

# JAVASCRIPT DEVELOPMENT

Sasha Vodnik, Instructor

# HELLO!

1. Pull changes from the `svodnik/JS-SF-7` repo to your computer:
  - Open the terminal
  - `cd` to the `JSD/JS-SF-7-resources` directory
  - Type **`git pull`** and press **return**
2. In your code editor, open the following folder:  
`JSD/JS-SF-7-resources/03-conditionals-functions/starter-code`

**JAVASCRIPT DEVELOPMENT**

---

# **CONDITIONALS AND FUNCTIONS**

# LEARNING OBJECTIVES

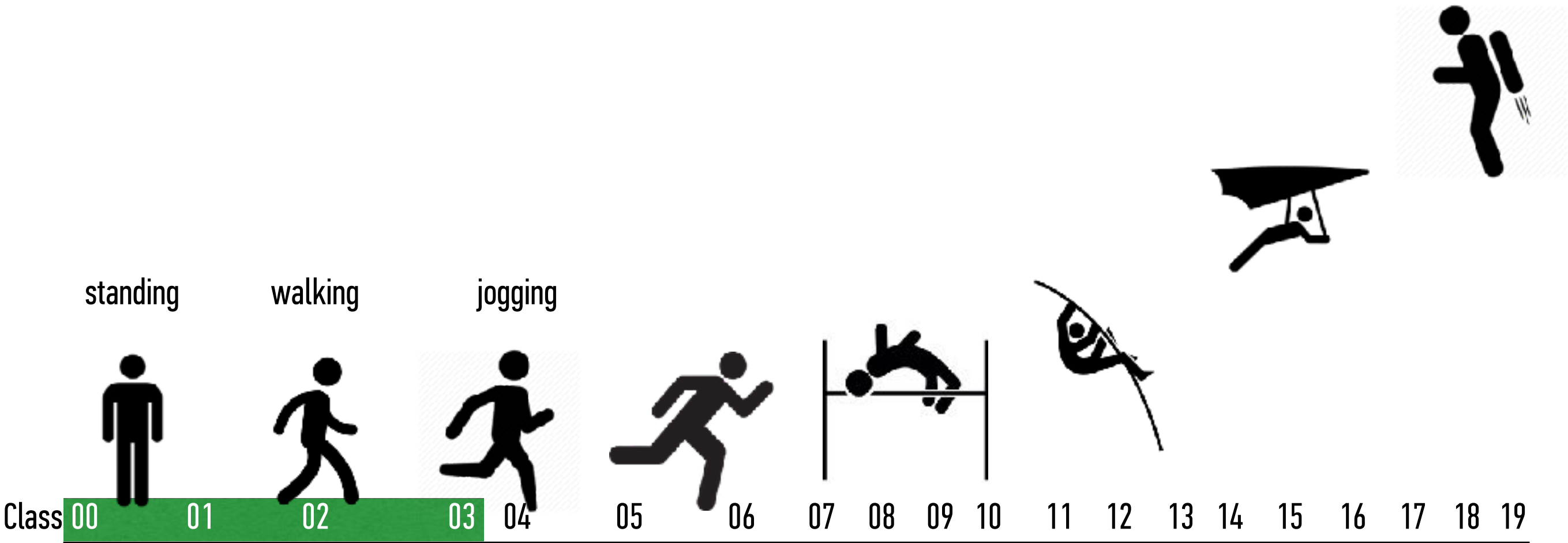
At the end of this class, you will be able to

- Use Boolean logic to combine and manipulate conditional tests.
- Use `if/else` conditionals to control program flow based on Boolean tests.
- 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
- Function declarations & function expressions
- Parameters
- The return statement

# Where we are



# Checkin and questions

- The **most significant thing I learned** about Data Types and Loops is \_\_\_\_\_.
- My **biggest outstanding question** about Data Types and Loops is \_\_\_\_\_.

## **EXIT TICKET FEEDBACK**

- “Need more time coding and seeing correct answers”
- “Folks need to quiet down when the class resumes”



# **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 }
```

# BOOLEAN VALUES

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



# 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**

