Valentina Silveira
Eagle ID: 901365377

CSCI 2503 – Survey of Programming Language
Georgia Southern University
Department of Computer Science
Fall 2023

Assignment 4
Description

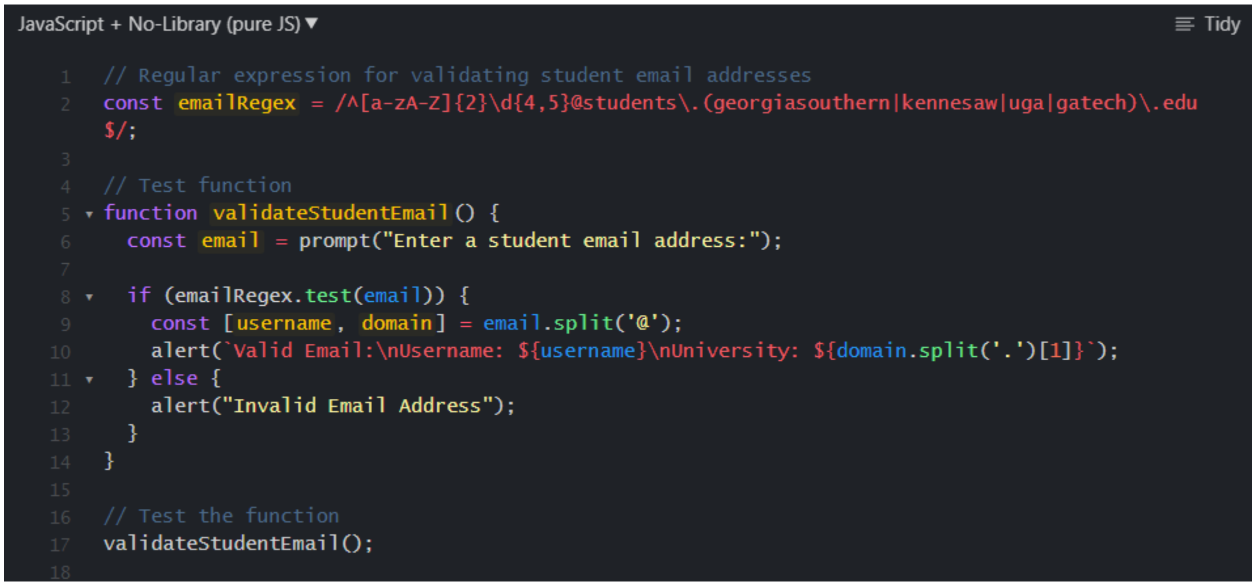
1. Regular Expressions –
Write a JavaScript snippet, using phone.js as a template, that
validates student email addresses based on the following rules and notes:

- All usernames for emails start with two letters and 4-5 numbers (only alphanumeric characters allowed)
- There is a required subdomain of "students" for each domain (e.g. students.georgiasouthern.edu)
- The USG is only testing this for the following university domains: georgiasouthern, kennesaw, uga, and gatech; no other email addresses should validate against this regular expression.
- The TLD will always be .edu

If an email address is validated, only show the username and the domain/university in an alert window. Use the provided phone number example as a starting point for completing this regular expression.
For simplicity, use the prompt()/alert() windows from the Input/Output JavaScript examples for both input and output.
Use https://jsfiddle.net/ or some other means to test your code. Spend some time testing small parts of your regex before trying to put it all together.

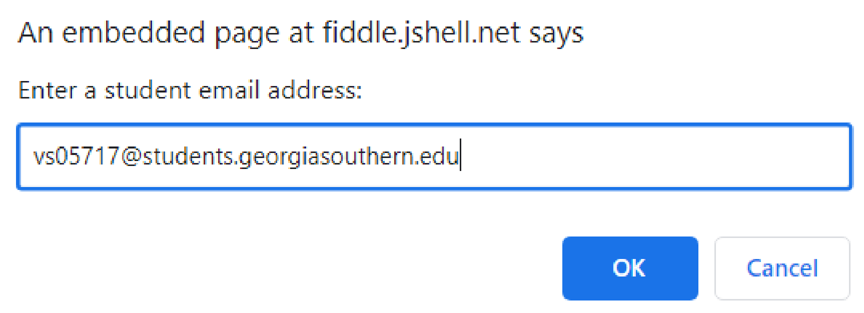
Valentina Silveira
Eagle ID: 901365377

Answer:



