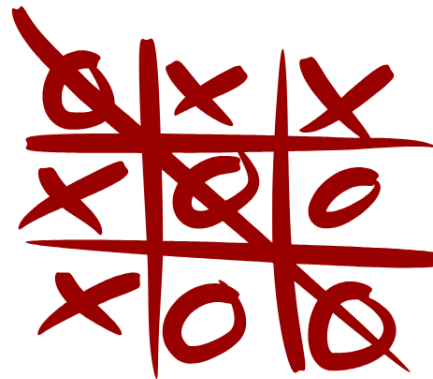




UNIVERSITÀ
DI PARMA

strutture dati



insieme

- collezione di elementi *non ordinata* e *senza ripetizioni*
 - senza chiavi o indici
- metodi
 - aggiunta (*add*) e rimozione (*discard*)
- operatori
 - appartenenza (*in*), unione (*|*) e intersezione (*&*)

```
numeri = {1, 4, 5}
numeri.add(4)                                # {1, 4, 5}
intersezione = numeri & {4, 5, 6}            # {4, 5}, intersezione
unione = numeri | {3, 4}                     # {1, 3, 4, 5}, unione

insieme_vuoto = set()                         # ⚠ {} is an empty dict
```

dizionario

- **dizionario** (*mappa, array associativo*)
- insieme non ordinato di **coppie chiave / valore**
- le **chiavi** sono **uniche**: hanno la funzione di **indice** per accedere al valore corrispondente
- le **chiavi** possono essere un qualsiasi **tipo immutabile** (int, str ...)



dizionario funzionalità

```
#definizione di dizionario {}  
tel = {"aldo": 2030, "giovanni": 3321}  
  
#accesso a un elemento  
print(tel["giovanni"]) #3321  
  
#aggiunta di una coppia chiave/valore  
tel["giacomo"] = 2044  
  
#visualizzazione dizionario  
print(tel)             #{'aldo': 2030, 'giovanni': 3321, 'giacomo': 2044}  
  
#visualizzazione chiavi  
print(list(tel))       ['aldo', 'giovanni', 'giacomo']
```

dizionario funzionalità

```
#lista di coppie
print(list(tel.items()))
#[('aldo', 2030), ('giovanni', 3321), ('giacomo', 2044)]

#sequenza di elementi
for k in tel.keys():
    print(k,tel[k])
#aldo 2030
#giovanni 3321
#giacomo 2044
```

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{pmatrix}$$

MATRICI

liste multidimensionali

```
a = [['A', 'B', 'C', 'D'],  
      ['E', 'F', 'G', 'H'],  
      ['I', 'L', 'M', 'N']]          # 2D
```

```
b = ['A', 'B', 'C', 'D',  
      'E', 'F', 'G', 'H',  
      'I', 'L', 'M', 'N']           # 1D
```

```
# conversione 2D -> 1D (dalla matrice a alla matrice b)  
indice = riga * num_colonne + colonna
```

```
# conversione 1D -> 2D (dalla matrice b alla matrice a)  
riga = indice // num_colonne  
colonna = indice % num_colonne
```

matrice - somma colonne

```
matrix = [[2, 4, 3, 8],  
          [9, 3, 2, 7],  
          [5, 6, 9, 1]]  
  
rows = len(matrix)  
cols = len(matrix[0])  
  
for x in range(cols):  
    total = 0  
    for y in range(rows):  
        val = matrix[y][x]  
        total += val  
    print("Col #", x, "sums to", total)
```


lista pseudo-matrice

```
matrix = [2, 4, 3, 8,  
          9, 3, 2, 7,  
          5, 6, 9, 1]  
rows = 3 # dato non ricavabile dalla pseudo-  
matrice  
cols = len(matrix) // rows  
  
for x in range(cols):  
    total = 0  
    for y in range(rows):  
        val = matrix[y * cols + x] # 2D -> 1D  
        total += val  
    print("Col #", x, "sums to", total)
```

matrici - inizializzazione

```
cols = 3      #dato noto
rows = 4      #dato noto
#inizializzazione di tutti gli elementi a ' '
matrix = [[' ' for x in range(cols)] for y in range(rows)]

#metodo alternativo
matrix = []
for y in range(rows):
    new_row = []
    for x in range(cols):
        new_row.append(' ')
    matrix.append(new_row)
```

strutture dati
esercizi



file CSV

- leggere una **matrice di interi** da un file testuale **CSV**
 - file CSV \Rightarrow **Comma-Separated Values**: valori riga per riga, separati da virgole
- memorizzare i dati in una lista semplice (**pseudo-matrice**)
- inferire le **dimensioni** della matrice (rows×cols) in base a:
 - numero di righe del file
 - numero di valori in una riga
- da angolo in basso a destra, **sommare sulla diagonale**

