:::{list-table} :header-rows: 1 :align: center :widths: 90 10

- ▪ Fondamenti di programmazione

  Canale A-L - Prof. Sterbini

  AA 25-26 - Lezione 10

  - :::{image}attachment:5a837f06-202c-4c6e-9259-dd7a6dc01c97.png :width: 200px ::: :::

Stile di programmazione funzionale

Abbiamo visto che possiamo passare **funzioni** come se fossero oggetti, per poi **applicarle/eseguirle** ai dati da elaborare

- esempio: la funzione che passiamo per il parametro `key` di `sorted` o `min` o `max`

Ci sono molte altre operazioni ottenute dall'applicazione di una funzione ad un gruppo di elementi

Come "applicare/eseguire" una funzione passata come algomento

```
def applica_funzione(funzione, valore): # l'argomento 'funzione' riceve un oggetto di tipo Function
    return funzione(valore)             # eseguo la Function fornendogli il valore
```

Basta usare il **nome della variabile che contiene l'oggetto funzione**, come se fosse il nome della funzione

In [1…

```
# Esempio
def quadrato(X): return X**2
def cubo(X):     return X**3

print(quadrato, cubo)

def calcola_funzione(funzione, X):
```

```
    return funzione(X)

print('calcolo il cubo     di 5:', calcola_funzione(cubo,     5))
print('calcolo il quadrato di 5:', calcola_funzione(quadrato, 5))
```

```
<function quadrato at 0x1076ec040> <function cubo at 0x1076ec0e0>
calcolo il cubo     di 5: 125
calcolo il quadrato di 5: 25
```

Trasformazioni (`map`)

Spesso dobbiamo trasformare una sequenza di valori in un nuova sequenza, è una operazione molto comune che si può semplificare usando la funzione **`map`** che ha solo bisogno di sapere quale trasformazione applicare a ciascun dato (o gruppo di dati)

- `map(funzione, contenitore1, contenitore2, ...)` produce un contenitore in cui ciascun elemento è ottenuto eseguendo la `funzione` sul primo elemento dei contenitori, sul secondo, sul terzo e così via

La generatore risultante ha tanti valori quanti il più piccolo contenitore

Invece di scrivere

```
trasformati = []
for i in range(len(L1)):
    x = L1[i]
    y = L2[i]
    z = L3[i]
    trasformati.append(funzione(x,y,z))
```

Che può dare `IndexError` se L1 è più lunga di L2 o di L3

Oppure di scrivere

```
trasformati = []
for x,y,z in zip(L1, L2, L3):
    trasformati.append(funzione(x,y,z))
```

possiamo scrivere

```
trasformati = map(funzione, L1, L2, L3)
```

In [2…
```
help(map)
```

```
Help on class map in module builtins:

class map(object)
 |  map(func, *iterables) --> map object
 |
 |  Make an iterator that computes the function using arguments from
 |  each of the iterables.  Stops when the shortest iterable is exhausted.
 |
 |  Methods defined here:
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __next__(self, /)
 |      Implement next(self).
 |
 |  __reduce__(...)
 |      Return state information for pickling.
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  __new__(*args, **kwargs)
 |      Create and return a new object.  See help(type) for accurate signature.
```

In [3…
```
# Esempio : sommiamo elemento per elemento 3 contenitori di 6, 5, e 4 elementi
L1 : list[int]      = [  1,  3,  6, 99, 12, 42] # gli ultimi sono ignorati
T2 : tuple[int,...] = (  4, 67, 12, 91,  3)     # l'ultimo è ignorato
S3 : set[int]       = { 54, 67, 23, 90 }        # l'ordine potrebbe essere diverso -> 90 67 54 23

somme = []
for A,B,C in zip(L1, T2, S3):
```

```
    somme.append(A + B + C)

print(*somme)
```

```
95 137 72 213
```

In [4…
```
# Esempio : sommiamo elemento per elemento 3 contenitori di 6, 5, e 4 elementi
L1 : list[int]      = [  1,  3,  6, 99, 12, 42] # gli ultimi sono ignorati
T2 : tuple[int,...] = (  4, 67, 12, 91,  3)     # l'ultimo è ignorato
S3 : set[int]       = { 54, 67, 23, 90 }        # l'ordine potrebbe essere diverso -> 90 67 54 23

def somma(A:int,B:int,C:int) -> int:
    return A + B + C

somme = map( somma, L1, T2, S3 )
print(*somme)
```

```
95 137 72 213
```