JavaScript + No-Library (pure JS) ▼

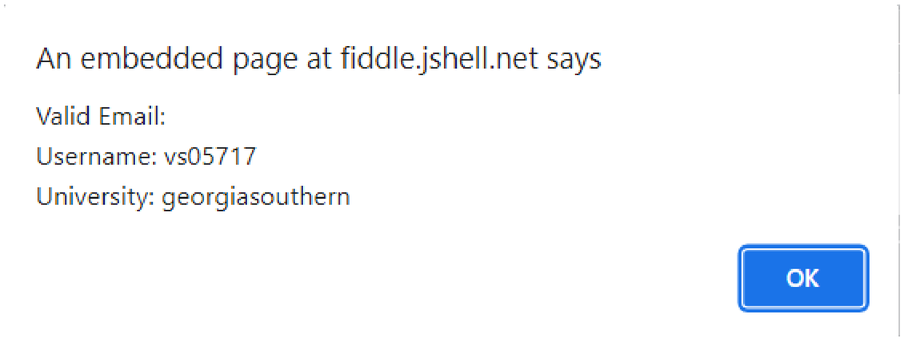
```javascript
// Regular expression for validating student email addresses
const emailRegex = /^[a-zA-Z]{2}\d{4,5}@students\.(georgiasouthern|kennesaw|uga|gatech)\.edu
$/;

// Test function
function validateStudentEmail() {
  const email = prompt("Enter a student email address:");

  if (emailRegex.test(email)) {
    const [username, domain] = email.split('@');
    alert(`Valid Email:\nUsername: ${username}\nUniversity: ${domain.split('.')[1]}`);
  } else {
    alert("Invalid Email Address");
  }
}

// Test the function
validateStudentEmail();
```



An embedded page at fiddle.jshell.net says

Enter a student email address:

vs05717@students.georgiasouthern.edu



An embedded page at fiddle.jshell.net says

Valid Email:
Username: vs05717
University: georgiasouthern

QUESTION(S):
In addition to the validation uses we have seen (email addresses/phone numbers), list at least 3 other potential uses of a regular expression.

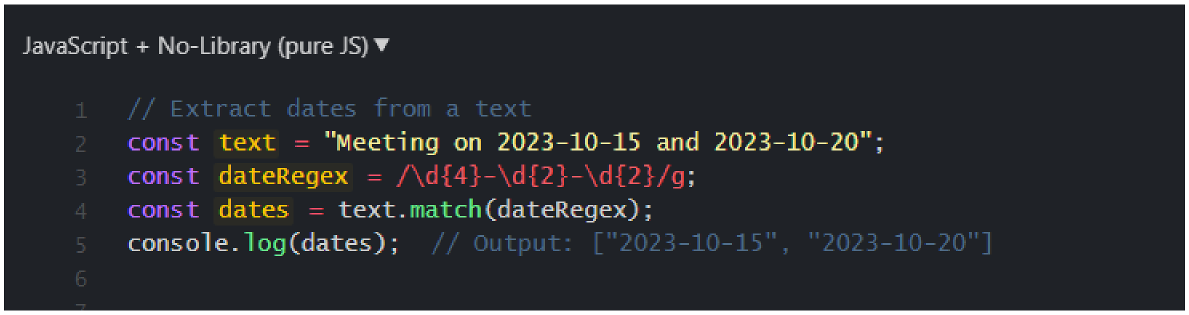
Regular expressions (regex) are versatile and can be applied in various scenarios for pattern matching and text manipulation. Here are three other potential uses of regular expressions:

Valentina Silveira
Eagle ID: 901365377

Answer:

1. Data Extraction from Text:

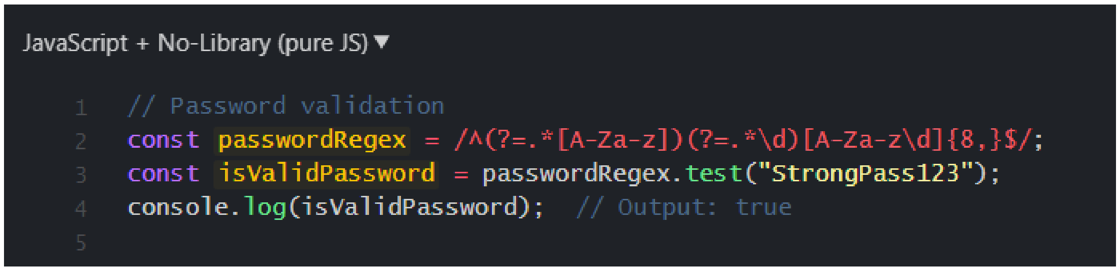
- Regular expressions can be used to extract specific pieces of information from a block of text. For example, extracting dates, URLs, or keywords from a document or web page.



```javascript
// Extract dates from a text
const text = "Meeting on 2023-10-15 and 2023-10-20";
const dateRegex = /\d{4}-\d{2}-\d{2}/g;
const dates = text.match(dateRegex);
console.log(dates);  // Output: ["2023-10-15", "2023-10-20"]
```

