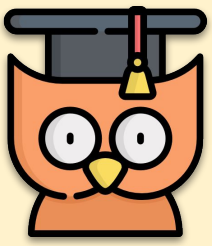
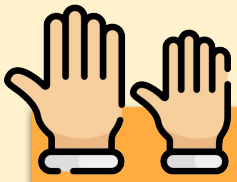
Lesson 3 - Regular Expressions



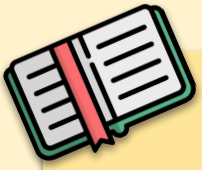
Learning Outcomes:

- What are regular expressions and how to use them



REMEMBER:

Put up your hand. We love to help!



Regular Expressions

Regular expressions, or regex for short, are used to find specific patterns in text. It does this by using a special set of characters to indicate what you want to match. These expressions can be simple, but they are often quite long and complicated and can be difficult to understand. We will learn the basics of writing regexes in this lesson. We will begin by looking at this regex

"^[a-z0-9._%+-]+@[a-z0-9.-]+\\.[a-z]{2,}$" or the simplified version

"[a-z0-9._%+-]+@[a-z0-9.-]+.[a-z]{2,}"

Matches function

The matches function returns a boolean and tells you if the string you are looking at matches the string you are searching for.

Below are two example pieces of code. Note that a lowercase t is not the same as an uppercase T.

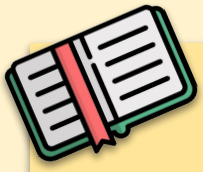
```
String sentence = "The";
println(sentence.matches("The"));
//print true
```

```
String sentence = "The";
println(sentence.matches("the"));
//print true
```

But what if we want to match for any three letter word. We can use the “.” symbol to indicate any character. If we use three dots we will match any three letter word. We can also change any of the dots to match three letter words that have specific characters. Try the two examples below and change the word.

```
String sentence = "The";
println(sentence.matches("..."));
//print true
```

```
String sentence = "The";
println(sentence.matches("T.."));
//print true
```



Regexes

- What about if we only want to match strings that start with a specific set of characters, but don't care how long it is? To do that we use the "+" symbol. This allows there to be more than one of the preceding character and still match.

```
String sentence = "The";
println(sentence.matches("T.+"));
//print true
```

- We can also tell it how many to match. We do this using the {} symbols, with a number. This will first match the T and then match two of any character.

```
String sentence = "The";
println(sentence.matches("T.{2}"));
//print true
```

- This can be changed to match a range of characters after the T, between 2 and 5 characters.

```
String sentence = "They";
println(sentence.matches("T.{2,5}"));
//print true
```

- If we want to match a range of characters we can simply write the characters between square brackets []. This will match any three letter word that only contains vowels.

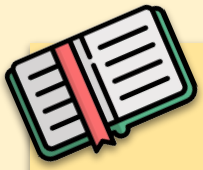
```
String sentence = "aee";
println(sentence.matches("[aeoiu]{3}"));
//print true
```

- We can do this in another way if the characters we want to match are in order. The example below will match any three letter word that only contains lowercase letters.

```
String sentence = "the";
println(sentence.matches("[a-z]{3}"));
//print true
```

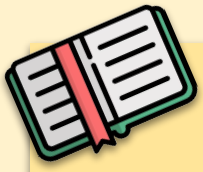
- We can change this to also match with uppercase letters, spaces and any length of a string.

```
String sentence = "the park is closed";
println(sentence.matches("[a-zA-Z ]+"));
//print true
```



