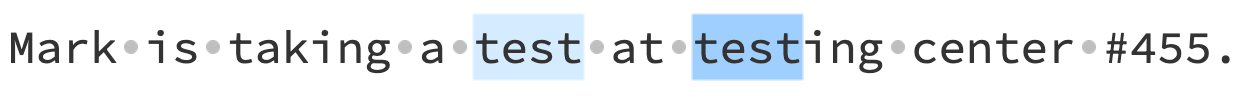
Regex cheat sheet

## Basic syntax

**test** - literally looking for the combination "test"



**[tes]** - looking for 1 of the letters "t", "e", and "s"



**[^tes]** - any letter EXCEPT "t", "e", and "s"

****

**[e-t]** - any letter BETWEEN "e" and "t"



**[0-9]** - any digit between 0 and 9



## Predefined classes

**\w** - any word character (a-z, A-Z, 0-9, \_)



**\W** - any non-word character

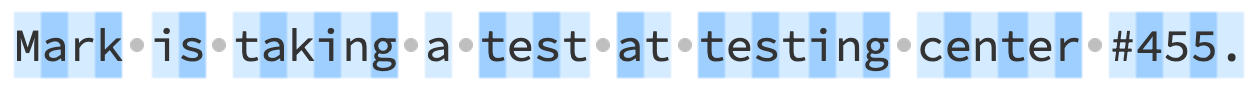


**\s** - any white-space character



^ This will match tabs, new lines, etc. too.

**\S** - any non-white-space character



**\d** - any digit



**\D** - any non-digit character



**.** - any character



**\** - escape character. Ignores the original meaning of a symbol and literally looks for it

* E.g. the symbol "." means "match any character" (literal meaning). However, if we want to literally look for a dot in the text, we write "\."



**\b** - word boundary. Matches a position where a word character (\w) is next to a non-word character (\W). This is frequently used to note the boundaries of a word, because in a word, at the start, we have a letter, preceded by a non-word character, and at the end, we have a letter, followed by a non-word character. In both cases, we can use a word boundary (\b).

| **Boundary type** | **Regex** | **Result** |
| --- | --- | --- |
| Without word boundary | test |  |
| With word boundary | \btest\b  (meaning "test" should be a separate word) |  |

**^** (caret) - start of the string (also called starting anchor). It means that the **entire string** (not just the current word) has to **start** with the character(s) written after that

| Boundary type | Regex | Result |
| --- | --- | --- |
| Without starting anchor | test | Example 1:    Example 2: |
| With starting anchor | ^test (meaning the entire string must start with "test") | Example 1:    Example 2: |

**$** (dollar) - end of the string (also called ending anchor). It means that the **entire string** (not just the current word) has to **end** with the character(s) written before that.

## Quantifiers

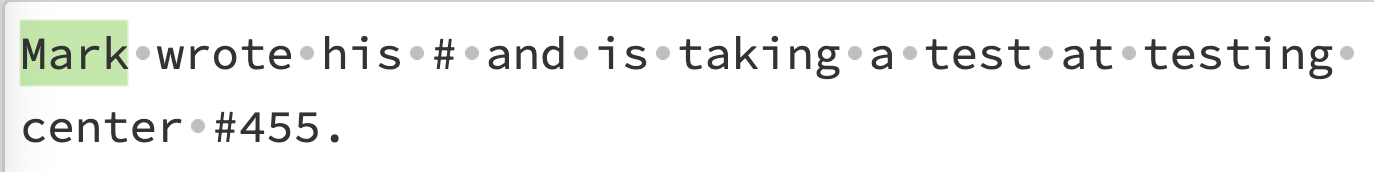
| **Quantifier** | **Meaning** | **Example** | **Example result** |
| --- | --- | --- | --- |
| + | 1 or more | #[0-9]+ |  |
| \* | 0 or more | #[0-9]\* |  |
| ? | 0 or 1 | #[0-9]? |  |
| {n} | exactly n | #[0-9]{2} |  |
| {n,} | n or more | #[0-9]{2,} |  |
| {n,m} | between n and m | #[0-9]{1,2} |  |

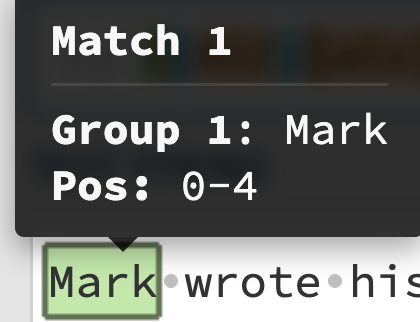
## Groups

Used to save a part of the match (e.g. the first name, the date, etc.) and access it later.

### Unnamed - (expression)

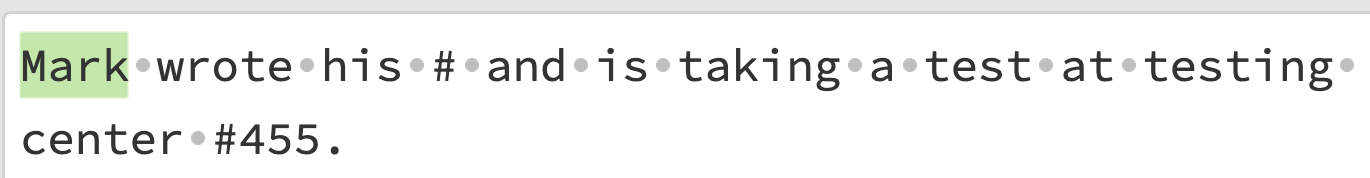
**([A-Z][a-z]+)** - match an uppercase letter, followed by 1 or more lowercase letters, and group them together (as one whole)