2. Data Validation in Forms:

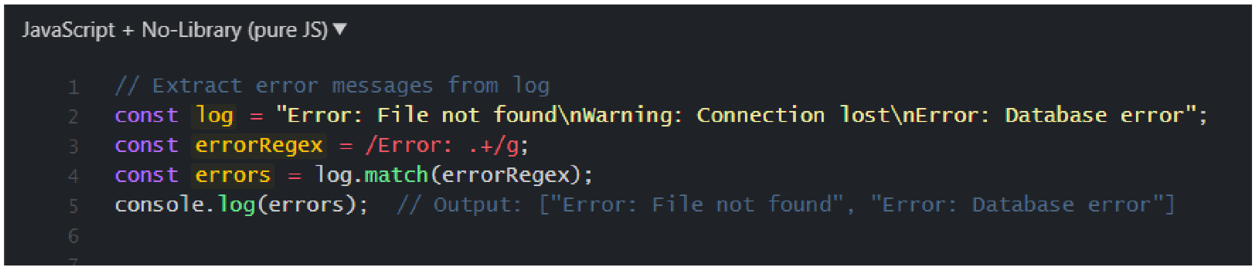
- Regular expressions are commonly used to validate user inputs in forms, such as validating zip codes, credit card numbers, or passwords.



```javascript
// Password validation
const passwordRegex = /^(?=.*[A-Za-z])(?=.*\d)[A-Za-z\d]{8,}$/;
const isValidPassword = passwordRegex.test("StrongPass123");
console.log(isValidPassword);  // Output: true
```

3. Log Analysis and Filtering:

- Regular expressions are useful for parsing and filtering log files or large datasets. They can help identify specific patterns or errors within logs.



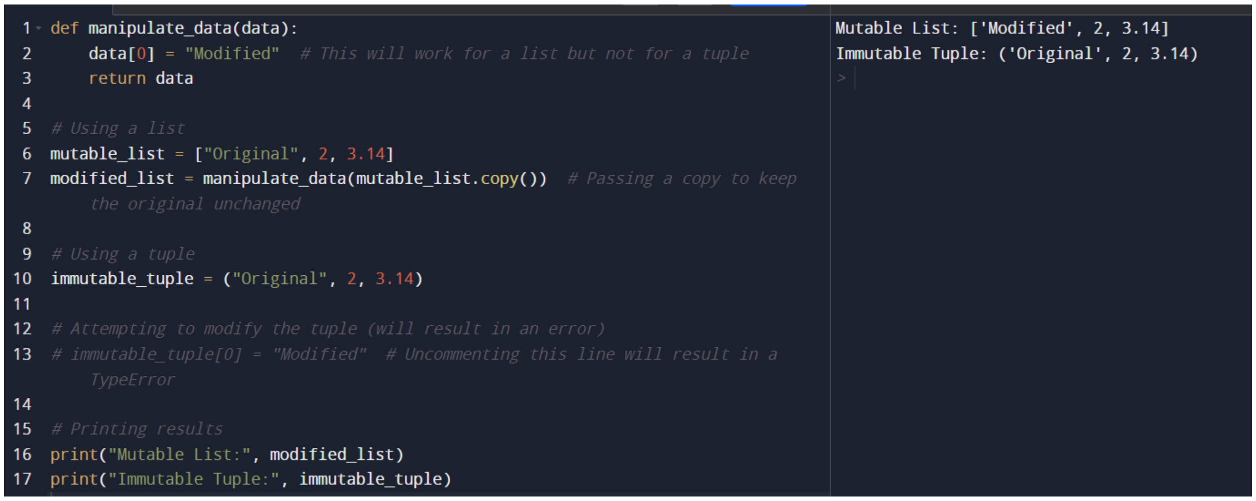
```javascript
// Extract error messages from log
const log = "Error: File not found\nWarning: Connection lost\nError: Database error";
const errorRegex = /Error: .+/g;
const errors = log.match(errorRegex);
console.log(errors);  // Output: ["Error: File not found", "Error: Database error"]
```

These are just a few examples, and regular expressions are widely used in various fields for tasks such as text search and replace, syntax highlighting, and more. However, it's important to use them judiciously, as complex regular expressions can be challenging to read and maintain.

