Regularni izrazi

Razbiranje podatkov iz niza (razčlenjevanje besedil) je v splošnem zahteven problem. Prav tako je težek problem prepoznavanje nizov in ugotavljanje ali ti ustrezajo določenim pogojem.

Prepoznavanje celega števila

Poskusimo napisati program, ki bo prepoznal niz, ki predstavlja celo število, in v primeru uspešnega prepoznavanja, nam bo to število tudi vrnil. Zavedati se moramo dejstva, da programu lahko načeloma podtaknemo katerikoli niz. Toda »dobri« nizi so le tisti ki:

- na začetku morda vsebujejo znak '-' ali znak '+',
- vsi preostali znaki so številke.

Program bi tako lahko napisali na naslednji način.

```
def celoStevilo(niz):
    if niz == "":
        print("Niz ne predstavlja celega števila.")
        return
    zac = 0
    if niz[0] in "+-":
        zac = 1
    if niz[zac] == "0":
        print("Niz ne predstavlja celega števila.")
        return
    for i in range(zac, len(niz)):
        if not niz[i].isdigit():
            print("Niz ne predstavlja celega števila.")
            return
    return int(niz)
```

Če na kratko opišemo delovanje programa: če je niz prazen, se izpiše sporočilo, da niz ne predstavlja celega števila in program se zaključi. Sicer ima niz vsaj en znak. Spremenljivka `zac` predstavlja indeks, na katerem se začne niz po morebitnem predznaku, in ta indeks ima vrednost 0 (ni predznaka) ali 1 (je predznak). Prvi znak po predznaku (na indeksu `zac`) ne sme biti stevka `'0'`. Preostane nam le še, da preverimo, ali so vsi znaki po indeksu `zac` stevke.

Regularni izrazi

Lahko si predstavljamo, kako bi se šele program zapletel, če bi morali razpoznati realno število. Dolgo časa nazaj so modreci ugotovili, da se je razpoznavanja nizov možno lotiti preko posebnih vzorcev, ki jih imenujemo regularni izrazi. Regularne izraze opišemo kot nize sestavljene iz zaporedja znakov in rezerviranih znakov, ki predstavljajo vzorce.

Dobesedni vzorci

Alfanumerični znaki predstavljajo vzorce, ki se ujemajo natanko s temi znaki. Oglejmo si program, ki prepozna besedo `"avto"`. Preprost program bi izgledal takole:

```
def prepoznaj(niz):
    return niz == "avto"
```

Program, ki pa bi s pomočjo regularnih izrazov prepoznal enak niz, pa izgleda nekoliko bolj zapleteno:

```
import re

def avto(niz):
    vzorec = r"avto"
    rezultat = re.match(vzorec, niz)
    return rezultat != None
```

Vzorec `"avto"` predstavlja vzorec za besedo, ki se začne s črko `a`, nadaljuje s črko `v`, potem s `t` in nato z `o`. Vzorec zapišemo v nizu, in ker ima v regularnih izrazih poševnica nazaj (`'\'`) prav tako pomen ubežnega zanka, se dogovorimo, da bomo pred vzorce vedno pisali znak `'r'`. Ta znak pred nizom predstavlja določilo, da naj se niz interpretira dobesedno (brez interpretacije ubežnih znakov). Tako je npr. niz `r"\a"` (poševnica se interpretira dobesedno) enak nizu `"\\a"` (prva poševnica je ubežni znak, druga poševnica pa pove, da želimo zapisati znak za poševnico).

A ugotovimo lahko, da tudi klica `avto("avtomobil")` ali `avto("avtomat")` vrneta `True`. Funkcija `match` namreč vrne poseben objekt, ki ga shranimo v spremenljivko `rezultat`, a le v primeru, če se začetek niza ujema z vzorcem. V nasprotnem vrne `None`. Zadnja vrstica preveri, če je v spremenljivki `rezultat` kaj ali nič (`None`).

