

# Soluzioni degli esercizi

---

Queste soluzioni sono proposte soprattutto per favorire un'acquisizione progressiva delle conoscenze. Bisogna partire dall'assunto che esse *non* siano le uniche o le migliori. Prima di studiarle, ognuno deve cercare in autonomia le *proprie*, che potranno anche essere molto diverse da quelle proposte. Alcune delle soluzioni seguenti potrebbero essere incomplete e presentare solo alcune idee per risolvere gli aspetti più critici del problema.

## Esercizi capitolo 8 - Sequenze di dati

### Istogramma con barre orizzontali

```
W, H = 600, 400

values = []
txt = g2d.prompt("Val? ")
while txt:
    values.append(float(txt))
    txt = g2d.prompt("Val? ")

n, max_val = len(values), max(values)
for i, v in enumerate(values):
    pos = 0, i * H / n
    size = v * W / max_val, (H / n) - 1
    g2d.draw_rect(pos, size)
```

 [https://fondinfo.github.io/play/?exs/c08\\_histogram.py](https://fondinfo.github.io/play/?exs/c08_histogram.py)

Tutte le barre hanno lo stesso valore di  $x$  e  $h$ .

Il valore di  $y$  è proporzionale a  $i$ .

Il valore di  $w$  è proporzionale a  $v$ .

### Risultati casuali

```
rolls = int(input("Rolls? "))
results = [0] * 11

for r in range(rolls):
    die1 = randint(1, 6)
```

```
die2 = randint(1, 6)
val = die1 + die2
results[val - 2] += 1

for i, v in enumerate(results):
    #print(i + 2, v)
    print(f"{i + 2:2}", "=" * v) # f-string, string repetition
```

▶ [https://fondinfo.github.io/play/?exs/c08\\_dice.py](https://fondinfo.github.io/play/?exs/c08_dice.py)

I risultati possibili sono nell'intervallo [2, 12], usiamo quindi una lista di 11 contatori. Contatore `results[0]`: quante volte esce il 2. Contatore `results[1]`: quante volte esce il 3, ecc.

### Riempimento

```
def fill(values: list[int], pos: int):
    if values[pos] == 0:
        values[pos] = 1
        for d in (-1, 1):
            pos1 = pos + d
            while 0 <= pos1 < len(values) and values[pos1] == 0:
                values[pos1] = 1
                pos1 += d

data = [int(v) for v in "0022000000002000"]
fill(data, 9)
print(data)
```

▶ [https://fondinfo.github.io/play/?exs/c08\\_fill.py](https://fondinfo.github.io/play/?exs/c08_fill.py)

### Merge

```
def merge(a: list, b: list) -> list:
    result = []
    while a or b:
        if a and b:
            d = a if a[0] <= b[0] else b
        else:
            d = a if a else b
        result.append(d.pop(0))
    return result

data1 = [1, 4, 5, 7]
data2 = [2, 3, 6, 8, 9, 10]
merged = merge(data1, data2)
print(data1, data2, merged)
```

▶ [https://fondinfo.github.io/play/?exs/c08\\_merge.py](https://fondinfo.github.io/play/?exs/c08_merge.py)

Non è necessario fare un ordinamento completo, è sufficiente confrontare i primi valori delle due liste. Tra i due, viene estratto quello più piccolo. Sia `data1` che `data2` alla fine restano vuote.

## Clamp di lista

```
def clamp(values: list[int], a: int, b: int):
    for i, v in enumerate(values):
        if v < a:
            values[i] = a
        if v > b:
            values[i] = b

data = [3, 4, 6, 7, 3, 5, 6, 12, 4]
clamp(data, 5, 10)
print(data)
```

 [https://fondinfo.github.io/play/?exs/c08\\_clamp.py](https://fondinfo.github.io/play/?exs/c08_clamp.py)

## Shuffle

```
from random import randrange

def shuffle(values: list):
    for i, v in enumerate(values):
        j = randrange(len(values))
        values[i], values[j] = values[j], values[i]

data = list(range(10))
shuffle(data)
print(data)
```

 [https://fondinfo.github.io/play/?exs/c08\\_shuffle.py](https://fondinfo.github.io/play/?exs/c08_shuffle.py)