lezione11

October 30, 2023

1 Fondamenti di Programmazione

Andrea Sterbini

lezione 11 - 30 ottobre 2023

2 RECAP: Files

```
# apro il file e metto nella variabile F l'oggetto che ottengo
with open(<filename>, mode=<mode>, encoding='utf8') as F:
     codice che legge/scrive il file F
     # uscendo dal with il file viene chiuso automagicamente
```

Una volta aperto il file **F** potete usare i metodi: - `F.read()` legge tutto il contenuto (o una sua parte) - `F.readline()` legge UNA riga (compreso `'\n'`) - `F.readlines()` legge TUTTE le righe (compresi `'\n'`) e torna una lista - `F.write(<string>)` scrive un testo nel file (dovete aggiungere voi `'\n'`) - `print( ..., file=F)` stampa nel file (molto comodo) - `F.seek(0)` torna alla posizione 0 (inizio)

3 ANALISI: Come trovare il file più importante dato un gruppo di parole cercate

PURTROPPO Le parole più frequenti sono in genere troppo comuni e poco significative (con poco contenuto semantico)

Esempio: articoli, verbi ausiliari, preposizioni, avverbi, parole comuni ...

Quindi una parola sarà "interessante" se **appare spesso in un file** ma **appare in pochi file**. Calcoliamo: - appare spesso in un file: **Term Frequency** (tf) la frequenza percentuale della parola nel file - appare in pochi file: **Inverse Document Frequency** (idf) la percentuale di file che la contiene

Vedi https://en.wikipedia.org/wiki/tf-idf

3.1 TF: Term Frequency (frequenza % di una parola in ciascun file)

- per ciascun file
  - contiamo le parole
  - dividiamo per il numero totale di parole

[142]:
```
# ho un elenco di documenti con il loro encoding, ottenuto unzippando files.zip
files : dict[str,str] = {
    'files/holmes.txt'       : 'utf-8-sig',
    'files/alice.txt'        : 'utf-8-sig',
    'files/frankenstein.txt' : 'utf-8-sig',
    'files/alice_it.txt'     : 'latin',
    'files/prince.txt'       : 'utf-8-sig'
    }
```

[143]:
```
# per estrarre le parole da un file che contiene anche segni di interpunzione
def estrai_parole(filename : str, encoding : str) -> list[str]:
    # apro il file in lettura e ne leggo il contenuto
    with open(filename, encoding=encoding) as FILE:
        text = FILE.read()
        # se voglio lo converto in lowercase (ma potrei preferire di no)
        text = text.lower()
        # calcolo l'insieme dei caratteri presenti nel testo
        caratteri = set(text)
        # ne estraggo solo i caratteri non-alpha e creo un dizionario per␣
 ↪translate
        nonalfa = { c: ' ' for c in caratteri if not c.isalpha() }
        # li rimpiazzo tutti con spazio con translate e maketrans
        text = text.translate(str.maketrans(nonalfa))
        # uso split sul testo in cui ho cambiato tutti i non-apha in spazi
        return text.split()

# esempio: parole del libro Alice nel paese delle meraviglie
parole = estrai_parole('files/alice_it.txt', 'latin')

### NOTA: come usare meno memoria? lavorando riga per riga
print(parole[:30])
```

```
['charles', 'lutwidge', 'dodgson', 'alice', 'nel', 'paese', 'delle',
'meraviglie', 'questo', 'e', 'book', 'è', 'stato', 'realizzato', 'anche',
'grazie', 'al', 'sostegno', 'di', 'e', 'text', 'editoria', 'web', 'design',
'multimedia', 'http', 'www', 'e', 'text', 'it']
```

[144]:
```
# per contare quante sono le parole in percentuale
def conta_parole(parole : list[str] ) -> dict[str,float]:
    # trovo le parole uniche usando un insieme
    parole_uniche = set(parole)
    # e costruisco il dizionario parola:conteggio usando count  ### INEFFICIENTE!
 ↪!!
    N = len(parole)
    return { parola: parole.count(parola)/N for parola in parole_uniche }

# rifaccio i conti sulle parole di Alice
```

```
conteggi = conta_parole(parole)

print('alcuni conteggi:', list(conteggi.items())[:20],'\n')

# le 50 parole più presenti in Alice sono:
più_presenti = sorted(conteggi, reverse=True, key=lambda K: conteggi[K])[:50]
print('le 50 più presenti sono:', più_presenti)
```

```
alcuni conteggi: [('gli', 0.0035661476907019223), ('nembo',
4.348960598416978e-05), ('affaticata', 4.348960598416978e-05), ('volete',
8.697921196833956e-05), ('gioiosa', 4.348960598416978e-05), ('affollarono',
4.348960598416978e-05), ('superba', 4.348960598416978e-05), ('levandosi',
0.00013046881795250934), ('starnutivano', 4.348960598416978e-05), ('buia',
4.348960598416978e-05), ('letterarie', 4.348960598416978e-05), ('pigliatelo',
4.348960598416978e-05), ('clivi', 4.348960598416978e-05), ('destò',
4.348960598416978e-05), ('cuore', 8.697921196833956e-05), ('camminano',
4.348960598416978e-05), ('negl', 4.348960598416978e-05), ('diverse',
0.00013046881795250934), ('germogliate', 4.348960598416978e-05), ('giornate',
4.348960598416978e-05)]

le 50 più presenti sono: ['e', 'il', 'la', 'di', 'che', 'non', 'un', 'alice',
'a', 'si', 'disse', 'in', 'ma', 'con', 'le', 'per', 'è', 'una', 'se', 'era',
'i', 'l', 'più', 'come', 'rispose', 'd', 'da', 'perchè', 'del', 'mi', 'gli',
'al', 'regina', 'della', 'così', 'lo', 'quando', 'poi', 'o', 'aveva', 'cosa',
're', 'sono', 'io', 'questo', 'ne', 'tu', 'tutti', 'testuggine', 'qualche']
```