In [5…
```
# lo possiamo scrivere anche con una lambda perchè **somma calcola una sola espressione**
somme = map( lambda x,y,z: x+y+z , L1, T2, S3 )
print(*somme)
```

```
95 137 72 213
```

In [6…
```
# oppure lo possiamo scrivere con una list comprehension e zip (per lo stesso motivo)
somme2 = [ x+y+z for x,y,z in zip(L1,T2,S3) ]
print(*somme2)
```

```
95 137 72 213
```

Funzione `filter` per estrarre da un contenitore gli elementi che soddisfano una condizione

Invece di scrivere

```
sottoinsieme = []
for elemento in contenitore:
    if selettore(elemento):             # solo se la condizione è vera
        sottoinsieme.append(elemento)   # lo aggiungo al risultato
```

Scriviamo

```
sottoinsieme = filter( selettore, contenitore )
```

Basta che la funzione `selettore` torni un risultato True/False (tenere/buttare)

In [7…

```
help(filter)
```

```
Help on class filter in module builtins:

class filter(object)
 |  filter(function or None, iterable) --> filter object
 |
 |  Return an iterator yielding those items of iterable for which function(item)
 |  is true. If function is None, return the items that are true.
 |
 |  Methods defined here:
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __next__(self, /)
 |      Implement next(self).
 |
 |  __reduce__(...)
 |      Return state information for pickling.
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  __new__(*args, **kwargs)
 |      Create and return a new object.  See help(type) for accurate signature.
```

In [8…

```
# Esempio: estraiamo da un gruppo di valori tutti i valori multipli di 5
from random import choices
```

```
valori = choices(range(1000), k=20)         # genero 20 valori casuali

multipli_di_5 = []
for x in valori:
    if x%5==0:
        multipli_di_5.append(x)

print('valori       : ', *valori)
print('multipli di 5: ', *multipli_di_5)
```

```
valori       :  3 468 482 694 224 823 871 748 343 218 954 356 222 929 899 711 235 93 51 401
multipli di 5:  235
```

In [9…

```
# Facciamolo con filter ed una funzione che torna vero/falso
def is_divisible_by_5(X):  return X%5==0

multipli_di_5 = filter(is_divisible_by_5, valori)

print('valori       : ', *valori)
print('multipli di 5: ', *multipli_di_5)
```

```
valori       :  3 468 482 694 224 823 871 748 343 218 954 356 222 929 899 711 235 93 51 401
multipli di 5:  235
```

In [1…

```
# Lo possiamo scrivere anche con una lambda che torna vero/falso
multipli_di_5 = filter(lambda X: X%5==0, valori)

print('multipli di 5: ', *multipli_di_5)
```

```
multipli di 5:  235
```

In [1…

```
# oppure con una list-comprehension condizionata
multipli_di_5 = [ X for X in valori if X%5==0 ]

print('multipli di 5: ', *multipli_di_5)
```

```
multipli di 5:  235
```

AND logico di più condizioni (`all`)

Spesso ci troviamo a voler controllare se una certa condizione è True per tutti gli elementi di un contenitore e scriviamo

```
def controlla_tutti(contenitore):
    for el in contenitore:
        if not condizione(el):  # cerco un controesempio
            return False        # se lo trovo torno False
    return True                 # non ne ho trovato nessuno, torno True
```

Questo si può semplificare usando la funzione `all(contenitore_di_True_False)` che torna True se tutti gli elementi forniti sono True

```
def controlla_tutti(contenitore):
    return all( map( condizione, contenitore ) )
```

`map` crea la lista di True/False e `all` controlla che siano tutti True

NOTA: **smette non appena ha trovato un False** (comportamento short-circuit)

In [1…
```
help(all)
```

```
Help on built-in function all in module builtins:

all(iterable, /)
    Return True if bool(x) is True for all values x in the iterable.

    If the iterable is empty, return True.
```

In [1…
```
# Esempio che controlla se tutti i valori elencati sono pari
L = [ 2, 4, 73, 94, 0, -34 ]     # ce n'è uno dispari

def is_pari(X): return X%2==0

sono_tutti_pari = all( map( is_pari, L) )

print('Sono pari? ', sono_tutti_pari)
```

```
Sono pari?  False
```

In [1…
```
# possiamo scriverlo con una lambda
L = [ 2, 4, 72, 94, 0, -34 ]     # ora sono pari
```

```
sono_tutti_pari = all( map( lambda X: X%2==0 , L) )

print('Sono pari? ', sono_tutti_pari)
```

```
Sono pari?  True
```

