

Svolgete gli esercizi nei file indicati e poi salvate i file in una cartella zippata il cui nome deve essere formato dal **vostro cognome e dal vostro nome (in minuscolo)** separati da un **'\_'** (ad esempio, rossi\_mario). **Il file zippato deve contenere solo i quattro file contenenti il codice richiesto dagli esercizi e deve essere salvato sul desktop.**

**NON MODIFICATE IL CODICE GIA' SCRITTO NEI FILE FORNITI DALLA DOCENTE. SE MODIFICATE IL SUDDETTO CODICE, L'ESERCIZIO NON SARA' VALUTATO, SENZA ECCEZIONE ALCUNA.**

1. Scrivere nel file `esercizio1.py` una funzione che prende in input una sequenza di richieste (oggetti) e passa ciascuna richiesta ad una catena di gestori **ciascuno dei quali e' una coroutine. Le prime due prendono come argomenti due coroutine (ricevitore e successore), la terza prende come argomento solo una coroutine (il successore).**
  - Se la richiesta e' un oggetto di tipo `int` allora la richiesta viene gestita dal gestore `gestore_int` che stampa "Richiesta gestita da gestore\_int" e invia ad un ricevitore la coppia (oggetto, "file\_int") dove oggetto e' l'oggetto ricevuto e "file\_int" e il nome di un file.
  - Se la richiesta e' un oggetto di tipo `str` allora la richiesta viene gestita dal gestore `gestore_str` che stampa "Richiesta gestita da gestore\_str" e invia ad un ricevitore la coppia (oggetto, "file\_str") dove oggetto e' l'oggetto ricevuto e "file\_str" e il nome di un file.
  - Se l'oggetto non e' gestito dai gestori sopra descritti allora la richiesta viene gestita dal gestore `gestoreDiDefault` che stampa "Messaggio da gestoreDiDefault: non e' stato possibile gestire la richiesta {}".
  - I ricevitori sono delle **coroutine `writeInFile`** che ricevono tuple della forma (oggetto, nome\_file) e scrivono l'oggetto sotto forma di stringa seguita da uno spazio nel file `nome_file`.
2. Scrivere nel file `esercizio2.py` una versione modificata della catena dell'esercizio precedente in cui, oltre ai gestori gia' presenti nella versione precedente, vi sia un altro gestore chiamato **`gestore_tuple` che prende come argomento solo una coroutine (il successore).**
  - Se la richiesta e' una tupla allora la richiesta viene gestita da `gestore_tuple` che si comporta come segue: se il primo elemento della tupla e' il nome di un tipo (classe) definito nel modulo `esercizio2.py` allora il gestore crea un'istanza della classe utilizzando i restanti elementi della tupla come argomenti del metodo `__init__` della classe e stampa "Richiesta gestita da gestore\_tuple: e' stata creata un'istanza della classe {} con i seguenti attributi {}", dove al posto delle parentesi devono comparire il nome della classe e il contenuto di `__dict__` dell'istanza creata. Se il primo elemento della tupla non e' il nome di un tipo o e' un tipo che non e' definito in `esercizio2.py` allora il gestore stampa "Richiesta gestita da gestore\_tuple: istanza non creata perche' il primo elemento della tupla non e' una classe definita nel modulo esercizio2.py".

3. Si consideri la classe `ClasseBase` inserita tra i commenti nel file `esercizio3.py`. Si scriva un decoratore di nome `ClasseBase` che possa essere applicato ad una qualsiasi classe in modo che la classe così decorata si comporti come se fosse derivata da `ClasseBase`.
4. Scrivere nel file `esercizio4.py` un programma in cui vi è una classe **`CorsoDiLaurea`** le cui istanze possono essere osservate da un numero arbitrario di osservatori e che oltre agli attributi **`nome`** e **`ultima_matricola`**, ha i seguenti attributi che determinano lo stato delle sue istanze:
  - a) **`studenti`**: dizionario degli studenti iscritti al corso di laurea. Il dizionario è un'istanza di `dict` e contiene coppie chiave-valore dove chiave è una stringa che rappresenta il numero di matricola e valore è una tupla della forma (Cognome, Nome).
  - b) **`mediaVotoLaurea`**: un numero che rappresenta la media dei voti di laurea.
  - c) **`mediaVotoLaureaOK`**: flag che viene settato a `True` se e solo se il valore di `mediaVotoLaurea` diventa maggiore di 100 ed è settato a `False` se `mediaVotoLaurea` diventa minore o uguale di 100. Ogni volta che questo flag cambia valore (valore diverso da quello precedente) viene fatta la notifica agli osservatori.
  - d) **`numero_studenti`**: un intero che rappresenta il numero di iscritti al corso di Laurea.

Gli attributi `mediaVotoLaurea`, `mediaVotoLaureaOK` e `numero_studenti` sono accessibili con il loro nome e modificabili con `'='`.

`nome` e `ultima_matricola` sono stringhe inizializzate con le stringhe passate come argomenti ad `__init__`.

Scrivere inoltre gli osservatori **`Segreteria`** e **`Storico`**.

**`Segreteria`** deve stampare

- "Cambio stato: con le ultime immatricolazioni, il numero di studenti del Corso di Laurea in {} e {} \n" se il cambio stato è dovuto a nuove immatricolazioni (`numero_studenti` incrementato).
- "Cambio stato: con l'ultima seduta di Laurea, il numero di studenti del Corso di Laurea {} e {} \n" se il cambio stato è dovuto alla cancellazione di studenti dal dizionario (`numero_studenti` decrementato).
- "Cambio stato: con l'ultima seduta di Laurea, il voto medio del Corso di Laurea in {} e uguale a {} \n", se il cambio stato è dovuto ad una **modifica** del valore di `mediaVotoLaurea`.
- "Cambio stato: con l'ultima seduta di Laurea, il voto medio del Corso di Laurea in {} e diventato superiore a 100 \n" oppure "Cambio stato: con l'ultima seduta di Laurea, il voto medio del Corso di Laurea in {} e diventato minore o uguale di 100 \n" se il cambio stato è dovuto ad un aggiornamento di `mediaVotoLaureaOK` che ne ha aumentato o diminuito il valore, rispettivamente.

**`Storico`** deve creare una lista di tuple ognuna delle quali è una **`namedtuple`** di nome **`Tripla`** con i seguenti campi: `"nomeCorsoLaurea"`, `"votoMedio"`, `"tempo"`. Ogni volta che viene **modificato** il flag `mediaVotoLaureaOK`, l'osservatore **`Storico`** aggiunge alla lista una tripla con il nome del corso di Laurea, il nuovo voto medio di Laurea e il tempo (data e ora) in cui è cambiato il valore del flag `mediaVotoLaureaOK`.

**`Storico`** ha anche il metodo **`storia()`** che restituisce la suddetta lista di tuple.

NB: le classi **`Observed`** e **`CorsoDiLaurea`** sono già state inserite nel file `esercizio4.py` e devono essere completate.