

COMP(2041|9044) 25T2 — Python Regular Expressions

<https://www.cse.unsw.edu.au/~cs2041/25T2/>

<https://www.cse.unsw.edu.au/~cs2041/25T2/>

COMP(2041|9044) 25T2 — Python Regular Expressions

1 / 27

Regular Expression History Revisited

- 1950s mathematician Stephen Kleene develops theory
- 1960s Ken Thompson develops syntax and practical implementation, two versions:
 - POSIX **Basic Regular Expressions**
 - limited syntax, e.g no |
 - used by `grep` & `sed`
 - needed when computers were every slow to make regex matching faster
 - POSIX **Extended Regular Expressions** - superset of Basic Regular Expressions
 - used by `grep -E` & `sed -E`
- 1980s Henry Spencer produces open source regex C library
 - used many place e.g. postgresql, tcl
 - extended (added features & syntax) to Ken's regex language.
- 1987 Perl (Larry Wall) copied Henry's library & extended much further
 - available outside Perl via **Perl Compatible Regular Expressions** library
 - used by `grep -P`
- 1990s Python standard **re** package also copied Henry's library
 - added most of the features in Perl/PCRE
 - many commonly used features are common to both
- we will cover some (not all) useful extra regex features found in both Python & Perl/PCRE
- note <https://regex101.com/> lets you specify which regex language

<https://www.cse.unsw.edu.au/~cs2041/25T2/>

COMP(2041|9044) 25T2 — Python Regular Expressions

2 / 27

Python **re** package - useful functions

```
re.search(regex, string, flags)
# search for a *regex* match within *string*
# return object with information about match or `None` if match fails
# optional parameter flags modifies matching,
# e.g. make matching case-insensitive with: `flags=re.I`

re.match(regex, string, flags)
# only match at start of string
# same as `re.search` stating with `^`

re.fullmatch(regex, string, flags)
# only match the full string
# same as `re.search` stating with `^` and ending with `$`
```

<https://www.cse.unsw.edu.au/~cs2041/25T2/>

COMP(2041|9044) 25T2 — Python Regular Expressions

3 / 27

Match objects

- `re.search`, `re.match`, `re.fullmatch` return a match object if a match succeeds, `None` if it fails
 - hence their return can be used to control `if` or `while`

```
print("Destroy the file system? ")
answer = input()
if re.match(r'yes|ok|affirmative', answer, flags=re.I):
    subprocess.run("rm -r /", Shell=True)
```

- the match object can provide useful information:

```
>>> m = re.search(r'[aiou].*[aeiou]', 'pillow')
>>> m
<re.Match object; span=(1, 5), match='illo'>
>>> m.group(0)
'illo'
>>> m.span()
(1, 5)
>>>
```

Capturing Parts of a Regex Match

- brackets are used for grouping (like arithmetic) in extended regular expressions
- in Python (& PCRE) brackets also capture the part of the string matched
- **group(*n*)** returns part of the string matched by the *n*th-pair of brackets

```
>>> m = re.search(r'(\w+)\s+(\w+)', 'Hello Andrew')
>>> m.groups()
('Hello', 'Andrew')
>>> m.group(1)
'Hello'
>>> m.group(2)
'Andrew'
```

- **\number** can be used to refer to group *number* in an `re.sub` replacement string

```
>>> re.sub(r'(\d+) and (\d+)', r'\2 or \1', "The answer is 42 and 43?")
'The answer is 43 or 42?'
```

Back-referencing

- **\number** can also be used in a regex as well
- usually called a back-reference
 - e.g. `r'^(\d+) (\1)$'` match the same integer twice

```
>>> re.search(r'^(\d+) (\d+)$', '42 43')
<re.Match object; span=(0, 5), match='42 43'>
>>> re.search(r'^(\d+) (\1)$', '42 43')
None
>>> re.search(r'^(\d+) (\1)$', '42 42')
<re.Match object; span=(0, 5), match='42 42'>
```

- back-references allow matching impossible with classical regular expressions
- python supports up to 99 back-references, `\1`, `\2`, `\3`, ..., `\99`
 - `\01` or `\100` is interpreted as an octal number

- `(?:...)` is a non-capturing group
 - it has the same grouping behaviour as `(...)`
 - it doesn't capture the part of the string matched by the group

```
>>> m = re.search(r'.*(?:[aeiou]).*([aeiou]).*', 'abcde')
>>> m
<re.Match object; span=(0, 5), match='abcde'>
>>> m.group(1)
'e'
```

Greedy versus non-Greedy Pattern Matching

- The default semantics for pattern matching is **greedy**:
 - starts match the first place it can succeed
 - make the match as long as possible
- The `?` operator changes pattern matching to **non-greedy**:
 - starts match the first place it can succeed
 - make the match as short as possible

```
>>> s = "abbbc"
>>> re.sub(r'ab+', 'X', s)
'Xc'
>>> re.sub(r'ab+?', 'X', s)
'Xbbbc'
```

