SISTEMI INFORMATIVI PER IL MANAGEMENT MODULI 1 E 2

PRIMO APPELLO XX/06/2022

Esercizio 1 Excel (3 punti)

Si consideri il foglio Excel mostrato nella figura seguente. La colonna B contiene 100 valori casuali (il range contenente tali valori è B2:B101) tra 0 e 999. Vogliamo sapere quali tra i numeri 100, 200, 300, 400 e 500 sono stati generati. A tal fine, si scriva una formula logica in E2 che restituisce Vero se il numero 100 è stato generato, e falso in caso contrario. Si copi tale cella nelle celle sottostanti per completare la tabellina.
Si illustri brevemente il ragionamento seguito

Suggerimento: si utilizzi una formula in forma matrice per conteggiare il numero di occorrenze del valore cercato.

	A	B	C	D	E
1		*Valori Casuali tra 0 e 999*			
2		783		100	FALSO
3		618		200	FALSO
4		811		300	FALSO
5		706		400	FALSO
6		629		500	VERO
7		365			
8		500			
9		589			

Soluzione

La formula è la seguente.
=SOMMA(SE($B$2:$B$101=D2;1;0)) >=1
La funzione interna SE(), scritta in forma matrice, restituisce un range composto che avrà tanti 1 quante le occorrenze del valore cercato. Se la somma effettuata su tale range è maggiore di 1, significa che il valore cercato è stato generato e, pertanto, viene visualizzato VERO.

Esercizio 2 Excel (4 punti)

Si consideri il file Excel mostrato in figura. La colonna B riporta 4 stringhe di differente lunghezza. Vogliamo ordinare tali stringhe per lunghezza crescente, come mostrato in colonna D.

	A	B	C	D
1		***Dati***		***Dati Ordinati***
2		abcd		a
3		a		ab
4		ab		abc
5		abc		abcd
6				

Si combinino opportunamente le funzioni seguenti per ottenere il risultato voluto:

- Lunghezza().

- Dati.Ordina.Per(matrice; per_matrice1; [tipo_ordinamento1];…) dove gli input rappresentano: la matrice da ordinare, la matrice (o vettore) su cui basare l'ordinamento e il tipo di ordinamento.
- LET(nome1; valore1;calcolo;…) dove gli input sono: il nome che vogliamo assegnare ad una variabile, il valore assegnato a tale variabile, il calcolo che sfrutta la variabile definita.
- MAP(Matrice, lambda) dove gli input sono la matrice che si vuole trasformare e una funzione lamba che opera (dato per dato) sulla matrice da modificare.
- Lambda(parametro_o_calcolo,…)

Senza l'uso di LET il punteggio scende a 3 punti, senza l'utilizzo di MAP (sostituito dall'utilizzo di una colonna d'appoggio) il punteggio scende a 1.

Soluzione.

La formula è la seguente.

=LET(r;B2:B5;DATI.ORDINA.PER(r;MAP(r;LAMBDA(v;LUNGHEZZA(v)))))

In questo modo assegniamo alla variabile r il range contenente le stringhe da ordinare.
La funzione Lambda prede una stringa v e ne restituisce la lunghezza.
La funzione Map riceve il range r e, tramite la Lambda restituisce un range con le stesse dimensioni di r dove però le stringhe sono sostituite dalla loro lunghezza.
La funzione Dati.Ordina.Per, infine, ordina il range r in funzione dei valori contenuti nel range restituito da Map.

Esercizio 3 Python (3 punti)

Si considerino le seguenti tuple contenenti il nome, il cognome e gli anni di regno dei 7 re di Roma.

```
>>> Nomi = ('Tarquinio', 'Numa', 'Anco', 'Romolo', 'Tullo', 'Servio', 'Tarquinio')
>>> Cognomi = ('Prisco', 'Pompilio', 'Marzio', " ", 'Ostilio', 'Tullio', 'Il Superbo')
>>> Date = ('616-578 a.C.','715-673 a.C.','640-616 a.C.','753-716 a.C.', '672-640 a.C.',
                                                    '578-534 a.C.', '534-510 a.C')
```

L'ordine con cui sono stati inseriti i nomi dei 7 re è casuale, ma lo stesso ordine è stato utilizzato anche per i Gognomi e per le Date. Per cui, presa una generica posizione 'i', la tripletta (Nomi[i], Cognomi[i], Date[i]) descrive correttamente un re di Roma.
Si chiede di:

- Usare le tre tuple per creare, tramite list comprehension, una lista contenente 7 dizionari, uno per ogni re di Roma. Si scelgano a piacere le chiavi dei dizionari. Ad esempio la lista potrebbe essere così fatta: [{'nome': 'Tarquinio', 'cognome': 'Prisco', 'Da_a': '616-578 a.C.'} ,…]
- Usare la funzione sorted() o il metodo sort() per ordinare la lista in ordine cronologico, ossia dal primo re (Romolo) all'ultimo (Tarquinio il Superbo). Si noti che tutte le dati sono avanti Cristo.

