Cognome e Nome _____

Matricola _____

Programmazione e Algoritmi 2
A.A. 2025/2026

Appello del 15 gennaio 2026

COMPITO N° 1

PRIMA PARTE

Esercizio 1 [10 punti]

Si consideri il seguente codice Python.

```
class A:
    @property
    def valore (self) -> int:
        return 10

class B:
    def __eq__(self, o: object) -> bool:
        print ("eq di B")
        if isinstance(o, B): return True
        if isinstance(o, A): return True
        return NotImplemented

class C (B):
    pass
```

```
class D (A):
    @property
    def valore (self) -> int:
        return 20

class E (C, D):
    pass

c = C()
d = D()
e = E()

print (d==c)
print (e.valore)
```

Denotando la classe `object` con **O**, si calcoli la linearizzazione di tutte le classi, e si scriva in tabella il procedimento e il risultato finale:

classe	linearizzazione (MRO)
A	L(A) = A + merge (L(O), O) = A + merge (O, O) = AO
B	
C	
D	
E	

Si scriva nel riquadro a destra cosa **esclusivamente** ciò che viene stampato a video dal codice.

Nel caso in cui il codice dia errore in qualche punto (poiché ad esempio non è possibile calcolare qualche MRO), escludere dal codice il numero minimo di linee in modo da non ottenere errori, commentandole con **#** sul listato stampato in alto su questo foglio, e quindi stampare cosa risulta dal nuovo codice così ottenuto.

Esercizio 2 [12 punti]

Si consideri il seguente array di numeri interi:

[10, 2, 6, 15, 29, 3, 27, 13, 31]

Mostrare **passo-passo** l'esecuzione dell'algoritmo Heap-Sort**, inclusa la fase di build-max-heap**.

Esercizio 3 [12 punti]

Si implementino in Python le classi **Premio** e **PremioConContributo** per gestire le raccolte punti con premi organizzate da una catena commerciale.

Attenzione: Tutto il codice deve essere annotato con i tipi opportuni.

[6 punti] Si scriva la *classe* **Premio** con i seguenti attributi di istanza:

- **_codice** (tipo **int, Final**)
- **_descrizione** (tipo **str**)
- **_punti** (tipo **int**, rappresenta il numero di punti necessari per ottenere il premio)

ed i seguenti metodi di istanza:

- *__init__* che inizializza un oggetto della *classe* **Premio** assegnando codice, descrizione e punti
- proprietà (`@property`) in lettura per tutti gli attributi, con lo stesso nome (senza _);
- proprietà in scrittura per descrizione e punti.
- metodo **__eq__ (self, o:object)** che restituisce **True** se **o** è un Premio con lo stesso codice di self, (indipendentemente dalla descrizione e dai punti), **False** altrimenti.
- metodo **__str__ (self)** che restituisce la rappresentazione come stringa di un Premio, nella forma `Premio` **`x`** `con codice` **`y`**`: punti` **`z`**
  dove x, y e z rappresentano la descrizione, il codice e i punti del premio **self**, rispettivamente.

[3 punti] Si scriva la *classe* **PremioConContributo** che estende Premio ed ha, inoltre, il seguente attributo:

- **_prezzo** (tipo **int**, esprime in centesimi di euro il contributo da pagare per ottenere il premio)

La *classe* **PremioConContributo** ha i seguenti metodi di istanza:

- *__init__* che inizializza un oggetto della *classe* **PremioConContributo** prendendo in input gli opportuni parametri (incluso il prezzo) e richiamando opportunamente il metodo __init__ della superclasse
- proprietà (`@property`) **prezzo** in lettura e scrittura per l'attributo **_prezzo**.

[3 punti] Scrivere (fuori dalle classi) un metodo

filtra_premi (premi:list[Premio], budget_prezzo:int) -> list[PremioConContributo]

che restituisce una nuova lista contenente tutti e soli i premi, contenenti nella lista **premi** ricevuta come parametro, che sono premi con contributo aventi un prezzo da corrispondere al più uguale a **budget_prezzo**.

SECONDA PARTE

Esercizio 4 [12 punti]

Per chi svolge solo la seconda parte: tenere presente, senza svolgerlo, la traccia dell'esercizio 3.
Attenzione: Tutto il codice deve essere annotato con i tipi opportuni.
Si assuma, senza scrivere nulla, l'esistenza di una classe **Cliente** con i seguenti attributi di istanza:

- **_numero_tessera** (tipo **int, Final**)
- **_nome** (tipo **str**)
- **_cognome** (tipo **str**)

ed i seguenti metodi di istanza:

- *__init__* che inizializza un oggetto della *classe* **Cliente** assegnando numero_tessera, nome e cognome.
- proprietà (`@property`) in lettura per tutti gli attributi, con lo stesso nome (senza _);
- metodo **__eq__ (self, o:object)** che restituisce **True** se **o** è un Cliente con lo stesso numero_tessera di self, (indipendentemente dagli altri attributi), **False** altrimenti.

Si assuma, senza scrivere nulla, l'esistenza di una classe **Richiesta**, che rappresenta la richiesta di un premio da parte di un cliente, con i seguenti attributi di istanza:

- **_cliente** (di tipo **Cliente**)
- **_premio** (di tipo **Premio**)

ed i seguenti metodi di istanza:

- *__init__* con tutti i parametri necessari, che inizializza un oggetto della *classe* **Richiesta** assegnando **_cliente** e **_premio**
- Proprietà in lettura (senza _) per ciascun attributo

Nota: Se necessario, si aggiungano alle classi dell'**Esercizio 3** e/o a quelle sopra descritte tutti i metodi necessari per far funzionare correttamente il codice richiesto, e si discuta quanto fatto.

