REGULAR EXPRESSIONS

Regular Expressions, commonly called **regex**, are used to match particular patterns in text.  A regex is a description of a pattern.

Here's an example regex to match StarID values, which are two letters, four numbers, and two letters, and in this example, to be considered a match, the StarID has to be a separate word. So we do want to match **ab1234cd** but not **abcd1234efghijk**.

**\b[a-zA-Z]{2}\d{4}[a-zA-Z]{2}\b**

A regex is made up of tokens - characters or blocks of characters that have a specific meaning to a regex processor. A piece of text is considered a match if it matches the tokens given, in the same order.

* \b means match a word boundary (like the very start of end of the text, or a space, or other whitespace)
* [a-zA-Z] means match a letter in the range a-z or A-Z
* {2} means exactly two of the preceding match tokens - so match two characters in the range a-z or A-Z.
* \d means match a digit
* {4} means match exactly 4 of the preceding token. In this example, match 4 digits in a row.
* [a-zA-Z] as before
* {2} as before
* \b a word boundary, as before

You can see this in action here: <https://regex101.com/r/JUgGg9/4> . Try typing in your StarID and see if it is identified as a match.

Regexs are made of tokens, and plain text. Each character, except special tokens, will match itself.  Each token describes what type of character(s) should be matched. (So if you want to write a regex to match special regex characters, aka metacharacters, like {} [] $ ^ . \ etc, you'll need to escape them. See link below for tutorials. )

Recently I wrote a program to scrape data from one part of the MCTC website, and used regex to identify things that look like ITEC class codes, which are in the format ITEC 1234-01, with the first digit of the class code being a 1 or 2.  So, I used the the regex  **\bITEC [12]\d{3}-\d{2}\b** to match ITEC class codes. This matches

* a word boundary
* the text 'ITEC ' exactly,
* Exactly one character from the list 12  (so either 1, or 2)
* followed by exactly three more digits,
* a dash,
* and exactly 2 more digits.
* another word boundary

So it matches **ITEC 2905-90** but not **ITEC 4904-90**.  Try it here: <https://regex101.com/r/JUgGg9/3>

Regex are very useful.

* For example, you might want to find zip codes by matching parts of text that have 5 consecutive numbers.
* Or, verify an email address is in the correct format of [local\_name@domain](mailto:local_name@domain).
* Your text editor uses regular expressions to figure out what each part of each line means, and highlight the different parts appropriately.
* Security application administrators use regex to scan email and files for potential security breaches, for example, to look for unencrypted credit card numbers
* System administrators use regex to look for files whose filename matches certain patterns. Or, to search inside files for text that matches certain patterns
* Web applications can use regex to match a various request URLs to particular routes, without having to specify each URL individually
* Web scrapers can use regex to look for particular parts of text on a website; for example, a scraper for a real estate website might want to look for text that seems to be part of an address.

Here's a couple of sites for more information. Regex101 is great for testing regex.

<https://regex101.com/>

<http://www.regular-expressions.info/>

Regex can be extremely useful for solving many problems. However, it's important to know what they are good at, and what they don't work so well with. Unfortunately, you can end up in a situation with a really complicated regex that can take a really long time to run (as in these examples of catastrophic backtracking <http://www.regular-expressions.info/catastrophic.html>), and be very difficult to understand, debug and modify. And maybe regex wasn't even the best way to solve your problem anyway. Hence the well-known quote:

*Some people, when confronted with a problem, think "I know, I'll use regular expressions." Now they have two problems.*

*Jamie Zawinski, 1997*

So while it's very useful to know regex, it's also important to know what types of problems they are good at solving, and what type of problems you should look for another way to solve them. Kinda like with many other programming tools and strategies too :)

*Regular expressions are like a particularly spicy hot sauce – to be used in moderation and with restraint only when appropriate. Should you try to solve every problem you encounter with a regular expression? Well, no ... If you drench your plate in hot sauce, you're going to be very, very sorry later. In the same way that I can't imagine food without a dash of hot sauce now and then, I can't imagine programming without an occasional regular expression. It'd be a bland, unsatisfying experience.*

*Jeff Atwood,*[*https://blog.codinghorror.com/regular-expressions-now-you-have-two-problems/*](https://blog.codinghorror.com/regular-expressions-now-you-have-two-problems/)

**Using regex in code**

OK. So, regex can be very helpful. How do you use them in your code? Python (and other languages) come with support for using regex.

<https://docs.python.org/3/library/re.html>

Here's some example code;

<https://github.com/minneapolis-edu/regex>

regex = '[abc]{2}' # Match letter a or b or c twice in a row - so 'aa' or 'bb' or 'ab' or 'cb' ...  
text = 'this string should actually, absolutely have a match' # Only the first match (ac) will be returned. Others (ab) will be ignored  
match = re.search(regex, text)  
if match:  
 print('match: ', match.group()) # group() returns the entire matched text, ac  
 # Can find the positions in the string that the regex pattern matches  
 print(match.start(), match.end(), match.span()) # 19, 21, (19, 21) The position the match started and ended, span is a tuple of both values

One thing to watch out for is the need to escape characters in your Python strings. \b is a regex token, which means match on a word boundary. In a Python string, the \ character has a special meaning, to escape tabs, newlines etc, like in \n. So to use a literal \ in your regex string, you'll need to escape it with another \ like so

import re  
regex = '\\bITEC [12]\\d{3}-\\d{2}\\b' # The regex to match a word boundary is \b. But \ is a special character in a Python string, so need to escape it as \\  
  
test\_string = 'I need to take ITEC 2950-02 Career Prep, to graduate' # Expect this to match  
match = re.search(regex, test\_string)  
print(match) # A match

But it's easier to use Python's raw strings. Preface your string with r and then you don't need to escape characters like \

import re  
regex = r'\bITEC [12]\d{3}-\d{2}\b' # The regex to match a word boundary is \b. But \ is a special character in a Python string, so need to escape it as \\  
  
test\_string = 'I need to take ITEC 2950-02 Career Prep, to graduate' # Expect this to match  
match = re.search(regex, test\_string)  
print(match) # A match

You can also find all of the matches in a string with findall(),

# findall returns a list of matching, non-overlapping, strings  
  
test\_string = 'This string has three matches, ITEC 1150-10, ITEC 2650-90 and ITEC 2454-90'   
  
match\_strings = re.findall(regex, test\_string)  
print(match\_strings)  
for code in match\_strings:  
   print('A matched ITEC class code is ', code) # All three matches

And the re library also has functions to replace parts of a string by regex, or split a string by regex, which can be very useful if you are doing string manipulation.