Soluzione.

```
Re = [{'nome':N, 'cognone':C, 'da_a':D} for N,C,D in zip(Nomi,Cognomi,Date)]
Re_Ordinati = sorted(RR, key = lambda x: int(x['da_a'][:3]), reverse = True
```

Esercizio 4 Python (5 punti)

Si completi la funzione produttoria(), di modo che, operando ricorsivamente, restituisca la produttoria dei numeri ricevuti in ingresso.

```
def produttoria(*vals):
        …
```

Ad esempio:

```
>>> produttoria(2,4,6,8,10)
< 3840 >
```

Soluzione.

```
def produttoria(*fattori):
  if fattori == tuple(): return
  if len(fattori) == 1: return fattori[0]
  return fattori[0] * produttoria(*fattori[1:])
```

Esercizio 5 Python (16 o 6 punti)

Si consideri la seguente classe che permette di gestire matrici bidimensionali.

```
class Matrix:
  def __init__(self, rc = (4,4), f = lambda r, c: None):
    self.f = f
    self._rc = rc
    self._mtx = [[f(r,c) for c in range(rc[1])] for r in range(rc[0])]

  def shape(self):
    return self._rc

  @property
  def show(self):
    frt = lambda x: f'{str(x):>6s}' if x != None else f'{"Nan":>6s}'
    out = ""
    for row in self._mtx:
      out += " ".join([frt(r) for r in row] + ["\n"])
    print(out)

  def __str__(self):
    return f'Matrix with {self._rc[0]} row and {self._rc[1]} colums'

  def __repr__(self):
    …

  def __setitem__(self ,rc, val): #rc sarà la tupla con indice di riga e indice di colonna
    …

  def __getitem__(self, rc):
```

```
    …
  def ravel(self):
    …

  def __iter__(self):
    …

  @property
  def iter_row(self):
    …

  @property
  def iter_col(self):
    …

  def __matmul__(self, other):
    …
```

Si chiede di[1]:

- Descrivere brevemente il funzionamento dell'inizializzatore ed il ruolo giocato dalla funzione f. **(1 punto)**
- Creare due funzioni *f,* da passare al costruttore di modo che si crei: (i) la matrice identità e (ii) una matrice con tutti 1 sulla prima colonna, tutti 2 sulla seconda, e così via per le altre (**2 punti**)
- Completare i metodi __setitem__() e __getitem__() per avere accesso posizionale agli elementi della matrice. **(2 punti)**
- Completare il metodo ravel() di modo che restituisca una lista contenente tutti gli elementi della matrice letti riga per riga. **(2 punti)**.
- Sfruttare il metodo ravel prima definito[2], per completare il metodo __iter__ di modo di rendere iterabile l'oggetto. In particolare, l'iteratore restituirà uno alla volta i valori contenuti nella lista generata da ravel() a patto che non siano nulli. **(2 punti)**
- Completare i metodi iter_row() e iter_col() per generare due iteratori che restituiscono, rispettivamente, una riga alla volta e una colonna alla volta della matrice. **(3 punti)**
- Sfruttare i due iteratori iter_row() e iter_col() per completare il metodo __matmul__, associato all'operatore @ che indica il prodotto matriciale tra due matrici di dimensioni compatibili (es. M1 @ M2, esegue il prodotto delle due matrici). Nel farlo si generi un'eccezione se le matrici non sono compatibili, o se una o entrambe le matrici hanno valori nulli. **(5 punti)**

In alternativa:

- Si spieghi il funzionamento di tutti i metodi della classe che sono già stati definiti. **(3 punti)**
- Si sostituisca la list comprehension self._mtx = [[f(r,c) for c in range(rc[1])] for r in range(rc[0])] con due cicli for annidati. **(1.5 punti)**
- Si completi il metodo __repr__() di modo che restituisca una stringa che, se eseguita, genererebbe una matrice con le stesse dimensioni di quella di partenza, ma con valori tutti nulli (1.5 punti)

---

[1] Descrivere non vuol dire limitarsi a spiegare qual è il risultato di un metodo, ma dettagliarne il funzionamento.

