# regex

ביטויים רגולריים (regular expression או בקיצור regex) מאפשרים למתכנת למצוא תבניות של תווים בקוד במקום לבצע שאילתות מורכבות על הטקסט.  
הביטוי הרגולרי של שפות תכנות בנוי על בסיס ההגדרה המתמטית של ביטוי רגולרי.  
אז למה צריך בכלל ביטויים רגולרים?   
נסתכל על הדוגמא הבאה- נניח אנחנו רוצים לבנות פונקציה שבודקת האם מחרוזת מסויימת מייצגת מספר או לא.  
אם ננסה לבנות פונקציה ייעודית פשוטה ואינטואיטיבית זה ייקח לנו הרבה מאוד שורות קוד, מספר ישראלי מוגדר, נכון לכתבית שורות אלה, בצורה 05x-xxx-xxxx ובמקום ה-x מספרים ב- [1,9]:

def is\_a\_phone\_number(text:str)->bool:

    '''Checks if a string is a valid  phone number '''

    # 05x-xxx-xxxx x ∈ [1,9]

    if len(text) != 12:

        return False

    if text[0:2] != '05':

        return False

    nums = text.split('-')

    if len(nums) != 3:

        return False

    for num in nums:

        if not num.isdecimal():

            return False

        if num == nums[2] and len(num)!= 4:

            return False

    return True

והעסק יכול להסתבך אף יותר, נניח אנחנו רוצים למצוא את כל המספרי טלפון שנמצאים במחרוזת כלשהי, הגישה הפשוטה ביותר תיהיה לבצע חיפוש שלם:

from typing import List

def find\_phone\_numbers(text:str)-> List[str]:

    '''Finds all phone number in a string '''

    list\_of\_phone\_numbers = list()

    for i in range(len(text)):

        if is\_a\_phone\_number(text[i:i+12]):

           list\_of\_phone\_numbers.append(text[i:i+12])

    return list\_of\_phone\_numbers

סהכ כל התהליך לקח לנו כ-20 שורות קוד, אבל אם היינו יכולים לקצר את התהליך ולהוריד אותו לשלוש שורות קוד בלבד? בשביל זה נצטרך להשתמש בביטויים רגולרים.  
המודול של regex נמצא כבר בספרייה הסטנדרטית של פייתון ואפשר להשתמש בו ללא צורך בהתקנה של תוספים זרים.   
בשביל להשתמש בביטויים רגולרים נצטרך לייבא את המודול re ולו יש כמה פונקציות חשובות.   
הראשונה היא הפונקציה compile() שמקבלת איזשהו ביטוי רגולרי ומחזירה אובייקט מטיפוס pattern (תבנית), עם האובייקט הזה אפשר לבצע שאילתות על טקסט מסוים כמו למשל למצוא את כל המופעים של מבנה הנ"ל או מופע אחד שלו וכו'.   
לפני שניכנס לפרטים בנוגע לשימוש ב-regex ,נראה איך היינו פותרים את הבעיה:

from typing import List

def find\_phone\_numbers\_regex(text:str)-> List[str]:

    import re

    phone\_pattern = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')

    phones = phone\_pattern.findall(text)

    return phones

לא נראה שינוי מהותי, אבל תחשבו שעכשיו לא צריך בכלל את הפונקציה is\_a\_phone\_number() , כך שזה חסך מעל לעשר שורות קוד.

משתנים-

כפי שאמרנו קודם ביטויים רגולרים מבוססים על תחבירים מתמטים פשוטים.  
בתחביר רגולרי צריך להגדיר את סוג הטיפוסים השונים שעליהם התחביר בנוי, הטיפוסים יכולים להיות מספריים או אותיות, למשל ראינו בדוגמא לעיל עם מספרי הטלפון טיפוס של מספרים: '\d' שימו לב שעבור ביטוי מדויק שאנחנו רוצים למצוא אנחנו ממש כותבים אותו, למשל בדוגמא כשרצינו לבדוק שבביטוי יש את התו '-' ממש כתבנו אותו בשאילתה.  
בשביל למצוא איזשהן אותיות לטיניות (באל"ף בי"ת האנגלי הסטנדרטי) או מספרים ותווים שכחים נשתמש ב-'\w' , למשל אם נרצה לבדוק האם ביטוי מכיל אות לטנית סטנדרטית (לפחות אחת).  
קבוצה הפוכה לשני הטיפוסים יהיו אותן האותיות רק באותיות גדולה, למשל נניח אנחנו רוצים למצוא את הביטוי הראשון בטקסט בלי אותיות לטיניות או מספרים ותווים מיוחדים נחפש search('\W') ,או הביטוי הראשון שאין בו מספרים search(r'\D')  
רווח הוא גם נחשב תו שכיח בטקסט ואותו מגדירים ב- '\s' או בלי רווח '\S' .  
  
