SISTEMI INFORMATIVI PER IL MANAGEMENT MODULI 1 E 2

PREAPPELLO 27/05/2022

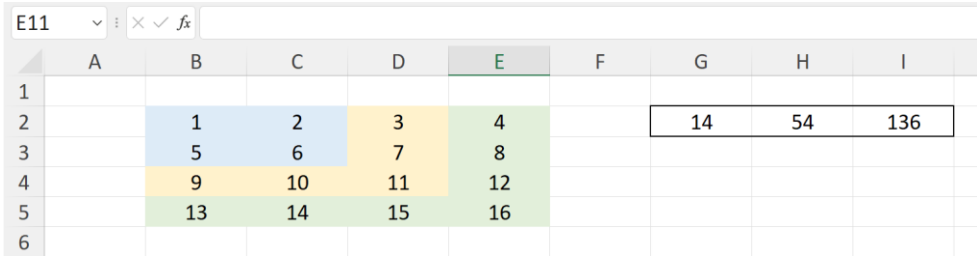
Esercizio 1 Excel (4 punti)

Si consideri il foglio Excel mostrato nella figura seguente. 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'.

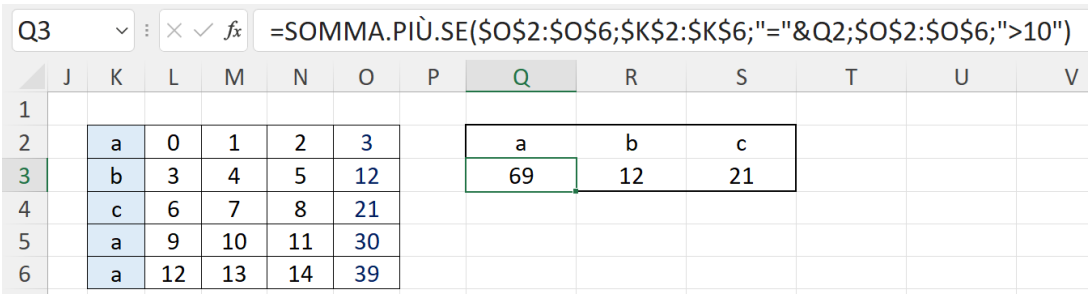
Si commenti brevemente la logica seguita.

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.



	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
```