



Learn • Code • Create

Code Today
Create
Tomorrow!



Make
Websites
Interactive!

JAVASCRIPT for KIDS Js



A Fun and Easy Guide to Learn JavaScript
Step-by-Step!



Easy Lessons
for Beginners



Real Examples
& Practice



Fun Projects
to Build



Quizzes &
Challenges



Step-by-Step
Learning



Perfect for
8-15 Years

```
1 // My First JavaScript Program
2 let name = "Coding Star";
3 let age = 10;
4 console.log("Hello, Kids! 🙌");
5 console.log("Welcome to JavaScript!");
6 let message = `I am ${name} and I love coding!`;
7 alert(message);
8 console.log("Let's create something amazing! 🚀");
```

Think Like
a **Coder**,
Have Fun,
Build the
Future!

HTML

CSS

JS



10
Modules



50
Lessons



150+
Topics



10+
Projects



Quizzes
& More!



www.codescompiler.com

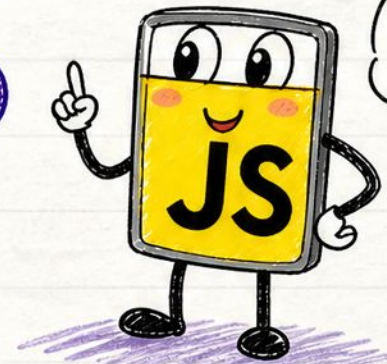


Module 1: Introduction to JavaScript

Lesson 1

What is JavaScript?

JavaScript is a programming language that helps us create **interactive** and **fun** things on the web!



1 What is Programming?



Programming is like giving **instructions** to a computer so it can perform tasks for us.

You write the steps, the computer follows them!

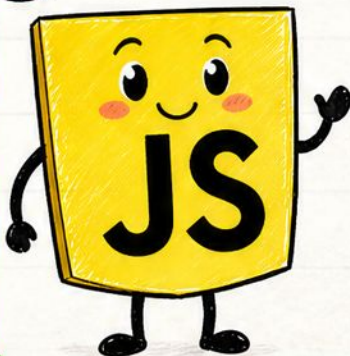
Example:

- ✓ Wake up
- ✓ Brush teeth
- ✓ Eat breakfast
- ✓ Go to school
- ... and more!

Just like a recipe or a set of steps!

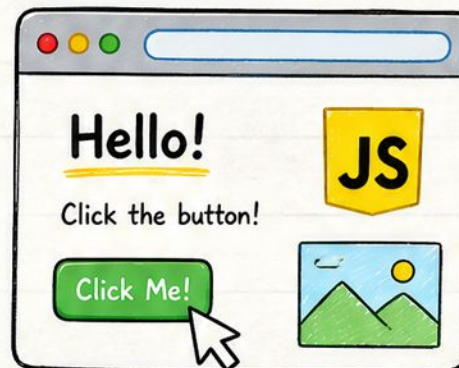


2 What is JavaScript?



JavaScript is a programming language that makes websites **interactive**.

- ✓ It runs in your web browser.
- ✓ It can update content without reloading the page.
- ✓ It can respond to your clicks, key presses and more!



Changes content

Handles events

Makes it interactive!

3 Where JavaScript is Used?

JavaScript is used everywhere on the web!



Websites

Makes pages interactive and dynamic.



Web Games

Powers fun games that run in your browser.



Web Apps

Like online editors, notes apps, and productivity tools.



E-commerce

For smooth shopping experiences and real-time updates.



Mobile Apps (with frameworks)

JavaScript helps build mobile apps too!



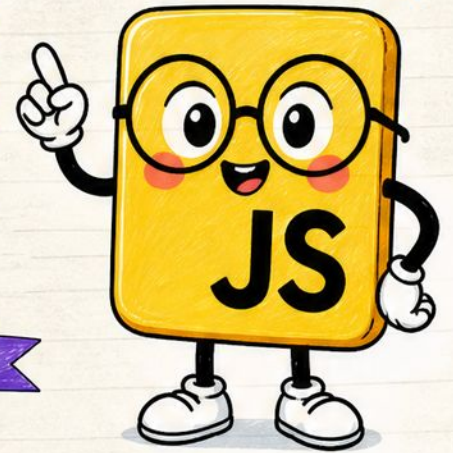
www.codescompiler.com



JavaScript
brings websites
to life! 😊

How JavaScript Works

Small Code,
Big Magic! ✨



Module 1: Introduction to JavaScript

Lesson 2: How JavaScript Works

1 Browsers and JavaScript

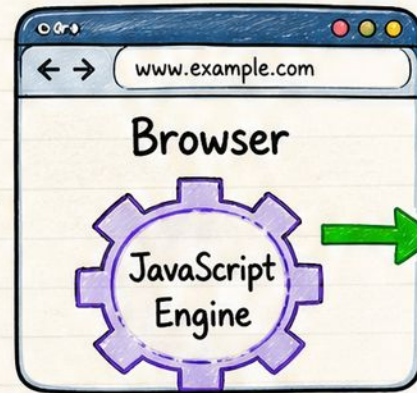
When you open a website, your browser reads the **HTML** and **CSS** to show the page. When it sees **JavaScript**, it reads and runs the code.



Browsers have a special engine called the **JavaScript Engine** that executes the code.

```
<HTML>
<CSS>
<script>
  JS
</script>
```

Website Files



Interactive
Web Page ✨



Browser reads the code
and shows the page!

2 Making Websites Interactive

JavaScript can change HTML and CSS, respond to user actions, and update the page without reloading!



You click,
JavaScript reacts!

User Action



JavaScript
Runs Code



Something
Happens!



What JavaScript Can Do:

- ✓ Change content (text, images, styles)
- ✓ Respond to clicks, key presses, scrolls
- ✓ Show/hide elements
- ✓ Get data and update the page



3 Real-Life Examples

JavaScript is used everywhere!

Image Slider



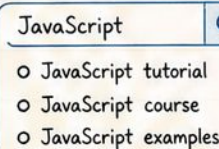
Slides change
without reloading.

Form Validation



Checks your input
before submitting.

Live Search



Shows results as
you type.

Interactive Maps



Zoom, pan and explore
smoothly.

Animations & Games



Makes websites fun
and engaging!



www.codescompiler.com



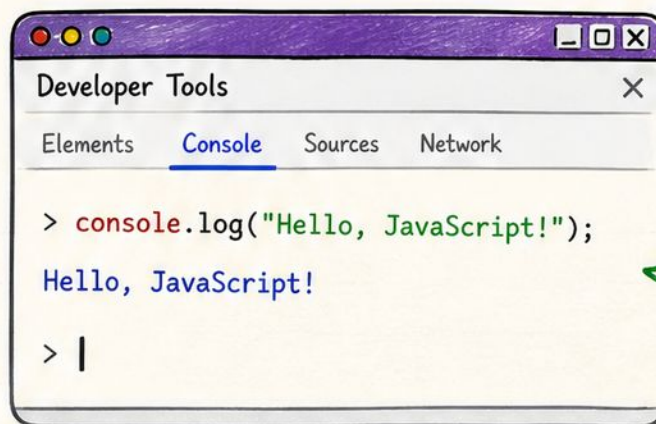


Setting Up JavaScript

Let's get JavaScript ready to run! There are **3** easy ways to start using JavaScript.

1 Using Browser Console

The browser console lets you write and test JavaScript instantly. Open **DevTools** and start coding!



Type your code and see results instantly!

2 Adding JavaScript to HTML

You can add JavaScript to your HTML file using the **<script>** tag.



```
<!DOCTYPE html>
<html>
  <head>
    <title>My First Page</title>
  </head>
  <body>
    <h1>Welcome!</h1>

    <script>
      console.log("Hello from HTML!");
    </script>
  </body>
</html>
```

Tip! ☆

Place **<script>** before **</body>** so your page loads first.

</>

3 Running Your First Script

Save your HTML file and open it in your browser. Then, open the console to see your script in action!



1. Save HTML File



2. Open in Browser

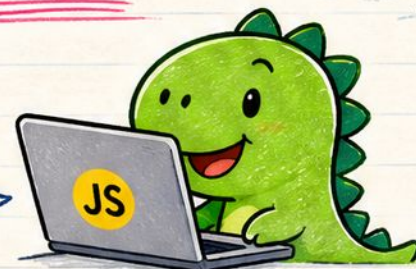


3. Check Console



Great job! You're all set to start your JavaScript journey. Keep practicing and have fun! ♥

Code today, Create tomorrow! ♥



JS

Your First JavaScript Program



JavaScript helps us make web pages interactive.
Let's see how we can **display messages** using JavaScript!

1

The alert() Function

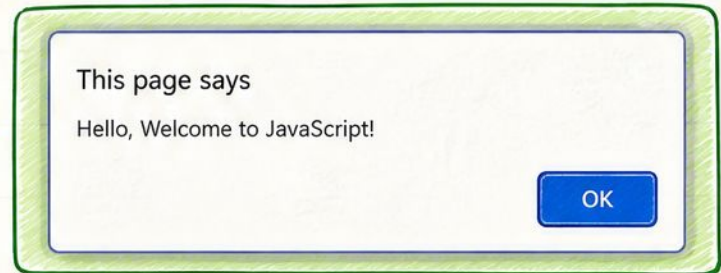
The **alert()** function shows a message in a **pop-up dialog box** on the screen.

Syntax:

```
alert("message");
```

Example

```
alert("Hello, Welcome to JavaScript!");
```



2

The console.log() Function

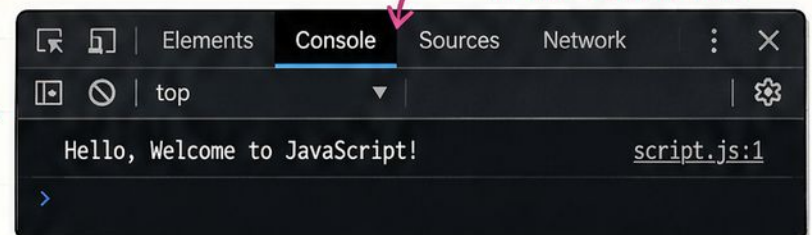
The **console.log()** function displays a message in the browser's **console**.
console window.

Syntax:

```
console.log("message");
```

Example

```
console.log("Hello, Welcome to JavaScript!");
```



You can see this in the Console tab.

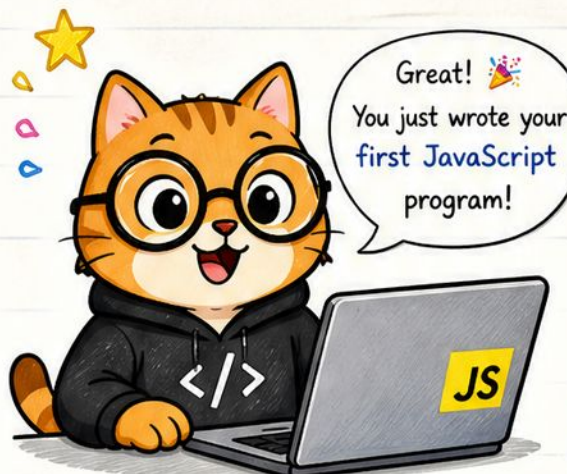
3

Displaying Messages

Using **alert()** or **console.log()**, we can show messages to users or for **debugging** our code.

When to use?

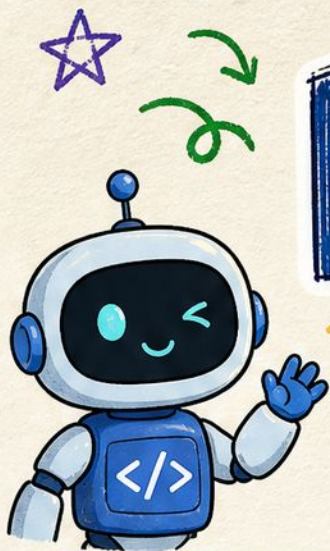
- ✓ **alert()** → For important messages or alerts to users.
- ✓ **console.log()** → For developers to check values and results.



Quick Recap

- ★ **alert()** shows a pop-up message.
- ★ **console.log()** shows message in console.
- ★ Both help us display messages in different ways!





Mini Project



In this mini project, you will use variables and data to create simple and fun programs!

Variables store information. We can use that information to create awesome programs!

1 Welcome Message Program

Create a program that takes the user's name and shows a warm welcome message.

Example:

Enter your name: Riya
Output: Welcome, Riya!
Happy Coding! 😊



2 Favorite Hobby Display

Create a program that asks the user for their favorite hobby and then displays it in a fun way.

Example:

Enter your favorite hobby: Painting
Output: Wow! Painting is a wonderful hobby! 🎨



3 Greeting App

Create a simple greeting app that asks for the user's name and the time of day, then shows an appropriate greeting.

Example:

Name: Arjun
Time: Morning
Output: Good Morning, Arjun!
Have a great day! ☀️



Tip:

Variables help us store and use information in exciting ways!

Think ✨ Code 💻 Create 🚀
That's the coder's way!

