

Programmazione 1

Esami del 2018

STEFANO GUALANDI

October 9, 2018

Premessa

Questo documento presenta la raccolta dei testi di esame degli appelli del corso di Programmazione 1 tenuto nel 2018 agli studenti del primo anno del corso di laurea triennale di matematica, presso l'Università di Pavia.

I files dati citati nei testi d'esame sono disponibili al sito:

`http://www-dimat.unipv.it/gualandi/programming/exams/Raccolta2018.zip`

IMPORTANTE: I testi di questi esami possono essere utilizzati per svolgere degli esercizi. In nessun modo si deve intendere che questi temi di esame sono indicativi di quanto verrà chiesto agli appelli futuri. Quanto verrà richiesto agli esami futuri, rifletterà direttamente il programma svolto a lezione nell'anno accademico corrispondente.

Prima Parte

Nella prima parte dell'esame dovete usare esclusivamente le funzioni ricorsive e le `pairlists`, come studiate nella prima parte del corso. Dovete importare la libreria `pairlists` che viene fornita nella vostra cartella d'esame.

Svolgere i seguenti esercizi, completando il file `soluzione_template.py`:

1. Scrivere una funzione che prende in input una lista `As` di coefficienti di un polinomio di ordine n , e restituisce due funzioni, la prima funzione permette di calcolare in un dato punto la derivata prima del polinomio, la seconda funzione permette di calcolare la derivata seconda del polinomio.

Ad esempio, dato il polinomio:

$$3x^4 + 5x^3 + x - 2$$

i cui coefficienti sono $a_4 = 3, a_3 = 5, a_2 = 0, a_1 = 1, a_0 = -2$, il polinomio viene rappresentato dalla lista `As = [3,5,0,1,-2]`. Si deve scrivere una funzione `F(As)` che può essere usata nel modo seguente:

```
G, H = F(As)
print('Derivata prima in x=2:', G(2))
print('Derivata seconda in x=2:', H(2))
```

2. Scrivere una funzione che implementa l'algoritmo di *merge-sort* utilizzando le `pairlist`. Implementate una funzione `MergeSort(Ls, Compare)` che prende in input una lista di numeri interi (lista di tipo `pairlist`), e una predicato di confronto `Compare(x, y)` tra due numeri interi, e restituisce una nuova lista ordinata in base alla funzione di confronto data in input.

Ad esempio, data una lista di numeri casuali, la funzione può essere usata nel modo seguente:

```
Ls = MakeRandomInts(10, 0, 100)
print(Ls)
print(MergeSort(Ls, lambda x,y: x > y))
```

Seconda Parte

Nella seconda parte dell'esame potete usare qualsiasi libreria di python.

3. Si scriva una funzione `ParseNames(filename)` che legga il file `.csv` chiamato `"nomi_california.csv"` che trovate nella vostra cartella d'esame. Ogni riga del file contiene le seguenti informazioni:

- (a) La stringa `"CA"` che si riferisce allo stato `"California"`
- (b) Le lettere `"F"` o `"M"` per distinguere nomi femminili e maschili
- (c) Un intero che indica l'anno di riferimento
- (d) Un nome proprio di persona
- (e) Il numero di persone che nell'anno di riferimento hanno ricevuto quel nome

Le informazioni lette dal file `.csv` devono essere memorizzate in una lista di tuple, una tupla per ogni riga del file.

Le prime cinque righe del file input da utilizzare sono le seguenti:

CA,F,1910,Mary,295
 CA,F,1910,Helen,239
 CA,F,1910,Dorothy,220
 CA,F,1910,Margaret,163
 CA,F,1910,Frances,134