חוץ מזה יש כמה סימונים שעוזרים לנו למצוא טקסט לפי מיקום החיפוש בטקסט, כך נוכל לראות מופעים שנראים בתחילת הטקסט, בסופו, או במיקום אחר:   
'\A' – מאתר אם החיפוש מופיע בתחילת הטקסט, למשל search(r'\AHead') בודק האם הביטוי'Head' נמצא בראש הטקסט.  
'\B'- בודק האם הביטוי נמצא אבל **לא** בתחילת הטקסט למשל search(r'\Bmiddle') או לא בסופו: search('middle\B').  
'\b' – ההפך מ-'\B', בודק האם הביטוי נמצא בתחילת הטקסט (search(r'\bHead')) או בסופה( search(r'end\b') )  
'\Z'- בודק האם הביטוי נמצא בסוף הטקסט search(r'end\Z').  
  
לפני שנמשיך, חלקכם בטח שאלתם מה זה ה-r הזה בתחילת המחרוזת, אז ה-r מסמל מחרוזת מיוחדת מסוג raw string, זאתי מחרוזת שמתעלמת מ-'\' ומתייחסת אליו כאילו הוא סוג של תו, כלומר במקום שנכתוב '\\' נוכל לכתוב רק פעם אחת עם ציון שזאתי מחרוזת r לפני.  
  
קבוצות –   
אפשר להגדיר קבוצת תווים יחד באותה השאילתה ע"י שימוש ב-'[ ]', וכל מה שיכנס לסוגריים יחשב כחלק מהקבוצה, למשל אם נרצה למצוא את המופע הראשון של האותיות a,b או האותיות הגדולות שלהן נשתמש ב-search(r'[abAB]') וזה יחזיר לנו את המופע הראשון שבו מופיעה אחת מהאותיות שציינו.  
גם טווח אפשר להגדיר עם התו- '-' בתוך הסוגריים, למשל בין האותיות a ל-k יהיה [a-k] או המספרים בין אחד ל-7 זה  
 [1-7] וכו'.  
שימו לב שהקבוצה היא קבוצת תווים ולא מילים שלמות.  
אם נרצה לשלול קבוצה נשתמש באופרטור ^ בתחילת המילה, למשל כדי למצוא מתי הפעם הראשונה בביטוי בה לא מופיע אחד מהתווים a,b או c : search(r'[^abc]').  
  
