# Regular Expression

1.It describe a set of strings that match that pattern

* Match happens between one character and the patter describe that character. For example, if we want to match ‘a’, we can use ‘a’, we can also use [a-z], the former can just match ‘a’, but the second can also match more things(b-z). However, no matter which RegEX we use, they just match one by one! You may assign the location, but you still match one by one. (to be specific, it cannot match the number doesn’t equal to 200 directly, if you want to match that, you should use “(([0-1,3-9][0-9]{2})|(2[1-9][0-9])|(20[1-9]))”)

2.some application

* Use RegEX for password validation, if there is at least one lowercase letter, at least 1 uppercase letter and at least 1 digit and at least 1 special character and no whitespaces, we can use the RegEX: “^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\d)(?=.\*[@$!%\*?&])[^\s]{8,}$” to check. ?= means match but not extract, .\*means matching anything for 0-many times until can match [a-z](or something else), if all the condition can be suit, then extract [^\s]no blank space, and this number should be more than 8 times before the end. There is how we use Re to match character, and what we mean that character by character.
* Match the error with certain type in log

3.Match function(python Re)

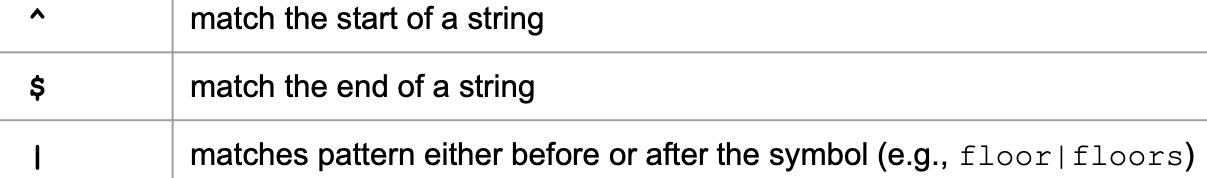
* search(): find first(location) # empty when cannot find
* match():find at the first char # empty when cannot find
* findall():list of matches # without location
* finditer():iterator with all objects matched #location and match, when using, making it into list first, if want to extract content for each object(re.Match) in the list, use .group()
* sub(pattern, sub, sentence) # new sentence which replace pattern with sub

# when we match, if we use flags = re.I, it means, we match no matter if the word is upper case or is lower case

4.Match sign

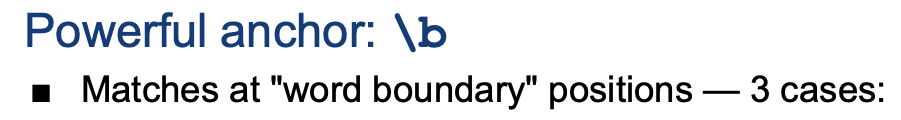
For many words, they can match themselves. For example, we can use “a” for “a”. However, for some character, they can match more than themselves. And we call these characters, “metacharacters”

* \ signals a special sequence(for example, \d is not “\” and “d”, instead means [0-9]
* . (period) match any character except line breaks



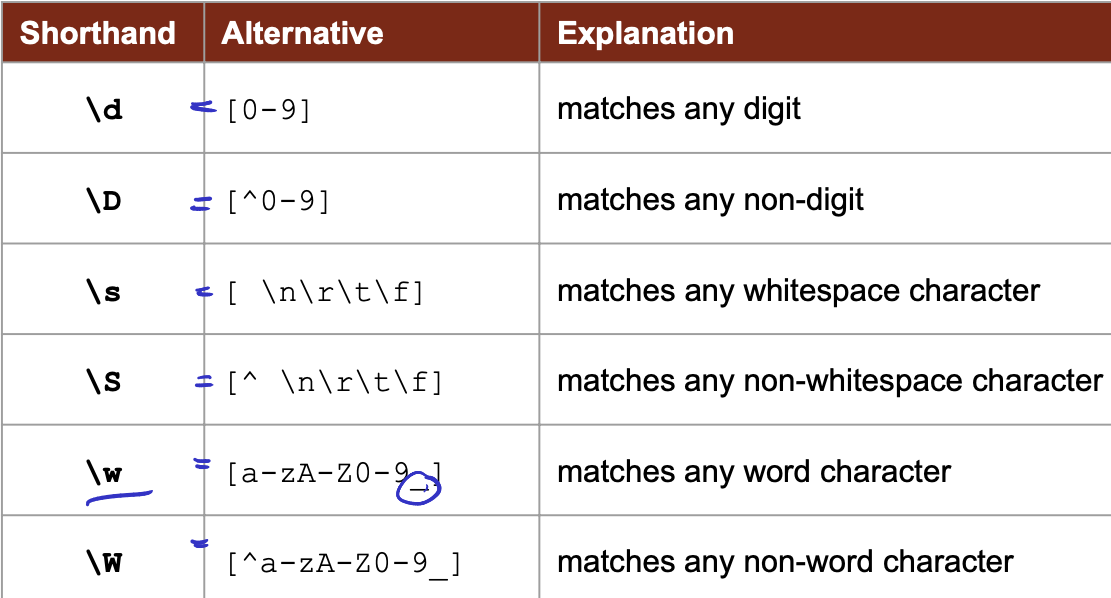
* Anchors: maps the position(indicate if the position is also the correct position, that is what we want)

