Unit 3 · Lesson 1 — Variables & Data Types

Class: Introduction to Web Development · **Unit:** Unit 3 — JavaScript Fundamentals I **Topic:** Variables & Data Types · **Lesson:** 1 of 14 · **Type:** Direct Instruction / Guided Practice

Learning Standards

Common Core

- **CCSS.MATH.PRACTICE.MP2** — Reason abstractly and quantitatively.
- **CCSS.MATH.PRACTICE.MP6** — Attend to precision.
- **CCSS.ELA-LITERACY.L.9-10.6** — Acquire and use accurately domain-specific words.

NYS Computer Science & Digital Fluency

- **9-12.CT.1** — Create computational artifacts.
- **9-12.CT.4** — Implement a program that uses variable types.

CDOS — Standard 3b: Career Majors. Students who choose a career major will acquire the career-specific technical knowledge/skills necessary to progress toward gainful employment, career advancement, and success in postsecondary programs.

Materials

- Chromebooks / laptops
- Google Classroom
- Slide deck (Lesson 3.1)
- GitHub repository for Lesson 3.1
- CodeDash
- Exit Ticket (Google Forms)
- Projector / display

Vocabulary

JavaScript, variable, declare, assign, reassign, value, `let`, data type, number, string, `camelCase`, naming convention, syntax error, comment, `//`, console, `console.log()`

Differentiation & Accommodations

SPED

- Provide a “Variable Anatomy” card: `let variableName = value;` with each part labeled and color-coded
- Use a physical analogy with actual containers (boxes with labels) to demonstrate variables before coding
- Chunk the Code Solo into 5 micro-tasks: (1) Declare a number variable, (2) Declare a string variable, (3) Declare a blank variable and fill it later, (4) Reassign a variable, (5) Label each section with a `//` comment
- Provide a naming conventions reference card showing valid vs. invalid variable names with examples
- Allow extended time on Code Solo; reduce required tasks from full set to core 3
- Display a permanent syntax poster: `let myVariable = "hello";` visible throughout the JS units

ELL

- Pre-teach “variable” with a physical analogy: “A variable is a labeled box. The label is the name. The thing inside is the value. You can swap what’s inside.”
- Provide bilingual vocabulary cards for: variable, declare, assign, value, string, number, with simple definitions and visuals
- Use color-coded projected code: keyword (`let`) in purple, variable name in blue, value in green
- Provide sentence frames: “A variable stores `___` .” / “The data type of `___` is `___` because `___` .”
- Pre-teach `camelCase` by connecting to the student’s name: “ `juanCarlos` — no spaces, capitalize each new word after the first.”
- Pair ELL students with a coding partner during Code Solo for real-time vocabulary clarification; allow `//` comments to be written in the student’s home language first, then translated with the partner

Lesson

Opening Task

Do Now (5 min): Project the problem from the slide deck exactly as written — no instructions beyond the prompt:

`x = 2`
`5x`

Prompt: *"How do we solve this in math? Be prepared to share the **process** you used, not just the answer."* Students work on paper or a whiteboard for 2–3 minutes, then 2–3 students narrate their steps at the board. Land the class on the process, in these words: **`x` is holding the value `2`, so wherever we see `x`, we substitute that value in** — `5(2) = 10`. Write the substitution step on the board and leave it up; the lesson returns to it in five minutes.

Ask (DOK 2): *"In this problem, what exactly is `x`? What does it hold, and could it hold a different value tomorrow?"* Chart 2–3 student definitions of "variable" in their own words. Look for the idea that `x` is a **name that stands in for a value**, and that the value can change from one problem to the next — that is the whole concept, already in students' hands before any code appears.

Bridge out of the Do Now: *"Today we use this exact same idea in code. Programmers store information in variables too — they just write it a little differently."*

> **Academic Integration (Math — MP2: Abstract Reasoning):** Variables in programming are the same abstraction students already use in algebra. In math, `x = 2` means `x` holds the value `2`, and evaluating `5x` means substituting that stored value in. In JavaScript, `let x = 2;` does the same thing, and `x * 5` substitutes the same way. The teacher should name this explicitly as the segue out of the Do Now: "You've been using variables since algebra. `let score = 100` is the same idea as 'let `score` equal 100,' and reassigning `score = 150` is like moving to the next problem where `x` holds a new value. The concept is identical — only the syntax is new."

Aim

What are variables and data types, and how do developers use them to store information in a program?