4. Scrivere una funzione `PlotTrend(Ls, Name)` che prende in input la lista restituita dalla funzione `ParseName`, un nome specificato dalla stringa `Name`, e disegni un plot del trend di quanto quel nome sia stato utilizzato anno per anno. In pratica, il plot deve avere in ascissa gli anni in cui il nome è stato utilizzato e in ordinata il numero di persone a cui effettivamente è stato dato quel nome. Usare la libreria `matplotlib`.
5. Scrivere una funzione `NameCounter(Ls, Name)` che prende in input la lista restituita dalla funzione `ParseName`, e conti il numero totale di persone nate sui diversi anni, con il `Name` passato in input alla funzione.
6. Scrivere una funzione `StringCounter(Ls, Str)` che prende in input la lista restituita dalla funzione `ParseName`, e conti il numero totale di persone nate sui diversi anni, che abbiano un nome che contiene la stringa `Str` passato in input alla funzione.
7. Scrivere una funzione `ProbabilityName(Ls)` che prende in input una lista di tuple come quella restituita dalla funzione `ParseName`, e restituisca un dizionario con una *chiave* per ogni nome che appare nella lista `Ls`, e come *valore* un numero che indichi il rapporto percentuale tra quante volte quel nome è stato usato, rispetto al totale del numero di persone considerate nella lista `Ls`.
8. Scrivere una funzione `TopNames(Ls, n)` che prende in input una lista di tuple come quella restituita dalla funzione `ParseName` e un numero intero positivo `n`, e che restituisce la lista di nomi più usati tra quelli contenuti nella lista `Ls`.
9. Scrivere una funzione `NamesOfTheYear(Ls)` che prende in input una lista di tuple come quella restituita dalla funzione `ParseName`, e che restituisca un dizionario che abbia una *chiave* per ogni anno considerato nel file, e come *valore* corrispondente ad una data chiave una tupla con i tre nomi più utilizzati in quell'anno.

Prima Parte

Nella prima parte dell'esame dovete usare esclusivamente le funzioni ricorsive e le `pairlists`, come studiate nella prima parte del corso. Dovete importare la libreria `pairlists` che viene fornita nella vostra cartella d'esame. Dovete svolgere i seguenti esercizi, completando il file `soluzione_template.py` e utilizzando le funzioni di test che trovate nel blocco main in coda al file.

1. **(Punti 3)** Scrivere una funzione `RadiceQuadrata(x)` che prende in input un numero `x` e restituisce la sua radice quadrata. Implementare questa funzione utilizzando una ricerca per bisezione.
2. **(Punti 4)** Scrivere una funzione che implementa l'algoritmo *quick-sort* utilizzando le `pairslist`. Implementate la funzione `QuickSort(Ls, Compare)` che prende in input una lista di numeri interi (lista di tipo `pairslist`), un predicato di confronto `Compare(x, y)` tra due numeri interi, e restituisce una nuova lista ordinata in base alla funzione `Compare`.

Seconda Parte

Nella seconda parte dell'esame potete usare qualsiasi libreria di python.

Si consideri un file di testo, chiamato `student-mat.csv`, contenente delle informazioni di studenti iscritti ad un liceo, e di cui si riportano sotto le prime 5 righe:

```
449905;F;18;course;0;4;3;6;5;6;6
447960;F;17;course;0;5;3;4;5;5;6
448096;F;15;other;3;4;3;10;7;8;10
446135;F;15;home;0;3;2;2;15;14;15
451296;F;16;home;0;4;3;4;6;10;10
```

Ogni riga contiene le seguenti informazioni, ordinate dal primo all'ultimo campo:

1. **Mat:** Numero di matricola (intero)
 2. **Sex:** Sesso
 3. **Age:** Età (intero)
 4. **Reason:** Motivo per aver scelto il liceo. Possibili valori: "home" per vicinanza a casa, "reputation" per il prestigio del liceo, "course" per il tipo di corsi offerti, e "other" per altri motivi
 5. **Failures:** Numero di volte che lo studente è stato bocciato
 6. **Famrel:** Qualità delle relazioni in famiglia (intero: da 1=scarse, a 5=eccellenti)
 7. **Freetime:** Quantità di tempo libero (intero: da 1=molto poco, a 5=tanto)
 8. **Absences:** Numero di assenze a scuola (intero: da 0 a 93)
 9. **G1:** Voto primo semestre (intero: da 0 a 20)
 10. **G2:** Voto secondo semestre (intero: da 0 a 20)
 11. **G3:** Voto finale (intero: da 0 a 20)
3. **(Punti 4)** Si scriva una classe chiamata `Student` che abbia i seguenti attributi: `mat`, `sex`, `age`, `reason`, `failures`, `famrel`, `freetime`, `absences`, `G1`, `G2`, `G3`. La classe ha una funzione `__init__(self, row)` che prende in input una lista di stringhe come letta dal file `student-mat.csv` e inizializza ciascun attributo elencato sopra. Inoltre la classe ha una funzione `__str__(self)` che, usando i suoi attributi, stampa una stringa formattata, ad esempio, come la seguente:

Student ID: 449905, sex: F, age: 18, G3: 6, G2: 6, G1: 5

4. **(Punti 3)** Si scriva una funzione `ParseStudents(filename)` che legge il file `student-mat.csv` che trovate nella vostra cartella d'esame. Le informazioni lette dal file `.csv` devono essere memorizzate in una lista di oggetti di tipo `Student`, un oggetto per ogni riga del file. La funzione `ParseStudents(filename)` restituisce tale lista.
5. **(Punti 3)** Scrivere una funzione `ComputeAgeAverage(Ls)` che prende in input la lista restituita dalla funzione `ParseStudents` e restituisce l'età media degli studenti contenuti in `Ls`.
6. **(Punti 4)** Scrivere una funzione `ComputeAgeStdDev(Ls)` che prende in input la lista restituita dalla funzione `ParseStudents`, e restituisce la deviazione standard dell'età degli studenti contenuti in `Ls`. Si ricorda che la deviazione standard di una popolazione si calcola come:

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2}$$

in cui x_i è l'età dell' i -esimo studente, μ l'età media e N il numero totale di studenti.

7. **(Punti 3)** Scrivere una funzione `CountGenders(Ls)` che prende in input la lista restituita dalla funzione `ParseStudents`, e restituisce un dizionario con una chiave per ogni gender trovato (campo `sex`), e che associ a ciascun chiave il numero totale di studenti trovati con quella chiave (gender).
8. **(Punti 4)** Scrivere una funzione `GradeSort(Ls, n=10)` che prende in input la lista restituita dalla funzione `ParseStudents`, un numero intero `n`, e restituisce in output la sotto lista dei primi `n` studenti ordinati lessicograficamente utilizzando prima il voto `G3`, poi il voto `G2` e infine il voto `G1`. Ad esempio: uno studente con ($G1 = 20, G2 = 19, G3 = 16$) è inferiore ad uno con ($G1 = 7, G2 = 20, G3 = 16$).

Per poter svolgere questo esercizio utilizzando la funzione `sorted()` di python si deve aggiungere alla classe `Student` il metodo `__lt__(self, b)` che confronta l'oggetto `self` con l'oggetto `b`, utilizzando il criterio descritto sopra.

9. **(Punti 5)** Scrivere una funzione `FilterSort(Ls, P, msg)` che prende in input la lista `Ls` restituita dalla funzione `ParseStudents`, un predicato `P` che prende in input un oggetto di tipo `Student`, e una stringa `msg` che contiene un messaggio da stampare a video.

La funzione deve (i) trovare gli studenti in `Ls` per cui il predicato `P` è verificato, (ii) selezionare i primi 3 studenti della sotto lista così trovata usando la funzione `GradSort`, e (iii) infine deve stampare a video il testo di `msg` seguito dai primi 3 studenti trovati al punto precedente.

Ad esempio la funzione `FilterSort` può essere usata nel modo seguente:

```
FilterSort(Ls, lambda x: x.age == 20, "Migliori voti dei ventenni:")
```

Aggiungere in coda al `main` due chiamate a `FilterSort` in modo che stampino a video (1) i voti delle migliori studentesse che hanno una famiglia solida oppure tanto tempo libero, e (2) i voti degli studenti maschi che non hanno una famiglia solida oppure hanno poco tempo libero. Suggerimento: usare la `FilterSort` passandogli delle opportune funzioni `lambda`.

Cognome:

Nome:

Matricola:

Postazione:

Prima Parte

Nella prima parte dell'esame dovete usare esclusivamente le funzioni ricorsive e le `pairlists`, come studiate nella prima parte del corso. Dovete importare la libreria `pairlists` che viene fornita nella vostra cartella d'esame. Dovete svolgere i seguenti esercizi, completando il file `soluzione_template.py` e utilizzando le funzioni di test che trovate nel blocco main in coda al file.

