```
"""
     ESAME SEMINARIO FONDAMENTI ALGORITMI PYTHON   21 Giugno 2018


    - NON CAMBIATE IL NOME DELLE FUNZIONI
        - SE VOLETE POTETE AGGIUNGERE FUNZIONI VOSTRE, MA DOVETE SEMPRE
          RICHIAMARLE DALLE FUNZIONI CHE VI HO SCRITTO IO

    - QUANDO SCRIVO 'RITORNA', LA FUNZIONE DEVE _RITORNARE_ DEI VALORI,
      NON STAMPARLI CON LA PRINT !!!!

    - COMUNQUE, PRINT ADDIZIONALI DI DEBUG SONO CONCESSE

    - NON RIASSEGNATE I PARAMETRI IN INPUT! TRADOTTO: NON SCRIVETE _MAI_
      QUALCOSA DEL GENERE !!!

          def f(x):
             x = .....

    - OGNI FUNZIONE E' SEGUITA DAGLI ASSERT DI TEST, SE QUANDO ESEGUITE
      IL CODICE NON VEDETE ERRORI PROBABILMENTE IL CODICE E' GIUSTO
      (MA NON FIDATEVI TROPPO, I TEST NON SONO ESAUSTIVI !)


    - USARE EDITOR SPYDER.
      - Per eseguire tutto lo script: F5
      - Per eseguire solo la linea corrente o la selezione:  F9

    *** DA FARE: IMPLEMENTATE QUESTE 4 FUNZIONI: ***

    - powers
    - onomat
    - flip
    - gen

"""



"""
ESERCIZIO 1


RITORNA un dizionario in cui le chiavi sono numeri interi da 1 a n inclusi, e i rispettivi
valori sono i quadrati delle chiavi

powers(3)

Ritorna

{
 1:1,
 2:4,
 3:9
}

"""
def powers(n):
    # scrivi qui


# INIZIO TEST - NON TOCCARE !!!
assert powers(1) == {1:1}
assert powers(2) == {
                        1:1,
                        2:4
                    }
assert powers(3) == {
                        1:1,
                        2:4,
                        3:9
                    }

assert powers(4) == {
                        1:1,
                        2:4,
                        3:9,
                        4:16
                    }
# FINE TEST


"""
ESERCIZIO 2


INPUT:
  - frase da arricchire
  - Il sentimento da usare, che è codificato come un valore numerico.
  - un dizionario di sentimenti, in cui si associa al codice numerico
di ogni sentimento un dizionario contenente un espressione onomatopeica tipica per quel sentimento,
```

```
e la posizione in cui deve figurare all'interno di una frase. Le posizioni sono indicate come 'i'
per inizio e 'f' per fine.


OUTPUT

- La frase arricchita con l'espressione onomatopeica scelta in base al sentimento. L'espressione
  va aggiunta sempre prima o dopo la frase, e sempre separata da uno spazio.



Per esempio

sentimenti = {
                1:  {
                        "espressione": "Gulp!",
                        "posizione": "i"
                    }
                2:  {
                        "espressione": "Sgaragulp !",
                        "posizione": "i"
                    }
                3:  {
                        "espressione": "Uff..",
                        "posizione": "f"
                    }
}


onomat("Ma quelli sono i bassotti!", 1, sentimenti)

Deve tornare

"Gulp! Ma quelli sono i bassotti!"

Mentre

onomat("Non voglio alzarmi dall'amaca.", 3, sentimenti)

Deve tornare

"Non voglio alzarmi dall'amaca. Uff.."


NOTA: Ricordarsi lo spazio tra espressione e frase!

"""
def onomat(frase, sentimento, sentimenti):

    # scrivi qui



# INIZIO TEST - NON TOCCARE !!!


sentimenti = {
                1:  {
                        "espressione": "Gulp!",
                        "posizione": "i"
                    },
                2:  {
                        "espressione": "Sgaragulp!",
                        "posizione": "i"
                    },
                3:  {
                        "espressione": "Uff..",
                        "posizione": "f"
                    },
                4:  {
                        "espressione": "Yuk yuk!",
                        "posizione": "f"
                    },
                5:  {
                        "espressione": "Sgrunt!",
                        "posizione": "i"
                    },
                6:  {
                        "espressione": "Gasp!",
                        "posizione" : "i"
                    }
            }


assert onomat("Mi chiamo Pippo.", 4, sentimenti) == "Mi chiamo Pippo. Yuk yuk!"
assert onomat("Quel topastro mi ha rovinato un'altra rapina!", 5, sentimenti) == "Sgrunt! Quel topastro mi
ha rovinato un'altra rapina!"
assert onomat("Non voglio alzarmi dall'amaca.", 3, sentimenti) == "Non voglio alzarmi dall'amaca. Uff.."