Motivation

Start from the Do Now and generalize it: "In math, we stored the value `2` inside `x` so we could use it later — that's exactly what code does. Programmers store information in variables so the program can use it later, in as many places as it needs." Say plainly that this is the whole job of a variable in both subjects: **give a piece of information a name so you can reuse it.**

Then open it up to the class: *"So what kinds of information might a program need to store?"* Take 5–6 answers and chart them on the board. Seed the brainstorm with the video game example if it stalls: "Think about a game you play. To keep the game running, what does it have to remember about you?" — the player's name, their score, their health, the level they're on, how many lives are left. Push for non-game answers too: a username on a login screen, the number of items in a shopping cart, the temperature in a weather app. Land the point: every app, game, and website is storing information in variables the entire time it's running. Today students create their own.

Instructional Objectives — SWBAT

- Define variables, explain their purpose, and declare and initialize them using the `let` keyword.
- Use JavaScript naming conventions (`camelCase`, no spaces, descriptive names) when creating variables.
- Identify and explain whether a variable is a number or a string data type based on its value.
- Reassign a variable's value without re-declaring it, and explain why re-declaring causes an error.
- Use `console.log()` to display and explain variable values in the console.
- Write single-line comments (`//`) to label and explain code.

Presentation

From Math to Code (4 min) — Return to the board and the Do Now. Restate the process: `x` held `2`, so `5x` became `5(2) = 10` — the value was **substituted in** wherever the name appeared. Then say plainly: "Variables in programming work almost exactly the same way. A variable is a name that holds a value, and everywhere you use the name, the program substitutes the value." Introduce the box analogy from the slide: a variable is a physical box with a label on the outside and an item inside.

- The **label** on the box is the variable's **name**.
- The **item inside** the box is the stored **value**.

On the slide, the box is labeled `name` and the item inside is `shawn`. The one difference from math: in math, `x` usually holds a number; in programming, the box can hold text, numbers, and much more.

Ask (DOK 1): *"In `let score = 100;`, which part is the label on the box, and which part is the item inside?"* Quick cold-call; every student should be able to answer before moving on.

Declaring a Variable (5 min) — Walk the syntax one piece at a time, matching the slide:

```
let variableName = value;
```

- `let` — the keyword that tells the computer you are **creating** a variable.
- `variableName` — the **label** on your container.
- `=` — the **assignment operator**; it stores the value in the box. (Say it out loud: in code, `=` means "put this in there," not "these are equal.")
- `value` — the data placed inside the box.
- `;` — the semicolon that ends the statement.

Naming rules, projected and left up:

- Must start with a letter — no numbers first.
- No spaces; use `camelCase` for multi-word names (`playerScore`, not `player score`).

- Make names descriptive: `score` tells the reader something; `x` does not.

Then show that a variable can also start **empty**, straight from the slide:

```
let variableName;
```

Nothing after the name but a semicolon. This creates a blank box for later use — useful when you know you'll need to store something but don't have the value yet (a score before the game starts, a name before the player types it).

Ask (DOK 2): *"Why is it important to name a variable `playerScore` instead of `x`? Who benefits from the good name?"* Push past "so I remember" to the real answer: **other developers**, and your future self reading this code in three weeks. Nothing about `x` is wrong to the computer — the name is for humans.

> **Academic Integration (ELA — Domain-Specific Vocabulary):** Variable naming is a vocabulary precision exercise. Just as ELA teaches students to choose precise words over vague ones ("sprinted" vs. "went"), JavaScript naming conventions teach students to choose descriptive names (`playerScore` vs. `x`). The teacher should connect: "In English, you choose specific words to communicate clearly. In JavaScript, you choose specific variable names so your code communicates clearly to other developers."

Code Along — Part 1 (3 min) — Students open the Code Along activity for Unit 3 · Lesson 1 in CodeDash. The file opens with one variable already declared and two numbered tasks:

```
// CODE ALONG, PART 1
let favoriteColor;

// 1. Create a variable called `favoriteFood`.


// 2. Create a variable called `favoriteNumber`.
```

Narrate the given line first: "`let favoriteColor;` creates the box and labels it — but the box is still empty. That's legal, and we'll fill it in a few minutes." Then students write the two declarations themselves. Circulate for the two errors that show up immediately: a missing semicolon, and `favorite food` written with a space instead of `camelCase`. Leave the values empty for now — the next segment is what decides *what kind* of value goes in each box.

Data Types (5 min) — A data type is what *kind* of information a value is; it decides how the computer treats that value. Two types today, straight from the slide:

```
let cash = 10;          // number — no quotes, used for math
let name = "Charles";   // string — text wrapped in quotes
```

- **Numbers** have no quotes: `42`, `3.14`. You can do math with them.
- **Strings** are text wrapped in quotes: `"hello"`, `'world'`. Either single or double quotes work, as long as they match.