In [1…

```
# oppure con una list comprehension
L = [ 2, 4, 72, 94, 0, -34 ]      # tutti pari

sono_tutti_pari = all( X%2==0 for X in L ) # costruiamo implicitamente una tupla di True/False

print('Sono pari? ', sono_tutti_pari)
```

```
Sono pari?  True
```

OR logico di più condizioni (`any`)

Spesso ci troviamo a voler controllare se una certa condizione è True per almeno uno degli elementi di un contenitore e scriviamo

```
def esiste_uno(contenitore):
    for el in contenitore:
        if condizione(el):  # cerco almeno un caso vero
            return True
    return False            # se non ne ho trovato nessuno torno False
```

Questo si può semplificare usando la funzione `any(contenitore_di_True_False)` che torna False se tutti gli elementi forniti sono False

```
def esiste_uno(contenitore):
    return any( map( condizione, contenitore ) )
```

`map` crea la lista di True/False e `any` controlla che ce ne sia almeno uno True

NOTA: **smette non appena ha trovato il True** (comportamento short-circuit)

In [1…

```
help(any)
```

```
Help on built-in function any in module builtins:

any(iterable, /)
    Return True if bool(x) is True for any x in the iterable.

    If the iterable is empty, return False.
```

In [1…
```python
# Esempio che controlla se almeno uno dei valori elencati è dispari
L = [ 2, 4, 72, 28, 94, 0, -34 ]

def is_dispari(X): return X%2!=0

almeno_uno_dispari = any( map( is_dispari, L) )

print("Ce n'è uno dispari? ", almeno_uno_dispari)
```

```
Ce n'è uno dispari?  False
```

In [1…
```python
# possiamo scriverlo con una lambda
L = [ 2, 4, 72, 27, 94, 0, -34 ]

almeno_uno_dispari = any( map( lambda X: X%2!=0, L) )

print("Ce n'è uno dispari? ", almeno_uno_dispari)
```

```
Ce n'è uno dispari?  True
```

In [1…
```python
# oppure con una list comprehension
L = [ 2, 4, 72, 27, 94, 0, -34 ]

almeno_uno_dispari = any( X%2!=0 for X in L )

print("Ce n'è uno dispari? ", almeno_uno_dispari)
```

```
Ce n'è uno dispari?  True
```

Analisi: Anagrammi

Due parole sono anagrammi se **contengono le stesse lettere lo stesso numero di volte**

- Soluzione 1: contiamo le lettere

In [2…
```python
def isAnagramma(testo : str, testo1 : str) -> bool:
    # conto i caratteri nelle due stringhe    O(n)
    # confronto i due dizionari               O(n)
    def conta_lettere(stringa : str) -> dict[str,int]:
        frequenze : dict[str,int] = {}
        for c in stringa:
            frequenze[c] = frequenze.get(c, 0) + 1
        return frequenze
    frequenze1 = conta_lettere(testo)
    frequenze2 = conta_lettere(testo1)
    return frequenze1 == frequenze2

isAnagramma('onepaper', 'paperino')
```

Out[2…
```
False
```

Forma canonica

Altre definizioni, sono anagrammi se:

- **una si ottiene dall'altra per spostamento di lettere**
- oppure **hanno la stessa forma "standardizzata"** (o "forma canonica")

NOTA: la forma "canonica" deve essere **unica**

Altro esempio di riduzione a forma canonica: **formula booleana ==> forma SOP ==> tabella di verità**

Come **"standardizzare"** le parole in modo che siano uguali?

- **ordiniamo le lettere**

In [2…
```python
def isAnagramma2(testo : str, testo1 : str) -> bool:
    # ordino le due stringhe      O(n log(n))
```

```
    # le confronto                  O(n)
    ordinata1 = sorted(testo)
    ordinata2 = sorted(testo1)
    return ordinata1 == ordinata2

isAnagramma2('paperone','eaepnrpo')
```

Out[2… True

Analisi: Realizzazione di un database come lista di dizionari

- ciascuna scheda è un dizionario **info -> valore**
- il database è una lista di schede