Keep Practicing,
Keep Creating! ✨





What are Variables?

Variables help computers remember and use information!



A variable is like a **labeled box** that stores a value. The value can be changed and used in a program.

1

Storing Information

Variables store data that can be used later in our programs.



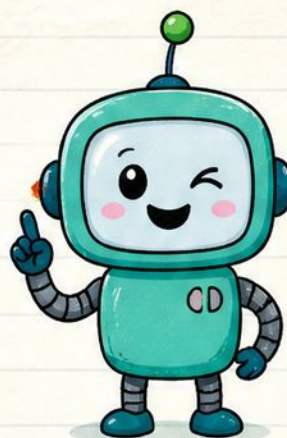
Example:

age stores the number 10.

age



10



Think of a variable as a box with a name and a value inside.

2

Naming Variables

Every variable needs a name so we can use it in our program.

Rules for naming:

- ✓ Start with a letter or underscore (_)
- ✓ Use letters, numbers, or underscores
- ✓ No spaces
- ✓ Use meaningful names

Examples:



score

player1

total_sum



1score

total score

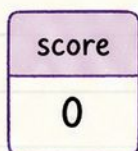
@value

Good names make your code easy to read and understand!

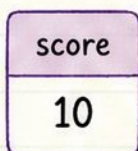
3

Updating Values

The value stored in a variable can change during program execution.



Start



After 1 point



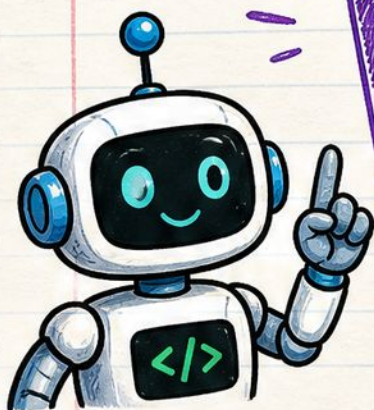
After more points

Example Code:

```
score = 0           # Start with 0
score = score + 10  # Add 10
score = score + 15  # Add 15 more
print(score)        # Output: 25
```

Variables can change, and that's what makes our programs powerful!





DATA TYPES

Data types tell the computer what kind of value a variable holds. Let's explore the three most common data types!



1

STRINGS



They are written inside quotes: single (') or double (").

EXAMPLES:

```
name = "CodeCompiler"  
greeting = 'Hello, coder!'  
language = "Python"
```

Think of strings as words in a sentence! ❤️

EXTRA:

Strings can include letters, numbers, spaces and symbols! ★

2

NUMBERS



Numbers are used for counting, measuring and calculations. They can be whole numbers or decimal numbers.

EXAMPLES:

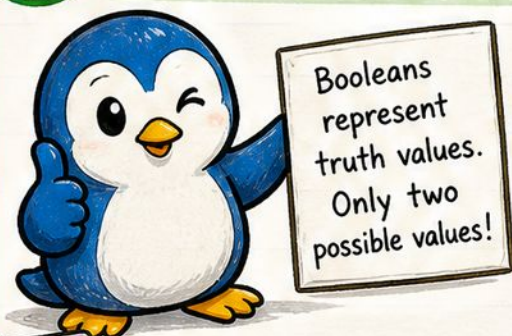
```
age = 21  
pi = 3.14  
temperature = -5
```

Use numbers when you need to do math!



3

BOOLEANS



They can only be: **True** or **False**

EXAMPLES:

```
is_raining = True  
has_homework = False  
is_weekend = True
```

Think of them as YES (True) or NO (False)!

✓ True

✗ False



MODULE 2: VARIABLES AND DATA

LESSON 3

WORKING with TEXT

Text is all around us—let's learn how computers work with it!



In programming, text is called a **STRING**.



1 STRING BASICS

A **string** is a sequence of characters (such as letters, numbers, symbols or spaces) surrounded by quotes.

Examples:

"Hello" "123" "Code is fun!" "😊"

In code:

```
// Using single quotes  
'Hello, world!'  
  
// Using double quotes  
"Hello, world!"
```

Both single (') and double (") quotes work in most languages!



2 COMBINING STRINGS

We can **join** strings together to make new text.

Ways to combine:

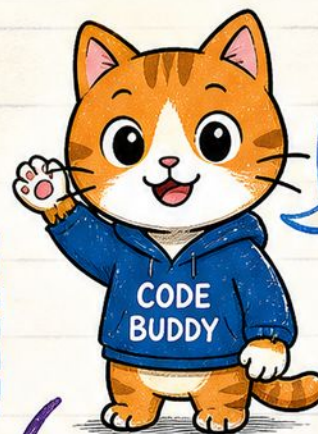
Using the + operator

```
"Hello, " + "World!"  
// Result: "Hello, World!"
```

OR

Using a function

```
"Hello, ".concat("World!")  
// Result: "Hello, World!"
```



Combining strings helps us build sentences, messages, and more!

3 TEMPLATE LITERALS

Template literals (**backticks** ` `) allow us to create strings in a more readable and powerful way.

Use **\${}** to insert variables inside the text!

Example:

```
let name = "Alice";  
let age = 10;  
  
let message = `Hello, ${name}!  
You are ${age} years old.`;  
  
// Result:  
Hello, Alice!  
You are 10 years old.
```

Why it's awesome:

- ★ Easier to read
- ★ Supports variables
- ★ Can include line breaks
- ★ Perfect for messages and multi-line text!

COOL!

Practice makes perfect—keep coding and explore!



Strings help us communicate with users, show information, and make our programs friendly and fun!

</>

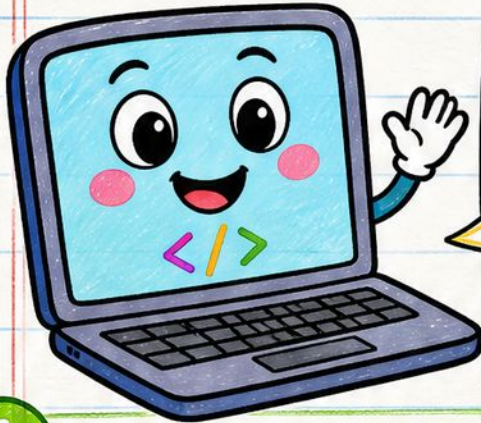
www.codescompiler.com

</>

Module 2: Variables and Data

Lesson 4: User Input

User Input



User input lets our programs ask questions and get information from the user. Then we can use that information to do something awesome!

Programs are more fun when they can talk to **YOU!**

1

Using prompt()

We use `prompt()` to ask the user for input.

```
let name = prompt("What is your name?");
```

The `prompt()` shows a box and waits for the user to type something.

What is your name?

OK

Cancel

2

Collecting Information

The user types their answer, and we **store** it in a **variable** to use later in our program.



How old are you?

12

OK

Cancel

```
let age = prompt("How old are you?");  
// user typed: 12  
// age now stores "12"
```

3

Displaying Results

We use the input to **show something** back to the user.

```
alert("Hello, " + name +  
! Nice to meet you!");
```

Message from webpage



Hello, Alex!
Nice to meet you!

OK

In Short



Ask with `prompt()`



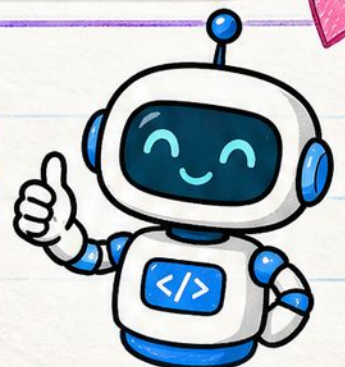
User types an answer



Store it in a **variable**



Show results to the user



Module 3: Math and Operators

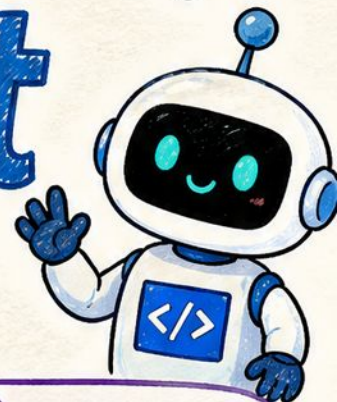
Lesson 5: Mini Project

Let's Code!



Mini Project

In this mini project, you will build 3 fun programs using math, operators and your creativity!



1

My Profile Card

Create a program that shows a profile card about you!

- ★ Your Name
- ★ Your Age
- ★ Your Hobby
- ★ A short bio about you



≡ MY PROFILE ≡

👤 Name : Alex
📅 Age : 11
🎮 Hobby : Coding
❤️ Bio : I love coding and building cool projects!

Use variables and text to make it personal!

2

Favorite Food Program

Create a program that helps you choose and explore your favorite food!

- ★ Store your favorite food name
- ★ Give it a yummy score (1 to 10)
- ★ Double the score using math operators
- ★ Show a message about how delicious it is!

```
food = "Pizza"
score = 8
double_score = score * 2

print("My favorite food is", food)
print("Yummy Score:", score)
print("Double Score:", double_score)

# Pizza is super delicious! 🍕
```

Use math operators like * to make it awesome!

3

Fun Fact Generator

Create a program that generates a fun fact using numbers!

- ★ Use numbers and operators
- ★ Do some cool calculation
- ★ Show a fun fact as output

≡ FUN FACT ≡

Did you know?

There are 24 hours in a day!

That's 1440 minutes!

And 86400 seconds!

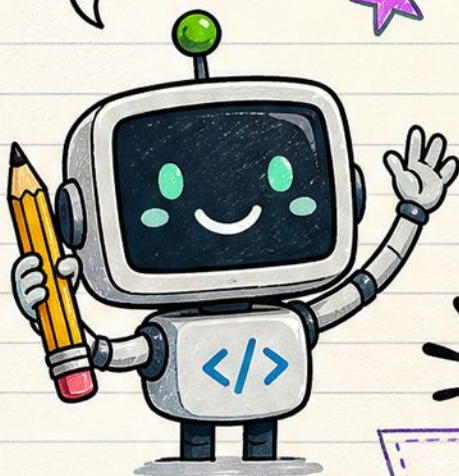
Time is amazing! ⌚ ✨

Use math operators to discover amazing facts!

www.codescompiler.com



Math is everywhere!
Let's learn the **basics**!



MODULE 3: MATH AND OPERATORS

LESSON 1:

Basic Math

What is Math?

Math helps us solve problems, count things, and make smart decisions every day!

In programming and in real life, we use math to solve problems. Let's explore **three basic math operations**!



1 ADDITION

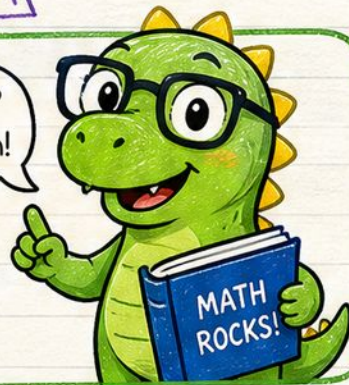
Addition means combining or adding numbers together to get a **total**.



$$3 + 2 = 5$$

Three plus two equals **five**.

We use '+' for addition!



2 SUBTRACTION

Subtraction means taking one number away from another to find the **difference**.



$$5 - 2 = 3$$

Five minus two equals **three**.

We use '-' for subtraction!



3 MULTIPLICATION

Multiplication means adding a number to itself a certain number of **times**.



$$2 \times 3 = 6$$

Two times three equals **six**.

We use 'x' for multiplication!



Why it Matters?

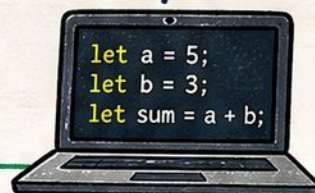
- ✓ Helps us solve real-life problems
- ✓ Useful in coding, games, and apps
- ✓ Builds strong thinking skills

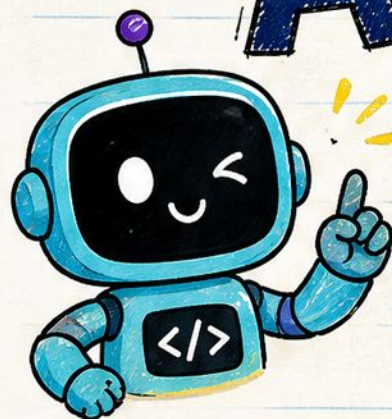
Try It!

- 1) $4 + 3 = \underline{\quad}$
- 2) $7 - 2 = \underline{\quad}$
- 3) $3 \times 4 = \underline{\quad}$

</>

```
let a = 5;  
let b = 3;  
let sum = a + b;
```





Advanced Math

Module 3:
Math and
Operators

Lesson 2:
Advanced Math

In advanced math, we use **operators** to solve problems efficiently in programming



1 DIVISION

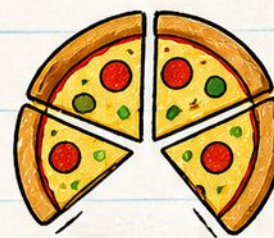
Division (/) is used to divide one number by another and get the quotient.