3.1.1 Vedete che le parole più presenti sono anche le più comuni e meno significative

[145]:
```
# frequenza delle parole nel file iesimo
#    - tf_i  = # presenze / # parole del file
# per tutti i file costruiamo il dizionario del numero di files in cui appaiono
frequenze : dict[str,dict[str,float]]= {}      # conteggio delle parole nei file:
 ↪ frequenze[name][parola] -> % di parole

# conto le frequenze di tutte le parole contenute in ciascun file
for filename, encoding in files.items():
    print(filename)
    parole = estrai_parole(filename, encoding)
    conteggio = conta_parole(parole)
    frequenze[filename] = conteggio
```

```
files/holmes.txt
files/alice.txt
files/frankenstein.txt
files/alice_it.txt
files/prince.txt
```

3.2 DF: Document Frequency (% di documenti che contengono la parola)

[146]:
```
# conto i file che contengono una parola
# presenze[parola] -> # di file che la contengono
presenze : dict[str,float] = {}
for filename in files:
    for parola in frequenze[filename]:
        presenze[parola] = presenze.get(parola, 0) + 1

# divido per il numero di file per avere la presenza %
Nfile = len(files)
for parola in presenze:
    presenze[parola] /= Nfile

print('esempio delle DF:', list(presenze.items())[:30])
print('\nparole presenti in tutti i file:', *(p for p,f in presenze.items() if f␣
 ↪== 1))
```

```
esempio delle DF: [('clue', 0.4), ('trincomalee', 0.2), ('picture', 0.8),
('relatives', 0.4), ('amateur', 0.2), ('whoever', 0.6), ('anybody', 0.2),
('through', 0.8), ('draught', 0.4), ('being', 0.8), ('resemblance', 0.6),
('same', 0.8), ('famous', 0.4), ('damage', 0.8), ('gallows', 0.2), ('arduous',
0.4), ('monotony', 0.2), ('necessary', 0.6), ('begun', 0.8), ('help', 0.8),
('aside', 0.6), ('travel', 0.4), ('attacked', 0.6), ('local', 0.2),
('performer', 0.2), ('investments', 0.2), ('patients', 0.2), ('counsellor',
0.2), ('theft', 0.2), ('mexico', 0.4)]

parole presenti in tutti i file: o http state www idea so care web e book data
in rose non or come re grave c it scale do far all son me d no s dare fine i
place a
```

3.2.1 IDF: Inverse Document Frequency = logaritmo dell'inverso della Document Frequency

- **meno** documenti contengono la parola **più è grande l'inverso** e quindi il logaritmo (anche se cresce lentamente)
- **più** documenti contengono la parola **più è piccolo l'inverso e il logaritmo**

Le parole **rare** sono più interessanti (hanno IDF più grande)

[147]:
```
from math import log

# log dell'inverso della frequenza nei documenti
#    - idf   = log( # di file / # di file che contengono la parola )
IDF : dict[str,float] = { parola :  log(1/quante)
                                    for parola,quante in presenze.items() }

list(IDF.items())[:30]
# NOTA: ogni parola appare in almeno UN file quindi 1/N è sempre calcolabile
```

[147]:
```
[('clue', 0.9162907318741551),
 ('trincomalee', 1.6094379124341003),
 ('picture', 0.22314355131420976),
 ('relatives', 0.9162907318741551),
 ('amateur', 1.6094379124341003),
 ('whoever', 0.5108256237659907),
 ('anybody', 1.6094379124341003),
 ('through', 0.22314355131420976),
 ('draught', 0.9162907318741551),
 ('being', 0.22314355131420976),
 ('resemblance', 0.5108256237659907),
 ('same', 0.22314355131420976),
 ('famous', 0.9162907318741551),
 ('damage', 0.22314355131420976),
 ('gallows', 1.6094379124341003),
 ('arduous', 0.9162907318741551),
 ('monotony', 1.6094379124341003),
 ('necessary', 0.5108256237659907),
 ('begun', 0.22314355131420976),
 ('help', 0.22314355131420976),
 ('aside', 0.5108256237659907),
 ('travel', 0.9162907318741551),
 ('attacked', 0.5108256237659907),
 ('local', 1.6094379124341003),
 ('performer', 1.6094379124341003),
 ('investments', 1.6094379124341003),
 ('patients', 1.6094379124341003),
 ('counsellor', 1.6094379124341003),
 ('theft', 1.6094379124341003),
 ('mexico', 0.9162907318741551)]
```

