Cognome e Nome _____

Matricola _____

Programmazione e Algoritmi 2 A.A. 2026/2027

Esonero del 13 novembre 2026

Compito n° 1

Esercizio 1 [10 punti]

Si consideri il seguente codice Python.

```
class X:
    def __eq__(self, o: object) -> bool:
        print ("EQ di X")
        return NotImplemented

class A(X):
    def __eq__(self, o: object) -> bool:
        print ("EQ di A")
        if isinstance(o, A) or \
            isinstance (o, X):
            return True
        return NotImplemented

class B:
    def __eq__(self, o: object) -> bool:
        print ("EQ di B")
        if isinstance(o, B) or \
        isinstance (o, X):
            return True
        return NotImplemented
```

```
class C (B, X, A):
    pass

class D (A, B):
    pass


x = X()
a = A()
b = B()
c = C()
d = D()

print (x==a)
print (x==b)
print (x==c)
```

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)
X	L(X) = X + merge (L(O), O) = X + merge (O, O) = XO
A	
B	
C	
D	

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 la seguente sequenza/array di numeri interi:

[158, 11, 236, 72, 199, 35, 267, 121, 58, 180]

1. **[5 punti]** Mostrare **passo-passo l'evoluzione del max-heap** che si ottiene, a partire dall'heap vuoto, inserendo le chiavi nell'ordine indicato.
2. **[7 punti]** Mostrare passo-passo l'esecuzione dell'Heap-sort, inclusa la procedura **build-max-heap,** sull'array indicato.

Esercizio 3 [12 punti]

Attenzione: Tutto il codice deve essere annotato con i tipi opportuni. Si implementino in Python le classi **Pizza** e **PizzaCornicione**. Le parti evidenziate **non** devono essere svolte in questo compito.

[4 punti] La *classe* **Pizza** ha i seguenti attributi di istanza:

- **_nome** (tipo **str, Final**)
- **_ingredienti** (tipo **list[str]**), che contiene gli ingredienti sulla pizza

ed i seguenti metodi di istanza:

- *__init__* che inizializza un oggetto della *classe* **Pizza** assegnando nome e ingredienti (prende come parametri una stringa per il nome e una lista di stringhe per gli ingredienti).
- proprietà (`@property`) in lettura per tutti gli attributi, con lo stesso nome (senza _);
- proprietà (`@property`) **rossa** che restituisce **True** se "pomodoro" è tra gli ingredienti, **False** altrimenti;
- metodo **contiene (self, ingrediente:str) -> bool** che restituisce **True** se e solo se **ingrediente** è tra gli ingredienti sulla pizza.
- metodo **__eq__ (self, o:object)** che restituisce **True** se **o** è una Pizza con lo stesso nome (senza tener conto degli ingredienti), e gestisce i rimanenti casi restituendo in modo opportuno **False** e/o **NotImplemented**.

[4 punti] La *classe* **PizzaCornicione** estende Pizza ed ha, inoltre, il seguente attributo:

- **_ingredienti_cornicione** (tipo **list[str]**), che contiene gli ingredienti nel cornicione

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

- *__init__* che inizializza un oggetto della *classe* **PizzaCornicione** prendendo in input gli opportuni parametri (inclusa la lista di stringhe per gli ingredienti del cornicione) e richiamando opportunamente il metodo __init__ della superclasse;
- proprietà (`@property`) **ingredienti_cornicione** in sola lettura per l'attributo **_ ingredienti_cornicione**;
- metodo **contiene (self, ingrediente:str) -> bool** che sovrascrive il metodo della superclasse e restituisce **True** se e solo se **ingrediente** è tra gli ingredienti sulla pizza e/o nel cornicione.

Si assuma, senza scrivere nulla, l'esistenza di una classe **Menù** con

- una variabile di istanza **_pizze** (tipo **list[Pizza]**), contenente tutte le pizze del menù di una pizzeria.

[4 punti] Aggiungere alla classe **Menù** i seguenti metodi:

- **lista_tutti_ingredienti (self) -> list[str]**
  che restituisce una lista contenente, senza ripetizioni, tutti gli ingredienti presenti in almeno una pizza della lista **self._pizze**.
- **estrai_pizze_con_cornincione (self, ingredienti_richiesti:list[str]) -> list[Pizza]**
  che restituisce una lista contenente tutte le pizze con cornicione presenti nella lista **self._pizze** che contengano (sulla pizza o nel cornicione) tutti gli ingredienti della lista **ingredienti_richiesti**.

Regole per lo svolgimento della prova scritta:

- Per svolgere il compito si hanno a disposizione **100** minuti.
- Scrivere **subito** nome, cognome, matricola su OGNI FOGLIO (**compreso questo**).
- 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 (smartphone, smart*, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista sulla cattedra**
- Mettere in vista sul banco un valido documento di identità.