Think: How many times does it go?

EXAMPLE:

$$20 / 4 = 5$$

Dividend Divisor Quotient



20 pizzas shared equally among 4 friends = 5 each!

2 MODULUS

Modulus (%) gives the remainder after division. It tells what is left over.

Think: What's left after dividing?

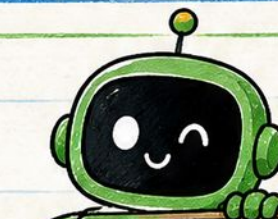
EXAMPLE:

$$20 \% 4 = 0$$

20 divided by 4 leaves no remainder.

$$22 \% 4 = 2$$

22 ÷ 4 leaves remainder 2.



Useful in:

- ✓ Checking odd/even
- ✓ Wrapping around
- ✓ Patterns & cycles
- ✓ Indexing

3 EXPONENTS

Exponents (^) show how many times a number (base) is multiplied by itself.

Think: Power up your number!

EXAMPLE:

$$2 ^ 3 = 8$$

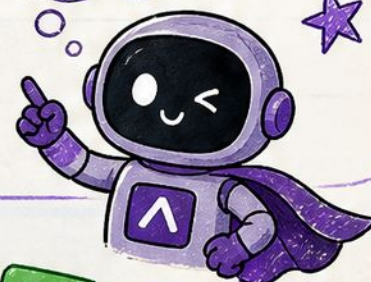
Base Exponent Result

EXPANDED FORM:

$$2 ^ 3 = 2 \times 2 \times 2 = 8$$

Used in:

- ✓ Fast calculations
- ✓ Scientific notation
- ✓ Game development
- ✓ Algorithms



Practice makes perfect!



Master these operators, and you'll level up your coding superpowers!



Let's make
coding easier
and smarter!



ASSIGNMENT OPERATORS



Assignment operators are used to store values in variables and update them.

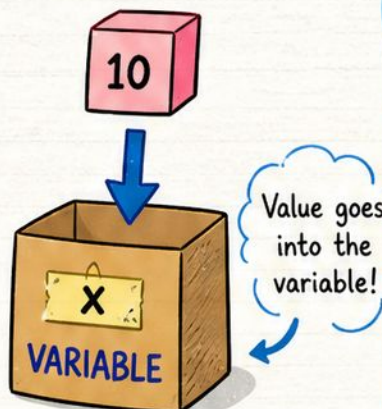
1

Assigning Values

We use the **assignment operator (=)** to store a value in a variable.

Example:

```
x = 10           // x stores 10
name = "CodeKid" // name stores "CodeKid"
pi = 3.14        // pi stores 3.14
```



Common Assignment Shortcut Operators

Operator	Meaning (Same as)
=	$x = y$
+=	$x = x + y$
-=	$x = x - y$
*=	$x = x * y$
/=	$x = x / y$
%=	$x = x \% y$

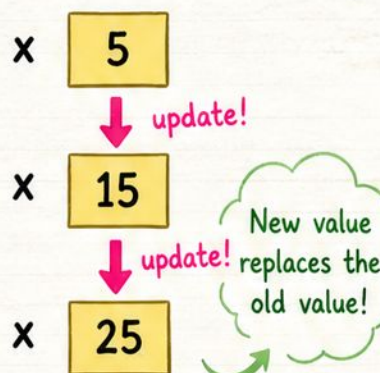
2

Updating Variables

We can **change the value** of a variable by assigning a new value to it.

Example:

```
x = 5           // x is 5
x = 15          // now x is 15
x = x + 10      // now x is 25
```



3

Shortcut Operators

Shortcut operators let us update variables in a **shorter way**.

Example:

```
x = x + 5
x -= 5
x *= 2
x /= 2
x %= 3
```

Same as

```
x += 5
x -= 5
x *= 2
x /= 2
x %= 3
```



Less typing,
more coding!

Quick Note

- ✓ = is the basic assignment operator.
- ✓ It stores values in variables.
- ✓ Shortcut operators save time and make code cleaner!



</>

CODE
FUN!1010
0101

#



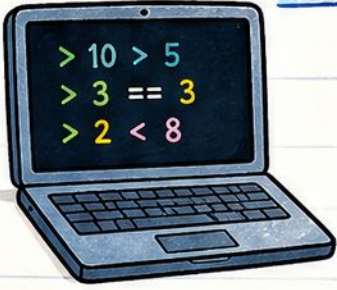


Module 3: Math and Operators

Lesson 4



Operators help us compare values!



Comparison Operators

Comparison operators are used to compare two values. They return either **True** or **False**.



1 Equal To

Checks if two values are the **same**.

==

(double equal)

EXAMPLE:

5 == 5 → **True**

7 == 3 → **False**

Same values!



2 Greater Than

Checks if the value on the left is **greater than** the value on the right.

>

(greater than)

EXAMPLE:

9 > 4 → **True**

2 > 7 → **False**

Bigger value!



3 Less Than

Checks if the value on the left is **less than** the value on the right.

<

(less than)

EXAMPLE:

3 < 6 → **True**

8 < 2 → **False**

Smaller value!



QUICK TIP

Comparison operators always return **True** or **False**. You can use them in conditions and decisions in code!

Where are they used?

- ✓ In if statements
- ✓ In loops
- ✓ In making decisions in programs



Mini Project

Use what you've learned to build these fun mini projects!

1 Calculator Program

Build a simple calculator that can add, subtract, multiply and divide two numbers.

Take two numbers and an operator as input and show the result!



```
# Simple Calculator
a = float(input("Enter first number: "))
b = float(input("Enter second number: "))
op = input("Choose operation (+, -, *, /): ")
if op == '+':
    print(a + b)
elif op == '-':
    print(a - b)
elif op == '*':
    print(a * b)
elif op == '/':
    print(a / b if b != 0 else "Cannot divide by zero")
else:
    print("Invalid operation")
```

CALCULATOR

First Number

12

Second Number

8

Operation

*

Result

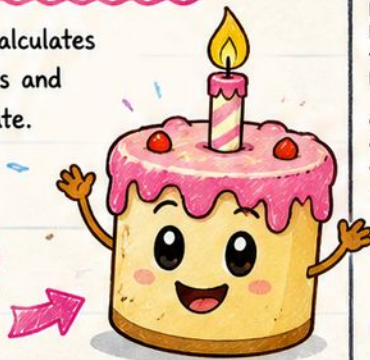
96

Try all 4 operations!

2 Age Calculator

Create a program that calculates your age in years, months and days from your birth date.

Input your birth date and today's date to find your exact age!



```
from datetime import date

birth = input("Enter your birth date (YYYY-MM-DD): ")
b_year, b_month, b_day = map(int, birth.split('-'))
today = date.today()
b_date = date(b_year, b_month, b_day)

age_years = today.year - b_date.year
age_months = today.month - b_date.month
age_days = today.day - b_date.day

if age_days < 0:
    age_months -= 1
if age_months < 0:
    age_years -= 1
    age_months += 12

print(f"Your age is {age_years} years, {age_months} months, and {age_days} days.")
```

AGE CALCULATOR

Birth Date

2010-05-15

Today's Date

2024-05-20

Your Age

14 Years

0 Months

5 Days

3 Score Tracker

Make a program that keeps track of scores, shows the total and the highest score.

Add scores, view total and find the highest score achieved!



```
scores = []
while True:
    choice = input("\n1. Add Score\n2. View Scores\n3. Exit\nChoose: ")
    if choice == '1':
        score = int(input("Enter score: "))
        scores.append(score)
        print("Score added!")
    elif choice == '2':
        if scores:
            total = sum(scores)
            highest = max(scores)
            print("Scores:", scores)
            print("Total:", total)
            print("Highest Score:", highest)
        else:
            print("No scores yet.")
    elif choice == '3':
        print("Good luck and keep scoring!")
        break
    else:
        print("Invalid choice. Try again.")
```

SCORE TRACKER

Scores: [10, 25, 15, 30]

Total Score

80

Highest Score

30

Great job! You're building awesome programs!



Introduction to Conditions

In coding, computers make decisions using **CONDITIONS**!



Conditions help programs choose the right action based on a situation.



1

True and False

Conditions are statements that can be either **TRUE** or **FALSE**.

Think of it like a yes or no question!



Examples:

Is it raining?



→ **TRUE** or **FALSE**

Is 5 greater than 3?

$5 > 3$

→ **TRUE** or **FALSE**

Is the light on?



→ **TRUE** or **FALSE**

Every condition has only two possible answers!

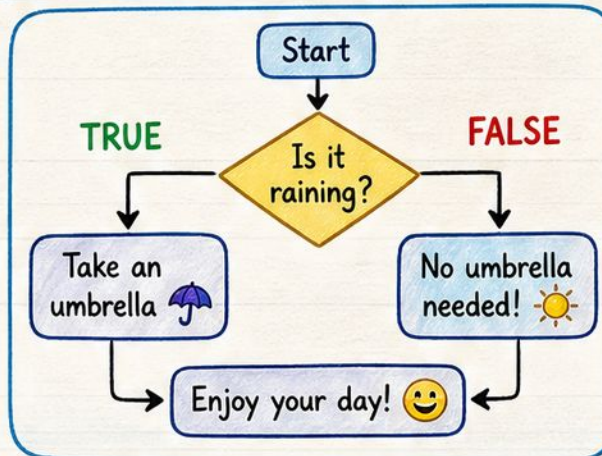


2

Decision Making

Conditions help programs make decisions.

The program chooses a path based on whether a condition is **TRUE** or **FALSE**.



Just like in real life!

We make decisions every day based on different situations.



3

Why Conditions Matter

Conditions make programs smart and interactive. They allow programs to:

- ✓ React to different situations
- ✓ Give the right output
- ✓ Help in games, apps, websites and more!

Games



If score > 100, you level up!

Apps



If weather is sunny, show sunglasses suggestion.

Websites



If user is logged in, show dashboard.

Smart Devices



If heart rate > 100, show alert!

Conditions make code powerful!

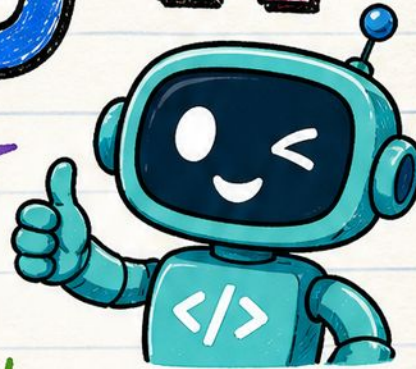




Using if



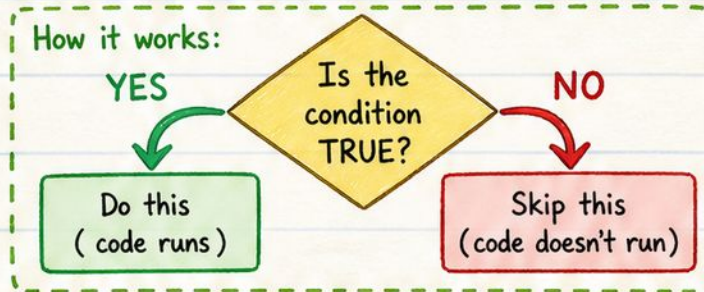
The **if** statement helps our program make smart decisions!



I check conditions and help your program decide!

1 Checking Conditions

The **if** statement checks if a condition is **TRUE** or **FALSE**.



EXAMPLE:

Check if a number is greater than 10.

```
if number > 10:
    print("Number is greater than 10")
```



2 Simple Decisions

If the condition is **TRUE**, the program does something. If **FALSE**, it skips it.



Think of it like:
If it's raining,
I take an umbrella.
(If not, I don't!)



EXAMPLE:

Check if it is raining.

```
is_raining = True
if is_raining:
    print("Take an umbrella!")
```



3 User Choices

We can use **if** to respond to choices made by the user.



Example: Choose your favorite color and we'll give a message!

EXAMPLE:

User enters a color.

```
color = input("Enter your favorite color: ")
if color == "blue":
    print("Great choice! Blue is awesome!")
```



KEY TAKEAWAY

- ✓ **if** checks a condition.
- ✓ It helps the program make simple decisions.
- ✓ It can use user choices to do smart things!

PRO TIP

Indentation is important in Python! The code inside **if** runs only when the condition is **TRUE**.



