https://regex101.com/

Your new best friend <3

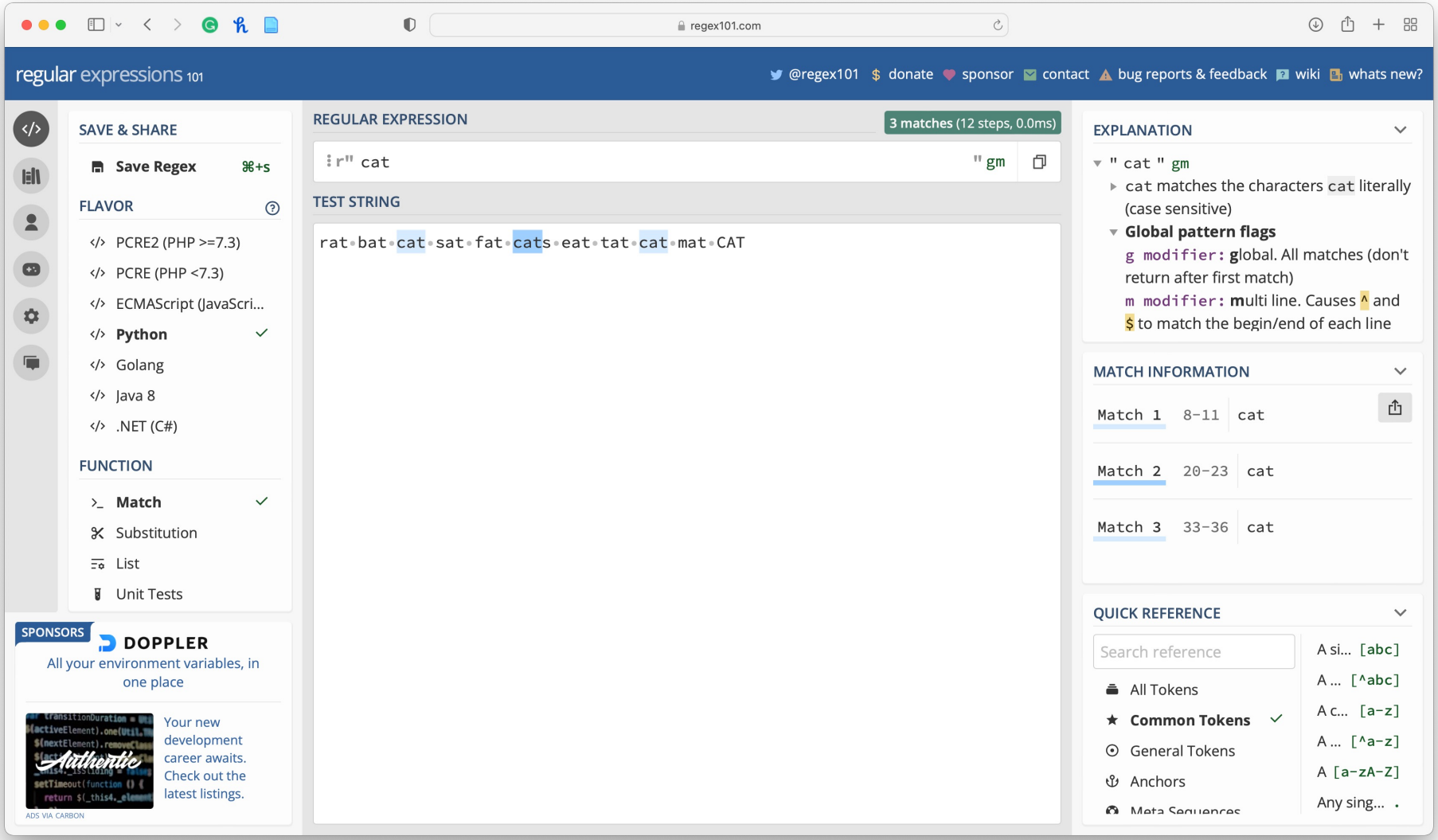
Why use regex101.com?

RegEx101 - Basics

- First, lets try to match the word `cat` in this string:

```
rat bat cat sat fat cats eat tat cat mat CAT
```