```
var 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

```
var 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**

# LOGICAL OPERATORS

- Operators that let you chain conditional expressions

&&	AND	Returns <code>true</code> when both left and right values are <code>true</code>
	OR	Returns <code>true</code> when at least one of the left or right values is <code>true</code>
!	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`

## **VERIFYING TRUTHINESS AND FALSINESS**

- 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
  - » This is a simple shortcut to verifying truthiness and falsiness

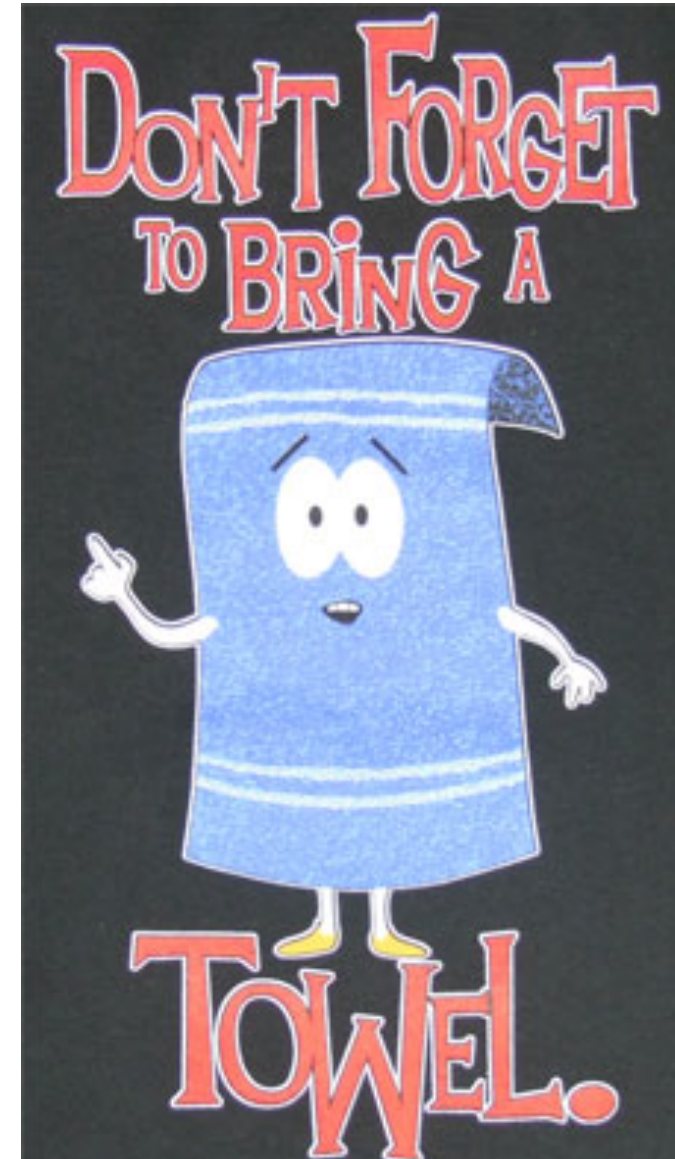
# **LAB: AGES**

# FUNCTIONS

- A **function** is a reusable statement, or a group of reusable statements, that can be called anywhere in a program.
- A function avoids the need to rewrite the same statement(s) over and over.
- Functions enable software developers to segment large, unwieldy applications into smaller, more manageable pieces.

# DRY (DON'T REPEAT YOURSELF)

- A key tenet of software engineering.
- Our goal is to create programs with as little code as possible, while maintaining complete clarity.
- Functions are a critical component of doing this.



# FUNCTION DECLARATION SYNTAX

```
function name(parameters) {  
    statement;  
}
```



# FUNCTION DECLARATION EXAMPLE

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

# FUNCTION EXPRESSION SYNTAX

```
var name = function(parameters) {  
    statement;  
};
```

# FUNCTION EXPRESSION EXAMPLE

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

# ARROW FUNCTIONS

- Part of newest versions of JavaScript
- You may see them in online examples, but we won't be using them in class
- They simplify some complex syntax

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

# CALLING A FUNCTION

- Invoking, or **calling**, a function executes the code defined inside this function.
- Calling a function is different from defining it. A function is not called when it's defined.
- You call a function by specifying the function name with parentheses after it

# 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.

# OBJECTS

- A separate data type
- defined by code enclosed in braces { } — but not the same thing as a function
- Can contain properties and methods
- Functions are used to define methods

# OBJECT EXAMPLE

```
var person = {  
  fName: 'Kamala',  
  lName: 'Harris',  
  speak: function() {  
    console.log("Hello world!");  
  }  
}
```

```
person.speak()  
=> 'Hello world!'
```



# **PARAMETERS**


# DOES THIS CODE SCALE?

```
function helloSean () {  
  console.log('hello, Sean');  
}
```

```
function helloKellyanne () {  
  console.log('hello, Kellyanne')  
}
```


# USING A PARAMETER

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



A red arrow points from the word "parameter" to the variable "name" in the function definition.

```
sayHello('Sean');
```



A red arrow points from the word "argument" to the string "Sean" in the function call.

```
=> 'Hello Sean'
```

```
sayHello('Kellyanne');
```

```
=> 'Hello Kellyanne'
```

# USING MULTIPLE PARAMETERS

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

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

# **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:
```

```
  var x = 1;  
  var y = 2;  
  console.log(x + y)  
}
```

# **LAB: DICE**



## **LEARNING OBJECTIVES – REVIEW**

- Use `if/else` conditionals to control program flow based on Boolean tests.
- Use Boolean logic to combine and manipulate conditional tests.
- 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 and Closures**

- Determine the scope of local and global variables
- Create a program that hoists variables
- Understand and explain closures

# **Exit Tickets!**

# Q&A