Unione europea
Fondo sociale europeo

RegioneEmilia-Romagna

RETE POLITECNICA
La tecnica per crescere

Materiale didattico per partecipante al corso "**TECNICO ESPERTO NELL'ANALISI E NELLA VISUALIZZAZIONE DEI DATI**" – Rif.P.A. 2021-15998/RER – approvata con DGR n. 1263 del 02/08/2021 di IFOA – Istituto Formazione Operatori Aziendali

REGEX

- Le espressioni regolari (RE, Regex o Regex pattern) sono un mini linguaggio di programmazione altamente specializzato incorporato in Python e reso disponibile attraverso il modulo re (import re)

- Usando le REGEX si specificano le regole per l'insieme di possibili stringhe che si desidera abbinare
  - Ad esempio: set di frasi inglesi, o indirizzi e-mail, o comandi TeX, o qualsiasi cosa tu voglia

REGEX

- Il linguaggio delle RE è relativamente piccolo -> non tutte le attività di elaborazione delle stringhe possono essere eseguite utilizzando le RE
- Rendono ricerca e sostituzione del testo enormemente più potenti
- Soddisfano il principio 80/20
- Valide in ogni linguaggio e indipendenti dalla piattaforma

REGEX – validatore online

- Al seguente link trovate una pagina di test dove testare le vostre Regex:

https://regex101.com/

Corrispondenza tra caratteri

- Poiché vengono utilizzate per operare sulle stringhe, inizieremo con l'attività più comune: corrispondenza dei caratteri
- La maggior parte delle lettere e caratteri semplicemente corrispondono a sé stessi
- Alcuni caratteri, detti metacaratteri, non corrispondo a sé stessi (li vedremo dopo)

- La regex più semplice?
  - bank
- Cerca esattamente la parola bank

Cornice []

- Rappresenta 1 carattere
- I caratteri tra [] sono legati da una relazione di tipo OR, l'ordine non conta

- Esempi:
  - [bt]ank // Cerca sia bank che tank
  - [abc]1 // trova a1, b1 o c1
  - [cba]1 // trova a1, b1 o c1
  - file[0123456789] // trova file0,file1,file2 ... o file9

Intervalli: -

- Utilizzando il carattere trattino: '-' definiamo un intervallo, o range
- E' una scorciatoia:
  - Anziché scrivere [abcdefghijklmnopqrstuvwxyz]
  - Basta scrivere [a-z]

  - Anziché scrivere [0123456789]
  - Basta scrivere [0-9]

  - file[0-9] // trova file0,file1,file2 ... o file9
  - [a-z] // trova a, b, c oppure ... z

Quantificatori

quantificatore	significato	regex	esempio
?	zero o 1	abc?	ab, abc
*	zero o più	abc*	ab,abc,abcc,abccc,abcccc,... etc
+	uno o più	abc+	abc,abcc,abccc,abcccc,...etc
{n}	esattamente n volte	abc{2}	abcc
{n,m}	da n a m volte	abc{2,3}	abcc,abccc

Quantificatori

- Agisce sul carattere che si trova direttamente alla sua sinistra
- Può essere affiancato a qualunque carattere ma anche a una cornice
- Esempi:
  - [bt]anks? // trova bank, tank, banks oppure tanks
  - [bt]?ank // trova bank, tank oppure ank
  - ab?c? // trova a, ab, abc oppure ac
  - ca*t // trova ct, cat, caat, caaat e così via
  - ca+t // trova cat, caat, caaat e così via
  - ab{1,3}c // trova abc, abbc e abbbc

Quantificatori ‘non golosi’

- I quantificatori normali sono “golosi” (in inglese greedy), cioè cercano l’occorrenza il più grande possibile.
- Problema: se utilizzo una espressione del tipo /”.*”/ troverò tutte le parole racchiuse tra doppi apici? Purtroppo no!

- Esempio:
  - testo = 'class="pluto" id="pippo“’;
  - virgolette = re.findall'/".*"/’, t);
  - Troverà un'unica occorrenza: "pluto" id="pippo"

Quantificatori ‘non golosi’

- Come vedete non è il risultato sperato! Come fare quindi?
- Basta aggiungere un punto interrogativo alla fine dei quantificatori!
- Esempio:
  - testo = 'class="pluto" id="pippo“’;
  - virgolette = re.findall(‘/".*?"/’, t);
  - Ora troverà "pluto" e "pippo"

- Questo vale per qualsiasi quantificatore descritto in precedenza!

