Table of Contents

Fondamenti di Programmazione

Andrea Sterbini

lezione 10 - 7 novembre 2022

Wooclap: che avete capito finora?

1 Vai a wooclap.com

2 Immettere il codice dell'evento nel banner superiore

Codice evento
F22LEZ10

RECAP: list comprehension
sintassi semplificata per costruire contenitori semplici

```
<parentesi che indica il tipo di contenitore>
   <espressione che calcola un elemento>
   <ciclo/i>
   <condizione/i>
   ...
<parentesi chiusa>
```

In [1]:
```
## Esempio
[   (x, y)              # ciascun elemento è una coppia (x,y)
    for x in range(5) # per ciascun x in 0,1,2,3,4
    if  x%2             # ma solo se x è dispari
    for y in range(5*x)    # e per ogni y in 0,1,2,3,4
    if  y%3                 # ma solo se y non è mult. di 3
   ]
```

Out[1]:
```
[(1, 1),
 (1, 2),
 (1, 4),
 (3, 1),
 (3, 2),
 (3, 4),
 (3, 5),
 (3, 7),
 (3, 8),
 (3, 10),
 (3, 11),
 (3, 13),
 (3, 14)]
```

In [8]:
```
## Tipi di contenitori
[      x*2 for x in range(10) ]  # lista
{      x*2 for x in range(10) }  # insieme    (singolo elemento)
{   x: x*2 for x in range(10) }  # dizionario (coppia chiave : valore)
tuple( x*2 for x in range(10) )  # tupla

B = 16
A = 3 if B%2==1 else 23
A

# Possiamo usare espressioni condizionate
# che tornano valori diversi per casi diversi
[ x*2 if x%3 else x*4
      for x in range(10) ]
```

Out[8]:
```
[0, 2, 4, 12, 8, 10, 24, 14, 16, 36]
```

RECAP: sorted

sorted e **sort**, **min** e **max** ammettono un parametro **key** che accetta una funzione che fa da "criterio di ordinamento"che:

- accetta un solo valore (l'elemento della lista)
- produce un qualcosa che sarà usato da sorted per ordinare i valori (trasformazione di Schwartz)

In [11]:
```
## Esempio: ordino una lista di interi mettendo:
#  - prima i valori pari, in ordine decrescente
#  - poi i valori dispari, in ordine crescente

# soluzione 1: separo i due gruppi, li ordino, poi li concateno
L = [ 1, 5, 2, 4, 7, 1, 9, 4, 8, 6, ]
pari    = [ x for x in L if x%2==0 ]  # estraggo i pari
dispari = [ x for x in L if x%2    ]  # e i dispari
print(pari, dispari)
pari.sort(reverse=True)               # li ordino
dispari.sort()
pari + dispari                        # li concateno
```

```
[2, 4, 4, 8, 6] [1, 5, 7, 1, 9]
```

Out[11]:
```
[8, 6, 4, 4, 2, 1, 1, 5, 7, 9]
```

In [14]:
```
# Soluzione 2: per ogni valore creo una coppia che contiene
# - l'indicazione se sono pari o dispari che separerà i due gruppi
#   ad esempio il resto della divisione per 2
#   che mette prima i pari (resto 0) e poi i dispari (resto 1)
# - il valore:
#   - così com'è per ordinarlo in senso crescente se è dispari
#   - oppure negato per ordinarlo in senso decrescente se era pari
def criterio(x):
    return x%2, (x if x%2 else -x)

[ criterio(x) for x in L ]

sorted([ criterio(x) for x in L])
sorted(L, key=criterio)
```

Out[14]: `[8, 6, 4, 4, 2, 1, 1, 5, 7, 9]`

RECAP: funzioni anonime (espressioni `lambda`)

- accettano uno o più argomenti
- calcolano SOLO una espressione

Sintassi: **`lambda <argomenti>: <espressione>`**

Sono utilissime per piccole trasformazioni dei dati, per sorted ed altre operazioni "funzionali"

