

JAVASCRIPT DEVELOPMENT

Sasha Vodnik, Instructor

HELLO!

1. Pull changes from the `svodnik/JS-SF-10-resources` repo to your computer:
 - Open the terminal
 - `cd` to the `Documents/JSD/JS-SF-10-resources` directory
 - Type `git pull` and press **return**
2. In your code editor, open the following folder:
`Documents/JSD/JS-SF-10-resources/03-conditionals-functions`

JAVASCRIPT DEVELOPMENT

CONDITIONALS & FUNCTIONS

LEARNING OBJECTIVES

At the end of this class, you will be able to

- Use Boolean logic to combine and manipulate conditional tests.
- Use `if/else` conditionals to control program flow.
- Differentiate among `true`, `false`, `truthy`, and `falsy`.
- Describe how parameters and arguments relate to functions
- Create and call a function that accepts parameters to solve a problem
- Define and call functions defined in terms of other functions
- Return a value from a function using the `return` keyword
- Define and call functions with argument-dependent return values

AGENDA

- Comparison operators
- Logical operators
- Conditional statements
- Functions

CONDITIONALS & FUNCTIONS

WEEKLY OVERVIEW

WEEK 2

Data Types & Loops / Conditionals & Functions

WEEK 3

(holiday) / Scope & Objects

WEEK 4

Slackbot Lab / JSON & DOM

EXIT TICKET QUESTIONS

1. I am a little confused about the functions within the iterating things. Like the straight-up for loop makes a lot of sense to me, but the other ones (forEach etc), I am confused when I need to use "return" and exactly what they are doing.
2. let vs var, I know we might get to this but are both these things able to be used by browsers? Do certain types of loops/functions warrant a var vs a let?
3. Can arrays be type-sensitive with the data types they contain? In today's example, we had instances of arrays containing both string and number elements. Can you declare an empty array to be of a specific type (i.e: string only) so that there is an error if one were to append a number to it?
4. What exercises can I do to ensure my confidence with forLoops?

How to you decide what to have for dinner?

- What factors do you consider?
- How do you decide between them?

CONDITIONALS

CONDITIONAL STATEMENTS

- Decide which blocks of code to execute and which to skip, based on the results of tests that we run
- Known as **control flow statements**, because they let the program make decisions about which statement should be executed next, rather than just going in order

if STATEMENT

```
if (expression) {  
  code  
}
```



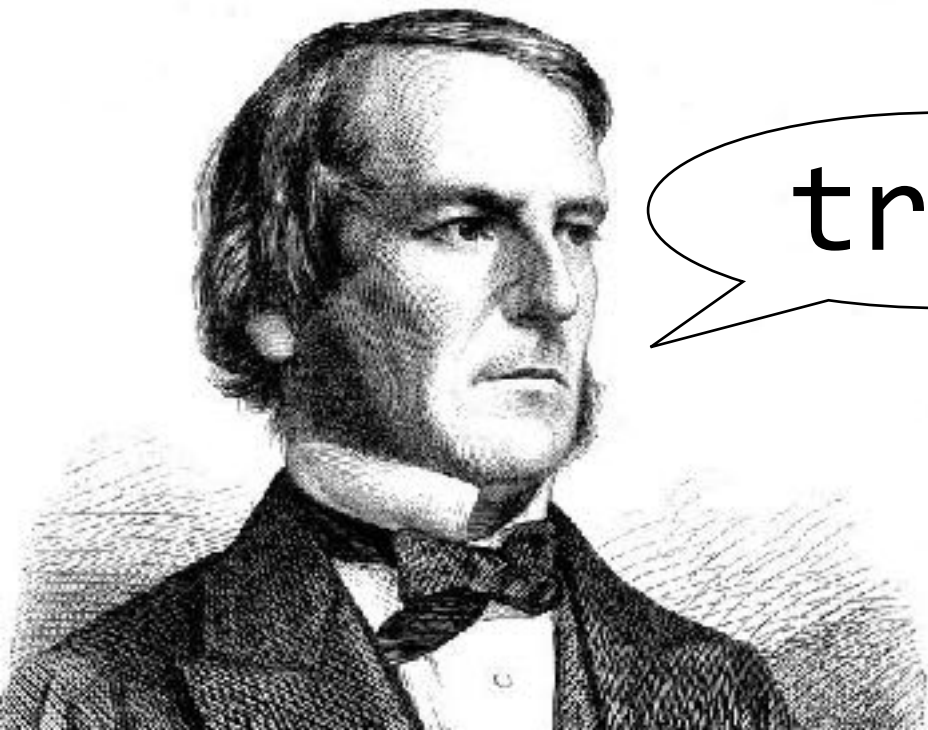
```
if (expression) { code }
```



- JavaScript doesn't care about white space, so these are equivalent.
- **However**, putting block contents on a separate line is best practice for code readability.