[2] Quando un quesito suggerisce di ricorrere a metodi precedentemente definiti, qualora tali metodi non siano stati sviluppati, è comunque lecito assumere di averlo fatto e richiamarli nel codice.

Segue un esempio di utilizzo delle classi che una funzione lamba per generare una matrice con valori casuali tra 0 e 5:

```
>>> from random import randrange
>>> f_rand = lambda r,c: randrange(0,6)
>>> M1 = Matrix(rc = (3,2), f = f_rand)
>>> M2 = Matrix(rc = (2,3), f = f_rand)
>>> M3 = Matrix(rc = (3,3), f = f_rand)
>>> M4 = Matrix(rc = (2,2))
>>> M1[0,0] = 6 #inseriamo in alto a sinistra il valore 6, che non può essere stato generato con randrange(0,6)
>>> M1.show
< 6     1
  1     1
  5     5 >
>>> M2.show
< 2     2     3
  4     2     0 >
>>> M1.ravel()
< [6, 1, 1, 1, 5, 5] >
>>> list(M1.iter_col)
< [[6, 1, 5], [1, 1, 5]] >
>>> New = M1 @ M2
>>> New.show
< 16    14    18
    6     4     3
   30    20    15 >
>>> M1 @ M3
< Exception: Dimensioni incompatibili >
>>> M1 @ M4
< Exception: Uno o più valori nulli >
```

Soluzione.

```
class Matrix:
    def __init__(self, rc = (4,4), f = lambda r,c: None):
        self.f = f
        self._rc = rc
        self._mtx = [[f(r,c) for c in range(rc[1])] for r in range(rc[0])]

    def shape(self):
        return self._rc

    @property
    def show(self):
        frt = lambda x: f'{str(x):>6s}' if x != None else f'{"Nan":>6s}'
        out = ""
        for row in self._mtx:
```

```
            out += " ".join([frt(r) for r in row] + ["\n"])
        print(out)

    def __repr__(self):
        return f'Matrix({self._rc})'

    def __str__(self):
        return f'Matrix with {self._rc[0]} row and {self._rc[1]} colums'

    def __setitem__(self,rc, val):
        r,c = rc
        self._mtx[r][c] = val

    def __getitem__(self,rc):
        r,c = rc
        return self._mtx[r][c]

    def ravel(self):
        r = []
        for row in self._mtx:
            r += [v for v in row]
        return r

    def __iter__(self):
        for item in self.ravel():
            if item != None: yield item

    @property
    def iter_row(self):
        for r in self._mtx:
            yield r

    @property
    def iter_col(self):
        for c in range(self._rc[1]):
            yield [self[r,c] for r in range(self._rc[0])]


    def __matmul__(self, other):
        r1,c1 =  self.shape()
        r2, c2 = other.shape()
        if c1 != r2: raise Exception('incompatible dimensions')
        all_val = self.ravel() + other.ravel()
        if None in all_val: raise Exception('uno o più valori sono nulli')
        new = Matrix(rc=(r1,c2))
        r = 0
```

```
    for row in self.iter_row:
        c = 0
        for col in other.iter_col:
            new[r,c] = sum([row[i]*col[i] for i in range(c1)])
            c += 1
        r += 1
    return new
```

Il metodo __init__ riceve una tupla che definisce le dimenioni della matrice (numero di righe e numero di colonne) ed una funzione lambda usata nella list comprehension per definire i valori da assegnare a ciascun elemento della matrice. La funzione lambda di default si limita a generare sempre un valore nullo indipendentemente dell'indice di riga r e di quello di colonna c.

