LECTURE 14

Regular Expressions and JSON

Checking that user input has the right shape, and exchanging structured data with a server — two skills every real project needs.

CSC336 Web Technologies

OVERVIEW

In This Lecture

1. Regex syntax: literals, character classes, quantifiers, and anchors
2. Groups, alternation, backreferences, and flags (g, i, m)
3. JavaScript's regex methods: test, exec, match, replace, split
4. Validating and parsing form data with regex
5. JSON syntax, and converting with JSON.stringify and JSON.parse

What Is a Regular Expression?

```
const pattern = /hello/;
console.log(pattern.test("hello world")); // true
console.log(pattern.test("goodbye"));      // false

// Dynamic pattern from a variable
const word = "hello";
const dynamicPattern = new RegExp(word);
```

A regex describes a set of strings -- use it to test a shape, find matches, or replace parts of a string.

Literals and Character Classes

```
/cat/       // matches the exact text "cat"

/[abc]/     // a single "a", "b", or "c"
/[a-z]/     // any single lowercase letter (a range)
/[0-9]/     // any single digit
/[^0-9]/    // ^ inside [] means NOT a digit

/\d/  /\D/  // digit / non-digit       -- same as [0-9]
/\w/  /\W/  // word char / non-word    -- same as [A-Za-z0-9_]
/\s/  /\S/  // whitespace / non-whitespace
/./         // any character except a newline
```

Quantifiers -- How Many Times

```
/a*/      // 0 or more "a"s
/a+/      // 1 or more "a"s
/a?/      // 0 or 1 "a" (optional)
/a{3}/    // exactly 3 "a"s
/a{2,4}/ // between 2 and 4 "a"s
/a{2,}/  // 2 or more "a"s
```

Anchors -- Positions, Not Characters

```
/^Hello/       // ^ = "start of string"
/world$/       // $ = "end of string"
/^Hello world$/ // must match the ENTIRE string exactly
/\bcat\b/      // \b = word boundary -- matches "cat" not "concatenate"
```

Groups, Alternation, Backreferences

```
// Groups -- apply a quantifier to more than one char
/(ab)+/.test("ababab"); // true -- "ab" repeated 1+ times

// Alternation -- "or"
/cat|dog/.test("I have a dog"); // true

// Backreference -- \1 refers back to group 1
/(\w+)\s\1/.test("hello hello"); // true -- repeated word
/(\w+)\s\1/.test("hello world"); // false -- different words
```

Flags Change How the Whole Regex Behaves

Flag	Meaning
g	Global -- find ALL matches, not just the first
i	Case-insensitive matching
m	Multiline -- ^ and $ match the start/end of each line

Flags in Practice

```
const text = "Cat cat CAT";
console.log(text.match(/cat/gi)); // ["Cat", "cat", "CAT"]
```

test() and exec()

```
// test() -- returns true/false, called on the regex
const hasDigit = /\d/;
console.log(hasDigit.test("abc123")); // true

// exec() -- returns match details, called on the regex
const dateRegex = /(\d{4})-(\d{2})-(\d{2})/;
const result = dateRegex.exec("Event date: 2026-09-05");
console.log(result[0]); // "2026-09-05" -- full match
console.log(result[1]); // "2026" -- first group
```

match() -- a String Method

```
const text = "Call 123-456-7890 or 987-654-3210";
console.log(text.match(/\d{3}-\d{3}-\d{4}/g));
// ["123-456-7890", "987-654-3210"]
```

match() and matchAll() are called ON THE STRING, with the regex as the argument -- the opposite of test()/exec().

replace() and split()

```
const messy = "hello   world    there";
console.log(messy.replace(/\s+/g, " ")); // "hello world there"

// Captured groups in the replacement, with $1, $2, ...
const date = "2026-09-05";
console.log(date.replace(/(\d{4})-(\d{2})-(\d{2})/, "$3/$2/$1")); // "05/09/2026"

const csvLine = "Ali, Sara,  Bilal";
console.log(csvLine.split(/,\s*/)); // ["Ali", "Sara", "Bilal"]
```

Validating an Email and a Password

```
function isValidEmail(email) {
  const emailRegex = /^[\w.+-]+@[\w-]+\.[a-zA-Z]{2,}$/;
  return emailRegex.test(email);
}
isValidEmail("student@comsats.edu.pk"); // true
isValidEmail("not-an-email");           // false

function isStrongPassword(password) {
  // 8+ chars, one uppercase, one lowercase, one digit
  const strongRegex = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d).{8,}$/;
  return strongRegex.test(password);
}
isStrongPassword("Abcdefg1"); // true
isStrongPassword("abcdefgh"); // false -- no uppercase, no digit
```

Validating a Phone Number

```
function isValidPakPhone(phone) {
  // e.g. 0300-1234567
  return /^03\d{2}-\d{7}$/.test(phone);
}
console.log(isValidPakPhone("0300-1234567")); // true
```

Regex Validates Shape, Not Truth

WARNING

A regex can confirm that student@comsats.edu.pk LOOKS LIKE an email address, but it cannot confirm the mailbox actually exists. Always pair client-side shape validation with real server-side verification for anything important.