BOOLEAN VALUES

- A separate data type
- Only valid values are true or false
- Named after George Boole, a mathematician



true

false



COMPARISON OPERATORS

>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
===	strict equal (use this one)
==	coercive equal (AVOID)
!==	strict not equal (use this one)
!=	coercive not equal (AVOID)

TYPE COERCION

- JavaScript “feature” that attempts to make it possible to run a comparison operation on two objects of different data types
- Results are sometimes unpredictable
- `==` and `!=` use coercion if necessary to arrive at an answer — avoid them
- `===` and `!==` do not use coercion — best practice is to use these rather than the coercive operators

if STATEMENT

```
let weather = "sunny";  
  
if (weather === "sunny") {  
    console.log("Grab your sunglasses");  
}
```

if/else STATEMENT

```
var weather = "sunny";  
  
if (weather === "sunny") {  
    console.log("Bring your sunglasses");  
} else {  
    console.log("Grab a jacket");  
}
```


else if STATEMENT

```
var weather = "sunny";

if (weather === "sunny") {
    console.log("Bring your sunglasses");
} else if (weather === "rainy") {
    console.log("Take an umbrella");
} else {
    console.log("Grab a jacket");
}
```

TERNARY OPERATOR

- A compact if/else statement on a single line
- “ternary” means that it takes 3 operands

TERNARY OPERATOR

(expression) ? trueCode : falseCode;

TERNARY OPERATOR

- Can produce one of two values, which can be assigned to a variable in the same statement

```
let name = (expression) ? trueCode : falseCode;
```

BLOCK STATEMENTS

- Statements to be executed after a control flow operation are grouped into a block statement
- A block statement is placed inside braces

```
{  
    console.log("Grab your sunglasses.");  
    console.log("Enjoy the beach!");  
}
```

LOGICAL OPERATORS

- Operators that let you chain conditional expressions

&&	AND	Returns true when both left and right values are true
	OR	Returns true when at least one of the left or right values is true
!	NOT	Takes a single value and returns the opposite Boolean value

TRUTHY AND FALSY VALUES



FALSY VALUES

- All of these values become `false` when converted to a Boolean:

`false`

`0`

`""`

`NaN`

`null`

`undefined`

- These are known as **falsy values** because they are equivalent to `false`

TRUTHY VALUES

- All values other than `false`, `0`, `""`, `NaN`, `null`, and `undefined` become `true` when converted to a Boolean
- All values besides these six are known as **truthy values** because they are equivalent to `true`
- `'0'` and `'false'` are both **truthy**! (Why?)

BEST PRACTICES

- Convert to an actual Boolean value
 - Adding ! before a value returns the *inverse* of the value as a Boolean
 - Adding !! before a value gives you the *original* value as a Boolean
- Check a value rather than a comparison



just use
`if (name)`



instead of
`if (name === false)`

LAB — CONDITIONALS



EXERCISE

TYPE OF EXERCISE

‣ Pair

LOCATION

‣ `starter-code > 1-ages-lab`

TIMING

15 min

1. Write a program that outputs results based on users' age. Use the list of conditions in the `app.js` file.
2. BONUS 1: Rewrite your code to allow a user to enter an age value, rather than hard-coding it into your program. (Hint: Read up on the [window.prompt method](#).)
3. BONUS 3: Rewrite your code to use a [switch statement](#) rather than if and else statements.

FUNCTIONS

FUNCTIONS



GROUP STEPS

Allow us to group a series of statements together to perform a specific task



REUSABLE

We can use the same function multiple times



STORE STEPS

Not always executed when a page loads.
Provide us with a way to 'store' the steps needed to achieve a task.

CONDITIONALS & FUNCTIONS

**DRY =
DON'T
REPEAT
YOURSELF**



FUNCTION DECLARATION SYNTAX

```
function name(parameters) {  
    // do something  
}
```

FUNCTION DECLARATION EXAMPLE

```
function speak() {  
    console.log("Hello!");  
}
```


FUNCTION EXPRESSION SYNTAX

```
let name = function(parameters) {  
    // do something  
};
```

FUNCTION EXPRESSION EXAMPLE

```
let speak = function() {  
  console.log("Hello!");  
};
```

ARROW FUNCTION SYNTAX

```
let name = (parameters) => {  
    // do something  
};
```

ARROW FUNCTION EXAMPLE

```
let speak = () => {  
  console.log("Hello!");  
};
```

CONDITIONALS & FUNCTIONS

CALLING A FUNCTION

```
function pickADescriptiveName() {  
    // do something  
}
```

To run the function, we need to *call* it. We can do so like this:

```
pickADescriptiveName();
```

pickADescriptiveName() + parentheses

EXERCISE — WRITING FUNCTIONS



EXERCISE

KEY OBJECTIVE

- Practice defining and executing functions

TYPE OF EXERCISE

- Individual/paired

LOCATION

- `starter-code` > `3-functions-exercise` (part 1)

EXECUTION

4 min

1. Follow the instructions under Part 1

FUNCTION EXPRESSION VS FUNCTION DECLARATION

- Function expressions define functions that can be used anywhere in the scope where they're defined.
- You can call a function that is defined using a function declaration before the part of the code where you actually define it.
- Function expressions must be defined before they are called.


