  
3. If it is false, work out the other code you want to run!

**Script:**

//Not sure where to begin? Check the Hint!

if (12+2 === 14) {

console.log("I finished my first course!");

} else {

console.log("hello");

}

**Output:** I finished my first course!

**Code Your Own Adventure!**

**Confirm**

You have some programming skills. Time to make something you can show people! We're going to show you how to program a 'code your own adventure' game. It'll have a basic story line, have the user make some choices, and have a happy ending. Then you can modify it as you wish and show off your creative talents!

One note before we begin. Each of the following exercises will ask you to add on to your previous code. Make sure not to delete or change anything that you did in a previous exercise. Good luck!

1. It's always polite to ask your user if they are ready to play.
2. Using the confirm command, make sure your user is ready to play. For example, I would use the sentence "I am ready to play!". Add whatever phrase you would like in the confirm.

To get the user to confirm something, the syntax looks like:

confirm("I understand confirm!");

**Script:**

//Check if the user is ready to play!

confirm("I am ready to play!");

**Output:**

Confirm prompt – true

**Old enough to play?**

This will involve two pieces of code:

1. First we will use prompt to ask a user for their age, like this:
2. var age = prompt("What's your age");

The variable age will hold the user's response.

1. Then we will use an if/else statement based on the age of the user. Here's an outline of the code, similar to what we've seen before:

if(age is less than 13)

{

// do this code

}

else // "otherwise"

{

// do this code

}

1. Under the existing code, declare a variable age.
2. Make age equal to prompt("What's your age");. See the example above.
3. Then write an if/else statement. *If* age is less than 13, use console.log to tell the user that they're allowed to play but you take no responsibility.
4. Else, use console.log and give them an encouraging message to play on!

Recall the syntax for if / else statements.

if (condition) {

// code code code

} else {

// code code code

}

**Script:**

//Check if the user is ready to play!

confirm("I am ready to play!");

var age = prompt("What's your age");

if (age < 13) {

console.log("You can play, but we're not responsible.");

} else {

console.log("Get your game on!!");

}

**Output:** age confirm both – “game on”

**Adding some story**

Let's set up the scene for your story. Under all your previous code, print out the following introduction, exactly as it is written.

"You are at a Justin Bieber concert, and you hear this lyric 'Lace my shoes off, start racing.'"

Print the introduction using console.log. Remember that the introduction is a **string**, so make sure to keep it between quotes.

When I was playing, I made a couple of errors!

1. Don't forget to use console.log to print stuff out.
2. The introduction is a string, so don't forget to write it between quotes.
3. Copy and paste the string exactly as it is written in the instructions.

**Script:**

//Check if the user is ready to play!

confirm("I am ready to play!");

var age = prompt("What's your age?");

if (age < 13) {

console.log("You can play, but we're not responsible.");

} else {

console.log("Get your game on!!");

}

console.log("You are at a Justin Bieber concert, and you hear this lyric 'Lace my shoes off, start racing.'");

**Output:** confirm Ready to Play, insert age and in console you’ll see:

Get your game on!!

You are at a Justin Bieber concert, and you hear this lyric 'Lace my shoes off, start racing.'

**First move!**

Next, your user is about to talk to Justin Bieber.

1. Under your existing code, print out the storyline: "Suddenly, Bieber stops and says, 'Who wants to race me?'"
2. Then declare a variable userAnswer. Make it equal a prompt that asks the user "Do you want to race Bieber on stage?". This will be the question that you ask your user.

**Script:**

//Check if the user is ready to play!

confirm("I am ready to play!");

var age = prompt("What's your age?");

if (age < 18) {

console.log("You can play, but we're not responsible.");

} else {

console.log("Get your game on!!");

}

console.log("You are at a Justin Bieber concert, and you hear this lyric 'Lace my shoes off, start racing.'");

console.log("Suddenly, Bieber stops and says, 'Who wants to race me?'");

var userAnswer = prompt("Do you want to race Bieber on stage?");

**Output:** confirm prompt, then age prompt, then message in console :

Get your game on!!

You are at a Justin Bieber concert, and you hear this lyric 'Lace my shoes off, start racing.'

Suddenly, Bieber stops and says, 'Who wants to race me?'

**The story heats up!**

Now you have to create different scenarios. Good thing we know how to do that using if / else statements.

If userAnswer is "yes", print out: "You and Bieber start racing. It's neck and neck! You win by a shoelace!"

Otherwise, print out: "Oh no! Bieber shakes his head and sings 'I set a pace, so I can race without pacing.'"

Remember: = is for assignment, and === is to check if things are equal!

Use an if / else statement to write out the last part of this game!

Make sure to check the console for error messages and suggestions for how to fix them.

Cut and paste the sentences that you have to print out to avoid typos. If you don't type things exactly right, things don't work!

Remember, if/else works like this:

if (condition) {

// Do something

} else { // Otherwise...

// Do something else!

}

**Script:**

//Check if the user is ready to play!

confirm("I am ready to play!");

var age = prompt("What's your age?");

if (age < 18) {

console.log("You can play, but we're not responsible.");

} else {

console.log("Get your game on!!");

}

console.log("You are at a Justin Bieber concert, and you hear this lyric 'Lace my shoes off, start racing.'");

console.log("Suddenly, Bieber stops and says, 'Who wants to race me?'");

var userAnswer = prompt("Do you want to race Bieber on stage?");

if (userAnswer === "yes"){

console.log("You and Bieber start racing. It's neck and neck! You win by a shoelace!");

} else {

console.log("Oh no! Bieber shakes his head and sings 'I set a pace, so I can race without pacing.'");

}

**Output:** confirm prompt, then age prompt, ask if you want to race prompt, then message in console –

Get your game on!!

You are at a Justin Bieber concert, and you hear this lyric 'Lace my shoes off, start racing.'

Suddenly, Bieber stops and says, 'Who wants to race me?'

You and Bieber start racing. It's neck and neck! You win by a shoelace!

**Asking for feedback**

It is worthwhile asking your user for feedback!

1. Create a variable called feedback and prompt the user to rate your game out of 10.
2. If feedback is greater than 8, print out: "Thank you! We should race at the next concert!"
3. Otherwise, print out:  
   "I'll keep practicing coding and racing."

Remember, your prompt should look something like this:

var feedback = prompt("Message");

**Script:**

//Check if the user is ready to play!

confirm("I am ready to play!");

var age = prompt("What's your age?");

if (age < 18) {

console.log("You can play, but we're not responsible.");

} else {

console.log("Get your game on!!");

}

console.log("You are at a Justin Bieber concert, and you hear this lyric 'Lace my shoes off, start racing.'");

console.log("Suddenly, Bieber stops and says, 'Who wants to race me?'");

var userAnswer = prompt("Do you want to race Bieber on stage?");

if (userAnswer === "yes"){

console.log("You and Bieber start racing. It's neck and neck! You win by a shoelace!");

} else {

console.log("Oh no! Bieber shakes his head and sings 'I set a pace, so I can race without pacing.'");

}

var feedback = prompt("Please provide a rating between 1 and 10.");

if (feedback > 8) {

console.log("Thank you! We should race at the next concert!");

} else {

console.log("I'll keep practicing coding and racing.");

}

**Output:** confirm prompt, then age prompt, ask if you want to race prompt, then a rating prompt, then message in console:

Get your game on!!

You are at a Justin Bieber concert, and you hear this lyric 'Lace my shoes off, start racing.'

Suddenly, Bieber stops and says, 'Who wants to race me?'

You and Bieber start racing. It's neck and neck! You win by a shoelace!

I'll keep practicing coding and racing.

Another example script of the above:

**Script:**

//Check if the user is ready to play!

confirm("I am ready to play!");

var age = prompt("What's your age?");

if (age < 18) {

console.log("You can play, but we're not responsible.");

} else {

console.log("Get your game on!!");

}

console.log("Snow White and Batman were hanging out at the bus stop, waiting to go to the shops. There was a sale on and both needed some new threads. You've never really liked Batman. You walk up to him.");

console.log("Batman glares at you.");

var userAnswer = prompt("Are you feeling lucky, punk?");

if (userAnswer === "yes"){

console.log("Batman hits you very hard. It's Batman and you're you! Of course Batman wins!");

} else {

console.log("You did not say yes to feeling lucky. Good choice! You are a winner in the game of not getting beaten up by Batman.");

}

var feedback = prompt("Please provide a rating between 1 and 10.");

if (feedback > 8) {

console.log("This is just the beginning of my game empire. Stay tuned for more!");

} else {

console.log("I slaved away at this game and you gave me that score?! The nerve! Just you wait!");

}

**Output:** prompts then message:

You can play, but we're not responsible.

Snow White and Batman were hanging out at the bus stop, waiting to go to the shops. There was a sale on and both needed some new threads. You've never really liked Batman. You walk up to him.

Batman glares at you.

You did not say yes to feeling lucky. Good choice! You are a winner in the game of not getting beaten up by Batman.

I slaved away at this game and you gave me that score?! The nerve! Just you wait!

**Introduction to Functions in JS**

**Introduction**

Programming is simply a way to give instructions to the computer.

In [Getting Started](http://www.codecademy.com/courses/getting-started-v2/2/3), we learned about if / else statements.

We want to keep learning ways to instruct the computer to perform repeatable tasks efficiently.

Let's briefly review! Use an if / else to check how fast you're driving.

1. *If* speed is greater than 80, use console.log to print "Slow down"
2. Otherwise (else), use console.log to print "Drive safe"

In this case "Drive safe" should be logged, since 65 is less than 80.

Remember, if/else works like this:

if (condition) {

// Do something

} else { // Otherwise...

// Do something else!

}

**Script:**

var speed = 55;

if (speed > 80) {

console.log("Slow Down");

} else {

console.log("Drive Safe");

}

**Output:** Drive Safe

**Introducing Functions**

Programming is similar to baking cakes. Seriously! Imagine you are trying to teach your friend Jane how to bake many different types of cakes.

Each cake takes in different ingredients (ie. **inputs**). But the 'bake' instructions are always the same. For example:

1. Pre-heat the oven at 300 degrees
2. Mix all the ingredients in a bowl
3. Put contents into oven for 30 mins

And the **output** will be a different cake each time.

It is tedious to have to repeat to Jane the same 'bake' instructions every time. What if we could just say 'bake' and Jane would know to execute those three steps? That is exactly what a function is!

1. [Line 3](javascript:void(0)) declares the function and gives it a name.
2. Focus on [line 4](javascript:void(0)) and [line 5](javascript:void(0)). The code within the curly brackets { } is the code we want to use again and again. (i.e. the 'bake' instructions)
3. [Line 4](javascript:void(0)) declares a variable called val. [Line 5](javascript:void(0)) prints the value of that variable.
4. On [line 8-11](javascript:void(0)), we explain what **calling** a function means.
5. On [line 12](javascript:void(0)), replace the 6 with any number and press Save & Submit Code. Do this a few times to see the beauty of functions!

**Script:**

//This is what a function looks like

var divideByThree = function (number) {

var val = number / 3;

console.log(val);

};

//On line 11, we call the function by name

//Here, it is called 'dividebythree'

//We tell the computer what the number input is (ie. 6)

//The computer then uses the code inside the function

divideByThree(12);

**Output:** 4

**Function syntax**

A function takes in inputs, does something with them, and produces an output.

Here's an example of a function:

var sayHello = function(name) {

console.log('Hello ' + name);

};

1. First we declare a function using var, and then give it a name sayHello. The name should begin with a lowercase letter and the convention is to use lowerCamelCase where each word (except the first) begins with a capital letter.
2. Then we use the function keyword to tell the computer that you are making a function
3. The code in the parentheses is called a **parameter**. It's a placeholder word that we give a specific value when we call the function. Click "Stuck? Get a hint!" for more.
4. Then write your block of reusable code between { }. Every line of code in this block must end with a ;.

You can run this code by "calling" the function, like this:

sayHello("Emily");

Calling this function will print out Hello Emily.

1. On [line 11](javascript:void(0)), call the greeting function and put in a name that you want the greeting function to include.
2. Press "Save & Submit Code" and see the function get into action! Saves you so much time.

When we want to join together two strings, we put a plus sign.

console.log("Hey" + "you"); will print out Heyyou. That's not what we want!

If you want a space between words, you must add that space as well!

console.log("Hey" + " " + "you"); will print out Hey you

This joining of strings is called **concatenation**

**Script:**

//Below is the greeting function

//See line 7

//We can join strings together using the plus sign +

//See the hint for more details about how this

var greeting = function (name) {

console.log("Great to see you," + " " + name);

};

//On line 11, call the greeting function!

var username = "Matt";

greeting(username);

**Output:** Great to see you, Matt

**How does a function work?**

Let's break down exactly how a computer thinks when it sees the code for a function.

var functionName = function( ) {

// code code code

// code code code

// (more lines of code)

};

1. The var keyword declares a variable named functionName.
2. The keyword function tells the computer that functionName is a function and not something else.
3. Parameters go in the parentheses. The computer will look out for it in the code block.
4. The code block is the reusable code that is between the curly brackets { }. Each line of code inside { } must end with a semi-colon.
5. The entire function ends with a semi-colon.

To use the function, we **call** the function by just typing the function's name, and putting a parameter value inside parentheses after it. The computer will run the reusable code with the specific parameter value substituted into the code.

Let's make a function that tells the world what you want to eat.

1. Declare your function and call it foodDemand.
2. You can call the parameter anything you like. But we'll call it food because that is the thing that is going to change each time we call the function.
3. Your reusable block of code follow this. Surround it with the right brackets. The code you want to repeat is: console.log("I want to eat" + " " + food);
4. Call the function and put in a specific food you want!

The parameter is basically what makes a function useful. Why? You want to use your code over and over again. But you want to use that code with different inputs! Some examples below:

1. We had the **bake** function. We could then call it putting in different ingredients.
2. We had the **calculate** function. We then called it putting in different specific numbers.
3. We had the **greeting** function. We then called it putting in different specific names.
4. We now have the **foodDemand** function. You will call it putting in whatever food name you want.

**Script:**

//Write your foodDemand function below.

//Last hint: In your reusable block of code, end each line

//with a ;

var foodDemand = function(food) {

console.log("I want to eat"+" "+ food);

};

foodDemand("taco");

**Output:** I want to eat taco

**Tying it all together**

**Why is the code organized like it is on** [**lines 2-5**](javascript:void(0))**?**

The computer can understand the code without such spacing. But it makes editing a lot easier and is best practice.

**Do I have to put a semi-colon at the end of each line of code in the reusable block? And at the end of the entire function?**

Yes. At the end of each line of code (within the { }) and after the entire function (after the { }), please put a semi-colon. The semi-colon acts like a period in a sentence. It helps the computer know where there are stopping points in the code.

A big part of programming is debugging. That just means figuring out what the heck went wrong with your code. Why didn't it run?

1. Look at [line 9](javascript:void(0)). It has many syntax errors. See how lack of spacing makes debugging hard?
2. Fix the function on [line 9](javascript:void(0)). Make sure the syntax is right. Make sure it looks nice.
3. Call the greeting function once it is fixed! Don't forget to pass in a specific name.

**Script:**

//Nicely written function

var calculate = function (number) {

var val = number \* 10;

console.log(val);

};

calculate(25);

//Badly written function with syntax errors

//greeting var func{name}(console.log(name)))}

var greeting = function(name) {

console.log(name);

};

greeting("Matt");

**Output:**

250

Matt

**Don't Repeat Yourself (D.R.Y)**

The D.R.Y. principle is really important in programming. No repeating!

Any time you find yourself typing the same thing, but modifying only one small part, you can probably use a function.

The 'small part' that you find yourself modifying will be the parameter. And the part that you keep repeating will be the code in the reusable block - the code inside { }.

You are a creature of habit. Every week you buy 5 oranges. But orange prices keep changing!

1. You want to declare a function that calculates the cost of buying 5 oranges.
2. You then want to calculate the cost of the 5 all together.
3. Write a function that does this called orangeCost().
4. It should take a parameter that is the cost of an orange, and multiply it by 5.
5. It should log the result of the multiplication to the console.
6. Call the function where oranges each cost 5 dollars.

What is the one bit of input that changes each time? It would be the price. So give your parameter the name price. And when you call your function, put in a number for the price to see how much 5 oranges cost!

**Script:**

var orangeCost = function(price) {

console.log(price \* 5);

};

orangeCost(5);

**Output:** 25

**Return keyword**

Nice job! Now, when we call a function, we don't always want to just print stuff. Sometimes, we just want it to return a value. We can then use that value (ie. the output from the function) in other code. Let's learn about the return keyword, then we'll see how to use functions with an if / else statement in the next exercise!

The return keyword simply gives the programmer back the value that comes out of the function. So the function runs, and when the return keyword is used, the function will immediately stop running and return the value.

In our example we have a function called timesTwo() that takes in a number and returns the number multiplied by two.

1. On [line 7](javascript:void(0)), after the equals sign, call the function timesTwo with any parameter you want
2. [Line 8](javascript:void(0)) prints out newNumber. Notice how the value we return from timesTwo() is automatically assigned into newNumber.

To call the function, we just use the name of the function. We then put in a value for the number parameter. eg. timesTwo(8);

**Script:**

//Parameter is a number, and we do math with that parameter

var timesTwo = function(number) {

return number \* 2;

};

// call timesTwo here

var newNumber = timesTwo(4);

console.log(newNumber);

**Output:** 8

**Functions, return and if / else**

When we call a function, its return value is just the result from running the function. That value can then be used just like any other value in JavaScript!

Look at the if statement starting on [line 7](javascript:void(0)). The if statement is checking whether the result of calling the function named quarter is divisible by 3.

1. Define a function called quarter which has a parameter called number.
2. This function returns a value equal to one quarter of the parameter. (i.e. number / 4;)
3. Call the function inside the if statement's **condition** (and put in a parameter value!) such that "The statement is true" is printed to the console.

This is a little tricky so let's take it slow. The code in the { } in the function should be

return number/4

On [line 7](javascript:void(0)), the if statement's condition calls the function! It needs you to put in number value.

Recall % is modulo. It gives you the remainder when one number is divided by another.  
eg. 13 % 3 is 1.

We want quarter(some number) that when divided by 3 has remainder 0.

**Script:**

// Define quarter here.

var quarter = function(number) {

return number/4;

};

if (quarter(24) % 3 === 0 ) {

console.log("The statement is true");

} else {

console.log("The statement is false");

}

**Output:** The statement is true

**Functions with two parameters**

So far we've only looked at functions with one parameter. But often it is useful to write functions with more than one parameter. For example, we can have the following function:

var areaBox = function(length, width) {

return length \* width;

};

With more than one parameter, we can create more useful functions

To call a function with more than one parameter, just enter a value for each parameter in the parentheses. For example, areaBox(3,9); would return the area of a box with a length of 3 and a width of 9.

1. Write a function called perimeterBox that returns the perimeter of a rectangle.
2. It should have two parameters.
3. One formula for perimeter is length + length + width + width;
4. Call the function and pass in any value for length and width you like.

Remember for functions, the parameters go in parentheses ( ). And the block of reusable code goes in curly brackets { }.

**Script:**

// Write your function starting on line 3

var perimeterBox = function(length, width) {

return length + length + width + width;

};

perimeterBox(4,5);

**Output:** 18

**Global vs Local Variables**

Let's talk about an important concept: **scope**. Scope can be global or local.

Variables defined **outside** a function are accessible anywhere once they have been declared. They are called **global variables** and their scope is **global**.

For example:

var globalVar = "hello";

var foo = function() {

console.log(globalVar); // prints "hello"

}

The variable globalVar can be accessed anywhere, even inside the function foo.

Variables defined **inside** a function are **local variables**. They cannot be accessed outside of that function.

For example:

var bar = function() {

var localVar = "howdy";

}

console.log(localVar); // error

The variable localVar only exists inside the function bar. Trying to print localVar outside the function gives a error.

Check out the code in the editor. Until now you've been using the var keyword without really understanding why. The var keyword creates a new variable **in the current scope**. That means if var is used outside a function, that variable has a global scope. If var is used inside a function, that variable has a local scope.

On [line 4](javascript:void(0)) we have not used the var keyword, so when we log my\_number to the console outside of the function, it will be 14.

Change [line 4](javascript:void(0)) to use the var keyword. Notice that the value of my\_number in the function is now 14 and outside the function is 7.

Using my\_number without the var keyword refers to the **global** variable that has already been declared outside the function in [line 1](javascript:void(0)). However, if you use the var keyword inside a function, it declares a new **local** variable that only exists within that function.

**Script:**

var my\_number = 7; //this has global scope

var timesTwo = function(number) {

var my\_number = number \* 2;

console.log("Inside the function my\_number is: ");

console.log(my\_number);

};

timesTwo(my\_number);

console.log("Outside the function my\_number is: ")

console.log(my\_number);

**Output:**

Inside the function my\_number is:

14

Outside the function my\_number is:

7

**Functions recap**

Okay let's review. You have learned a lot about functions today. We better not forget it all!

We first discovered how to use functions to perform the same piece of logic repeatedly, without having to type the same code again. This saves you time!

1. Write a function called nameString()
2. It should take name as a parameter.
3. The function returns a string equal to "Hi, I am" + " " + name.
4. Call nameString() by passing it your name, and use console.log to print the output.

The code block in the function should look like:

return "Hi, I am" + " " + name;

Just a reminder: we need to add " " to the string so we have a space between the word 'am' and the word 'name.

**Script:**

var nameString = function(name) {

return "Hi, I am" + " " + name;

};

console.log(nameString("Mathew"));

**Output:** Hi, I am Mathew

**Functions & if / else**

An especially useful application of reusable code is if/else statements. These can be very wordy, and a pain to type repeatedly.

We are going to write a function that checks how many hours of sleep a night you're getting. Inside the function will be an if/else statement. We want the function to check many different numbers of hours to see whether a person is getting enough sleep.

1. Write a function named sleepCheck that takes the parameter numHours
2. Inside the function, write an if statement where if the number of hours of sleep is greater than or equal to 8, the computer will return "You're getting plenty of sleep! Maybe even too much!";.
3. Otherwise (else) if the number of hours of sleep is less than 8, have the computer return "Get some more shut eye!";

Then call the function with different hours of sleep

1. Call the function with 10 hours of sleep, like this: sleepCheck(10);
2. Call the function with 5 hours of sleep.
3. Call the function with 8 hours of sleep.

Here's a list of of comparison operators:

* > Greater than
* < Less than
* <= Less than or equal to
* >= Greater than or equal to
* === Equal to
* !== Not equal to

Here's an example of if/else syntax:

if (condition1) {

return "some string";

}

else {

return "another string";

}

Don't forget to put semi-colons at the end of each line of code inside the { }. And to put a semi-colon at the very end of the function.

**Script:**

// Write your function below.

// Don't forget to call your function!

var sleepCheck = function(numHours){

if(numHours >= 8){

return "You're getting plenty of sleep! Maybe even too much!";

} else {

return "Get some more shut eye!";

}

}

sleepCheck(5);

**Output:** "Get some more shut eye!"

**Build "Rock, Paper, Scissors"**

**The Game**

Rock paper scissors is a classic 2 player game. Each player chooses either rock, paper or scissors. The possible outcomes:

* Rock destroys scissors.
* Scissors cut paper.
* Paper covers rock.

Our code will break the game into 3 phases:  
a. User makes a choice  
b. Computer makes a choice  
c. A compare function will determine who wins

**User Choice**

We start by first asking the user which option they want to pick. We will later use this choice in the compare function to determine the winner.

1. Declare a variable called userChoice.
2. Make the variable equal to the answer we get by asking the user "Do you choose rock, paper or scissors?"

Remember to use prompt to ask the user a question, like so:

var example = prompt("Question");

**Script:**

var userChoice = "";

prompt("Do you choose rock, paper or scissors?");

**Output:** a prompt asking to select rock, paper, or scissors

**Computer Choice: Part 1**

Awesome! We now need the computer to make a choice. The game is only going to be fun if the computer chooses randomly. Luckily JavaScript has something that can help with this.

If we declare a variable and make it equal to Math.random(), that variable will equal a number between 0 and 1.

1. Under your previous code, declare a variable called computerChoice and make it equal to Math.random().
2. Print out computerChoice so you can see how Math.random() works. This step isn't needed for the game - just useful for learning!