Then run the demo that makes it stick:

```
console.log(5 + 5);       // 10  — number addition
console.log("5" + "5");   // "55" — string concatenation (glued together)
```

Same symbol, completely different behavior — because the data types are different. Let the surprise land before explaining it.

Ask (DOK 2): *"What is the difference between `5` and `"5"`? Predict what `"10" + 5` prints before I run it."* Take predictions from three students, then run it (`"105"`). Name the takeaway: quotes are not decoration — they change what the value **is**.

Assigning & Updating Values (4 min) — A variable that was declared empty gets filled the same way a full one gets changed: with the assignment operator alone — **do not type `let` again**:

```
let score;      // 1. create the variable — first time only
score = 10;     // 2. update the value — no `let`
score = 25;     // 3. update it again
console.log(score); // 25
```

Show the common mistake and let students see the actual error message:

```
let score = 10;
let score = 25; // ⚠ SyntaxError: Identifier 'score' has already been declared
```

Say it plainly: "`let` builds the box. You only build it once. After that you're just swapping what's inside." Tie it back to the Do Now one last time: in math, moving to the next problem where `x = 7` doesn't create a new `x` — it's the same `x` holding a new value.

Code Along — Part 2 (5 min) — Back to the CodeDash file. Students now have everything they need to finish it:

```
// CODE ALONG, PART 2
//  3. Assign values for each variables.
// - Use a string value of your favorite color.
// - Use a string value of your favorite food.
// - Use a number value of your favorite number.

// 4. Console log each variable on a new line.
```

Do the first one together on the projector — `favoriteColor = "teal";` — and name what just happened: "No `let`. The box already exists; I'm putting something in it." Students finish the other two on their own, then log all three. Before running, ask for a prediction: *"Which of your three values will show up in the console with quotes around it, and which won't?"* Run it — the two strings print as text, the number prints bare. That difference IS the data type, visible on screen. Circulate for the three predictable errors: a second `let` on an existing variable, `favoriteNumber = "7";` with quotes (a string that only looks like a number), and `console.log(favoriteFood)` misspelled — send students to the console to read `ReferenceError: favoriteFod is not defined` and translate it out loud as "you asked for a box that doesn't exist."

Comments (3 min) — Introduce comments as notes written for **humans**; JavaScript skips them entirely. Anything after `//` on a line is a comment:

```
// This section sets up the player
let playerName = "Mario";
let playerScore = 0; // points start at zero
```

Two uses students will rely on immediately: **labeling** sections of code so it's readable, and **commenting out** a line to temporarily turn it off while testing — a debugging move they'll use all year.

Ask (DOK 2): Project this and take predictions before running it:

```
let score = 10;
// score = 25;
console.log(score);
```

"What prints — `10` or `25`? Why?" Answer: `10`. The middle line is a comment, so JavaScript never runs it. Follow up: *"If comments do nothing, why would a professional developer write them?"* Land on: code is read far more often than it is written.

Code Solo (15 min) — "Build a Creature" Story — Students open the Lesson 3.1 exercise in CodeDash: a Mad Libs–style story page with a button. The comments in the starter walk them through **twelve** variables to create and give values to — ten strings and two numbers:

Variable	Data type	Value
`place`	string	any place they want
`creatureName`	string	a name for their creature
`color`, `color2`	string	two **different** colors
`emotion`	string	any emotion
`character`	string	a name for a character

Variable	Data type	Value
`number`, `number2`	number	any number greater than 1
`item`, `item2`	string	two **different** items
`verb`	string	any verb
`adverb`	string	any adverb

The last task is the payoff. At the bottom of the starter is code students have not learned yet — it is already written for them, and their only job is to **replace each `null` with the right variable name**:

```
// 14. Replace the null values with the appropriate variable you created above.
// - Afterwards, run your code and press the button to see your story ✨

document.querySelector("button").addEventListener("click", function() {
    document.querySelector(".place").innerHTML = null;
    document.querySelector(".creature-name").innerHTML = null;
    document.querySelector(".color").innerHTML = null;
    document.querySelector(".color-2").innerHTML = null;
    document.querySelector(".emotion").innerHTML = null;
    document.querySelector(".character").innerHTML = null;
    document.querySelector(".creature-name-again").innerHTML = null;
    document.querySelector(".number").innerHTML = null;
    document.querySelector(".item").innerHTML = null;
    document.querySelector(".character-again").innerHTML = null;
    document.querySelector(".item-2").innerHTML = null;
    document.querySelector(".verb").innerHTML = null;
    document.querySelector(".adverb").innerHTML = null;
    document.querySelector(".number-2").innerHTML = null;
    document.querySelector(".story-container").style = "display:block";
});
```

Set expectations before releasing them: “You are not expected to understand these `document.querySelector` lines yet — that’s Lesson 2. Today you only swap out the `null`s. Everything after `.innerHTML =` is a spot where your stored value gets dropped into the story.”