אופרטורים –

יש כמה אופרטורים לשאילתות:  
**'+' –** מופע אחד או יותר של ביטוי כלשהו, למשל המילה no או nooo… נחפש search(r'no+')  
**'\*' –** אפס או יותר מופעים של התו: למשל אם מופיע הביטוי yess… או yes או רק ye : search('yes\*').  
**'{}' –** בודק אם בטקסט יש ביטוי מסויים כשתו (או מילה) בסיומו מלווה מספר פעמים מוגדר, למשל בשביל למצוא את המילה all נוכל להשתמש בשאילתה search(r'al{2}') שמראה ש-l אמור להופיע פעמיים בדיוק.  
**'.'-** מחליף כל תו למעט '\n' ('\w' מכיל רק מספרים, אותיות לטיניות סטנדרטיות ו-'\_' , אבל לא תווים כמו '\n' וכדו').  
למשל search(' he..o') יביא כל מילה או ביטוי שעטוף באותיות he ונגמרות ב-o.  
**'^' -** מתחיל עם, כמו '\A' למשל אם הטקסט מתחיל עם המילה hello : search('^hello')  
**'$'-** נגמר עם, כמו '\Z' למשל נגמר עם המילה hello : search('hello$')  
**'|'** **–** אופרטור או, בשביל לברור בין כמה אפשרויות למשל: yes או no נעשה search('yes|no').  
**'()' –** לכידה לקבוצה אחת, כדי להחשיב כמה ביטויים יחד בקבוצה, שימושי בעיקר כשרוצים לפרק לחלקים את הביטוי אותו אנחנו רוצים למצוא, אם נחזור לדוגמא הראשונה שלנו עם מספרי הטלפון, יכולנו לפרק את הקוד לנניח קידומת והמספר עצמו, למשל: search(r'(\d\d\d)-(\d\d\d-\d\d\d\d)'), סתם ככה לא נראה איזשהו שינוי בקוד אם נשתמש בפונקציה group(), אבל עכשיו נוכל להוסיף לה פרמטר שיגדיר איזו קבוצה נרצה, למשל בשביל לקבל רק את הקבוצה הראשונה נוסיף ארגומנט עם הסיפרה אחת: group(1) זה יחזיר רק את הקבוצה הראשונה שנתפסה, במקרה זה הקידומת של המספר, או אם נבצע findall זה יחלק כל מציאה ל-tuple לפי הקבוצות שהגדרנו:

from typing import List

def find\_phone\_numbers\_regex(text:str)-> List[str]:

    import re

    phone\_pattern = re.compile(r'(\d\d\d)-(\d\d\d-\d\d\d\d)')

    phones = phone\_pattern.findall(text)

    return phones

txt = '054-444-4444 gj fdkl055-555-5555mvkdl053-545-4545'

find\_phone\_numbers\_regex(txt)  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
[('054', '444-4444'), ('055', '555-5555'), ('053', '545-4545')]

פונקציות –

בביטויים רגולרים יש כמה פונקציות שכיחות:   
הפונקציה compile() כפי שראינו מביאה לנו תבנית של ביטוי רגולרי , אבל אין חובה לקמפל כל פעם ביטוי ואפשר לבצע כל פונקציה שאנחנו מבצעים עם אובייקט תבנית ישירות, רק צריך להגדיר את הטקסט עליו אנחנו עובדים כארגומנט לפונקציה. למשל שתי הדרכים הבאות שוות:

pattern = re.compile(r'(\d\d\d)-(\d\d\d-\d\d\d\d)')

pattern\_groups = pattern.search(txt)

print(f'pattern {pattern\_groups.group()}')

direct\_groups = re.search(r'(\d\d\d)-(\d\d\d-\d\d\d\d)' , txt)

print(f'direct {direct\_groups.group()}')

חוץ מ-group אפשר לקבל גם את האינדקס של תחילת המופע הראשון וסופו עם הפונקציה span(), כמו כן כל שאילתה שאנחנו מבצעים שומרת גם את המחרוזת עליה היא פועלת במשתנה string (אם הוא לא None) :

txt = '054-444-4444 gj fdkl055-555-5555mvkdl053-545-4545'

span = re.search(r'(\d\d\d)-(\d\d\d-\d\d\d\d)' , txt)

print(span.span())

print(span.string)  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
(0, 12)   
054-444-4444 gj fdkl055-555-5555mvkdl053-545-4545

הפונקציה split() דומה לפונקציה split() של מחרוזות רק שמאפשרת לבצע את הפיצול עם regex .

phone\_number\_row = 'Tom Pythonovitz, 055-555-5555 Tammi Pythonovitz, 054-444-4444'

list\_of\_users = re.split(r', \d\d\d-\d\d\d-\d\d\d\d',phone\_number\_row)

print(list\_of\_users)  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
['Tom Pythonovitz', ' Tammi Pythonovitz', '']

עוד פונקציה שיכולה לשנות את המחרוזת היא הפונקציה sub שמשנה את השאילתה למחרוזת מוגדרת, למשל להחליף בין רווחים ל-$, ואפשר גם להוסיף פרמטר לכמה מופעים יתרחש השינוי:

txt = "The rain in Spain"

x = re.sub("\s", "9", txt)

print(x)

y = re.sub("\s", "9", txt, 2)

print(y)  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
The9rain9in9Spain   
The9rain9in Spain

טקסט