Negazione: [^]

- La negazione si applica a tutti i caratteri della cornice in cui compare l'operatore ^
- Non è possibile limitarla solo ad alcuni

- Esempi:
  - [^a] // trova qualunque carattere eccetto la lettera a
  - [^0-9] // trova qualunque carattere non numerico

Sequenze speciali o Alias

- Alcuni range sono talmente frequenti che sono state create delle scorciatoie (alias) dedicate
- Non sono quindi indispensabili, si possono ottenere gli stessi risultati usando la cornice in modo esteso

alias	significato	coorrisponde a
\d	digit (numero)	[0-9]
\w	word (parola)	[a-zA-Z0-9_] Include il carattere underscore
\s	spazio, tab o newline	[\t\r\n]
\D	qualsiasi non numerico	^\d
\W	quasiasi non alfanumerico	^\w
\S	quasiasi ma non lo spazio	^\s

Metacaratteri: punto .

- Significa qualsiasi carattere ad eccezione di quelli che identificano una riga nuova (\n e \r)

- Esempio:
  - testo = "espressioni regolari!”
  - risultato = re.findall(‘.’, testo)
  - Troverà tutti i caratteri

Metacaratteri: inizio riga ^

- Significa inizio di una riga
- (fate attenzione: ^ all'inizio di un gruppo significa negazione)

- Esempio:
  - testo = "espressioni regolari! "
  - risultato = re.findall('^.', testo)
  - Troverà qualsiasi carattere a inizio riga, quindi solamente la lettera 'e'

  - ^ A # trova solo il testo che inizia per A

Metacaratteri: fine riga $

- Significa fine di una riga

- Esempio:
  - testo = "espressioni regolari! "
  - risultato = re.findall(‘.$’, testo)
  - Troverà qualsiasi carattere a fine riga, quindi solamente iil punto esclamativo ‘!’

  - A$ # trova solo il testo che finisce per A

Confine di parola: \b

- l confine \b definisce dove la parola comincia e finisce
- Immaginiamo di voler cercare le parole ‘for’ e ‘she’:
  - (for|she) # trova tutte le occorrenze di for e she, ma quindi anche before
- Potremmo tentare cercando solo le occorrenze precedute e succedute da uno spazio:
  - [ ](for|she)[ ] # meglio non seleziona più before ma se c’è una frase con dentro ‘for she’ trova solo il ‘for’ e non il ‘she’ poichè lo spazio che precede ‘she’ è già stato rintracciato come spazio che segue il ‘for’
- Soluzione corretta:
  - \b(for|she)\b # trova she oppure for

Metacaratteri: backslash \

- Rappresenta il carattere di escaping
- Capita di dover cercare proprio il punto '.' oppure il carattere '['
- Poiché fanno parte della sintassi delle Regex occorre precederli col carattere '\'

- Esempi:
  - \. // trova il carattere punto
  - * // trova tutti gli asterischi
  - \\ // trova tutti i backslash

Metacaratteri: |

- Rappresenta la condizione OR
- Se A e B sono espressioni regolari A|B troverà corrispondenza in qualunque stringa corrisponda ad A oppure a B
- Ha un ordine di precedenza molto basso

- Esempi:
  - Brad|Angelina Pitt //trova o ‘Brad’ o ‘Angelina Pitt’

Metacaratteri: ()

- Le parentesi tonde servono per definire dei gruppi di caratteri
- Servono per chiarire che non vogliamo trovare tutte le occorrenze della parte di regex alla loro sinistra
- Nei gruppi è possibile utilizzare l'espressione logica OR per poter ricercare una serie di caratteri o un'altra
- E' possibile combinarlo con altri operatori (ad es. i quantificatori)

Metacaratteri: ()

- Esempi:
  - (Brad|Angelina) Pitt # trova ‘Brad Pitt’ o ‘Angelina Pitt’
  - (dog)+ # trova dog,dogdog,dogdogdog e così via
  - java(bean)? # trova java o javabean

Asserzioni

- Per specificare delle RE più complesse
  - Ad. Es: ricercare solo quelle parole che iniziano con la lettera “c” ma che la seconda lettera non sia una vocale
- (?=pattern) Asserzione lookahead positiva.
  - Valida l’espressione precedente solo se la condizione pattern è verificata
- Esempio:
  - Lodovica (?=Marchesi) # corrisponderà con Lodovica ' solo se seguito da ‘Marchesi'