Why Implementing a Regex Matching isn't Easy

- regex matching starts match the first place it can succeed
- but a regex can partly match many places

```
>>> re.sub(r'ab+c', 'X', "abbabbbbbbbbabbbc")
'abbabbbbbbbX'
```

- and may need to **backtrack**, e.g:

```
>>> re.sub(r'a.*bc', 'X', "abbabbbbbbbcabbb")
'Xabbb'
```

- poorly designed regex engines can get very slow
 - have been used for denial-of-service attacks
- Python (PCRE) regex matching is **NP-hard** due to back-references

- `re.findall` returns a list of the matched strings, e.g:

```
>>> re.findall(r'\d+', "-5==10zzz200_")
['5', '10', '200']
```

- if the regex contains `()` only the captured text is returned

```
>>> re.findall(r'(\d)\d*', "-5==10zzz200_")
['5', '1', '2']
```

- if the regex contains multiple `()` a list of tuples is returned

```
>>> re.findall(r'(\d)\d*(\d)', "-5==10zzz200_")
[('1', '0'), ('2', '0')]
>>> re.findall(r'([^\s]*)', "Hopper, Grace Brewster Murray")
[('Hopper', 'Grace')]
>>> re.findall(r'([A-Z])([aeiou])', "Hopper, Grace Brewster Murray")
[('H', 'o'), ('M', 'u')]
```

re.split

- `re.split` splits a string where a regex matches

```
>>> re.split(r'\d+', "-5==10zzz200_")
['-', '==', 'zzz', '_']
```

- like `cut` in Shell scripts - but more powerful
- for example, you can't do this with `cut`

```
>>> re.split(r'\s*,\s*', "abc,de, ghi ,jk , mn")
['abc', 'de', 'ghi', 'jk', 'mn']
```

see also the string `join` function

```
>>> a = re.split(r'\s*,\s*', "abc,de, ghi ,jk , mn")
>>> a
['abc', 'de', 'ghi', 'jk', 'mn']
>>> ':'.join(a)
'abc:de:ghi:jk:mn'
```

Example - printing the last number using `re.search`

```
# Print the last number (real or integer) on every line
# Note: regexp to match number: -?\d+\.\?\d*
# Note: use of assignment operator :=
import re, sys
for line in sys.stdin:
    if m := re.search(r'(-?\d+\.\?\d*)\D*$', line):
        print(m.group(1))
```

source code for `print_last_number.0.py`

Example - printing the last number using re.findall

```
# Print the last number (real or integer) on every line
# Note: regexp to match number: -?\d+\.\d*
# Note: use of assignment operator :=
import re, sys
for line in sys.stdin:
    if m := re.search(r'(-?\d+\.\d*)\D*$', line):
        print(m.group(1))
```

source code for print_last_number.0.py

#Example - finding numbers #0 {shrink}

```
##### #!!INCLUDECODE
code/find_numbers.0.py[3:] # # #Example - finding numbers #1 {shrink}
##### #!!INCLUDECODE
code/find_numbers.1.py[3:]
```

Example - counting enrollments with regexes & dicts

```
course_names = {}
with open(COURSE_CODES_FILE, encoding="utf-8") as f:
    for line in f:
        if m := re.match(r"(\S+)\s+(.*\S)", line):
            course_names[m.group(1)] = m.group(2)
enrollments_count = {}
with open(ENROLLMENTS_FILE, encoding="utf-8") as f:
    for line in f:
        course_code = re.sub(r"\|.*\n", "", line)
        if course_code not in enrollments_count:
            enrollments_count[course_code] = 0
        enrollments_count[course_code] += 1
for (course_code, enrollment) in sorted(enrollments_count.items()):
    # if no name for course_code use ???
    name = course_names.get(course_code, "???")
    print(f"{enrollment:4} {course_code} {name}")
```

source code for count_enrollments.0.py

Example - counting enrollments with split & counters

```
course_names = {}
with open(COURSE_CODES_FILE, encoding="utf-8") as f:
    for line in f:
        course_code, course_name = line.strip().split("\t", maxsplit=1)
        course_names[course_code] = course_name
enrollments_count = collections.Counter()
with open(ENROLLMENTS_FILE, encoding="utf-8") as f:
    for line in f:
        course_code = line.split("|")[0]
        enrollments_count[course_code] += 1
for (course_code, enrollment) in sorted(enrollments_count.items()):
    # if no name for course_code use ???
    name = course_names.get(course_code, "???")
    print(f"{enrollment:4} {course_code} {name}")
```