In [15]:
```
## Esempio:
# ricerca del massimo rispetto ad un ordinamento complicato
## Usiamo il *key* di prima nella funzione *max*
# come se fosse una *sorted*
sorted(L, key=lambda x: (x%2, x if x%2 else -x))
```

Out[15]: `[8, 6, 4, 4, 2, 1, 1, 5, 7, 9]`

RECAP: funzioni definite internamente a funzioni

- sono disponibili SOLO all'interno della funzione "esterna"
- possono usare gli argomenti e le variabili definiti nella funzione "esterna"

Sono comodissime quando:

- vogliamo **impedire** che siano usate altrove
- vogliamo rendere più **leggibile** il codice dando un nome alla trasformazione
- la trasformazione da applicare ai dati ha bisogno di **più di una** istruzione
- dobbiamo usare **UN SOLO argomento** ma i conti da fare hanno bisogno di altri parametri aggiuntivi disponibili nella funzione esterna

In [17]:
```
# Esempio: data una tabella come lista di dizionari
agenda = [
    {'nome': 'Paperino','cognome':'Paolino',      'telefono':'555-1313',  'indirizzo'
    {'nome': 'Gastone', 'cognome':'Paperone',     'telefono':'555-1717',  'indirizzo'
    {'nome': 'Paperon', 'cognome':"de' Paperoni", 'telefono':'555-99999', 'indirizzo'
    {'nome': 'Archimede','cognome':'Pitagorico',  'telefono':'555-11235', 'indirizzo'
    {'nome': 'Pietro',  'cognome':'Gambadilegno', 'telefono':'555-66666', 'indirizzo'
    {'nome': 'Trudy',   'cognome':'Gambadilegno', 'telefono':'555-66666', 'indirizzo'
    {'nome': 'Topolino','cognome':'Mouse',        'telefono':'555-12345', 'indirizzo'
    {'nome': 'Minnie',  'cognome':'Mouse',        'telefono':'555-54321', 'indirizzo'
    {'nome': 'Pippo',   'cognome':"de' Pippis",   'telefono':'555-33333', 'indirizzo'
    ]
```

In [19]:
```
# Se voglio cercare i valore massimo rispetto ad una colonna data
def cerca_massimo(ag, colonna) :
    # key DEVE essere una funzione *di un solo argomento*
    # ma per estrarre il valore mi serve anche la colonna
    def estrai_valore(riga) :
        return riga[colonna]
    return max(ag, key=estrai_valore)
cerca_massimo(agenda, 'cognome')
```

Out[19]:
```
{'nome': 'Pippo',
 'cognome': "de' Pippis",
 'telefono': '555-33333',
 'indirizzo': 'via dei Pioppi 1',
 'città': 'Topolinia'}
```

In [19]:
```
# per una estrazione così semplice posso usare anche una lambda
# NOTA: anche la lambda ha accesso alle variabili della funzione esterna
def cerca_massimo(ag, colonna) :
    return max(ag, key=lambda riga: riga[colonna])
cerca_massimo(agenda, 'nome')
```

Out[19]:
```
{'nome': 'Trudy',
 'cognome': 'Gambadilegno',
 'telefono': '555-66666',
 'indirizzo': 'via dei Ladri 13',
 'città': 'Topolinia'}
```

RECAP: stile funzionale - funzioni che "trasformano" sequenze di dati

- **`map( <funzione>, <contenitore>, ...)`**
  che produce una nuova sequenza
- **`max( <contenitore>, key=<funzione>)`**
  che torna il massimo (e **min** il minimo)
- **`filter(<predicato>, <contenitore>)`**
  che estrae i valori che soddisfano il predicato
- **`all(<contenitore>)`**
  che torna **True** se TUTTI i valori sono **True**
- **`any(<contenitore>)`**
  che torna **True** se ALMENO UNO dei valori è **True**

In [21]:
```
## Esempio: calcoliamo i cubi di una lista di valori con map
LI = [ 3, 5, 2, 8 ]
list(map( lambda x: x**3, LI ))
# lo possiamo fare anche definendo una funzione 'cubi' oppure con una list-comprehens
#[ x**3 for x in LI]
```

