

# **Programiranje I**

*izpiski s predavanj*

*Nace Kovačič*

## TO DO

- regularni izrazi - sklici na skupine
- zajem podatkov - normalizacija podatkov
- analiza podatkov - transformacija razpredelnic
- analiza podatkov - vektorizacija
- uvod v ocaml - parametrični polimorfizem
- funkcijsko programiranje - anonimne funkcije
- funkcijsko programiranje - delna aplikacija funkcij in curryirane funkcije
- funkcijsko programiranje - curryiranje

## regularni izrazi

neko zaporedje znakov, ki opisuje vzorec v besedilu

- obstajajo boljša orodja za pridobivanje podatkov s spletnih strani (*beautiful soup*)
- v VSCodu lahko iščemo *navadno besedilo* ali pa *regularne izraze*, ki so podani v iskalniku (če vključimo možnost `.*`)

### legenda osnovnih znakov

.	.....	katerikoli znak
\.	.....	pika
\\\	.....	backslash
\d	.....	števka <sup>1</sup>
\D	.....	komplement števk
\w	.....	alfanumerični znak
\W	.....	komplement alfanumeričnih znakov
\s	.....	whitespace <sup>2</sup>
\S	.....	komplement whitespace -a
.DOTALL	.....	pojdi tudi čez nove vrstice

- če želimo, da se vzorec ponovi vsaj *n - krat*, ga damo v *zavite oklepaje*
- v *oglate oklepaje* vstavimo znake, ki jih želimo poiskati
- če želimo, da določenih znakov ne najde, uporabimo `[^znaki]`
- če želimo poiskati več besed, jih damo v *oglate oklepaje*, mednje pa vstavimo `|`
- če želimo poiskati vse znake med dvema vrednostima uporabimo *vezaj*
- če mora biti znak na *začetku*, napišemo predenj `^`
- če mora biti znak na *koncu*, napišemo na konec `$`

---

<sup>1</sup>enakovredno izrazu **[0-9]**

<sup>2</sup>presledek, tab, etc.

### *ponovitve elementov*

? .....	0 ali 1
plus .....	1 ali več
*	0 ali več <sup>3</sup>

### *skupine*

nastanejo, ko damo nekaj v oklepaje

- skupin je lahko več
- lahko jih poimenujemo (*?P<naslov>*)

### *json*

- podatke o filmih je pametno shraniti v *json* datoteko
- *standarden tekstovni format*, ki izhaja iz java-scripta
- podpira objekte
- manjkajočo vrednost označimo z *null*
- iz *Pythona* v *json* dobimo s funkcijo **json.dumps**
- iz *jsona* v *Python* dobimo s funkcijo **json.loads**

### *csv*

- bolje od *jsona* za statistično analizo, saj so podatki podani tabelarično
- v zgornji vrstici so navadno *naslovi stolpcov*
- vsaka vrstica predstavlja svoj *podatek* <sup>4</sup>

### *Pythonove knjižnice za obdelavo podatkov*

#### *re*

<b>re.findall(vzorec, besedilo)</b> .....	poišče vse ponovitve vzorca v besedilu
<b>re.groupbydict</b> .....	naredi slovar naših skupin
<b>re.groupbyiter</b> .....	iterator <i>groupdict</i>
<b>re.finditer</b> .....	PREVERI, KAJ NAREDI

#### *requests*

<b>requests.get(url)</b> .....	dobimo odziv spletne strani <sup>5</sup>
<b>requests.get.(url).text</b> .....	vrne <i>vsebino</i> spletne strani

<sup>3</sup>+ in \* imata *skromni* in *požrešni* način

<sup>4</sup>recimo vsak film je v svoji vrstici

<sup>5</sup>dobimo objekt tipa *Response*

### *csv*

`csv.DictWriter(csv, fieldnames)` ..... slovar zapišemo v ustreznata polja  
`csv.DictWriter(csv, fieldnames).writeheader()` ..... izpiši glavo

### *pandas*

*knjižnica za analizo podatkov*

- pogosto uporabljamo skupaj z *Jupyterjem*
- *Jupyter* je vizualno okolje za delo analizo podatkov v Pythonu<sup>6</sup>
- *Jupyter notebook* je sestavljen iz več celic<sup>7</sup>

`pd.read_csv` ..... prebere csv datoteko in prikaže tabelo  
`pd.read.head(n)` ..... dobimo *prvih n* elementov  
`pd.read.tail(n)` ..... dobimo *zadnjih n* elementov  
`pd.read[ime_stolpca]` ..... dostopamo do posameznega stolpca  
`pd.read[seznam imen]` ..... dostopamo do razpredelnice stolpcov  
`pd.read.iloc[i]` ..... dostopamo do vrednosti na *i - tem* mestu  
`pd.read.loc[k]` ..... dostopamo do ključa *k*  
`pd.read.groupby` ..... razpredelnice, kjer so vrstice združene glede na lastnost  
`pd.read.count(pogoj)` ..... štejemo ponovitve  
`pd.read.size()` ..... pove, koliko je *vnosov*, ne glede na dane podatke  
`pd.read.merge(stolpci)` ..... dve tabeli *združi* po stolpcih z istimi imeni  
`pd.read.join()` ..... PREVERI  
`pd.read.mean()` ..... izračuna *povprečje*  
`pd.read.sort_values()` ..... uredi po velikosti

Vrednosti v stolcih lahko tudi *filtriramo z logičnimi vrednostmi*. Stolpce lahko tudi urejamo po vrednostih (*naraščajoče* ali *padajoče*).