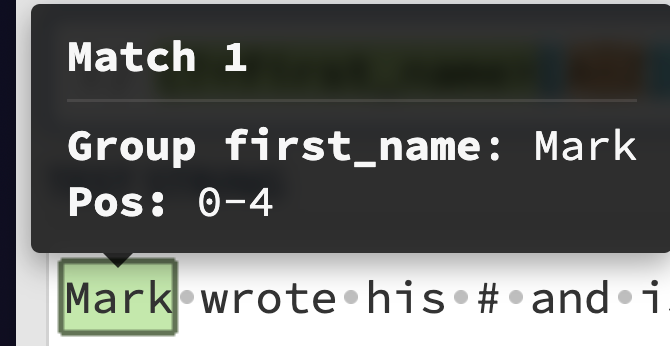




### Named - (?<name>expression)

**(?<first\_name>[A-Z][a-z]+)** - match an uppercase letter, followed by 1 or more lowercase letters, and group them together (as one whole). Save under the name "first\_name".



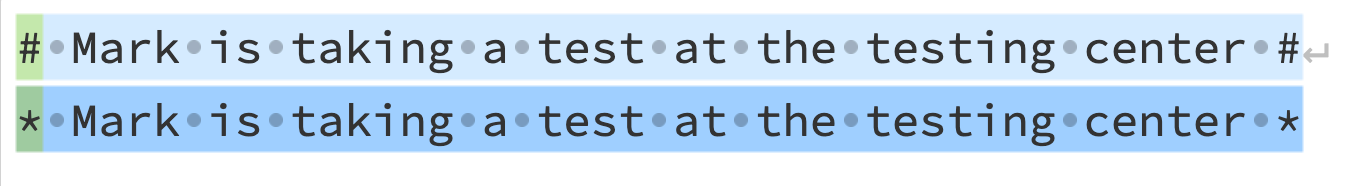


## Backreferences

Once you have a group, you can directly look for the same match further in your regex.

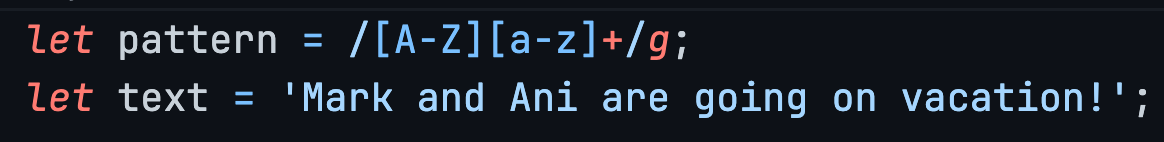
\1 - look for the same match as the one for group #1





## Regex methods

For all methods below, we'll use the following pattern and string:

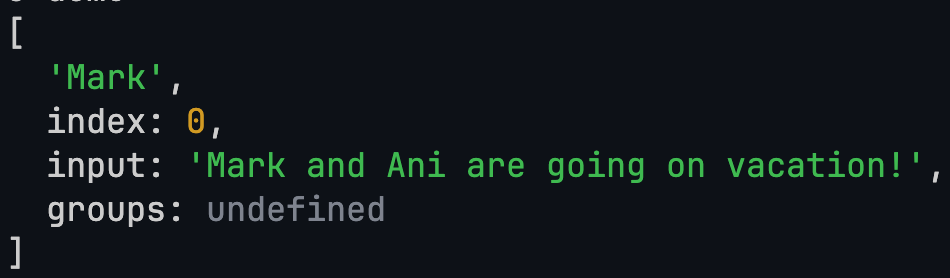


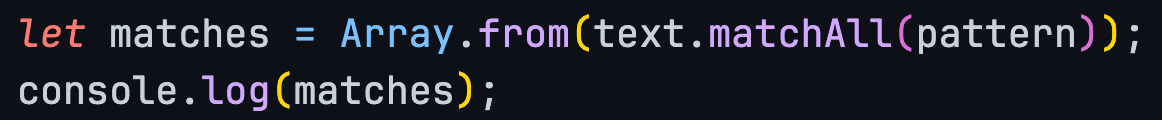
The regex looks for a word that starts with an uppercase letter and then has 1 or more lowercase letters (meaning, it detects whole words that start with an uppercase, in this example, names). We can see that there are 2 matches to that regex in our text - "Mark" and "Ani"

* **pattern.test(str)** - tests if str satisfies the requirements of pattern
  + Returns true (there **is** a match) or false (there is **no** match)
  + Sample code:  
    

Result:



* **pattern.exec(str)** 
  + Returns **detailed** info about **only 1 match at a time**
    - The matched substring itself
    - The index where the match occurred
    - The whole inputted string
    - The groups of the matched substring
  + **Sample code**:  
    
  + **Result**:  
    
  + **Note**: If you want to take the next match, you'll have to execute the .exec method again.
* **str.match(pattern)**
  + Returns **basic info** (only the matched substring) about all matches (array)
  + **Sample code**:  
    
  + **Result**:  
    

* **str.matchAll(pattern)**
  + It's the **ultimate boss** - combines the superpowers of .**exec** and .**match**
  + Returns **detailed info** about **all matches** (Regex iterator that we'll have to turn into an array)
  + **Sample code**:  
    
  + **Result**:  
    