Le due funzioni lambda richieste sono le seguenti:

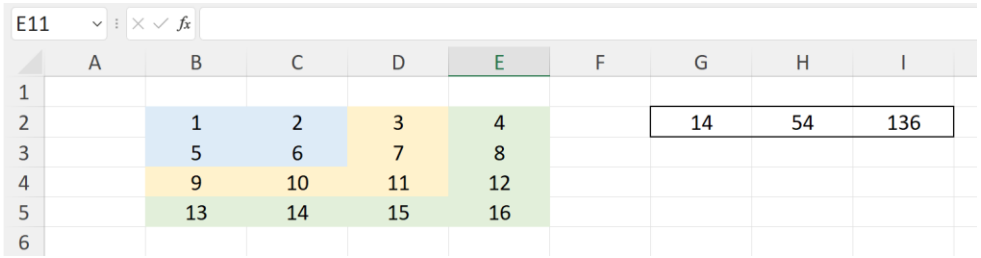
```
f = lambda r,c: c + 1 #valore 1 sulla prima colonna, 2 sulla seconda, ecc.
fi = lambda r,c: 1 if r == c else 0 #matrice identità
```

Si scriva una formula in G2 di modo che vi venga restituita la somma delle celle del range 2x2 evidenziato in azzurro. La soluzione ovvia prevederebbe di utilizzare la formula "Somma()", ma in questo caso è invece richiesto che la funzione in G2 sfrutti opportunamente la formula Scarto(), di modo che:

- copiata in H2 restituisca la somma del range 3x3 evidenziato in azzurro ed in giallo,
- copiata in I2 4 restituisca la somma del range 4 x 4 evidenziato in azzurro, giallo e verde.

Si aggiungano eventuali righe e/o colonne d'appoggio, se ritenuto necessario e si commenti brevemente la logica seguita.

Si ricorda che la funzione scarto ha la seguente sintassi: Scarto(rif; righe; colonne; [alt]; [largh])



Soluzione

La formula è la seguente:

= SOMMA(SCARTO($B$2;0;0;C2;C2))

Il riferimento fisso in $B$2 e l'offset di 0 righe e 0 colonne fa sì che il range (su cui si effettua la somma) parta sempre da tale cella. Il numero di righe e di colonne di cui deve comporsi il range è invece definito tramite il riferimento relativo alla cella C2. In questo modo, copiando al forula verso destra il riferimento diventerà C3 e poi C4 e, quindi le dimensioni del range passeranno da 2x2 a 3x3 e, infine, a 4x4.

Esercizio 2 Excel (4 punti)

Si consideri il range K2:N6 mostrato nella figura seguente. Si scriva una formula in Q3 di modo che vi venga visualizzata la somma di tutte le celle poste sulle righe:

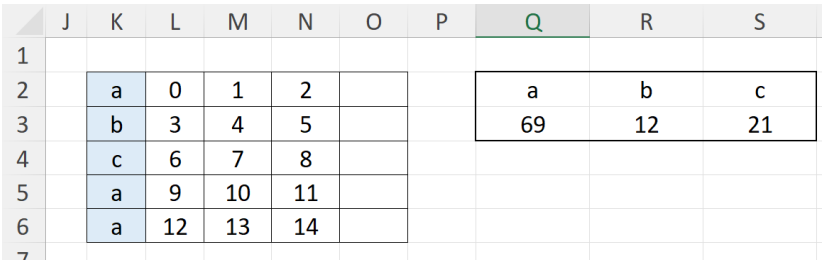
- aventi una somma maggiore di 10
- aventi la stessa etichetta visualizzata in Q2.

Ad esempio, nel caso in figura solo le due ultime righe soddisfano tali criteri e, pertanto, la somma cumulata vale 69.

Si faccia in modo che la formula scritta in Q3 possa essere direttamente copiata in R3 ed in S3, di modo da restituire la somma delle celle su righe con somma maggiore di 10 ed etichetta rispettivamente uguale a 'b' ed a 'c'.

Si commenti brevemente la logica seguita.

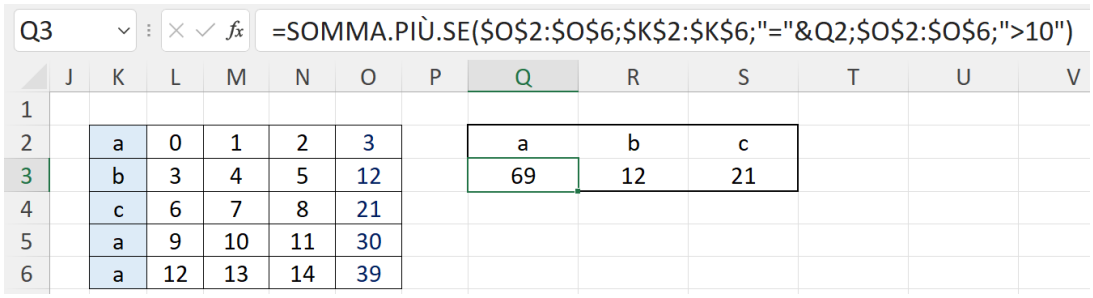
Suggerimento: si usi un'opportuna colonna d'appoggio e si utilizzi la formula Somma.Più.Se(intervallo somma; intervallo criteri, criteri, …).