3.2.2 finalmente posso vedere quale file "pesa di più" rispetto alle parole della query

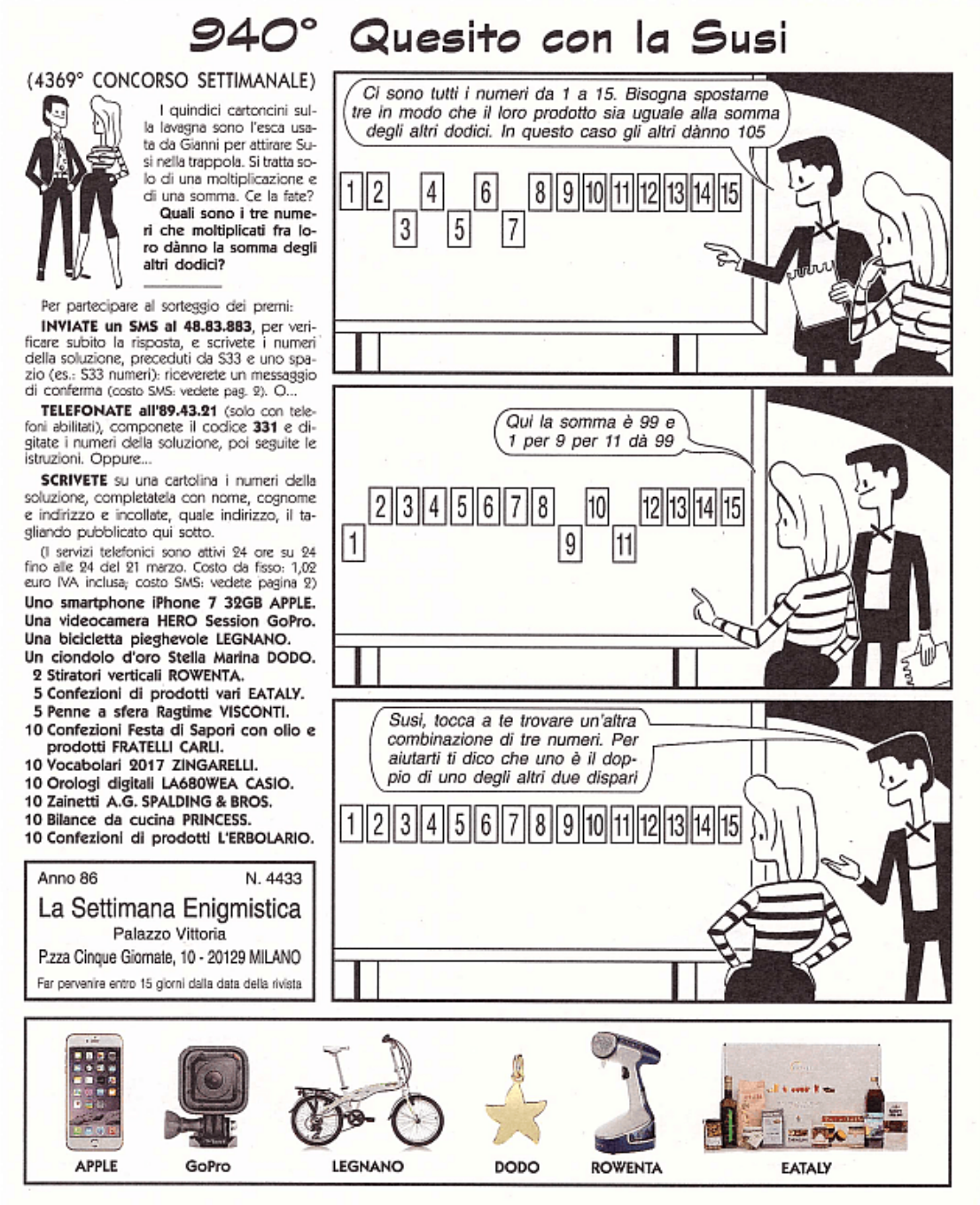
- per ciascun file
  - e per ciascuna parola di una query
    - ne prendo la frequenza % nel file (DF)
    - la moltiplico per IDF della parola
    - sommando questi contributi ottengo quanto quel file è utile per quella specifica query