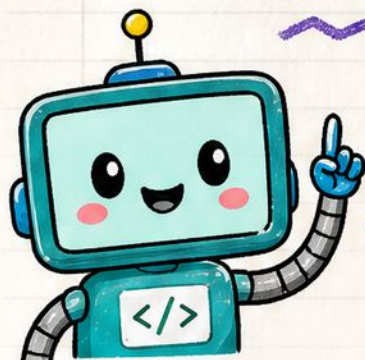


Module 4: Making Decisions

Lesson 3:



Using if...else

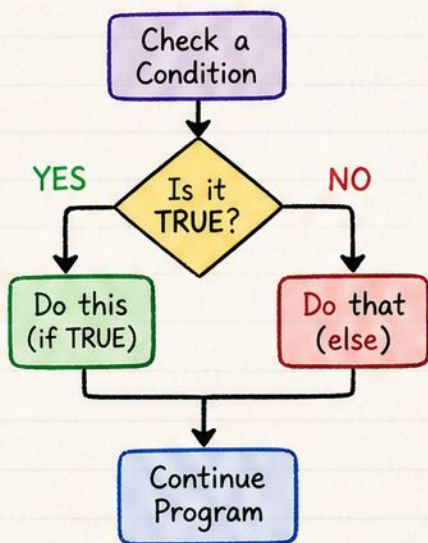


The **if...else** statement helps our programs make a choice between **TWO** possibilities. Only one of the outcomes will happen.

IF a condition is **TRUE**, do this...
ELSE, do that!

1 Two Possible Outcomes

With **if...else**, there are only **TWO** paths your program can take.



★ Only one path is chosen based on the condition.

2 Positive and Negative Results

We can use **if...else** to handle good results (**positive**) or bad results (**negative**).



Positive Result

Example:

Score ≥ 50

→ You Win!



Negative Result

Example:

Score < 50

→ Try Again!



if...else helps programs respond in different ways to different situations.

3 Game Decisions

if...else is used a lot in games to decide what happens next.

Example: Treasure Game



Is there a key?



YES

You unlock the chest!



NO

The chest stays locked.



Games use **if...else** to make decisions fun and interactive!

How it looks in code:

```
1 if score >= 50:
2     print("You Win!")
3 else:
4     print("Try Again!")
5
6 print("Thanks for playing!")
```

Remember:

if...else =

Make a choice,
One path at a time!



★ Key Takeaway ★

- ✓ **if...else** gives two possible outcomes.
- ✓ It handles positive and negative results.
- ✓ It helps make games and programs smarter!





Using else if



Think
Decide
Code! ♥

Use **else if** to check multiple conditions one by one.
Only the first true condition runs.

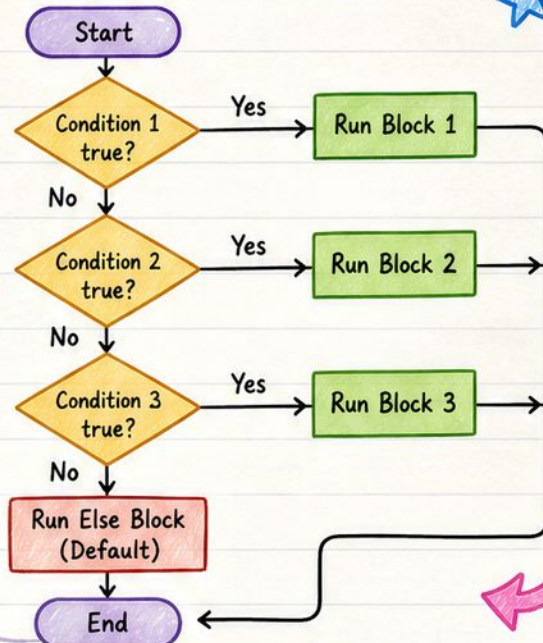
WHAT IS else if?

When you need to choose between many options, use **else if** to test another condition if the first one is **false**.

SYNTAX

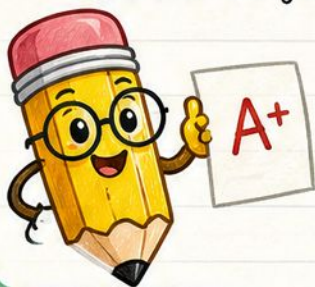
```
if (condition1) {
    // code block
} else if (condition2) {
    // code block
} else if (condition3) {
    // code block
} else {
    // code block
}
```

HOW IT WORKS



1 GRADE CHECKER

Check a student's marks and show the grade.



EXAMPLE CODE (Python)

```
marks = int(input("Enter marks: "))
if marks >= 90:
    grade = "A"
elif marks >= 75:
    grade = "B"
elif marks >= 60:
    grade = "C"
elif marks >= 40:
    grade = "D"
else:
    grade = "F"
print("Grade:", grade)
```

EXAMPLE OUTPUT

90 - 100	→	Grade: A
75 - 89	→	Grade: B
60 - 74	→	Grade: C
40 - 59	→	Grade: D
0 - 39	→	Grade: F



2 WEATHER ADVISOR

Give outfit advice based on the temperature.



EXAMPLE CODE (Python)

```
temp = float(input("Enter temperature (°C): "))
if temp >= 30:
    advice = "Wear light clothes and stay hydrated."
elif temp >= 20:
    advice = "Wear comfortable clothes."
elif temp >= 10:
    advice = "Wear a jacket."
elif temp >= 0:
    advice = "Wear warm clothes."
else:
    advice = "It's freezing! Wear heavy winter clothes."
print("Advice:", advice)
```

EXAMPLE OUTPUT

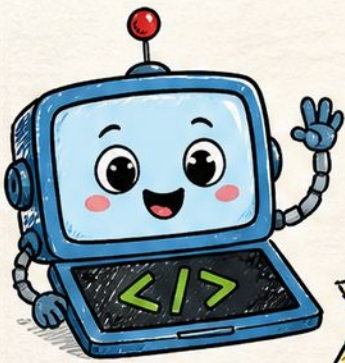
temp >= 30°C		Wear light clothes and stay hydrated.
20°C - 29°C		Wear comfortable clothes.
10°C - 19°C		Wear a jacket.
0°C - 9°C		Wear warm clothes.
temp < 0°C		It's freezing! Wear heavy winter clothes.

Remember: **else if** helps your program think step by step! ♥

www.codescompiler.com

More conditions.
Smarter decisions.
Better programs! ★





Mini Project

Module 5: Loops and Repetition - Lesson 5: Mini Project



1

Smart Quiz

Create a quiz that asks multiple questions, checks answers and shows the score at the end.



How it works:

- 1 Ask a question.
- 2 Take user answer.
- 3 Check if it's correct.
- 4 Update the score.
- 5 Repeat for all questions.
- 6 Show final score.



Q. What is the capital of India?

A. Mumbai

B. Delhi ✓

C. Kolkata

D. Chennai

Great! 🎉
Score: 3 / 5

Use loops to solve real-life problems with your code! 😊

2

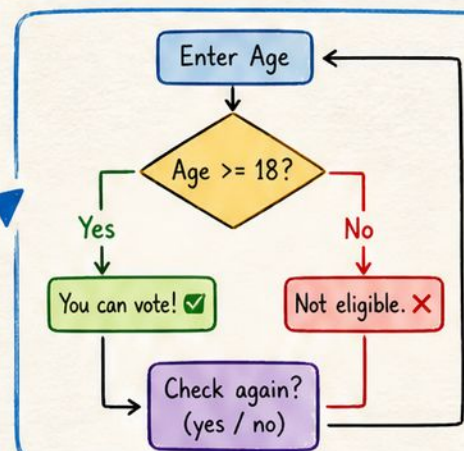
Voting Age Checker

Ask the user for their age and check if they are eligible to vote. Keep checking until the user wants to stop.



How it works:

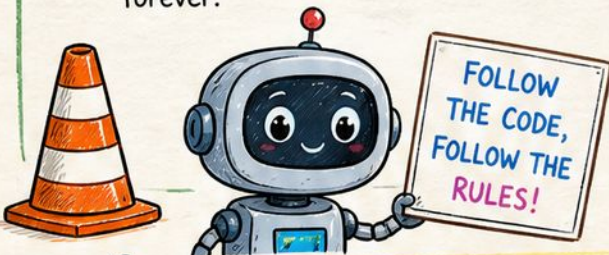
- 1 Take age as input.
- 2 If age ≥ 18 , show "You can vote!"
- 3 Else, show "Not eligible."
- 4 Ask if user wants to check again.
- 5 Repeat until user says "no".



3

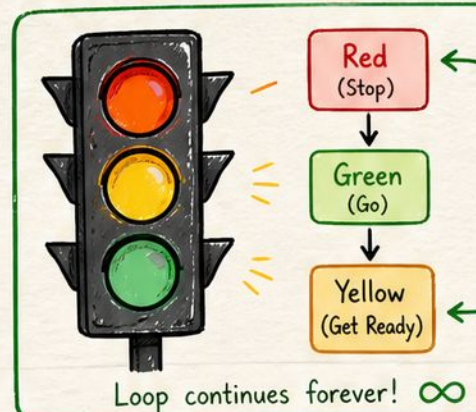
Traffic Light Simulator

Simulate a traffic light that changes colors in order: Red → Green → Yellow and repeats forever.



How it works:

- 1 Show Red light.
- 2 Wait for some time.
- 3 Show Green light.
- 4 Wait for some time.
- 5 Show Yellow light.
- 6 Wait for some time.
- 7 Repeat forever. ∞



TIPS

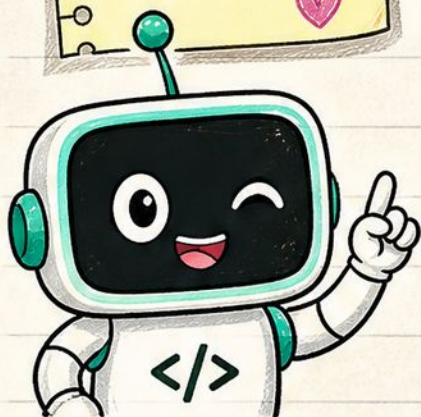
- ★ Use loops to repeat tasks easily.
- ★ Take input, process it and give output.
- ★ Test your project with different values.



Think,
Code, Create,
Repeat!



Loops help us repeat actions easily and make our code smarter!



Why Use Loops?



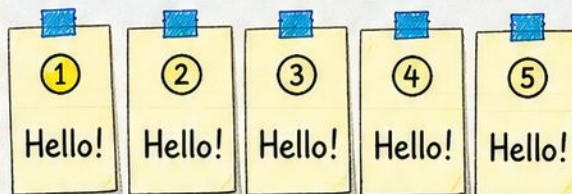
```
for i in range(n):  
    do something  
    again and again
```

Loops let us repeat a block of code multiple times with ease.

1 Repeating Tasks

Loops help us **repeat** the same task **over and over** without writing the code again and again.

Print "Hello!" 5 times



Instead of writing it 5 times...

...we use a loop!

```
for i in range(5):  
    print("Hello!")
```

2 Saving Time

Loops **save time** and **reduce mistakes**. Change it once, and it works everywhere!

Without Loops

More typing, more time, more chances of mistakes!



```
print("A")  
print("A")  
print("A")  
print("A")  
print("A")  
...
```



With Loops

Less code, more time to be creative!



```
for i in range(5):  
    print("A")
```

3 Loop Examples

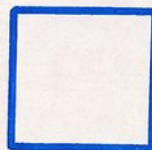
Loops are used in many real-life situations and programs!

Counting

1 2 3 ...
4 5

Count from 1 to 10

Drawing Shapes



Draw a square 4 times

Games



Run, jump or move repeatedly

Data Processing



Check each item in a list

Think Like a Coder!

Whenever you see a task that repeats, think... "Can I use a loop?"

Loops make your code **shorter, smarter** and **super powerful!**



Loops help us repeat things easily!

Module 5: Loops and Repetition

Lesson 2:

For Loops

A **for loop** repeats a block of code a certain number of times.

Syntax

```
for (initialization; condition; update) {  
    // code to repeat  
}
```

1 Counting Numbers

Use a for loop to count numbers in order.



```
for i in range(1, 6):  
    print(i)
```

Output

1
2
3
4
5

2 Repeating Messages

Use a for loop to repeat a message multiple times.



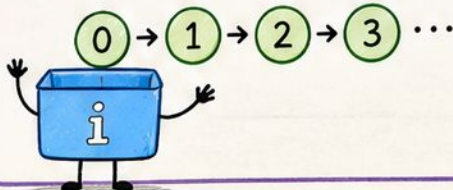
```
for i in range(3):  
    print("Hello, Code Explorer!")
```

Output

Hello, Code Explorer!
Hello, Code Explorer!
Hello, Code Explorer!

3 Loop Variables

The loop variable changes value each time the loop runs.



```
for i in range(4):  
    print("Value of i is:", i)
```

Output

Value of i is: 0
Value of i is: 1
Value of i is: 2
Value of i is: 3

How it works (in simple steps):

- 1 Initialize - Set the loop variable.
- 2 Check Condition - If true, run the code.
- 3 Update - Change the variable and repeat.

Think of it like...

A **for loop** is like a repeating machine that stops when the job is done!