Asserzioni

- Esercizio: Cercare nella stringa «cane crotalo canarino criceto cervo daino dromedario» solo le parole che iniziano con la c solo se seguite da una vocale

Asserzioni

- Soluzione:

```
testo =  "cane crotalo canarino criceto cervo daino dromedario"
print(re.findall('\\b(c(?=[aeiou])\w+)', testo))
```

Asserzioni

- (?!pattern) Asserzione lookahead negativa.
  - Valida l'espressione precedente solo se la condizione pattern NON è verificata
- Esempio: verificare il nome di un file la cui estensione non sia 'bat'
- Assumiamo nomeFile.estensione
- Con una asserzione negativa è semplicissimo:

  .*[.](?!bat$).*$ (oppure .*\.(?!bat$).*$)

  - Il $ a fine riga serve per assicurarsi che tutto il resto della stringa sia stato incluso nell'estensione

Asserzioni

- Esercizio: Cercare nella stringa «cane crotalo canarino criceto cervo daino dromedario» solo le parole che iniziano con la c solo se non sono seguite da una vocale

Asserzioni

- Soluzione:

```
testo =  "cane crotalo canarino criceto cervo daino dromedario"
print(re.findall('\\b(c(?![aeiou])\w+)', testo))
```

Asserzioni

- (?<=pattern) : asserzione lookbehind positiva
- Valida l'espressione successiva solo se la condizione pattern è verificata
- Esempio: trovare le parole che finiscono per 'sto' la cui lettere precedente sia 'a'
- Soluzione:

```
testo =  "cesto pasto fasto pesto costo"
print(re.findall('(\w+(?<=a)sto)', testo))
```

Asserzioni

- (?<!pattern) : asserzione lookbehind negativa
- Valida l'espressione successiva solo se la condizione pattern NON è verificata
- Esempio: trovare le parole che finiscono per 'sto' la cui lettere precedente non sia 'a'
- Soluzione:

```
testo =  "cesto pasto fasto pesto costo"
print(re.findall('(\w+(?<!a)sto)', testo))
```

Modifiers o flags

- I flag consentono di modificare alcuni aspetti del funzionamento delle espressioni regolari
- Sono disponibili nel modulo re sotto due nomi, un nome lungo come IGNORECASE e una forma breve di una lettera come I
- Possono essere specificati più flag, separati da OR bit a bit
- Esempio:
  - re.I | re.M # imposta sia i flag I che M

Modifiers o flags

Flag	Significato
DOTALL, S	Fa sì che il punto . matchi anche gli a capo
IGNORECASE, I	Rende in match case-insensitive
MULTILINE, M	Multi-line matching, influenza ^ and $
VERBOSE, X	Abilita i RE dettagliati, che possono essere organizzati in modo più pulito e comprensibile.
UNICODE, U	MaRende diversi escape come \w, \b, \se e \d dipendenti dal database dei caratteri Unicode.

REGEX – Python metodi utili

- re.compile(regex, flags)
- Per compilare una espressione regolare il un’istanza RegexObject
- Usato quando devo utilizzare la regex più volte
- La regex è passata al metodo compile come stringa

- Esempio:

```
import re
 p =   re.compile('ab*', re.IGNORECASE)
 #  trova a, ab, abb, A, AB, ABB, AbB, …
```

REGEX – Python metodi utili

- match() Determina se la RE corrisponde all'inizio della stringa.
- search() Ricerca all'interno di una stringa, trovando tutte le posizioni corrispondenti alla RE.
- findall() Trova tutte le sottostringhe corrispondenti alla RE, e le restituisce in una lista.
- finditer()Trova tutte le sottostringhe corrispondenti alla RE, e le restituisce in un iteratore.

- Se hanno successo restituiscono un’istanza MatchObject contenente informazioni riguardo alla corrispondenza: dove inizia e dove finisce, la sottostringa a cui corrisponde e altro.