Valentina Silveira
Eagle ID: 901365377

2. Python Tuples/Lists – Write a Python program that experiments with both tuples and lists.
QUESTION(S): In what situation might an immutable tuple provide more functionality than a mutable list? In what situation might a mutable list provide more functionality than an immutable tuple? Add your answer to the Word document created in Problem 1.

Answer:



```python
def manipulate_data(data):
    data[0] = "Modified"  # This will work for a list but not for a tuple
    return data

# Using a list
mutable_list = ["Original", 2, 3.14]
modified_list = manipulate_data(mutable_list.copy())  # Passing a copy to keep
    the original unchanged

# Using a tuple
immutable_tuple = ("Original", 2, 3.14)

# Attempting to modify the tuple (will result in an error)
# immutable_tuple[0] = "Modified"  # Uncommenting this line will result in a
    TypeError

# Printing results
print("Mutable List:", modified_list)
print("Immutable Tuple:", immutable_tuple)
```

```
Mutable List: ['Modified', 2, 3.14]
Immutable Tuple: ('Original', 2, 3.14)
>
```

Explanation:

- In the example, the manipulate_data function modifies the first element of the input list. However, this operation would not be possible with a tuple because tuples are immutable.

In what situation might an immutable tuple provide more functionality than a mutable list?

- Data Integrity: In situations where you want to ensure that the data remains constant and is not accidentally modified, you might choose to use an immutable tuple. For example, representing a set of constants or configuration settings.

In what situation might a mutable list provide more functionality than an immutable tuple?

- Dynamic Data: Lists are mutable, meaning you can modify, add, or remove elements after the list is created. In scenarios where you need to dynamically update or manipulate the data, a mutable list is more appropriate. For example, maintaining a list of user activities or items in a shopping cart.

3. C# Records –

Using C#, create a record for a Student that consists of fields for first name, last name, GPA, major, attempted hours, and completed hours. Write a C# program that creates at least three Student records and then prints information about each one. Attempt to change the values for one of the records. In a comment, explain why this does or does not work.

Answer:

Program:

```
using System;


// Define the Student record

record Student(string FirstName, string LastName, double GPA, string Major, int
AttemptedHours, int CompletedHours);


class Program

{

    static void Main()

    {

        // Creating three Student records

        var student1 = new Student("John", "Doe", 3.8, "Computer Science", 90, 75);

        var student2 = new Student("Jane", "Smith", 3.5, "Mathematics", 80, 65);

        var student3 = new Student("Alice", "Johnson", 3.9, "Physics", 95, 85);


        // Printing information about each student

        Console.WriteLine("Student 1: " + student1);

        Console.WriteLine("Student 2: " + student2);

        Console.WriteLine("Student 3: " + student3);
        // Attempting to change values for one record (Won't work due to immutability)

        // student1.GPA = 4.0; // This line will result in a compilation error

        // Comment Explanation:

        // Records in C# are immutable by default, meaning their values cannot be
changed after creation.

        // Attempting to modify a value in a record, like student1.GPA = 4.0; would result in
a compilation error.

        // Comparing to class-based approach: Records in C# are more concise and
provide value-based equality by default.

    }

}
```

Explanation:

- In this example, the Student record is created with various fields.

- Records in C# are immutable by default, meaning their values cannot be changed after creation. This makes them well-suited for scenarios where immutability is desired, such as representing data with value semantics.

- Attempting to modify a value in a record, like **student1.GPA = 4.0;**

would result in a compilation error due to immutability.

QUESTION(S): Compare and contrast this to writing a class for a Student (for instance, in Java). In other words, what things are easier/harder to do? Do you lose access to possibly wanted functionality when using a non-class based approach? Is there anything you can do with a record's field that could be prevented with a class's data member? Mention some differences between C# properties and Java data members. Submit the answers to these questions in the Word document created in Problem 1 in your zip file and include your working C# record type and test program

Valentina Silveira
Eagle ID: 901365377

Answer:

Comparison to Writing a Class (in Java):

- Easier Initialization: Records in C# provide a syntax for creating immutable data types. In Java, creating an equivalent class would require more boilerplate code for constructor, getters, setters, and equals/hashCode methods.

- Immutability by Default: In C#, records are immutable by default, which can help ensure data integrity. In Java, immutability often requires explicit effort and additional methods.

- Value-Based Equality: Records in C# provide value-based equality by default, meaning two records with the same values are considered equal. In Java, you would need to override the equals and hashCode methods explicitly.