Denimo, da želimo napisati takšen vzorec, da bi ustrezal le besedi `"avto"` in nobeni drugi besedi. V vzorcu moramo zato povedati, da je za črko `o` konec besede. Znak-vzorec za konec besede je znak `$`. Vzorec, s katerim se ujema natanko beseda `"avto"`, je oblike:

```
r"avto$"
```

Kot bomo videli kasneje, lahko z vzorci iščemo ujemanje tudi med deli besede in ne zgolj od začetka. Podobno, kot obstaja znak-vzorec za konec besede, obstaja tudi znak-vzorec za začetek besede. To je znak `'^'`. Naš natančen vzorec se torej glasi:

```
r"^avto$"
```

Znak-vzorec pika (`.`) predstavlja katerokoli črko. Tako lahko vzorec za besedo z natanko 4 črkami zapišemo kot:

```
r"^....$"
```

Poleg znakov `'.'`, `'$'` in `'^'` so rezervirani znaki (t.j. znaki s posebnim pomenom) še naslednji znaki:

```
* + ? { [ ] \ | ( )
```

Če želimo v vzorcu dobesedno enega od teh znakov, tak znak zabeležimo s pomočjo ubežnega znaka `'\'`. Torej, vzorec v katerem so trije znaki, na koncu pa je vprašaj, je takšen:

```
r"^...\?$"
```

Veliko posebnih znakov-vzorcev lahko navedemo s pomočjo ubežnega znaka. Posebej uporabni so naslednji znaki-vzorci:
`\d` – katera koli številka,
`\D` – kateri koli znak razen številke,
`\s` – kateri koli beli znak. Beli znaki so znaki kot so presledek, tabulator in skok v novo vrstico.
`\S` – kateri koli nebeli znak.
`\w` – kateri koli alfanumeričen znak ali podčrtaj,
`\W` – kateri koli znak, ki je ni alfanumerični znak in ni podčrtaj.

Poskusimo napisati vzorec za dve besedi, ki imata vsaka natanko 3 znake, vmes je beli znak, vsi znaki, ki sestavljajo besedi, pa so alfanumerični ali podčrtaji:

```
r"^\w\w\w\s\w\w\w$"
```

Hitro opazimo, da je s takim tipom vzorcev težko opisovati vzorce, ki bi opisovali npr. cela števila ali besede, ki se začnejo na črko `a`, `b` ali `c`. Za reševanje tovrstnih težav se bomo poslužili operacij na vzorcih.

Oklepaji

Oklepaji služijo za združevanje (grupiranje) vzorcev. Na skupinah lahko izvajamo operacije ali pa jih uporabimo za pridobivanje (ekstrakcijo) podnizev, kar si bomo ogledali kasneje.

Variante vzorcev

Da navedemo vzorec za znak, ki je natanko eden izmed znakov v izboru, lahko uporabimo operator oglati oklepaj. Vsi znaki-vzorci v oglatem oklepaju so naštete možnosti za vzorec, na katrega se lahko ujema predlagani niz. Primer: vse tričrkovne besede, ki se začnejo na eno izmed črk `a`, `b`, `c`, potem pa sledita alfanumerična znaka ali podčrtaja:

```
r"^[abc]\w\w$"
```

Primer: vse tričrkovne besede sestavljene le iz malih črk.

```
r"^[a-zA-Z][a-z][a-z]$"
```

Opazimo lahko, da v zaporedju v oglatem oklepaju lahko s pomočjo operatorja minus (−) naštejemo daljša zaporedja, kot so `a-z`, `A-Z` in `0-9`. Kot primer, vzorca `[abcd]` in `[a-d]` predstavljata enaka vzorca.