[148]:
```
# Esempio di query
query : list[str] = [ 'clock', 'hat', 'violin']

pesi : dict[str,float] = {}
for file in frequenze:
    importanza = 0.0
    for parola in query:
        df = frequenze[file].get(parola,0)    # DF della parola nel file (0 se
↪assente)
```

```
        idf = IDF.get(parola,0)              # IDF della parola nei file (0 se␣
 ↪assente)
        # aggiungo il peso della parola per quel file ovvero df*idf
        importanza += df * idf
    pesi[file] = importanza # importanza del file per questa query

# ordino e stampo i file per peso decrescente
for file,peso in sorted(pesi.items(), reverse=True, key=lambda coppia:␣
 ↪coppia[1]):
    print( f"{file:25} {peso:0.5%}")
```

```
files/holmes.txt          0.04715%
files/alice.txt           0.01008%
files/frankenstein.txt    0.00522%
files/prince.txt          0.00097%
files/alice_it.txt        0.00000%
```

4 INTERVALLO: Quesito con la Susi 940



4.1 Bisogna trovare la terna di valori che:

- devono essere diversi e compresi in [1,15]
- il loro prodotto == alla somma degli altri 12
- A è pari, B e C sono dispari
- A == 2*B

- prendo un valore A **pari** tra 1 e 15
  - prendo B = A//2
    - prendo C dispari tra 1 e 15 diverso da B
      - se ABC == somma degli altri
      - stampo A,B,C

NOTA: la somma di tutti i 15 valori -A-B-C è la somma dei rimanenti

[94]:
```
def susi_940():
    SOMMA = sum(range(1,16))        # oppure (1+15)*15//2=120
    for A in range(2,15,2):         # scandisco i pari tra 1 e 15
        B = A//2
        for C in range(1,16,2):     # scandisco i dispari tra 1 e 15
            if C == B: continue     # se C == B lo ignoro
            if A*B*C == SOMMA -A-B-C :
                print(A,B,C)
susi_940()
```

```
14 7 1
```

5 Proviamo con Wolfram Alpha

5.1 Più in generale, per trovare TUTTE le terne in una sequenza 1..N

Facciamo in modo che siano $A < B < C$ - per ciascun valore $A \in [1..N-2]$ (devo fermarmi al terzultimo) - per ciascun valore $B \in (A..N-1]$ (devo fermarmi al penultimo) - per ciascun valore $C \in (B..N]$ - se $A * B * C = SOMMA - A - B - C$ - stampo A, B, C

[149]:
```
def susi_940_generale(N, verbose=True):
    quanti = 0
    SOMMA = sum(range(1,N+1))         # vale anche (N+1)*N//2
    for A in range(1,N-1):            # devo fermarmi al terzultimo (limite N-2)
        for B in range(A+1,N):        # devo fermarmi al penultimo  (limite N-1)
            for C in range(B+1,N+1): # devo arrivare ad N
                if A*B*C == SOMMA-A-B-C:
```

```
                    if verbose : print(A,B,C)
                    quanti += 1
    return quanti

susi_940_generale(15)
```

```
1 7 14
1 9 11
3 5 7
```

[149]: 3

[155]:
```
for i in range(1,100):
    if susi_940_generale(i, verbose=False) == 8:
        print(i, end=' ', flush=True)
```

```
48 98
```

6 Altri tipi di file: JSON

File di testo con sintassi semplificata per rappresentare: - dizionari - liste - interi, stringhe, float, bool, None

Molto usati nelle applicazioni WEB

[9]:
```
## Esempio
import json

with open('api.github.com.json') as F:
    api = json.load(F)
!cat api.github.com.json
```

```
{
  "current_user_url": "https://api.github.com/user",
  "current_user_authorizations_html_url":
"https://github.com/settings/connections/applications{/client_id}",
  "authorizations_url": "https://api.github.com/authorizations",
  "code_search_url":
"https://api.github.com/search/code?q={query}{&page,per_page,sort,order}",
  "commit_search_url":
"https://api.github.com/search/commits?q={query}{&page,per_page,sort,order}",
  "emails_url": "https://api.github.com/user/emails",
  "emojis_url": "https://api.github.com/emojis",
  "events_url": "https://api.github.com/events",
  "feeds_url": "https://api.github.com/feeds",
  "followers_url": "https://api.github.com/user/followers",
  "following_url": "https://api.github.com/user/following{/target}",
  "gists_url": "https://api.github.com/gists{/gist_id}",
  "hub_url": "https://api.github.com/hub",
```

```
  "issue_search_url":
"https://api.github.com/search/issues?q={query}{&page,per_page,sort,order}",
  "issues_url": "https://api.github.com/issues",
  "keys_url": "https://api.github.com/user/keys",
  "label_search_url": "https://api.github.com/search/labels?q={query}&repository
_id={repository_id}{&page,per_page}",
  "notifications_url": "https://api.github.com/notifications",
  "organization_url": "https://api.github.com/orgs/{org}",
  "organization_repositories_url":
"https://api.github.com/orgs/{org}/repos{?type,page,per_page,sort}",
  "organization_teams_url": "https://api.github.com/orgs/{org}/teams",
  "public_gists_url": "https://api.github.com/gists/public",
  "rate_limit_url": "https://api.github.com/rate_limit",
  "repository_url": "https://api.github.com/repos/{owner}/{repo}",
  "repository_search_url": "https://api.github.com/search/repositories?q={query}
{&page,per_page,sort,order}",
  "current_user_repositories_url":
"https://api.github.com/user/repos{?type,page,per_page,sort}",
  "starred_url": "https://api.github.com/user/starred{/owner}{/repo}",
  "starred_gists_url": "https://api.github.com/gists/starred",
  "topic_search_url":
"https://api.github.com/search/topics?q={query}{&page,per_page}",
  "user_url": "https://api.github.com/users/{user}",
  "user_organizations_url": "https://api.github.com/user/orgs",
  "user_repositories_url":
"https://api.github.com/users/{user}/repos{?type,page,per_page,sort}",
  "user_search_url":
"https://api.github.com/search/users?q={query}{&page,per_page,sort,order}"
}
```