RegEx101 - Basics

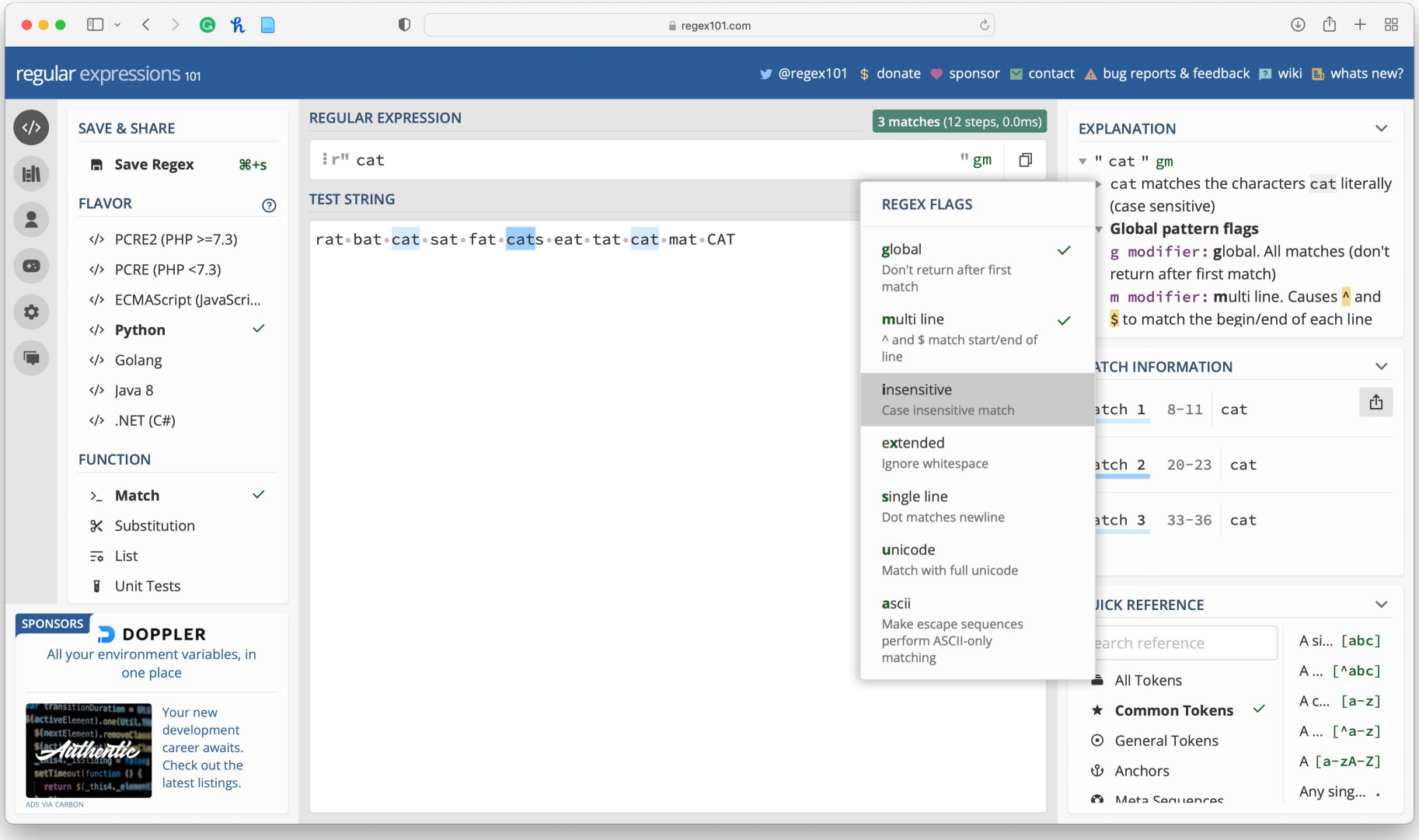


https://www.sitepoint.com/learn-regex/

RegEx101 - Flag

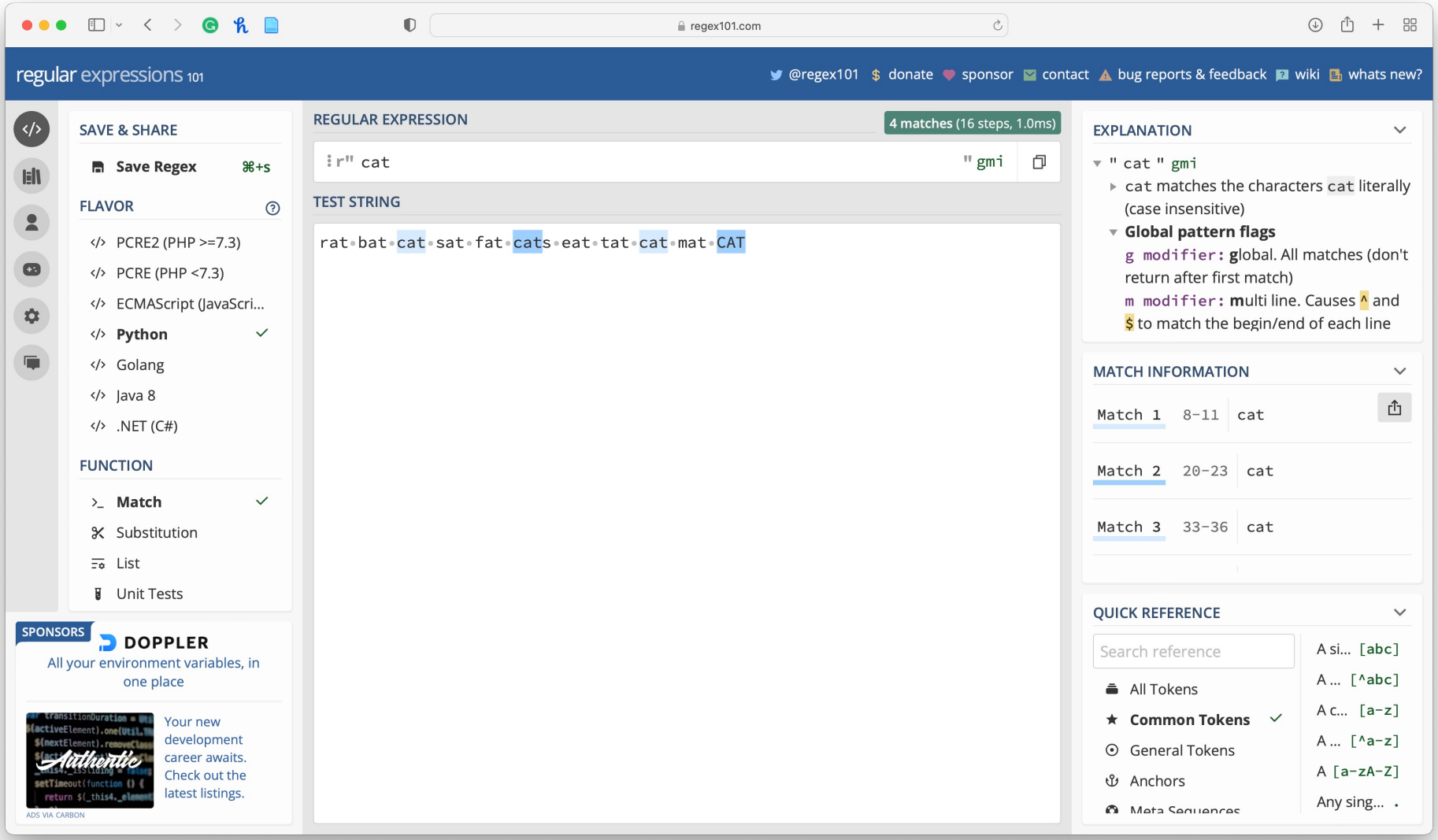
- As we can see, this is can see sensitive and will only match the word that are identical.
- We can make it insensitive by changing the flag.
- This will make it possible to match all the `cat`s ☺