5,7,2,11
1,3,12,9
4,6,10,8

nell'esempio, sommare: $8 + 12 + 7$ (celle dove $cols - x == rows - y$)

incolonnamento dati

- visualizzare due *tabelle* con i caratteri *ASCII*
 - 4 righe x 24 colonne, codici da 32 a 126
- tabella 1: mostrare in *ordine* i caratteri, *colonna per colonna*
- tabella 2: mostrare in *ordine* i caratteri, *riga per riga*

```
$ ( , 048<@DHLPTX\`dhlptx|  
!%) -159=AEIMQUY]aeimquy}  
"&*.26:>BFJNRVZ^bfjnrvz~  
#'+/37;?CGKOSW[_cgkosw{
```

```
!"#$%&'() *+,-./01234567  
89: ;<=>?@ABCDEFGHIJKLMNO  
PQRSTUVWXYZ[\]^_`abcdefg  
hijklmnopqrstuvwxyz{|}~
```

*usare sempre due cicli for annidati: esterno su y, interno su x
in ogni posizione, calcolare il carattere da visualizzare: $x * \text{ROWS} + y...$*

~400 a.c. – scitala spartana

- una **scitala** (dal greco σκυτάλη = bastone) era una piccola bacchetta utilizzata dagli Spartani per trasmettere messaggi segreti
- il messaggio veniva scritto su di una striscia di pelle arrotolata attorno alla scitala, come se fosse stata una superficie continua
- una volta srotolata e tolta dalla scitala la striscia di pelle, era impossibile capire il messaggio
- la decifrazione era invece possibile se si aveva una bacchetta identica alla scitala del mittente: vi si arrotolava nuovamente la striscia di pelle ricostruendo la primitiva posizione
- si tratta del più antico metodo di crittografia per trasposizione conosciuto



scitala spartana

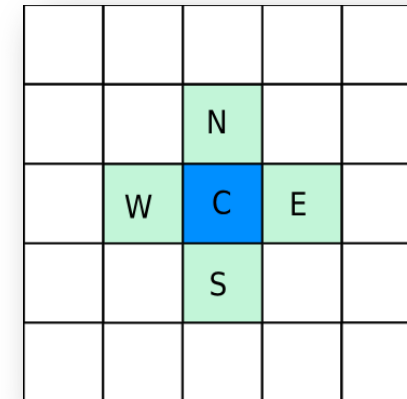
- leggere un intero file di testo
- inserire in una matrice i primi $W \times H$ caratteri
 - W colonne \times H righe, valori prefissati
 - riempire una riga della matrice dopo l'altra
 - da destra a sinistra, una riga alla volta (\rightarrow , \downarrow)
- scrivere il contenuto della matrice su console
 - scrivere una colonna della matrice dopo l'altra
 - prima riga su console = prima colonna della matrice...
 - dall'alto verso il basso, una colonna alla volta (\downarrow , \rightarrow)



usare una lista di liste (con dimensioni predefinite)

funzione di smooth

- scrivere una *funzione smooth*
 - **parametro**: matrice iniziale, di float
 - **risultato**: nuova matrice con smooth
 - matrici rappresentate come liste di liste
- **smooth**: per ogni cella in matrice iniziale
 - il risultato è la media dell'intorno
 - 5 valori: cella stessa e 4 adiacenti
 - attenzione alle celle esterne
 - sommare e contare solo i valori disponibili
 - 4 valori ai bordi, 3 valori agli angoli
- verificare la funzione con alcune matrici di test



		N		
	W	C	E	
		S		

spirale

- scrivere una funzione per riempire di **numeri crescenti** una **matrice** quadrata (o rettangolare)
 - seguire il percorso a spirale suggerito nella figura a fianco
 - dimensioni della matrice indicate dall'utente a runtime

*tenere traccia della direzione attuale (Δy , Δx)
avanzare fino al bordo o ad una cella già visitata,
poi cambiare la direzione in senso orario*

*coordinate raster, rotazione oraria di 90° : $(x', y') = (-y, x)$
in generale: $(x', y') = (x \cdot \cos(\theta) - y \cdot \sin(\theta), x \cdot \sin(\theta) + y \cdot \cos(\theta))$*