6.1 Attenzione:

- **NO COMMENT:** non si possono inserire commenti
- **NO COMMA:** non si può aggiungere una virgola in fondo a lista o dizionario
- **SOLO chiavi semplici** (NO tuple, SI int, float, str, none, bool)
- **SOLO doppi apici** " (NO singoli apici)
- **NO TUPLE:** si possono usare liste e poi convertirle

6.2 Abbastanza facili da usare in python

Conversione automatica per alcuni tipi di dati - `json.load(<file>) -> dict | list | tipo_base` - `json.dump(<obj>, <file>)`

Personalizzabile anche per altri tipi di oggetti (non ovvio, per casa per chi è interessato)

[10]:
```
# posso anche leggere da una stringa in formato JSON con la funzione 'loads'
XX = json.loads('''
[{"nome": "Minnie", "cognome": "Mouse", "telefono": "555-54321",
  "indirizzo": "via di M.me Curie 1", "città": "Topolinia"},
```

```
{"nome": "Pippo", "cognome": "de' Pippis", "telefono": "555-33333",
 "indirizzo": "via dei Pioppi 1", "città": "Topolinia"}]
''')
print(XX)
# oppure produrre la stringa in formato JSON da una struttura Python
print(json.dumps(XX))
```

```
[{'nome': 'Minnie', 'cognome': 'Mouse', 'telefono': '555-54321', 'indirizzo':
'via di M.me Curie 1', 'città': 'Topolinia'}, {'nome': 'Pippo', 'cognome': "de'
Pippis", 'telefono': '555-33333', 'indirizzo': 'via dei Pioppi 1', 'città':
'Topolinia'}]
[{"nome": "Minnie", "cognome": "Mouse", "telefono": "555-54321", "indirizzo":
"via di M.me Curie 1", "citt\u00e0": "Topolinia"}, {"nome": "Pippo", "cognome":
"de' Pippis", "telefono": "555-33333", "indirizzo": "via dei Pioppi 1",
"citt\u00e0": "Topolinia"}]
```

[11]:
```
# o produrre una stringa JSON da un oggetto Python
json.dumps(XX)
```

[11]:
```
'[{"nome": "Minnie", "cognome": "Mouse", "telefono": "555-54321", "indirizzo":
 "via di M.me Curie 1", "citt\\u00e0": "Topolinia"}, {"nome": "Pippo", "cognome":
 "de\' Pippis", "telefono": "555-33333", "indirizzo": "via dei Pioppi 1",
 "citt\\u00e0": "Topolinia"}]'
```

[156]:
```
# Un file json può contenere valori semplici (int, float, str, True=true,␣
 ↪False=false, None=null)
print(json.dumps(None))

print(json.dumps([False, True]))

## NOTA: le stringhe DEVONO essere racchiuse da doppi apici
print(json.dumps('Pape"rino'))
```

```
null
[false, true]
"Pape\"rino"
```