RegEx101 - Flag



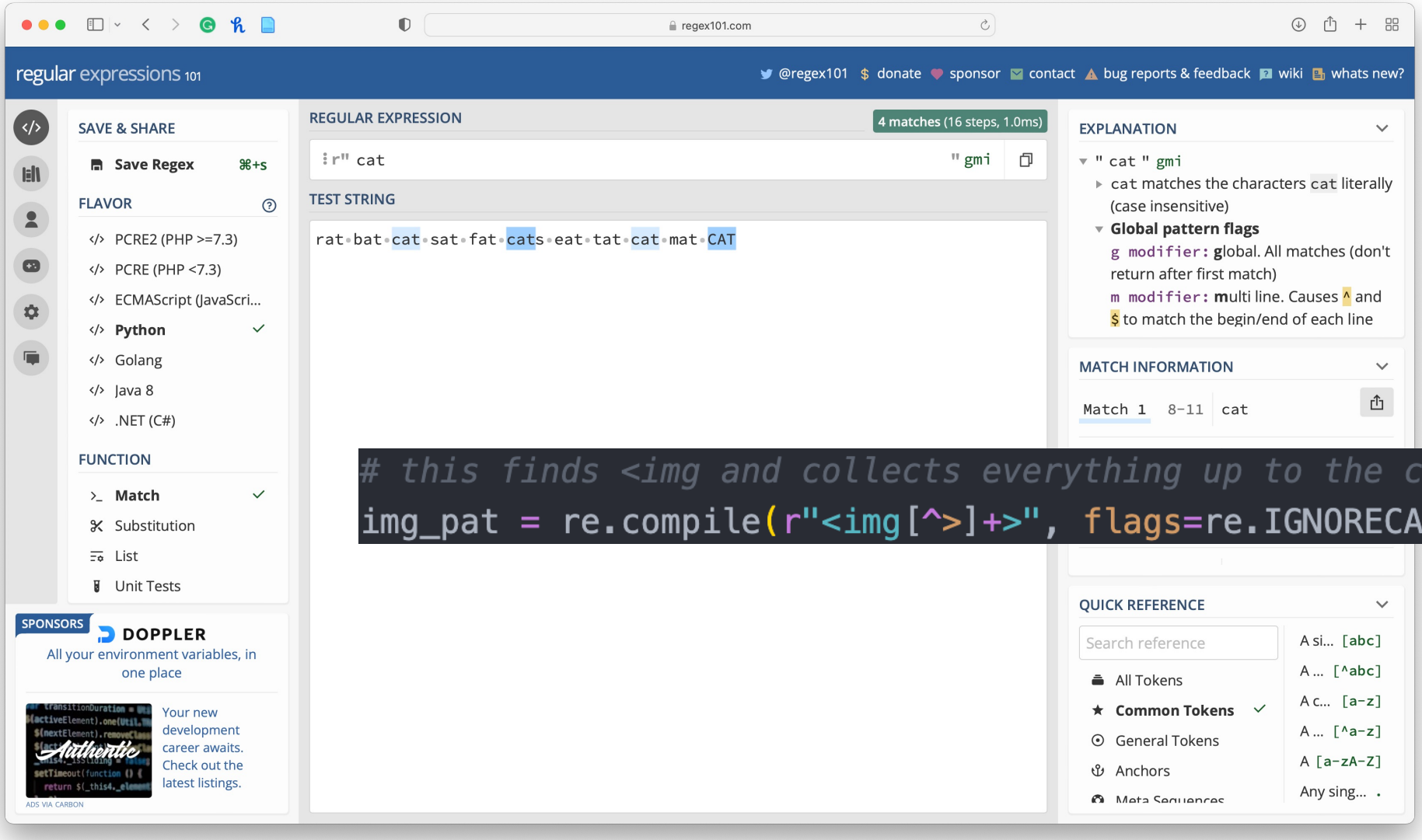
https://www.sitepoint.com/learn-regex/

RegEx101 - Flag



https://www.sitepoint.com/learn-regex/

RegEx101 – Flag (example from code)