Cognome e Nome _____

Matricola _____

Programmazione e Algoritmi 2 A.A. 2026/2027

Esonero del 13 novembre 2026

Compito n° 2

Esercizio 1 [10 punti]

Si consideri il seguente codice Python.

```
class X:
    def __eq__(self, o: object) -> bool:
        print ("EQ di X")
        return NotImplemented

class A:
    def __eq__(self, o: object) -> bool:
        print ("EQ di A")
        if isinstance(o, A) or \
            isinstance (o, X):
            return True
        return NotImplemented

class B(X):
    def __eq__(self, o: object) -> bool:
        print ("EQ di B")
        if isinstance(o, B) or \
        isinstance (o, X):
            return True
        return NotImplemented
```

```
class C (A, X, B):
    pass

class D (B, A):
    pass


x = X()
a = A()
b = B()
c = C()
d = D()

print (x==a)
print (x==b)
print (x==c)
```

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)
X	L(X) = X + merge (L(O), O) = X + merge (O, O) = XO
A	
B	
C	
D	

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 la seguente sequenza/array di numeri interi:

[143, 19, 258, 67, 204, 41, 279, 113, 56, 187]

1. **[5 punti]** Mostrare **passo-passo l'evoluzione del max-heap** che si ottiene, a partire dall'heap vuoto, inserendo le chiavi nell'ordine indicato.
2. **[7 punti]** Mostrare passo-passo l'esecuzione dell'Heap-sort, inclusa la procedura **build-max-heap,** sull'array indicato.

Esercizio 3 [12 punti]

Attenzione: Tutto il codice deve essere annotato con i tipi opportuni. Si implementino in Python le classi **Pizza** e **PizzaCornicione**. Le parti evidenziate **non** devono essere svolte in questo compito.

[4 punti] La *classe* **Pizza** ha i seguenti attributi di istanza:

- **_nome** (tipo **str, Final**)
- **_ingredienti** (tipo **list[str]**), che contiene gli ingredienti sulla pizza

ed i seguenti metodi di istanza:

- *__init__* che inizializza un oggetto della *classe* **Pizza** assegnando nome e ingredienti (prende come parametri una stringa per il nome e una lista di stringhe per gli ingredienti).
- proprietà (`@property`) in lettura per tutti gli attributi, con lo stesso nome (senza _);
- proprietà (`@property`) **rossa** che restituisce **True** se "pomodoro" è tra gli ingredienti, **False** altrimenti;
- metodo **contiene (self, ingrediente:str) -> bool** che restituisce **True** se e solo se **ingrediente** è tra gli ingredienti sulla pizza.
- metodo **__eq__ (self, o:object)** che restituisce **True** se **o** è una Pizza con lo stesso nome (senza tener conto degli ingredienti), e gestisce i rimanenti casi restituendo in modo opportuno **False** e/o **NotImplemented**.

[4 punti] La *classe* **PizzaCornicione** estende Pizza ed ha, inoltre, il seguente attributo:

- **_ingredienti_cornicione** (tipo **list[str]**), che contiene gli ingredienti nel cornicione

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

- *__init__* che inizializza un oggetto della *classe* **PizzaCornicione** prendendo in input gli opportuni parametri (inclusa la lista di stringhe per gli ingredienti del cornicione) e richiamando opportunamente il metodo __init__ della superclasse;
- proprietà (`@property`) **ingredienti_cornicione** in sola lettura per l'attributo **_ ingredienti_cornicione**;
- metodo **contiene (self, ingrediente:str) -> bool** che sovrascrive il metodo della superclasse e restituisce **True** se e solo se **ingrediente** è tra gli ingredienti sulla pizza e/o nel cornicione.

Si assuma, senza scrivere nulla, l'esistenza di una classe **Menù** con

- una variabile di istanza **_pizze** (tipo **list[Pizza]**), contenente tutte le pizze del menù di una pizzeria.

[4 punti] Aggiungere alla classe **Menù** i seguenti metodi:

- **lista_tutti_ingredienti (self) -> list[str]**
  che restituisce una lista contenente, senza ripetizioni, tutti gli ingredienti presenti in almeno una pizza della lista **self._pizze**.
- **estrai_pizze_con_cornincione (self, ingredienti_richiesti:list[str]) -> list[Pizza]**
  che restituisce una lista contenente tutte le pizze con cornicione presenti nella lista **self._pizze** che contengano (sulla pizza o nel cornicione) tutti gli ingredienti

Regole per lo svolgimento della prova scritta:

- Per svolgere il compito si hanno a disposizione **100** minuti.
- Scrivere **subito** nome, cognome, matricola su OGNI FOGLIO (**compreso questo**).
- 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 (smartphone, smart*, calcolatrici, palmari, computer, etc…) deve essere **completamente disattivato** e **depositato in vista sulla cattedra**
- Mettere in vista sul banco un valido documento di identità.