PARAMETERS

DOES THIS CODE SCALE?


```
function helloVal () {  
  console.log('hello, Val');  
}  
  
function helloOtto () {  
  console.log('hello, Otto')  
}
```

USING A PARAMETER

```
function sayHello(name) {  
  console.log('Hello ' + name);  
}
```



```
sayHello('Val');  
=> 'Hello Val'
```



```
sayHello('Otto');  
=> 'Hello Otto'
```

USING MULTIPLE PARAMETERS

multiple parameter names
separated by commas




```
function sum(x, y, z) {  
  console.log(x + y + z)  
}
```

```
sum(1, 2, 3);  
=> 6
```

USING DEFAULT PARAMETERS

default value to set for parameter if no argument is passed when the function is called



```
function multiply(x, y = 2) {  
  console.log(x * y)  
}
```

```
multiply(5, 6);
```

```
=> 30 // result of 5 * 6 (both arguments)
```

```
multiply(4);
```

```
=> 8 // 4 (argument) * 2 (default value)
```

EXERCISE — READING FUNCTIONS



EXERCISE

KEY OBJECTIVE

- Given a function and a set of arguments, predict the output of a function

TYPE OF EXERCISE

- Groups of 2 - 3

LOCATION

- `starter-code` > `3-functions-exercise` (part 2)

EXECUTION

3 min

1. Look at Part 2 A and B. Predict what will happen when each function is called.

EXERCISE — READING FUNCTIONS



EXERCISE

KEY OBJECTIVE

- Create and call a function that accepts parameters to solve a problem

TYPE OF EXERCISE

- Groups of 2 - 3

LOCATION

- starter-code > 3-functions-exercise (part 3)

EXECUTION

8 min

1. See if you can write one function that takes some parameters and combines the functionality of the *makeAPizza* and *makeAVeggiePizza* functions.
2. BONUS: Create your own function with parameters. This function could do anything!

EXERCISE — FUNCTIONS



EXERCISE

KEY OBJECTIVE

- Describe how parameters and arguments relate to functions

TYPE OF EXERCISE

- Turn and Talk

EXECUTION

1 min

1. Summarize why we would use functions in our programs. What purpose do they serve?
2. What is a parameter? What is an argument? How are parameters and arguments useful?

THE `return` STATEMENT

return STATEMENT

- Ends function's execution
- Returns a value — the result of running the function

return **STOPS A FUNCTION'S EXECUTION**

```
function speak(words) {  
  return words;  
  
  // The following statements will not run:  
  let x = 1;  
  let y = 2;  
  console.log(x + y);  
}
```

console.log() vs return



`console.log()`

VS



`return`

- ▶ Write a value at any point in a program to the browser console
- ▶ Helpful for developer in debugging
- ▶ Not seen by user or used by app

- ▶ Sends a value back wherever the current statement was triggered
- ▶ Can use a function to get a value and then use that value elsewhere in your app
- ▶ Does not appear in the console unless you're executing commands there

CONDITIONALS & FUNCTIONS

return in action

call `sum()` function,
passing 3 and 4 as
arguments

```
let z = sum(3,4);
```

with `x=3` and `y=4`,
return the result
of `x + y`, which is 7

```
function sum(x,y) {  
  return x + y;  
}
```

```
z = 7
```

Exit Tickets!

(Class #3)

LEARNING OBJECTIVES – REVIEW

- Use Boolean logic to combine and manipulate conditional tests.
- Use `if/else` conditionals to control program flow.
- Differentiate among `true`, `false`, `truthy`, and `falsy`.
- Describe how parameters and arguments relate to functions
- Create and call a function that accepts parameters to solve a problem
- Define and call functions defined in terms of other functions
- Return a value from a function using the `return` keyword
- Define and call functions with argument-dependent return values

NEXT CLASS PREVIEW

Scope & Objects

- Determine the scope of local and global variables
- Create a program that hoists variables
- Identify likely objects, properties, and methods in real-world scenarios
- Create JavaScript objects using object literal notation

Q&A