What Exactly is RegEx?

RegEx is short for regular expression. A RegEx is a sequence of characters that defines a search pattern. They allow you to create a set of rules for the kind of String you want, and then search through a larger body of text in order to find Strings that fit these rules.

Example: Writing A RegEx

At the most basic level, RegEx can allow you to find a specific String within a larger body of text. For example, if we were just looking for every instance of the word “cat” in a larger document, we would use the RegEx “cat”. However, this won’t get us very far practically. We can do much more than this.

Let’s say instead, we’re looking for every 3 letter word in a body of text that ends in “at” (e.g. “cat”, “bat”, “sat”, etc). For this, we can use “[a-z]at”. The brackets create a set of characters to search for, and the a-z gives a range of every character between a and z. Now instead, suppose we want every single word ending in “at”, regardless of length (e.g. “at”, “bat”, “seat”, etc). For this, we can use a quantifier. The * quantifier says that there can be any number, including 0, of the character proceeding it in the string it finds. This means “[a-z]*at” will do what we want; find every String consisting of some number of letters followed by “at”. (If we want to exclude “at” and only find words with 3 or more letters, we could instead use “[a-z]+at”, which requires there be one or more a-z characters before the “at”).

This RegEx still has some issues, though. For example, if we used it to find all of the words ending in “at” in the string “The cat was late,” it would match “cat”, but it would also match “lat”. We need some way to specify that the “at” actually be at the end of the word. For this, we can use a “word boundary,” which requires that there only be “word characters” (letters, numbers or underscores) on one side of the boundary. This is represented as “\b” in the RegEx syntax. So our new RegEx, “[a-z]*at\b”, will match “at”, “cat”, “seat”, and “great”, but not “ate”, “late”, “bat9”, or “at_home”. If we want to avoid example like “9cat” or “grey_cat”, we could add a word boundary to the beginning as well, for a final RegEx of “\b[a-z]*at\b”.

Characters and Quantifiers

Characters:

- \w: Any word character (letters, digits or underscores)
- \d: Any digit
- \s: Any whitespace character (space, tab, newline)
- \W: Any non-word character (NOT a letter, digit, or underscore)
- \D: Any non-digit character
- \S: Any non-whitespace character
- \b: A word boundary (requires there be word characters on only one side of the boundary)
- . : A wildcard. Matches any one character except a newline.
- [xyz]: Matches any character within the brackets
- [x-z]: Matches any character in the range between x and z on the ASCII character list
- [^x]: Matches any character that is not x
  - Can be combined with other techniques. (ie. [^b-x] matches any character not in the range b-x)
- (abc): Matches the string “abc”
- ^: If at the beginning of the RegEx, requires that the pattern be found at the start of the String being searched.
- $: If at the end of the RegEx, requires that the pattern end at the end of the String being searched.

Quantifiers
- +: One or more of the preceding character
- {x}: Exactly x of the preceding character
- {x, y}: Between x and y of the preceding character
- {x, }: X or more of the preceding character
- *: 0 or more of the preceding character
- ?: 0 or 1 of the preceding character

If no quantifier is present on a character or set of characters, the regex will look for examples with exactly one of the character.

Capital vs Lowercase
- [A-Z] will match all capital letters while [a-z] will match all lowercase letters
- [Aa] will match a lowercase or uppercase a
- [A-Za-z] will match any a to z letter (upper or lower case)

Escaping Characters
A number of characters have special meanings in regular expressions. For example, "." is used as a wildcard, matching any one character besides a newline. But what if we’re actually looking for every period in a document? For this, "\." will find every character that’s actually a period. This is known as “escaping” the special character. This will work for any character that has a special meaning in regular expressions (e.g. ()[^?]+).
Regex: Python Edition

To use regexes in python, we use the re library (documentation available here).

Some of the commonly used regex functions are:

```python
re.findall([pattern], [string])
```
Takes in a pattern (regex) and a string. Returns a list of all non-overlapping occurrences of the pattern in the string.

Example usage:
```python
re.findall(r"\d", "I have 45")
```
=> ["4", "5"]

```python
re.split([pattern], [string])
```
This is a replacement for the normal string split() function. It takes in a pattern and a string and splits the string on any part that matches the pattern.

Example usage:
```python
re.split(r"[\s\.]", "we hope you have a great time")
```
=> ["we", "hope", "you", "have", "a", "great", "time"]

```python
re.sub([pattern], [replacement], [string])
```
This takes in a pattern (regex), a replacement string for that pattern, and a string, and replaces all occurrences of the pattern in the string with the replacement provided.

Example usage:
```python
re.sub("-", " ", "a-good-day")
```
=> "a good day"