[13]:
```
# Esempio: data una tabella come lista di dizionari
agenda = [
    {'nome': 'Paperino','cognome':'Paolino',      'telefono':'555-1313', ␣
 ↪'indirizzo': 'via dei Peri 113',          'città': 'Paperopoli'},
    {'nome': 'Gastone', 'cognome':'Paperone',     'telefono':'555-1717', ␣
 ↪'indirizzo': 'via dei Baobab 42',         'città': 'Paperopoli'},
    {'nome': 'Paperon', 'cognome':"de' Paperoni", 'telefono':'555-99999',␣
 ↪'indirizzo': 'colle Papero 1',            'città': 'Paperopoli'},
    {'nome': 'Archimede','cognome':'Pitagorico',  'telefono':'555-11235',␣
 ↪'indirizzo': 'colle degli Inventori 1',   'città': 'Paperopoli'},
```

```
   {'nome': 'Pietro',  'cognome':'Gambadilegno', 'telefono':'555-66666',␣
↪'indirizzo': 'via dei Ladri 13',          'città': 'Topolinia'},
   {'nome': 'Trudy',   'cognome':'Gambadilegno', 'telefono':'555-66666',␣
↪'indirizzo': 'via dei Ladri 13',          'città': 'Topolinia'},
   {'nome': 'Topolino','cognome':'Mouse',        'telefono':'555-12345',␣
↪'indirizzo': 'via degli Investigatori 1', 'città': 'Topolinia'},
   {'nome': 'Minnie',  'cognome':'Mouse',        'telefono':'555-54321',␣
↪'indirizzo': 'via di M.me Curie 1',       'città': 'Topolinia'},
   {'nome': 'Pippo',   'cognome':"de' Pippis",   'telefono':'555-33333',␣
↪'indirizzo': 'via dei Pioppi 1',          'città': 'Topolinia'},
   ]
```

[14]:
```
import json
# prima la salvo nel file
with open('agenda.json', encoding='utf8', mode='w') as F:
    json.dump(agenda, F)

!cat agenda.json
```

```
[{"nome": "Paperino", "cognome": "Paolino", "telefono": "555-1313", "indirizzo":
"via dei Peri 113", "citt\u00e0": "Paperopoli"}, {"nome": "Gastone", "cognome":
"Paperone", "telefono": "555-1717", "indirizzo": "via dei Baobab 42",
"citt\u00e0": "Paperopoli"}, {"nome": "Paperon", "cognome": "de' Paperoni",
"telefono": "555-99999", "indirizzo": "colle Papero 1", "citt\u00e0":
"Paperopoli"}, {"nome": "Archimede", "cognome": "Pitagorico", "telefono":
"555-11235", "indirizzo": "colle degli Inventori 1", "citt\u00e0":
"Paperopoli"}, {"nome": "Pietro", "cognome": "Gambadilegno", "telefono":
"555-66666", "indirizzo": "via dei Ladri 13", "citt\u00e0": "Topolinia"},
{"nome": "Trudy", "cognome": "Gambadilegno", "telefono": "555-66666",
"indirizzo": "via dei Ladri 13", "citt\u00e0": "Topolinia"}, {"nome":
"Topolino", "cognome": "Mouse", "telefono": "555-12345", "indirizzo": "via degli
Investigatori 1", "citt\u00e0": "Topolinia"}, {"nome": "Minnie", "cognome":
"Mouse", "telefono": "555-54321", "indirizzo": "via di M.me Curie 1",
"citt\u00e0": "Topolinia"}, {"nome": "Pippo", "cognome": "de' Pippis",
"telefono": "555-33333", "indirizzo": "via dei Pioppi 1", "citt\u00e0":
"Topolinia"}]
```

[15]:
```
# poi la ricarico
with open('agenda.json', encoding='utf8') as F:
    L1 = json.load(F)
L1[:2]  # e ne mostro i primi 2 elementi
```

[15]:
```
[{'nome': 'Paperino',
  'cognome': 'Paolino',
  'telefono': '555-1313',
  'indirizzo': 'via dei Peri 113',
  'città': 'Paperopoli'},
```

```
 {'nome': 'Gastone',
  'cognome': 'Paperone',
  'telefono': '555-1717',
  'indirizzo': 'via dei Baobab 42',
  'città': 'Paperopoli'}]
```

7 File YAML (un altro modo di rappresentare dati annidati in testo semplice)

- dizionari (una **chiave** : **valore** per ciascuna riga)
- liste (valori su righe diverse, preceduti da -)
- dati semplici (str senza apici se possibile, True, False, null=None, interi, float)
- documenti multipli
- generici oggetti Python (advanced)

Per annidare le strutture si usa l'**indentazione**

[16]:
```
import yaml
with open('agenda.yaml', mode='w', encoding='utf8') as F:
    yaml.dump(agenda[:2], F)
!cat agenda.yaml
```

```
- "citt\xE0": Paperopoli
  cognome: Paolino
  indirizzo: via dei Peri 113
  nome: Paperino
  telefono: 555-1313
- "citt\xE0": Paperopoli
  cognome: Paperone
  indirizzo: via dei Baobab 42
  nome: Gastone
  telefono: 555-1717
```

7.1 leggere/scrivere file YAML

- `yaml.dump(<oggetto>, FILE)` salva l'oggetto nel file (che deve essere stato già aperto con open)
- `yaml.safe_load(FILE) -> <oggetto>` legge l'oggetto dal file (che deve essere stato già aperto con open)

[17]:
```
## posso decodificare direttamente del testo
testo = """
  none: [~, null]
  bool: [true, false, on, off]
  int: 42
  float: 3.14159
  list:
    - LITE
```

```
    - RES_ACID
    - SUS_DEXT
  dict:
    hp: 13
    sp: 5
"""
yaml.safe_load(testo)
```