Variante vzorcev lahko podamo tudi z operatorjem '|'. Primer vzorca eno ali dvomestno število, pri čemer se dvomestno število ne začne z 0.

```
r"^(\d$)|([1-9]\d$)"
```

Ponavljanje in možnosti

Operacije za ponavljanje podvzorcev so naslednje:
`*` - nič ali več ponovitev najmanjšega predhodnega vzorca
`+` - ena ali več ponovitev najmanjšega predhodnega vzorca
`{m}` – natanko m ponovitev najmanjšega predhodnega vzorca
`{m, n}` – m do n ponovitev najmanjšega predhodnega vzorca
`?` - kvečjemu ena ponovitev najmanjšega predhodnega vzorca

Najmanjši predhodni vzorec je en znak-vzorec, ki se nahaja neposredno pred operatorjem za ponavljanje, oziroma cel vzorec v oklepajih, neposredno za katerim sledi operator.

Oglejmo si nekaj primerov:

Primer: vse besede sestavljene iz malih črk, vključno s prazno besedo.

```
r"^[a-z]*$"
```

Primer: vse besede sestavljene iz malih črk, brez prazne besede.

```
r"^[a-z]+$"
```

Primer: zaporedje besed sestavljenih iz alfanumeričnih znakov ali presledkov, med katerimi je natanko en presledek.

```
r"^([a-z]+ )*[a-z]+$"
```

Primer: besede iz malih črk, dolžine 5 do 7 števk.

```
r"^[a-z]{5,7}$"
```

Primer: besede, ki ponazarjajo neničelna cela števila, pred katerimi je morda predznak.

```
r"^[+-]?[0-9]+$"
```

Tak vzorec ima rahlo lepotno napako – cela števila so lahko napisana s poljubnim številom ničel na levi.

Uporaba skupin

Prepoznavanje vzorcev je en vidik dela z zapisi v nizih, velikokrat pa potrebujemo tudi pridobivanje podatkov iz vzorca. Oglejmo si naslednji primer. Radi bi prebrali vektor celih števil oblike:

```
(123123,-2,212)
```

Vzorec ima torej obliko:

```
r"^\([+-]?[0-9]+,[+-]?[0-9]+,[+-]?[0-9]+\)$"
```

Da bi lahko iz vzorca »pridobili« tri števila, dodamo dodatne oklepaje, ki združujejo podvzorce, ki predstavljajo posamezno komponento vektorja. Vzorec sedaj izgleda takole:

```
r"^\(([+-][0-9]+),([+-][0-9]+),([+-][0-9]+)\)$"
```

V vzorcu so tri skupine. Ker se skupine ne gnezdijo, so le-te oštevilčene po vrsti od 1 do 3. Če bi se oklepaji gnezdili, določimo številko skupine tako, da preštejemo vse začetne oklepaje, ki se pojavijo do vključno oklepaja, ki predstavlja skupino. Primer:

```
(....(..(.))..(..(.)...(.))
1    2  3     4  5     6
```

Naslednji program z uporabo skupin iz niza, ki ga prepozna kot vnos za vektor celih števil, generira vektor kot trojico treh celih števil:

```
def vektor(niz):
    vzorec = r"^\(([+-]?[0-9]+),([+-]?[0-9]+),([+-]?[0-9]+)\)$"
    rezultat = re.match(vzorec, niz)
    if rezultat == None:
        print("Nerazpoznaven vektor!")
        return
    x = int(rezultat.group(1))
    y = int(rezultat.group(2))
    z = int(rezultat.group(3))
    return (x,y,z)
```

Preizkusimo funkcijo in kaj hitro ugotovimo, da program ne razpozna vektorjev oblik:

```
(1,  2,  3)
 (  1, -2,3)
(1,2,3   )
```

Glede na vzorec je to čisto jasno – vzorec ne dopušča morebitnih presledkov (oz. belih znakov) okoli oklepajev in vejic. Če želimo nekoliko bolj robusten vhod za vektor, vzorec popravimo:

```
r"^\s*\(\s*([+-]?[0-9]+)\s*,\s*([+-]?[0-9]+)\s*,\s*([+-]?[0-9]+)\s*\)\s*$"
```