In [2…
```
Scheda = dict[str, str]  # una Scheda è un dizionario 'informazione' -> 'valore'
Agenda = list[Scheda]    # una Agenda è una lista di Schede

agenda : 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 4
    {'nome': 'Paperon', 'cognome':"de' Paperoni", 'telefono':'555-99999', 'indirizzo': 'colle Papero 1',
    {'nome': 'Archimede','cognome':'Pitagorico',  'telefono':'555-11235', 'indirizzo': 'colle degli Inve
    {'nome': 'Pietro',  'cognome':'Gambadilegno', 'telefono':'555-66666', 'indirizzo': 'via dei Ladri 13
    {'nome': 'Trudy',   'cognome':'Gambadilegno', 'telefono':'555-66666', 'indirizzo': 'via dei Ladri 13
    {'nome': 'Topolino','cognome':'Mouse',        'telefono':'555-12345', 'indirizzo': 'via degli Invest
    {'nome': 'Minnie',  'cognome':'Mouse',        'telefono':'555-54321', 'indirizzo': 'via di M.me Curi
    {'nome': 'Pippo',   'cognome':"de' Pippis",   'telefono':'555-33333', 'indirizzo': 'via dei Pioppi 1
    ]
```

Ricerca di un record in tempo lineare $O(N)$

Per cercare una scheda con un certo **valore** in una data **colonna**:

- scandisco tutte le schede
  - se la scheda corrente corrisponde, la ritorno
- oppure torno None

In [2…
```
from typing import Optional
# per cercare un numero di telefono (un record)
# sapendo quale colonna usare e quale valore cerchiamo
def cerca_lineare(ag      : Agenda,
                  colonna : str,
                  valore  : str) -> Optional[Scheda] :
    # per tutti i record dell'agenda
    for record in ag:
        # se *la colonna esiste ed inoltre contiene il valore cercato*
        if corrisponde_alla_query(record, colonna, valore):
            # torniamo il record
            return record
    # altrimenti alla fine torniamo None
    return None

cerca_lineare(agenda, 'cognome', 'Pitagorico')
```

Out[2…
```
{'nome': 'Archimede',
 'cognome': 'Pitagorico',
 'telefono': '555-11235',
 'indirizzo': 'colle degli Inventori 1',
 'città': 'Paperopoli'}
```

In [2…
```
# per vedere se un record corrisponde alla query (nome della colonna e valore cercato)
def corrisponde_alla_query(riga    : Scheda,
                           colonna : str,
                           valore  : str) -> bool:
    return (colonna in riga                # la colonna deve esistere
            and riga[colonna] == valore)   # E il valore deve corrispondere
```

Ricerca con query multicolonna

In [2…

```python
# una query è un dizionario colonna -> valore
# per cercare su più colonne
def cerca_multicolonna_lineare(
    ag : Agenda, query : Scheda) -> list[Scheda] :
    # IN : agenda, query: coppie colonna/valore (con un dizionario)
    # OUT: lista dei record che soddisfano tutte le condizioni
    # all'inizio l'elenco di record risultante è []
    trovati = []
    for record in ag: # scandiamo l'agenda e per tutti i record
        # se *corrispondono alla query*
        if corrisponde_alla_query_multicolonna(record, query):
            trovati.append(record) # aggiungo il record all'output
    # torno l'elenco di record
    return trovati
```

In [2…

```python
# con una list comprehension
def cerca_multicolonna_lineare(
    ag : Agenda, query : Scheda) -> list[Scheda] :
    return [record  for record in ag
                    if corrisponde_alla_query_multicolonna(record,query) ]
```

In [2…

```python
# per determinare se un record corrisponde ad una query
def corrisponde_alla_query_multicolonna(
    record : Scheda, query : Scheda ) -> bool :
    # per ciascuna delle coppie colonna : valore cercato
    for colonna, cercato in query.items():
        # se colonna NON è nel record e o non ha quel valore
        if not corrisponde_alla_query(record, colonna, cercato):
            # torno False
            return False
    # altrimenti alla fine torno True
    return True

# MA QUESTO NON E' ALTRO CHE UN AND LOGICO!

Q = { 'città' : 'Topolinia',
     'cognome' : 'Gambadilegno'  }
```

```
cerca_multicolonna_lineare(agenda,Q)
```

Out[2…

```
[{'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'}]
```

RECAP: Stile di programmazione funzionale

- **all** : torna True se TUTTI i valori sono True (e smette al primo False)
- **any** : torna True se ALMENO un valore è True (e smette al primo True)
- **filter** : torna solo gli elementi per cui è vero un **predicato**
- **map** : torna una "lista" di elementi trasformati
- ...