[17]:
```
{'none': [None, None],
 'bool': [True, False, True, False],
 'int': 42,
 'float': 3.14159,
 'list': ['LITE', 'RES_ACID', 'SUS_DEXT'],
 'dict': {'hp': 13, 'sp': 5}}
```

[18]:
```
with open('esempio.yaml', mode='w', encoding='utf8') as F:
    print(testo, file=F)

!cat esempio.yaml
# oppure leggerlo direttamente da file
with open('esempio.yaml') as F:
    EX = yaml.safe_load(F)
EX
```

```
 none: [~, null]
 bool: [true, false, on, off]
 int: 42
 float: 3.14159
 list:
   - LITE
   - RES_ACID
   - SUS_DEXT
 dict:
   hp: 13
   sp: 5
```

[18]:
```
{'none': [None, None],
 'bool': [True, False, True, False],
 'int': 42,
 'float': 3.14159,
 'list': ['LITE', 'RES_ACID', 'SUS_DEXT'],
 'dict': {'hp': 13, 'sp': 5}}
```

8 Leggere pagine o file da Internet

Ci sono molte librerie, la più comune è **requests** - permette di eseguire richieste sia di tipo **GET** che **POST** - **GET** classico url che inserite nel browser, volendo con parametri codificati direttamente nell'URL - **POST** richiesta che una form html fa al server, con tutti i contenuti dei campi della form - con parametri - torna un codice di errore/OK - e decodifica il testo della pagina html o json

Molto potente, permette anche streaming, sessioni, ...

[157]:
```
# importo la libreria
import requests

# leggo la pagina di python.org
pagina = requests.get('https://python.org')

# lo status code ci dice se tutto è andato bene
status = pagina.status_code

# se è un testo nell'attributo text trovo il testo decodificato
contenuto = pagina.text

print('status:\t\t', status, '\nencoding:\t', pagina.encoding,'\n', contenuto[:
 ↪500])
```

```
status:          200
encoding:        utf-8
 <!doctype html>
<!--[if lt IE 7]>   <html class="no-js ie6 lt-ie7 lt-ie8 lt-ie9">   <![endif]-->
<!--[if IE 7]>      <html class="no-js ie7 lt-ie8 lt-ie9">          <![endif]-->
<!--[if IE 8]>      <html class="no-js ie8 lt-ie9">                 <![endif]-->
<!--[if gt IE 8]><!--><html class="no-js" lang="en" dir="ltr">  <!--<![endif]-->

<head>
    <!-- Google tag (gtag.js) -->
    <script async
src="https://www.googletagmanager.com/gtag/js?id=G-TF35YF9CVH"></script>
    <script>
      window.d
```

8.1 le pagine JSON vengono automaticamente decodificate

[20]:
```
# leggo da internet una pagina JSON
pagina_json = requests.get('https://api.github.com')

# la decodifica avviene automaticamente col metodo json()
risultato = pagina_json.json()
risultato
```

```
[20]: {'current_user_url': 'https://api.github.com/user',
       'current_user_authorizations_html_url':
      'https://github.com/settings/connections/applications{/client_id}',
       'authorizations_url': 'https://api.github.com/authorizations',
       'code_search_url':
      'https://api.github.com/search/code?q={query}{&page,per_page,sort,order}',
       'commit_search_url':
      'https://api.github.com/search/commits?q={query}{&page,per_page,sort,order}',
       'emails_url': 'https://api.github.com/user/emails',
       'emojis_url': 'https://api.github.com/emojis',
       'events_url': 'https://api.github.com/events',
       'feeds_url': 'https://api.github.com/feeds',
       'followers_url': 'https://api.github.com/user/followers',
       'following_url': 'https://api.github.com/user/following{/target}',
       'gists_url': 'https://api.github.com/gists{/gist_id}',
       'hub_url': 'https://api.github.com/hub',
       'issue_search_url':
      'https://api.github.com/search/issues?q={query}{&page,per_page,sort,order}',
       'issues_url': 'https://api.github.com/issues',
       'keys_url': 'https://api.github.com/user/keys',
       'label_search_url': 'https://api.github.com/search/labels?q={query}&repository_
      id={repository_id}{&page,per_page}',
       'notifications_url': 'https://api.github.com/notifications',
       'organization_url': 'https://api.github.com/orgs/{org}',
       'organization_repositories_url':
      'https://api.github.com/orgs/{org}/repos{?type,page,per_page,sort}',
       'organization_teams_url': 'https://api.github.com/orgs/{org}/teams',
       'public_gists_url': 'https://api.github.com/gists/public',
       'rate_limit_url': 'https://api.github.com/rate_limit',
       'repository_url': 'https://api.github.com/repos/{owner}/{repo}',
       'repository_search_url': 'https://api.github.com/search/repositories?q={query}{
      &page,per_page,sort,order}',
       'current_user_repositories_url':
      'https://api.github.com/user/repos{?type,page,per_page,sort}',
       'starred_url': 'https://api.github.com/user/starred{/owner}{/repo}',
       'starred_gists_url': 'https://api.github.com/gists/starred',
       'topic_search_url':
      'https://api.github.com/search/topics?q={query}{&page,per_page}',
       'user_url': 'https://api.github.com/users/{user}',
       'user_organizations_url': 'https://api.github.com/user/orgs',
       'user_repositories_url':
      'https://api.github.com/users/{user}/repos{?type,page,per_page,sort}',
       'user_search_url':
      'https://api.github.com/search/users?q={query}{&page,per_page,sort,order}'}
```