# FINE TEST
```

```
"""

Esercizio 3

Prende una matrice come lista di liste in ingresso contenenti zeri e uni, e
RITORNA una nuova matrice (sempre come lista di liste), costruita prima
invertendo tutte le righe della matrice di input e poi rovesciando tutte le righe

- Invertire una lista vuol dire trasformare gli 0 in 1 e gli 1 in 0.
  Per esempio,
    [0,1,1] diventa [1,0,0]
    [0,0,1] diventa [1,1,0]

- Rovesciare una lista vuol dire che rovesciare l'ordine degli elementi:

    Per esempio
      [0,1,1] diventa [1,1,0]
      [0,0,1] diventa [1,0,0]

Combinando inversione e rovesciamento, per esempio se partiamo da


[
  [1,1,0,0],
  [0,1,1,0],
  [0,0,1,0]
]


Prima invertiamo ciascun elemento:

[
  [0,0,1,1],
  [1,0,0,1],
  [1,1,0,1]
]


e poi rovesciamo ciascuna riga:

[
  [1,1,0,0],
  [1,0,0,1],
  [1,0,1,1]

]

Suggerimenti

 - per rovesciare una lista usare il metodo .reverse() come in   mia_lista.reverse()
   NOTA: mia_lista.reverse() modifica mia_lista, *non* ritorna una nuova lista !!
 - ricordarsi ll return !!
"""
def flip(matrice):

    # scrivi qui


# INIZIO TEST - NON TOCCARE !!!

assert flip([[]]) == [[]]
assert flip([[1]]) == [[0]]
assert flip([[1,0]]) == [[1,0]]

m1 = [
      [1,0,0],
      [1,0,1]
     ]
mat_attesa1 = [
    [1,1,0],
    [0,1,0]
]
assert flip(m1) == mat_attesa1


m2 = [
  [1,1,0,0],
  [0,1,1,0],
  [0,0,1,0]
]

mat_attesa2 = [
  [1,1,0,0],
  [1,0,0,1],
  [1,0,1,1]

]

assert flip(m2) == mat_attesa2

# verifica che l'm originale non è cambiato !
```

```
assert m2 == [
  [1,1,0,0],
  [0,1,1,0],
  [0,0,1,0]
]

# FINE TEST

"""

ESERCIZIO 4

Ritorna un generatore che genera una sequenza contenente n stringhe, con queste
regole:

- si suppone che il primo elemento della sequenza sia ad indice zero

se si sta generando una stringa:
- ad indice pari, la stringa deve contenere la lettera 'a'
- ad indice dispari, la stringa deve contenere la lettera 'b'
- ad indice divisibile per 3, la stringa deve contenere la lettera 'c'.
  Tale lettera deve essere posta alla fine della stringa

Per esempio:

list(gen(7))

deve dare

['ac','b','a','bc','a','b','ac']

Perchè

Indice    Stringa   Perchè
  0        'ac'     0 è pari e divisibile per 3
  1        'b'      1 è dispari
  2        'a'      2 è pari
  3        'bc'     3 è dispari e divisibile per tre
  4        'a'      4 è pari
  5        'b'      5 è dispari
  6        'ac'     6 è pari e divisibile per tre

Suggerimenti
- se volete usare i moduli, per vedere se un indice i è divisibile per m scrivete i % m == 0
- se avete letto la documentazione dei generatori, in alternativa ai moduli c'è
  un'altro modo più furbo che potete usare
"""

def gen(n):

    # scrivi qui



# INIZIO TEST - NON TOCCARE !!!


import types
assert isinstance(gen(0), types.GeneratorType) # verifico che pari ritorni un generatore e non una lista

assert list(gen(1)) == ['ac']
assert list(gen(2)) == ['ac','b']
assert list(gen(3)) == ['ac','b','a']
assert list(gen(4)) == ['ac','b','a','bc']
assert list(gen(5)) == ['ac','b','a','bc','a']
assert list(gen(6)) == ['ac','b','a','bc','a','b']
assert list(gen(7)) == ['ac','b','a','bc','a','b','ac']

# FINE TEST
```