Test Your Regex Interactively

TIP

Tools like regex101.com let you build and test a pattern against sample strings with live highlighting -- extremely useful while you're still learning the syntax.

JavaScript Object Notation

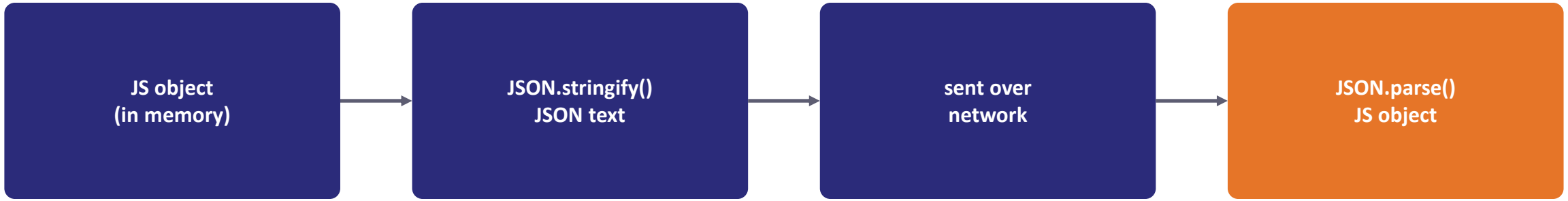
```
{
  "name": "Ayesha",
  "age": 22,
  "isStudent": true,
  "courses": ["Web Technologies", "Databases"],
  "address": { "city": "Lahore", "zip": "54000" },
  "graduationYear": null
}
```

JSON looks like a JS object literal, but is a strict, language-independent format for exchanging data between any two systems.

JSON's Stricter Syntax Rules

1. Keys must be double-quoted strings -- "name", never name or 'name'
2. String values must also use double quotes, not single quotes
3. Allowed value types: string, number, boolean, null, object, or array
4. No trailing commas, and no comments are allowed anywhere in JSON
5. No functions, undefined, or dates -- JSON only represents plain data

Object <-> JSON Text, Round Trip



JSON.stringify turns a live JS value into a plain string for the network or a file; JSON.parse turns JSON text back into a usable JS value.

JSON.stringify() -- Object to JSON Text

```
const student = {
  name: "Bilal",
  age: 21,
  courses: ["Web Technologies", "OOP"],
};

const jsonText = JSON.stringify(student);
console.log(jsonText);
// '{"name":"Bilal","age":21,"courses":["Web Technologies","OOP"]}'

// Pretty-print with indentation (useful for debugging)
console.log(JSON.stringify(student, null, 2));
```

JSON.parse() -- JSON Text to Object

```
const jsonText = '{"name":"Bilal","age":21}';
const obj = JSON.parse(jsonText);
console.log(obj.name);     // "Bilal"
console.log(obj.age + 1);  // 22 -- it's a real number, not a string
```

Invalid JSON Throws an Error

WARNING

JSON.parse throws a SyntaxError if the string is not valid JSON (single quotes, a trailing comma, etc). Always wrap JSON.parse on external data in a try...catch block.

Handling Bad JSON Safely

```
try {
  const data = JSON.parse(someText);
} catch (error) {
  console.error("Invalid JSON:", error.message);
}
```

You will use JSON.stringify/parse constantly once your JavaScript talks to a server through the Fetch API, starting next lecture.

KEY TAKEAWAYS

Lecture 14 in Seven Points

1. A regex describes text using literals, character classes (\d, \w, \s, [...]), quantifiers, and anchors (^, $, \b).
2. Groups () capture parts of a match; | means alternation; backreferences (\1) refer back to an earlier group.
3. test() and exec() are regex methods; match(), matchAll(), replace(), and split() are string methods that accept a regex.
4. Flags g, i, and m change global, case-insensitive, and multiline matching behavior.
5. Regex validates the SHAPE of input (emails, phone numbers, passwords) but cannot confirm the data is actually true.
6. JSON is strict: double-quoted keys and strings only, no comments, no trailing commas, no functions.
7. JSON.stringify converts a JS value to JSON text; JSON.parse converts it back -- always inside a try...catch for external data.