In [2…

```
# per determinare se un record corrisponde ad una query
# versione funzionale
def corrisponde_alla_query_FUN(
    record : Scheda, query : Scheda ) -> bool :
    return all(
        corrisponde_alla_query(record, chiave, valore) # tutte le info della query devono essere vere
        for chiave, valore in query.items()
    )

R = {'1':'a', '2':'b', '3':'c'}
Q = {'2':'b', '1':'a'}
corrisponde_alla_query_FUN(R, Q)
```

Out[2…

```
True
```

In [3…

```python
# per determinare se un record corrisponde ad una query
# versione con insiemi
def corrisponde_alla_query_SET(
    record : Scheda, query : Scheda ) -> bool :
    # L'intersezione di record e query è == query
    # ovvero query - record == set vuoto
    set_query  = set(query.items())
    set_record = set(record.items())
    return set_query - set_record == set()

# NOTA: la scheda può contenere altri valori oltre a quelli della query

corrisponde_alla_query_SET(R,Q)
```

Out[3…

```
True
```

In [3…

```python
# per cercare su più colonne con list comprehension
def cerca_multicolonna_lineare_LC(
    ag : Agenda, query : Scheda) -> list[Scheda] :
    return [
        record for record in ag
               if corrisponde_alla_query_FUN(record, query)
    ]

Q = { 'città' : 'Topolinia',
     'cognome' : 'Gambadilegno'  }
cerca_multicolonna_lineare_LC(agenda,Q)
```

Out[3…

```
 [{'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'}]
```

In [3…

```python
# usando una funzione filtro che sceglie cosa tenere
def cerca_multicolonna_lineare_filter(ag : Agenda, query : Scheda) -> list[Scheda] :
    # filtro i record con la funzione corrisponde_a_q
    # che usa implicitamente il parametro query della funzione che la racchiude
    def corrisponde_a_q(record):
        # le "inner functions" possono leggere il namespace locale
        return corrisponde_alla_query_SET(record, query)
    return list(filter(corrisponde_a_q, ag))

cerca_multicolonna_lineare_filter(agenda,Q)
```

Out[3…
```
[{'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'}]
```

In [3…
```python
def cerca_multicolonna_lineare_lambda(ag : Agenda, query : Scheda) -> list[Scheda] :
    # lo stesso si può fare con una lambda e filter
    return list(filter(
        lambda record: corrisponde_alla_query_SET(record, query),
        ag))
```

Ordinamento delle schede

Proviamo ad ordinare le schede rispetto ad una colonna il cui nome viene fornito come parametro

In [3…
```python
# per ordinare una agenda rispetto ad una colonna
def ordina_rispetto_a_colonna(ag : Agenda, colonna : str) -> Agenda :
    # usiamo sorted con il parametro key
    # che deve tornare il valore rispetto al quale vogliamo ordinare
    def criterio(riga : Scheda) -> str:
        return riga.get(colonna, '')  # se la colonna non esiste le schede sono tutte equivalenti
    return sorted(ag, key=criterio)

print(*ordina_rispetto_a_colonna(agenda, 'cognome'), sep='\n')
```

```
{'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', 'cit
tà': 'Topolinia'}
{'nome': 'Paperino', 'cognome': 'Paolino', 'telefono': '555-1313', 'indirizzo': 'via dei Peri 113', 'cit
tà': 'Paperopoli'}
{'nome': 'Gastone', 'cognome': 'Paperone', 'telefono': '555-1717', 'indirizzo': 'via dei Baobab 42', 'ci
ttà': 'Paperopoli'}
{'nome': 'Archimede', 'cognome': 'Pitagorico', 'telefono': '555-11235', 'indirizzo': 'colle degli Invent
ori 1', 'città': 'Paperopoli'}
{'nome': 'Paperon', 'cognome': "de' Paperoni", 'telefono': '555-99999', 'indirizzo': 'colle Papero 1',
'città': 'Paperopoli'}
{'nome': 'Pippo', 'cognome': "de' Pippis", 'telefono': '555-33333', 'indirizzo': 'via dei Pioppi 1', 'ci
ttà': 'Topolinia'}
```