1. **(Punti 4)** Scrivere una funzione `Norma(x, t)` che prende in input una lista di numeri `x` (ovvero un vettore di $\mathbb{R}^n$) e un intero `t` e restituisce la norma del vettore. Se `t = 0` restituisce la norma infinito, ovvero $\max_i |x_i|$; se `t = 1` calcola la norma 1, ovvero $\sum_i |x_i|$; se `t = 2` la norma 2, ovvero $\sqrt{\sum_i x_i^2}$.
2. **(Punti 4)** Scrivere una funzione `Integrale(F, a, b, dx)` che calcoli l'integrale definito di una funzione `f` tra i limiti `a` e `b`, che può essere approssimato numericamente usando la formula:

$$\int_a^b f(x) dx = [f(a + dx/2) + f(a + dx/2 + dx) + f(a + dx/2 + 2dx) + \dots]dx$$

per valori piccoli di `dx`.

Seconda Parte

Nella seconda parte dell'esame potete usare qualsiasi libreria di python.

Si consideri un file di testo, chiamato `iris.csv`, contenente delle informazioni di fiori raccolti da un biologo, e di cui si riportano sotto le prime 5 righe:

```
sepalLength,sepalWidth,petalLength,petalWidth,species
5.1,3.5,1.4,0.2,setosa
4.9,3,1.4,0.2,setosa
4.7,3.2,1.3,0.2,setosa
4.6,3.1,1.5,0.2,setosa
```

Ogni riga contiene le seguenti informazioni, ordinate dal primo all'ultimo campo:

1. **sepalLength**: lunghezza del sepal in cm
 2. **sepalWidth**: larghezza del sepal in cm
 3. **petalLength**: lunghezza del petalo in cm
 4. **petalWidth**: larghezza del petalo in cm
 5. **species**: nome della specie
-
3. **(Punti 4)** Si scriva una classe chiamata `Flower` che abbia i seguenti attributi: `sepalLength`, `sepalWidth`, `petalLength`, `petalWidth`, `species`. La classe ha una funzione `__init__(self, row)` che prende in input una lista di stringhe come letta dal file `iris.csv` e inizializza ciascun attributo elencato sopra. Inoltre la classe ha una funzione `__str__(self)` che, usando i suoi attributi, stampa una stringa formattata come la seguente:

```
specie: setosa, lun petalo: 1.4, lar petalo: 0.2, lun sepal: 5.1, lar sepal: 3.5
```

4. **(Punti 3)** Si scriva una funzione `ParseFlowers(filename)` che legge il file `iris.csv` che trovate nella vostra cartella d'esame. Le informazioni lette dal file `.csv` devono essere memorizzate in una lista di oggetti di tipo `Flower`, un oggetto per ogni riga del file. La funzione `ParseFlowers(filename)` restituisce tale lista.
5. **(Punti 4)** Scrivere una funzione `ComputeAverage(Ls, attributo)` che prende in input la lista restituita dalla funzione `ParseFlowers` e restituisce la media dell'*attributo* specificato in input per i fiori contenuti in `Ls`. La variabile di input `attributo` può accettare una delle seguenti stringhe: `sepalLength`, `sepalWidth`, `petalLength`, `petalWidth`. Se il valore di `attributo` non è uno di quelli contemplati, la funzione solleva una eccezione di tipo `TypeError`.
6. **(Punti 4)** Scrivere una funzione `ComputeStdDev(Ls, attributo)` che prende in input la lista restituita dalla funzione `ParseFlowers`, e restituisce la deviazione standard rispetto a `attributo` dei fiori contenuti in `Ls`. Si ricorda che la deviazione standard di una popolazione si calcola come:

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2}$$

in cui x_i è l'attributo dell' i -esimo fiore, μ il valore medio dell'attributo e N il numero totale di fiori.

Se il valore di `attributo` non è uno di quelli contemplati, la funzione solleva una eccezione di tipo `TypeError`.

