Tempo per la soluzione 1 ora

risposta corretta = 4 punti, risposta sbagliata = -1 punti, nessuna risposta = 0 punti

Risposte:

1	2	3	4	5	6	7	8

1) Calcola l'output

```
def process():
  base = [i for i in range(7)]        # [0,1,2,3,4,5,6]
  a = base[2:6]                  # copia
  b = a                        # alias
  c = base                     # alias
  a.append(99)                      # muta a e b
  c[3] = -3                    # muta base
  for k in range(2):                # prepend due volte
    b.insert(0, -k)
  d = c[1:5]                    # copia
  d[1] = 77                     # NON muta base
  c[4] = 44                     # muta base
  e = d[:]                     # copia
  if len(e) == len(d): e.pop()         # accorcia e
  if e and e[-1] == 77: a.append(8)      # controllo semplice
  return base, a, b, c, d, e

B,A,BB,C,D,E = process()
print(B); print(A); print(BB); print(C); print(D); print(E)
```

A.
[0, 1, 2, -3, 44, 5, 6]
[0, -1, 2, 3, 4, 99, 8]
[0, -1, 2, 3, 4, 99, 8]
[0, 1, 2, -3, 44, 5, 6]
[1, 77, -3, 44]
[1, 77, -3]
B.
[0, 1, 2, -3, 44, 5, 6]
[-1, 0, 2, 3, 4, 99, 8]
[-1, 0, 2, 3, 4, 99, 8]
[0, 1, 2, -3, 44, 5, 6]
[1, 77, -3, 44]
[1, -3, 44]
C.
[0, 1, 2, -3, 44, 5, 6]
[-1, 0, 2, 3, 4, 5, 99]
[-1, 0, 2, 3, 4, 5, 99]
[0, 1, 2, -3, 44, 5, 6]
[1, 77, -3, 4]
[1, 77, -3]
D. Errore di runtime

2): Trova l'errore

Obiettivo del programma: Per ogni chiave in keys, se esiste in cfg aggiunge 0 alla lista; altrimenti crea una nuova lista con [1]. Alla fine stampa il dizionario aggiornato.

```
cfg = {"u": ["ann","bob"], "meta": {"ok": True}, "ids": [1]}
keys = ["u","ids","x","u","x","y"]
for k in keys:
  if k in cfg and isinstance(cfg[k], list):
```

```
    cfg[k].append(0)
  else:
    cfg[k].append(1)    # <-- UNA SOLA RIGA è sbagliata, correggila
pairs = []
for k in ["u","ids","x","y"]:
  if k in cfg:
    pairs.append((k, len(cfg[k]) if isinstance(cfg[k], list) else -1))
print(cfg); print(pairs)
```

A. Sostituire con cfg[k] = [1]
B. Sostituire con cfg[k] = 1
C. Sostituire con cfg[k].insert(0,1)
D. Sostituire con cfg.setdefault(k,[1]) *(non usare: non è consentito)*

3) Calcola l'output

```
def tally(words):
  cleaned = []
  for w in words:
    w2 = w.strip()
    if w2: cleaned.append(w2)
  hist = {}
  for w in cleaned:
    k = len(w)
    if k not in hist: hist[k] = 0
    hist[k] = hist[k] + 1
  seq = []
  for k in hist: seq.append((k, hist[k]))
  # riordino "manuale" crescente per chiave senza sort avanzati
  ordered = []
  used = set()
  while len(ordered) < len(seq):
    m = None
    for (k,_) in seq:
      if k in used: continue
      if m is None or k < m: m = k
    ordered.append((m, hist[m])); used.add(m)
  return ordered

print(tally(["aa","b","", "ccc","dd","x"]))
```

A. [(1,2),(2,2),(3,1)]
B. [(1,1),(2,2),(3,1)]
C. [(1,2),(2,1),(3,2)]
D. [(1,1),(2,3),(3,1)]

4) Calcola l'output

```
def f(n):
  if n <= 1: return n
  if n % 2 == 0: return f(n//2)
  return 1 + f(n-1)

vals = [15, 10, 5, 8]
out = []
s = 0
for v in vals:
  r = f(v)
  out.append((v, r))
  s += r
```

```
print(out); print(s)
```

A. [(15, 4), (10, 2), (5, 3), (8, 1)] 10
B. [(15, 3), (10, 2), (5, 2), (8,1)] 8
C. [(15, 4), (10, 1), (5, 3), (8,1)] 9
D. [(15, 4), (10, 2), (5, 2), (8, 1)] 9

5) Calcola l'output

```
def transform(seq):
  out = []
  for i in range(len(seq)):
    s = 0
    for j in range(i+1):       # 0..i
      s += seq[j] * (i - j + 1) # peso crescente a sinistra
    out.append(s)
  return out

a = [3, 1, 2]
b = [x-1 for x in a]            # singola comprehension ok
t1 = transform(a)
t2 = transform(b)
diff = [t1[i]-t2[i] for i in range(len(t1))]
print(t1); print(t2); print(diff)
```

A. [3, 7, 14] [2, 4, 9] [1, 3, 5]
B. [3, 7, 13] [2, 4, 7] [1, 3, 6]
C. [3, 6, 12] [2, 3, 9] [1, 3, 3]
D. [3, 7, 14] [2, 4, 10] [1, 3, 4]

6 Trova l'errore / correggi UNA riga
Obiettivo del programma: Leggere index.txt (un path per riga), filtrare solo i file .txt esistenti e concatenarli in merged.txt.

```
import os

def merge_from_index(index_path, out_path):
  with open(index_path, "r") as idx:
    paths = []
    for line in idx:
      p = line.strip()
      if p: paths.append(p)
  with open(out_path, "w") as out:
    for p in paths:
      if os.path.isfile(p) and p.endswith(".txt"):
        with open(p, "r") as src:
          for row in src:
            out.write(row)

merge_from_index("index.txt", "merged.txt")
print("OK")
```

A. Sostituire con os.path.isfile(p)
B. Aprire out_path con "a"
C. Aggiungere src.close()
D. Rimuovere and p.endswith(".txt")

7) Calcola l'output

```
def deco(f):
  def w(x):
    return f(x) + f(x+1)
  return w
```

```
def add(k):
  return lambda x: x + k

def mul(k):
  return lambda x: x * k

@deco
def shift(x):
  return x + 2

pipeline = [add(1), mul(3), shift]   # funzioni come oggetti
def run(pipes, v):
  r = v
  for fn in pipes:
    r = fn(r)
  return r

print(run(pipeline, 1), run([mul(2), shift], 2))
```

A. 15 8
B. 14 8
C. 17 13
D. 12 8

8) Calcola l'output

```
class A:
  def __init__(self, v):
    self.v = v
  def f(self, x):
    return self.v + x

class B(A):
  def __init__(self, v, w):
    super().__init__(v)
    self.w = w
  def f(self, x):
    return super().f(x) + self.w

class C(B):
  def __init__(self, v, w, name):
    super().__init__(v, w)
    self.name = name
  def f(self, x):
    return self.name + ":" + str(super().f(x))

objs = [A(2), B(1,3), C(2,2,"Z")]
res = []
for o in objs:
  res.append(o.f(2))
print(res)
```

A. [4,6,'Z:6']
B. [4,5,'Z:5']
C. [4,6,'Z:4']
D. [2,4,'Z:2']