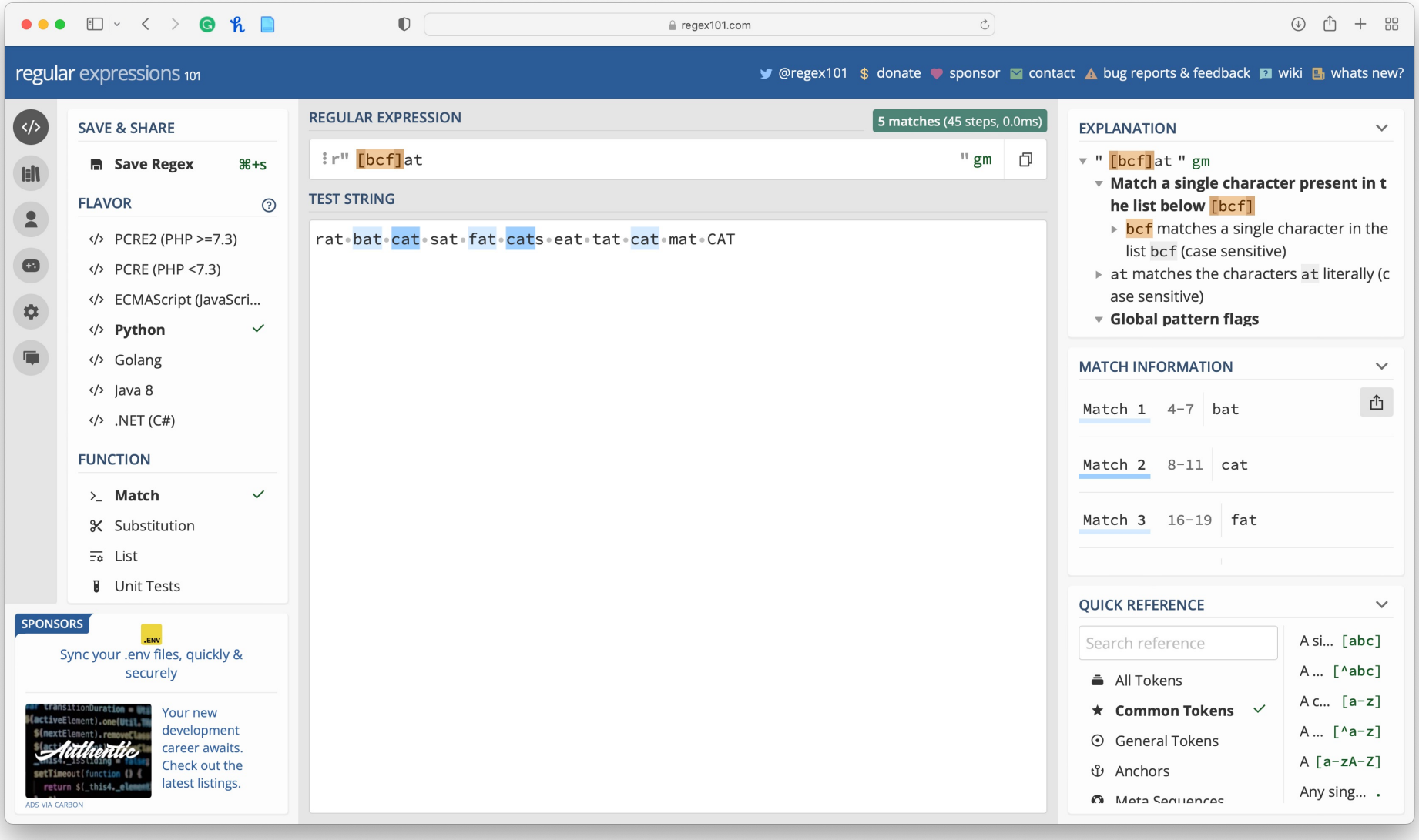


https://www.sitepoint.com/learn-regex/

RegEx101 - Character Sets

- If we wanted to match “bat”, “cat”, and “fat”, can we do this by using character sets, denoted with [] (brackets).
- Basically, you put in multiple characters that you want to get matched.
- For example, `[bcf]at` will match multiple strings as follows:

RegEx101 - Character Sets

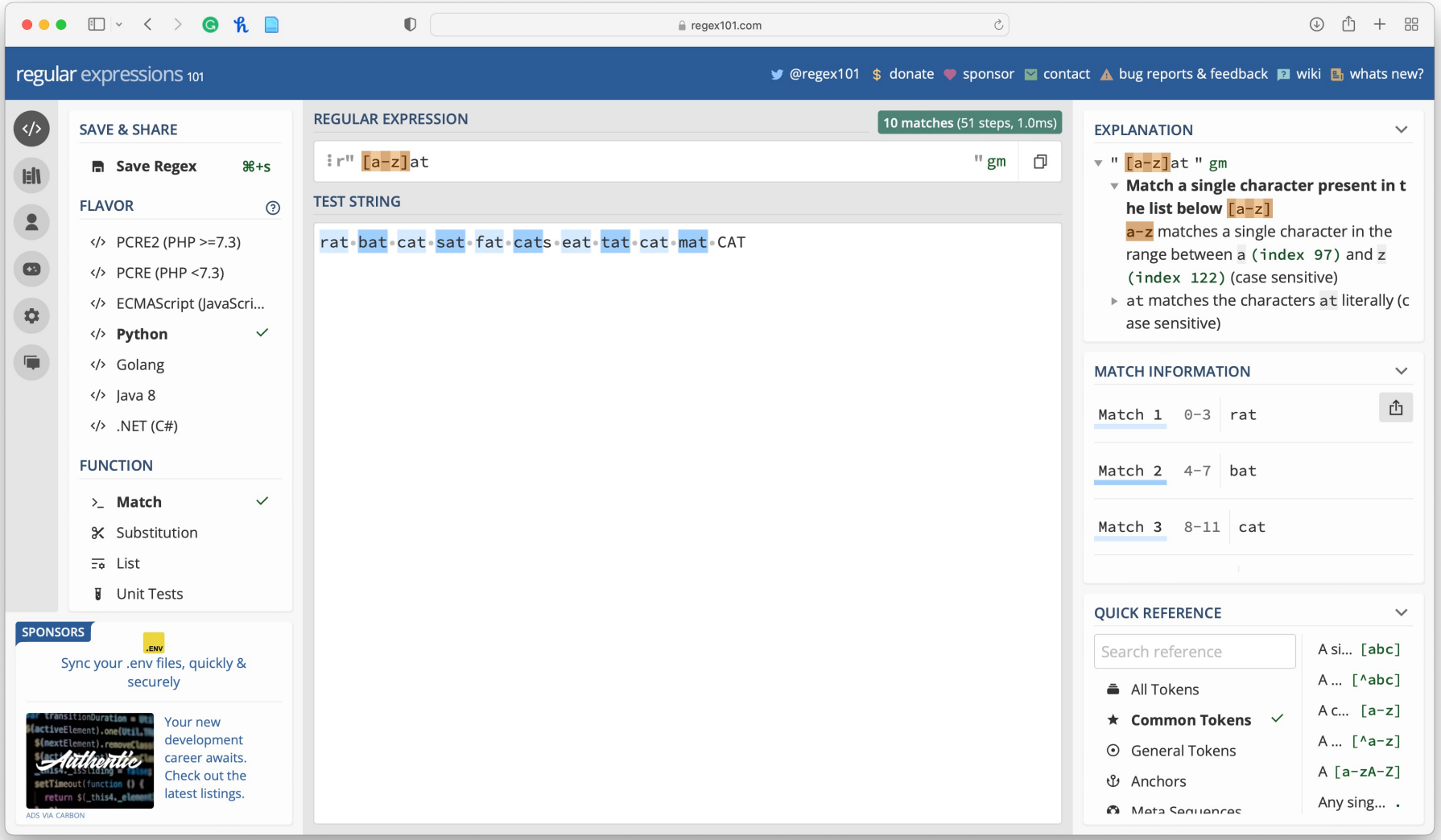


https://www.sitepoint.com/learn-regex/

RegEx101 - Ranges

- Let’s assume we want to match all words that end with at.
- We could supply the full alphabet inside the character set, but that would be tedious.
- The solution is to use ranges like this `[a-z]at`:

RegEx101 - Ranges



https://www.sitepoint.com/learn-regex/

RegEx101 - Ranges

- **Partial range**: selections such as `[a-f]` or `[g-p]`.
- **Capitalized range**: `[A-Z]`.
- **Digit range**: `[0-9]`.
- **Symbol range**: for example, `[#$%&@]`.
- **Mixed range**: for example, `[a-zA-Z0-9]` includes all digits, lower and upper case letters.

- Do note that a range only specifies multiple alternatives for a single character in a pattern.

RegEx101 - Repeating Characters

- Let’s say you’d like to match all three-letter words. You’d probably do it like this: `[a-z][a-z][a-z]`
- This would match all three-letter words.
- But what if you want to match a five- or eight-character word?
- There’s a better way to express such a pattern using the { } curly braces notation.
- All you have to do is specify the number of repeating characters.

RegEx101 - Repeating Characters

Here are examples:

- `a{5}` will match “aaaaa”.
- `n{3}` will match “nnn”.
- `[a-z]{4}` will match any four-letter word such as “door”, “room” or “book”.
- `[a-z]{6,}` will match any word with six or more letters.
- `[a-z]{8,11}` will match any word between eight and 11 letters.
- `[0-9]{11}` will match an 11-digit number.

RegEx101 - Metacharacters

- Metacharacters allow you to write regular expression patterns that are even more compact!
- `\d` matches any digit that is the same as `[0-9]`
- `\w` matches any letter, digit and underscore character
- `\s` matches a whitespace character — that is, a space or tab
- `\t` matches a tab character only

RegEx101 - Metacharacters

- Here are some exampels:
- `\w{5}` matches any five-letter word or a five-digit number
- `\d{11}` matches an 11-digit number

RegEx101 - Special Characters

- Special characters take us a step further into writing more advanced pattern expressions:

`+` : One or more quantifiers (preceding character must exist and can be optionally duplicated).