Challenges

1. Write a regex to match the words [“the”, “are”, “use”, “one”], but not the words [“there”, “they”, “I”, “we”]
2. Write a regex to match the words ["the", "there", "they", "two"], but not the words ["are", "use", "I", "we"]
3. Write a regex to match the words ["had", "bad", "hid", "add"], but not the words ["academy", "two", "I", "we"]
4. Write a regex to match the words ["aeo", "iou", "eee", "aee"], but not the words ["ate", "eat", "hid", "aeee"]
5. Write a regex to match the words ["hat", "cat", "cut", "about"], but not the words ["ate", "eaten", "hid", "well"]
6. Write a regex to match the words ["wood", "wool", "loop", "pool"], but not the words ["cold", "old", "told", "well"]
7. Write a regex to match the words ["wood", "woolen", "would", "woken"], but not the words ["woolenly", "old", "told", "wad"]
8. Write a regex to match the words ["running", "walking", "swimming", "typing"], but not the words ["tag", "convince", "overtake", "pong"]
9. Write a regex to match the words ["abc", "def", "ghi", "jkl"], but not the words ["mno", "pqr", "stu", "vwx"]
10. Write a regex to match the words ["two", "three", "four", "five"], but not the words ["six", "seven", "eight", "nine"]
11. Write a regex to match the words ["beeps", "sheeps", "bleps", "shephard"], but not the words ["box", "seed", "drops", "bottle"]
12. Write a regex to match the words ["Observe", "Promise", "Wait", "Invest"], but not the words ["generate", "examine", "discuss" ,"choose"]



Regexes

- We can also use the ^ to indicate the characters that we don't want to match. The example below will match anything that doesn't contain lowercase letters.

```
String sentence = "123";
println(sentence.matches("[^a-z]+"));
//print true
```

- There are several special characters that you can use in a regex that are shorthand for things we are already able to do. The first of these is "\\d". This matches with any digit.

```
String sentence = "123";
println(sentence.matches("\\d{3}"));
//print true
```

- If we capitalise the letter it will do the opposite.

```
String sentence = "£$l";
println(sentence.matches("\\D{3}"));
//print true
```

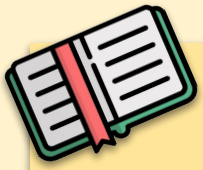
- "\\s" will match with whitespace. "\\w" will match with word characters, which seems to include letters and digits.
- Previously we used the "+" symbol to match with strings of any length. In the example below we want to know if the word starts with the letter a. This works fine if there are at least two letters, like in the code below. Unfortunately the "+" symbol requires that there is at least one character matching it.

```
String sentence = "at";
println(sentence.matches("a.+"));
//print true
```

- To fix this we use the "*" symbol. This doesn't require there to be an additional character to match with the ".".

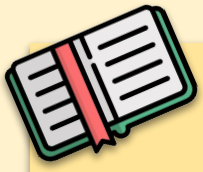
```
String sentence = "a";
println(sentence.matches("a.*"));
//print true
```

```
String sentence = "a";
println(sentence.matches("a.+"));
//print false
```



Challenges

1. Write a regex to match the words [“oo”, “wool”, “loop”, “pool”], but not the words [“cold”, “old”, “told”, “well”]
2. Write a regex to match the words [“1234”, “0000”, “4312”, “7777”], but not the words [“123o”, “12345”, “000”, “100.”]
3. Write a regex to match the words [“had”, “bad!”, “hide.”, “a”], but not the words [“academy10”, “2”, “10.”, “well!123”]
4. Write a regex to match the words [“the”, “t”, “they”, “there”], but not the words [“ate”, “eat”, “hid”, “aeee”]
5. Write a regex to match the words [“hid”, “riddance”, “hiding”, “harriett”], but not the words [“ate”, “eaten”, “absent”, “well”]
6. Write a regex to match the words [“102A”, “900D”, “600H”, “555T”], but not the words [“102a”, “old”, “1000DS”, “H500”]



Regexes

- In the examples we have looked at so far, the case (upper or lower) have been quite important, but we don't care about the case. To ignore the case we can use "(?i)" in our regex. You can see this in the example below. There are other commands that begin with ?, but for the moment we don't need them.

```
String sentence = "Monday";
println(sentence.matches("(?i)monday"));
//prints true
```

- Sometimes we have two possible regexes that we want to match. We can do this using the or symbol (|). The example code will match with either the word bread or banana.

```
String sentence = "bread";
println(sentence.matches("b(read|anana)"));
//prints true
```

- The last concept we are going to learn about in this section is called capture groups. The syntax for capture groups is the open and close brackets (). We are going to learn about capture groups by looking at a set of examples.
- In this example we are looking for the pattern "go" repeated over and over again. We capture the pattern "go" and then the + indicates that we are looking for a repetition of it.