In [3…

```python
# per ordinare una agenda rispetto ad una colonna
# stavolta usando lambda
def ordina_rispetto_a_colonna_L(ag : Agenda, colonna : str) -> Agenda :
    # usiamo sorted con il parametro key
    # che deve tornare il valore rispetto al quale vogliamo ordinare
    return sorted(ag, key=lambda riga: riga.get(colonna, ''))

print(*ordina_rispetto_a_colonna_L(agenda, 'cognome'), sep='\n')
```

```
{'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', 'cit
tà': 'Topolinia'}
{'nome': 'Paperino', 'cognome': 'Paolino', 'telefono': '555-1313', 'indirizzo': 'via dei Peri 113', 'cit
tà': 'Paperopoli'}
{'nome': 'Gastone', 'cognome': 'Paperone', 'telefono': '555-1717', 'indirizzo': 'via dei Baobab 42', 'ci
tà': 'Paperopoli'}
{'nome': 'Archimede', 'cognome': 'Pitagorico', 'telefono': '555-11235', 'indirizzo': 'colle degli Invent
ori 1', 'città': 'Paperopoli'}
{'nome': 'Paperon', 'cognome': "de' Paperoni", 'telefono': '555-99999', 'indirizzo': 'colle Papero 1',
'città': 'Paperopoli'}
{'nome': 'Pippo', 'cognome': "de' Pippis", 'telefono': '555-33333', 'indirizzo': 'via dei Pioppi 1', 'ci
tà': 'Topolinia'}
```

Tempi

tutti i tempi sono lineari rispetto al numero di schede dell'agenda (perchè è disordinata), si può far meglio?

potremmo costruire un indice ...

- usando un dizionario **valore -> lista di indici in cui appare quel valore**
- con velocità di ricerca molto alta (O(1) nel caso medio per i dizionari)

In [3…

```python
# un indice è un dizionario valore -> lista di posizioni nella Agenda che lo contengono
Indice = dict[str,list[int]]
# per costruire un indice
def crea_indice(ag: Agenda, colonna : str ) -> Indice:
    # IN agenda, colonna
    # OUT lista che contiene coppie [valori]:
    #     posizioni originali dei record che contengono quel valore
    # all'inizio il dizionario valori:posizioni è vuoto
    indice : Indice = {}
```

```
    # per ogni record dell'agenda e sua posizione, ottenuta con enumerate
    for i,record in enumerate(ag):
        # estraggo dal record il valore per quella colonna (o stringa vuota se non c'è)
        valore = record.get(colonna, '')
        # e aggiungo la posizione alla lista associata a quel valore (lista vuota se non c'è)
        indice[valore] = indice.get(valore, []) + [i]
    # ritorno il dizionario valori -> posizioni
    return indice

indice_cognome = crea_indice(agenda, 'città')
print(indice_cognome)
```

```
{'Paperopoli': [0, 1, 2, 3], 'Topolinia': [4, 5, 6, 7, 8]}
```

In [3…

```
## Visto che ci siamo costruiamo TUTTI gli indici
Indici = dict[str , Indice]
colonne : list[str] = ['nome','cognome','telefono','indirizzo','città']
indici : Indici = {
    colonna : crea_indice(agenda, colonna)
        for colonna in colonne
}
indici
```

Out[3…
```
{'nome': {'Paperino': [0],
  'Gastone': [1],
  'Paperon': [2],
  'Archimede': [3],
  'Pietro': [4],
  'Trudy': [5],
  'Topolino': [6],
  'Minnie': [7],
  'Pippo': [8]},
 'cognome': {'Paolino': [0],
  'Paperone': [1],
  "de' Paperoni": [2],
  'Pitagorico': [3],
  'Gambadilegno': [4, 5],
  'Mouse': [6, 7],
  "de' Pippis": [8]},
 'telefono': {'555-1313': [0],
  '555-1717': [1],
  '555-99999': [2],
  '555-11235': [3],
  '555-66666': [4, 5],
  '555-12345': [6],
  '555-54321': [7],
  '555-33333': [8]},
 'indirizzo': {'via dei Peri 113': [0],
  'via dei Baobab 42': [1],
  'colle Papero 1': [2],
  'colle degli Inventori 1': [3],
  'via dei Ladri 13': [4, 5],
  'via degli Investigatori 1': [6],
  'via di M.me Curie 1': [7],
  'via dei Pioppi 1': [8]},
 'città': {'Paperopoli': [0, 1, 2, 3], 'Topolinia': [4, 5, 6, 7, 8]}}
```

Ricerca tramite indici in tempo $O(1)$

- ricerca singola

- ricerca multipla

In [3…
```python
## Per cercare con ricerca singola avendo l'indice
def cerca_con_indice( ag : Agenda, index : Indice, valore : str ):
    # se il valore è nell'indice
    # torno tutti i record nelle posizioni indicate
    # se non è nell'indice torno lista vuota
    return [ ag[i] for i in index.get(valore,[]) ]

cerca_con_indice(agenda, indici['cognome'], 'Gambadilegno')
```