	J	K	L	M	N	O	P	Q	R	S
1										
2		a	0	1	2			a	b	c
3		b	3	4	5			69	12	21
4		c	6	7	8					
5		a	9	10	11					
6		a	12	13	14					

Soluzione

La formula è mostrata nella figura seguente.



Q3 =SOMMA.PIÙ.SE($O$2:$O$6;$K$2:$K$6;"="&Q2;$O$2:$O$6;">10")

	J	K	L	M	N	O	P	Q	R	S	T	U	V
1													
2		a	0	1	2	3		a	b	c			
3		b	3	4	5	12		69	12	21			
4		c	6	7	8	21							
5		a	9	10	11	30							
6		a	12	13	14	39							

La somma viene fatta sul range $O$2:$O$6, colonna d'appoggio con le somme di ciascuna riga del range di partenza. Le condizioni fatte sulla prima e sull'ultima colonna (la stessa su cui viene effettuata la somma), impongono che l'etichetta sia la stessa lettera scritta in Q2 (cella con riferimenti relativi, di modo da diventare R2 ed S2 copiando la formula verso destra) e che la somma su riga sia maggiore di 10.

Esercizio 1 Python (2 punti)

Usare una *comprehension* per creare una lista contenente tutti i cubi perfetti divisibili per 2, che si possono ottenere con i numeri da 0 a 99 compresi [0, 8, 64, 216, …, 941192].

Soluzione

```
cubes = [c**3 for c in range(100) if (c**3)%2 == 0]
```

Esercizio 2 Python (4 punti)

Si scriva una funzione che riceve un dizionario generico (dct) e restituisce un dizionario con tre chiavi, come indicato di seguito:

- alla chiave *k* è associata la lista delle chiavi di dct,
- alla chiave *v* è associata la lsita dei valori di dct,
- alla chiave *kv* è associata la lista contenente le 2-tuple chiave-valore di dct.

Ad esempio, partendo da dct = {1:10, 2:20, 3:30} si otterrebbe {'k':[1,2,3], 'v':[10,20,30], 'kv':{(1,10), (2,20), (3,30)].

Soluzione

```
def describe_dict(dct):
   return {'k':list(dct.keys()),
           'v': list(dct.values()),
           'kv': list(dct.items())}
```

In alternativa:

```
def describe2(dct):
   return dict(zip(('k','v','kv'),(list(dct.keys()),list(dct.values()),list(dct.items()))))
```

Esercizio 3 (7 punti)

Si scriva la closure *Potenza(n)* che genera funzioni che calcolano la potenza ennesima di un numero.
Ad esempio:
>>> quadrato = Potenza(2)
>>> quadrato(4)
< 16 >
Per ottenere il punteggio massimo la funzione "interna" che calcola la potenza dovrebbe essere scritta in maniera ricorsiva. Si provveda, inoltre a disegnare il diagramma di stack relativo al calcolo di 2^4
Suggerimento: la funzione ricorsiva può essere direttamente definita all'interno della closure. In alternativa può essere definita come funzione a sé stante, per poi essere opportunamente richiamata all'interno della closure.

Soluzione

```
def Potenza(n):
   def inner(val, m = None):
      if m == None: m = n
      if m == 0: return 1
      return val*inner(val, m-1)
   return inner
```

In alternativa:

```
def r_pot(val, n = 2):
   if n == 0: return 1
   return val*r_pot(val, n-1)
def Potenza(n):
   def inner(val):
      return r_pot(val,n)
   return inner
```

>>> quarta = Potenza(4)

```
quarta(2, m = None)                                          = 32
      --->                                                     ↑
          2 x quarta(2, m = 3)                                 = 16
              --->                                             ↑
                  2 x quarta(2, m = 2)                         = 8
                      --->                                     ↑
                          2 x quarta(2, m = 1)                 = 4
                              --->                             ↑
                                  2 x quarta(2, m = 0)         = 2
                                      --->                     ↑
                                          quarta(2, m = 0)     = 1
```

Esercizio 4 Python (10 punti)

Si crei una classe che consente di inserire, giorno per giorno, le seguenti informazioni metereologiche: *(i)* temperatura minima [°C], *(ii)* temperatura massima [°C], *(iii)* pioggia caduta [mm] e *(iv)* velocità del vento [m/s].