Grafe rišemo z *%matplotlib inline*, tako, da uporabimo `.plot()`. *Razsevni diagram* dobimo z `plot.scatter()`.

---

<sup>6</sup>prednost je v tem, da lahko Python datoteko izvajamo sproti, da nam ni treba ponovno restartat vsakič, ko kaj spremenimo

<sup>7</sup>koda (*Python*) in besedilo (*Markdown*)

*naivni Bayesov kvantifikator*

*enostaven primer strojnega učenja*

- odločamo se, ali nekaj pripada nekemu razredu ali ne<sup>8</sup>

Zanima nas verjetnost dogodka  $D_i$ , ob pogoju, da vsebuje korene  $K_1, \dots, K_n$ .

$$P(D_i | K_1 \cap \dots \cap K_n)$$

Uporabimo **Bayesov izrek**

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Velja

$$P(D_i | K_1 \cap \dots \cap K_n) = \frac{P(K_1 \cap \dots \cap K_n | D_i) \cdot P(D_i)}{P(K_1 \cap \dots \cap K_n)}$$

Ker je klasifikator naiven, velja

$$P(K_1 \cap \dots \cap K_n | D_i) = P(K_1 | D_i) \cdot \dots \cdot P(K_n | D_i)$$

oz.

$$P(D_i | K_1 \cap \dots \cap K_n) = \frac{P(K_1 | D_i) \cdot \dots \cdot P(K_n | D_i) \cdot P(D_i)}{P(K_1 \cap \dots \cap K_n)}$$

---

<sup>8</sup>uporabno za *spam*

## OCaml

*funkcijsko programiranje*

- vrednosti definiramo z *let*
- presledki niso pomembni
- komentarje pišemo v *oklepaje* in *zvezdice*
- v interaktivni konzoli damo na koncu *;;*
- ima zelo rad *type*
- vrstni red je vedno ime\_funkcije prva\_spremenljivka druga\_spremenljivka  
<sup>9</sup>
- v funkciji lahko definiramo *lokalne definicije*
- *char* ima *enojne* narekovaje, *string* pa *dvojne*
- lahko definiramo več stvari hkrati z *and* <sup>10</sup>
- deljenje je *celoštevilsko* <sup>11</sup>
- *logične vrednosti* so tipa *bool*
- če želimo funkcijo definirati po *kosih*, uporabimo *match*
- pri *match* je *vrstni red* pomemben
- *rekurzivne funkcije* definiramo z *let rec*
- *nabore* pišemo kot *n-terice*
- *prazen nabor* imenujemo *unit*
- *seznamy* so *homogeni* <sup>12</sup>
- z :: *lepimo* sezname <sup>13</sup>

---

<sup>9</sup>temu pravilu pravimo *aplikacija*

<sup>10</sup>DIFFERENCE BETWEEN AND AND IN

<sup>11</sup>za *float* dodamo operatorju piko na konec

<sup>12</sup>vsi elementi imajo *enak* tip

<sup>13</sup>funkcija je *desno asociativna*

## *funkcijsko programiranje*

- tipi v OCamlu so navadno *statični*<sup>14</sup>
- OCaml ima od Pythona *bogatejše tipe*
- klic funkcije je *repen*, če se izvede *zadnji*
- funkciji, kjer so vsi klici *repno rekurzivni* pravimo *repno rekurzivna funkcija*
- Python je *proceduralni*<sup>15</sup> in *imperativni*<sup>16</sup> jezik
- OCaml je *funkcijski* in *deklarativeni*<sup>17</sup>

## *funkcije višjega reda*

*funkcije, ki sprejemajo druge funkcije*

- če funkciji dveh elementov podamo zgolj en element, dobimo *novo* funkcijo, ki še čaka na drugi argument<sup>18</sup>
- postopek *curryiranja*<sup>19</sup>

### *fold*

#### *fold right*

- sprejme tri spremenljivke<sup>20</sup>
- če imamo *prazen seznam*, vrnemo *začetno vrednost*, če je seznam tipa  $x :: xs$ , funkcijo rekurzivno uporabimo na tem repu in tej vrednosti dodamo  $x$

#### *fold left*

- repno rekurzivna

---

<sup>14</sup>razlika med *statičnimi* ali *dinamičnimi* tipi je v tem, da se statični tipi preverijo, preden se funkcija požene

<sup>15</sup>programe pišemo kot zaporedja ukazov

<sup>16</sup>z ukazi spreminjaamo spremenljivke

<sup>17</sup>opišemo stanje

<sup>18</sup> $f x y = (f x) y$

<sup>19</sup>PREVERI

<sup>20</sup>naš seznam, začetno vrednost, funkcijo f

test