7. **(Punti 3)** Scrivere una funzione `CountSpecies(Ls)` che prende in input la lista restituita dalla funzione `ParseFlowers`, e restituisce un dizionario con una chiave per ogni specie trovata (campo `species`), e che associ a ciascuna chiave il numero totale di fiori trovati con quella chiave.
8. **(Punti 4)** Scrivere una funzione `LengthSort(Ls, n=10)` che prende in input la lista restituita dalla funzione `ParseFlowers`, un numero intero `n`, e restituisce in output la sotto lista dei primi `n` fiori ordinati al decrescere della somma della lunghezza del petalo e del sepal.

Per poter svolgere questo esercizio utilizzando la funzione `sorted()` di python si deve aggiungere alla classe `Flower` il metodo `__lt__(self, b)` che confronta l'oggetto `self` con l'oggetto `b`, utilizzando il criterio descritto sopra.

9. **(Punti 4)** Scrivere una funzione `FilterSort(Ls, P, msg)` che prende in input la lista `Ls` restituita dalla funzione `ParseFlowers`, un predicato `P` che prende in input un oggetto di tipo `Flower`, e una stringa `msg` che contiene un messaggio da stampare a video.

La funzione deve (i) trovare i fiori in `Ls` per cui il predicato `P` è verificato, (ii) selezionare i primi 3 fiori trovati usando la funzione `LengthSort`, e (iii) infine deve stampare a video il testo di `msg` seguito dai primi 3 fiori trovati al punto precedente.

Ad esempio la funzione `FilterSort` può essere usata nel modo seguente:

```
FilterSort(Ls, lambda x: x.petalLength > 2.5, "Fiori trovati:")
```

Cognome:

Nome:

Matricola:

Postazione:

Prima Parte

Nella prima parte dell'esame dovete usare esclusivamente le funzioni ricorsive e le `pairlists`, come studiate nella prima parte del corso. Dovete importare la libreria `pairlists` che viene fornita nella vostra cartella d'esame. Dovete svolgere i seguenti esercizi, completando il file `soluzione_template.py` e utilizzando le funzioni di test che trovate nel blocco main in coda al file.

1. **(Punti 3)** Scrivere una funzione `Euclide(P1, P2)` che prende in input due coppie di coordinate geografiche (*longitudine, latitudine*) e ne calcola la distanza euclidea, come se fossero due punti in un piano.

Scrivere poi la funzione `Haversine(P1, P2)` che calcola la distanza di Haversine tra i due punti, che tiene conto della curvatura della superficie terrestre. La funzione di Haversine utilizza le coordinate geografiche trasformate in radianti. Siano (λ_1, ϕ_1) le coordinate del primo punto in radianti, e siano (λ_2, ϕ_2) quelle del secondo. Inoltre sia $\Delta\phi = \phi_1 - \phi_2$ e $\Delta\lambda = \lambda_1 - \lambda_2$.

La formula da implementare per calcolare la distanza di Haversine è composta dai seguenti calcoli:

$$a = \sin^2\left(\frac{\Delta\phi}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\Delta\lambda}{2}\right)$$

da cui si ricava

$$c = 2 \operatorname{atan2}(\sqrt{a}, \sqrt{1-a})$$

da cui si calcola la distanza richiesta (in metri)

$$d = R c$$

in cui $R = 6371000$ è il raggio della terra in metri.

La funzione `Haversine(P1, P2)` da implementare deve calcolare e restituire d . Per l'implementazione utilizzare le funzioni di python seguenti: `from math import radians, sin, cos, atan2, sqrt`.

2. **(Punti 5)** Scrivere una funzione che implementa l'algoritmo di *merge-sort* utilizzando le `pairslist`. Implementate una funzione `MergeSort(Ls, Compare)` che prende in input una lista di numeri interi (lista di tipo `pairslist`), e una predicato di confronto `Compare(x, y)` tra due numeri interi, e restituisce una nuova lista ordinata in base alla funzione di confronto data in input.

Seconda Parte

Nella seconda parte dell'esame potete usare qualsiasi libreria di python.

