Data science

Regex (in Python)

2025-2026

Regex - Regular expressions

Regex

- Regular expressions:

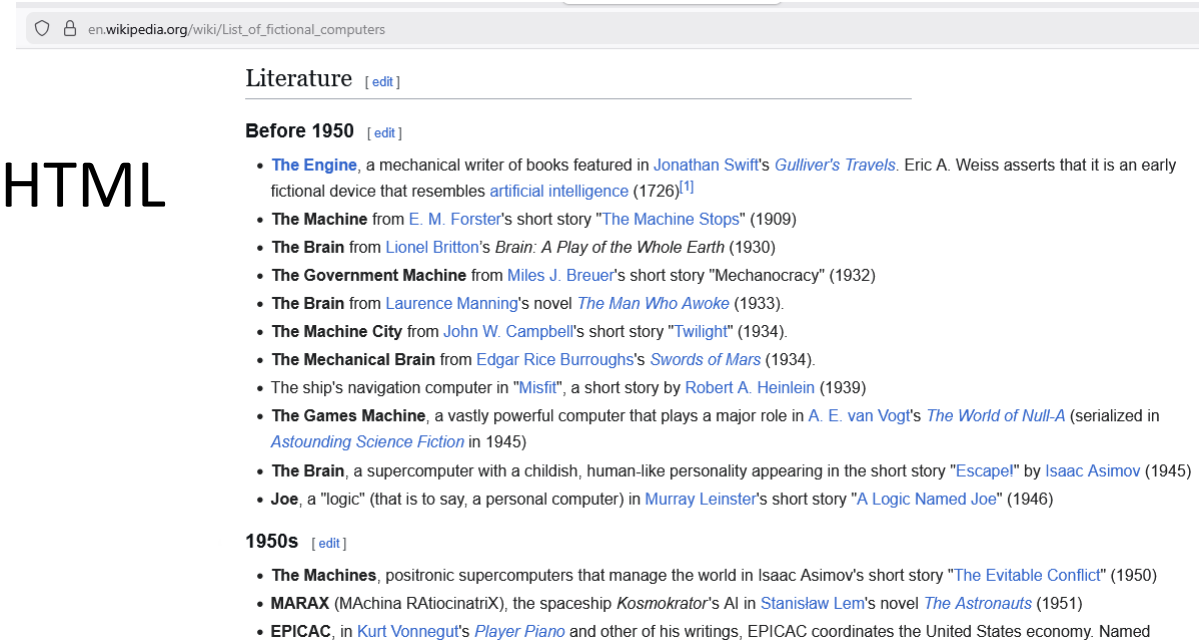
  A **regular expression** (shortened as **regex** or **regexp**;[1] also referred to as **rational expression**) is a sequence of characters that specifies a *search pattern* in text. Typically, such patterns are used by string-searching algorithms for "find" or "find and replace" operations on strings, or for input validation.

Regex

- Regexes can be used in any language
  - But we'll be using Python
- By themselves, they are rather theoretical, but in web scraping or data cleaning, you will see their usefulness

Regex

- A regex is an expression which matches (or not matches) another string. This means you can check if a string is a properly formed date, or a gender, or …
- You can also use regexes to extract parts of strings. Like taking al the fictional computers from wikipedia
  - Note: they're all bold
  - This means between "<b>"-tags in HTML



Basic regexes - How do we match stuff?

- Not all strings are so straightforward to match as “abc” or “<b>”
- What if we want to match:
  - any digit
  - a whitespace
  - a word that starts with any uppercase letter
  - a word at the start of the line

→ **Tokens**

Regex	Matches
\d	Any digit
\D	Any non-digit
\s	Any whitespace (space, but also newline)
\S	Everything but \s
\w	Matches any alphanumeric char (incl. “_”)
\W	Everything but \w
.	Any character, but only once
[a-z]	Any lowercase character
[A-Z]	Any uppercase character
[0-9]	Any digit, equal to \d
$	End of string
^	Start of string

Basic regexes - How do we match stuff?

- Not all strings are so straightforward to match as “abc” or “<b>”
- What if we want to match:
  - Any vowel
  - A character that is either an “a”, “b” or “c”
  - A number between 1-5

Regex	Matches
[abc]	“a”, “b” or “c”
[^abc]	Everything but “a”, “b” or “c”
[a-z]	Any lowercase character
[A-Z]	Any uppercase character
[0-9]	Any digit, equal to \d
abc|def	“abc” or “def”

→ **Collections**

Basic regexes - How do we match stuff?

- Not all strings are so straightforward to match as “abc” or “<b>”
- What if we want to match:
  - “b” zero or more times
  - just one “@” in an email address
  - exactly three “w” in a url (www)

→ **Quantifiers**

Quantifier	Multiplies how?
*	Zero or more times
+	One or more times
?	Zero or once
{3}	Exactly three times
{3,6}	Three to six times
{3,}	Three or more times
{,6}	Up to six times

Basic regexes - All together now!