For example, the expression `c+at` will match “cat”, “ccat” and “ccccccccat”. You can repeat the preceding character as many times as you like and you’ll still get a match.

`?` : Zero or one quantifier (preceding character is optional).

For example, the expression `c?at` will only match “cat” or “at”.

RegEx101 - Special Characters

`*` : Zero or more quantifier (preceding character is optional and can be optionally duplicated).

For example, the expression `c*at` will match “at”, “cat” and “ccccccat”.

It’s like the combination of + and `?`.

`\` : this “escape character” is used when we want to use a special character literally.

For example, `c\*` will exactly match “c*” and not “ccccccc”.

RegEx101 - Special Characters

`[^]`: this “negate” notation is used to indicate a character that should not be matched within a range.

For example, the expression `b[^a-c]ld` will not match “bald” or “bbld” because the second letters a to c are negative.

However, the pattern will match “beld”, “bild”, “bold” and so forth.

. : this “do” notation will match any digit, letter or symbol except newline.

For example, `.{8}` will match a an eight-character password consisting of letters, numbers and symbols. for example, “password” and “P@ssw0rd” will both match.

RegEx101 - Special Characters

- From what we've learned so far, we can create an interesting variety of compact but powerful regular expressions.
- For example:

  `.+` matches one or an unlimited number of characters. For example, "c" , "cc" and "bcd#.670" will all match.

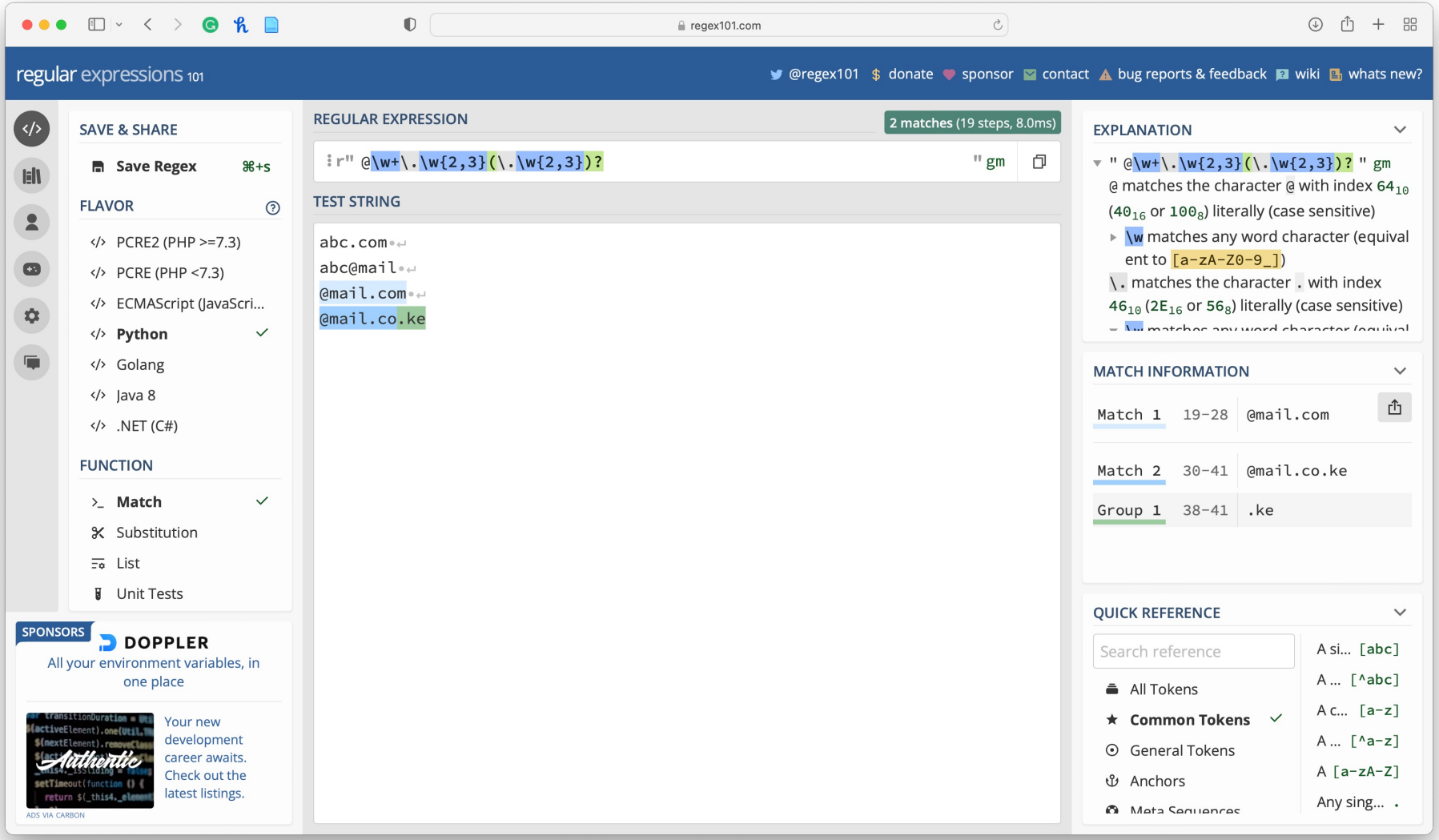
  `[a-z]+` will match all lowercase letter words irrespective of length, as long as they contain at least one letter. For example, "book" and "boardroom" will both match.

