(There will be many additional questions similar to the quizzes.)

Question 1. (5 points) What does the main method of MyClass Tester print? ( )

A. true

B. false

A screenshot of a computer code

Description automatically generated

Question 2. (5 points) Consider the following class definitions:

A group of white text boxes

Description automatically generated with medium confidence

A screenshot of a test

Description automatically generated

Question 3. (5 points) Select the correct memory model for this code ( )

A diagram of a computer program

Description automatically generated with medium confidence

Question 4. (5 points) Assume you have a Document object, d, that stores the following text: "lalalaaaa! lala, la!".

What array does the following call to getTokens return?

A black text on a white background

Description automatically generated

Question 5. (5 points) Which of the following lines of code correctly assign a String containing the text "My String" to the variable 'text'? ( ) (Select all correct options.)

A group of text boxes

Description automatically generated

Question 6. (5 points) Assume you have a String variable s that stores the text

"%one%%two%%%three%%%%"

Which of the following calls to s.split will return the String array as follows:

["%", "%%", "%%%", "%%%%"] ( ) (Select all correct options.)

A group of black text boxes

Description automatically generated

A screenshot of a computer program

Description automatically generated

B. The regex pattern [^, ]+ matches one or more characters that are not commas or spaces. Let's break down this pattern and explain its components:

* [^, ] : Defines a negated character set
* ^ inside the brackets means "not"
* , and space are the characters being negated
* : Quantifier meaning "one or more"

Matching Behavior

This pattern will:

* Match any sequence of characters that does not contain commas or spaces
* Require at least one character to be present
* Match as many characters as possible (greedy matching)

Examples

**Matches:**

* "Hello"
* "123"
* "abc\_def"
* "!@#$%^&\*()\_+"

**Partial Matches in:**

* "Hello, world" (matches "Hello" and "world" separately)
* "a b,c" (matches "a", "b", and "c" separately)

**Non-Matches:**

* " " (single space)
* "," (single comma)
* "" (empty string, doesn't satisfy the "one or more" requirement)

Usage Tips

1. **CSV Parsing**: This pattern is particularly useful for parsing comma-separated values (CSV) data where fields might contain spaces.
2. **Data Cleaning**: It can be used to extract contiguous blocks of non-space, non-comma characters from a string.
3. **Tokenization**: The pattern can help in breaking down a string into tokens, ignoring spaces and commas as separators.
4. **Validation**: It can be used to ensure that input doesn't contain spaces or commas.

Practical Applications

1. Parsing CSV-like data with potential spaces in fields:

text

Input: "John Doe,30,New York, NY"

Matches: "John", "Doe", "30", "New", "York", "NY"

1. Extracting words or identifiers:

text

Input: "variable1, variable2, variable3"

Matches: "variable1", "variable2", "variable3"

1. Cleaning data by removing commas and spaces:

text

Input: "Hello, world! How are you?"

Matches: "Hello", "world!", "How", "are", "you?

C. The regex pattern "[a-z()0-9]+" matches one or more characters that are lowercase letters, digits, or parentheses. Let's break it down:

* "[a-z()0-9]+" is the full regex pattern
* The square brackets [] define a character set, matching any single character inside
* a-z matches any lowercase letter from a to z
* () matches literal parentheses characters
* 0-9 matches any digit from 0 to 9
* The + quantifier means "one or more" of the preceding character set

The regex pattern [^,]+ matches one or more characters that are not commas. Let's break down this pattern:

- [^,] defines a negated character set

- ^ inside the square brackets means "not"

- , is the character being negated

- + is a quantifier meaning "one or more"

## Matching Behavior

This pattern will match:

- Any sequence of characters that does not contain a comma

- The match will be as long as possible (greedy matching)

- Multiple matches can occur in a string if separated by commas

## Examples

\*\*Matches:\*\*

- "Hello world"

- "123"

- "a b c"

- "!@#$%^&\*()\_+"

\*\*Partial Matches in:\*\*

- "abc,def" (matches "abc" and "def" separately)

- "1,2,3,4" (matches "1", "2", "3", and "4" separately)

\*\*Non-Matches:\*\*

- "" (empty string, doesn't satisfy the "one or more" requirement)

- "," (single comma)

## Usage Tips

- This pattern is often used to parse comma-separated values (CSV) data

- It can extract individual fields from a comma-delimited string

- To match the entire string without commas, use anchors: ^[^,]+$

- For case-insensitive matching, add the i flag (if supported by your regex engine)

## Practical Applications

1. Parsing CSV data:

Input: "John,Doe,30,New York"

Matches: "John", "Doe", "30", "New York"

2. Extracting non-comma content:

Input: "Hello, world! How are you?"

Matches: "Hello", " world! How are you?"

3. Validating comma-free input:

Pattern: ^[^,]+$

Valid: "NoCommasHere"

Invalid: "This,Has,Commas"

The regex pattern ^[^,]+$ is designed to match an entire string that does not contain any commas. Let's break down this pattern and explain its components:

Pattern Explanation

* ^ : Anchors the match to the start of the string
* [^,] : Matches any single character that is not a comma
* + : Matches one or more of the preceding character set
* $ : Anchors the match to the end of the string

Matching Behavior

This pattern will:

* Match an entire string from start to end
* Accept any characters except commas
* Require at least one character to be present

Examples

**Matches:**

* "Hello world"
* "123456789"
* "This is a sentence without commas"
* "!@#$%^&\*()\_+"

**Non-Matches:**

* "Hello, world" (contains a comma)
* "Line 1,Line 2" (contains a comma)
* "" (empty string, doesn't satisfy the "one or more" requirement)

The regular expression [a-z]+|[()0-9]+ is designed to match two different patterns. Let's break it down and explain its components:

Pattern Breakdown

1. [a-z]+: This part matches one or more lowercase letters from a to z.
2. |: This vertical bar acts as an "OR" operator, separating the two patterns.
3. [()0-9]+: This section matches one or more characters that are either:
   * An opening parenthesis (
   * A closing parenthesis )
   * Any digit from 0 to 9

Behavior

This regex will match strings that consist of either:

* A sequence of one or more lowercase letters  
  OR
* A sequence of one or more parentheses and/or digits

Examples of Matching Strings

* abc (matches the first pattern)
* hello (matches the first pattern)
* 123 (matches the second pattern)
* (456) (matches the second pattern)
* ()()9 (matches the second pattern)

Non-Matching Examples

* ABC (uppercase letters not allowed)
* abc123 (mixed letters and numbers not allowed in a single match)
* a(1)b (mixed pattern not allowed)

Usage

This regex is useful for separating text into two distinct categories:

1. Words consisting of lowercase letters
2. Numeric or parenthetical expressions

It could be applied in various scenarios, such as:

* Parsing simple mathematical expressions
* Separating words from numeric identifiers in a text
* Identifying and categorizing different types of tokens in a string

A screenshot of a computer code

Description automatically generated

A screenshot of a white sheet with black text

Description automatically generated