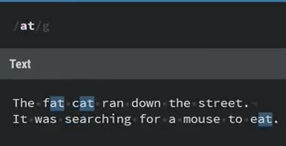
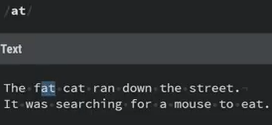
**REGULAR EXPRESSIONS**

**Regular expressions** are patterns/model, which can be used to mach string or part of the string. **Two forward slash** is used to indicate a regular expression. Dart uses JavaScript regular expression.

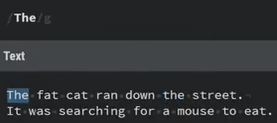
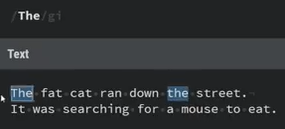
Example:

***The fat cat ran down the street. It was searching for a mouse to eat.***

* 1. **‘g’** means **global** and it is used to search all similar character or word in a sentence. like:
* /at/**g** is used to search for all ***at*** in the above statement. if g is removed, it will search for only the first ***at***.

 and .

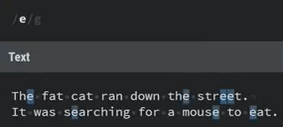
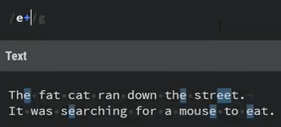
* 1. **‘I’** is for case insensitive. It is used to search string without considering if it is upper/lower case. Example when used with **global**:

 and .

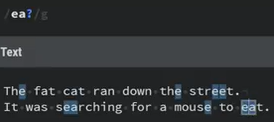
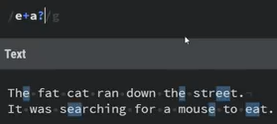
* 1. .

**SPECIAL CHARACTERS:**

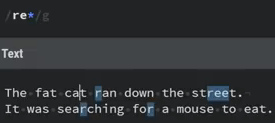
* 1. **‘+’** is use to search for one or more of the character it proceeds. Like the below images shows single **e** search and one or more **e** search. notice that I included global and how ee in street is selected in the two images.

 and  .

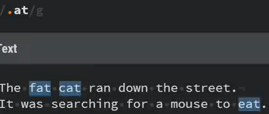
* 1. **‘?’** is use to optionally search the character it proceeds. like **/ea?/g** is used to search **‘e’** which can optionally have **‘a’** after it and **/e+a?/g** is used to find one or more **‘e’** that is optionally proceeded by **‘a’**. Example:

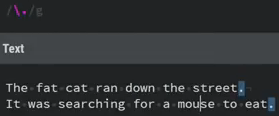
 and  .

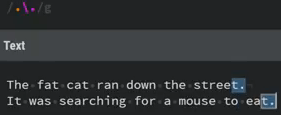
* 1. **‘\*’** is use to match zero, one or more of the character it proceeds.

. Here all **“r”** is match but attaching **“e”** to it is optional.

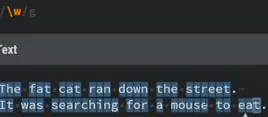
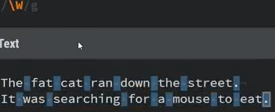
* 1. **“.”** is use to match anything except a new line.

: Here **f, c** and **e** where represented with **‘.’**.

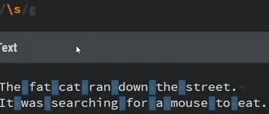
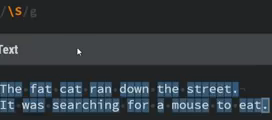
: **‘\.’** is was used to search for all the **‘.’**.

: **‘.\.’** was used to search **‘.’** and the any character before it.

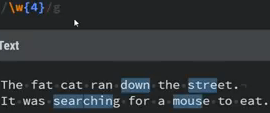
* 1. ‘\w’ is use to match all word and ‘\W’ is the opposite.

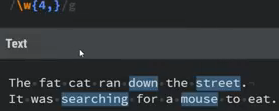
 and .

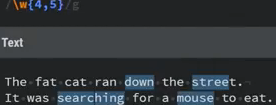
* 1. ‘\s’ is use to match space and ‘\S’ is the opposite.

 and .

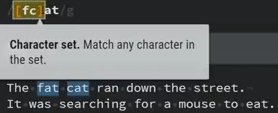
* 1. ‘\w{…}’ is use to get to get words that has a minimum of the number(s) in the curly brace.

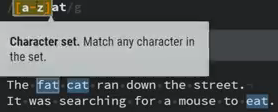
: Here 4 characters in a row was selected.

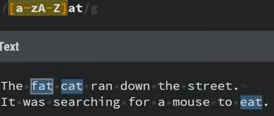
: Here 4 or more characters in a row was selected.

: Here 4 or 5 characters in a word is selected for words with a minimum of 4 or 5 characters.

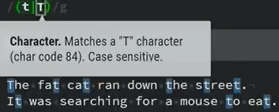
* 1. ‘[…]’ is use to select all the any of the character you include in the square bracket.

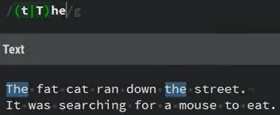
: Here **f or c** that is before **‘at’** is selected with the **‘at’** included. remember when dot was ‘.’ proceed ‘at’, eat was also selected then—check **no. 7** above.

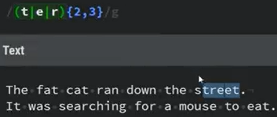
: Here all characters between **a to z** and ends with **‘at’** are selected including the **‘at’**.

