

1) Implementare la classe **CircularPositionalList** che implementa una sequenza la cui rappresentazione interna è basata su una Positional List circolare. La classe deve supportare i seguenti metodi:

first()	restituisce la Position dell'elemento che è identificato come il primo oppure None se la lista è vuota
last()	restituisce la Position dell'elemento che è identificato come l'ultimo oppure None se la lista è vuota
before(p)	restituisce l'elemento nella Position precedente a p, None se p non ha un predecessore e ValueError se p non è una position della lista
after(p)	restituisce l'elemento nella Position successiva a p, None se p non ha un successore e ValueError se p non è una position della lista
is_empty()	restituisce True se la lista è vuota e False altrimenti
is_sorted()	restituisce True se la lista è ordinata e False altrimenti
add_first(e)	Inserisce l'elemento e in testa alla lista e restituisce la Position del nuovo elemento
add_last(e)	Inserisce l'elemento e in coda alla lista e restituisce la Position del nuovo elemento
add_before(p, e)	Inserisce un nuovo elemento e prima del nodo nella Position p e restituisce la Position del nuovo elemento
add_after(p, e)	Inserisce un nuovo elemento e dopo il nodo nella Position p e restituisce la Position del nuovo elemento
find(e)	Restituisce una Position contenente la prima occorrenza dell'elemento e nella lista o None se e non è presente
replace(p, e)	Sostituisce l'elemento in Position p con e restituisce il vecchio elemento
delete(p)	Rimuove e restituisce l'elemento in Position p dalla lista e invalida p
clear()	Rimuove tutti gli elementi della lista invalidando le corrispondenti Position
count(e)	Restituisce il numero di occorrenze di e nella Lista
reverse()	Inverte l'ordine degli elementi nella lista
copy()	Restituisce una nuova CircularPositionalList che contiene gli stessi elementi della lista corrente memorizzati nello stesso ordine

Inoltre, la classe deve supportare i seguenti operatori:

x + y	Crea una lista con tutti gli elementi di x e tutti gli elementi di y inseriti dopo l'ultimo elemento di x
p in x	restituisce True se p è presente nella lista e False altrimenti
x[p]	Restituisce l'elemento contenuto nella position p
len(x)	Restituisce il numero di elementi contenuti in x
x[p] = e	Sostituisce l'elemento nella position p con e
del p	Rimuove l'elemento nella position p invalidando la position
__iter__	Generatore che restituisce gli elementi della lista a partire da quello che è identificato come primo fino a quello che è identificato come ultimo
__str__	Rappresenta il contenuto della lista come una sequenza di elementi, separati da virgole, partendo da quello che è identificato come primo

Tutti i metodi che prendono in input delle Position devono validarle e restituire un TypeError se l'input non è del tipo corretto ed un ValueError se la Position non appartiene alla lista corrente oppure è una Position non valida.

2) Scrivere un generatore **bubblesorted** che ordina gli elementi della CircularPositionalList e li restituisce nell'ordine risultante. Il generatore non deve modificare l'ordine in cui sono memorizzati gli elementi nella lista.

3) Scrivere una funzione **merge** che prende in input due CircularPositionalList ordinate e le fonde in una nuova CircularPositionalList ordinata.

4) Implementare la classe ScoreBoard che utilizza una CircularPositionalList per memorizzare i migliori x risultati di un gioco. I risultati sono rappresentati dalla classe Score che contiene il nome del player, il suo score, e la data in cui il punteggio è stato conseguito. La classe deve implementare i seguenti metodi:

<code>__init__(self, x = 10)</code>	Crea uno Scoreboard di dimensione x.
<code>__len__(self)</code>	Restituisce la dimensione dello Scoreboard.
<code>size(self)</code>	Restituisce il numero di Score presenti nello Scoreboard.
<code>is_empty(self)</code>	Restituisce True se non ci sono Score nello Scoreboard e False altrimenti.
<code>insert(self, s)</code>	Inserisce un nuovo Score nello ScoreBoard se e solo se non è peggiore dei risultati correntemente salvati. Non incrementa la dimensione dello Scoreboard
<code>merge(self, new)</code>	Fonde lo Scoreboard corrente con new selezionando i 10 migliori risultati.
<code>top(self, i = 1)</code>	Restituisce i migliori i Score nello Scoreboard.
<code>last(self, i = 1)</code>	Restituisce i peggiori i Score nello Scoreboard.

5) Scrivere uno script verifica.py che testi tutte le funzionalità implementate. In particolare se `l_uno` è una lista e `l_due = l_uno.copy()`, allora la stessa operazione su entrambe le sequenze deve produrre lo stesso risultato.