Regex	Matches
a	The character “a”
abc	The string “abc”

Quantifier	Multiplies how?
*	Zero or more times
+	One or more times
?	Zero or once
{3}	Exactly three times
{3,6}	Three to six times
{3,}	Three or more times
{,6}	Up to six times

Regex	Matches
[abc]	“a”, “b” or “c”
[^abc]	Everything but “a”, “b” or “c”
abc|def	“abc” or “def”
\d	Any digit
\D	Any non-digit
\s	Any whitespace (space, but also newline)
\S	Everything but \s
\w	Matches any alphanumeric char (incl. “_”)
\W	Everything but \w
.	Any character, but only once
[a-z]	Any lowercase character
[A-Z]	Any uppercase character
[0-9]	Any digit, equal to \d

Regex groups

- What if you want to match more than one item at a time?
- When searching an email address you want to get:
  - First name
  - Last name
  - Student
  - Domain
  - Country

stef.vanwolputte@thomasmore.be
jos.eemlen@student.thomasmore.be

→ **Groups**

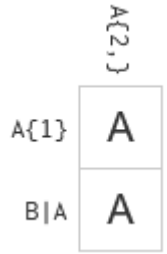
Regex groups

- A regex group is anything between brackets (“(“ and “)”)
- You can refer to these groups using \1, \2, \3, …
  - \1 is the first group, \2 the second, etcetera…
- When applying regexes to strings, the groups are returned individually
  - When you don’t specify groups, the entire found text is returned

- Example: word that begins and ends with the same letter:
  (.).*\1
- Email address example:
  ^([a-z]+)\.([a-z]+)@(student\.)?([a-z]+)\.([a-z]{2,3})$

Regex

- The previous slides are all you need to know
- Now you can apply Regex in any language
  - PowerShell, Perl, Java, C#, …
- Let's try that first: https://regexcrossword.com/
- Example:
  - A{2,} : At least two A's -> AA
  - A{1} : Exactly one A -> A
  - B|A : B or A -> A



- You should be able to finish the beginners, intermediate and expert puzzles
- When in doubt, use https://regex101.com/ to explain why something is wrong


WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP

Regex in Python

Regex in Python

- All well and good, but how do we use this in Python?
- First “import re” (usually not given an alias)
- Then use one of the following functions:
  - re.search(regex, s): returns the first match of the regular expression, even if this match is in the middle of the string
  - re.match(regex, s): returns the first match of the regular expression, but only if this match is in the beginning of the string
  - re.finditer(regex, s): returns an iterator consisting of all matches of regex
  - re.findall(regex, s): returns a list of all matches of the regex
  - re.sub(regex, new_text, s): substitutes all matches of the regex in the input string s with new_text and returns this

Example:

```
import re
s = "I went to the zoo and saw _lions_, _tigers_ and _meerkats_."
```

- We start out with the text on top of this slide.

```
found = re.search(r"lion", s)
print(found)
```

```
<re.Match object; span=(27, 31), match='lion'>
```

- Note: “found” is not a string, but an object
  - https://docs.python.org/3.10/library/re.html#match-objects
- When using this object remember you’re using groups in regexes
  - Hence the constant usage of ‘group’
  - When only looking for one match, group will always be zero (the first group, also the default option)

Example: Match-object

```
found = re.search(r"lion", s)
print(found.group(), found.start(), found.end())
```

```
lion 27 31
```

- But did we visit the flamingos?

```
found = re.search(r"flamingo", s)
print(found)
print("We found it!") if found else print("Nothing here...")
```

```
None
Nothing here...
```

- If nothing found, return value is of type “None” which evaluates to False in an if statement

```
found = re.match(r"tiger", s)
print(found)

found = re.match(r"I\s", s)
print(found)
```

```
None
<re.Match object; span=(0, 2), match='I '>
```

Example: greedy *

- Find the first animal (between underscores):

```
found = re.search(r"_.*_", s)
print(found.group())
```

- No correct result:

```
_lions_, _tigers_ and _meerkats_
```

- * is greedy: it will take as much as it can
  - 2 solutions: match only characters, or match anything but underscore

```
found = re.search(r"_[a-zA-Z]*_", s)
print(found.group())
```

```
_lions_
```

```
found = re.search(r"_[^_]*_", s)
print(found.group())
```

Example: iterators and iterables

- We want to know all animals

```
found = re.finditer(r"_[a-zA-Z]*_", s)
for f in found:
    print(f.group())
```

```
_lions_
_tigers_
_meerkats_
```

- **finditer**: returns an iterator, not an iterable
- **findall**: returns a list, which is an iterable
  - https://www.geeksforgeeks.org/python-difference-iterable-iterator/
  - Any iterable can be used to generate an iterator. This is what is done implicitly when used in a loop.
  - Iterables: string, list, tuple, …
- You're used to iterables, so use **findall** instead

Example: find more

```
found = re.findall(r"_[a-zA-Z]*_", s)
[print(f) for f in found]
```

```
_lions_
_tigers_
_meerkats_
```

- Since we're using findall, we don't get Match-objects anymore but a simple list of matches.
- But the underscores are annoying. We need them to search for the animals, but we know they're there and don't need them in the result.
- The solution: parenthesis. Put the part of the regex you're interested in between them.

```
found = re.findall(r"_([a-zA-Z]*)_", s)
[print(f) for f in found]
```

```
lions
tigers
meerkats
```

Example: replace

- Let's get rid of all the underscores.

```
new_s = re.sub(r"_", r"", s)
print(new_s)
```

```
I went to the zoo and saw lions, tigers and meerkats.
```

- Or let's replace every T that is followed by another letter with an X.

```
new_s = re.sub(r"t[a-zA-Z]+", r"X", s)
print(new_s)
```

```
I went X X zoo and saw _lions_, _X_ and _meerkaX_.
```

- What happened? The entire regex was replaced, not just the T. To fix that we have to use groups!

Example: replace using groups

```
new_s = re.sub(r"t([a-zA-Z]+)", r"X\1", s)
print(new_s)
```

- "\1" is the first group that was found
  - You can also use \2, \3, …
  - A group is anything between brackets
- Can also be used in the expression itself, like checking if the first character is the same as the last:

```
found = re.match(r"(.{1}).*\1$", "hannah")
print(found)

found = re.match(r"(.{1}).*\1$", "kasper")
print(found)
```

```
<re.Match object; span=(0, 6), match='hannah'>
None
```

Time to practice

Exercises

- R-numbers
- Numbers in text
- Find text
- …

Exercises from the advent of code 2015:

- Day 5
- Day 6