Your Math.random() code should look something like this:

var computerChoice = Math.random();

**Script:**

var userChoice = "rock";

prompt("Do you choose rock, paper or scissors?");

var computerChoice = Math.random();

console.log(computerChoice);

**Output:** a prompt to choose rock, paper, scissors, 0.7922923036516631

**Computer Choice: Part 2**

We have computerChoice but it now equals a random number between 0 and 1. We need to somehow translate this random number into a random choice of rock, paper, or scissors. How do we do this?!

1. If computerChoice is between 0 and 0.33, make computerChoice equal to "rock".
2. If computerChoice is between 0.34 and 0.66, make computerChoice equal to "paper".
3. If computerChoice is between 0.67 and 1, make computerChoice equal to "scissors".

But there are three outcomes! Using if / else only lets us have two outcomes. What now?! We need to use if / else if / else. See the hint for the full syntax. You will laugh at how easy it is.

1. Under your existing code, write out the if / else if / else statement.
2. In the respective code blocks, change the value of computerChoice based on the rules stated above. Remember, you do NOT have to use var when changing the value of a variable that already exists.

Syntax for if / else if / else should be spaced like:

if (condition1) {

code code code;

} else if (condition2) {

code code code;

} else {

code code code;

}

**Script:**

var userChoice = "";

prompt("Do you choose rock, paper or scissors?");

var computerChoice = Math.random();

//console.log(Math.random(1));

if (computerChoice <= 0.33) {

computerChoice = "rock";

} else if (computerChoice >= 0.66 ) {

computerChoice = "paper";

} else {

computerChoice = "scissors";

}

console.log(computerChoice);

**Output:** a prompt to choose rock, paper, scissors, based on your choice the computer will make it’s choice

**Both choices are the same!**

Now comes the fun part! We need to create a function. It will take two parameters (ie. the two choices made) and then return the winning choice.

When programming a game like this, you have to first figure out all the various outcomes. One outcome is that the choice the user makes is equal to the choice the computer makes.

1. We carried over the code from the previous section, but it is a comment. Leave it there for now.
2. Below the comment, declare a function called compare.
3. It takes two parameters, choice1 and choice2.
4. Inside the function, write an if statement. *If* choice1 === choice2, then return "The result is a tie!"

Your code should look something like this:

var compare = function(choice1, choice2) {

if(choice1 === choice2) {

return "The result is a tie";

}

};

Why do we call the parameters choice1 and choice2 instead of making use of the variable names userChoice and computerChoice?

Because parameter names are meant to be general. This makes them more flexible so that later we can pass any variables or values through the function. In our case, we will pass computerChoice and userChoice to the function.

For example, what if this was a game involving two human choices (ie. not a human against the computer). It wouldn't make sense to have computerChoice as a parameter!

**Script:**

/\*var userChoice = prompt("Do you choose rock, paper or scissors?");

var computerChoice = Math.random();

if (computerChoice <0.34){

computerChoice = "rock";

}else if(computerChoice <=0.67){

computerChoice = "paper";

}else{

computerChoice = "scissors";

}\*/

var compare = function(choice1, choice2) {

if (choice1 === choice2) {

return "The result is a tie!";

} else {

}

};

**Output:** nothing

**What if choice1 is rock?**

You're doing great! Now we consider the other scenarios. Let's break the problem down a little. What if choice1 is "rock"? Given choice1 is "rock",

a. if choice2 === "scissors", then "rock" wins.  
b. if choice2 === "paper", then "paper" wins.

How do we structure this? It's a bit different from what we have already seen. We will first have an if statement. And then the code inside that if statement will be... another if statement!

Let's code our outline from above:

1. Inside the compare() function under the existing code, write an else if statement where the condition is choice1 === "rock".
2. Inside this else if statement, write an if / else statement. *If* choice2 === "scissors", return "rock wins". *Else*, return "paper wins".

This putting if statements inside if statements is a little tricky at first. You code should look something like this:

if(choice1 === choice2) {

return "The result is a tie";

}

else if(choice1 === "rock") {

if(choice2 === "scissors") {

return "rock wins";

}

else {

return "paper wins";

}

}

**Script:**

/\*var userChoice = prompt("Do you choose rock, paper or scissors?");

var computerChoice = Math.random();

if (computerChoice <0.34){

computerChoice = "rock";

}else if(computerChoice <=0.67){

computerChoice = "paper";

}else{

computerChoice = "scissors";

}\*/

var compare = function(choice1,choice2){

if(choice1 === choice2)

return "The result is a tie!";

if (choice1 === "rock"){

if (choice2 === "scissors") {

return "rock wins";

} else {

return "paper wins";

}

}

};

**Output:** nothing

**What if choice1 is paper?**

Now what if choice1 is "paper"? Given choice1 is "paper",

a. if choice2 === "rock", then "paper" wins.  
b. if choice2 === "scissors", then "scissors" wins.

1. Inside the compare() function under the existing code, write another else if statement where the condition is choice1 === "paper".
2. Inside this else if statement, write an if / else statement. *If* choice2 === "rock", return "paper wins". *Else*, return "scissors wins".

This nesting of if / else blocks can be tricky, especially at first. Your code should have this basic outline:

if (condition) {

// do something

}

else if (condition) {

if (condition) {

// do something

}

else {

// do something else

}

}

**Script:**

/\*var userChoice = prompt("Do you choose rock, paper or scissors?");

var computerChoice = Math.random();

if (computerChoice <0.34){

computerChoice = "rock";

}else if(computerChoice <=0.67){

computerChoice = "paper";

}else{

computerChoice = "scissors";

}\*/

var compare = function(choice1,choice2){

if(choice1 === choice2)

return "The result is a tie!";

if (choice1 === "rock"){

if (choice2 === "scissors") {

return "rock wins";

} else {

return "paper wins";

}

}

if (choice1 === "paper"){

if (choice2 === "rock") {

return "paper wins";

} else {

return "scissors wins";

}

}

if (choice1 === "scissors"){

if (choice2 === "rock") {

return "rock wins";

} else {

return "scissors wins";

}

}

};

**Output:** nothing

**What if choice1 is scissors?**

Lastly, what if choice1 is "scissors"? Given choice1 is "scissors",

a. if choice2 === "rock", then "rock" wins.  
b. if choice2 === "paper", then "scissors" wins.

1. Under your old code, use the same structure as the past two exercises and finish off your function.
2. We need to use the variables userChoice and computerChoice. Uncomment the first lines of the code by deleting /\* on [line 1](javascript:void(0)) and \*/ on [line 9](javascript:void(0)), the variables are now active and can be called.
3. Call your function and pass in userChoice and computerChoice as your two arguments.
4. Press Save & Submit Code and your game should work! Congratulations!

This putting if statements inside if statements is a little tricky at first. Below is an outline of what the syntax should look like:

if (condition) {

// do something

}

else if (condition) {

if (condition) {

// do something

}

else {

// do something else

}

}

**Script:**

var userChoice = prompt("Do you choose rock, paper or scissors?");

var computerChoice = Math.random();

if (computerChoice <0.34){

computerChoice = "rock";

}else if(computerChoice <=0.67){

computerChoice = "paper";

}else{

computerChoice = "scissors";

}

var compare = function(choice1,choice2){

if(choice1 === choice2)

return "The result is a tie!";

if (choice1 === "rock"){

if (choice2 === "scissors") {

return "rock wins";

} else {

return "paper wins";

}

}

if (choice1 === "paper"){

if (choice2 === "rock") {

return "paper wins";

} else {

return "scissors wins";

}

}

if (choice1 === "scissors"){

if (choice2 === "rock") {

return "rock wins";

} else {

return "scissors wins";

}

}

};

compare(userChoice, computerChoice);

**Output:** a prompt to pick rock, paper, scissors and then the output if you won or not

**Next Steps**

Congratulations on making your awesome game! But now comes the best bit. You have the skills to build a game of your own design! Below are some ideas:

1. What if a user makes an inappropriate choice like 'dog'? How can we extend the function to handle that?
2. What if players in the game could also choose Rope in this game?
3. In this version, if both players make the same choice, the game returns a tie. What if the game didn't end there but instead asked both players for new choices?

**Script:**

var userChoice = prompt("Do you choose rock, paper or scissors?");

var computerChoice = Math.random();

if (computerChoice <0.25){

computerChoice = "rock";

}else if(computerChoice <=0.50){

computerChoice = "paper";

}else if(computerChoice <=0.75){

computerChoice = "rope";

}else{

computerChoice = "scissors";

}

var compare = function(choice1,choice2){

if(choice1 === choice2)

userChoice;

//return "The result is a tie!";

if (choice1 === "rock"){

if (choice2 === "scissors") {

return "rock wins";

} else if (choice2 === "paper"){

return "paper wins";

} else {

return "rope wins";

}

}

if (choice1 === "paper"){

if (choice2 === "rock") {

return "paper wins";

} else if (choice2 === "scissors"){

return "scissors wins";

} else {

return "rope wins";

}

}

if (choice1 === "rope"){

if(choice2 === "rock") {

return "rock wins";

} else if(choice2 === "scissors"){

return "scissors wins";

} else {

return "paper wins";

}

}

if (choice1 === "scissors"){

if (choice2 === "rock") {

return "rock wins";

} else if (choice2 === "scissors") {

return "scissors wins";

} else {

return "rope loses";

}

}

};

compare(userChoice, computerChoice);

**Output:** a prompt to pick rock, paper, scissors and then the output if you won or not

**Introduction to 'For' Loops in JS**

**Why use for loops?**

We are learning how to program because we don't want to do boring, repetitive work! The computer should do that.

This first exercise is a good example of exactly why you want to learn for loops.

1. Use five console.log statements to print out the numbers 1 to 5.
2. Try not getting angry at me for this annoying exercise.
3. Head over to the next exercise to see how we can use for loops to do this task more efficiently.

Be sure to use a separate console.log statement for each number from 1 to 5!

Remember, console.log() works like this:

console.log(/\* thing to print out \*/);

**Script:**

// Write five console.log statements.

console.log(1);

console.log(2);

console.log(3);

console.log(4);

console.log(5);

**Output:**

1

2

3

4

5

**First for loop**

Instead of manually typing in console.log five times, we can use a for loop to do this. The aim of this exercise is just to show you how a for loop looks, and demonstrate how useful it is. Subsequent exercises will

a. walk you through the syntax bit by bit  
b. explain how the computer thinks as it executes a for loop.

We initially focus on using for loops just to count numbers to keep things simple. But by section 3, we will show you how to do more fancy things!

1. The for loop in the code will print out 1 to 5 and use far less code than you used in the previous exercise.
2. Change the 6 to 11 and press Save & Submit Code. This will see the computer print out 1 to 10!

We want to print out 1 to 10. Why then is 11 in the code? Well, this is because we want the loop to continue to run where counter is less than 11. But once counter is equal to 11, we want it to stop. More on this part of the for loop syntax coming up!

**Script:**

// Example of a for loop

for( var counter = 1; counter < 11; counter ++) {

console.log(counter);

}

**Output:**

1

2

3

4

5

6

7

8

9

10

**Starting the for loop**

Congratulations! You've just run your first for loop. But what you're probably really keen to do is write your own for loop. Below is the general syntax of the for loop. We want to focus on the first line in the next few exercises.

**Syntax**

for (var i = 1; i < 11; i = i + 1) {

/\* your code here \*/;

}

Every for loop makes use of a counting variable. Here, our variable is called i (but it can have any name). The variable has many roles. The first part of the for loop tells the computer to start with a value of 1 for i. It does this by declaring the variable called i and giving it a value of 1.

When the for loop executes the code in the code block—the bit between { }—it does so by starting off where i = 1.

1. This for loop starts off at 1 and will end at 10.
2. Change the for loop such that it will start off at 5!

You'll want to adjust the var i = 1; on [line 3](javascript:void(0)).

**Script:**

// Change where the for loop starts.

for (var i = 5; i < 11; i = i + 1){

console.log(i);

}

**Output:**

5

6

7

8

9

10

**Ending the for loop**

We know how to control where the for loop starts. How do we control where it ends? Well, the second part of the for loop determines that.

**Syntax**

for (var i = 1; i < 11; i = i + 1) {

code code code;

}

Here, this for loop will keep running until i = 10 ( *i.e.* while i < 11). So when i = 2, or i = 9, the for loop will run. But once i is no longer less than 11, the loop will stop.

1. We know this for loop counts from 1 to 10.
2. Change this for loop such that it starts at 4.
3. Change this for loop such that it counts up to *and including* 23. ( *i.e.* we do NOT want 24 to be printed out!)
4. Run your for loop and see it count from 4 to 23!

Changing where the for loop starts means replacing the 1 in var i = 1; with the number you want the for loop to start at.

Changing where the for loop ends is slightly more tricky. If you want the for loop to stop printing at 23, then the condition should be i < 24.

**Script:**

// Edit this for loop

for (var i = 4; i < 24; i = i + 1){

console.log(i);

}

**Output:** prints out 4 to 23

**Controlling the for loop**

We can now control where the for loop starts and ends. What about controlling what happens in between?

The examples we've looked at have used i = i + 1. This has meant we have incremented (increased) the variable i by 1 each time.

**Rules to learn**

a. A more efficient way to code to increment up by 1 is to write i++.  
b. We decrement down by 1 by writing i--.  
c. We can increment up by any value by writing i += x, where x is how much we want to increment up by. *e.g.*, i += 3 counts up by 3s.  
d. We can decrement down by any value by writing i -= x. (See the Hint for more.)  
e. Be **very** careful with your syntax—if you write a loop that can't properly end, it's called an **infinite loop**. It will crash your browser!

1. This code counts every number from 0 to 35.
2. Make it start counting from 5. Please!
3. Stop the counting when it prints out 50.
4. Only count every fifth number. So we want to increment i by 5.

The changes to be made:

a. Replace 0 with the value you want the for loop to start.  
b. Replace 36 with where you want the for loop to end.  
c. Currently, the for loop increments up by 1. You want it to increment up by 5!

Note: x += y is the same as x = x + y. And x -= y is the same as x = x - y.

**Script:**

// Edit this for loop

for (var i = 5; i < 51; i+=5) {

console.log(i);

}

**Output:** prints out 5 -50

**How does it work?**

We've gone through the three bits of syntax for a for loop. But how exactly does it work? Let's imagine the steps the computer takes to run the for loop on the right.

1. It starts off with i = 2
2. It then asks: Is i currently less than 13? Because i = 2, this is true and we continue.
3. We do NOT increment now. Instead, if the condition is met, we run the code block.
4. Here, the code block prints out the value of i. It is currently 2 so 2 will be printed out.
5. Once the code block is finished, the for loop then increments / decrements. Here, we add 1.
6. Now i = 3. We check if it is less than 13. If it is true, we run the code block.
7. The code block runs, and then we increment.
8. We repeat these steps until the condition i < 13 is no longer true.
9. Make the computer start counting at 8.
10. Keep counting while i < 120.
11. Count up by increments of 12.

Remember, if you accidentally make a loop that never ends, your browser will crash!

Make sure to use i += 12, and not i + 12. The latter will crash your browser, because i will never be updated and the loop will go on forever!

**Script:**

// Example for loop

for (var i = 8 ; i < 120; i+=12) {

console.log(i);

}

**Output:**

8

20

32

44

56

68

80

92

104

116

**Practice counting down**

for loops only run when the condition is true.

**Do not run the code without changing it!**

It is important that there is a way for the for loop to end. If the for loop is always going to be true, then you will be stuck in an infinite loop and your browser will crash! Look at the code. It is bad.

a. It begins at i = 1.   
b. It will keep going as long as i >= 1.  
c. Because now i = 1, the code will run.  
  
d. We increment i by 1 and now i = 2. This satisfies the condition. We run the code.   
e. Increment i by 1 and now i = 3. This satisfies the condition that i >= 1. We run the code.  
f. We will keep incrementing the code up by 1. It will **always** satisfy the condition. The loop NEVER ends. This will crash your computer!

1. Change this code such that it starts counting from 10.
2. We want it to stop once it gets to 0.
3. We want it to count *down* by 1.

In the end, the numbers 10..0 inclusive, should be printed.

The hard part here is the condition. When we were counting up, the condition was always "keep going as long as i < some number."

Here, we want to count down. So we keep counting down as long as i >= some number. Because we decrement down, i will keep getting smaller until it no longer satisfies the condition.

**Script:**

//Example of infinite loop - it will crash your computer.

for( var i = 10 ; i >= -5 ; i-=1 ) {

console.log(i);

}

**Output:**

10

9

8

7

6

5

4

3

2

1

0

-1

-2

-3

-4

-5

**Last practice for loop**

You have a great handle on for loops now! This will be the last practice one before we look at cool ways to use them.

The next exercise introduces you to arrays. So instead of just counting numbers up and down, we can make the computer do many more interesting things with loops.

Once more, for practice: write a for loop that gets the computer to count down from 100 until 0 by 5. This time, make sure not to print 0.

Break the for loop down into its three main bits.

a. The iterator i begins at 100.  
b. We want the condition to let us print when i is greater than 0.   
c. We want to count down by 5. You should use i -= 5.

Remember, the syntax looks like this:

for (start; end; increment) {

// Do something!

}

Where start is the starting condition (like i = 0), end is the ending condition (like i < 11), and increment is how much i should go up or down by (like i++).

**Script:**

//Write your very own for loop

for (var i=100; i>=5; i-=5) {

console.log(i);

}

**Output:**

100

95

90

85

80

75

70

65

60

55

50

45

40

35

30

25

20

15

10

5

**Meet arrays**

Variables can store numbers or strings. But so far, we've only been able to store ONE number or ONE string. Good thing we have arrays. Arrays:

a. store **lists** of data  
b. can store **different data types** at the same time  
c. are **ordered** so the position of each piece of data is fixed

**Example**:

var names = ["Mao","Gandhi","Mandela"];

var sizes = [4, 6, 3, 2, 1, 9];

var mixed = [34, "candy", "blue", 11];

**Syntax**:  
var arrayName = [data, data, data];

Any time you see data surrounded by [ ], it is an array.

Make your own array called junk. Put 4 bits of data in it (first 2 strings, then 2 numbers).

1. Declare it using var.
2. Put [ ] around your data.
3. Separate each bit of data with a comma.
4. End it with a semi-colon.
5. Use console.log to print out junk.

**Script:**

//You are now declaring an array

//Arrays are an awesome data structure!

var junk = ["junk1", "junk2", 1, 2];

console.log(junk);

**Output:**

[ 'junk1', 'junk2', 1, 2 ]

**Array positions**

It's nice that we can put a list of data into an array. But now we need to learn how to get access to the data inside the array.

The position of things in arrays is fixed. So we just need to know the array name (here, it is junkData), and the position of the data we want, and we're done.

Small complication: the position (or the index) of each bit of data is counted starting from 0, not 1.

1. First element in the array: junkData[0]
2. Third element in the array: junkData[2]

Arrays have 0-based indexing, so we start counting the positions from 0.

Print out the fourth element of the array.   
1. Start with figuring out how to express what the fourth element in the array is.  
2. Then use console.log() to print things out!

Remember, the syntax is:

arrayName[index]

where index is a number starting at 0.

**Script:**

//Practice array

var junkData = ["Eddie Murphy", 49, "peanuts", 31];

console.log(junkData[3]);

**Output:** 31

**Loops and arrays I**

Awesome job! You've now learned about arrays, and how to access one element of the array. But what if there were 100 elements in the array?

For arrays, a useful way to systematically access every element in the array is to use a for loop!

**How does it work?**  
  
1. [Line 3](javascript:void(0)) declares the array. It has 4 elements.  
2. We then start the for loop on [line 5](javascript:void(0)).  
3. We see i starts off at value 0.   
4. The for loop runs until i < 4 (because cities.length equals 4. The array cities has 4 elements in it; see the Hint for more.)  
5. We will increment i by 1 each time we loop over.  
6. We print out cities[0], which is "Melbourne".  
7. We then start the loop again. Except now i = 1.   
8. It will print out cities[1], which is "Amman".   
9. This continues until i is no longer less than cities.length.

1. Change the elements in the cities array. You can put in as many elements as you like.
2. Run the for loop and see them all printed out!

This combination of for loops and arrays is used a lot. In particular, pay attention to the condition part of the for loop where we have

i < cities.length

This is a pattern we often see.

The length of the cities array is 4. But when we are accessing the array, the last index position of the array is array[3]—zero-based indexing! Keep this in mind!

Remember, arrays look like this:

var arrayName = [1, 2, 3];

**Script:**

//Let's print out every element of an array using a for loop

var cities = ["Melbourne", "Amman", "Helsinki", "NYC", "Reston", "Sterling"];

for (var i = 0; i < cities.length; i++) {

console.log("I would like to visit" + " " + cities[i]);

}

**Output:**

I would like to visit Melbourne

I would like to visit Amman

I would like to visit Helsinki

I would like to visit NYC

I would like to visit Reston

I would like to visit Sterling

**Loops and arrays II**

It's time for you to write your own array and loop over the array. Remember to:

1. Put commas between each element in the array.
2. Put semi-colons between each bit of the for loop.
3. We suggest you use i as the iterator.
4. Beware of infinite loops!
5. Enjoy yourself while smashing through this coding!

Instructions

1. Create an array called names filled with 5 names.
2. Write a for loop that prints "I know someone called " followed by names[i]. Make sure there's a space between "called" and the name!
3. Run your code and the five sentences should print out.

Click "Stuck? Get a hint!" for an example of how to write a for loop.

1. Create an array called names filled with 5 names.
2. Write a for loop that prints "I know someone called " followed by names[i]. Make sure there's a space between "called" and the name!
3. Run your code and the five sentences should print out.

Click "Stuck? Get a hint!" for an example of how to write a for loop.

Your code should look something like this:

var names = ["put", "your", "five", "names", "here"];

for (var i = 0; i < someNum; i ++) {

console.log(something);

}

Where someNum is the length of the names array and something is "I know someone called " + names[i].

**Script:**

//Ask a question on the Q&A Forum if you get stuck!

var names = ["Connor", "Olivia", "Michael", "Andrew", "Leslie"];

for (var i = 0; i < names.length; i++) {

console.log("I know someone called"+" "+names[i]);

}

**Output:**

I know someone called Connor

I know someone called Olivia

I know someone called Michael

I know someone called Andrew

I know someone called Leslie

**Conclusion**

You've done an awesome job! Loops are always a little tricky when you first meet them. But they are worth learning because they are really useful.

What now? You have so many useful tricks up your sleeve:

a. if / else statements  
b. functions  
c. for loops  
d. booleans, arrays, variables, etc.

**Search Text for Your Name**

**What you'll be building**

In this project, you'll be writing a short program that checks a block of text for your name.

Specifically, it will check the text for the first letter of your name, then push (add) the number of characters equal to your name's length to an array. By inspecting the array, you'll be able to see if your name was mentioned!

Once you've got the hang of loops, arrays, and if statements, we'll talk about ways this project could be improved.

**Script:**

/\*jshint multistr:true \*/

text = "Blah blah blah blah blah blah Eric \

blah blah blah Eric blah blah Eric blah blah \

blah blah blah blah blah Eric";

var myName = "Eric";

var hits = [];

for(var i = 0; i < text.length; i++){

if (text[i] == "E"){

for(var j = i; j < (myName.length + i); j++){

hits.push(text[j]);

}

}

}

if (hits.length === 0){

console.log("Your name wasn't found!");

} else {

console.log(hits);

}

**Output:**

[ 'E', 'r', 'i', 'c', 'E', 'r', 'i', 'c', 'E', 'r', 'i', 'c', 'E', 'r', 'i', 'c' ]

**Declare your variables**

We'll start by declaring the three variables we'll be using: text, myName, and hits.

Since text could be quite long, you can use a backslash (\) at the end of each line to make your string "wrap" to the next line, like this:

var text = "Hey, how are you \

doing? My name is Emily.";

In this way, using backlashes (\) is useful because you can avoid really long lines!

You can ignore the /\*jshint... line for now. All that does is tell the console to stop worrying about our use of backslash characters for wrapping long lines of text.

Declare and set the following three variables:

1. text, and make it a string containing some text. Place your name in there a couple of times!
2. myName, and make it a string containing just your name.
3. hits, and make it an empty array.

Click "Stuck? Get a hint!" if you want to look at examples for declaring and setting variables.

To create a new string, you could do something like:

var myName = "Emily";

To create a new array, you could do something like this:

var hits = [];

**Script:**

/\*jshint multistr:true \*/

var text = "This is a story about a guy name Matt.\

Matt was skinny but his belly was fat.\

He ate too much food.";

var myName = "Matt";

var hits = [];

**Output:** nothing

**Your first "for" loop**

Awesome! Now let's write our outer for loop.