[6 punti] Si scriva la *classe* **GestionePremi** con i seguenti attributi di istanza:

- **_clienti** (**list** di **Cliente**)
- **_premi** (**list** di **Premio**)
- **_saldi_punti** (**dict** con chiave **Cliente** e valori **int** che rappresentano il saldo del punteggio)
- **_richieste_accettate** (**list** di Richieste)

ed i seguenti metodi di istanza:

- *__init__* che inizializza un oggetto della *classe* **GestionePremi** creando liste vuote per _clienti, _premi e _richieste_accettate e in dizionario vuoto per _saldi_punti.
- metodo **add_cliente (self, c:Cliente, punteggio_iniziale:int=0)** che aggiunge il cliente c alla lista _clienti, se non è già presente in essa, assegnandogli il punteggio iniziale ricevuto come parametro.
- metodo **add_premio (self, p:Premio)** che aggiunge il premio p alla lista _premi, se non è già presente in essa

[3 punti] Aggiungere alla classe **GestionePremi** un metodo di istanza
processa_richieste (self, richieste:list[Richiesta]) -> None
che aggiunge alla lista self._richieste_accettate tutte e soli le richieste "accettabili", per le quali il cliente richiedente ha i punti necessari per il premio richiesto (il metodo deve anche sottrarre, per ogni richiesta accettata, il corrispondente quantitativo di punti dal saldo punti del cliente richiedente). Se la lista self._clienti è lunga **nc**, la lista self._premi è lunga **np** e la lista richieste è lunga **nr**, la complessità nel caso medio (con l'uso di dizionari) deve essere *O(**nc** + **np** + **nr**).*

[3 punti] Aggiungere alla classe **GestionePremi** una proprietà in lettura
premio_più_richiesto (self) -> Premio|None
che restituisce uno dei premi presenti con il maggior numero di occorrenze nella lista self._richieste_accettate (None se tale lista è vuota). Se la lista self._premi è lunga **np** e la lista self._richieste_accettate è lunga **nr**, la complessità nel caso medio (con l'uso di dizionari) deve essere *O(**np** + **nr**).*
Suggerimento: si usi un dizionario per memorizzare il numero di richieste accettate per ogni premio.

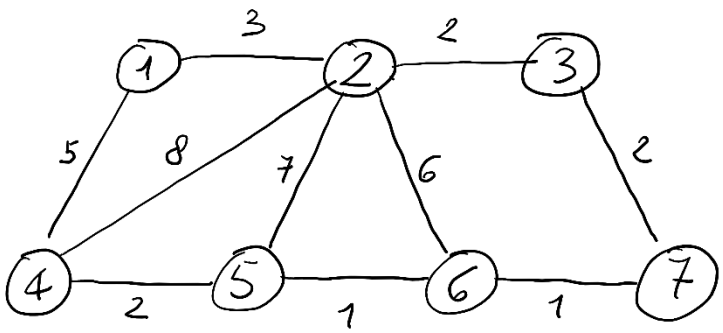
Esercizio 5 [10 punti]

Si consideri la sequenza di chiavi **[7, 19, 18, 20, 25, 5, 23, 30]**

1. **[5 punti]** Mostrare passo-passo l'evoluzione della tabella Hash di dimensione 11 (con hashing doppio), che si ottiene **inserendo** le chiavi nell'ordine indicato.
2. **[5 punti]** A partire dalla tabella ottenuta al punto precedente, mostrare passo-passo l'evoluzione a seguito delle seguenti operazioni, da eseguirsi in sequenza:
    a. **cancellazione** di **7** e **23**;
    b. **ricerca** di **5**, **11** e **18** (mostrare tutti gli accessi effettuati in tabella e l'esito della ricerca)
    c. **inserimento** di **40** e **16**.

Esercizio 6 [12 punti]

Si consideri il grafo diretto in figura.



Mostrare una possibile esecuzione **passo-passo** dell'algoritmo di **Dijkstra** per i cammini minimi da singola sorgente, a partire dal nodo sorgente **2**. Ad ogni passo, bisogna mostrare il contenuto della coda con priorità utilizzata dall'algoritmo e del vettore dei padri.

Che peso ha il cammino minimo dal nodo 2 al nodo 5?

Regole per lo svolgimento della prova scritta:

- Per svolgere il compito si hanno a disposizione **100** minuti per chi svolge solo la seconda parte e **180** minuti per chi svolge il compito totale (chi non consegna dopo 100 minuti automaticamente rinuncia al voto dell'esonero).
- Scrivere **subito** nome, cognome, matricola su OGNI FOGLIO (**compresi quelli della traccia, negli appositi spazi**).
- Durante la prova scritta **non** è possibile abbandonare l'aula.
- Non è ammesso **per nessun motivo** comunicare in qualsiasi modo con altre persone
- È possibile consultare appunti, libri e dispense.
- Qualsiasi strumento elettronico di calcolo o comunicazione (telefoni cellulari, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista lontano dalla vostra postazione.**
- Mettere in vista sul banco un valido documento di identità.

Cognome e Nome _____

Matricola _____

COMPITO N° 1