Out[21]:
```
[27, 125, 8, 512]
```

In [30]:
```
# Esempio: estraiamo tutte le stringhe da una lista eterogenea
LE = [ 'uno', 2, 'tre', 4, 'cinque' ]
F = filter( lambda x: type(x)== str, LE  )
#next(F)
#next(F)
#next(F)
# next(F) # genera un errore perchè non ci sono altri elementi
list(F)
```

Out[30]:
```
['uno', 'tre', 'cinque']
```

La funzione `zip` per fondere più contenitori

- prende prima i primi elementi di ciascuna lista, poi i secondi ...

Sintassi: **`zip( Contenitore1, Contenitore2, ...)`**

Il risultato è una lista di tuple

- comodissima per scandire in parallelo più liste senza usare indici

In [33]:
```
# Esempio: unisco più liste
L1 = [ 1, 2, 3, 4, 5, 6, 7 ]  # 7 elementi
S2 = 'abcdef'                 # 6 elementi
L3 = ['A', 'B', 'C', 'D']     # 4 elementi
list(zip(L1, S2, L3))
# NOTA: torna una sequenza della lunghezza MINIMA tra le tre
```

Out[33]:
```
[(1, 'a', 'A'), (2, 'b', 'B'), (3, 'c', 'C'), (4, 'd', 'D')]
```

In [36]:
```
## Esempio: li stampo su 3 colonne con indici
# (generando un errore perchè la prima è più lunga)
for i in range(min(len(L1),len(S2),len(L3))):
    print(L1[i], S2[i], L3[i], sep='\t')
```

```
1	a	A
2	b	B
3	c	C
4	d	D
```

In [37]:
```
## Esempio: li stampo su 3 colonne con zip
for a, b, c in zip(L1, S2, L3):
    print(a, b, c, sep='\t')
```

```
1	a	A
2	b	B
3	c	C
4	d	D
```

Files e filesystem

- i files sono blocchi di bytes
- possono essere di testo (txt, py) o binari (png, mp4, mp3, jpeg ...)
- sono memorizzati nella memoria di massa (HD o SSD o DVD) o in rete
- sono organizzati in una struttura di directory ad albero

Lettura e scrittura di files

Per leggere e scrivere i file bisogna "aprirli", ovvero chiedere al sistema operativo (SO) di preparare le strutture dati necessarie ad interagire con la memoria di massa.

Python per interagire col file-system usa la libreria **`os`** e la funzione **`open`**

Sintassi: **`open(<filename> : str,`**
`mode : str = <modo>,`
`encoding : str = <encoding> ) -> File`

`<filename>` è una stringa che indica il nome del file da "aprire"

- viene letto dalla directory corrente
- oppure da una altro punto del filesystem se si indica il percorso che ne individua la posizione

Esempi:

- **`paperino.txt`** viene cercato nella directory corrente
- **`/usr/bin/python`** viene cercato nella directory **`/usr/bin`**

`<mode>` è una stringa che indica che il file viene aperto

Inizia con un carattere

- **`r`** (read) solo per leggerne il contenuto (default)
- **`w`** (write) per scriverci dentro
- **`a`** (append) per aggiungere nuovo contenuto alla fine

che può essere seguito dal carattere

- **`t`** il file viene aperto in modo testo (il contenuto viene interpretato come caratteri)
- **`b`** il file viene aperto in modo binario (il contenuto non viene interpretato automaticamente)

`<encoding>` è una stringa che indica come viene de/codificato un file di testo

Esempi

- **`utf8`** codifica unicode
- **`latin`** codifica latin
- **`ascii`** codifica ASCII
- **...**

alla fine il file deve essere "chiuso"

In questo modo vengono:

- rilasciate le strutture del SO liberando memoria
- **completate le scritture** dei file su HD !!!!!