Below your existing code, create a for loop that starts at 0, continues until it reaches the end of text, and increments by 1 each time. (This means it will check each character in the string.) There's no need to write anything between the {}s of your loop just yet.

Remember the syntax for a for loop:

for(var i = 0; i < condition; i++){

//Do something!

}

In place of "condition," put the condition that will stop the loop from running. In this case, our condition should be less than text.length.

**Script:**

/\*jshint multistr:true \*/

var text = "This is a story about a guy name Matt.\

Matt was skinny but his belly was fat.\

He ate too much food.";

var myName = "Matt";

var hits = [];

for(var i = 0; i < text.length; i++) {

}

**Output:** nothing

**Your "if" statement**

Nice work! Now let's move on to the if statement.

We'll want to place the if statement inside our for loop to make sure the program checks the if statement each time it moves forward through the loop. Essentially, the for loop is saying: "Hey program! Go through every letter in 'text'." The if statement will say: if you see something interesting, push that text into an array!"

You can treat a string like an array of characters. For instance, you know that

var myArray = ['hello', 'world'];

myArray[0]; // equals 'hello'

But this also works on strings!

var myName = 'Eric';

myName[0]; // equals 'E'

Add your if statement in the body of your for loop. It should check to see whether the current letter is equal to the first letter of your name. (Capitalization counts!)

There's no need to put anything between the {}s of your if just yet.

Remember, an if statement looks like this:

if (some condition) {

// Do something!

}

You can check the ith letter of text like so:

// Assuming your name starts with 'E'

if (text[i] === 'E') {

// Do something!

}

**Script:**

/\*jshint multistr:true \*/

var text = "This is a story about a guy name Matt.\

Matt was skinny but his belly was fat.\

He ate too much food.";

var myName = "Matt";

var hits = [];

for(var i = 0; i < text.length; i++) {

if (text[i] === "M"){

}

}

**Output:** nothing

**Your second "for" loop**

Okay! Last loopy step: add *another* for loop, this time inside the body of your if statement (between the if's {}s).

This loop will make sure each character of your name gets pushed to the final array. The if statement says: "If we find the first letter of the name, start the second for loop!" This loop says: "I'm going to add characters to the array until I hit the length of the user's name." So if your name is 11 letters long, your loop should add 11 characters to hits if it ever sees the first letter of myName in text.

For your second for loop, keep the following in mind:

First, you'll want to set your second loop's iterator to start at the first one, so it picks up where that one left off. If your first loop starts with

for(var i = 0; // rest of loop setup

your second should be something like

for(var j = i; // rest of loop setup

Second, think hard about when your loop should stop. Check the Hint if you get stuck!

Finally, in the body of your loop, have your program use the .push() **method** of hits. Just like strings and arrays have a .length method, arrays have a .push() method that adds the thing between parentheses to the end of the array. For example,

newArray = [];

newArray.push('hello');

newArray[0]; // equals 'hello'

Your loop should stop when it hits the value of the first iterator (say, i) plus the length of your myName variable.

**Script:**

/\*jshint multistr:true \*/

var text = "This is a story about a guy name Matt.\

Matt was skinny but his belly was fat.\

He ate too much food.";

var myName = "Matt";

var hits = [];

for (var i = 0; i < text.length; i++) {

if (text[i] === "M"){

for(var j = i; j < (myName.length + i); j++){

hits.push(text[j]);

}

}

}

**Output:** 8

**Log it!**

Perfect! You've now got the engine of your search program running. It will:

1. Loop through the array,
2. Compare each letter to the first letter of your name, and if it sees that letter:
3. It will push that letter and all the letters that follow it to an array, stopping when the number of letters it pushes are equal to the number of letters in your name.

Under your existing code (and outside all your loops!), set up an if/else statement. If you don't have any hits, log "Your name wasn't found!" to the console. Otherwise, log the hits array to the console.

Feel free to peek back at the first exercise if you need help! Your console.log() code should look something like this:

if(/\*hits.length is 0\*/) {

console.log("Your name wasn't found!");

} else {

console.log(hits);

}

**Script:**

/\*jshint multistr:true \*/

var text = "This is a story about a guy name Matt.\

Matt was skinny but his belly was fat.\

He ate too much food.";

var myName = "Matt";

var hits = [];

for (var i = 0; i < text.length; i++) {

if (text[i] == "M"){

for(var j = i; j < (myName.length + i); j++){

hits.push(text[j]);

}

}

}

if (hits.length === 0){

console.log("Your name wasn't found!");

} else {

console.log(hits);

}

**Output:** [ 'M', 'a', 't', 't', 'M', 'a', 't', 't' ]

**Victory!**

You've done it! Nice work.

Now, as we mentioned, this system isn't perfect. For instance, if the paragraph contains both "Eric" and "Eddie", we'll see this in our hits array:

['E','r','i','c','E','d','d','i','e'];

Think about how you might fine-tune this program to make sure it *only* finds exact matches for your name. Search the Internet to see if there are any built-in JavaScript string methods that can help!

**Script:**

/\*jshint multistr:true \*/

var text = "This is a story about a guy name Matt.\

Matt was skinny but his belly was fat.\

He ate too much food. But what about Melvin?";

var myName = "Matt";

var hits = [];

for (var i = 0; i < text.length; i++) {

if (text[i] == "M"){

for(var j = i; j < (myName.length + i); j++){

hits.push(text[j]);

}

}

}

if (hits.length === 0){

console.log("Your name wasn't found!");

} else {

console.log(hits);

}

**Output:** [ 'M', 'a', 't', 't', 'M', 'a', 't', 't', 'M', 'e', 'l', 'v' ]

**Introduction to 'While' Loops in JS**

**While we're at it**

Great work with for loops! As a reminder, for loop syntax looks like this:

for (var i = start; i < end; i++) {

// do something

}

The counter variable i starts at "start", and stops looping when it reaches "end."

But what if you didn't know ahead of time when to stop looping? Say, for example, you wanted to keep choosing playing cards from a deck until you get a spade. You don't know how many cards you'll need to choose, so a for loop won't work.

In situations like these where you don't know in advance when to stop looping, we can use a while loop.

Check out the while loop in the editor. Can you guess what it will do? Hit "Save & Submit Code" when you think you know! (The answer is in the Hint.)

Don't worry about the Math.floor bit for now—we'll explain it soon!

The code in the editor keeps flipping a coin until it is tails. Here's how it works:

1. In [line 1](javascript:void(0)), we create a variable named coinFace, which is a random number that is either 0 (heads) or 1 (tails).
2. Then in [lines 3-5](javascript:void(0)) we keep flipping the coin as long as the coin turns up heads. If coinFace is 0 (heads), then the condition in the while loop will evaluate to true, and we flip the coin again.
3. If coinFace is 1 (tails), then the condition will be false, so we break out of the while loop and print Tails! Done flipping.

**Script:**

var coin = Math.floor(Math.random() \* 2);

while(coin){

console.log("Heads! Flipping again...");

var coin = Math.floor(Math.random() \* 2);

}

console.log("Tails! Done flipping.");

**Output:** Tails! Done flipping.

**While syntax**

The while loop is ideal when you want to use a loop, but you don't know how many times you'll have to execute that loop.

In the example you just saw, the computer was randomly flipping a coin: while the coin came up heads (when coinFace equalled 0), it would flip again, and it would stop flipping once it got tails (when coinFace was 1). Since the flip was random, we didn't know ahead of time how many loops we'd need.

The syntax looks like this:

while(condition){

// Do something!

}

As long as the condition evaluates to true, the loop will continue to run. As soon as it's false, it'll stop. (When you use a number in a condition, as we did earlier, JavaScript understands 1 to mean true and 0 to mean false.)

Since you've already mastered for loops, this simpler syntax should be a breeze for you.

Try it yourself—complete the while loop in the editor so it will print out "I'm learning while loops!". Do this by adding the condition between the parentheses—don't change [line 5](javascript:void(0)), or you could get an infinite loop!

**Script:**

var understand = true;

while(understand){

console.log("I'm learning while loops!");

understand = false;

}

**Output:**

I'm learning while loops!

false

**A fellow of infinite loops**

Great work!

We mentioned **infinite loops** in the previous exercise. If you give a while loop a condition that is true and you don't build in a way for that condition to possibly become false, the loop will go on forever and your program will crash. No good!

To prevent this from happening, you *always* need a way to ensure the condition between your while parentheses can change.

You'll see the same code from the last exercise in the editor to the right, only we've taken out the part that changes the loop's condition.

**Script:**

understand = true;

while(understand){

console.log("I'm learning while loops!");

//Change the value of 'understand' here!

understand = false;

}

**Output:**

I'm learning while loops!

False

**Brevity is the soul of programming**

You may have noticed that when we give a variable the boolean value true, we check that variable directly—we don't bother with ===. For instance,

var bool = true;

while(bool){

//Do something

}

is the same thing as

var bool = true;

while(bool === true){

//Do something

}

but the first one is faster to type. Get in the habit of typing exactly as much as you need to, and no more!

If you happen to be using numbers, as we did earlier, you could even do:

var myNumber = 1;

while(myNumber) {

// Do something!

}

Your condition should only be:

while(bool)

**Script:**

var bool = 1;

while(bool){

console.log("Less is more!");

bool = 0;

}

// 1 equals true

// 0 equals false

**Output:**

Less is more!

0

**Practice makes perfect**

Okay. Time for you to create a while loop from scratch!

We've set up a function, loop, for you to write your while loop in, as well as created the empty loop.

Remember to set up the condition you're checking *outside* the loop—if you do it *in* the loop, it will keep resetting and the loop could go on forever!

Write a while loop that logs "I'm looping!" to the console three times. You can do this however you like, but NOT with three console.log calls. Check the Hint if you need help!

We actually know we need to loop three times, so we could use a for loop, but we'll use while this time.

If we create a variable called count and set it to 0 outside the loop (on [line 2](javascript:void(0))), then do count++ each time we console.log() inside the loop, we'll be able to track how many loops we've made. If we set the while condition to be count < 3, that should do the trick!

**Script:**

//Remember to make your condition true outside the loop!

var i = 0;

var loop = function(){

while(i < 3){

//Your code here!

console.log("I'm looping!");

i += 1;

}

};

loop();

**Output:**

I'm looping!

I'm looping!

I'm looping!

**Solo flight**

Great work! Let's try another. This time, no help at all! (Well, *some* help—check the Hint if you get stuck.)

Inside the soloLoop function, write a while loop that takes an initial condition that's true. Your loop should log "Looped once!" to the console, then change that initial condition to false.

MAKE SURE to set your condition to false in the body of your loop. Otherwise, you'll loop forever!

Remember, while loops look like this:

while(condition) {

// Do something!

}

So your loop should do something like this:

var myCondition = true;

while(myCondition) {

console.log("Looped once!");

myCondition = false;

}

**Script:**

//Remember to make your condition true outside the loop!

var soloLoop = function(){

//Your code here!

var loop = true;

while(loop) {

console.log("Looped once!");

loop = false;

}

};

soloLoop();

**Output:** Looped once!

**When to 'while' and when to 'for'**

As we mentioned, for loops are great for doing the same task over and over when you know ahead of time how many times you'll have to repeat the loop. On the other hand, while loops are ideal when you have to loop, but you don't know ahead of time how many times you'll need to loop.

As you saw, however, you can combine a while loop with a counter variable to do the same kind of work a for loop does. In these cases, it's often a matter of preference.

Write two loops in the editor: one while, one for. No restrictions on this one; just make sure your loops are syntactically correct, and be careful to avoid infinite loops!

Remember your syntax! While loops look like this:

while(condition) {

// Do while condition is true

}

For loops look like this:

for(start; end; increment) {

// Do something!

}

For example, you might have a for loop that looks like this:

for (var i = 1; i < 11; i++) {

// Print the numbers 1 - 10

console.log(i);

}

**Script:**

var fart=true;

while(fart){

console.log("Boy, that was stinky!");

fart=false;

}

for (var i=1; i<11; i++){

console.log(i);

}

**Output:**

Boy, that was stinky!

1

2

3

4

5

6

7

8

9

10

**The 'do' / 'while' loop**

Sometimes you want to make sure your loop runs *at least one time* no matter what. When this is the case, you want a modified while loop called a do/while loop.

This loop says: "Hey! Do this thing one time, *then* check the condition to see if we should keep looping." After that, it's just like a normal while: the loop will continue so long as the condition being evaluated is true.

It runs once because do tells it to, but then never again because loopCondition is false!

**Script:**

loopCondition = false;

do {

console.log("I'm gonna stop looping 'cause my condition is " + String(loopCondition) + "!");

} while (loopCondition === true);

**Output:**

I'm gonna stop looping 'cause my condition is false!

**To learn it, you gotta 'do' it**

Your turn! Now that you've seen how do/while loops work, you can easily write your own. (Check the Hint if you need a syntax refresher!)

Your loop should print a string of your choice to the editor one time. Remember: make sure you give your while condition a way to become false, or it'll loop forever!

Write a do/while loop inside the function we've created for you, getToDaChoppa. The function should log a string of your choice to the console. do it now!

Remember, the syntax looks like this:

do {

// The thing to do at least once!

} while( /\* some condition \*/ );

**Script:**

var getToDaChoppa = function(){

//Write your do/while loop here!

loop = false;

do {

console.log("This is do do ");

}while (loop===true);

};

getToDaChoppa();

**Output:** This is do do

**Review**

To finish up and prove your loop mastery, write three syntactically correct loops in the editor: one for, one while, and one do. Beware of infinite loops!

Remember your loop syntax!

For example:

for (var i = 1; i < 11; i++) {

// Prints numbers 1 - 10

console.log(i);

}

While example:

var condition = true;

while(condition) {

console.log("Hello!");

// Avoid infinite loops!

condition = false;

}

Do/while example:

// Can be false from the start, since

// do/while runs at least one time

var condition = false;

do {

console.log("I'm printed once!");

} while(condition);

**Script:**

for (var i=0; i < 10; i++) {

console.log(i);

}

var understand = true;

while(understand){

console.log("I'm learning while loops!");

understand = false;

}

loopCondition = false;

do {

console.log("I'm gonna stop looping 'cause my condition is " + String(loopCondition) + "!");

} while (loopCondition === true);

**Output:**

0

1

2

3

4

5

6

7

8

9

I'm learning while loops!

I'm gonna stop looping 'cause my condition is false!

**Dragon Slayer!**

**What you'll be building**

Now that you know how to use while loops, we'll combine them with some other **control flow** statements (like if/else) to create a dragon slaying mini game.

In this game, you’ll battle a dragon. It will take 4 hits to slay the dragon, and if you miss even one hit, the dragon will defeat you!

Do you understand how it works? (No worries if not—we'll go through it step-by-step!)

Run it a few times to see how you fare against the dragon!

**Script:**

var slaying = true;

var youHit = Math.floor(Math.random() \* 2);

var damageThisRound = Math.floor(Math.random() \* 5 + 1);

var totalDamage = 0;

while (slaying) {

if (youHit) {

console.log("You hit the dragon and did " + damageThisRound + " damage!");

totalDamage += damageThisRound;

if (totalDamage >= 4) {

console.log("You did it! You slew the dragon!");

slaying = false;

} else {

youHit = Math.floor(Math.random() \* 2);

}

} else {

console.log("The dragon burninates you! You're toast.");

slaying = false;

}

}

**Output:** The dragon burninates you! You're toast.

false

**Declare your variables**

All right! Let's start by declaring the variables we'll be using. We'll need:

1. a variable to check if we're still slaying
2. a variable to check if we hit the dragon
3. a variable to keep track of how much damage we've dealt the dragon this round
4. a variable to keep track of total damage

Declare and set the following variables:

1. slaying equal to true
2. youHit to Math.floor(Math.random() \* 2). This sets youHit to a random number that's either 0 (which JavaScript reads as false) or 1 (which JavaScript reads as true).
3. damageThisRound to Math.floor(Math.random()\*5 + 1). This sets damageThisRound to a random number that's between 1 and 5 (up to and including 5).
4. totalDamage to 0

Click on "Stuck? Get a hint!" for more details on how Math.floor(Math.random()\*5 + 1) works.

How does this code work?

Math.floor(Math.random() \* 5 + 1);

1. First we use Math.random() to create a random number from 0 up to 1. For example, 0.5
2. Then we multiply by 5 to make the random number from 0 up to 5. For example, 0.5 \* 5 = 2.5
3. Next we use Math.floor() to round down to a whole number. For example, Math.floor( 2.5 ) = 2
4. Finally we add 1 to change the range from between 0 and 4 to between 1 and 5 (up to and including 5)

**Script:**

var slaying = true;

var youHit = Math.floor(Math.random() \* 2);

var damageThisRound = Math.floor(Math.random() \* 5 + 1);

var totalDamage = 0;

**Output:** nothing

**The 'while' loop**

Awesome! Now let's add in our while loop. We want to run the whole game as long as we're trying to kill the dragon—that is, while slaying is true.

When checking variables like slaying that are set to true, you don't need to write something like:

while(slaying === true)

You can just write

while(slaying)

It also helps to give your variables names that make the code look more like regular English. while(slaying) { /\*Do this\*/ } is easy to remember because it's so close to everyday speech!

Create a while loop that only executes when slaying is true. For this exercise, set slaying to false in the body of the loop. (We want to make sure the loop can exit—no infinite loops for us!)

Your code should look something like this:

var slaying = true;

while(slaying) {

slaying = false;

}

**Script:**

var slaying = true;

var youHit = Math.floor(Math.random() \* 2);

var damageThisRound = Math.floor(Math.random() \* 5 + 1);

var totalDamage = 0;

while(slaying) {

slaying = false;

}

**Output:** false

**The first 'if' statement**

Great! Now we want to add a couple of branches to our program so it can handle different outcomes. You know what this means: if and else!

Inside your while loop, create an if/else statement that checks the value of youHit. If it's 1 (true), it should log a congratulatory message to the console, saying that you hit the dragon. If it's 0 (false), it should log a message to the console saying that the dragon defeated you.

Either way, slaying should be set to false, since either you beat the dragon (and the slaying's over) or the dragon beat you!

Your code should look something like this:

while(slaying) {

if (youHit) {

console.log("You hit!");

} else {

console.log("You missed!");

}

slaying = false;

}

**Script:**

var slaying = true;

var youHit = Math.floor(Math.random() \* 2);

var damageThisRound = Math.floor(Math.random() \* 5 + 1);

var totalDamage = 0;

while(slaying) {

if(youHit === 1) {

console.log("You killed the beast!");

} else {

console.log("You have failed!");

}

slaying = false;

}

**Output:** You killed the beast!

false

**The second 'if' statement**

Good work! We're almost there.

In the first branch of our if statement (right after the console.log() where we congratulate the player for hitting the dragon), let's set totalDamage equal to totalDamage + damageThisRound. There's a shortcut for this: the += operator! When you type

totalDamage += damageThisRound;

you're telling JavaScript to add totalDamage and damageThisRound together, then assign that new value to totalDamage.

Go ahead and set totalDamage to totalDamage plus damageThisRound.

Then, inside your first if statement, create a second if statement that checks to see if totalDamage is greater than or equal to 4. If so, it should log to the console that the player slew the dragon and set slaying equal to false (since the dragon's dead, the slaying is over).

If totalDamage isn't greater than or equal to 4, youHit should be assigned a new random 1 or 0. (This is as easy as setting youHit to the same expression you used when you first declared it.)

Your code should now look something like this:

while (slaying) {

if (youHit) {

console.log("You hit!");

totalDamage += damageThisRound;

if (totalDamage >= 4) {

console.log("You win!");

slaying = false;

} else {

youHit = Math.floor(Math.random() \* 2);

}

} else {

console.log("You lose!");

slaying = false;

}

}

**Script:**

var slaying = true;

var youHit = Math.floor(Math.random() \* 2);

var damageThisRound = Math.floor(Math.random() \* 5 + 1);

var totalDamage = 0;

while(slaying) {

if(youHit === 1) {

console.log("You hit the beast!");

totalDamage += damageThisRound;

if(totalDamage >= 4){

console.log("You've slain the beast!");

slaying = false;

}

} else {

console.log("You have failed!");

if(totalDamage <=4){

youHit = Math.floor(Math.random() \* 2);

slaying = false;

}

}

}

**Output:**

You hit the beast!

You hit the beast!

You've slain the beast!

false

**Well done!**

You did it! You've written your own dragon-slaying game.

Feel free to look back at the first exercise for ideas on how you can customize your console.log() messages to the player. You can also invent your own!

Also try changing the values of some of the variables to see how it affects the player's odds of winning!

**Script:**

var slaying = true;

var youHit = Math.floor(Math.random() \* 2);

var damageThisRound = Math.floor(Math.random() \* 5 + 1);

var totalDamage = 0;

while (slaying) {

if (youHit) {

console.log("You hit the dragon and did " + damageThisRound + " damage!");

totalDamage += damageThisRound;

if (totalDamage >= 4) {

console.log("You did it! You slew the dragon!");

slaying = false;

} else {

youHit = Math.floor(Math.random() \* 2);

}

} else {

console.log("The dragon burninates you! You're toast.");

slaying = false;

}

}

**Output:** The dragon burninates you! You're toast.

false

**More on Control Flow in JS**

**If / else**

You've learned about if and else, and how they control what your program does. Here's a quick refresher on the syntax:

if (/\* Some condition \*/) {

// Do something

} else if (/\* Some other condition \*/) {

// Do something else

} else { // Otherwise

// Do a third thing

}

Write an if / else statement inside the isEven function. It should return true; if the number it receives is evenly divisible by 2. Otherwise (else), it should return false;.

Make sure to return - don't use console.log()!

Remember, you can use modulo (%) to see if a number is evenly divisible by another number.

Be sure to return true; or return false; in your code—don't console.log()!

**Script:**

var isEven = function(number) {

// Your code here!

if (number % 2 === 0){

return true;

} else {

return false;

}

};

isEven(2);

**Output:** true

**If / else if / else**

Good! Let's also get some practice in with else if, as well as learn about a fancy new function: isNaN.

If you call isNaN on something, it checks to see if that thing *is not* a number. So:

isNaN('berry'); // => true

isNaN(NaN); // => true

isNaN(undefined); // => true

isNaN(42); // => false

Be careful: if you call isNaN on a string that looks like a number, like '42', JavaScript will try to help by automatically converting the string '42' to the number 42 and return false (since 42 *is* a number).

Note that you *can't* just do

isNaN(unicorns);

unless you've already defined the variable unicorns. You *can*, however, do

isNaN("unicorns"); // => true

Add an else if branch to your existing if/else statement. If the number put into the function is not a number at all, instead of return true; or return false;, the function should return a string that tells the user that their input isn't a number. (This string can say whatever you like.)

if (condition1) {

return "some string"

}

else if (condition2) {

return "another string"

}

else {

return "yet another string"

}

**Script:**

var isEven = function(number) {

// Your code here!

if (number % 2 === 0){

return true;

} else if (isNaN(number)){

return number + " is not a number!";

} else {

return false;

}

};

isEven("ghjhj");

**Output:**

"ghjhj is not a number!"

**For or while**

Great! Just one more bit of review and we'll move on to the new stuff.

Create a for or while loop in the editor. It can do anything you like! (Just be careful—if you accidentally create an infinite loop, you'll crash your browser.) Check the Hint if you need a syntax review.

For example:

for (var i = 1; i < 11; i++) {

// Prints numbers 1 - 10

console.log(i);

}

While example:

var condition = true;

while(condition) {

// Says "Hello!"

console.log("Hello!");

// Avoid infinite loops

condition = false;

}

**Script:**

var test = true;

while(test){

console.log("whoop");

test = false;

}

**Output:**

Whoop

False

**Sneak preview: the switch statement**

As you might imagine, if you have a lot of choices you want to cover in a program, it might be annoying to type else if () ten times. That's why JavaScript has the switch statement!

switch allows you to preset a number of options (called cases), then check an expression to see if it matches any of them. If there's a match, the program will perform the action for the matching case; if there's no match, it can execute a default option.

Can you see how the switch statement works? Hit Save & Submit Code to take it for a test drive!