source code for count_enrollments.1.py

Example - counting first names

```
already_counted = set()
first_name_count = collections.Counter()
with open(ENROLLMENTS_FILE, encoding="utf-8") as f:
    for line in f:
        _, student_number, full_name = line.split("|")[0:3]
        if student_number in already_counted:
            continue
        already_counted.add(student_number)
        if m := re.match(r".*,\s+(\S+)", full_name):
            first_name = m.group(1)
            first_name_count[first_name] += 1
# put the count first in the tuples so sorting orders on count before name
count_name_tuples = [(c, f) for (f, c) in first_name_count.items()]
# print first names in decreasing order of popularity
for (count, first_name) in sorted(count_name_tuples, reverse=True):
    print(f"{count:4} {first_name}")
```

source code for count_first_names.py

Example - finding duplicate first names using dict of dicts

```
course_first_name_count = {}
with open(ENROLLMENTS_FILE, encoding="utf-8") as f:
    for line in f:
        course_code, _, full_name = line.split("|")[0:3]
        if m := re.match(r".*,\s+(\S+)", full_name):
            first_name = m.group(1)
        else:
            print("Warning could not parse line", line.strip(),
                  ↪ file=sys.stderr)
            continue
        if course_code not in course_first_name_count:
            course_first_name_count[course_code] = {}
        if first_name not in course_first_name_count[course_code]:
            course_first_name_count[course_code][first_name] = 0
        course_first_name_count[course_code][first_name] += 1
for course in sorted(course_first_name_count.keys()):
    for (first_name, count) in course_first_name_count[course].items():
        if count >= REPORT_MORE_THAN_STUDENTS:
            print(course, "has", count, "students named", first_name)
```

source code for duplicate_first_names.0.py

Example - finding duplicate first names using split & defaultdict of counters

```
course_first_name_count = collections.defaultdict(collections.Counter)
with open(ENROLLMENTS_FILE, encoding="utf-8") as f:
    for line in f:
        course_code, _, full_name = line.split("|")[0:3]
        given_names = full_name.split(",")[1].strip()
        first_name = given_names.split(" ")[0]
        course_first_name_count[course_code][first_name] += 1
for (course, name_counts) in sorted(course_first_name_count.items()):
    for (first_name, count) in name_counts.items():
        if count > REPORT_MORE_THAN_STUDENTS:
            print(course, "has", count, "students named", first_name)
```

source code for duplicate_first_names.1.py

Example - Changing Filenames with Regex

```
# written by andrewt@unsw.edu.au for COMP(2041|9044)
#
# Change the names of the specified files
# by substituting occurrences of regex with replacement
# (simple version of the perl utility rename)
import os
import re
import sys
if len(sys.argv) < 3:
    print(f"Usage: {sys.argv[0]} <regex> <replacement> [files]",
          file=sys.stderr)
    sys.exit(1)
regex = sys.argv[1]
replacement = sys.argv[2]
for old_pathname in sys.argv[3:]:
    new_pathname = re.sub(regex, replacement, old_pathname, count=1)
    if new_pathname == old_pathname:
        continue
    if os.path.exists(new_pathname):
        print(f"{sys.argv[0]}: '{new_pathname}' exists", file=sys.stderr)
        continue
    try:
        os.rename(old_pathname, new_pathname)
    except OSError as e:
        print(f"{sys.argv[0]}: '{new_pathname}' {e}", file=sys.stderr)
```

source code for rename_regex.py

<https://www.cse.unsw.edu.au/~cs2041/25T2/>

COMP(2041|9044) 25T2 — Python Regular Expressions

22 / 27

Example - Changing Filenames with Regex & Eval

```
# written by andrewt@unsw.edu.au for COMP(2041|9044)
#
# Change the names of the specified files
# by substituting occurrences of regex with replacement
# (simple version of the perl utility rename)
#
# also demonstrating argument processing and use of eval
# beware eval can allow arbitrary code execution,
# it should not be used where security is important
import argparse
import os
import re
import sys
parser = argparse.ArgumentParser()
# add required arguments
parser.add_argument("regex", type=str, help="match against filenames")
parser.add_argument("replacement", type=str, help="replace matches with")
parser.add_argument("filenames", nargs="+", help="filenames to be changed")
# you may omit the Boolean arguments
parser.add_argument(
    "-d", "--dryrun", action="store_true", help="show changes but don't make them"
)
parser.add_argument(
    "-v", "--verbose", action="store_true", help="print more information"
)
parser.add_argument(
    "-e",
    "--eval",
    action="store_true",
    help="evaluate replacement as python expression, match available as _",
)
# optional integer argument which defaults to 1
parser.add_argument(
    "-n",
    "--replace_n_matches",
    type=int,
    default=1,
    help="replace n matches (0 for all matches)",
)
args = parser.parse_args()
def eval_replacement(match):
    # eval gives python replacement string as Python
    # with the variable _ set to the matching part of the filename
    _ = match.group(0)
    if not args.eval:
        return args.replacement
    _ = match.group(0)
    return str(eval(args.replacement))
for old_pathname in args.filenames:
    try:
        new_pathname = re.sub(
            args.regex, eval_replacement, old_pathname,
            count=args.replace_n_matches
        )
    except OSError as e:
        print(
            f"{sys.argv[0]}: '{old_pathname}': '{args.replacement}': {e}",
            file=sys.stderr,
        )
        continue
    if new_pathname == old_pathname:
        if args.verbose:
            print(f"no change", old_pathname)
        continue
    if os.path.exists(new_pathname):
        print(f"{sys.argv[0]}: '{new_pathname}' exists", file=sys.stderr)
        continue
    if args.dryrun:
        print(f"dryrun: '{old_pathname}' would be renamed to '{new_pathname}'")
        continue
    if args.verbose:
        print(f"renaming", old_pathname, "to", new_pathname)
    try:
        os.rename(old_pathname, new_pathname)
    except OSError as e:
        print(f"{sys.argv[0]}: '{new_pathname}': {e}", file=sys.stderr)
```