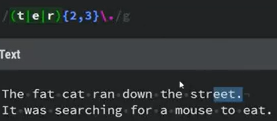
: all characters between **a to z** and **A to Z** and ends with **‘at’** are selected including the **‘at’**.

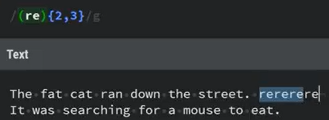
* 1. ‘(…)’ is use to represent group, thus characters inside the bracket act on eachother.

: means match t or T.

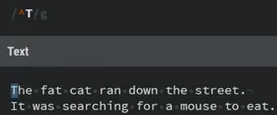
: Matches **t or T** that **he** follows with including the **he**. if **t and T** were not in **parentheses**, then it will match **t or The**.

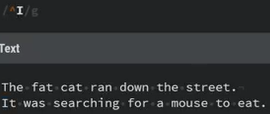
: Matches 2 or 3 characters that contains **t or e or r**. Here **tre and et** were selected because, in the case of **tre**, the characters are already completely 3, which prompt the next selection that is just **et**.

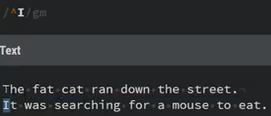
: Here notice that the first **tr** was not selected, this is because there is no **‘.’** to complete the logic, thus **‘eet.’** was selected.

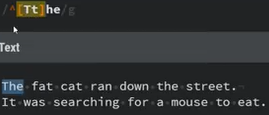
: Here re is seen as one, thus it is needed to be seen for 2 or 3 time, which was 3times in this case.

* 1. **‘^’** is use to match the beginning of a String or the beginning of the line. Unless **multiline** (m) is specified.

: it matches **‘T’** at the beginning of the line.

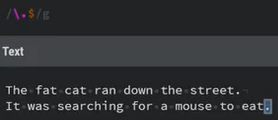
: It didn’t match **‘I’** because it only sees the first line as the beginning of the line.

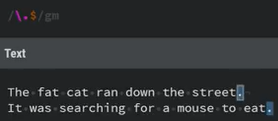
: **‘I’** is now matched because ‘m’ was included. **m** is for **multiline**.

: Notice that only the very first ‘The’ was matched although it was meant to matched **T or t** he.

From all the examples, notice that ‘^’ is used to select the same character that begins a line in a sentence or page.

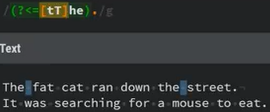
* 1. ‘$’ can be used to match the end of the line.

: Only the last ‘.’ was matched.

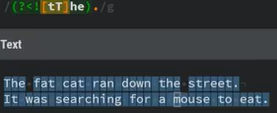
: Here multiline is enabled thus the two ‘.’ that end the two lines were matched.

* 1. **(?<=)** is a **positive look behind** used to match groups before the main expression without including the main expression in the result.

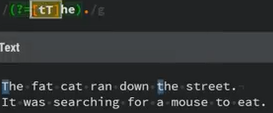


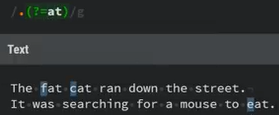
: This is used to match any character before **the or The** which is **space** in this case.

* 1. **(?<!)** is a negative look behind used to match the opposite of positive look behind.

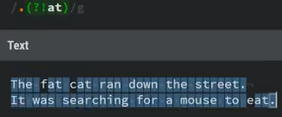
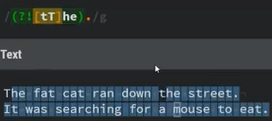
: Here all other characters except **white space** – *the character that follows* ***the or The***– was matched.

* 1. **(?=)** is the **positive look ahead** formed from removing the less then sign from positive look behind.

: This is used to select the first character starting from **the or The**, which is **t and T** in this case. If I had used **‘..’** at the back of the expression, it will select the first two characters starting from **the or The**, which is **th and Th**.

: Is used to get a character before **‘at’**.

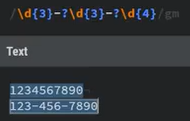
* 1. **(?!)** is the negative look ahead

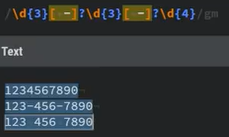
 and : As you can see, it negates positive look ahead.

* 1. **\d** is use to select **digits**, that’s from **0 to 9**.

: Was used to select all digit individually.

: was used to select a minimum and maximum of 10 digit at once.

: Was used to select 3 digits, optional -, 3 digits, optional - and 4 digits globally and in multiple lines.

: Was used to match select **3 digits**, **optional white space** or -, **3 digits**, **optional white space** or **-** and **4 digits** globally and in multiple lines.

* 1. DONE!.

**DART REGULAR EXPRESSION**

To test how efficient a particular pattern can be, use regex testing websites for it. like:

[**https://regexr.com/**](https://regexr.com/)

To use pattern in **regexp**, proceed the pattern with ‘**r**’ to make it a raw string. This will ensure that every symbol in the pattern is not seen as special character.

Example:

void main(List<String> args) {

  var words = "Amazing grace! How sweet the sound"

      "That saved a wretch like me!"

      "I once was lost, but now am found;"

      "Was blind, but now I see."

      "’Twas grace that taught my heart to fear,"

      "And grace my fears relieved;"

      'How precious did that grace appear'

      'The hour I first believed.';

  // A regExp to select all words that

  //starts with Capital letter.

  var regExp = RegExp(r"([A-Z]\w+)");

  var matches = regExp.firstMatch(words);

  print(matches?.groupNames);

  print(matches?.groupCount);

  print(matches![0]);

}

This will print out:

()

1

Amazing

Anyway, there are more to regexp, so small small you will get to be strong in using it with constant practice.