**Script:**

var lunch = prompt("What do you want for lunch?","Type your lunch choice here");

switch(lunch){

case 'sandwich':

console.log("Sure thing! One sandwich, coming up.");

break;

case 'soup':

console.log("Got it! Tomato's my favorite.");

break;

case 'salad':

console.log("Sounds good! How about a caesar salad?");

break;

case 'pie':

console.log("Pie's not a meal!");

break;

default:

console.log("Huh! I'm not sure what " + lunch + " is. How does a sandwich sound?");

break;

}

**Output:** prompt asking what you eat for lunch, then gives you a message.

Got it! Tomato's my favorite.

**Adding to an existing switch**

The switch statement is put together like this:

switch (/\*Some expression\*/) {

case 'option1':

// Do something

break;

case 'option2':

// Do something else

break;

case 'option3':

// Do a third thing

break;

default:

// Do yet another thing

}

JavaScript will try to match the expression between the switch() parentheses to each case. It will run the code below each case if it finds a match, and will execute the default code if no match is found.

Our switch statement needs a case for 'yellow'. Add it in and make it log a string of your choice to the console (it should be different from the default string).

Don't forget to end your case with a break statement—otherwise, it will go on and execute the code for default, too! We don't want that.

Your code should look something like this:

case 'yellow':

console.log("Your string here!");

break;

**Script:**

var color = prompt("What's your favorite primary color?","Type your favorite color here");

switch(color){

case 'red':

console.log("Red's a good color!");

break;

case 'blue':

console.log("That's my favorite color, too!");

break;

//Add your case here!

case 'yellow':

console.log("Yellow is sunny!");

break;

default:

console.log("I don't think that's a primary color!");

break;

}

**Output:** prompt that asks what your favorite color is the gives you a message

Yellow is sunny!

**Practice with switch**

Now that you've added cases to an existing switch, let's practice adding a default block.

Add the default block at the bottom of the switch statement, then run the code a few times with different inputs. switch: super useful!

Your code should add a default block after the break on [line 12](javascript:void(0)). It can do anything you like! For example:

default:

console.log("I don't know that candy!");

**Script:**

var candy = prompt("What's your favorite candy?","Type your favorite candy here.");

switch(candy){

case 'licorice':

console.log("Gross!");

break;

case 'gum':

console.log("I like gum!");

break;

case 'beets':

console.log("...is that even a candy?");

break;

// Add your code here!

default:

console.log("Howdy!");

break;

}

**Output:** prompt asking what your favorite candy is then gives you a message

I like gum!

**More practice with switch**

You know what they say: practice makes perfect!

We've given you the empty skeleton of a switch statement. Complete the existing case, then add at least one additional case and a default behavior with whatever console.log() calls you like.

**Script:**

var answer = prompt("Add your question here!");

switch(answer) {

case 'one':

console.log("Is that a question?");

break;

// Add your code here!

case 'two':

console.log("Really, REALLY?");

break;

default:

console.log("Tell it to someone else.");

break;

}

**Output:** prompt asking you to type in a question, then gives you a response message like

Tell it to someone else.

**All on your own**

Great work! Now it's time to put a switch statement together all on your lonesome.

Create your own switch statement in the editor. It can do anything you like! Make sure to include at least three cases and a default.

Remember, your syntax looks like this:

switch(condition) {

case 'abc':

// Thing to do

break;

case 'def':

// Another thing

break;

case 'ghi':

// Yet another thing

break;

default:

// Default thing

break;

**Script:**

var say = prompt("What do you want to say?");

switch(say) {

case 'What up!':

console.log("What up with you!");

break;

case 'Howdy':

console.log("Hello");

break;

case 'So':

console.log("So what!");

break;

default:

console.log("Who cares.");

break;

}

**Output:** prompt asking you what you want to say then gives you a response like

Who cares.

**Overview**

So far we've seen how to control our programs given a single condition: whether one variable is equal to a certain value, for instance. But what if we want to check *more* than one variable?

For this, we'll need **logical operators**. JavaScript has three: **and** (&&), **or** (||), and **not** (!).

Using these, we can check several variables at once! Check out the code in the editor.

Starting to make sense? Set both variables in the editor to true and hit Save & Submit Code to see what happens!

**Script:**

var iLoveJavaScript = true;

var iLoveLearning = true;

if(iLoveJavaScript && iLoveLearning) {

console.log("Awesome! Let's keep learning!");

} else if(!iLoveJavaScript || !iLoveLearning) {

console.log("Let's see if we can change your mind.");

} else {

console.log("You only like one but not the other? We'll work on it.");

}

**Output:** Awesome! Let's keep learning!

**And**

The logical operator **and** is written in JavaScript like this: &&. It evaluates to true when *both* expressions are true; if they're not, it evaluates to false.

true && true; // => true

true && false; // => false

false && true; // => false

false && false; // => false

Create two variables, hungry and foodHere, and set them both equal to true. Inside the eat function, create an if statement that returns true only if both hungry and foodHere are true, and false otherwise. The function eat should take no input and hungry and foodHere should both be globals.

**Script:**

// Declare your variables here

var hungry = true;

var foodHere = true;

var eat = function() {

// Add your if/else statement here

if(hungry && foodHere === true) {

return true;

} else {

return false;

}

};

eat(hungry,foodHere);

**Output:** true

**Or**

The logical operator **or** is written in JavaScript like this: ||. It evaluates to true when *one or the other or both* expressions are true; if they're not, it evaluates to false.

true || true; // => true

true || false; // => true

false || true; // => true

false || false; // => false

The **or** operator is written with two vertical bars ||. The vertical bar character is located right above the Enter key on your keyboard.

Create two variables, tired and bored, and set one equal to true and the other equal to false. (It doesn't matter which is which.) Inside the nap function, create an if statement that returns true if either tired or bored (or both!) are true, and false otherwise.

**Script:**

// Declare your variables here

var tired = true;

var bored = false;

var nap = function() {

// Add your if/else statement here

if (tired || bored) {

return true;

} else {

return false;

}

};

nap(tired, bored);

**Output:** true

**Not**

The logical operator **not** is written in JavaScript like this: !. It makes true expressions false, and vice-versa.

!true; // => false

!false; // => true

Declare a variable called programming and set it to false. Then, write an if/else statement inside happy so that happy returns true if programming is false and false otherwise.

Remember: !false is true. Even though programming = false, you can put an expression that evaluates to true (using !) in your if() statement.

**Script:**

// Declare your variables here

var programming = false;

var happy = function() {

// Add your if/else statement here

if(!programming){

return true;

} else {

return false;

}

};

happy(programming);

**Output:** true

**Code Your Own Adventure 2!**

**What you'll be building**

We told you this was just the beginning of our game empire! Now that you know more about JavaScript, you'll be able to create a much richer "choose your own adventure" game.

Play through the game in the editor a couple of times. Neat, right? Head on to the next section when you're ready to start building your own.

**Script:**

var troll = prompt("You're walking through the forest, minding your own business, and you run into a troll! Do you FIGHT him, PAY him, or RUN?").toUpperCase();

switch(troll) {

case 'FIGHT':

var strong = prompt("How courageous! Are you strong (YES or NO)?").toUpperCase();

var smart = prompt("Are you smart?").toUpperCase();

if(strong === 'YES' || smart === 'YES') {

console.log("You only need one of the two! You beat the troll--nice work!");

} else {

console.log("You're not strong OR smart? Well, if you were smarter, you probably wouldn't have tried to fight a troll. You lose!");

}

break;

case 'PAY':

var money = prompt("All right, we'll pay the troll. Do you have any money (YES or NO)?").toUpperCase();

var dollars = prompt("Is your money in Troll Dollars?").toUpperCase();

if(money === 'YES' && dollars === 'YES') {

console.log("Great! You pay the troll and continue on your merry way.");

} else {

console.log("Dang! This troll only takes Troll Dollars. You get whomped!");

}

break;

case 'RUN':

var fast = prompt("Let's book it! Are you fast (YES or NO)?").toUpperCase();

var headStart = prompt("Did you get a head start?").toUpperCase();

if(fast === 'YES' || headStart === 'YES') {

console.log("You got away--barely! You live to stroll through the forest another day.");

} else {

console.log("You're not fast and you didn't get a head start? You never had a chance! The troll eats you.");

}

break;

default:

console.log("I didn't understand your choice. Hit Run and try again, this time picking FIGHT, PAY, or RUN!");

}

**Output:** prompts you to make choices then gives you results.

**Prompt**

First, we'll need to use a prompt statement to ask our user what he or she wants to do. Recall that we use prompt like this:

var answer = prompt("Question to the user");

We store the result of using prompt in a variable so we can use the user's response to influence what the program does.

Prompt the user for input and store it in a variable called user. You can ask anything!

Why not try:

var user = prompt("What's your name?");

**Script:**

var question = prompt("What would you like to do?");

var user = prompt("What's your name?");

**Output:** prompts pop up

**.toUpperCase() and .toLowerCase()**

You may have noticed us use the .toUpperCase() function in the first exercise. We used it like this:

var answer = prompt("Question to the user").toUpperCase();

This converted the user's answer to ALL CAPS before saving it in the answer variable. This helps eliminate problems that might crop up if your program tests for 'YES' but your user typed in 'yes' or 'Yes'. The input becomes all caps before we test, so we only have to test for all caps!

You can also use .toLowerCase(), which converts a string to all lower-case letters.

Call either .toUpperCase() or .toLowerCase() on your prompt to ensure that the input you get from the user is capitalized the way you expect.

**Script:**

var question = prompt("What would you like to do?").toUpperCase();

**Outputs:** makes prompt answer all upper case

**Switch**

Great work! Now let's get to the heart of our game: the switch statement.

After your prompt, add a switch statement that will test for several different cases (that is, different possible user inputs). Create as many as you like! (Do at least three.) Don't forget to include a default block at the end that will provide a response if the user's choice doesn't match one of your cases.

Remember, switch works like this:

var user = prompt("Name?");

switch(user) {

case 'Buster':

console.log("Hey, brother!");

break;

case 'GOB':

console.log("I've made a huge mistake.");

break;

case 'Steve':

console.log("Steve Holt!");

break;

default:

console.log("I don't know you!");

**Script:**

var question = prompt("What would you like to do?").toUpperCase();

switch(question) {

case 'YES':

console.log("Then do it!!");

break;

case 'NO':

console.log("Lazy!");

break;

case 'WHAT':

console.log("Say again!");

break;

default:

console.log("Try again.");

break;

}

**Output:** prompts you and then base on your answer gives a response

**Logical operators**

Good! Now let's spice things up with some logical operators.

Add some if/else statements to your cases that check to see whether one condition **and** another condition are true, as well as whether one condition **or** another condition are true. Use && and || at least one time each.

Here's how you can nest an if / else statement inside a case:

switch(month) {

case 0:

if(day == 1) {

console.log("Happy New Year");

}

else {

console.log("Happy January");

}

break;

/\* ... \*/

}

**Script:**

var dragon = prompt("Do you want to fight, run, or tame the dragon?").toUpperCase();

switch (dragon) {

case 'FIGHT':

var sword = prompt("Do you have a strong sharp (YES or NO)?").toUpperCase();

var shield = prompt("Do you have a sturdy shield?").toUpperCase();

if(sword === 'YES' && shield === 'YES') {

console.log("You have slayed the dragon and live to tell your heroic story!");

} else {

console.log ("You fought the dragon without the proper weapon. The dragon ate you");

}

break;

case 'RUN':

var fast = prompt("So you think you can out run the dragon? Are you fast (YES or NO)?").toUpperCase();

var agile = prompt("Are you agile?").toUpperCase();

if(fast === 'YES' && agile === 'YES'){

console.log("You got away!");

} else {

console.log("The dragon picked you up and mid flight and took you back to it's nest for dinner.");

}

break;

case 'TAME':

var smart = prompt("Are you smart (YES or NO)?").toUpperCase();

var patience = prompt("Do you have patience?").toUpperCase();

if(smart === 'YES' || patience === 'YES'){

console.log("You tamed the dragon now he is your friend!");

} else {

console.log("That didn't go so good the dragon ate you!");

}

break;

default:

console.log("Sorry please only choose fight, run or tame.");

}

**Output:** prompts you and then base on your answer gives a response

**Arrays and Objects in JS**

**You know this!**

You already know about arrays, so we won't spend a lot of time going over them. Go ahead and [check out the past exercises](http://www.codecademy.com/courses/javascript-beginner-en-NhsaT/2?curriculum_id=502d635ceda910000200293b#!/exercises/0) on arrays if you need a refresher.

Create an array called list in the editor. It can contain any data you want! Make sure it has at least three elements.

**Script:**

var list = [1, 2, 3, 4];

**Output:** nothing

**Access by offset**

Good! Do you remember how to access an element of an array by offset (with the [] notation)? Check the Hint if you need help.

Log the third element of the languages array to the console. Make sure to do this by accessing the third element's index.

Remember, you can access an array's element like this:

var element = array[index];

If you want the first element of languages, you'd type

languages[0];

because arrays start counting their elements at 0.

**Script:**

var languages = ["HTML", "CSS", "JavaScript", "Python", "Ruby"];

console.log(languages[2]);

**Output:** Javascript

**Array properties**

Good work! If you remember, arrays have a property in common with strings: they can both use .length. When you call .length on an array, it returns the number of elements that array has.

Under your existing code, log the number of elements in languages to the console.

Your code should look something like this:

console.log(languages.length);

**Script:**

var languages = ["HTML", "CSS", "JavaScript", "Python", "Ruby"];

console.log(languages.length);

**Output:** 5

**Iterating over an array**

By combining all these ideas with a for loop, you can iterate over the languages array and print out each element in turn!

Go ahead and use a for loop to log each element of the languages array to the console.

Click "Stuck? Get a hint!" if you need a reminder of for loop syntax.

Your for loop should look like this:

for (var i = 0; i < languages.length; i++) {

// console.log the 'i'th element

// of the languages array

}

You should always include curly braces around the body of your for loop, even if it's only one line.

**Script:**

var languages = ["HTML", "CSS", "JavaScript", "Python", "Ruby"];

for(var i=0; i <= languages.length; i++){

console.log(languages[i]);

}

**Output:**

HTML

CSS

JavaScript

Python

Ruby

undefined

**Heterogeneous arrays**

Now that we've reviewed some array basics, it's time to cover a little new ground.

First, it's not necessary for you to put the same type of data in an array! For instance, you don't have to have

var pronouns = ["I", "you", "we"];

var numbers = [1, 2, 3];

You can have a **heterogeneous array**, which means a mixture of data types, like so:

var mix = [42, true, "towel"];

Create a heterogeneous array called myArray with at least three elements. The first element should be a number, the second should be a boolean (true or false), and the third should be a string. Feel free to add more elements of any type if you like!

**Script:**

var myArray = [1, true, "test"];

**Output:** nothing

**Arrays of arrays**

Good! The next thing to know is that not only can you put a mixture of types in an array, you can even put *other arrays* inside arrays. You can make a **two-dimensional array** by nesting arrays one layer deep, like so:

var twoDimensional = [[1, 1], [1, 1]];

This array is two-dimensional because it has two rows that each contain two items. If you were to put a new line between the two rows, you could log a 2D object—a square—to the console, like so:

[1, 1]

[1, 1]

Create a two-dimensional array called newArray in the editor. It should have three rows and three columns containing any data you like. Check the Hint if you need help!

To get three rows, you have to make sure to put three arrays inside of newArray. To get three columns, each of those three arrays should have three elements!

**Script:**

var newArray =[[1, 1, 1], [1, 1, 1], [1, 1, 1]];

**Output:** nothing

**Jagged arrays**

Great work! That's a fine-looking multidimensional array you've got there. (Yours is nested once, so it's a two-dimensional array, but if you really wanted, you could put arrays inside arrays inside arrays for even more dimensions.)

Sometimes you want arrays that aren't as nice and even as your 3 x 3 two-dimensional array: you may have three elements in the first row, one element in the second row, and two elements in the third row. JavaScript allows those, and they're called **jagged arrays**.

Create a jagged array called jagged. You can place whatever you like in it! The only requirement is that it have at least two rows (that is, the first two elements need to be arrays), and those rows cannot be the same length.

An example of a jagged array might be:

[[1,2],[3]];

Since there are two elements in the first nested array but only one element in the second nested array.

**Script:**

var jagged = [[1, 2, 3], [1]];

**Output:** nothing

**Nouns and verbs together**

Let's go back to the analogy of computer languages being like regular spoken languages. In English, you have nouns (which you can think of as "things") and verbs (which you can think of as "actions"). Until now, our nouns (data, such as numbers, strings, or variables) and verbs (functions) have been separate.

No longer!

Using **object**s, we can put our information and the functions that use that information *in the same place*.

You can also think of objects as combinations of key-value pairs (like arrays), only their keys don't have to be numbers like 0, 1, or 2: they can be strings and variables.

**Script:**

var phonebookEntry = {};

phonebookEntry.name = 'Oxnard Montalvo';

phonebookEntry.number = '(555) 555-5555';

phonebookEntry.phone = function() {

console.log('Calling ' + this.name + ' at ' + this.number + '...');

};

phonebookEntry.phone();

**Output:** Calling Oxnard Montalvo at (555) 555-5555...

**Object syntax**

Did you see that? The phonebookEntry object handled data (a name and a telephone number) as well as a procedure (the function that printed who it was calling).

In that example, we gave the **key** name the **value** 'Oxnard Montalvo' and the key number the value '(555) 555-5555'. An object is like an array in this way, except its keys can be variables and strings, not just numbers.

Objects are just collections of information (keys and values) between curly braces, like this:

var myObject = {

key: value,

key: value,

key: value

};

Using the above syntax as a guide, create an object, me, in the editor. It should have a name key with the value of your name (as a string) and an age key with the value of your age (as a number).

Your code should look something like this:

var me = {

name: 'Eric',

age: 26

};

**Script:**

var me = {

name: "Matt",

age: 42

};

**Output:** nothing

**Creating a new object**

Great work! You just created your very first object.

There are two ways to create an object: using **object literal notation** (which is what you just did) and using the **object constructor**.

Literal notation is just creating an object with curly braces, like this:

var myObj = {

type: 'fancy',

disposition: 'sunny'

};

var emptyObj = {};

When you use the constructor, the syntax looks like this:

var myObj = new Object();

This tells JavaScript: "I want you to make me a new thing, and I want that thing to be an Object.

You can add keys to your object after you've created it in two ways:

myObj["name"] = "Charlie";

myObj.name = "Charlie";

Both are correct, and the second is shorthand for the first. See how this is sort of similar to arrays?

Recreate your me object in the editor, but this time, use the object constructor. Once you make it, use either the [] notation or the . notation to give it a name property with a string value (your name) and an age property with a number value (your age).

Remember, JavaScript is case-sensitive—it cares about capitalization!

Note that you will see a little yellow triangle warning that tells you to use "object literal notation". Generally, object literal notation is considered preferable, but it is important that you learn both ways so that you recognize them in other people's code.

**Script:**

var me = new Object();

me.name = "Matt";

me.age = 42;

console.log(me);

**Output:** { name: 'Matt', age: 42 }

**Practice makes perfect**

Great work! Let's make a few more objects, just for practice.

Create three objects called object1, object2, and object3 in the editor. Use either literal notation or the object constructor, and give your objects any properties you like!

Literal syntax:

var myObj = {

key1: value,

key2: value

};

var myObj = {};

myObj.key1 = value;

myObj['key2'] = value;

Object constructor:

var myObj = new Object();

myObj.key1 = value;

myObj['key2'] = value;

**Script:**

var object1 = {

name: 'Matt'

};

var object2 = new Object();

var object3 = new Object();

object1.name = 'Matt';

object2.car = "Honda";

object3.year = 2003;

**Output:** 2003

**Heterogeneous arrays**

Let's warm up with some arrays! Let's make an array that's a veritable *potpourri* of data types.

Create an array, myArray. Its first element should be a number, its second should be a boolean, its third should be a string, and its fourth should be... an object! You can add as many elements of any type as you like after these first four.

[Booleans](http://www.codecademy.com/courses/getting-started-v2/1/6) can be true or false. To add a boolean to an array, we could do:

var bool1 = true;

myArray = [bool1];

To add an object to an array, we could do:

var myObj = {

type: 'fancy',

disposition: 'sunny'

};

myArray = [myObj];

**Script:**

var myArray = [1, true, "Hello", ['name']];

**Output:** nothing

**Multidimensional arrays**

Good! Now let's create a 2D array. Not only that, but a 2D array that's *jagged*. Remember, that means it's an array of arrays, and its nested arrays aren't all the same length! For example:

var aList = [ [1, 4, 2], [7] ];

Make an array called newArray. It should be 2D (that is, it should contain two elements that, in turn, are arrays) and jagged (those two arrays should be of different lengths, like in the example above).

The ultimate kicker? Make one of your inner arrays contain an object!

To add an object to an array, we could do:

var myObj = {

type: 'fancy',

disposition: 'sunny'

};

myArray = [myObj];

**Script:**

var newArray = [[1,2], [1,2, [3]], [1,2,"Matt"]];

**Output:** nothing

**Editing an existing object**

Nice work! Now let's do a little work with objects. We'll start by modifying an existing one.

Add a key called interests to myObject. Give this key an array value (the array can contain whatever you like).

Use the object syntax you see in the editor as a guide for adding your own key: value.

**Script:**

var myObject = {

name: 'Eduardo',

type: 'Most excellent',

// Add your code here!

interests: ['Art']

};

**Output:** nothing

**Creating your own objects**

You're almost there! Last step: forge your very own object in the fires of Mount JavaScript.

Create your own object called myOwnObject. Give it whatever properties you like! (Be sure to give it at least one.) You can use either literal notation or the object constructor.

**Script:**

var myOwnObject = new Object();

myOwnObject.name = "Matt";

myOwnObject.age = 42;

**Output:** 42

**Contact List**

**What you'll be building**

In this project, we'll combine our knowledge of objects and arrays to create a simple contact list. Then, using functions, we'll be able to log the entries in our contact list to the console, as well as search for a particular entry.

**Script:**

var friends = {};

friends.bill = {

firstName: "Bill",

lastName: "Gates",

number: "(206) 555-5555",

address: ['One Microsoft Way','Redmond','WA','98052']

};

friends.steve = {

firstName: "Steve",

lastName: "Jobs",

number: "(408) 555-5555",

address: ['1 Infinite Loop','Cupertino','CA','95014']

};

var list = function(obj) {

for(var prop in obj) {

console.log(prop);

}

};

var search = function(name) {

for(var prop in friends) {

if(friends[prop].firstName === name) {

console.log(friends[prop]);

return friends[prop];

}

}

};

list(friends);

search("Steve");

**Output:**

bill

steve

{ firstName: 'Steve',

lastName: 'Jobs',

number: '(408) 555-5555',

address: [ '1 Infinite Loop', 'Cupertino', 'CA', '95014' ] }

{"firstName":"Steve","lastName":"Jobs","number":"(408) 555-5555","address":["1 Infinite Loop","Cupertino","CA","95014"]}

**Creating your contact object**

First, we need to start with an object to hold our friends.

Create an object called friends. Feel free to use either object literal notation or the object constructor. Check the Hint if you need a syntax reminder.

Object literal:

var objectName = {};

Object constructor:

var objectName = new Object();

**Script:**

var friends = new Object();

**Output:** nothing

**Adding your friends**

Good! Now let's add some friends to our friends object.

Each friend will need a name, phone number, and so on. We will use a new object for each friend so that we can remember their information! That's right, we'll have objects within objects!

Add a few empty objects to your friends object. Make sure you add a friend named 'bill' and a friend named 'steve'. Use your friends' names as the keys for the empty objects.

You can add objects directly to friends, like this:

var friends = {

bill: {},

steve: {}

};

Or with the bracket ([]) or dot(.) notation, like this:

friends[bill] = {};

friends.steve = {};

Or with Object constructors, like this:

var friends = new Object();

friends.bill = new Object();

friends.steve = new Object();

Notice that "bill" and "steve" are not capitalized!

**Script:**

var friends = new Object();

var bill = new Object();

bill.name = "Bill";

bill.phone = 703123444;

var steve = new Object();

steve.name = "Steve";

steve.phone = 7301118888;

friends.bill = {};

friends.steve = {};

**Output:** {}

**Adding properties**

Next, let's add some properties to our friends. We'll want them to have a firstName, a lastName, and a number.

Give each of your friends a firstName, lastName, and number. The value for each of these should be a string (check the Hint if you need help). Make sure bill's first name is "Bill" and steve's first name is "Steve" (again, note the capitalization).