source code for change_names.py

<https://www.cse.unsw.edu.au/~cs2041/25T2/>

COMP(2041|9044) 25T2 — Python Regular Expressions

23 / 27

Example - Prejudice and Pride #0

```
# For each file given as argument replace occurrences of Elizabeth
# and shorter forms of Elizabeth with Darcy and vice-versa.
# Relies on Zaphod not occurring in the text.
# use custom temporary file
import re, sys, os
for filename in sys.argv[1:]:
    tmp_filename = filename + ".new"
    if os.path.exists(tmp_filename):
        print(f"{sys.argv[0]}: {tmp_filename} already exists\n",
              file=sys.stderr)
        sys.exit(1)
    with open(filename) as f:
        with open(tmp_filename, "w") as g:
            for line in f:
                changed_line = re.sub(r"Elizabeth|Lizzy|Eliza", "Zaphod",
                                     line)
                changed_line = changed_line.replace("Darcy", "Elizabeth")
                changed_line = changed_line.replace("Zaphod", "Darcy")
                g.write(changed_line)
            os.rename(tmp_filename, filename)
```

source code for change_names.0.py

<https://www.cse.unsw.edu.au/~cs2041/25T2/>

COMP(2041|9044) 25T2 — Python Regular Expressions

24 / 27

Example - Predjudice and Pride #1

```
# For each file given as argument replace occurrences of Elizabeth
# and shorter forms of the name with Darcy and vice-versa.
# Relies on Zaphod not occurring in the text.
# use tempfile to create temporary file - robust & secure
import re, sys, shutil, tempfile
for filename in sys.argv[1:]:
    with tempfile.NamedTemporaryFile(mode="w", delete=False) as tmp:
        with open(filename) as f:
            for line in f:
                changed_line = re.sub(r"Elizabeth|Lizzy|Eliza", "Zaphod",
↪ line)
                changed_line = changed_line.replace("Darcy", "Elizabeth")
                changed_line = changed_line.replace("Zaphod", "Darcy")
                tmp.write(changed_line)
    shutil.move(tmp.name, filename)
```

source code for change_names.1.py

Example - Predjudice and Pride #2

```
# For each file given as argument replace occurrences of Elizabeth
# and shorter forms of Elizabeth with Darcy and vice-versa.
# Relies on Zaphod not occurring in the text.
# modified text is stored in a list then file over-written
import re, sys
for filename in sys.argv[1:]:
    changed_lines = []
    with open(filename) as f:
        for line in f:
            changed_line = re.sub(r"Elizabeth|Lizzy|Eliza", "Zaphod", line)
            changed_line = changed_line.replace("Darcy", "Elizabeth")
            changed_line = changed_line.replace("Zaphod", "Darcy")
            changed_lines.append(changed_line)
    with open(filename, "w") as g:
        g.write("".join(changed_lines))
```

source code for change_names.2.py

Example - Predjudice and Pride #3

```
# For each file given as argument replace occurrences of Elizabeth
# and shorter forms of Elizabeth with Darcy and vice-versa.
# Relies on Zaphod not occurring in the text.
# modified text is stored in a single string then file over-written
import re, sys
for filename in sys.argv[1:]:
    changed_lines = []
    with open(filename) as f:
        text = f.read()
    changed_text = re.sub(r"Elizabeth|Lizzy|Eliza", "Zaphod", text)
    changed_text = changed_text.replace("Darcy", "Elizabeth")
    changed_text = changed_text.replace("Zaphod", "Darcy")
    with open(filename, "w") as g:
        g.write("".join(changed_text))
```

source code for change_names.3.py