https://www.sitepoint.com/learn-regex/

RegEx101 - Groups

- All the special characters we just mentioned only affect a single character or a range set.
- What if we wanted the effect to apply to a section of the expression?
- We can do this by creating groups using round brackets — `()`.
- For example, the pattern `book(.com)?`
  - will match both “book” and “book.com”, since we’ve made the “.com” part optional.
- Here’s a more complex example that would be used in a realistic scenario such as email validation:

RegEx101 - Groups



https://www.sitepoint.com/learn-regex/

RegEx101 - Alternate Characters

- We can specify alternate characters using the “pipe” symbol: |
- This is different from the special characters we showed earlier, as it affects all the characters on each side of the pipe symbol.
- For example, the pattern `sat|sit` will match both “sat” and “sit” strings.
- We can rewrite the pattern as `s(a|i)t` to match the same strings.

RegEx101 - Starting and Ending Patterns

- You may have noticed that some positive matches are a result of partial matching.
- For example, if I wrote a pattern to match the string “boo”, the string “book” will get a positive match as well, despite not being an exact match.
- To remedy this, we’ll use the following notations:

RegEx101 - Starting and Ending Patterns

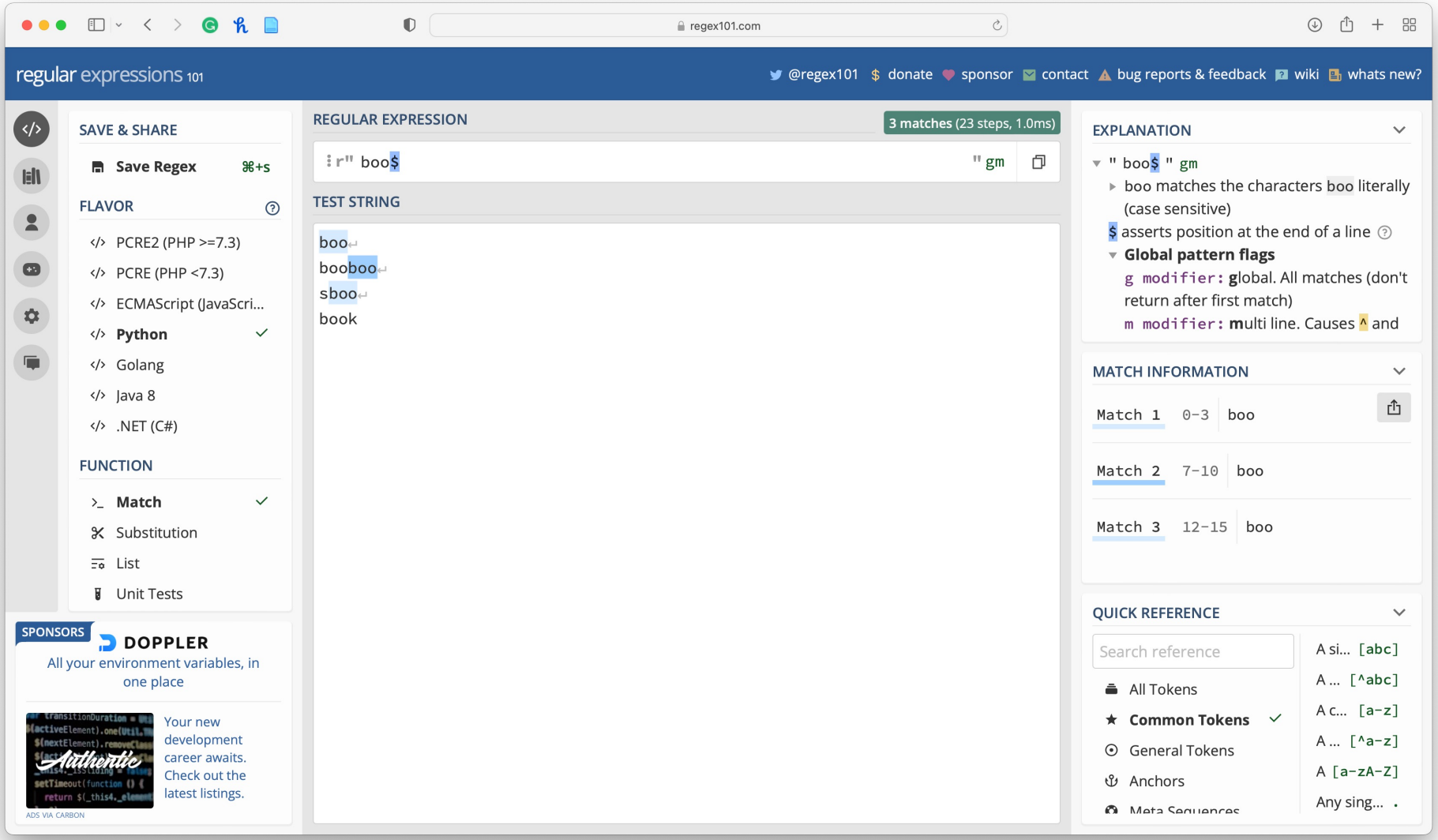
^ : placed at the start, this character matches a pattern at the start of a string.