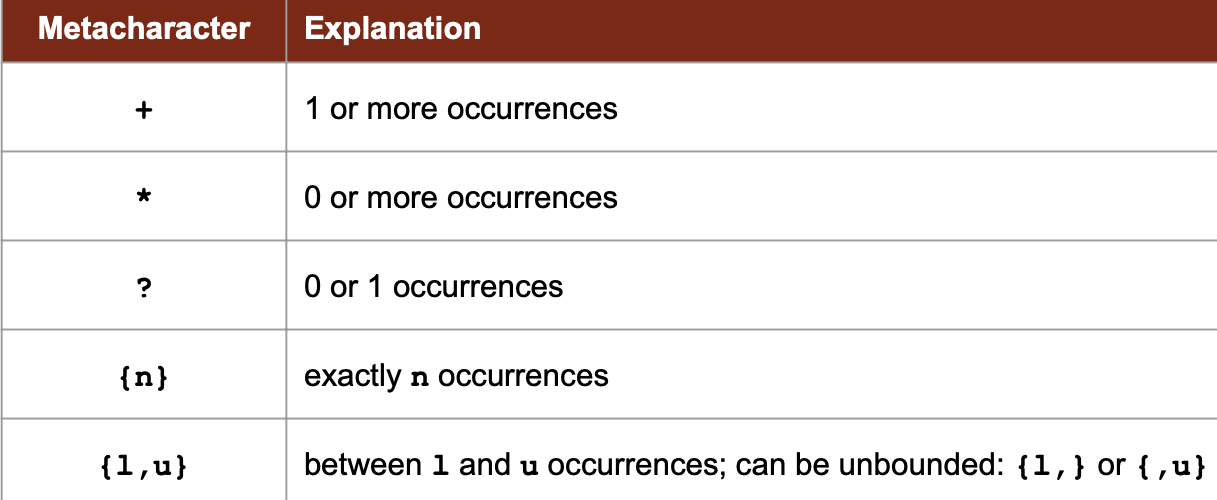




# this is especially for word[a-zA-Z0-9]

How to match 5-character word 



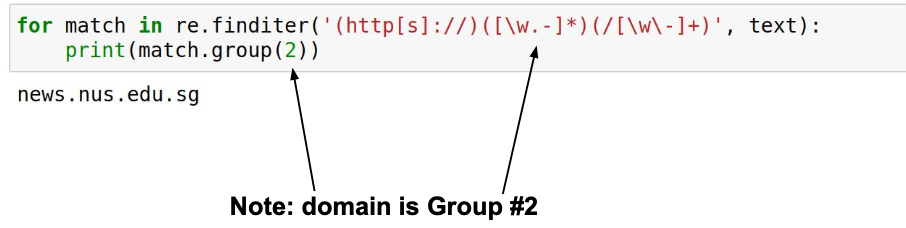


Some example:

1. \d{8}: 8个数字
2. \d?: 0或者1个数字
3. \d+: 1或者无数个数字（等价于{1,}）
4. [a-z] a或者b或者c或者d或者…..z

Since we see that, the [], just match one inside of it, what if we want to match more than one characters and want to use this characters as unit? For example, to match “ababababab”, we may want, “ab”{5},however it cannot be achieved by [].so we use () to denote, and name the content inside of it a group. Except doing this, when we create a group, it can be refer to later, as we memorize it as well.

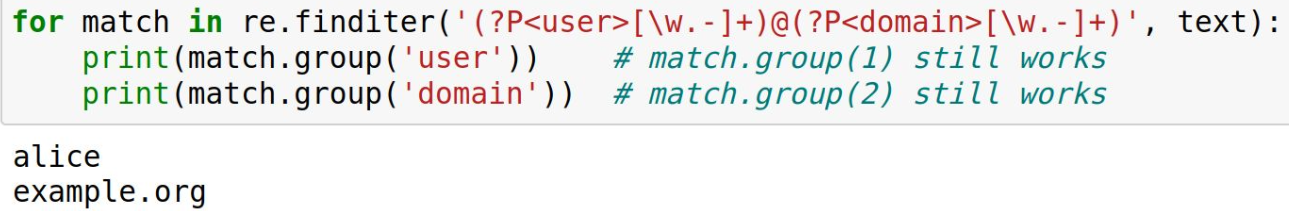
That is how we refer to that:



The number of the group is indexed by the order(location) it(just the first part of it) appear

For example, if we want to same content matched in the first group, we can use \1, to refer to the content（this is also called backreferences）

We can also give a name to the group





What if we just want to see around to see not only the things we want is matched, we also want to check if its surrounding matches? (even though we don’t want their surroundings, but we want to through the surrounding to make sure that is the content we want)

* (?=) after match the content, to see if the content after it also match(positive lookahead)
* (?!) after match the content, to see if the content after it is not match(negative lookahead)
* (?<=) after match the content, to see if the content before it match(positive lookbehind)
* (?<!)

For example:

(\d+)\s(?=SGD) that helps us capture the number of SGD