In [38]:

```python
# open: costruzione di un oggetto che fa da interfaccia al file su HD
# mode: 'r', 'w', 'a' con il modificatore 't' o 'b'
# encoding:

# se voglio aprire un file di testo e scriverci
F = open("pippo.txt", encoding='utf8', mode='w')
# posso usare 'write' per scrivere un testo
# (ricordando di metterci l'accapo in fondo alla riga)
F.write("pippo è andato al mare\n")
# oppure usare print con il parametro 'file' (sooo easy!!!)
print("kjglahgkjhkj kajhhk gj", file=F)
# oppure riusare write
F.write("pippo è andato al mare\n")

# ALLA FINE DEVO CHIUDERE IL FILE!!!
F.close()
# rilascio delle strutture dati e *salvataggio su HD!!!*
```

Aprire e chiudere files con un "contesto" `with`

La parola chiave **`with`** apre un "contesto" che ci assicura che le cose vengano "gestite bene" anche in caso di errore

Sintassi:

```
with open( <filename> ... ) as <variabile>:
    blocco di codice
```

In questo modo siamo SICURI che il file venga chiuso!

In [39]:
```python
# MOOOLTO MEGLIO: uso la parola chiave 'with'
#   per aprire un "contesto" in cui apro il file
#   e che mi assicura che il file verrà chiuso
#   correttamente SEMPRE (anche in caso di eccezioni)
with open('topolino.txt', mode='w', encoding='utf8') as F:
    F.write("pippo è andato al mare\n")
    print("kjglahgkjhkj kajhhk gj", file=F)
    F.write("pippo è andato al mare\n")
    assert False # anche se inserisco un errore il file viene salvato

# BAD STYLE: non conviene usare direttamente open e close
# - errori ed eccezioni
# - semplici dimenticanze
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Cell In [39], line 9
      7 print("kjglahgkjhkj kajhhk gj", file=F)
      8 F.write("pippo è andato al mare\n")
----> 9 assert False

AssertionError:
```

Spostarsi nel file con `seek`

In [40]:
```python
# per posizionarsi in un certo punto del file (0=inizio)

# se voglio 2 volte leggere il file e stamparlo
with open('topolino.txt', encoding='utf8') as F:
    testo = F.read()
    print(testo)
    # la seconda volta sono in fondo al file e non leggo niente
    testo = F.read()
    print('====================')
    print(testo)
```

```
pippo è andato al mare
kjglahgkjhkj kajhhk gj
pippo è andato al mare

====================
```

In [41]:
```python
# Con seek torno all'inizio
with open('topolino.txt', encoding='utf8') as F:
    testo = F.read()
    print(testo)
    F.seek(0)                # torno all'inizio
    testo = F.read()         # la seconda volta lo rileggo tutto
    print('====================')
    print(testo)
```

```
pippo è andato al mare
kjglahgkjhkj kajhhk gj
pippo è andato al mare

====================
pippo è andato al mare
kjglahgkjhkj kajhhk gj
pippo è andato al mare
```

Lettura di tutto il file, una riga, tutte le righe

In [43]:
```python
# read:     lettura di tutto il file (ok con file binari)
# readline: lettura da un file *di testo* di una linea per volta
# readlines: tutte assieme

# readline legge una sola riga (con in fondo l'accapo)
with open('topolino.txt', encoding='utf8') as F:
    riga = F.readline()
    print("$", riga.strip(), "$")   # strip toglie gli spazi prima e dopo
```

```
$ pippo è andato al mare $
```

In [44]:
```
# readlines legge tutte le righe e torna una lista di stringhe (con gli accapi)
with open('topolino.txt', encoding='utf8') as F:
    righe = F.readlines()
    for riga in righe:
        print(riga)
righe
```

```
pippo è andato al mare

kjglahgkjhkj kajhhk gj

pippo è andato al mare
```

Out[44]:
```
['pippo è andato al mare\n',
 'kjglahgkjhkj kajhhk gj\n',
 'pippo è andato al mare\n']
```

Usare il file come un generatore di righe (memory efficient)