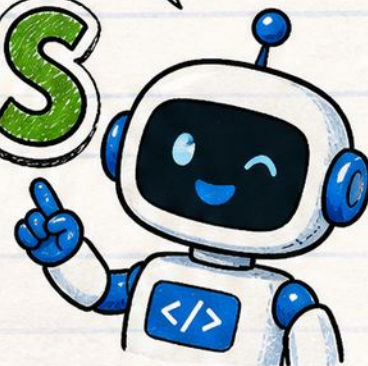
MODULE 5: LOOPS AND REPETITION

LESSON 3:

Loops help us repeat things! ❤️

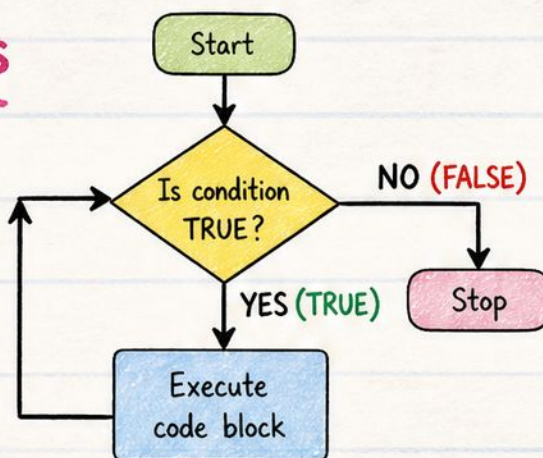
While Loops

A **while** loop repeats a block of code as long as a condition is **TRUE**.



1 Loop Conditions

A while loop runs only while the condition is **TRUE**. When the condition becomes **FALSE**, the loop stops.



EXAMPLE

```
count = 1
while count <= 5:
    print(count)
    count = count + 1
```

OUTPUT:

1
2
3
4
5



2 Repetition Until True

A while loop keeps repeating the code until the condition becomes **FALSE**.

"It's like saying: 'Keep doing this until it's no longer true.'"



The loop stops when the condition is no longer true!

EXAMPLE

```
num = 10
while num > 0:
    print(num)
    num = num - 1
```

OUTPUT:

10
9
8
...
2
1

The loop repeats while num is greater than 0. It stops when num becomes 0.

3 Infinite Loops

If the condition never becomes **FALSE**, the loop will run forever! This is called an **Infinite Loop**.



Be careful! Infinite loops can crash your program.

DANGER!



Never-ending loop!

EXAMPLE

```
while True:
    print("This runs forever!")
```



The condition is always **TRUE**, so the loop never stops!

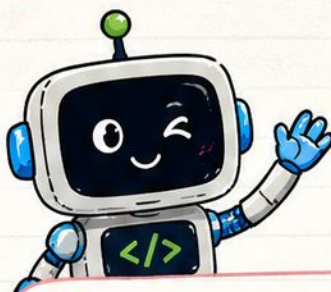
Practice loops. Think logically. Code like a pro! ❤️

www.codescompiler.com



Module 5

Loops and Repetition



Loop Controls



Loop controls help us change the normal flow of a loop. They make our programs smarter!

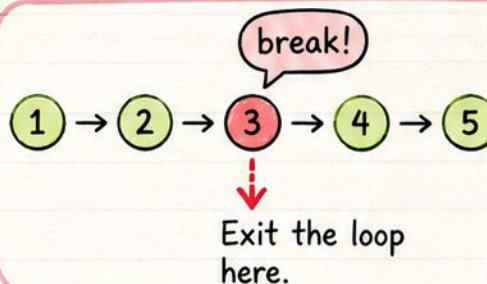
1 break

The **break** statement is used to exit the loop immediately, even if the loop condition is still True.

```
for i in range(1, 6):  
    if i == 3:  
        break  
    print(i)
```

Output:

1
2



break is like an emergency exit!



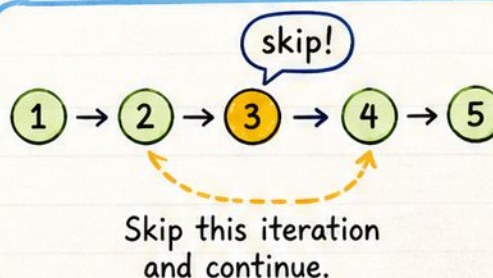
2 continue

The **continue** statement is used to skip the current iteration and move to the next one.

```
for i in range(1, 6):  
    if i == 3:  
        continue  
    print(i)
```

Output:

1
2
4
5



continue says, "Not this time!"



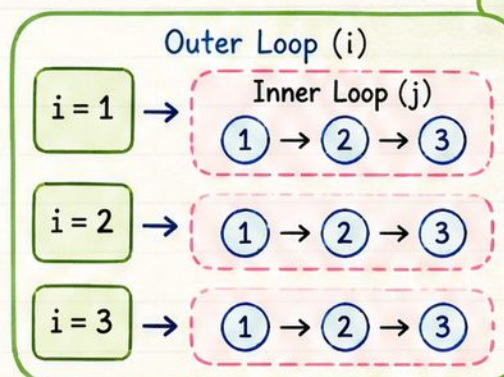
3 Nested Loops

A loop inside another loop. The inner loop runs completely for each iteration of the outer loop.

```
for i in range(1, 4):  
    for j in range(1, 4):  
        print(f"i={i}, j={j}")
```

Output:

i=1, j=1
i=1, j=2
i=1, j=3
i=2, j=1
i=2, j=2
i=2, j=3
i=3, j=1
i=3, j=2
i=3, j=3



Great for working with tables, patterns, and more!



Practice makes perfect! Keep coding!



Mini Project



In this mini project, we will create **three useful programs** using **functions**!



1 Number Counter

A program that counts numbers from 1 up to a limit you choose.

What it does:

- ✓ Takes a starting number and an ending limit.
- ✓ Displays numbers in order.

Example:

Start: 1
End: 10

1, 2, 3, 4, 5,
6, 7, 8, 9, 10

Great for counting up!

1 2 3

2 Countdown Timer

A program that counts down from a number to zero.

What it does:

- ✓ Takes a starting number.
- ✓ Counts down to 0.
- ✓ Shows time step by step.

Example:

Start: 5

5 ↓
4 ↓
3 ↓
2 ↓
1 ↓
0 ↓

Time's up! 🎉



Perfect for timers and games!

3 Multiplication Table Generator

A program that prints the multiplication table of any number.

What it does:

- ✓ Takes a number as input.
- ✓ Generates the multiplication table (up to 10).
- ✓ Helps in quick math practice.

Example:

Table of 7

7 × 1 = 7
7 × 2 = 14
7 × 3 = 21
7 × 4 = 28
7 × 5 = 35
7 × 6 = 42
7 × 7 = 49
7 × 8 = 56
7 × 9 = 63
7 × 10 = 70

Practice makes you perfect!



Why use Functions?

- ✓ Reuse code
- ✓ Organize better
- ✓ Make programs smarter!



You Did It!

You created three awesome programs using functions!

Keep Coding! 💖

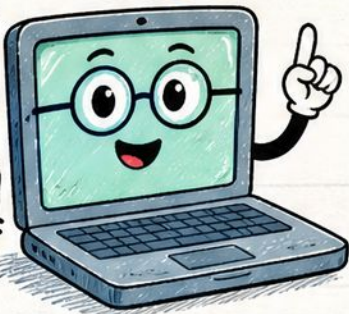
Skills You Practiced

- ★ Using functions
- ★ Taking input
- ★ Loops and logic
- ★ Displaying output



A function is a !
block of code that
performs a **specific**
task.

Let's break it
down!



What are Functions?



1 Reusable Code

Functions let us **write code once** and use it again and again.

```
print("Hello")
print("Hello")
print("Hello")
print("Hello")
```

Without Function
(repeated code)

```
def say_hello():
    print("Hello")
```

```
say_hello()
say_hello()
say_hello()
say_hello()
```

Write once,
use many
times!

2 Why Functions Matter



Avoids Repetition

No need to write the **same** code again and again.



Improves Organization

Breaks big programs into small, **easy-to-manage** pieces.



Easier Maintenance

Change the code in **one** place, and it updates everywhere.



Makes Code Smarter

Helps build **clean, readable,** and **professional** programs.

3 Function Examples

Functions can do all kinds of tasks!

1. Greet Someone

```
def greet(name):
    print("Hello, " + name + "!")
```

```
greet("Alice")
```

Hello, Alice!



2. Add Numbers

```
def add(a, b):
    return a + b
```

```
result = add(5, 3)
```

```
print(result) # 8
```



3. Draw a Shape

```
def draw_square(size):
    for i in range(4):
        print("*" * size)
```

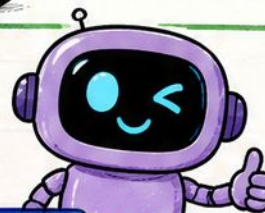
```
draw_square(4)
```

```
***
*
*
*
***
```



QUICK TIP

Think of functions like your **superpower tools**. They help you do more with less effort!



Use functions
and make your
code **awesome!**

Break it down.
Write it once.
Use it anywhere!

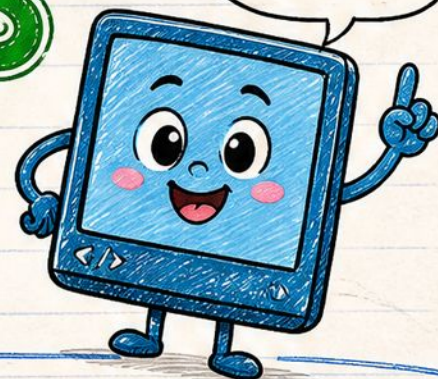
Creating Functions



Functions make your code smarter and easier to use!



A function is a block of code that performs a specific task. You can use it again and again!



1 FUNCTION SYNTAX

This is the basic structure of a function.

```
def function_name(parameters):
    # code block
    return value
```

def: keyword to define a function

parameters (optional): input values

the code to be executed

return: (optional) sends a result back

Example:

```
def add(a, b):
    result = a + b
    return result
```



Think of a function as a mini machine that does a job!

2 CALLING FUNCTIONS

To run a function, call it by its name and pass the required arguments.

```
# function call
result = add(5, 3)
```

- ✓ The function 'add' is called.
- ➔ Values 5 and 3 are passed as arguments.
- ★ The function runs and returns the result.

Output

result
8

3 NAMING FUNCTIONS

Good names make your code easy to read and understand.



Use names that describe what the function does!

✓ GOOD NAMES	✗ AVOID NAMES
- calculate_area - get_user_age - is_prime - print_message	- a - fun1 - data - doStuff

TIPS

- ★ Use letters, numbers, and underscores (_).
- ★ Start with a letter or underscore.
- ★ Be descriptive and meaningful.



Let's make
our functions
smarter!



Function Parameters

Parameters help functions
receive data to do their job!

What are
Parameters?

They are variables
listed in a function
that act as
placeholders for
the data we pass in.

1

Passing Data

We pass data (also called arguments) to a function. The parameters receive that data inside the function.



I bring the data
so the function
can use it!

EXAMPLE

```
def greet(name):
    print(f"Hello, {name}!")

greet("Alice")
```

name is a
parameter

Alice is passed in

Output:
Hello, Alice!

2

Multiple Parameters

A function can have more than one parameter. We pass multiple values, in order.



More data =
more power!

EXAMPLE

```
def add(a, b):
    return a + b

result = add(7, 5)
print(result)
```

a and b are
parameters

7 goes to a, 5 goes to b

Output: 12

Output: 12

Order
Matters!

The first value
goes to the
first parameter.

3

User Input

We can use user input as arguments and pass it to our functions!



You type it,
we use it!

EXAMPLE

```
def square(num):
    return num * num

user_num = int(input("Enter a number: "))
answer = square(user_num)
print(f"Square is {answer}")
```

num is a
parameter

The user
provides the
data. We pass
it to the
function!



QUICK TIP

Parameters are like containers.
You pass in values, and the function
uses them to do its task!



Remember

- ✓ Define parameters in the function.
- ✓ Pass arguments when calling it.
- ✓ Use the data inside the function.

Returning Values

Functions can do more than just work—they can **GIVE** something back!



A function can **return** a value back to the place where it was called.

1

Using return

The **return** statement sends a value back from the function to the caller.

★ It's like handing a result back to your friend!

EXAMPLE

```
def add(a, b):
    result = a + b
    return result

# The function returns the sum
```

return sends the value back!



2

Saving Results

You can save the returned value in a **variable** and use it later.

★ Store it, use it, or combine it!



EXAMPLE

```
def square(n):
    return n * n

# Save the returned value
num = 5
answer = square(num)
print(answer) # Output: 25
```

We saved the result in 'answer' and used it!



3

Reusing Functions

Return values let you **reuse** functions many times in different places.

★ Write once, use again and again!



EXAMPLE

```
def greet(name):
    return "Hello, " + name + "!"

# Reuse the function
msg1 = greet("Alice")
msg2 = greet("Bob")
msg3 = greet("Coder")
print(msg1)
print(msg2)
print(msg3)
```

One function, many uses!



REMEMBER:

- ★ Use **return** to send a value back.
- ★ Save the result in a **variable**.
- ★ Reuse your functions to keep code clean and smart!

Returning values makes your programs more **powerful**, **flexible**, and **fun** to build!



Module 7: Arrays and Objects

Lesson 5:

Mini Project

Let's Code!



In this mini project, we will create three cool features using **arrays** and **objects**. Let's get creative!