Here's an example:

var friends = {

bill: {

firstName: "Bill",

lastName: "Gates",

number: "(206) 555-5555"

}

};

**Script:**

var friends = new Object();

friends.bill = {

firstName: "Bill",

lastName: "Gates",

number: "(206) 555-5555"

};

friends.steve = {

firstName: "Steve",

lastName: "Jobs",

number: "(206) 555-5555"

};

**Output:** {"firstName":"Steve","lastName":"Jobs","number":"(206) 555-5555"}

**Tossing in an array**

Let's add another property to each of our friends and set it equal to an array. Give each of your friends an address property, and break up their address into an array, like so:

var friends = {

bill: {

firstName: "Bill",

lastName: "Gates",

number: "(206) 555-5555",

address: ['One Microsoft Way','Redmond','WA','98052']

}

};

Add an address property to each of your friends and set that property equal to an array value.

**Script:**

var friends = new Object();

friends.bill={

firstName: "Bill",

lastName: "Gates",

number: "785-922",

address: ['One Microsoft Way','Redmond','WA','98052']

};

friends.steve={

firstName: "Steve",

lastName: "Jobs",

number: "579-275",

address: ['One Microsoft Way','Redmond','WA','98052']

};

**Output:**

{"firstName":"Steve","lastName":"Jobs","number":"579-275","address":["One Microsoft Way","Redmond","WA","98052"]}

**List 'em all!**

Great work! Now let's add a couple of functions to help us go through our contacts.

The first function we'll create will be called list, and it will print out all the entries we have in our friends object. To do this, we'll want to use a bit of new syntax: a for/in loop.

It looks like this:

for (var key in object) {

// Access that key's value

// with object[key]

}

The "key" bit can be any placeholder name you like. It's sort of like when you put a placeholder parameter name in a function that takes arguments.

1. Create a function list that takes a single parameter.
2. In the body of the function, write a for/in loop.
3. In the loop, use console.log to print out the key. (For example, if you only have bill and steve as entries, list should just print out "bill" and "steve".)

Here's how to create a function:

var list = function (friends) {

// Do something here

}

Here is how to loop over the keys of an object:

for (var key in object) {

// Use object[key] to access

// the corresponding value

}

Here's how to use console.log:

console.log("Printing a string");

var myString = "Awesome!";

console.log(myString);

**Script:**

var friends = {};

friends.bill = {

firstName: "Bill",

lastName: "Gates",

number: "(206) 555-5555",

address: ['One Microsoft Way','Redmond','WA','98052']

};

friends.steve = {

firstName: "Steve",

lastName: "Jobs",

number: "(408) 555-5555",

address: ['1 Infinite Loop','Cupertino','CA','95014']

};

var list = function(friends) {

for(var key in friends) {

console.log(key);

}

};

/\*var search = function(name) {

for(var prop in friends) {

if(friends[prop].firstName === name) {

console.log(friends[prop]);

return friends[prop];

}

}

};\*/

list(friends);

/\* search("Steve"); \*/

**Output:**

bill

steve

bill

steve

**Search for a friend**

The second function we'll add will be called search, and it will take a first name as an argument. It will try to match the first name it receives to any of the first names in our friends contact list. If it finds a match, it will log our friend's contact information (firstName, lastName, number, address) to the console.

Define a function search that takes a single argument, name. If the argument passed to the function matches any of the first names in friends, it should log that friend's contact information to the console **and** return it.

Here's an outline of the function

First, create a function named search. Here's how to create a function:

var search = function(friends) {

// Do something here

};

Inside the function, loop over the keys of the friends object. Here's how to loop over the keys of an object:

for (var key in object) {

// Use object[key] to access

// the corresponding value

}

Inside the for loop, use an if/else statement to check whether the argument passed to the function === any of the first names in friends. Here's how to use an if/else statement:

if(condition) {

// Do one thing here

}

else {

// Do another thing here

}

If there's a match, use console.log to print that friend to the screen, and return it. Here's how to use console.log

console.log("Printing a string");

var myString = "Awesome!";

console.log(myString);

Your code should look something like this:

var search = function(name) {

for(var key in friends) {

if(friends[key].firstName === name) {

console.log(friends[key]);

return friends[key];

}

}

};

**Script:**

var friends = {};

friends.bill = {

firstName: "Bill",

lastName: "Gates",

number: "(206) 555-5555",

address: ['One Microsoft Way','Redmond','WA','98052']

};

friends.steve = {

firstName: "Steve",

lastName: "Jobs",

number: "(408) 555-5555",

address: ['1 Infinite Loop','Cupertino','CA','95014']

};

var list = function(obj) {

for(var prop in obj) {

console.log(prop);

}

};

var search = function(name) {

for(var prop in friends) {

if(friends[prop].firstName === name) {

console.log(friends[prop]);

return friends[prop];

}

}

};

list(friends);

search("Steve");

**Output:**

bill

steve

{ firstName: 'Steve',

lastName: 'Jobs',

number: '(408) 555-5555',

address: [ '1 Infinite Loop', 'Cupertino', 'CA', '95014' ] }

{ firstName: 'Steve',

lastName: 'Jobs',

number: '(408) 555-5555',

address: [ '1 Infinite Loop', 'Cupertino', 'CA', '95014' ] }

{ firstName: 'Steve',

lastName: 'Jobs',

number: '(408) 555-5555',

address: [ '1 Infinite Loop', 'Cupertino', 'CA', '95014' ] }

{ firstName: 'Bill',

lastName: 'Gates',

number: '(206) 555-5555',

address: [ 'One Microsoft Way', 'Redmond', 'WA', '98052' ] }

{"firstName":"Steve","lastName":"Jobs","number":"(408) 555-5555","address":["1 Infinite Loop","Cupertino","CA","95014"]}

**Victory!**

Great work! You've created your own list of contacts that you can search through. Try calling both your functions, adding additional friends, changing the fields you include for each friend, and more to customize your new object.

We did some basic logging of your contact list to the console, but we could have made it look even nicer. How might you format the output to look like this?

First Name: Steve

Last Name: Jobs

Number: (408) 555-5555

Address: 1 Infinite Loop

Cupertino, CA 95014

**Script:**

var friends = {};

friends.bill = {

firstName: "Bill",

lastName: "Gates",

number: "(206) 555-5555",

address: ['One Microsoft Way','Redmond','WA','98052']

};

friends.steve = {

firstName: "Steve",

lastName: "Jobs",

number: "(408) 555-5555",

address: ['1 Infinite Loop','Cupertino','CA','95014']

};

var list = function(obj) {

for(var prop in obj) {

console.log(prop);

}

};

var search = function(name) {

for(var prop in friends) {

if(friends[prop].firstName === name) {

console.log(friends[prop]);

return friends[prop];

}

}

};

list(friends);

search("Steve");

**Output:**

bill

steve

{ firstName: 'Steve',

lastName: 'Jobs',

number: '(408) 555-5555',

address: [ '1 Infinite Loop', 'Cupertino', 'CA', '95014' ] }

{"firstName":"Steve","lastName":"Jobs","number":"(408) 555-5555","address":["1 Infinite Loop","Cupertino","CA","95014"]}

**Introduction to Objects I**

**We've come a long, long...**

*If yellow triangle warnings appear in the editor next to any code we provide in any exercise, it is fine to ignore them.*

The very basic building block of JavaScript are primitive data types. We know of three primitives:

* **strings** (*e.g.* "dogs go woof!")
* **numbers** (*e.g.* 4, 10)
* **booleans** (*e.g.* false, 5 > 4)

We learned about the use of comparators (eg. >, <=, !==, etc.). One really important thing to note is that any time comparisons are made, a Boolean value is returned.

There is a long and ugly expression in the editor. Overall, it evaluates to a Boolean (i.e., either the entire statement is true, or it is false).

What does this expression in the editor evaluate to?

Declare a variable named answer. Assign to it the Boolean value that the expression evaluates to. Delete the default code in the editor and run your code.

**Focus on** each element of the expression separated by the || ("OR") operator. You will see that there are three smaller expressions. Evaluate those smaller expressions first, then use them use three Booleans and determine the overall expression's Boolean value.

For each of the smaller expressions, ask yourself these questions:

* Does 3 \* 90 equal 270?
* What's the opposite of false AND not-false?
* If we turn "bex" into all-uppercase letters, will it be the same as "BEX"?

Since we're dealing with the || ("OR") operator here, if the first statement OR the second statement OR the third statement is true, the entire statement is true, and the value of answer should be true.

**Script:**

(((3 \* 90) === 270) || !(false && (!false)) || "bex".toUpperCase() === "BEX");

var answer = true;

**Output:** true

**Through the hard times...**

We know two ways of storing data types. We can use variables or arrays. We use variables to store data (like strings or numbers) that we’d later want to access.

An array is exactly the same as a variable in that it stores data. The difference is that an array can store many more values while a variable can only store one.

To access arrays, we use bracket notation and remember that arrays use 0-based indexing (i.e., the first value in an array is at position 0).

Look at the array multiplesOfEight, and find the one that doesn't fit.

Replace X in [line 6](javascript:void(0)) such that the variable answer is assigned the Boolean value of true.

Array indexes start at 0. So the first element of an array has index 0.

For example, the first element of multiplesOfEight is 8, and its index is 0. The third element of multiplesOfEight is 24, and its index is 2.

The operator !== means "does NOT equal". For example,

10 !== 5;

evaluates to true because 10 does not equal 5.

**Script:**

// Here is an array of multiples of 8. But is it correct?

var multiplesOfEight = [8,16,24,32,40,58];

// Test to see if a number from the array is NOT a true

// multiple of eight. Real multiples will return false.

var answer = multiplesOfEight[8] % 8 !== 0;

**Output:** nothing

**...And the good!**

We're going to play a game of FizzBuzz. The rules are simple. We want to count from 1 to 20. But if the number is divisible by 3, we're going to print "Fizz". And if the number is divisible by 5 we're going to print "Buzz".

What will we print if the number is divisible by 3 AND 5? That's right! "FizzBuzz"!

There are many ways to do this, but we'd like you to use a nested conditional in this exercise.

1. Print out the numbers from 1 - 20.
2. The rules:
   * For numbers divisible by 3, print out "Fizz".
   * For numbers divisible by 5, print out "Buzz".
   * For numbers divisible by both 3 and 5, print out "FizzBuzz" in the console.
   * Otherwise, just print out the number.
3. Use a for loop to iterate through the numbers 1 to 20.
4. Use an if / else to check if the number is divisible by 3 or 5
5. Log to the console either "Fizz", "Buzz", or "FizzBuzz".

Here is how to use an if/else statement:

if(condition) {

// Do one thing here

}

else {

// Do another thing here

}

**Script:**

for (i=1; i<=20; i++){

if(i % 3 === 0 ){

if (i % 5 === 0) console.log("FizzBuzz")

else console.log("Fizz");

} else if (i % 5 === 0){

console.log("Buzz");

} else {

console.log(i);

}

}

**Output:**

1

2

Fizz

4

Buzz

Fizz

7

8

Fizz

Buzz

11

Fizz

13

14

FizzBuzz

16

17

Fizz

19

Buzz

**I have to celebrate you baby**

This exercise has lots of movies and reviews to type in. You might wonder, "Is this teaching coding or typing?!"

But there's a reason why there are so many cases to deal with. We want to show that if we used if-else statements, it would be inefficient. What alternative to if / else can we use?

Imagine you have a movie collection, and you want to write code that returns your review for each one. Here are the movies and your reviews:

* "Toy Story 2" - "Great story. Mean prospector."
* "Finding Nemo" - "Cool animation, and funny turtles."
* "The Lion King" - "Great songs."

Write a function named getReview that takes in a movie name and returns its review based on the information above. If given a movie name not found just return "I don't know!". Use a structure learned in an earlier lesson (NOT if/else statements) to write this function.

Try using a switch statement. Here's an example:

switch(movie) {

case "Toy Story 2":

// return review here

case "Finding Nemo":

// return review here

default:

// code to be executed if a movie name is not found

}

Because we are defining a function, we can make use of the return keyword!

Make sure that what you return matches the case of the review text

**Script:**

var getReview = function (movie) {

switch (movie){

case "Toy Story 2":

return "Great story. Mean prospector.";

break;

case "Finding Nemo":

return "Cool animation, and funny turtles.";

break;

case "The Lion King":

return "Great songs.";

break;

case "Matrix":

return "good trip out";

break;

case "Princess Bride":

return "awesome date night movie";

break;

case "Welcome to America":

return "Amjad's favorite";

break;

case "Remember the Titans":

return "love the sports";

break;

case "Why do I look like I'm 12?":

return "The Ryan and Zach story";

break;

case "Fighting Kangaroos in the wild":

return "Token Australian movie for Leng";

break;

default:

return "I don't know!";

break;

}

};

getReview("Toy Story 2");

**Output:** "Great story. Mean prospector."

**I have to praise you like I should!**

Congratulations for making it this far! We hope you're enjoying the courses and feel more comfortable programming in JavaScript.

We have a number of exciting things in store for you and can't wait to roll them out!

**Script:**

// any topics you want us to review, email contact@codecademy.com

// if you want to see the winning joke for this week, see the hint!

// if you think you have a better joke, send it in!

console.log("I'm ready for Objects!");

**Output:** I'm ready for Objects!

**Intro**

We have discussed four data types: numbers, strings, booleans and arrays.

In this lesson, we focus on a fifth data type: **objects**. This data type is a little bit more complex. Objects allow us to represent in code real world things and entities (such as a person or bank account). We do this by storing all relevant information in one place—an object.

How do we create an object? Like declaring a variable, or defining a function, we use var, followed by the name of the object and an equals sign. Each object then:

1. starts with {
2. has information inside
3. ends with };

Create an object called bob that has no information inside the brackets.

**We can** make an empty object similar to how we made variables, by using var myVar = { };. For this exercise, make an empty object named bob.

**Script:**

var bob = {};

**Output:** nothing

**Properties**

Let's review what we previously covered. Each piece of information we include in an object is known as a **property**. Think of a property like a **category label** that belongs to some object. When creating an object, each property has a name, followed by : and then the **value** of that property. For example, if we want Bob's object to show he is 34, we'd type in age: 34.

age is the property, and 34 is the value of this property. When we have more than one property, they are separated by **commas**. The last property does not end with a comma.

See the console for the object I have created about myself. Can you create an object called me that describes your age and which country you live in?

Use Spencer as a guide to make your own object called me. Fill in your own age and country properties.

Don't forget the comma between the two properties!

**Script:**

var Spencer = {

age: 22,

country: "United States"

};

// make your own object here called Me

var me = {

age: 30,

country: "United States"

};

**Output:** nothing

**Accessing Properties**

Now that we know how to make objects with properties, let's look at how we actually use them!

Notice our example objects bob and susan. In this case both bob and susan each have two properties, name and age.

After creating our objects we have added code to access these properties. Notice that we save bob's name, "Bob Smith", into the global variable name1. We do this in [line 10](javascript:void(0)).

When we typed in var name1 = bob.name; we are doing many things.

1. Declaring a new variable called name1
2. Assigning some value to this variable
3. That value is the value of bob.name (i.e., the value associated with the name property in the object bob).

Try following the same three steps for the variables name2 and age2.

**Script:**

var bob = {

name: "Bob Smith",

age: 30

};

var susan = {

name: "Susan Jordan",

age: 25

};

// here we save Bob's information

var name1 = bob.name;

var age1 = bob.age;

// finish this code by saving Susan's information

var name2 = susan.name;

var age2 = susan.age;

**Output:** nothing

**Accessing Properties, Part 2**

In the last exercise, we accessed properties using what is known as **dot notation**. Good name, right? So to access a property, we use ObjectName.PropertyName (e.g., bob.name)

In addition to dot notation, we can also access properties using **bracket notation**. In this case we use ObjectName["PropertyName"] to access the desired property. Note, we need " " around the property's name.

Take a look at our next example object called dog. Notice on [line 8](javascript:void(0)) how we save the dog's species into a variable by accessing the species property of dog using bracket notation.

Use bracket notation to save the dog's weight and age into variables as well.

Refer to [line 8](javascript:void(0)) for how to retrieve the value of an object's property using bracket notation. Your code should look remarkably similar.

Ignore the little yellow triangles. Dot notation is often considered better but it is important that you recognize both so that you can understand other people's code!

**Script:**

// Take a look at our next example object, a dog

var dog = {

species: "greyhound",

weight: 60,

age: 4

};

var species = dog["species"];

// fill in the code to save the weight and age using bracket notation

var weight = dog["weight"];

var age = dog["age"];

**Output:** nothing

**Another Way to Create**

The method we've used to create objects uses **object literal notation**—that is, creating a new object with { } and defining properties within the brackets.

Another way of creating objects without using the curly brackets { } is to use the keyword new. This is known as creating an object using a **constructor**.

The new keyword creates an empty object when followed by Object(). The general syntax is:

var objectName = new Object();

We then have to fill this object with properties and labels. How do we do that? Check out the creation of the object bob to see what we do. We create the name property for the object bob by using bob.name and assigning that to a value. Contrast this to how we define properties in [lines 6-7](javascript:void(0)) for the susan1 object.

Inspect the susan1 object carefully and note the use of object literal notation.

Use constructor notation to create susan2, which should have the same properties and values as susan1.

**Using the** new keyword involves two main steps.

1. Create an empty object. See how we did that for the bob ([line 2](javascript:void(0))).
2. Fill in the object using dot notation and assigning a value to it. For susan2, we have two properties (name and age).

**Script**

// Our bob object again, but made using a constructor this time

var bob = new Object();

bob.name = "Bob Smith";

bob.age = 30;

// Here is susan1, in literal notation

var susan1 = {

name: "Susan Jordan",

age: 24

};

// Make a new susan2 object, using a constructor instead

var susan2 = new Object();

susan2.name = "Susan Jordan";

susan2.age = 24;

**Output:** 24

**Putting it all together**

We've learned how to make objects in two different ways. Both are valid, and you can use which one you prefer.

Let's practice how to use both one more time.

Use literal notation to finish the snoopy object. Remember literal notation is the one where we fill in { } with separate properties and values with colons. Each property is separated by a comma.

snoopy should have two properties, a species of "beagle" and age of 10.

Then make buddy, a 5 year-old golden retriever, using constructor notation. This notation involves using the key word new to create an empty object. Then we fill it in using dot notation.

**For snoopy,**

1. Start by opening the curly bracket {
2. Fill it in using properties and values
3. Close the curly bracket };

**For buddy,**

1. Create a new object that is empty
2. Use dot notation to assign values to the properties: thisObject.propertyName = "value";

**Script:**

// help us make snoopy using literal notation

// Remember snoopy is a "beagle" and is 10 years old.

var snoopy = {

species: "beagle",

age: 10

};

// help make buddy using constructor notation

// buddy is a "golden retriever" and is 5 years old

var buddy = new Object();

buddy.species = "golden retriever";

buddy.age = 5;

**Output:** 5

**More Practice Making Objects**

Nice job! Let's do one more example to get the hang of making objects with desired properties.

Create an object named 'bicycle' that has 3 properties:

* a speed of 0
* a gear of 1
* a frame\_material of "carbon fiber"

Look back at the object Spencer in exercise 2. The object here will have a similar format, but the properties and values are different.

If you're using literal notation, remember to use : instead of = when defining properties and that commas separate properties in an object.

**Script:**

var BMW = {

cost: "too much",

speed: 220,

country: "Germany"

};

var bicycle = new Object();

bicycle.speed = 0;

bicycle.gear = 1;

bicycle.frame\_material = "carbon fiber";

**Output:** "carbon fiber"

**Function Review**

In this lesson we are going to focus on **methods**. Methods are an important part of object oriented programming (OOP). OOP is an important part of programming which we'll dive into later.

Methods are similar to functions. To prepare for methods, let's do a quick refresher on functions.  
Functions are defined using the function keyword followed by:

1. A pair of parentheses ( ) with optional parameters inside.
2. A pair of curly braces with the function's code inside { }.
3. A semicolon ;.

And when we call the function, we can put inputs (arguments) for the parameters.

For example, the square function on [line 2](javascript:void(0)) takes x as its parameter and returns that parameter squared.

Define the function multiply. It should take two parameters, x and y, and return the product.

Then call your function, passing in any two arguments.

Use square as a guide to write multiply. This time, you will have two parameters, and these should be separated by a comma inside the function's parentheses.

To call the function, just type the name of the function, followed by the arguments you want to pass. For example, if we wanted to call square, we would use square(2);

**Script:**

// Accepts a number x as input and returns its square

var square = function (x) {

return x \* x;

};

// Write the function multiply below

// It should take two parameters and return the product

var multiply = function (x,y){

return x \* y;

};

multiply(4,5);

square(3);

**Output:** 9

**So What's a Method?**

In the last section, we discussed properties. We can think of properties as variables associated with an object. Similarly, a **method** is just like a *function* associated with an object.

Let's look at bob, our same person object from the last lesson. Instead of just having the properties name and age ([line 3](javascript:void(0)) & 4), bob also has a *method* called setAge ([line 6](javascript:void(0))). As you can probably guess, this method sets bob's age to whatever argument you give it.

Notice how we define setAge kind of like we define a property. The big difference is that we put in a function after the equals sign instead of a string or number.

We call a method like a function, but we use ObjectName.methodName(). Look at [line 10](javascript:void(0)) where we use the method to change bob's age to 40. We did this by calling bob.setAge(40);.

Use the method setAge to set bob's age to 20.

Remember, to call functions that take parameters we include them in parentheses. In this case we will call bob.setAge(someNewAge);, and input 20 as our new age.

**Script:**

// here is bob again, with his usual properties

var bob = new Object();

bob.name = "Bob Smith";

bob.age = 30;

// this time we have added a method, setAge

bob.setAge = function (newAge){

bob.age = newAge;

};

// here we set bob's age to 40

bob.setAge(40);

// bob's feeling old. Use our method to set bob's age to 20

bob.setAge(20);

**Output:** nothing

**Why Are Methods Important?**

Methods serve several important purposes when it comes to objects.

1. They can be used to change object property values. The method setAge on [line 4](javascript:void(0)) allows us to update bob.age.
2. They can be used to make calculations based on object properties. Functions can only use parameters as an input, but methods can make calculations with object properties. For example, we can calculate the year bob was born based on his age with our getYearOfBirth method ([line 8](javascript:void(0))).

**Script:**

var bob = new Object();

bob.age = 30;

// this time we have added a method, setAge

bob.setAge = function (newAge){

bob.age = newAge;

};

bob.getYearOfBirth = function () {

return 2012 - bob.age;

};

console.log(bob.getYearOfBirth());

**Output:** 1982

**The "this" Keyword**

Our setAge method works great for bob because it updates bob.age, but what if we want to use it for other people?

It turns out we can make a method work for many objects using a new keyword, this. The keyword this acts as a placeholder, and will **refer to whichever object called that method** when the method is actually used.

Let's look at the method setAge ([line 2](javascript:void(0))) to see how this works. By using the keyword this, setAge will change the age property of any object that calls it. Previously, we had a specific object bob instead of the keyword this. But that limited the use of the method to just bob.

Then when we say bob.setAge = setAge; ([line 9](javascript:void(0))), it means whenever we type bob.setAge( ), this.age in the setAge method will refer to bob.age.

To show this way of making setAge works just like the one in exercise 2, use bob's setAge method to change his age to 50.

Just like in exercise 2, use bob.setAge(someNewAge); to use the method.

**Script:**

// here we define our method using "this", before we even introduce bob

var setAge = function (newAge) {

this.age = newAge;

};

// now we make bob

var bob = new Object();

bob.age = 30;

// and down here we just use the method we already made

bob.setAge = setAge;

// change bob's age to 50 here

bob.setAge(50);

**Output:** nothing

**"This" Works for Everyone**

Great! Now we can take advantage of the fact that the method setAge is not limited to a single object bob—we can reuse the same method for different objects! This allows us to avoid typing out a custom method each time. All because we used the placeholder this.

In the editor, we have the same code as last time, where we define setAge using this. We then set bob.setAge = setAge;. But this time we will reuse the setAge method for susan as well.

Make susan on [lines 11-13](javascript:void(0)), who should initially have an age of 25 and a susan.setAge method also equal to setAge.

Then use susan.setAge(35); to set susan's age to 35.

You can use our definition of bob on [lines 6-8](javascript:void(0)) as a guide to make susan. Then call the susan.setAge(someNewAge); to change susan's age to 35.