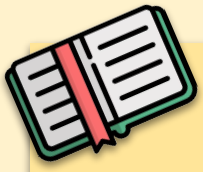
```
String sentence = "gogogo";
println(sentence.matches("(go)+"));
//prints true
```

- We can also reference this capture group later in the regex. This example will match the pattern "the" later in the regex.

```
String sentence = "the the";
println(sentence.matches("(the) \\1"));
//prints true
```

- We can also use the capture group to find a duplicate of a symbol. This example will match with any 4 letter word in which the middle two letters are the same.

```
String sentence = "good";
println(sentence.matches(".(.)\\1."));
//prints true
```



replaceAll

- So far we have only looked at matching text, but often once we match the text we want to change it. We can do this using the replaceAll function. The example below shows us replacing all of the vowels with the * symbol.

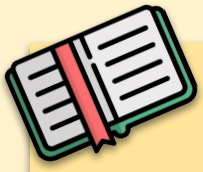
```
String sentence = "The Academy of Code";
sentence = sentence.replaceAll("[aeoiu]","*");
println(sentence);
// prints Th* Ac*d*my *f C*d*
```

- The replace all function can also be used to just remove characters. The code below find anything that is not a vowel and replaces it with nothing.

```
String sentence = "The Academy of Code";
sentence = sentence.replaceAll("[^aeoiu]","");
println(sentence);
// prints eaeooe
```

- Then we can use the string's length function to determine how many vowels were in the original string. **Note:** Be careful doing this if you need to keep the original string.

```
String sentence = "The Academy of Code";
sentence = sentence.replaceAll("[^aeoiu]","");
println(sentence);
// prints eaeooe
println(sentence.length());
// prints 6
```



Challenges

- How Many Vowels -

  Link: https://edabit.com/challenge/GBKphScsmDi9ek3ra
- Vowel Replacer - **Hint:** You will need to convert the char into a string.

  Link: https://edabit.com/challenge/iW7rtor54mbFQ2RrZ
- Owofied a Sentence - **Hint:** Look back at the strings and arrays lesson.

  Link: https://edabit.com/challenge/nuKniCXYbaCfrmjgX
- Remove Every Vowel from a String -

  Link: https://edabit.com/challenge/oMCKfdMqgt9kxqA2M
- Hashes and Pluses -

  Link: https://edabit.com/challenge/s8RHRY9hfmvYMuaeC
- Transforming Words into Binary Strings

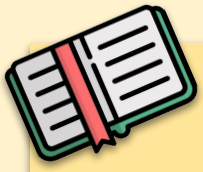
  Link: https://edabit.com/challenge/jwzMsyo2tbgn2KbGQ
- Letters Only -

  Link: https://edabit.com/challenge/HPcr7REWMLTosoXME
- ATM PIN Code Validation - **Hint:** You can use the | (or) symbol to help solve this, so try break the problem into parts and solve each part.

  Link: https://edabit.com/challenge/bL2E8p5DGWSNmEtAE
- Valid Hex Code - **Note:** Marked as Medium difficulty.

  Link: https://edabit.com/challenge/9zBJYnBekqAo52zEp

https://javascript.info/regexp-groups
https://blog.bam.tech/developer-news/getting-better-at-regular-expressions-with-3-games
https://regexcrossword.com/
https://gnosis.cx/publish/programming/regular_expressions.html



Additional resources

- In the this lesson we have covered the basics of how a regex works. There is a lot more to learn about how to use regexes, but now that you know the basics you are better equipped to learn the more about the more complicated parts on your own.
- Below are some helpful links I found, remember Google is always a good option for learning new material.
- https://www.tutorialspoint.com/java/java_regular_expressions.htm
- https://regex101.com/r/cO8lqs/2
- https://medium.com/factory-mind/regex-tutorial-a-simple-cheatsheet-by-examples-649dc1c3f285
- https://www.rexegg.com/regex-quickstart.html