1

Greeting Function



Create a **function** that takes a **name** as input and returns a friendly greeting message.

```
function greet(name) {  
  return "Hello, " +  
    name +  
    "! Welcome!";  
}
```

Example

```
greet("Ava")  
↓  
"Hello, Ava!  
Welcome!"
```

2

Area Calculator

Create an **object** that can calculate the area of different shapes using their dimensions.



Rectangle
(w x h)



Square
(s x s)



Circle
($\pi \times r^2$)

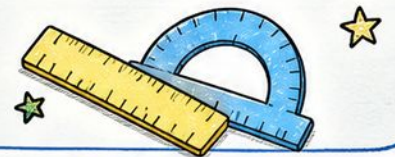


Triangle
($\frac{1}{2} \times b \times h$)

```
let areaCalculator = {  
  rectangle: function(w, h) {  
    return w * h;  
  },  
  square: function(s) {  
    return s * s;  
  },  
  circle: function(r) {  
    return Math.PI * r * r;  
  },  
  triangle: function(b, h) {  
    return 0.5 * b * h;  
  }  
};
```

Example

```
areaCalculator.rectangle(5, 3)  
→ 15  
areaCalculator.circle(7)  
→ 153.94
```



3

Random Joke Generator

Store a list of jokes in an **array** of **objects**. Create a function that returns a random joke.

Why do programmers prefer dark mode?
Because light attracts bugs! 😄

```
let jokes = [  
  { id: 1, joke: "Why do programmers prefer dark mode? Because light attracts bugs!" },  
  { id: 2, joke: "Why was the JavaScript developer sad? Because he didn't know how to Node." },  
  { id: 3, joke: "I would tell you a UDP joke, but you might not get it." }  
];  
  
function getRandomJoke() {  
  let index = Math.floor(Math.random() * jokes.length);  
  return jokes[index].joke;  
}
```

Example

```
getRandomJoke()  
→ "Why was the JavaScript developer sad? Because he didn't know how to Node."
```



Great job, coder! You just built three **awesome** features!
Keep practicing and keep building! ❤️

www.codescompiler.com



Code today,
Create tomorrow,
Change the world!

An array is a special variable that can hold **multiple items** of the **same type**, in order.

Introduction to Arrays

Let's store more, together!



1 Creating Arrays

You can create an array and store **multiple values** in it.



Think of an array as a **row of boxes**!

Index →

0	1	2	3	4
10	20	30	40	50

Array: numbers

```
let numbers = [10, 20, 30, 40, 50];
```

Key Idea

- Items are stored in order.
- Each item has an index starting from 0.

2 Accessing Items

You can access each item in an array using its **index**.



Use the **index** to get the item you want!

Index →

0	1	2	3	4
10	20	30	40	50

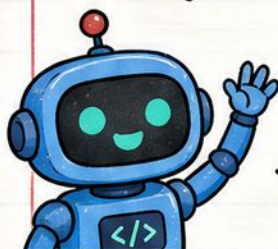
```
let numbers = [10, 20, 30, 40, 50];  
let thirdItem = numbers[2]; // 30
```

Examples

```
numbers[0] = 10  
numbers[1] = 20  
numbers[2] = 30  
numbers[3] = 40  
numbers[4] = 50
```

3 Array Length

The **length** property tells you how many items are in the array.



Length helps you **loop** through all items!

Index →

0	1	2	3	4
10	20	30	40	50

5 items

```
let numbers = [10, 20, 30, 40, 50];  
let size = numbers.length; // 5
```



Remember

Index starts at 0, so the last index is **length - 1**.

Last index here:
5 - 1 = 4

In Short



Create an array to store multiple items.



Access items using their index.



Use **.length** to find how many items it has.



Array Methods

Array methods help us work with arrays easily!

Think of an array as a list of items in a box.

1	2	3	4
---	---	---	---

Arrays have built-in methods that let us add, remove, and sort items with just a few lines of code!

1

Adding Items

We can add new items to an array using different methods.

- **push()** - adds to the end

```
let fruits = ["apple", "banana"];
fruits.push("orange");
```

Before:

"apple"	"banana"
---------	----------

After:

"apple"	"banana"	"orange"
---------	----------	----------

- **unshift()** - adds to the beginning

```
let fruits = ["banana", "orange"];
fruits.unshift("apple");
```

Before:

"banana"	"orange"
----------	----------

After:

"apple"	"banana"	"orange"
---------	----------	----------

Let's add something new!

2

Removing Items

We can remove items from an array using these methods.

- **pop()** - removes from the end

```
let fruits = ["apple", "banana", "orange"];
fruits.pop();
```

Before:

"apple"	"banana"	"orange"
---------	----------	----------

After:

"apple"	"banana"	removed!
---------	----------	----------

- **shift()** - removes from the beginning

```
let fruits = ["apple", "banana", "orange"];
fruits.shift();
```

Before:

"apple"	"banana"	"orange"
---------	----------	----------

After:

removed!	"banana"	"orange"
----------	----------	----------

Out with the old!

3

Sorting Arrays

We can sort items in an array in alphabetical or numerical order.

- **sort()** - sorts alphabetically

```
let names = ["Zara", "Bob", "Alice"];
names.sort();
```

Before:

"Zara"	"Bob"	"Alice"
--------	-------	---------

After:

"Alice"	"Bob"	"Zara"
---------	-------	--------

Let's get things in order!

- **sort((a, b) => a - b)** - sorts numbers

```
let numbers = [3, 1, 4, 2];
numbers.sort((a, b) => a - b);
```

Before:

3	1	4	2
---	---	---	---

After:

1	2	3	4
---	---	---	---

Remember! ★

- ✓ push() - add at end
- ✓ unshift() - add at start
- ✓ pop() - remove from end
- ✓ shift() - remove from start
- ✓ sort() - put in order

Practice makes perfect!
Try these methods with your own arrays.

Example Array:

"cat"	"dog"	"bird"
-------	-------	--------

You try!

MODULE 7

Arrays
and
Objects



Looping Through Arrays

Arrays help us store lists of items. Loops help us work with each item — **one by one!**



Lesson 3: Looping Through Arrays



Loops let us go through each element in an array to perform tasks like **showing data** or **searching** for something.



1 Using Loops

We use loops (like **for loop**) to visit each element in an array.

```
fruits = ["Apple", "Banana", "Mango", "Orange"]  
for i in range(len(fruits)):  
    print(fruits[i])
```

The loop starts at index 0 and goes to the **last index**.

Index →	0	1	2	3
fruits →	"Apple"	"Banana"	"Mango"	"Orange"

----- Loop moves from left to right ----->

Think of it like visiting each friend in a line! 😊

2 Displaying Data

Loops help us show all the items in an array clearly.



Great for making lists, menus, or reports!

Example Output:

Apple
Banana
Mango
Orange



All items displayed one by one!

3 Searching Arrays

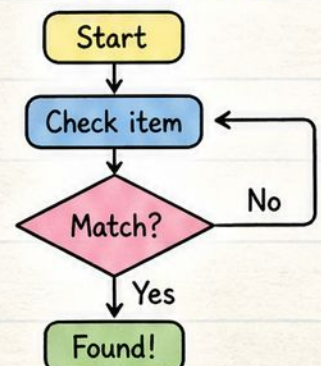
Loops help us search for a specific item in the array.



We check each item until we find it (or reach the end).

```
target = "Mango"  
found = False  
for item in fruits:  
    if item == target:  
        found = True  
        break  
print("Found!" if found else "Not Found.")
```

How it works:



www.codescompiler.com

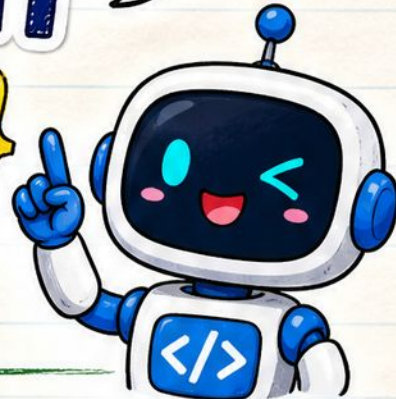


Lesson 4:

Introduction to Objects

Think of an object as a real-world thing with **features!**

In programming, an object is a collection of related data (properties) and their values.



1

Properties

Properties are the **characteristics** or features of an object. They describe the object.



Properties are like the questions we can ask about an object.

Example: A Car



- make
- model
- color
- year
- number of doors
- engine type

2

Values

Values are the **actual data** stored in the properties. They give specific information about the object.

Each property has a value. Values can change!

Example: A Car (Values)

```
car = {
  make: "Toyota",
  model: "Corolla",
  color: "Blue",
  year: 2020,
  numberOfDoors: 4,
  engineType: "Petrol"
}
```

← value of make
← value of model
← value of color
← value of year
← value of number of doors
← value of engine type

3

Real-Life Examples

Objects help us represent things from the **real world** in our programs. Here are more examples!



Objects make our code **organized**, **reusable**, and easy to understand!

Person



- name: "Alice"
- age: 14
- height: "150cm"
- city: "Delhi"

Book



- title: "The Jungle Book"
- author: "R. Kipling"
- pages: 189
- genre: "Adventure"

Student



- name: "Rahul"
- grade: 8
- rollNumber: 23
- section: "A"



Module 8: JavaScript and HTML

Lesson 5: Mini Project

Small ideas
Big learning!

Mini Project



Let's build 3 fun projects using
HTML, **CSS** & **JavaScript**!



1

Favorite Movies List



Create a webpage that shows your favorite movies in a list. Add titles, genres, and your ratings!



My Favorite Movies

Movie Title	Genre	Rating
Inception	Sci-Fi	★★★★★
The Lion King	Animation	★★★★☆
3 Idiots	Comedy/Drama	★★★★★
Interstellar	Sci-Fi	★★★★☆

You'll Practice:

- ✓ HTML Lists
- ✓ Tables
- ✓ Styling with CSS
- ✓ Looping with JS

2

Student Information Card



Design a student information card that displays name, photo, class, age and hobbies!

Student Information Card



Name : Aryan Sharma
 Class : 7th Grade
 Age : 13
 Hobbies : Coding, Reading, Playing Guitar

You'll Practice:

- ✓ HTML Elements
- ✓ CSS Styling
- ✓ Images
- ✓ Layout Design

3

Shopping List App



Build a simple app to add items to your shopping list. Mark items as bought and remove what you don't need!



My Shopping List

Enter item...

Add Item

☒ Milk	Bought	
☐ Bread		
☒ Eggs	Bought	
☐ Apples		

You'll Practice:

- ✓ Input & Buttons
- ✓ Add/Remove Items
- ✓ Events in JS
- ✓ Local Storage



Keep Coding,
Keep Growing!



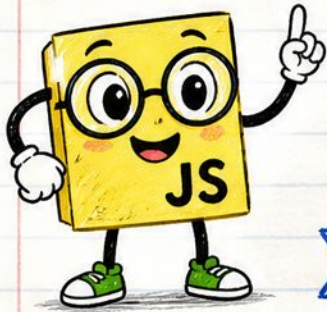
www.codescompiler.com



Module 8: JavaScript and HTML

Connecting JavaScript to HTML

Let's make
HTML Interactive
with JavaScript!



JavaScript brings life and interactivity to your HTML pages!

1 Script Tags

You can write JavaScript code directly inside your HTML file using `<script>` tags.

Best for:

- ★ Small scripts and quick tests.



HTML

```
<!DOCTYPE html>
<html>
<head>
  <title>My Page</title>
</head>
<body>
  <h1>Hello, World!</h1>
  <script>
    alert("Hello from JavaScript!");
  </script>
</body>
</html>
```

The script runs
right on this
page!

2 External JavaScript Files

You can keep your JavaScript code in a separate `.js` file and link it to your HTML using the `<script src>` tag.

Best for:

- ★ Larger projects and clean, organized code.



index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>My Page</title>
</head>
<body>
  <h1>Welcome!</h1>
  <script src="script.js"></script>
</body>
</html>
```

script.js

```
// This is an external
// JavaScript file

console.log("Hello
  from script.js!");

document.querySelector("h1")
  .style.color = "blue";
```

★ The `src` attribute tells the browser where the JavaScript file is located.

3 Browser Execution

When the browser loads your HTML page, it reads the `<script>` tags and executes the JavaScript code.

What happens?

- HTML is read and displayed.
- JavaScript runs line by line.
- The page becomes interactive!



index.html

```
<html>
<body>
  ...
  <script src="script.js">
  </script>
</body>
</html>
```

Browser
loads
HTML



Browser
executes
JavaScript



Page is
Interactive!



Remember: JavaScript + HTML = Powerful Web Pages!

www.codescompiler.com



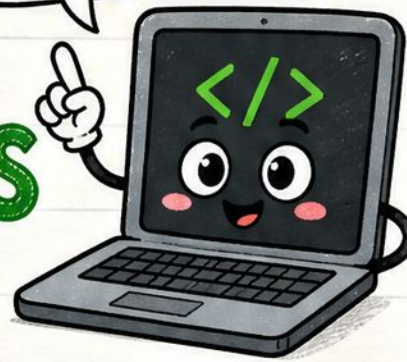
Module 8: JavaScript and HTML

Lesson 2:

Finding HTML Elements

JavaScript can talk to HTML by finding
and working with its elements!

JavaScript
needs elements
to do **magic**!



1 getElementById()

Finds a single element using its
unique ID.