Per evitare di gestire date, si scelga se codificare ciascun giorno con una 3-tupla (giorno, mese, anno) o come stringa. Ad esempio, il 1° gennaio 2000 potrebbe essere codificato come (1, 1, 2000) o come '01/01/2000'.

La classe è così definita:

```
class Meteo():
  def __init__(self, good_day = lambda x: x['rain'] == 0):
    self.ss= {} # conterrà dati nella seguente forma {(1,1,2000): {'tmin':-1, 'tmax':12, 'rain': 0, 'wind':0}
    self.gd = good_day
```

L'attributo self.ss è un dizionario annidato che verrà via via riempito con i dati metereologici, come mostrato nel commento al codice. L'attributo self.gd è una funzione di filtraggio che verrà utilizzata da altri metodi.

Si chiede di completare la classe introducendo:

- Il metodo insert() che permette di aggiungere un nuovo giorno (data) con i corrispettivi valori metereologici.
- Il metodo il_meteo() che riceve una data e, se presente, restituisce "sereno" se i mm di pioggia sono pari a zero, "piovoso" in caso contrario.
- Il metodo __iter__() che permette di far restituire uno alla volta i giorni che soddisfano la condizione di filtraggio associata al metodo self.gd; oltre a ciò i dati devono essere elencati in maniera ordinata, da quello meno recente a quello più recente.
- Il metodo stat() che restituisce la percentuale di giorni che soddisfano due condizioni di filtraggio, passate in input. Ad esempio, se si volesse conoscere la percentuale di giorni piovosi nel 2000 la prima funzione di filtraggio restituirà i soli giorni piovosi del 2000, mentre la seconda tutti i giorni del 2000 di modo che, dal rapporto si possa ottenere la percentuale richiesta.
  Si preveda inoltre che di default:
  - la prima condizioni di filtraggio mantiene solo i giorni piovosi,
  - la seconda non effettua alcun filtraggio.

Si mostri, infine, un esempio di utilizzo della classe e si chiami la funzione stat() per ottenere la % di giorni ventosi (vento > 5 m/s) del mese di gennaio.

Soluzione

```
class Meteo():
  def __init__(self, good_day = lambda x: x['rain'] == 0):
    self.ss= {}
    self.gd = good_day

  def __str__(self):
    return f'Data base with {len(self.ss)} days'

  def insert(self, tp_data, values):
    self.ss[tp_data] = dict(zip(('tmin','tmax','rain', 'wind'),values))

  def stat(self,
           flt1 = lambda k,v: v['rain'] > 0,
```

```
            flt2 = lambda k,v: True):
        per = len([(k,v) for k,v in self.ss.items() if flt1(k,v)])/len([(k,v) for k,v in self.ss.items() if flt2(k,v)])
        return f'{per:5.2%}'

    def il_meteo(self,data):
        x = self.ss.get(data,None)
        if x == None: return "missing date"
        if x['rain'] == 0: return "sunny day"
        else: return "rainy day"

    def __iter__(self):
        days = sorted(self.ss.keys(), key = lambda x:x[-1::-1])
        for d in days:
            stat = self.ss[d]
            if self.gd(stat):
                yield d, stat


m = Meteo()

ds = [(i,1,2000) for i in range(1,10)] #alcune date
ds[0] = (1,1,2002)
values = [(19,29,0,6),(19,25,0,0),(14,22,50,0),(19,25,0,0),(19,26,10,0),
      (19,25,0,0),(19,25,0,0),(14,22,50,0),(19,30,0,7)] #i corrispondenti valori

for d,v in zip(ds, values): #inserimento di date e valori
    m.insert(d,v)

m.gd = lambda v: v['rain'] == 0 and v["tmax"] <= 26

for d in m: # si elencano i giorni di pioggia con temperatura massima inferiore a 26
    print(d)

f1 = lambda k,v: v['wind'] > 5 and k[1] == 1
f2 = lambda k,v: k[1] == 1

x = m.stat(f1, f2) #% di giorni ventosi a gennaio
```