**Script:**

// here we define our method using "this", before we even introduce bob

var setAge = function (newAge) {

this.age = newAge;

};

// now we make bob

var bob = new Object();

bob.age = 30;

bob.setAge = setAge;

// make susan here, and first give her an age of 25

var susan = new Object();

susan.age = 25;

susan.setAge = setAge;

susan.setAge(35);

// here, update Susan's age to 35 using the method

susan.age = 35;

**Output:** 35

**Make Your Own Method**

Let's look at a new example and get practice writing methods.

Here we have defined an object rectangle starting on [line 1](javascript:void(0)). It has a two properties, height and width, which represents the height and width of the shape.

We have written a setHeight method which will update rectangle's height to the given parameter. This is very similar to setAge from our person example.

Note we have used the keyword this. this is still a placeholder, but in this scenario, this can only ever refer to rectangle because we defined setHeight to be explicitly part of rectangle by defining it as rectangle.setHeight.

Finish the method setWidth. It should take a parameter newWidth. It will change the property width to the given parameter.

Then use the two methods setHeight and setWidth to change rectangle's height to 6 and width to 8.

Use setHeight ([lines 5-7](javascript:void(0))) as a guide to write setWidth, by updating width instead.

Then call rectangle.setWidth(8); and rectangle.setHeight(6); at the end to update the dimensions.

**Script:**

var rectangle = new Object();

rectangle.height = 3;

rectangle.width = 4;

// here is our method to set the height

rectangle.setHeight = function (newHeight) {

this.height = newHeight;

};

// help by finishing this method

rectangle.setWidth = function (newWidth) {

this.width = newWidth;

};

// here change the width to 8 and height to 6 using our new methods

rectangle.setWidth(8);

rectangle.setHeight(6);

**Output:** nothing

**More Kinds of Methods**

Let's look at another method that calculates useful information about an object.

Here we have an object square with a sideLength property to represent the length of the square's side. This time, we have added a new method, calcPerimeter, which computes the perimeter of the square. Notice we make use of the keyword return (in the same way we use it in functions!).

Add another method called calcArea, which returns the area of square in terms of sideLength. Use the calcPerimeter function as a guide.

The calcArea method will look a lot like the calcPerimeter method, except it should return this.sideLength \* this.sideLength.

**Script:**

var square = new Object();

square.sideLength = 6;

square.calcPerimeter = function() {

return this.sideLength \* 4;

};

// help us define an area method here

square.calcArea = function() {

return this.sideLength \* this.sideLength;

};

var p = square.calcPerimeter(8);

var a = square.calcArea(6);

**Output:** nothing

**The Object Constructor**

We mentioned the term **constructor** back in section one, when we talked about making an object using the keyword new. A **constructor** is a way to create an object.

When we write bob = new Object( ); we are using a built-in constructor called Object. This constructor is already defined by the JavaScript language and just makes an object with no properties or methods.

This means we have to add our properties one at a time, just like we've been doing. To review, we've created bob using the constructor and defined the name property for you.

Finish making bob by defining the age property and setting it equal to 20

**Use line** 3 to guide you in making the age property. Use bob.age instead of bob.name, and set it equal to 20.

**Script:**

// here we make bob using the Object constructor

var bob = new Object();

bob.name = "Bob Smith";

// add bob's age here and set it equal to 20

bob.age = 20;

**Output:** 20

**Custom Constructors**

But this approach of adding in properties one at a time for every object is tedious! Instead of always using the boring Object constructor, we can make our own constructors.

This way we can set the properties for an object right when it is created. So instead of using the Object constructor which is empty and has no properties, *we can make our own constructors which have properties*.

To see how this works, look at our Person constructor in [lines 1](javascript:void(0))–4. This constructor is used to make Person objects. Notice it uses the keyword this to define the name and age properties and set them equal to the parameters given.

Now we can use this constructor to make our good friends bob and susan in only one line each! Look at [lines 7](javascript:void(0))–8: once we have the constructor, it's way easier to make people because we can include their name and age as arguments to their respective constructors.

Practice using the constructor to make a new Person called george, whose full name is "George Washington" and age is 275.

Use [lines 7](javascript:void(0)) and 8 as a guide to make george on [line 10](javascript:void(0)). Just use "George Washington" and 275 as the name and age.

**Script:**

function Person(name,age) {

this.name = name;

this.age = age;

}

// Let's make bob and susan again, using our constructor

var bob = new Person("Bob Smith", 30);

var susan = new Person("Susan Jordan", 25);

// help us make george, whose name is "George Washington" and age is 275

var george = new Person("George Washington", 275);

**Output:** nothing

**Try it Out!**

Let's look at another example and practice coding constructors. Here we have made a Cat constructor for you, with age and color properties.

Why is this Cat constructor so cool? It means if we have many cats and wanted to create an object for each cat, we could just use this constructor with the properties already defined.

This is much better than using the Object constructor which just gives us an empty object and needs us to define every property and value for each cat object we would create.

Finish the Dog constructor we have started on [line 7](javascript:void(0)). You can include whatever parameters and properties you want (age, name, breed, whatever you can think of!) Use the Cat constructor as an example.

Your constructor should look something like:

function Dog(myParameter) {

this.property = myParameter;

};

**Script:**

function Cat(age, color) {

this.age = age;

this.color = color;

}

// make a Dog constructor here

function Dog(age, breed) {

this.age = age;

this.breed = breed;

}

**Output:** nothing

**More Options**

In a constructor, we don't have to define all the properties using parameters. Look at our new Person example on [line 1](javascript:void(0)), and see how we set the species to be "Homo Sapiens" ([line 4](javascript:void(0))). This means that when we create any Person, their species will be "Homo Sapiens". In this way, the values associated with name and age are not yet assigned, but species will always have the same value.

In this case, both sally and holden will have the same species of "Homo Sapiens", which makes sense because that is the same across all people.

Create a new object called sally using the Person constructor. Her name is "Sally Bowles" and she is 39. Create another object called holden. His name is "Holden Caulfield" and he is 16.

Edit the sentence printed out such that it includes the age of sally and holden respectively.

Create the sally and holden objects with the Person constructor like we have been doing.

You can also make edits to the console.log statements by using the + operator so you print their ages.

**Script:**

function Person(name,age) {

this.name = name;

this.age = age;

this.species = "Homo Sapiens";

}

var sally = new Person("Sally Bowles", 39);

var holden = new Person("Holden Caulfield", 16)

console.log("sally's species is " + sally.species + " and she is " + sally.age);

console.log("holden's species is " + holden.species + " and he is " + holden.age);

**Output:**

sally's species is Homo Sapiens and she is 39

holden's species is Homo Sapiens and he is 16

**Constructors With Methods**

In addition to setting properties, constructors can also define methods. This way, as soon as the object is created it will have its own methods as well.

Here we have a Rectangle constructor, which sets the height and width properties equal to the arguments, just like our Person did with name and age.

Notice we have added a calcArea method. This calculates the area of the rectangle in terms of its height and width.

[Line 11](javascript:void(0)) creates a new object rex which makes use of the constructor. You can see how rex calls the calcArea method in [line 12](javascript:void(0)) and saves the result in a variable, area.

Define a new method on [line 8](javascript:void(0)), calcPerimeter, which calculates and returns the perimeter for a Rectangle in terms of height and width.

Use calcArea as a guide to code up calcPerimeter. Remember that a rectangles perimeter is all four sides added together, or 2 \* height + 2 \* width.

When referring to a rectangle's properties, be sure to use this.propertyName

**Script:**

function Rectangle(height, width) {

this.height = height;

this.width = width;

this.calcArea = function() {

return this.height \* this.width;

};

// put our perimeter function here!

this.calcPerimeter = function() {

return 2 \* this.height + 2 \* this.width;

};

}

var rex = new Rectangle(7,3);

var area = rex.calcArea();

var perimeter = rex.calcPerimeter();

**Output:** nothing

**Constructors in Review**

Constructors are a way to make objects with the keyword new. The most basic constructor is the Object constructor, which will make an object with no methods or properties.

For more complicated objects we can make our own constructors and put in whatever properties and methods we want.

Check out our example to the right to see objects in action. Our Rabbit constructor defines an adjective property and a describeMyself method.

Recall how these kind of custom constructors are important because they allow us to easily make many similar objects.

Create a new object rabbit1 with the adjective "fluffy", a new object rabbit2 with the adjective "happy", and a new object rabbit3 with the adjective "sleepy".

Use the method describeMyself to print out in the console a sentence about each object you just created!

To create an object using the rabbit constructor, we use the syntax:

var objectName = new constructorName(value for each property);

**And to** call the method, we use the syntax:

objectName.methodName( );

**Script:**

// first we can make the instructor

function Rabbit(adjective) {

this.adjective = adjective;

this.describeMyself = function() {

console.log("I am a " + this.adjective + " rabbit");

};

}

// now we can easily make all of our rabbits

var rabbit1 = new Rabbit("fluffy");

rabbit1.describeMyself();

var rabbit2 = new Rabbit("happy");

rabbit2.describeMyself();

var rabbit3 = new Rabbit("sleepy");

rabbit3.describeMyself();

**Output:**

I am a fluffy rabbit

I am a happy rabbit

I am a sleepy rabbit

**Arrays of Objects**

Remember that an object is just another *type*, like a string or number but more complex. This means that just as we can make arrays of numbers and strings, we can also make arrays of objects.

Here we have our Person constructor which should look familiar. We can use this constructor to make an array of Person objects, similar to how we might make an array of numbers but filling in people instead.

Add one more Person to the family array, "timmy", who is 6 years old.

In this case you can add a 4th element to the family using:

family[3] = new Person(name, age);

Here the name should be "timmy" and the age should be 6.

**Script:**

// Our person constructor

function Person (name, age) {

this.name = name;

this.age = age;

}

// Now we can make an array of people

var family = new Array();

family[0] = new Person("alice", 40);

family[1] = new Person("bob", 42);

family[2] = new Person("michelle", 8);

// add the last family member, "timmy", who is 6 years old

family[3] = new Person("timmy", 6);

**Output:** {"name":"timmy","age":6}

**Loop the loop**

Arrays filled with objects will work just like arrays filled with numbers and strings.

In the same way we may loop through an array of numbers to print them out or calculate a sum, we can loop through an array of objects and access properties or methods.

1. Write a person constructor called Person that has two properties (name and age).
2. Create an empty array called family.
3. There will be four objects in the array. Using your Person constructor, create the four objects and put them in the array. The order of the objects are:
   * "alice" who is 40
   * "bob" who is 42
   * "michelle" who is 8
   * "timmy" who is 6
4. Create a for-loop that loops through the family array and prints out the name property for each family member in order of creation.

If you need a quick brush-up on how for-loops work, review the [exercises on loops](http://www.codecademy.com/courses/programming-intro/6).

Accessing the name property in the array and then using this in the for-loop is the trickiest part. Remember that we use dot notation to get the value of a property from an object:

objectName.propertyName;

And we can use the same dot notation to get the value of a property for one particular object in an array.

arrayName[2].propertyName;

This will get the third object in the array, and then find the value associated with the property.

Note that the objects must be in the order specified in the instructions!

**Script:**

// Our Person constructor

function Person (name, age) {

this.name = name;

this.age = age;

}

// Now we can make an array of people

var family = new Array();

family[0] = new Person("alice", 40);

family[1] = new Person("bob", 42);

family[2] = new Person("michelle", 8);

family[3] = new Person("timmy", 6);

// loop through our new array

for (var i = 0; i < family.length; i++) {

console.log (family[i].name);

}

**Output:**

alice

bob

michelle

timmy

---

We're running a test below to make sure your code works.

alicebobmichelletimmy

**Passing Objects into Functions**

In addition to making arrays of Objects, we can use objects as parameters for functions as well. That way, these functions can take advantage of the methods and properties that a certain object type provides.

To see an example, take a look at the console. In addition to our Person constructor we have introduced a new function, ageDifference ([line 9](javascript:void(0))). This function takes two Person objects as parameters, and returns the difference in age between the two people.

Notice we would be in trouble here if we tried to call ageDifference and passed in strings instead of people, because strings don't have an age property. But because we know from our constructor that all Person objects will have an age property, we can pass any Person into ageDifference. We must be careful not to pass anything but Person objects into ageDifference.

We have created two example people, alice and billy. Complete [line 17](javascript:void(0)) by calling ageDifference and saving the result in our global diff variable.

We can call ageDifference just as we might call any other function. And it will take two parameters, the names of our two objects.

**Script:**

// Our person constructor

function Person (name, age) {

this.name = name;

this.age = age;

}

// We can make a function which takes persons as arguments

// This one computes the difference in ages between two people

var ageDifference = function(person1, person2) {

return person1.age - person2.age;

}

var alice = new Person("Alice", 30);

var billy = new Person("Billy", 25);

// get the difference in age between alice and billy using our function

var diff = ageDifference(alice,billy);

**Output:** nothing

**Try it Out!**

This time try making your own function that takes objects as parameters!

Here we have given you the Person constructor again, along with the ageDifference function as an example.

Now create a new function, olderAge. It should take two Person objects as parameters, and return the age of whatever Person is older. For example, with 30 year-old alice and 25 year-old bob, olderAge(alice, bob); should return 30, because that is alice's age and she is older than bob. If the two people have the same age then you can return that age.

Define a function called olderAge. We want the function to return the age of the person who is older.

In the if we want to compare person1.age to person2.age to see if person1 is older. If so, we should return person1.age. Else, we should return person2.age. Remember we can use if statements to see if one value is bigger than the other with:

if(var1 > var2)

**Script:**

// Our person constructor

function Person (name, age) {

this.name = name;

this.age = age;

}

// We can make a function which takes persons as arguments

// This one computes the difference in ages between two people

var ageDifference = function(person1, person2) {

return person1.age - person2.age;

};

// Make a new function, olderAge, to return the age of

// the older of two people

var olderAge = function(person1, person2){

if(person1.age > person2.age) {

return person1.age;

} else {

return person2.age;

}

};

// Let's bring back alice and billy to test our new function

var alice = new Person("Alice", 30);

var billy = new Person("Billy", 25);

console.log("The older person is "+ olderAge(alice, billy));

**Output:** The older person is 30

**What Are Objects For?**

Objects provide us with a way to represent real-world or virtual things. We can do this by storing information inside the object's properties. There are two basic ways to make objects:

**Literal Notation**, where we use

var Name = { };

**Constructor Notation**, where we use the keyword new.

We've given an example in literal notation to refresh your memory.

Make a new object, spencer2, with the same properties but using constructor notation and the Object constructor.

Make use of the constructor Object to create an empty object. Then use dot notation to fill it in with the appropriate properties and values.

**Script:**

var spencer = {

age: 22,

country: "United States"

};

// make spencer2 here with constructor notation

var spencer2 = new Object();

spencer2.age = 22;

spencer2.country = "United States";

**Output:** "United States"

**Properties**

Properties are like variables that belong to an object, and are used to hold pieces of information. Properties can be accessed in two ways:

* **Dot notation**, with ObjectName.PropertyName
* **Bracket** notation, with ObjectName["PropertyName"] (don't forget the quotes!)

In the editor, we have brought back our snoopy object, with a species and age property.

Set the global variable species to be snoopy's species and the variable age to be snoopy's age. For one use dot notation and the other use bracket notation!

Remember the notation is ObjectName.PropertyName for [line 7](javascript:void(0)), with the object name of snoopy and the property name of species. Then use ObjectName['PropertyName'] for [line 10](javascript:void(0)), with the property name age.

**Script:**

var snoopy = new Object();

snoopy.species = "beagle";

snoopy.age = 10;

// save Snoopy's age and species into variables

// use dot notation for snoopy's species

var species = snoopy.species;

// use bracket notation for snoopy's age

var age = snoopy["age"];

**Output:** 10

**Customizing Constructors**

In addition to the basic Object constructor, we can define our own custom constructors. These are helpful for two reasons:

1. We can assign our objects properties through parameters we pass in when the object is created.
2. We can give our objects methods automatically.

These both work to save us time and lines of code when we make objects.

Notice that without the constructor, it takes us 3 lines of code to make harry\_potter, an object that represents Harry Potter book 1.

Then in [line 7](javascript:void(0)) we introduce a constructor for a Book object, where we pass in the pages and author properties as parameters.

Use this constructor to make the\_hobbit, a book with 320 pages by "J.R.R. Tolkien". Notice by using the constructor you can do this in only one line instead of three!

To make a new book with the constructor we use:

var name = new Book(pages, author);

For this example we want our book to have a pages of 320, and author of "J.R.R. Tolkien".

**Script:**

// 3 lines required to make harry\_potter

var harry\_potter = new Object();

harry\_potter.pages = 350;

harry\_potter.author = "J.K. Rowling";

// A custom constructor for book

function Book (pages, author) {

this.pages = pages;

this.author = author;

}

// Use our new constructor to make the\_hobbit in one line

var the\_hobbit = new Object();

the\_hobbit.pages = 320;

the\_hobbit.author = "J.R.R. Tolkien";

//Using the Book constructor to make a new book Object in one line

var the\_hobbit = new Book(320, "J.R.R. Tolkien");

**Output:** "J.R.R. Tolkien"

**Methods**

**Methods** are like functions that are associated with a particular object.

They are especially helpful when you want to either:

1. Update the object properties
2. Calculate something based on an object's properties.

Here, we have included a Circle object, with a radius property representing the circle's radius. We have implemented an area function which calculates the circle's area. Notice we have used Math.PI to get the π value.

Define a method perimeter that calculates the perimeter of a circle.

Recall, the formula for the perimeter of a circle is 2 \* π \* radius.

**Script:**

function Circle (radius) {

this.radius = radius;

this.area = function () {

return Math.PI \* this.radius \* this.radius;

};

// define a perimeter method here

this.perimeter = function(){

return Math.PI\*2\*this.radius;

}

}

**Output:** nothing

**Building an Address Book**

**Digitizing People**

Meet Bob. Bob is our friend. But how do we get in touch with Bob?

Look at the code in the editor. We have Bob's information stored in an **associative array** named bob. bob has a **property** called firstName which has a **value** of "Bob". Similarly, it has properties lastName, phoneNumber and email which each have values.

To access the values for each property we write array.property. Check out [line 8](javascript:void(0)) where we log to the console bob.firstName.

Copying the format we used on [line 8](javascript:void(0)), fill in [lines 9](javascript:void(0)) and 10 so that Bob's lastName and email are printed out.

To access the value of a property, we write array.property. So to get Bob's phone number, we'd write bob.phoneNumber

**Script:**

var bob = {

firstName: "Bob",

lastName: "Jones",

phoneNumber: "(650) 777-777",

email: "bob.jones@example.com"

};

console.log(bob.firstName);

console.log(bob.lastName);

console.log(bob.phoneNumber);

console.log(bob.email);

**Output:**

Bob

Jones

(650) 777-777

[bob.jones@example.com](mailto:bob.jones@example.com)

**More People**

Just like with strings and numbers, we can put multiple objects into an array. We want to practice extracting information from different objects which are stored in the same array.

This allows us to put all of our contact objects into a unified list. If the objects are contact entries, then the list is the book binding that ties all of the contact entries together.

1. Create an object called mary. It has the same properties as bob. Her name is Mary Johnson, her phoneNumber is "(650) 888 - 8888" and her email is "mary.johnson@example.com".
2. Create an array called contacts. Put bob in first (at index 0), then mary (at index 1).
3. Write a console.log statement that prints out Mary's phone number.

Telephone numbers are strings, not numbers!  
You can get Mary's object by using standard array notation: arrayName[index].

Remember that the items in an array are numbered starting at 0. Once you have that, don't forget to specify the property you want that is associated to the object.

**Arrays can be created** just like so:

var myArray = ["item1", "item2", "etc"];

**Script:**

var bob = {

firstName: "Bob",

lastName: "Jones",

phoneNumber: "(650) 777-7777",

email: "bob.jones@example.com"

};

var mary = {

firstName: "Mary",

lastName: "Johnson",

phoneNumber: "(650) 888-8888",

email: "mary.johnson@example.com"

};

var contacts = [bob, mary];

console.log(mary.phoneNumber);

**Output:**

(650) 888-8888

**Displaying People**

We currently can print out information about any person in our contacts with console.log. That gets tiring. If only we knew some code that stores blocks of code that we can call.

Good thing we know about functions!

We can create a function that consistently displays a specific property of an object.

1. Define a function called printPerson that takes a parameter called person.
2. In the function body, print out the person parameter's firstName property by accessing it with a dot just like before. Then print a space, then their lastName in the same way.
3. Call the printPerson() function to print out the first item in the contacts array. The first item in an array is at position 0.
4. Then on the next line, call printPerson() again to print out the second item in the contacts array.

Don't worry if your output appears twice - we're just double checking your code!

Here's how to define a function:

var myFunction = function(param1) {

// Do something

}

Here's how to print a person's first name and last name, separated by a space:

console.log(person.firstName + " " + person.lastName);

Here's how to print the second item in an array:

var myArray = [0, 1, 2];

console.log(myArray[1]);

// prints 1

Since items in an array are numbered starting from position 0, the second item is in position 1.

**Script:**

var bob = {

firstName: "Bob",

lastName: "Jones",

phoneNumber: "(650) 777-7777",

email: "bob.jones@example.com"

};

var mary = {

firstName: "Mary",

lastName: "Johnson",

phoneNumber: "(650) 888-8888",

email: "mary.johnson@example.com"

};

var contacts = [bob, mary];

// printPerson added here

var printPerson = function(person) {

console.log(person.firstName + " " + person.lastName);

};

// Call the printPerson function and print out the the person in the contacts array.

printPerson(contacts[0]);

printPerson(contacts[1]);

**Output:**

Bob Jones

Mary Johnson

**Listing Everybody**

Address book programs usually have a screen that lists all of the contacts. Let's build that feature.

We could write out separate lines of code to display all of the people like in the last exercise, but that's tedious. Instead, we can use a for loop to do this automatically.

We'll be creating a function that lists all of the users.

1. Create a function called list that does not take any parameters.
2. At the start of the function, define a variable to store the number of items in the contacts array. Call it contactsLength.
3. All of the items in an array are numbered, starting at 0. To cycle through all of the elements of the array, create a for loop that cycles from 0 up to one less than the number of items in the contacts array.
4. Inside of the loop, add code to call printPerson, passing in the element of the array that the loop is currently at.
5. At the very bottom of the file, call the list function. The list function should then loop through every member of the contacts array and print its information.

If you need a review of how to use for loops, click [here](http://www.codecademy.com/courses/javascript-beginner-en-NhsaT?curriculum_id=506324b3a7dffd00020bf661). And remember the array.length property for getting the array length!

The number of items in an array can be determined with array.length.

To cycle to one less than a number in a for loop, you can use the < operator in the second loop parameter.

for loops have a number that is incremented. You can get the array item that is at that number with array[number].

**Script:**

var bob = {

firstName: "Bob",

lastName: "Jones",

phoneNumber: "(650) 777-7777",

email: "bob.jones@example.com"

};

var mary = {

firstName: "Mary",

lastName: "Johnson",

phoneNumber: "(650) 888-8888",

email: "mary.johnson@example.com"

};

var contacts = [bob, mary];

function printPerson(person) {

console.log(person.firstName + " " + person.lastName);

}

var list = function(){

var contactsLength = contacts.length;

for (var i = 0; i < contacts.length; i++) {

printPerson(contacts[i]);

}

};

list();

**Output:**

Bob Jones

Mary Johnson

Bob Jones

Mary Johnson

**Finding that Special Someone**

Let's say we're looking for someone in our address book with a specific last name.

We can do this with a technique for searching arrays called "linear search". With it, we use a loop to check through all of the items in the array one-by-one until we see the item that we want.

We can apply linear search to print out all of the people that have a particular last name.

We'll be creating a function that can search for people with a specific last name and print those people out with the printPerson function.

1. Create a function called search that takes a parameter called lastName. Leave the list function alone.
2. Like with the last exercise, define a variable and store the number of items in the array in it. (Since every function has its own context, or **scope**, you can call this variable contactsLength, too, if you like!)
3. Create a for loop that runs through all of the items in the array. For this step, the code for search is identical to that of list.
4. The twist comes here: in the body of the loop, rather than printing out every single item in the array, add an if statement that checks to see if the lastName property of the object is equal to the lastName argument. Have the function run printPerson on the person if and only if the lastName property of the person matches the lastName argument.
5. At the bottom of the file, call the search function, passing in "Jones" as the last name to search for.