```
// HTML
<h1 id="title">Hello, World!</h1>

// JavaScript
let heading = document.getElementById("title");
heading.style.color = "blue";
```

ID is like a
name tag —
it must be
unique!

returns
one element

index.html

```
<body>
  <h1 id="title">Hello, World!</h1>
  <p id="info">Welcome to
    JavaScript!</p>
  <button id="btn">Click Me</button>
</body>
```

2 Query Selectors

Finds element(s) using **CSS selectors**.

```
// First match only
let firstPara = document.querySelector("p");
firstPara.style.fontWeight = "bold";

// All matches (returns a NodeList)
let allParas = document.querySelectorAll("p");
allParas.forEach(p => p.style.color = "green");
```

You can use any **CSS selector**!

- p (by tag)
- .box (by class)
- #info (by id)
- div p (descendant)

Think like
CSS, select
like a **pro**!



3 Selecting Elements

Once you find an element, you can **read**,
change, or **interact** with it!



element.textContent	→ read or set text
element.innerHTML	→ read or set HTML
element.style.color	→ change style
element.addEventListener()	→ add events

You've found it,
now make it
awesome! ❤️



Tip

IDs should be
unique.

Use **classes** for
grouped elements.

Use **meaningful**
names!

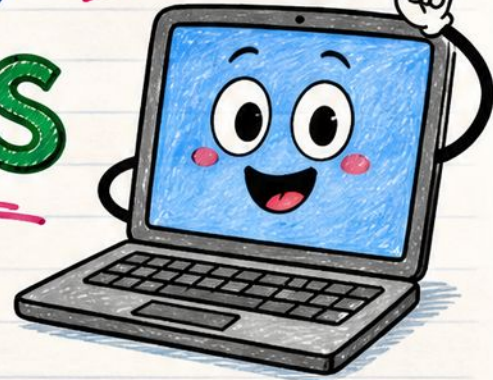
Module 8: JavaScript and HTML

Lesson 3:

Changing Web Pages

Small changes
in code,
Big changes
on the page! ♥

With JavaScript, we can update content
and make our web pages interactive!



1

Updating Text

We can change the text
on a web page
using JavaScript.



New
words,
new ideas!



Before



After



```
document.getElementById("title").innerText = "Hello, Friend!";
```

find the element → set new text

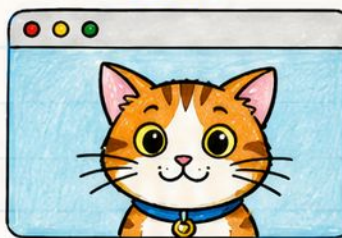
2

Changing Images

We can swap images
to show something
new and exciting!



Before



After



```
document.getElementById("petImage").src = "dog.png";
```

find the image → set new image

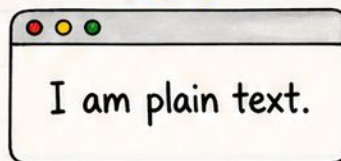
3

Modifying Styles

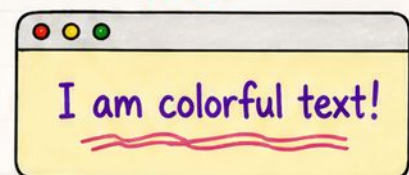
We can change colors,
sizes, and more to
make our page look
awesome!



Before



After



```
1 document.getElementById("msg").style.color = "purple";  
2 document.getElementById("msg").style.fontSize = "24px";  
3 document.getElementById("msg").style.backgroundColor = "#fff4a3";
```

change color → change size → change background

Practice
makes
perfect!



Experiment, explore, and have fun
making your web pages come alive!



Event Handling



Event Handling is how we **listen** for **user actions** and **respond** to them using **JavaScript**.

Events = Actions

- ✓ Clicks
- ✓ Mouse moves
- ✓ Key presses and more!

1

Button Clicks

When a user clicks a button, we can run a function.

Try clicking the button!

Click Me!

HTML

```
<button id="myBtn">Click Me!</button>
```

JS

```
document.getElementById("myBtn")
  .addEventListener("click", function() {
    alert("Button was clicked! 🎉");
  });
```

The click event runs the code inside the function!



2

Mouse Events

We can detect actions like moving the mouse, clicking, or hovering.



mousemove



click



mouseover

HTML

```
<div id="box">Hover over me!</div>
```

JS

```
const box = document.getElementById("box");
box.addEventListener("mouseover", function() {
  box.style.backgroundColor = "#ffe082";
  box.innerText = "Yay! You hovered! 😊";
});
```

Mouse events help make pages more interactive and fun!



3

Keyboard Events

We can listen for keys pressed on the keyboard.

Type in the box!

Type something...



HTML

```
<input type="text" id="myInput" placeholder="Type something...">
```

JS

```
document.getElementById("myInput")
  .addEventListener("keydown", function(event) {
    console.log("Key pressed: " + event.key);
  });
```

keydown detects when a key is pressed!



Summary

Events let our web pages respond to user actions. With JavaScript, we can make websites dynamic and awesome! 🚀

Remember:

- ★ Listen for the event
- ★ Write the code to handle it
- ★ Make something amazing!

Practice more,
Build more,
Become a
Coding Star! ★

Mini Project

Let's make web pages come alive!



Let's build 3 fun web projects using **HTML**, **CSS** & **JavaScript**!

Small projects,
Big Learning,
Endless Fun!

1 Click Counter

Count how many times a button is clicked.

- ★ Click the button.
- ★ Increase the count.
- ★ Show the count on the page.



Try clicking me!



2 Color Changer

Change the background color with a button click.

- ★ Click a color button.
- ★ The page background changes.
- ★ Make your page colorful!



3 Interactive Greeting Page

Greet the user with their name and make it interactive.

- ★ Ask the user for their name.
- ★ Show a personalized greeting.
- ★ Make it friendly & fun!



It's nice to meet you! 😊



Think
Code
Test
Improve

Build small.
Learn big.
Be a Web Wizard!



Great projects today,
awesome developer tomorrow!

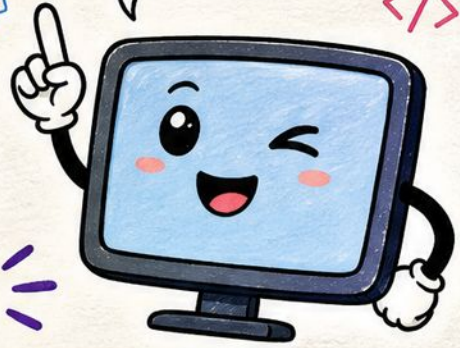


Module 9

Fun Web Interactions

Working with Forms

Forms help websites talk to users like **YOU!**



Lesson 1: Working with Forms

1 Input Fields

Input fields are places where users type or select information.



Different types of input fields collect different kinds of data.

Sign Up Form

 Name:

 Email:

 Age:

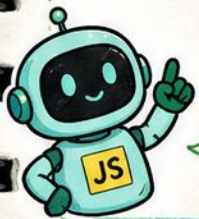
 Favorite Color:

Examples:

- Text field (e.g., Name)
- Email field (e.g., Email)
- Number field (e.g., Age)
- Dropdown (e.g., Color)

2 Reading Values

We use **JavaScript** to read the data users enter in the form.



We get the **value** from an input field and use it however we need!

```
// Reading values from a form
let name = document.getElementById("name").value;
let email = document.getElementById("email").value;
let age = document.getElementById("age").value;
let color = document.getElementById("color").value;

console.log("Name: " + name);
console.log("Email: " + email);
console.log("Age: " + age);
console.log("Favorite Color: " + color);
```

What happens?

- ✓ We find the input field by its ID.
- ✓ We read the value user typed or selected.
- ✓ Now we can use the data! 🎉

3 Form Validation

Validation checks if the data is **correct** before submitting the form.



Always validate to keep data clean and users happy! 😊

Example: Check Email

```
let email = document.getElementById("email").value;

if (email === "") {
  alert("Email is required!");
  return false;
}

if (!email.includes("@")) {
  alert("Please enter a valid email!");
  return false;
}

alert("Form submitted successfully! 🎉");
return true;
```

Common Checks:

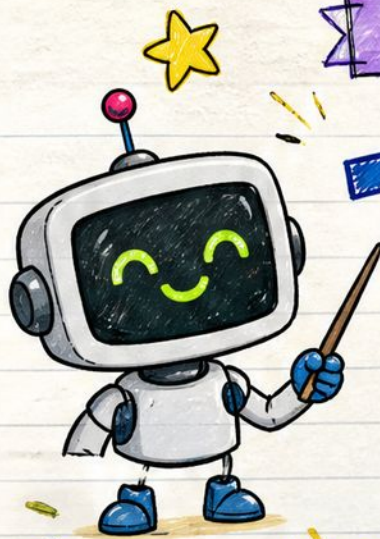
- ✓ Required Fields (should not be empty)
- @ Valid Email (contains @)
- 123 Number Range (in a proper range)
- 🔒 Password Strength (more secure)

★ Good validation = Happy users! ★

Module 9: Fun Web Interactions

Lesson 2:

Timers and Delays



Make things happen... after a delay or again and again!

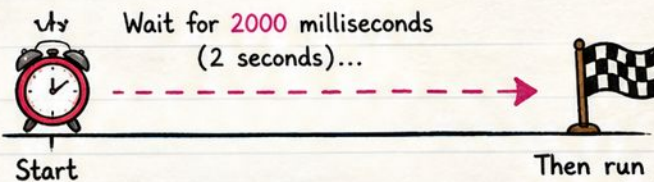
JavaScript gives us superpowers to control time on the web!

1 setTimeout()

Runs a function **once**, after a delay (in milliseconds).



Great for showing messages, popups, or starting something after a short wait!



```
setTimeout(() => {  
  console.log("Hello after 2 seconds!");  
  showMessage();  
}, 2000);
```

Runs once after 2 seconds.



I'll wait and then surprise you!

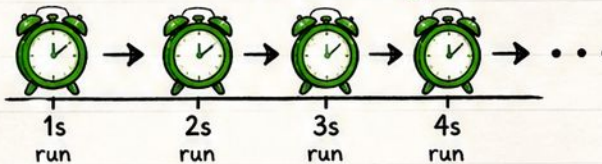
2 setInterval()

Runs a function **repeatedly**, every fixed interval (in milliseconds).



Perfect for clocks, counters, polls, or anything that should happen again and again!

Every 1000 milliseconds (1 second), run the code... **again and again!**



```
setInterval(() => {  
  console.log("Tick");  
  updateClock();  
}, 1000);
```

Runs every 1 second (repeats forever until stopped).



I love doing things on repeat!

To stop it:
`clearInterval(myTimer);`
(Don't forget to store the interval ID!)

3 Animation Timing

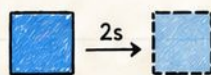
Control how animations move and feel using time! Smooth, fun, and awesome! ✨



Use CSS transitions, CSS animations, or JavaScript timing to create magic on screen!

It's all about duration, delay, and timing functions!

CSS Transition



```
box {  
  transition: all 2s ease;  
}
```

Changes happen smoothly over 2 seconds.

CSS Animation



```
@keyframes bounce {  
  0% { transform: translateY(0); }  
  50% { transform: translateY(-20px); }  
  100% { transform: translateY(0); }  
}
```

Steps (keyframes) happen inside the animation.

JavaScript Timing



```
setInterval(() => {  
  moveBall();  
}, 20);
```

Small interval = smoother animation!

Timing Tips

- ✓ Lower time = faster (but can be jumpy)
- ✓ Higher time = slower (but smoother)
- ✓ Use easing (CSS) for natural motion!