Iskanje podvzorcev

Poleg prepoznavanja vzorca v celotnem nizu nas velikokrat zanima iskanje pojavitev vzorcev v nizu. Primer: poiščimo vsa števila, ki se pojavijo v nizu

```
niz = "Vsaka izmed 10 rok ima 5 prstov, kar skupaj nanese 50 prstov"
```

Vemo, da je vzorec za število naslednje oblike:

```
vzorec ="[0-9]+"
```

Naslednja funkcija poišče vsa števila v nizu.

```
def poisciStevila(niz):
    vzorec = r"[0-9]+"
    rezultat = re.findall(vzorec, niz)
    return [int(i) for i in rezultat]
```

Uporabili smo funkcijo `findall`, ki za dan vzorec vrne seznam maksimalnih podnizev, ki se ujemajo z vzorcem.

Oglejmo si naslednji primer: poiščimo vse HTML značke v nizu. Program bi izgledal takole:

```
def znacke(bes):
    vzorec = "<\w+>"
    return re.findall(vzorec, bes)
```

Če želimo, da nam funkcija najde tudi zaključne značke (tiste, ki se začnejo s poševnico za znakom /), vzorec nekoliko popravimo:

```
vzorec = "</?\w+>"
```

Kot primer uporabe, si oglejmo, katere značke se pojavijo v dani datoteki.

```
def vseZnackeVDatoteki(ime):
    with open(ime, "r") as f:
        znacke = []
        vzorec = "</?\w+>"
        for vrstica in f:
            sez = re.findall(vzorec, vrstica)
            znacke += sez
    return znacke
```

Če še želimo ugotoviti, koliko je dejansko različnih značk, program nadgradimo takole:

```
def vseRazlicneZnackeVDatoteki(ime):
    znacke = vseZnackeVDatoteki(ime)
    return list(set(znacke))
```

Delitev nizov na besede

Namesto, da bi iskali pojavitve podvzorcev in potem te vrnemo v seznamu, lahko iščemo te pojavitve, a potem v seznamu vrnemo besede, ki jih dobimo z odstranitvijo teh vzorcev. Primer uporabe: če za vzorec izberemo poljubno neprazno zaporedje belih znakov in ločil:

```
vzorec = r"[\s,.?!;]+"
```

potem nam funkcija

```
def seznamBesed(niz):
    vzorec = r"[\s,.?!;]+"
    return re.split(vzorec, niz)
```

Razbije niz na besede. Primer uporabe

```
>>> seznamBesed("Kdor drugemu  ....   jamo koplje,   sam vanjo pade!")
['Kdor', 'drugemu', 'jamo', 'koplje', 'sam', 'vanjo', 'pade']
```

Ukaz split torej poišče maksimalne vzorce, vendar potem ne vrne v seznamu pojavitve teh, ampak vrne drobitev začetnega niza, če iz njega odstranimo najdene vzorce.

Zamenjava vzorcev

Poglejmo si program, ki v danem nizu poišče vsa cela števila in jih »prečrta« z znaki `X`.

```
import re

def precrtajStevke(niz):
    vzorec = "[0-9]+"
    return re.sub(vzorec, funkcijaZaZamenjavo, niz)

def funkcijaZaZamenjavo(rezultat):
    besedilo = rezultat.group(0)
    return len(besedilo)*"X"
```

V programu smo uporabili funkcijo `sub`, ki nam zamenja najdene vzorce. Kot drugi argument funkcije sub lahko podamo fiksen niz ali pa funkcijo, ki sprejme rezultat oblike (objekt), kot jo vrne funkcija `match`. Iz tega objekta lahko s pomočjo metode group dobimo celoten niz, ki se ujema z vzorcem (`group(0)`) ali morda podskupine (podvzorce). Ujemajoči se podniz tako obdelamo in vrnemo rezultat, ki nadomesti ujemajoči se podniz v začetnem vzorcu. Program deluje takole.

```
>>> precrtajStevke("Janez je 20 dni prejemal 100 evrov na dan in 5 dni
200 evrov na dan")
'Janez je XX dni prejemal XXX evrov na dan in X dni XXX evrov na dan'
```