8.2 Scaricare file binari

- il contenuto "raw" (crudo) è nell'attributo `content` della risposta

[21]:
```
## possiamo anche scaricare file binari
logo = requests.get('https://www.python.org/static/img/python-logo@2x.png')

# posso estrarre dagli headers le dimensioni e il tipo del file ()
print(logo.headers['Content-Type'], logo.headers['Content-Length'])
```

```
image/png 15770
```

[22]:
```
# se si tratta di un file di dati (immagine/audio/film ...)
# il contenuto lo trovo nell'attributo content
dati = logo.content

# per salvarlo posso aprire un file in scrittura ('w') e in modo binario ('b')
with open('logo.png', mode='wb') as F:
    F.write(dati)
```

[23]:
```
!ls -alh 'logo.png'
print(dati[:100])
```

```
-rw-r--r--  1 andrea  staff    15K Oct 27 16:45 logo.png
b'\x89PNG\r\n\x1a\n\x00\x00\x00\rIHDR\x00\x00\x02D\x00\x00\x00\xa4\x08\x06\x00\x
00\x00v;1\xf1\x00\x00\x00\tpHYs\x00\x00\x0b\x13\x00\x00\x0b\x13\x01\x00\x9a\x9c\
x18\x00\x00 \x00IDATx\x9c\xed\x9dy\x9c\x1ce\x9d\xff?\xdf\xaa\x9eL\x8e\xc91\xb9\x
c8\x1c9\x80If2\xe6 $\x18\x02\x01"+\xcb\x8d\xbb\xca'
```

Ed ecco il risultato

8.2.1 Per mostrare l'immagine in Jupyter senza salvare il file

basta fornire i dati binari alla classe **IPython.display.Image**

[24]:
```
from IPython.display import Image
dati = requests.get('https://i.imgflip.com/5vcovc.jpg').content
Image(dati)
```

[24]:

8.3 Passare parametri ad una GET

Si usa l'argomento `params` (un dizionario) - `parametri={ chiave: valore, ...}` - `requests.get( <URL>, params=parametri )`

[25]:
```
# Search Google for the requests Python library
response = requests.get(
    'https://google.com',
    params={'q': 'python requests'},
)
print(response.text[:1000])
```

```
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage"
lang="it"><head><meta content="text/html; charset=UTF-8" http-equiv="Content-
Type"><meta
content="/images/branding/googleg/1x/googleg_standard_color_128dp.png"
itemprop="image"><title>Google</title><script
nonce="zimG2HgdQw-02U7KpusOhg">(function(){var _g={kEI:'7sw7ZaTAFburxc8PzdWoyAU'
,kEXPI:'0,1365468,206,2415,2389,2316,383,246,5,1129120,1197757,380733,16115,2868
4,22430,1362,12317,2817,1931,12834,4998,17075,38444,2872,2891,3926,214,4209,3405
,606,30668,30022,6398,9937,20583,4,59617,4437,22604,6633,7596,1,42160,2,39755,56
79,1021,31122,4568,6255,23421,1246,59707,8155,23351,873,19632,8,1922,9779,42459,
3141,17057,928,4869,6,14430,20206,6819,1558,4235,4,1,1,14747,5375,782,1483,765,1
1151,4665,1804,27093,8176,11813,342,1291,26701,3052,4716,146,7849,1780,5222256,2
```

```
,602,5994363,2806525,141,795,29513,25,2,41,23,23,13,6,4,3,22,23940892,579,404352
8,14297,2375,40904,1761,1218,3,1562,5,542,3,1165,4,342,181,190,560,1391500,34616
1,2
```

8.4 Passare parametri ad altri metodi HTTP

Si usa l'argomento `data=<dizionario>` che contiene i parametri

```
requests.post(   'https://httpbin.org/post',   data={'key':'value'})
requests.put(    'https://httpbin.org/put',    data={'key':'value'})
requests.delete( 'https://httpbin.org/delete')
requests.head(   'https://httpbin.org/get')
requests.patch(  'https://httpbin.org/patch',  data={'key':'value'})
requests.options('https://httpbin.org/get')
```