Out[3…
```
 [{'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'}]
```

In [3…
```python
# Nella ricerca multi-colonna mi servono tutte le Schede
# che appaiono in tutti gli indici che contengono tutti i valori cercati
# quindi devo intersecare gli insiemi delle posizioni che soddisfano
# ciascuna chiave:valore che viene richiesta dalla query
def cerca_con_indici( ag : Agenda, indexes : Indici, query: Scheda ) -> Agenda :
    # inizialmente sono valide tutte le righe
    posizioni_ok = set(range(len(ag)))   # set di tutte le posizioni possibili
    # per ciascuna coppia   colonna -> valore    della query
    for colonna,valore in query.items():
        # se il nome della colonna non appare negli indici torno []
        if colonna not in indexes: return []
        # se il valore non appare nell'indice della colonna torno []
        if valore not in indexes[colonna]: return []
        # altrimenti prendo gli insiemi delle posizioni
        # corrispondenti ai valori delle colonne
        righe_ok = indexes[colonna][valore]
```

```
        posizioni_ok &= set(righe_ok)        #  e li interseco perchè devono soddisfare tutti i vincoli
    # alla fine torno la lista di righe rimaste nell'insieme
    # delle posizioni che ora soddisfano tutte le chiave->valore
    return [ ag[i] for i in posizioni_ok ]

cerca_con_indici(agenda, indici, {'cognome':'Mouse',
                                  'città':'Topolinia'})
```

Out[3…

```
[{'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'}]
```

PAUSA

Files e filesystem

- i files sono formati da blocchi di bytes consecutivi
- 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 = 'rt',
     encoding  : str = <encoding> ) -> File
```

`filename` è una stringa che indica il percorso 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`

Il path di un file

- **Linux/Mac** se inizia con `'/'` allora è un percorso **assoluto** che parte dalla radice dell'albero delle directory
- **Windows** se inizia con `<drive letter>:\\` è un percorso **assoluto** che parte dalla radice del drive
- altrimenti è un percorso **relativo alla directory corrente**
- per risalire di un livello si usa `'..'` (due punti)
- come separatore dei nomi delle directory e file Python usa il separatore del sistema operativo
  - `'\\'` backslash in Windows (va scritto doppio perchè è una sequenza di escape)
  - `'/'` slash in Unix/Linux/Macos

CONSIGLIO usate SEMPRE `'/'` lo slash anche in Windows, che Python sa usarlo correttamente

`mode` è una stringa che indica in che modo 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
- `x` (**exclusive**) crea un nuovo file e da errore se esiste già

che può essere seguito dal carattere

- `t` il file viene aperto in modo testo (il contenuto viene interpretato come caratteri - **default**)
- `b` il file viene aperto in modo binario (il contenuto non viene interpretato automaticamente)
  - file binari: **png, gif, pdf, mp3, mp4, jpeg, mov, doc, ...**
- `+` per leggere **E ANCHE** modificare il contenuto

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

Esempi

- `utf-8` codifica unicode
- `latin` codifica latin
- `ascii` codifica ASCII
- `cp1252` codifica di default usata da Windows
- **...**

`open(filename)` per default apre il file **in lettura, di testo, nel'encoding di default del sistema operativo** che è

- **utf-8** per Linux e Macos
- **Windows-1252 oppure utf-16** per Windows
- (ma in realtà dipende dalla codifica del file da leggere)

CONSIGLIO indicate sempre l'encoding del file **'utf-8'** quando lo create

ATTENZIONE: una volta usato, il file deve essere "chiuso"

In questo modo vengono:

- **rilasciate le strutture dati** del Sistem Operativo liberando memoria

- **completate le scritture** dei file su HD !!!!! (se mode='w')

In [4…
```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: 'utf8' di default su unix

# 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 mettere 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!!!*
```

In [4…
```python
# Controlliamo
!ls -al pippo.txt
!cat pippo.txt
```

```
-rw-r--r--  1 andrea  staff  71 Oct 26 12:20 pippo.txt
pippo è andato al mare
kjglahgkjhkj kajhhk gj
pippo è andato al mare
```

Aprire e chiudere (automaticamente) 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 [4…

```
# 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[42], line 7
      5     print("kjglahgkjhkj kajhhk gj", file=F)
      6     F.write("pippo è andato al mare\n")
----> 7     assert False # anche se inserisco un errore il file viene salvato
      9 # BAD STYLE: non conviene usare direttamente open e close
     10 # - errori ed eccezioni
     11 # - semplici dimenticanze