In [45]:
```
# scansione di un file riga per riga

# ma se voglio usare meno memoria posso usare direttamente
# il file F come un *generatore* di righe
# e leggerle una per volta
with open('topolino.txt', encoding='utf8') as F:
    # per ciascuna richiesta viene tornata la prossima riga
    for riga in F:
        print(riga)
```

```
pippo è andato al mare

kjglahgkjhkj kajhhk gj

pippo è andato al mare
```

In [60]:
```
with open('topolino.txt', encoding='utf8') as F:
    for riga in F:          # scandisco le righe del file
        print(riga)         # e le stampo
    F.seek(0)               # mi riporto all'inizio del file
    for riga in F:          # di nuovo scandisco le righe del file
        print(riga)
```

```
pippo è andato al mare

kjglahgkjhkj kajhhk gj

pippo è andato al mare

pippo è andato al mare

kjglahgkjhkj kajhhk gj

pippo è andato al mare
```

Encoding del testo

- i caratteri sono codificati come numeri
- esistono diverse codifiche (ascii, latin, windows, **unicode**)
- Python usa **utf-8** ma i file possono provenire da SO diversi

In [70]:
```
!file files/*.txt
```

```
files/alice_it.txt:     ISO-8859 text, with very long lines (1352)
files/alice.txt:        Unicode text, UTF-8 (with BOM) text, with CRLF line terminators
files/frankenstein.txt: ASCII text, with CRLF line terminators
files/holmes.txt:       Unicode text, UTF-8 (with BOM) text, with CRLF line terminators
files/prince.txt:       Unicode text, UTF-8 (with BOM) text, with CRLF line terminators
files/results.txt:      ASCII text
files/testo2.txt:       ASCII text
files/testo.txt:        Unicode text, UTF-8 text
```

In [46]:
```
# files e loro encoding
files = {
    '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'
    }
```

In [49]:
```
# leggo 'alice_it.txt'
with open('files/alice_it.txt', encoding='latin') as F:
    testo = F.read()
print(testo[:300])
```

```
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/ (http://www.e-text.it/)

QUESTO E-BOOK:

TITOLO: Alice nel paese delle meraviglie
AUTORE: Dodgson, Charles Lutwidge (alias Lewis Carroll)
NO
```

Come trovare le parole contenute in un file

- distinguiamo le lettere alfabetiche (che ci servono)
- dal resto che usiamo come separatore
- e magari trasformiamo tutto in minuscole

In [50]:
```python
def leggi_parole(filename, encoding):
    # apriamo il file
    with open(filename, encoding=encoding) as F:
        # leggo il contenuto
        testo = F.read()
    # lo metto in minuscole
    testo = testo.lower()
    # trovo i caratteri NON alfabetici
    nonalfa = trova_nonalfa(testo)
    # li sostituisco con spazi
    for carattere in nonalfa:
        testo = testo.replace(carattere, ' ')
    # spezzo il testo in parole
    return testo.split()
```

In [51]:
```python
def trova_nonalfa(testo : str) -> set[str] :
    # trovo i caratteri usati
    caratteri = set(testo)
    # ne estraggo i NON alfabetici
    return { c for c in caratteri if not c.isalpha()  }
```

In [88]:
```python
def rimpiazza_con_spazi2(testo : str, nonalfa : set[str]) -> str :
    # oppure un unico translate
    pass
```

In [55]:
```python
parole = leggi_parole('files/alice_it.txt', 'latin')
```

In [57]:
```python
%pprint
parole[:30]
```

```
Pretty printing has been turned OFF
```

Out[57]:
```
['charles', 'lutwidge', 'dodgson', 'alice', 'nel', 'paese', 'delle', 'meraviglie', 'quest
o', 'e', 'book', 'è', 'stato', 'realizzato', 'anche', 'grazie', 'al', 'sostegno', 'di', '
e', 'text', 'editoria', 'web', 'design', 'multimedia', 'http', 'www', 'e', 'text', 'it']
```