REGEX – Python metodi utili

- Per interrogare il MatchObject:
  - group() restituisce la stringa corrispondente alla RE
  - start() restituisce la posizione iniziale della corrispondenza
  - end() restituisce la posizione finale della corrispondenza
  - span() restituisce una tupla contenente la (start, end) posizione della corrispondenza

REGEX – Python metodi utili

- Esempio:

```
line =  "Cats are smarter than dogs“
matchObj =  re.match( r'(.*) are (.*?) .*', line, re.M|re.I)
if matchObj:
   print "matchObj.group() : ", matchObj.group()
   print "matchObj.group(1) : ", matchObj.group(1)
   print "matchObj.group(2) : ", matchObj.group(2)
else:
   print "No match!!"
```

REGEX – Python metodi utili

- Per modificare una stringa:
  - split() divide la stringa in una lista, dividendola dove è valida la RE
  - sub() trova tutte le sottostringhe per cui la RE sia valida e le sostituisce con una stringa diversa
  - subn() fa la stessa cosa ma torna anche il numero di sostituzioni

- Esempio:
  - s = '100 NORTH MAIN ROAD'
  - re.sub('ROAD$', 'RD.', s)
  - //'100 NORTH BROAD RD.' sostituisce ROAD con RD solo se si trova alla fine della stringa

Ricerca e sostituzione

- Qui la pagina di test non ci può aiutare, occorre Pycharm
- Supponiamo di avere un file composto da 100 righe come queste

  31-01-10_backup32

  24-01-10_backup1

  24-02-10_backup_mona

  11-03-09_backup_lisa
- Vogliamo sostituirle dal formato europeo (gg-mm-aa) a quello americano (mm-gg-aaaa)
- Nota: supponiamo ogni data dal 2000 in poi

Ricerca e sostituzione

\d{2}-\d{2}-\d{2}_backup.* // trova le nostre righe

- Per ogni riga desideriamo sostituire aree specifiche quindi ricorriamo all'operatore di raggruppamento

(\d{2})-(\d{2})-(\d{2})_backup(.*) // ci siamo

- A questo punto tutto quel che dobbiamo fare è sostituire le righe trovate con:

{Gruppo2}-{Gruppo1}-20{Gruppo3}_backup{Gruppo4}

- Il che si traduce nella seguente espressione di sostituzione

\\2-\\1-20\\3_backup\\4

Ricerca e sostituzione

- Codice Python:

```
import re
testo =  " 31-01-10_backup32 \n 24-01-10_backup1 \n 24-02-10_backup_mona \n 11-03-09_backup_lisa"
print("Testo con date in formato: gg-mm-aa:\n" +  testo)
testo =  re.sub(r'(\d{2})-(\d{2})-(\d{2})_backup(.*)', '\\2-\\1-20\\3_backup\\4', testo)
print("Testo con date in formato: mm-gg-aaaa: \n" +  testo)
```

Esercizio: indirizzo email

- Regole per cui una stringa possa essere considerata una email:
  - Il nome utente può contenere lettere, numeri, underscore e trattini ma deve iniziare con una lettera
  - Ci deve essere una chiocciola
  - Il dominio può contenere solo lettere seguite da un punto seguito da altre lettere

- Ricordate:
  - Quantificatori
  - Escaping

Esercizio: indirizzo email

- Soluzione:
  - [a-z][\w-]*@[a-z]+\.[a-z]+

- [a-z] : indica che vogliamo una lettera iniziale
- [\w-]* : seguita da 0 o più (*) caratteri alfanumerici (\w) o trattini (-)
- @ : esattamente il carattere @
- [a-z]+ : 1 o più lettere
- \. : esattamente il carattere punto
- [a-z]+ : 1 o più lettere

Esercizio: indirizzo email

- Supponete di voler aggiornare le regole in modo da convalidare solo i domini più importanti
- L'indirizzo email deve finire con 'com' oppure 'net'

- Ricordate:
  - Il concetto di Gruppi
  - Operatore OR

Esercizio: indirizzo email

- Soluzione:
  - [a-z][\w-]*@[a-z]+\.(com|net)

Esercizio: numero di telefono

- Regole per cui una stringa possa essere considerata unnumero di telefono:
  - Il numero deve iniziare con 3 numeri, il primo deve essere un 3
  - Seguito da un trattino
  - Seguito da 7 caratteri numerici
  - Es.: 348-1234567

Esercizio: numero di telefono

- Soluzioni:
  - Utilizzando gli alias:
    - 3\d{2}-\d{7}
  - Senza alias
    - 3[0-9]{2}-[0-9]{7}

Esercizio: Sito

- Regole per cui una stringa possa essere considerata un sito:
  - Deve iniziare con www
  - Seguito da un punto
  - Seguito da caratteri alfanumerici
  - Seguito da
  - Es.: 348-1234567

Esercizio: numero di telefono

- Soluzioni:
  - Utilizzando gli alias:
    - 3\d{2}-\d{7}
  - Senza alias
    - 3[0-9]{2}-[0-9]{7}

Riferimenti

- Esercizi: https://www.w3resource.com/python-exercises/re/index.php