$: placed at the end, this character matches a pattern at the end of the string.

RegEx101 - Starting and Ending Patterns

- To fix the above situation, we can write our pattern as boo$.
- This will ensure that the last three characters match the pattern.
- However, there’s one problem we haven’t considered yet, as the following image shows:

RegEx101 - Starting and Ending Patterns

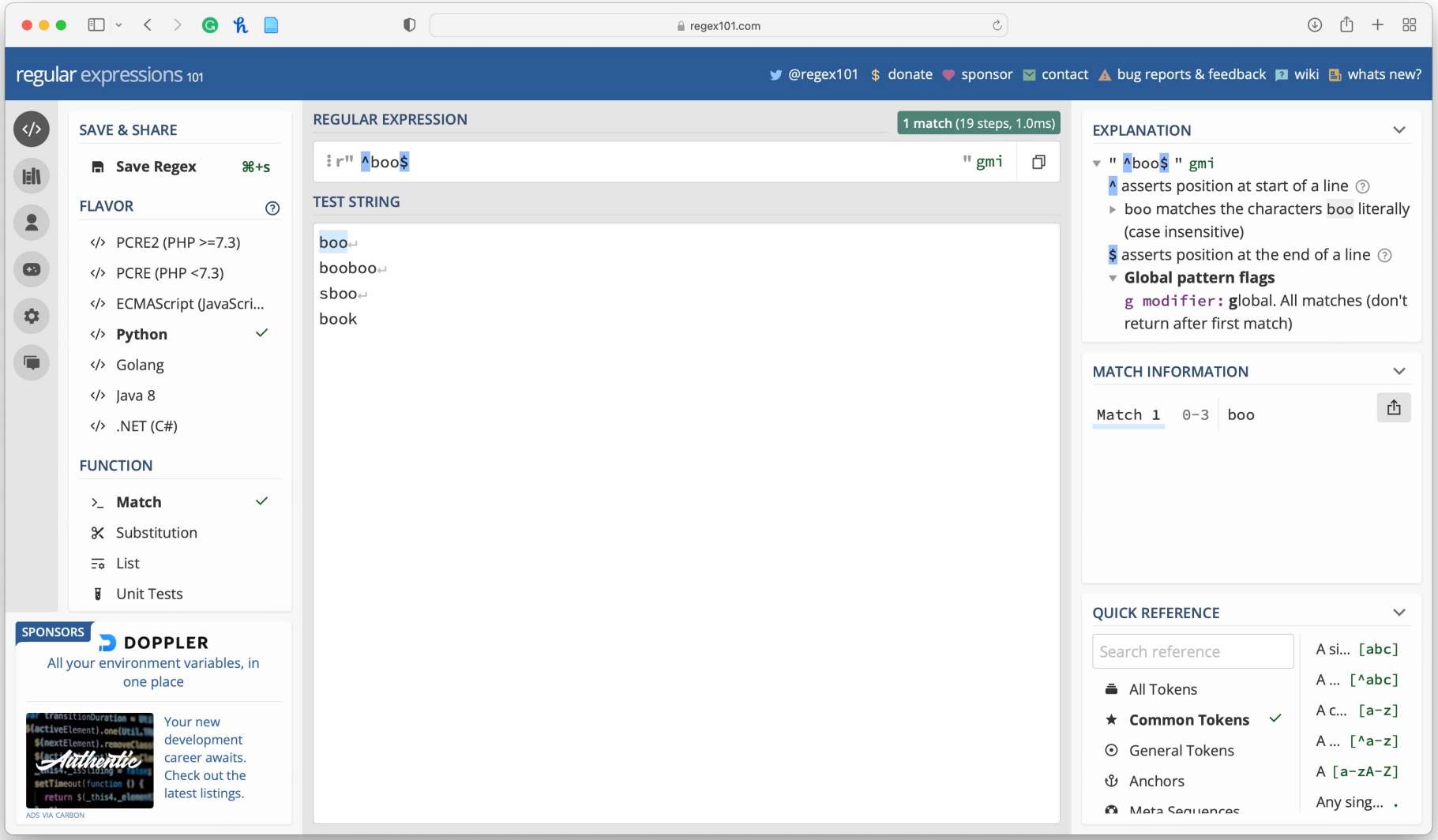


https://www.sitepoint.com/learn-regex/

RegEx101 - Starting and Ending Patterns

- The string “sboo” and “booboo” get a match because they still fulfill the current pattern-matching requirements.
- To fix this, we can update the pattern as follows: ^boo$
- This will strictly match the word “boo”.
- If you use both of them, both rules are enforced.
- For example, ^[a-z]{5}$ strictly matches a five-letter word.
  - If the string has more than five letters, the pattern doesn’t match.

RegEx101 - Starting and Ending Patterns



https://www.sitepoint.com/learn-regex/