- Loss of Explicit Control: With records, you lose some explicit control over how data is accessed and modified, which may be a drawback in certain situations where you need fine-grained control over getter/setter behavior.

Differences between C# Properties and Java Data Members:

- Properties in C#: In C#, properties provide a way to confine fields and expose them through getter and setter methods. Properties allow additional logic to be added to the getter or setter if needed.

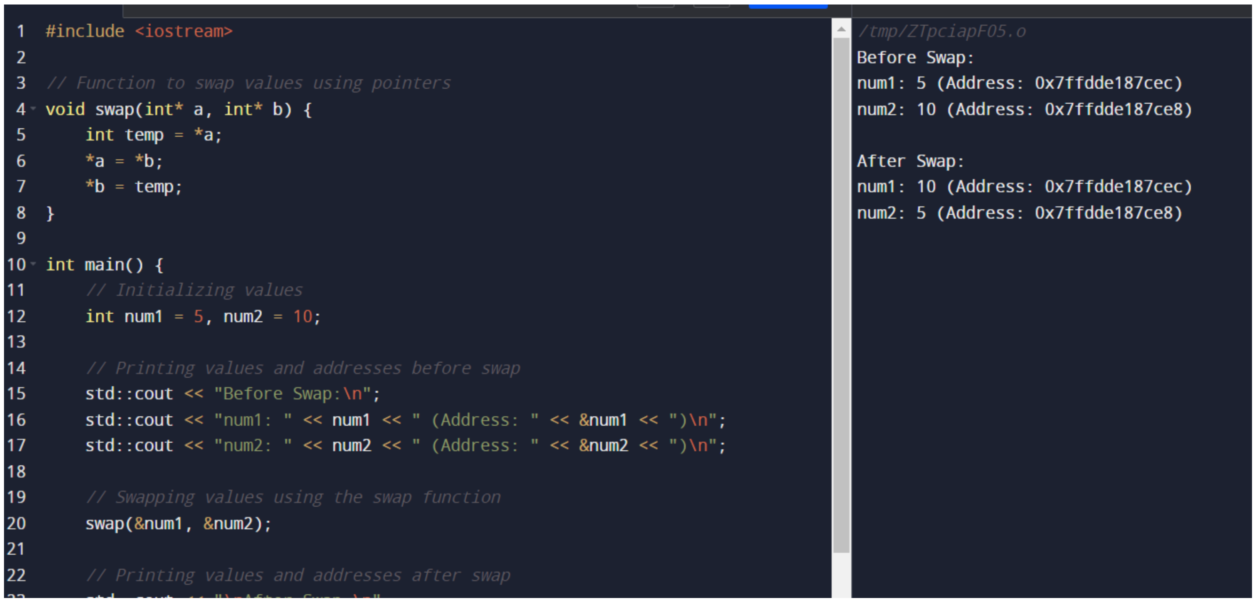
- Java Data Members: In Java, fields (data members) in a class are typically accessed directly. In recent versions of Java, there's been a move towards using getter and setter methods for better encapsulation and control.

Valentina Silveira
Eagle ID: 901365377

4. C++ pointers –

Write a C++ program that implements a swap() method that takes two int pointers as parameters. Using these passed pointers, swap the two parameter values. Make sure this method is defined and implemented before the main method to avoid compilation issues.
In the main method, populate the values and prove that these values were actually swapped by printing their addresses and values before and after the swap with some descriptive text.
Submit your working C++ test program in your zip file.

Answer:



```cpp
#include <iostream>

// Function to swap values using pointers
void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main() {
    // Initializing values
    int num1 = 5, num2 = 10;

    // Printing values and addresses before swap
    std::cout << "Before Swap:\n";
    std::cout << "num1: " << num1 << " (Address: " << &num1 << ")\n";
    std::cout << "num2: " << num2 << " (Address: " << &num2 << ")\n";

    // Swapping values using the swap function
    swap(&num1, &num2);

    // Printing values and addresses after swap
```

```
/tmp/ZTpciapF05.o
Before Swap:
num1: 5 (Address: 0x7ffdde187cec)
num2: 10 (Address: 0x7ffdde187ce8)

After Swap:
num1: 10 (Address: 0x7ffdde187cec)
num2: 5 (Address: 0x7ffdde187ce8)
```

In this program:

- The swap() function takes two integer pointers as parameters and swaps the values they point to.

- In the main function, two integers num1 and num2 are initialized with values.

- The addresses and values of num1 and num2 are printed before the swap.

- The swap() function is called with the addresses of num1 and num2 as arguments to swap their values.

- The addresses and values are printed again after the swap to demonstrate that the values have been swapped.