```
<html>  
<body>  
<fun/>  
</body>  
</html>
```



Practice. Experiment. Have Fun with Time!



www.codescompiler.com



You've got the power of time!

Lesson 3

Random Numbers

Random numbers are numbers that are chosen by **chance**. They are used in games, apps, quizzes and so much more!

Computers can't decide on their own... but they can generate **random numbers**!



★ Random numbers make our programs more fun and exciting! ★

1 Generating Random Values

Computers generate random numbers within a **range**.

Example:

```
import random
num = random.randint(1, 100)
print(num)
```

What happens?

The program picks a random number between **1 and 100** each time you run it.

Common Uses

- Games
- Simulations
- Quizzes
- Lotteries
- And more!



2 Dice Roller

Let's roll a dice! A dice has 6 sides (numbers 1 to 6).



How it works

```
import random
dice = random.randint(1, 6)
print("You rolled:", dice)
```

Example Output

You rolled: **4**



3 Lucky Number Generator

Generate a **lucky number** and see if it matches your guess!



Feeling lucky? Let's find out!

How it works

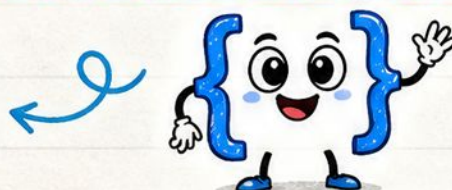
```
import random
lucky = random.randint(1, 50)
guess = int(input("Guess a number (1-50): "))
if guess == lucky:
    print("🎉 You got it! Lucky you!")
else:
    print("Try again! Better luck next time.")
```

What you can do!

- ✓ Create your own number range
- ✓ Add hints
- ✓ Keep score
- ✓ Make it a challenge with friends!



Random numbers make every run **unique** and every moment **exciting**!



Keep experimenting and have fun coding!

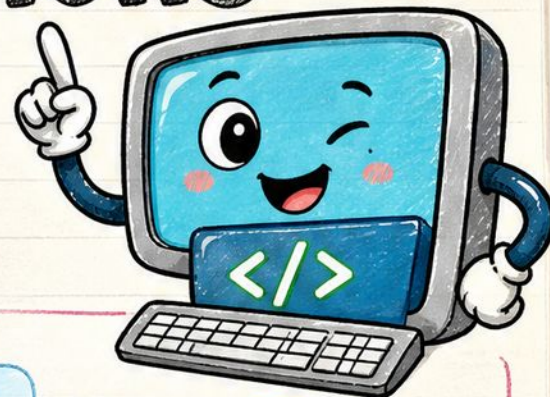




Simple Animations

Let's animate!

Animations make websites **fun** and **engaging**.
Let's bring elements to life on the web!



1

Moving Objects

Move elements from one place to another.



Example



We can use **CSS** transitions or animations to move objects smoothly!

```
.car {  
  transition: transform 2s linear;  
}  
  
.car:hover {  
  transform: translateX(300px);  
}
```

Try it!

Hover over the car and watch it move!

2

CSS + JavaScript

CSS makes it **smooth**.
JavaScript makes it **smart**!

CSS

Style & Animate

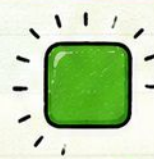
+

JS

Add Logic & Control

CSS

```
.box {  
  width: 60px;  
  height: 60px;  
  background: #4CAF50;  
  border-radius: 10px;  
  transition: all 0.5s ease;  
}
```



Click the box and see the **magic**!

JavaScript

```
document.querySelector(".box").addEventListener("click", () => {  
  document.querySelector(".box").classList.toggle("big");  
});
```

```
.box.big {  
  transform: scale(1.5);  
  background: #2196F3;  
}
```

3

Interactive Effects

Animations respond to user actions like **hover**, **click** or **scroll**.

Examples:

Hover Effect



Changes when you hover over it.

Click Effect



Animates when you click.

Scroll Effect



Animations trigger as you scroll.

Cool Ideas

- ☒ Animate buttons
- ☒ Loading spinners
- ☒ Page transitions
- ☒ Scroll reveals ...and more!



Lesson 5:

Mini Project

Let's build awesome things with JavaScript!



</>
Keep Coding!



1 Digital Clock

A simple digital clock that shows the current time in hours, minutes and seconds.



What we learn:

- ✓ Date & Time in JS
- ✓ setInterval()
- ✓ Updating the DOM
- ✓ Styling with CSS



2 Dice Game

Roll two dice and see who wins! Player with higher dice wins.



What we learn:

- ✓ Random numbers
- ✓ Events (click)
- ✓ DOM manipulation
- ✓ Game logic



3 Reaction Timer Game

Test your reflexes! Click the button as soon as it turns green. Measure your reaction time.



Get Ready...



Your Time: 0.000 s

Best Time: 0.000 s

What we learn:

- ✓ setTimeout()
- ✓ Events & Timing
- ✓ Calculating time
- ✓ Improving reflexes!

Challenge yourself!



Practice makes perfect!



These mini projects help you strengthen your JavaScript skills with fun and creativity!



A great project starts with a great plan! 💡

Project Planning

Plan it
Build it
Make it Awesome! ✨

Good planning = Successful Project ★



Before we start coding, let's plan our project step by step!

1 Understanding Requirements



Know what your project should do and who will use it.

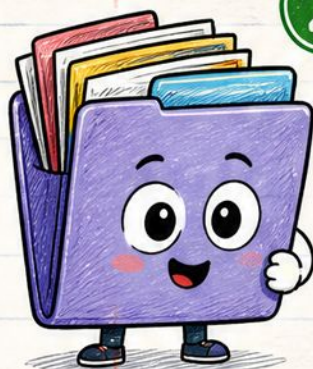
- ✓ What is the main goal?
- ✓ Who is the target user?
- ✓ What are the must-have features?
- ✓ Any special rules or limitations?

Example

Project: **To-Do List App**

- 🎯 Goal: Help users manage tasks
- 👥 Users: Students, Anyone
- ★ Must-have: Add, Edit, Delete, Mark Complete
- ⚠️ Limitations: Use only HTML, CSS, JavaScript

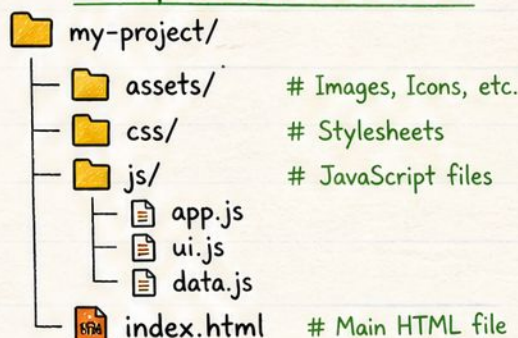
2 Organizing Code



Keep your code clean, simple and easy to manage.

- ✓ Plan your folder structure.
- ✓ Use meaningful file names.
- ✓ Break code into smaller parts (functions, modules).
- ✓ Follow consistent naming and indentation.

Example Folder Structure



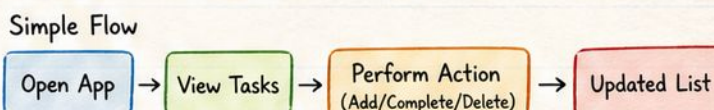
3 Designing Features



Think about the features and how they will work.

- ✓ List all the features.
- ✓ Plan how each feature behaves.
- ✓ Draw simple flow or wireframes.
- ✓ Focus on user experience (UX).

Example: To-Do List App Features



★ Pro Tip

Spending time on planning saves time on coding and debugging!



Plan well,
Code better!

Remember

- ✓ Understand the problem
- ✓ Plan your solution
- ✓ Build step by step
- ✓ Test and improve



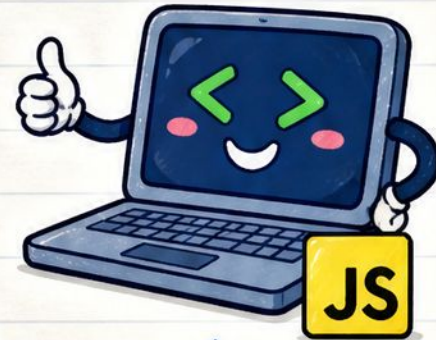
Module 10: Final JavaScript Projects



Lesson 2: Quiz Game Project

Let's build a fun Quiz Game using JavaScript!

Quiz Game Project



In this project, we build an **interactive Quiz Game** where players **answer questions**, **earn points**, and see their **final results**!

Think!
Answer!
Score!

1

Questions and Answers

Display one question at a time with multiple choices.

The player selects an answer by clicking a button.

Instant feedback lets the player know if the answer is **correct** or **incorrect**.

Q1. What is JavaScript?

- A. A programming language ✓
- B. A markup language
- C. A database
- D. An operating system



We use:

- ✓ Questions Array
- ✓ Options Array
- ✓ Correct Answer Index

2

Score Tracking

Increase the **score** when the player answers correctly.

Keep track of the total questions and current score.

Show the score during the quiz.



Score Board

Score: 3

Total Questions: 10

Every correct answer earns you 1 point!

3

Results Screen

At the end of the quiz, show the results screen.

Display the final score, percentage, and a message based on performance.

Give the option to **play again**!

Quiz Completed! 🎉

Your Score

7 / 10

70%

Good Job! Keep Learning! 👏

Play Again



www.codescompiler.com





Module 10

Final JavaScript Projects

Lesson 3: Memory Game Project

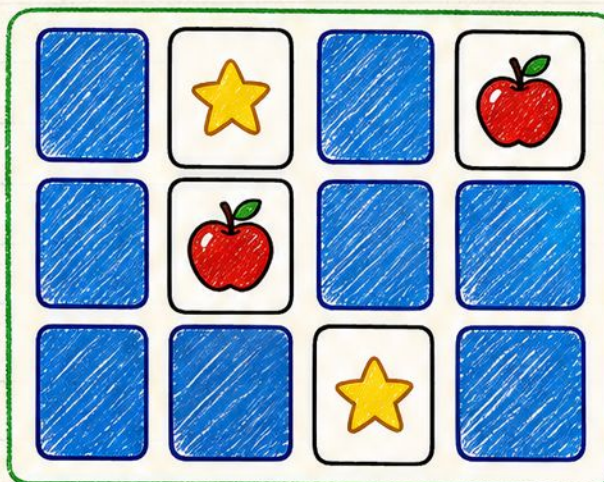
Build a fun
Memory Game
using JavaScript!

Memory Game Project

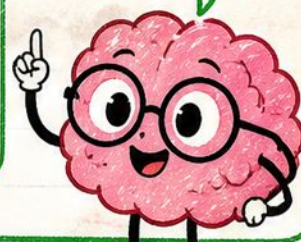
1 Matching Cards

The game has cards placed face down. Flip two cards at a time to find a matching pair!

★ Matching pairs stay open. Keep going until you find them all!

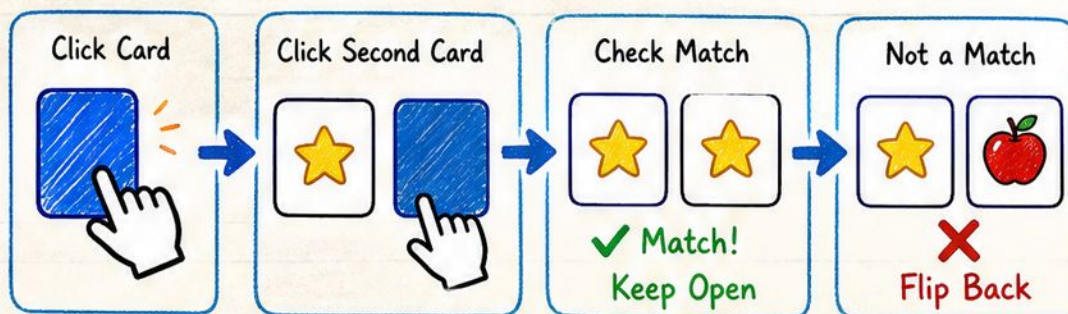


Find all matching pairs!



2 Game Logic

- Flip a card when it's clicked.
- Flip a second card.
- If the cards match, keep them open.
- If they don't match, flip them back.
- Count the moves and keep playing!



Use JavaScript to handle clicks, compare cards, update the board, and track moves.

3 Winning Conditions

You win when all pairs are matched!



Show a success message and your final moves.



Module 10

Final JavaScript Projects

Lesson 4

You code
the story.
Players live
it!

Interactive Story Game



An Interactive Story Game lets the player make **choices**, shape the **adventure**, and reach different **endings**!



1

Choices and Decisions

The player makes choices that change what happens next in the story.

★ Every choice leads to a **new path**!



2

Multiple Endings

Different choices lead to different outcomes. There are many possible endings!

★ Replay and explore **all the endings**!

Your story, your ending!

You Win!



You saved the kingdom and became a hero!

A New Friend



You found a dragon and gained a friend!

Lost in Time



You were trapped in a time loop...



3

User Interaction

Players interact by clicking buttons, typing answers, or exploring the story.

★ The player is **part** of the **adventure**!

Ways to interact

Click Choices

What do you do?

Look around

Move forward

Type Answers

What is your name?

Alex

Submit

Explore & Collect



Code today,
create **magic**
tomorrow!

You imagine it.
You build it.
They play it!

JS

Final Capstone Project



This is your chance to build something awesome!

Bring everything you've learned together and build a complete, interactive web project!

1 Complete Interactive Website

Build a fully functional website that users can interact with!

- ☒ Responsive Design
- ☒ Interactive Features
- ☒ User Friendly
- ☒ Dynamic Content



What to include?

- Navigation
- Sections
- Forms
- Buttons
- Animations
- And more!



2 HTML + CSS + JavaScript

Use the core technologies to power your project.



Structure

+



Style

+



Behavior

★ Combine them to create amazing experiences on the web!



3 Project Showcase

Present your project and share your work with the world!

- ★ Show your skills
- ★ Get feedback
- ★ Inspire others
- ★ Be proud of your journey!



You did it!



www.codescompiler.com