AssertionError:
```

In [4…

```
# Controlliamo
!ls -alh topolino.txt
!cat topolino.txt
```

```
-rw-r--r--  1 andrea  staff    71B Oct 26 12:20 topolino.txt
pippo è andato al mare
kjglahgkjhkj kajhhk gj
pippo è andato al mare
```

Spostarsi nel file con `seek`

In [4…

```python
# per posizionarsi in un certo punto del file (0=inizio)

# se voglio leggere il file e stamparlo 2 volte  di seguito
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 [4…

```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, di una riga, di tutte le righe

NOTA Sistemi Operativi diversi usano diversi separatori di riga

- **Windows** `'\r\n'` (carriage return, line feed)
- **Unix** `'\n'` (solo line feed)

Python riconosce entrambi i casi e converte automaticamente la coppia `'\r\n'` dei file di testo per Windows nel solo `'\n'`

CONSIGLIO usate SOLO `'\n'` per andare a capo quando create file di testo

Per leggere il contenuto di un file ci servono le funzioni:

- **`file.read()`** che permette di leggerne tutto il contenuto (o una parte)
- **`file.readline()`** che permette di leggere una riga da un file di testo
- **`file.readlines()`** che permette di leggere tutte le righe da un file di testo
- **`file.close()`** per chiudere il file, finire di scriverlo su disco e rilasciare la memoria usata

In [4…

```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 le righe assieme (sempre per file di testo)

# 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 [4…
```python
# 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[4…
```
['pippo è andato al mare\n',
 'kjglahgkjhkj kajhhk gj\n',
 'pippo è andato al mare\n']
```

NOTA: ciascuna riga termina con `'\n'` (tranne talvolta l'ultima se nel file non c'era)

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

In [4…
```python
# 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)

# il ciclo si ferma automaticamente quando il file è finito e non ci sono più righe
```

```
pippo è andato al mare

kjglahgkjhkj kajhhk gj

pippo è andato al mare
```

Con il metodo `file.seek(posizione : int)` posso spostarmi nel file

Per i file di testo, che possono contenere righe di lunghezza diversa, è utile solo per **tornare all'inizio**

In [4…

```python
with open('topolino.txt', encoding='utf8') as F:
    for riga in F:      # scandisco le righe del file
        print(riga)     # e le stampo
    print('================================')
    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 (Windows, Unix, Macos, ...)

In [5…
```
!file files/*.txt       # esamino i file per vedere di che tipo sono
```

```
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/testo.txt:        Unicode text, UTF-8 text
files/testo2.txt:       ASCII text
```

In [5…
```
# files e loro encoding
# BOM = # Byte Order Mark: https://en.wikipedia.org/wiki/Byte_order_mark
files = {
    'files/holmes.txt'       : 'utf-8-sig', # utf-8 preceduto dal BOM
    'files/alice.txt'        : 'utf-8-sig',
    'files/frankenstein.txt' : 'utf-8-sig',
    'files/alice_it.txt'     : 'latin',
    'files/prince.txt'       : 'utf-8-sig'
    }
```

In [5…
```
# leggo 'alice_it.txt' e ne stampo i primi 400 caratteri
with open('files/alice_it.txt', encoding='latin') as F:
    testo = F.read()
print(testo[:400])
```

```
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/

QUESTO E-BOOK:

TITOLO: Alice nel paese delle meraviglie
AUTORE: Dodgson, Charles Lutwidge (alias Lewis Carroll)
NOTE:

DIRITTI D'AUTORE: no

LICENZA: questo testo è distribuito con la licenza
 specificata al segue
```

In [5…

```python
# oppure leggo solo 400 caratteri dal file
with open('files/alice_it.txt', encoding='latin') as F:
    testo = F.read(400)
print(testo)
```

```
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/

QUESTO E-BOOK:

TITOLO: Alice nel paese delle meraviglie
AUTORE: Dodgson, Charles Lutwidge (alias Lewis Carroll)
NOTE:

DIRITTI D'AUTORE: no

LICENZA: questo testo è distribuito con la licenza
 specificata al segue
```

Come scrivere in un file

Si usa il metodo **write**

```
with open(filename, mode='w', encoding='utf8') as FILE:
    FILE.write(stringa_o_sequenza_di_byte)
```

Oppure la funzione **print** col parametro **file**

```
with open(filename, mode='w', encoding='utf8') as FILE:
    print(cose_da_stampare ..., file=FILE)
```

che ci assicura di chiudere il file in qualsiasi modo siamo usciti (anche in caso d'errore)

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
```