**To get the** current value of the lastName parameter, you can use the dot syntax on array items that you can with normal variables: array[number].lastName;.

**Script:**

var bob = {

firstName: "Bob",

lastName: "Jones",

phoneNumber: "(650) 777-7777",

email: "bob.jones@example.com"

};

var mary = {

firstName: "Mary",

lastName: "Johnson",

phoneNumber: "(650) 888-8888",

email: "mary.johnson@example.com"

};

var contacts = [bob, mary];

function printPerson(person) {

console.log(person.firstName + " " + person.lastName);

}

function list() {

var contactsLength = contacts.length;

for (var i = 0; i < contactsLength; i++) {

printPerson(contacts[i]);

}

}

//list();

/\*Create a search function

then call it passing "Jones"\*/

var search = function(lastName){

var contactsLength = contacts.length;

for (var i = 0; i < contactsLength; i++) {

if (( contacts [i] .lastName ) === lastName) {

printPerson(contacts [i]);

}

}

}

search("Jones");

**Output:** Bob Jones

**We Made a Friend!**

We have our address book in the contacts array, but what if we make a new friend and want to add them as well?

Objects, just like other types of data, can be put into arrays with a array[position] = object statement. To append something to the end of the array, you need to put it in the position one after the last item.

Since arrays are numbered starting at zero, the number of the last item in the array will be one less than the quantity of items in the array. The size of the array is thus the position to insert at.

The length of an array, like the length of a string, can be found with array.length.

We can do the insert in a succinct way by adding the new object directly into the array position without even giving it a name. This can be confusing, but we will be able to refer to it by its array position, so it does not need a direct name. Do it like this:

contacts[contacts.length] = {

firstName: firstName,

lastName: lastName,

phoneNumber: phoneNumber,

email: email

};

(Assuming you defined the add function with the parameters firstName, lastName, phoneNumber, and email.)

That will automatically create a new object and add it into the array. Pretty neat.

We'll be creating a function that allows us to add our new friend to the address book.

1. Create a function called add with the parameters firstName, lastName, and email, phoneNumber.
2. In this new function, you want to create a new contact object like bob and mary. Instead of having this object's property values be filled with strings though, set them to the appropriate function parameters passed in.
3. Add this new contact object to the contacts array.
4. Call add with whatever first name, last name, phone number, and email arguments you like.
5. Make sure you call the list function, to check if your new entry is added. And delete any other function that logs output in the console, i.e 'search' function.

Run the code!

**More specifically**, an object can be added to the end of the array with array[array.length] = object.

**Script:**

var bob = {

firstName: "Bob",

lastName: "Jones",

phoneNumber: "(650) 777-7777",

email: "bob.jones@example.com"

};

var mary = {

firstName: "Mary",

lastName: "Johnson",

phoneNumber: "(650) 888-8888",

email: "mary.johnson@example.com"

};

var contacts = [bob, mary];

function printPerson(person) {

console.log(person.firstName + " " + person.lastName);

}

function list() {

var contactsLength = contacts.length;

for (var i = 0; i < contactsLength; i++) {

printPerson(contacts[i]);

}

}

/\*Create a search function

then call it passing "Jones"\*/

var search = function(lastName){

var contactsLength = contacts.length;

for (var i = 0; i < contactsLength; i++) {

if (( contacts [i] .lastName ) === lastName) {

printPerson(contacts [i]);

}

}

};

var add = function(firstName, lastName, email, phoneNumber){

new Object();

this.firstName = firstName;

this.lastName = lastName;

this.email = email;

this.phoneNumber = phoneNumber;

contacts[contacts.length] = this;

};

add("Fake", "Dude", "f@ke.none", "555.555.5555");

list();

**Output:**

Bob Jones

Mary Johnson

Fake Dude

**Introduction to Objects II**

**An Objective Review**

Let's review the basics of objects covered in our previous [lesson on objects](http://www.codecademy.com/courses/spencer-sandbox?curriculum_id=506324b3a7dffd00020bf661). Recall we can create objects using either literal notation or constructor notation.

* **Literal notation** creates a single object. Literal notation uses curly brackets { } and the object's default properties are defined within the brackets using property:value notation.
* **Constructor notation** involves defining an object *constructor*. And like defining a function, we use the function keyword. You can think of this constructor as a "template" from which you can create multiple objects. To create a new object from a constructor, we use the new keyword.

Finish the james object by adding properties to it. His job should be "programmer" and should have a married property set to false.

Create a new gabby object using the Person constructor. She should have a job of "student" and her married property should be true.

Take a look at these objects created using literal and constructor notation:

var obj = {

property: "value",

otherProp: 42

};

var obj = new Object("value", 42);

Remember properties of objects created with literal notation should be separated by commas.

**For other** reminders about how objects work, revisit the [first course on objects](http://www.codecademy.com/courses/spencer-sandbox).

**Script:**

var james = {

// add properties to this object!

job: "programmer",

married: false

};

function Person(job, married) {

this.job = job;

this.married = married;

}

// create a "gabby" object using the Person constructor!

var gabby = new Person("student",true);

**Output:** nothing

**Fun with Functions**

Recall that we can add methods (i.e., functions associated with objects) to a constructor:

function someObject() {

this.someMethod = function() {

};

}

Suppose we said var someObj = new someObject();. When we call someObj.someMethod(), the code between the curly brackets { } above will run.

Add a speak method to the Person constructor. Whenever speak is called, it should print "Hello!" to the console.

Take a look at how someMethod is defined above. If you need further reminders about how methods in constructors work, refer to the ["Constructors With Methods"](http://www.codecademy.com/courses/spencer-sandbox/3#!/exercises/4) exercise.

Note speak should use console.log and not return anything.

**Script:**

function Person(job, married) {

this.job = job;

this.married = married;

// add a "speak" method to Person!

this.speak = function(speak) {

console.log("Hello!");

};

}

// Create a new Person constructor

var user = new Person("Job Name", false);

user.speak();

**Output:** Hello!

**Literally Speaking**

In the last exercise, we added methods to objects via constructor notation. We can also add methods to objects in literal notation:

var someObj = {

aProperty: value,

someMethod: function(some, params) { }

};

When we call someObj.someMethod(some, values);, the code between the curly brackets { } will run.

Note here we see a method that takes parameters. Methods defined in both constructors and literal notation can take parameters, just like normal functions.

Take a look at the partially-defined james object. Complete the speak method such that the last two lines in the editor will cause "Hello, I am feeling great" and "Hello, I am feeling just okay" to be printed to the console.

Because the calls to speak involve passing in parameters, this lets us know we should probably define a parameter in for the speak function in [line 4](javascript:void(0)). Add a parameter such as mood between the parentheses.

The speak method should directly print to the console, not return anything.

**Script:**

var james = {

job: "programmer",

married: false,

speak: function(mood) {

if (mood === "great"){

console.log ("Hello, I am feeling" + " " + mood);

}

else if(mood === "just okay"){

console.log ("Hello, I am feeling" + " " + mood);

}

}

};

james.speak("great");

james.speak("just okay");

**Output:**

Hello, I am feeling great

Hello, I am feeling just okay

**Can I See Your References?**

Remember when defining a method for an object, it's easy to reference other properties in that object: just use this.propertyName!

When that method is called, this.propertyName will always refer to the most recent value of propertyName.

Take a look at the james object. Complete the sayJob method so that it will print to the console "Hi, I work as a [job]", where [job] is the value of the job property.

Then in [line 14](javascript:void(0)), change the job for james to "super programmer". Although the method calls in [lines 11](javascript:void(0)) and 17 are exactly the same, their output should be different because James' job changed!

You should use this.job inside your method. It should not be a parameter, but is used as a part of the console.log statement.

Remember to use console.log in your method; there's no need to return anything.

**Script:**

var james = {

job: "programmer",

married: false,

sayJob: function() {

// complete this method

console.log("Hi, I work as a " + this.job);

}

};

// james' first job

james.sayJob();

// change james' job to "super programmer" here

james.job = "super programmer";

// james' second job

james.sayJob();

**Output:**

Hi, I work as a programmer

Hi, I work as a super programmer

**Who's in Your Bracket?**

And finally, let's go over retrieving property values. Throughout this section, we've been using **dot notation** to get the value of an object's property:

someObj.propName;

However, remember that we can also use **bracket notation**:

someObj["propName"];

An advantage of bracket notation is that we are not restricted to just using strings in the brackets. We can also use variables whose values are property names:

var someObj = {propName: someValue};

var myProperty = "propName";

someObj[myProperty];

The last line is *exactly the same* as using someObj["propName"];.

Take advantage of the ability to use variables with bracket notation.

In [line 7](javascript:void(0)), set aProperty to a string of the first property in james (ie. the job property).

Then print james's job using bracket notation and aProperty.

What are the properties of james? Set aProperty to be the name (not value!) of the first property (in order as they are defined in the code) of james.

aProperty **should** be a String. Recall that Strings are essentially "text surrounded by quotes". We want our text here to be the name of james's first property.

You should then use console.log to print.

**Script:**

var james = {

job: "programmer",

married: false

};

// set to the first property name of "james"

var aProperty = "job";

// print the value of the first property of "james"

// using the variable "aProperty"

console.log(james[aProperty]);

**Output:** programmer

**I.D., Please**

Alright! Let's get our hands dirty and start exploring some really cool stuff about objects in JavaScript. But before we can do that, how can we even tell if something is an object (as opposed to, say, a number or string)? It would be great if we could tell what *type* something is in JavaScript. Good thing there's a handy built-in operator to do this!

Say we have a variable thing and we don't know what type thing is. We can call typeof thing to figure this out. Generally, the most useful types are "number," "string," "function," and of course, "object."

As an example, the following example will print "object":

var someObject = {someProperty: someValue};

console.log( typeof someObject );

In [lines 3-6](javascript:void(0)), we have an object, a number, and a string (in that order). Complete [lines 7-9](javascript:void(0)) so they will print the appropriate types of these variables.

Use the typeof operator!

Note the peculiar syntax of typeof: it is followed immediately by only single a variable—nothing else. The parentheses in the example above are part of the console.log call, not the typeof operator.

**Script:**

// complete these definitions so that they will have

// the appropriate types

var anObj = { job: "I'm an object!" };

var aNumber = 42;

var aString = "I'm a string!";

console.log( typeof anObj ); // should print "object"

console.log( typeof aNumber ); // should print "number"

console.log( typeof aString ); // should print "string"

**Output:**

object

number

string

**Know Thyself**

In the last exercise, we used typeof to figure out what type a variable in JavaScript is. Since we know how to tell objects apart from everything else now, let's focus on them.

You wouldn't know it, but every object in JavaScript comes with some baggage (stay tuned for more on this!). Part of this baggage includes a method called hasOwnProperty. This lets us know if an object has a particular property.

We show how to use hasOwnProperty in the last two lines. It returns true or false, based on whether an object has a certain property.

You should finish myObj by giving it a name property. Make sure that myObj does not have a nickname property so that the last line will print false.

**Script:**

var myObj = {

// finish myObj

name: "object"

};

console.log( myObj.hasOwnProperty('name') ); // should print true

console.log( myObj.hasOwnProperty('nickname') ); // should print false

**Output:**

true

false

**Dressed to Impress**

Let's get some practice working with hasOwnProperty. It is an invaluable tool when working with objects!

Try to run the code in the editor as it is. You should get an error because shorts is not a property of the suitcase object.

Let's write some code to test for this so we can avoid this nasty error later.

Remove the console.log statement.

Write an if statement that checks to see if suitcase has the shorts property.

If your if statement evaluates to true, print the value of the shorts property.

If your if statement evaluates to false, set the shorts property to any value you wish using dot notation. Then print the value of the shorts property.

Recall that obj.hasOwnProperty('propertyName') will return true or false based on whether or not obj has the propertyName property.

You should probably use an if-else statement:

if( expression ) {

} else {

}

When you're setting the shorts property, feel free to use either dot notation or bracket notation:

suitcase.shorts = "red";  
or  
suitcase["shorts"] = "red";

**Script:**

var suitcase = {

shirt: "Hawaiian"

};

if (suitcase.hasOwnProperty("shorts")){

console.log(suitcase.shorts);

} else {

suitcase.shorts = "Cargo";

console.log(suitcase.shorts);

}

//console.log(suitcase.shorts);

**Output:** Cargo

**Getting IN-timate**

Now let's learn how to work with all the properties that belong to an object. First, let's define an object:

var dog = {

species: "bulldog",

age: 3,

color: brown

};

To print out all elements, we can use a for/in loop, like this:

for(var property in dog) {

console.log(property);

}

In the loop we use console.log to print out each key. Remember the "property" bit can be any placeholder name you like.

Use a for-in loop to print out all the properties of nyc.

Look at the for-in loop example above as a reference. Your code should look extremely similar.

**Script:**

var nyc = {

fullName: "New York City",

mayor: "Michael Bloomberg",

population: 8000000,

boroughs: 5

};

// write your for-in loop here

for(var x in nyc) {

console.log(x);

}

**Output:**

fullName

mayor

population

boroughs

**List ALL the Properties!**

We've just seen how to print all of an object's property names with a for-in loop. But how do we print out all the values associated with every property? Surprise! The for-in loop will be our friend again! Let's get there slowly. Our dog object can help us.

var dog = {

species: "bulldog",

age: 3,

color: brown

};

First, remember that  
dog.species = dog["species"] = "bulldog";

And if we say:

var x = "species";

then

dog[x] = "bulldog";

We see that by assigning the property name to a variable, we can then use the variable name in bracket notation to get the property's value. So to get all the values from the dog object, we would use the for-in loop and the bracket notation we just saw above. See the hint to see the code to print the property values for dog.

Write another for-in loop, but this time print the value of each property in nyc.

This exercise is a little tricky. To print all the property values of the dog object, the code is:

for(var x in dog) {

console.log(dog[x]);

}

**Script:**

var nyc = {

fullName: "New York City",

mayor: "Michael Bloomberg",

population: 8000000,

boroughs: 5

};

// write a for-in loop to print the value of nyc's properties

for(var x in nyc){

console.log(nyc[x]);

}

var x ="fullName";

console.log(nyc[x]);

**Output:**

New York City

Michael Bloomberg

8000000

5

New York City

**Class is in Session**

Alright, it's time to learn the basics of **object-oriented programming**! Often abbreviated OOP, this is a very important programming paradigm that is widely used in the industry today.

Let's start by introducing *classes*. We learned in the last [course](http://www.codecademy.com/courses/spencer-sandbox/3/1?curriculum_id=506324b3a7dffd00020bf661) that constructors are a way to make objects, but they actually do even more than that.

When you make a constructor, you are in fact defining a new **class**. A class can be thought of as a *type*, or a category of objects—kind of like how Number and String are types in JavaScript.

Take a look at our Person example taken from [Introduction to Objects I](http://www.codecademy.com/courses/spencer-sandbox?curriculum_id=506324b3a7dffd00020bf661). In this case bob and susan are two separate objects, but both belong to the class Person.

Make your own class, Circle, by building a constructor for it. The constructor for Circle should have one property, radius, and take one argument for the initial radius.

Use the Person constructor as a guide to make your Circle constructor. Your constructor will only take one argument in the parentheses though, and should set this.radius equal to that argument.

**Script:**

function Person(name,age) {

this.name = name;

this.age = age;

}

// Let's make bob again, using our constructor

var bob = new Person("Bob Smith", 30);

var susan = new Person("Susan Jordan", 35);

// make your own class here

function Circle(radius){

this.radius = radius;

}

**Output:** nothing

**Teach Snoopy**

So we know that a class will have certain properties and methods, but what keeps track of what a given class can or can't do? What a class has or doesn't have? That is the job of the **prototype**.

JavaScript automatically defines the prototype for class with a constructor. For example, our Dog constructor ensures that the Dog prototype has a breed property. Remember, the Dog prototype keeps track of what Dog has, doesn't have, can, or can't do.

We know we can add methods to objects, and in [line 7](javascript:void(0)) we add the bark method to buddy. Hit run and you will see one "Woof" printed when buddy barks. Notice what happens when we try to get snoopy to bark in [line 17](javascript:void(0)) though. Even though snoopy is of the class Dog, he doesn't know how to bark because only buddy had bark added as a method.

To fix this, start at [line 15](javascript:void(0)) add a bark method for the snoopy object. You can make it just like the bark method for buddy, or type in whatever barking noise you want instead of "Woof".

Take a look at how the bark method was defined for buddy at [line 7](javascript:void(0)). Your solution should look extremely similar, except this time we're giving snoopy a bark method.

**Script:**

function Dog (breed) {

this.breed = breed;

}

// here we make buddy and teach him how to bark

var buddy = new Dog("Golden Retriever");

buddy.bark = function() {

console.log("Woof");

};

buddy.bark();

// here we make snoopy

var snoopy = new Dog("Beagle");

// we need you to teach snoopy how to bark here

snoopy.bark = function() {

console.log("Woof, Woof");

};

// this causes an error, because snoopy doesn't know how to bark!

snoopy.bark();

**Output:**

Woof

Woof, Woof

**How do Classes Help Us?**

Classes are very important in object-oriented programming. This is because a class tells us helpful information about objects, and you can think of an object as a particular instance of a class.

For example, look at our Person class again in the console. We know that any Person will have a name and age, because they are in the constructor. This allows us to create a function like printPersonName, which will take a Person as an argument and print out their name. We know the function will work on any Person, because name is a valid property for that class.

Make a Person called me with your own name and age, and print your name using printPersonName.

Use [line 10](javascript:void(0)) as a guide to making me, but use your own name and age when making the Person. Then use [line 11](javascript:void(0)) as a guide to call the function, calling printPersonName with me as the argument instead of bob.

**Script:**

function Person(name,age) {

this.name = name;

this.age = age;

}

// a function that prints the name of any given person

function printPersonName(p) {

console.log(p.name);

}

var bob = new Person("Bob Smith", 30);

printPersonName(bob);

// make a person called me with your name and age

// then use printPersonName to print your name

var me = new Person("Matt", 42);

printPersonName(me);

**Output:**

Bob Smith

Matt

**Prototype to the Rescue**

Here we have very similar code as last time, but there is an important difference. Instead of using buddy.bark to add the bark method to just the buddy object, we use Dog.prototype.bark.

Click run this time, and both buddy and snoopy can bark just fine! Snoopy can bark too even though we haven't added a bark method to that object. How is this so? Because we have now changed the *prototype* for the class Dog. **This immediately teaches *all* Dogs the new method.**

In general, if you want to add a method to a class such that all members of the class can use it, we use the following syntax to *extend the prototype*:

className.prototype.newMethod =

function() {

statements;

};

**Script:**

function Dog (breed) {

this.breed = breed;

}

// here we make buddy and teach him how to bark

var buddy = new Dog("golden Retriever");

Dog.prototype.bark = function() {

console.log("Woof");

};

buddy.bark();

// here we make snoopy

var snoopy = new Dog("Beagle");

/// this time it works!

snoopy.bark();

**Output:**

Woof

Woof

**Prototype Practice**

Here we have created a new class, Cat, and its constructor. We also have two cats that would like to meow, but currently Cats have no meow method.

Add a meow method to the Cat prototype so that all cats can now meow. This method should print to the console "Meow!". Then call this method for each cat.

Look at how we modified the Dog prototype in the previous exercise. Your method this time should follow a pattern of Cat.prototype.meow = function () { }; and insert relevant code between the curly brackets.

Use console.log to print.

Call your meow method using the object.method() pattern.

**Script:**

function Cat(name, breed) {

this.name = name;

this.breed = breed;

}

// let's make some cats!

var cheshire = new Cat("Cheshire Cat", "British Shorthair");

var gary = new Cat("Gary", "Domestic Shorthair");

// add a method "meow" to the Cat class that will allow

// all cats to print "Meow!" to the console

Cat.prototype.meow = function(){

console.log("Meow!");

};

// add code here to make the cats meow!

cheshire.meow();

gary.meow();

**Output:**

Meow!

Meow!

**It's All in the Genes**

In object-oriented programming, **inheritance** allows one class to see and use the methods and properties of another class. You can think of it as a child being able to use his or her parent's money because the child *inherits* the money.

We will learn more about inheritance as we continue this lesson, but for now let's just refresh our memories about how classes and objects work.

Create a class named Animal with two properties, name and numLegs. The Animal constructor should have two arguments whose values are assigned to name and numLegs.

Next, change the prototype of Animal and add a method sayName that prints to the console "Hi my name is [name]", where [name] is the value of name.

Click "Stuck? Get a hint!" for examples of how to create a class and how to add a method to an object's prototype.

Finally, we have provided the last two lines to test your constructor and sayName method. Don't change these!

Return to the previous lesson for help on creating your own classes. Remember how we created a Person class?

function Person(name,age) {

this.name = name;

this.age = age;

};

Also recall how we added a method to a class's prototype:

Dog.prototype.bark = function() {

console.log("Woof");

};

To reference an Animal's name property when changing its prototype, be sure to use this.name.

Remember when we want to insert the value of a variable in a String, we use the + operator:

var num = 3;

var str = "The number is " + num!";

**Script:**

// create your Animal class here

function Animal(name, numLegs) {

this.name = name;

this.numLegs = numLegs;

}

// create the sayName method for Animal

Animal.prototype.sayName = function() {

console.log("Hi my name is " + this.name);

};

// provided code to test above constructor and method

var penguin = new Animal("Captain Cook", 2);

penguin.sayName();

**Output:**

Hi my name is Captain Cook

**Marching Penguins**

Let's say we're dealing with a lot of Penguins. It sure would be nice to create a Penguin class so that perhaps later we can give it some methods unique to a penguin and not confuse it with the Animal class.

Create a brand new Penguin class constructor starting in [line 11](javascript:void(0)). A penguin is an animal so it should also have the name and numLegs properties as well as a sayName method that prints the same thing as Animal's sayName method.

We're not done with animals yet, so we have still included the Animal constructor and its sayName method. The last two lines test your Penguin code.

Is there really a difference between the Penguin and Animal constructors and sayName method? Except for the "Penguin" part, not really...

**Script:**

// create your Animal class here

function Animal(name, numLegs) {

this.name = name;

this.numLegs = numLegs;

}

function Penguin(name, numLegs) {

this.name = name;

this.numLegs = numLegs;

}

// create the sayName method for Animal

Animal.prototype.sayName = function() {

console.log("Hi my name is " + this.name);

};

Penguin.prototype.sayName = function() {

console.log("Hi my name is " + this.name);

};

// provided code to test above constructor and method

var penguin = new Animal("Captain Cook", 2);

penguin.sayName();

**Output:** Hi my name is Captain Cook

**DRY Penguins**

Creating a brand new Penguin was nice, but we did end up reusing a lot of the same code as the Animal class. This goes against the "DRY" principle of programming: Don't Repeat Yourself.

Inheritance can help us here! A Penguin is an Animal, so they should have all the same properties and methods as Animal. Whenever this **X is-a Y** relationship exists, there's a good chance that we should be using inheritance.

Remember, *inheritance* lets us see and use properties and methods from another class. To say that Penguin *inherits from* Animal, we need to set Penguin's prototype to be Animal.

Create a new Penguin class. The Penguin constructor can be more unique than the generic Animal one because all penguins have 2 legs. Your constructor should only take a name parameter, and within the constructor itself, set this.numLegs to 2.

Set the Penguin class's prototype to a new instance of Animal by adding this line after you make the constructor:

Penguin.prototype = new Animal();

This means that Penguin inherits properties and methods from Animal.

Does the body of your Penguin constructor look like this?

this.name = name;

this.numLegs = 2;

Penguin.prototype should be set outside the constructor, not within it.

**Script:**

// the original Animal class and sayName method

function Animal(name, numLegs) {

this.name = name;

this.numLegs = numLegs;

}

Animal.prototype.sayName = function() {

console.log("Hi my name is "+this.name);

};

// define a Penguin class

function Penguin(name) {

this.name = name;

this.numLegs = 2;

}

// set its prototype to be a new instance of Animal

Penguin.prototype = new Animal();

**Output:** {}

Black (and White) Penguin Magic

Now for some black magic and to see the power of inheritance!

We never defined a sayName method for Penguin, but what happens when we try to call it?

Create a Penguin object with the variable name penguin and any name you'd like.

Then call penguin.sayName();.

Then be amazed.

After your code from last time, create an object named penguin like we've been doing before. After this, call penguin.sayName( );.

