

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 SUDETTO CODICE, L'ESERCIZIO NON SARA` VALUTATO, SENZA ECCEZIONE ALCUNA.

1. Scrivere nel file `esercizio1_da_6.py` o nel file `esercizio1_da_10.py`, a seconda della versione che si decide di svolgere, una coroutines `searcher(c1,c2,receiver1, receiver2)` che prende in input due caratteri `c1` e `c2` e due coroutines `receiver1, receiver2`, e si comporta come segue: ogni volta che riceve qualcosa verifica se questa e` il nome di un file **esistente** e nel caso in cui lo sia cerca all'interno del file le stringhe che cominciano con `c1` e quelle che cominciano con `c2` Le prime vengono inviate a `receiver1` mentre le seconde a `receiver2`. Nel caso in cui non esista un file con quel nome, la coroutines esegue solo la stampa della seguente stringa "Il file {} e` inesistente", dove al posto delle parentesi deve comparire il nome del file.
Scrivere inoltre una coroutines `listCreator(stop)` che ogni volta che riceve una stringa la inserisce in una lista (**la lista e` una variabile locale alla coroutines**) e stampa la lista aggiornata con l'aggiunta della nuova parola. I parametri `receiver1` e `receiver2` di `searcher` sono due coroutines `listCreator`.

Versione da al massimo 6 punti: la coroutines `listCreator` non fa niente altro rispetto a quanto sopra descritto (l'input `stop` viene ignorato).

Versione da al massimo 10 punti: La coroutines smette di ricevere parole non appena riceve una parola uguale alla stringa `stop` passata come argomento. Nell'implementazione della coroutines `searcher` occorre tenere conto del fatto che uno o entrambi i `receiver` potrebbero non ricevere piu` le parole inviate. Se ad un certo punto entrambi i `receiver` smettono di ricevere parole il `searcher` deve smettere anch'esso di ricevere stringhe.

Suggerimento: potete usare `re.findall(r'\w+', testo)` per estrarre parole da un testo.

2. Scrivere nel file `esercizio2.py` un decorator factory `decFact(L1,L2)` che prende in input una lista di stringhe e una stringa di oggetti e produce un decoratore che fa in modo che le istanze della classe nascano non solo con le variabili di istanza aggiunte dal metodo `__init__` della classe ma anche con le seguenti variabili di istanza:
 - per ogni $i = 1, \dots, \text{len}(L1)$, una variabile con nome uguale a quello della i -esima stringa di `L1` e valore uguale all' i -esimo oggetto di `L2`. Nel caso in cui `__init__` della classe originaria aggiungeva gia` una variabile di istanza con nome uguale all' i -esima stringa di `L1` allora il valore della variabile deve essere quello assegnato da `__init__` della classe originaria.

Appello del 17 gennaio 2020

3. Si considerino le classi Cane e Persona fornite nel file modulo.py. Scrivere la classe Casa con due cani e una persona (padrona del cane). La classe Casa fa uso di un mediatore per fare in modo che
- quando almeno uno dei due cani abbaia allora viene settata a True un flag di allerta (variabile self.allerta nella bozza di __init__ fornita in esercizio3.py).
 - quando il padrone torna a casa, se il flag allerta e` True, verifica per ciascun cane se tra l'ora in cui e` tornato a casa e l'ora in cui il cane ha mangiato per l'ultima volta sono trascorse piu` di 4 ore e in questo caso da` da mangiare al cane. Se nessuno dei due cani ha abbaiato tra il momento in cui il padrone e` uscito e quello in cui ha fatto ritorno (il flag e` False) allora il padrone al suo ritorno non fa niente.
- NB: puo` essere che il cane che abbaia non sia quello che ha fame o che ne abbaia uno solo ma che entrambi abbiano fame, o ancora che almeno uno dei cani abbaia ma nessuno dei due abbia fame.
- Suggerimento.** Per ciascuno dei due punti creare un callable: uno dei due deve essere associato ad entrambi i cani e l'altro deve essere associato al padrone. La differenza in ore tra due orari ora1 e ora2 si calcola cosi` : $(ora1-ora2).total_seconds()/60/60$.

4. Scrivere la funzione **cerca** all'interno del file esercizio4.py. Se la funzione ha bisogno di invocare altre procedure, fornire anche quest'ultime. La funzione cerca prende in input una lista di stringhe **listaParole** e una lista di nomi di file **listaDiFile**, e il parametro **concorrenza**. Facendo uso di **multiprocessing.JoinableQueue**, la funzione cerca deve stampare per ciascuno dei file di listaDiFile la parola di listaParole che appare piu` volte nel file. Cio` deve essere fatto con un processo separato per ogni file di listaDiFile e **le stampe devono essere effettuate nell'ordine in cui terminano i processi. Il callable usato per effettuare la ricerca nel singolo file deve prendere in input la lista di parole e il nome del file.**

Se non siete in grado di usare multiprocessing.JoinableQueue potete usare concurrent.futures ma saranno detratti dei punti