Two teaching points to make out loud, both of which come straight back to the Do Now:

- **Write the variable name, not a string.** `= place;` drops in the value the student stored; `= "place";` puts the literal word “place” on the page. This is the substitution from `5x` all over again — the name gets replaced by what it holds.
- **`creatureName` and `character` each get used twice** (`.creature-name` and `.creature-name-again`, `.character` and `.character-again`). One variable, many places — store it

once, reuse it everywhere. Call this out; it's the strongest argument for variables the lesson has.

Then they press the button and read their story. Circulate for the five predictable errors: quotes around a variable name, a `null` left behind (that line prints the word "null" — a great visible clue), a second `let` on an existing variable, spaces in a variable name, and a misspelled name in the substitution step (`ReferenceError` in the console). Early finishers: change a value and re-run to see the story rewrite itself, and add a `//` comment above each section of their variables labeling what it's for.

> **Academic Integration (Math — MP6: Precision):** JavaScript syntax demands the same precision as mathematical notation. A missing quote mark, an unclosed parenthesis, or a misspelled variable name will break the program — just as a misplaced decimal point changes a math answer entirely. During circulation, reinforce: "Check every character. Programming rewards precision."

Ask (DOK 3): During circulation, pose to individuals or tables: *"You're writing a program to track a student's grade. What variables would you need, and what data type would each one be? Why?"* Listen for students choosing a number for the grade and a string for the name — and for anyone who catches that a student ID full of digits might still be better stored as a string.

Thought-Provoking Questions (DOK)

- **[DOK 1 — Recall]** What keyword do you use to declare a variable in JavaScript, and which part of `let score = 100;` is the name versus the value? *(Woven into the From Math to Code segment.)*
- **[DOK 2 — Skill/Concept]** What is the difference between `5` and `"5"`? What happens when you add two strings together versus two numbers? *(Woven into the Data Types segment. Requires comparison and prediction.)*
- **[DOK 2 — Skill/Concept]** The computer runs `let x = 100;` and `let playerScore = 100;` identically, and it ignores comments entirely — so why do descriptive names and `//` comments matter at all? Who are they for? *(Woven in twice: the naming half during Declaring a Variable, the comment half during the Comments segment. Requires reasoning about audience and code readability.)*
- **[DOK 3 — Strategic Thinking]** You're building a program to track a student's grade. What variables would you need? What data type would each one be? Why? *(Woven into Code Solo. Requires planning and data modeling.)*
- **[DOK 4 — Extended Thinking]** Variables in math (`x = 2`) and variables in programming (`let x = 2;`) look almost identical. Are they the same concept? What's similar, and what can a programming variable do that an algebra variable cannot? *(Use as the closing prompt. Requires cross-domain comparison and analysis of abstraction.)*

Closure & Assessment

Summary — Rapid review, calling back to the Do Now: “What is a variable — in math *and* in code?” “What are the two data types we learned?” “What’s the difference between `5` and `"5"`?” “What happens if I type `let` twice on the same variable?” “What does `//` do?” Ask 1–2 students to share a variable they created. Close on the DOK 4 prompt above and reinforce: “Variables are the foundation of every program. Everything we build from now on uses them.”

Immediate Application — Students complete the Exit Ticket (Google Forms), 3 items: (1) Write a line of JavaScript that declares a variable named `favoriteMovie` holding the title of your favorite movie, and state its data type. (2) A classmate writes `let score = 10;` then `let score = 25;` and the program crashes — what should the second line say instead, and why? (3) In one sentence: how is `let x = 2;` in JavaScript like `x = 2` in math class?

Extension / Homework — Codecademy practice: *Learn JavaScript* — the **Variables** chapter. Complete the lessons through declaring variables with `let`, assigning values, and logging them; bring your progress screen to the next class. Stretch: back in your “Build a Creature” file, change five of your values and re-run to watch the story rewrite itself, then try two experiments and note what happens in a `//` comment — what does the console say if a variable name starts with a number, and what shows up on the page if you forget the quotes around a string value?