**Script:**

// the original Animal class and sayName method

// the original Animal class and sayName method

function Animal(name, numLegs) {

this.name = name;

this.numLegs = numLegs;

}

Animal.prototype.sayName = function() {

console.log("Hi my name is "+this.name);

};

// define a Penguin class

function Penguin(name) {

this.name = name;

this.numLegs = 2;

}

// set its prototype to be a new instance of Animal

Penguin.prototype = new Animal();

var penguin = new Penguin("Happy Feet");

penguin.sayName();

**Output:**

Hi my name is Happy Feet

**Penguins, Properties, and the Prototype**

We saw in the last exercise how Penguin inherited the sayName method from Animal. We now explore how classes can inherit properties as well.

For simplicity, we've defined a new Penguin class that doesn't inherit anything from Animal.

Create an Emperor class that takes a single name parameter and sets its name property to be this value. Don't set a numLegs property in the constructor.

Similar to what we did in the previous exercise, make Emperor inherit from Penguin by setting the prototype of Emperor to be Penguin.

Create a new emperor object that is an instance of the Emperor class with any name you'd like. Then use console.log to print the number of legs emperor has—this should have been inherited from Penguin!

Below is how we said Penguin inherited from Animal. Your code for Emperor and Penguin will be similar.

Penguin.prototype = new Animal();

To access emperor's number of legs, use emperor.numLegs;.

**Script:**

function Penguin(name) {

this.name = name;

this.numLegs = 2;

}

// create your Emperor class here and make it inherit from Penguin

function Emperor(name) {

this.name = name;

}

// create an "emperor" object and print the number of legs it has

Emperor.prototype = new Penguin();

var emperor = new Emperor("Empie");

console.log(emperor.numLegs);

**Output:** 2

**Up the Food-I-mean-Prototype Chain**

A penguin is an animal and an emperor penguin is a penguin. Are emperor penguins animals too? Of course!

The "prototype chain" in JavaScript knows this as well. If JavaScript encounters something it can't find in the current class's methods or properties, it looks up the *prototype chain* to see if it's defined in a class that it inherits from. This keeps going upwards until it stops all the way at the top: the mighty Object.prototype (more on this later). By default, all classes inherit directly from Object, unless we change the class's prototype, like we've been doing for Penguin and Emperor.

Let's see how going up the prototype chain works! We've defined some classes and inheritance patterns: Emperor inherits from Penguin which inherits from Animal. We've also created an instance of the Emperor class.

Without modifying anything other than [lines 22-24](javascript:void(0)), complete the console.log statements to print the appropriate responses.

Remember how the prototype chain works: if a property is not defined for a class, this class's prototype chain will be traversed upwards until one is found (or not) in a parent (higher) class.

You should be printing properties of myEmperor.

**Take advantage** of the prototype chain here. myEmperor.numLegs; should be the value 2.

We want to return a Boolean value of true. Remember that any time you use a comparator like > or < or !==, this will return a Boolean!

**Script:**

// original classes

function Animal(name, numLegs) {

this.name = name;

this.numLegs = numLegs;

this.isAlive = true;

}

function Penguin(name) {

this.name = name;

this.numLegs = 2;

}

function Emperor(name) {

this.name = name;

this.saying = "Waddle waddle";

}

// set up the prototype chain

Penguin.prototype = new Animal();

Emperor.prototype = new Penguin();

var myEmperor = new Emperor("Jules");

console.log( myEmperor.saying ); // should print "Waddle waddle"

console.log( myEmperor.numLegs ); // should print 2

console.log( myEmperor.isAlive ); // should print true

**Output:**

Waddle waddle

2

True

**Open to the Public**

In JavaScript all properties of an object are automatically public. **Public** means that they can be accessed outside the class. Think of these properties as the information a class is willing to *share*.

Look at the Person class. It has 3 public properties: firstName, lastName, and age. On [lines 8](javascript:void(0)) and 9, we access the firstName and lastName properties of john and assign them to myFirst and myLast.

Notice that we are free to access the firstName and lastName properties, which is what we mean when we say they are *public*.

Declare a third variable called myAge and use it to store the age property of the john object.

**Use lines** 8-9 as a guide, and remember how to declare a variable with var myVar. Set myAge equal to john.age.

var myAge = \_\_\_\_\_\_\_\_\_\_\_\_\_

**Script:**

function Person(first,last,age) {

this.firstName = first;

this.lastName = last;

this.age = age;

}

var john = new Person('John','Smith',30);

var myFirst = john.firstName;

var myLast = john.lastName;

var myAge = john.age;

//declare variable myAge set to the age of the john object.

**Output:** nothing

**Private Variables**

Good! But what if an object wants to keep some information hidden?

Just as functions can have local variables which can only be accessed from within that function, objects can have private variables. **Private** variables are pieces of information you do not want to publicly share, and they can only be directly accessed from within the class.

The Person class has been modified to have a private variable called bankBalance. Notice that it looks just like a normal variable, but it is defined inside the constructor for Person without using this, but instead using var. This makes bankBalance a private variable.

Create an object john using the Person constructor. He can have any name and age you wish.

Next, try to print his bankBalance. What happens?

Look to the previous exercise for hints on how to create john.

Will you be able to print bankBalance like we did in the previous exercise? Well, you'll never know if you don't try!

**Script:**

function Person(first,last,age) {

this.firstname = first;

this.lastname = last;

this.age = age;

var bankBalance = 7500;

}

// create your Person

var john = new Person('John','Smith',30);

var myFirst = john.firstName;

var myLast = john.lastName;

//declare variable myAge set to the age of the john object.

var myAge = john.age;

// try to print his bankBalance

console.log(john.bankBalance);

**Output:** undefined

**Accessing Private Variables**

Although we cannot directly access private variables from outside the class, there is a way to get around this. We can define a public method that returns the value of a private variable.

Here we have included similar code from last time, but here we have added a method getBalance. Modify getBalance so that it returns bankBalance.

Then on [line 17](javascript:void(0)), create a new variable named myBalance and set its value to John's bank balance. You can do this by calling your newly-defined getBalance method for john. Then print myBalance.

[Line 14](javascript:void(0)) should still print undefined!

On [line 9](javascript:void(0)), return the variable bankBalance.

Then on [line 17](javascript:void(0)) call the method john.getBalance(); and save the result in a new variable called myBalance. Use console.log to print myBalance.

**Script:**

function Person(first,last,age) {

this.firstname = first;

this.lastname = last;

this.age = age;

var bankBalance = 7500;

this.getBalance = function() {

// your code should return the bankBalance

return bankBalance;

};

}

var john = new Person('John','Smith',30);

console.log(john.bankBalance);

// create a new variable myBalance that calls getBalance()

var myBalance = john.getBalance();

console.log(myBalance);

**Output:**

undefined

7500

**Private Methods**

Why did that code work? An object's private variables can only be accessed by other methods that are part of that same object. So, we just used an object's public method to access a private variable!

Methods can also be private within a class and inaccessible outside of the class. Changing this.returnBalance from the last exercise to var returnBalance makes this method private. If you run the program trying to access the method you get an undefined error this time.

The way to access a private method is similar to accessing a private variable. You must create a public method for the class that returns the private method.

Create a method called askTeller within the Person class that returns the returnBalance method. This means that it returns the method itself and **NOT** the result of calling that method. So you should **NOT** have parentheses after returnBalance.

Because askTeller returns a method, we need to call it to make it any use. This is what var myBalance = myBalanceMethod(); does.

Your method should resemble how we defined getBalance last time—you should use this.askTeller = function() { }. Don't declare askTeller with var.

You return method the same way that you would return simple variables. Be careful not to call the method though and leave out parentheses in your return statement!

**Script:**

function Person(first,last,age) {

this.firstname = first;

this.lastname = last;

this.age = age;

var bankBalance = 7500;

var returnBalance = function() {

return bankBalance;

};

// create the new function here

this.askTeller = function() {

return returnBalance;

};

}

var john = new Person('John','Smith',30);

console.log(john.returnBalance);

var myBalanceMethod = john.askTeller();

var myBalance = myBalanceMethod();

console.log(myBalance);

**Output:**

undefined

7500

**Passing Arguments**

The askTeller function has been modified within the Person class to directly give you your balance. However, it now needs the account password in order to return the bankBalance.

Create a new variable called myBalance that calls the askTeller function with a password argument, 1234.

The askTeller function is public so no new functions need to be created. myBalance should look about the same as it did in the last exercise except with a password as a parameter.

Recall how to pass argument:

var myVariable = otherVariable.function(argument);

**Script:**

function Person(first,last,age) {

this.firstname = first;

this.lastname = last;

this.age = age;

var bankBalance = 7500;

this.askTeller = function(pass) {

if (pass == 1234) return bankBalance;

else return "Wrong password.";

};

}

var john = new Person('John','Smith',30);

/\* the variable myBalance should access askTeller()

with a password as an argument \*/

var myBalance = john.askTeller(1234);

**Output:** nothing

**Looks For-In To Me**

Objects aren't so foreign if you really think about it!

Remember you can figure out the *type* of a variable by using typeof myVariable;. Types we are concerned with for now are "object", "string", and "number".

Recall the for-in loop:

for(var x in obj) {

executeSomething();

}

This will go through all the properties of obj one by one and assign the property name to x on each run of the loop.

Let's combine our knowledge of these two concepts.

Examine the languages object. Three properties are strings, whereas one is a number.

Use a for-in loop to print out the three ways to say hello. In the loop, you should check to see if the property value is a string so you don't accidentally print a number.

You should use an if statement in combination with the typeof operator to figure out whether or not something is a "string". If it's a "string", then print it!

Make sure you're checking the property value (e.g., "Hello!") and not the property name (e.g., english). Recall that if we save a property name to a variable, we can access the value associated with that name using bracket notation. See [here](http://www.codecademy.com/courses/objects-ii/1#!/exercises/4) to review this.

**Script:**

var languages = {

english: "Hello!",

french: "Bonjour!",

notALanguage: 4,

spanish: "Hola!"

};

// print hello in the 3 different languages

for(var x in languages){

if (typeof languages[x] === "string"){

console.log(languages[x]);

}

}

**Output:**

Hello!

Bonjour!

Hola!

**Hello? Yes, This is Dog**

We should all know by now what's so cool about using prototype: we can define a method for a class, and any instance of the class (i.e., object created using that class's constructor) can use that method.

Remember that classes and the prototype are important to OOP!

Add the sayHello method to the Dog class by extending its prototype.

sayHello should print to the console: "Hello this is a [breed] dog", where [breed] is the dog's breed.

Recall how we previously added a method to the Dog class:

Dog.prototype.bark = function() {

console.log("Woof");

};

To access a dog's breed from within the method, use this.breed.

**Script:**

function Dog (breed) {

this.breed = breed;

}

// add the sayHello method to the Dog class

// so all dogs now can say hello

Dog.prototype.sayHello = function() {

console.log("Hello this is a " + this.breed + " dog");

};

var yourDog = new Dog("golden retriever");

yourDog.sayHello();

var myDog = new Dog("dachshund");

myDog.sayHello();

**Output:**

Hello this is a golden retriever dog

Hello this is a dachshund dog

**So Meta I Can't Take It!**

Do you remember how we said every JavaScript object has some baggage associated with it? Part of this baggage was the hasOwnProperty method available to all objects. Now let's see where this came from...

If we have just a plain object (i.e., not created from a class constructor), recall that it automatically inherits from Object.prototype. Could this be where we get hasOwnProperty from? How can we check?

Let's first see what type Object.prototype is. Do this in [line 2](javascript:void(0)) and save it into prototypeType.

If all goes well, you should realize that Object.prototype itself is an object! And since all objects have the hasOwnProperty method, it's pretty easy to check if hasOwnProperty comes from Object.prototype. Do this in [line 6](javascript:void(0)) and the result may be surprising.

To see what type Object.prototype is, we should use typeof Object.prototype.

The property we want to check for is actually "hasOwnProperty", so [line 6](javascript:void(0)) should look like:

Object.prototype.hasOwnProperty("hasOwnProperty");

**Script:**

// what is this "Object.prototype" anyway...?

var prototypeType = typeof Object.prototype;

console.log(prototypeType);

// now let's examine it!

var hasOwn = Object.prototype.hasOwnProperty("hasOwnProperty");

console.log(hasOwn);

**Output:**

object

true

**Private Eye**

Recall that:

* **Public** properties can be accessed from outside the class
* **Private** properties can only be accessed from within the class

Using constructor notation, a property declared as this.property = "someValue;" will be public, whereas a property declared with var property = "hiddenValue;" will be private.

In this exercise, hit run and you'll see that all your grades are exposed! You really just want people to know your overall GPA.

Modify the StudentReport class so that no grades will be printed to the console in the for-in loop.

However, getGPA should still function properly in the last line.

**You should** be changing public variables (this.grade) to private variables (var grade).

If we want getGPA to be able to be called from outside this class, should we change it to be private?

You should find yourself needing to modify getGPA itself. this.grade1 will not be available if you did not declare it previously. Perhaps changing it to simply grade1 will work?

**Script:**

function StudentReport() {

var grade1 = 4;

var grade2 = 2;

var grade3 = 1;

this.getGPA = function() {

return (grade1 + grade2 + grade3) / 3;

};

}

var myStudentReport = new StudentReport();

for(var x in myStudentReport) {

if(typeof myStudentReport[x] !== "function") {

console.log("Muahaha! "+myStudentReport[x]);

}

}

console.log("Your overall GPA is "+myStudentReport.getGPA());

**Output:** Your overall GPA is 2.3333333333333335

**Building a Cash Register**

**Shut the Shop!**

You are working for a large supermarket and the cash register has just failed. The boss is not happy as he can't make any money.

To save the day it happens that you let slip to your boss that you know JavaScript and can build a quick virtual cash register until head office sends support staff.

Your boss is over the moon and wants you to get started right away.

Create a new object called cashRegister with the property total initialized to 0.

Then change the property total to 2.99 using dot notation.

**Look to** the [Objects Lessons](http://www.codecademy.com/courses/objects-ii) or read about [Objects in the Glossary](http://www.codecademy.com/glossary/javascript#objects) if you have any trouble with this project!

**Since we** only need one cash register use object literal notation.

**Script:**

//Create the object called cashRegister

//and initialize its total property

var cashRegister = {

total: 0

};

//Using dot notation change the total property

cashRegister.total = 2.99;

**Output:** 2.99

**Manually Add It Up?**

Great! The bossman can see that you can tell the cash register the total. But we need the cash register to do more.

Your boss wants a way to manually add the cost of each item. We have written the add method for you. There are two things we should note.

1. We are using literal notation to include the method add.
2. We've used the += operator. This is a shorthand way of saying
3. this.total = this.total + itemCost;

In general, a += b; means "add b to a and put the result of that addition back into a. This is also available for the other basic arithmetic functions: -=, \*=, and /= do what you expect.

Use the add method to sum up the cost of the following four items.

1. Eggs 0.98
2. Milk 1.23
3. Magazine 4.99
4. Chocolate 0.45

If we only call the method once, it will just add the first item. So to add up the cost of four items, how many times will we have to call the method?

**Remember to** call a method of an object you would use objectName.methodName(anyParameter);.

**We have** to call the method four times. Each time, we should pass in a different number.

**Script:**

var cashRegister = {

total:0,

add: function(itemCost){

this.total += itemCost;

}

};

//call the add method for our items

var eggs = cashRegister.add(0.98);

var milk = cashRegister.add(1.23);

var magazine = cashRegister.add(4.99);

var chocolate = cashRegister.add(0.45);

//Show the total bill

console.log('Your bill is '+cashRegister.total);

**Output:** Your bill is 7.65

**Short-Term Memory**

But this method only works as long as you can remember the cost of every item in the store. We need something like a bar code scanner where just knowing the item name will automatically add the cost of that item to the total.

So we create a method called scan. This method takes some item parameter, and adds the cost of this item to the total. item is a string.

We also use a switch statement. Previously, we would have probably used multiple if-else statements. Here, things work in a similar way.

For example, if the item is "eggs" ([line 8](javascript:void(0))), we then call the add method, passing through 0.98 as the itemCost. This will add 0.98 to cashRegister.total. If instead the item is "milk" or "chocolate" or "magazine", the relevant itemCost is added. Note no default case is needed for this switch statement.

1. Write the add method which has a single parameter, itemCost. It will add the itemCost to the total.
2. We have partially written the scan method for you and started a switch statement. Add the following 2 items to the switch statement:

* "magazine", 4.99
* "chocolate", 0.45

Finally, use the scan method to buy "eggs" twice and a "magazine" three times.

**We know** add and scan are both methods because they use the keyword function and are found inside an object. Look to the previous exercise for hints on how to implement add.

**In the** scan method, we make use of the add method. We call it with this so that add refers to its own defined method.

**Previous switch statements** we've seen all had a default case. But that was because those were if-else cases where we needed a catch-all outcome. But for a scanner, this is not needed.

**The point** of this exercise is to show that we no longer have to remember the price of each item. The parameter we pass through is the item name, not the item cost.

**Comma** don't forget that a comma (,) will be needed after closing the new add method

**Script:**

var cashRegister = {

total:0,

//insert the add method here

add: function(itemCost){

this.total += itemCost;

},

scan: function(item) {

switch (item) {

case "eggs":

this.add(0.98);

break;

case "milk":

this.add(1.23);

break;

//Add other 2 items here

case "magazine":

this.add(4.99);

break;

case "chocolate":

this.add(0.45);

break;

}

return true;

}

};

//Scan 2 eggs and 3 magazines

cashRegister.scan("eggs");

cashRegister.scan("eggs");

cashRegister.scan("magazine");

cashRegister.scan("magazine");

cashRegister.scan("magazine");

//Show the total bill

console.log('Your bill is '+cashRegister.total);

**Output:** Your bill is 16.93

**I Have to Scan It More Than Once?**

Is that a smile on the boss's face? Well, there was one until he realized that your system requires every item to be scanned individually. He finds this pretty inefficient and you probably agree. Let's get real—it was pretty annoying having to call the scan method five times in the previous exercise!

What can we do? What is the limitation of the scan method? Well, it has just one parameter, item, and you can't specify anything related to quantity.

Modify the scan method such that if we tell it the quantity of each item, it will be able to add the right amount to the total. Since you currently tell scan nothing about quantity, it may be useful to create another parameter.

Scan 4 of each item using your improved scan method. Previously we would have needed to call scan 16 times! Now it is down to 4.

**Modify the** scan method to have a second parameter, quantity. Now, we must call it using something like cashRegister.scan("chocolate", 1);

**In the** scan method, we currently only add 0.45 when "chocolate" is passed. How can we improve this? Let's multiply this by the quantity parameter. And we need to do this for each item.

**This means**, for "chocolate", we need the code to be :

this.add(0.45 \* quantity);

**Unit Test** Our test expects the second parameter to be quantity as the examples above.

**Script:**

var cashRegister = {

total:0,

add: function(itemCost){

this.total += itemCost;

},

scan: function(item, quantity) {

switch (item) {

case "eggs": this.add(0.98 \* quantity); break;

case "milk": this.add(1.23 \* quantity); break;

case "magazine": this.add(4.99 \* quantity); break;

case "chocolate": this.add(0.45 \* quantity); break;

}

}

};

// scan each item 4 times

cashRegister.scan("eggs", 4);

cashRegister.scan("milk", 4);

cashRegister.scan("magazine", 4);

cashRegister.scan("chocolate", 4);

//Show the total bill

console.log('Your bill is '+cashRegister.total);

**Output:** Your bill is 30.6

Bleep Bleep

The boss looks down at his pager to see Register 8 needs assistance. They have scanned an item too many times and need to void the last transaction.

So he turns to you and says: "Okay JavaScript Ninja! What do we do now?!"

We need to keep track of how much the last transaction was. Modify the add method to keep track of the amount of the last transaction. This should be tracked in a new property, lastTransactionAmount.

Add a method called voidLastTransaction that subtracts the last amount transacted from total.

Then use the new method to void the last item we scanned. Finally, scan only 3 of the same item instead.

**Remember that** when you add a property or method to an object in literal notation, you should follow it with a , (comma) unless it is the last item of the object.

**How should** you update lastTransactionAmount? Each time add runs, you should change the value of lastTransactionAmount to itemCost so that lastTransactionAmount always refers to the cost of the last transaction.

**Script:**

var cashRegister = {

total:0,

//Dont forget to add your property

add: function(itemCost) {

this.total += itemCost;

this.lastTransactionAmount = itemCost;

},

scan: function(item,quantity) {

switch (item) {

case "eggs": this.add(0.98 \* quantity); break;

case "milk": this.add(1.23 \* quantity); break;

case "magazine": this.add(4.99 \* quantity); break;

case "chocolate": this.add(0.45 \* quantity); break;

}

return true;

},

//Add the voidLastTransaction Method here

voidLastTransaction: function(lastTransactionAmount){

this.total -= this.lastTransactionAmount;

}

};

cashRegister.scan('eggs',1);

cashRegister.scan('milk',1);

cashRegister.scan('magazine',1);

cashRegister.scan('chocolate',4);

//Void the last transaction and then add 3 instead

cashRegister.voidLastTransaction();

cashRegister.scan('chocolate',3);

//Show the total bill

console.log('Your bill is '+cashRegister.total);

**Output:** Your bill is 8.55

**Over the Moon**

Great! The store is ticking along making money again. The boss is so happy you have just been given a bonus staff discount to the value of 20%.

However the current system doesn't know how to apply the different levels of staff discount that apply. Now the rest of the staff is not happy and demanding you make improvements!

Let's sort it out so that staff can get their well deserved discount.

Create an object constructor called StaffMember which takes two parameters—name and discountPercent. And then have the (public) properties name and discountPercent equal the parameters.

To help, we have already created two employees using this constructor. Sally and Bob already have their staff discount set up: Sally getting 5% off and Bob getting 10%.

Create a new instance of the object for yourself called me with your massive staff discount bonus of 20%.

**Nothing new** is being introduced here! We think you got this!

**Script:**

// create a constructor for the StaffMember class

var StaffMember = function(name, discountPercent){

this.name = name;

this.discountPercent = discountPercent;

};

var sally = new StaffMember("Sally",5);

var bob = new StaffMember("Bob",10);

//Create a StaffMember for yourself called me

var me = new StaffMember("Me", 20);

**Output:** nothing

**You Deserved It!**

Whew! It's been a long day fixing cash registers and now let's actually apply our well-earned discount. Now that we have our objects representing the staff, let's update our cash register to actually apply the discount.

On [line 10](javascript:void(0)) create a new object called me of type StaffMember for yourself with a staff discount of 20%

Create a new method called applyStaffDiscount in the cashRegister object which accepts a parameter employee. When this method is called, cashRegister should apply the staff member's discountPercent to total.

Under the comment, 'Apply your staff discount by passing the me object, call your new applyStaffDiscount and pass the object me.

**To calculate** the percent discount use:

this.total -= this.total \* (employee.discountPercent / 100);

We need to divide by 100 because we need to convert our percentage into a decimal fraction.

**Script:**

function StaffMember(name,discountPercent){

this.name = name;

this.discountPercent = discountPercent;

}

var sally = new StaffMember("Sally",5);

var bob = new StaffMember("Bob",10);

// Create yourself again as 'me' with a staff discount of 20%

var me = new StaffMember("Me", 20);

var cashRegister = {

total:0,

lastTransactionAmount: 0,

add: function(itemCost){

this.total += (itemCost || 0);

this.lastTransactionAmount = itemCost;

},

scan: function(item,quantity){

switch (item){

case "eggs": this.add(0.98 \* quantity); break;

case "milk": this.add(1.23 \* quantity); break;

case "magazine": this.add(4.99 \* quantity); break;

case "chocolate": this.add(0.45 \* quantity); break;

}

return true;

},

voidLastTransaction : function(){

this.total -= this.lastTransactionAmount;

this.lastTransactionAmount = 0;

},

// Create a new method applyStaffDiscount here

applyStaffDiscount : function(employee){

this.total -= (this.total\*(employee.discountPercent/100));

}

};

cashRegister.scan('eggs',1);

cashRegister.scan('milk',1);

cashRegister.scan('magazine',3);

// Apply your staff discount by passing the 'me' object

// to applyStaffDiscount

cashRegister.applyStaffDiscount(me);

// Show the total bill

console.log('Your bill is '+cashRegister.total.toFixed(2));

console.log('Your bill is '+cashRegister.total);

**Output:**

Your bill is 13.74

Your bill is 13.744