Si consideri un file di testo, chiamato `train.csv`, contenente delle informazioni sulle corse dei Taxi di New York del 2016. Per ogni corsa vengono date le seguenti informazioni, una per ogni colonna, di cui sono da considerarne solo la data e ora inizio corsa, la durata, le coordinate inizio corsa e le coordinate fine corsa.

```
id: identificativo unico della corsa
vendor_id: identificativo unico dell'operatore
pickup_datetime: data e ora inizio corsa
dropoff_datetime: data e ora fine corsa
passenger_count: numero di passeggeri
```

```
pickup_longitude: longitudine inizio corsa
pickup_latitude: latitudine inizio corsa
dropoff_longitude: longitudine fine corsa
dropoff_latitude: latitudine finecorsa
store_and_fwd_flag: se connesso al server
trip_duration: durata del viaggio
```

3. **(Punti 3)** Si scriva una funzione `DateTime2Hour(buf)` che prende in input una stringa nel formato data e ora (esempio: "2016-05-25 13:59:58") e restituisce un intero che rappresenta l'ora (nell'esempio restituisce l'intero 13).
4. **(Punti 2)** Si scriva una funzione `BestN(Ds, n=5)` che prende in input un dizionario e restituisce i primi n elementi della lista di coppie (**chiave, valore**) ordinate in ordine decrescente.
5. **(Punti 4)** Si scriva una funzione `ParseTrips(filename)` che legge il file `train.csv` che trovate nella vostra cartella d'esame. Le informazioni lette dal file `.csv` devono essere memorizzate in una lista di tuple, una tuple per ogni riga. La tupla deve contenere in ordine: un numero intero che da la durata del viaggio; un numero intero (compreso tra 0 e 24) che da l'ora di inizio corsa (usare la funzione `DateTime2Hour`; una coppia di float che rappresenta le coordinate di inizio corsa; una coppia di float che rappresenta le coordinate di fine corsa).
6. **(Punti 3)** Scrivere una funzione `ComputeTrips(Ls)` che prende in input la lista restituita dalla funzione `ParseTrips` e restituisce un dizionario, con una chiave per ogni ora, e come valore il numero di corse effettuate che iniziano a quell'ora. Stampare a video i primi 3 elementi usando la funzione `BestN`.
7. **(Punti 4)** Scrivere una funzione `ComputeMeanHour(Ls)` che prende in input la lista restituita dalla funzione `ParseTrips` e restituisce un dizionario, con una chiave per ogni ora, e come valore il la durata media delle corse che iniziano a quell'ora. Stampare a video i primi 3 elementi usando la funzione `BestN`.
8. **(Punti 5)** Scrivere una funzione `ComputeNormalizedMean(Ls)` che prende in input la lista restituita dalla funzione `ParseTrips` e restituisce un dizionario, con una chiave per ogni ora, e come valore la media dei rapporti tra la distanza di Haversine percorsa dal viaggio e la sua durata. Stampare a video i primi 3 elementi usando la funzione `BestN`.
9. **(Punti 4)** Scrivere una funzione `ComputeError(Ls)` che prende in input la lista restituita dalla funzione `ParseTrips` e calcola l'errore quadratico medio tra la distanza euclidea e la distanza di Haversine per tutti i viaggi. Ovvero deve calcolare:

$$\text{RMSE}(x, y) = \sqrt{\frac{\sum_{i=1, \dots, n} (x_i - y_i)^2}{n}}$$

dove x è il vettore contenente la distanza di Haversine e y la distanza euclidea (chiaramente x_i e y_i devono riferirsi allo stesso viaggio).

Cognome:

Nome:

Matricola:

Postazione:

Prima Parte

Nella prima parte dell'esame dovete usare esclusivamente le funzioni ricorsive e le `pairlists`, come studiate nella prima parte del corso. Dovete importare la libreria `pairlists` che viene fornita nella vostra cartella d'esame. Dovete svolgere i seguenti esercizi, completando il file `soluzione_template.py` e utilizzando le funzioni di test che trovate nel blocco main in coda al file.

1. **(Punti 3)** Una funzione f è definita dalla regola seguente:

$$f(n) = \begin{cases} n & \text{if } n < 3 \\ f(n-1) + 2f(n-2) + 3f(n-3) & \text{if } n \geq 3 \end{cases}$$

- (a) Si scriva una procedura che calcoli f usando un **processo ricorsivo**.
- (b) Si scriva una procedura che calcoli f usando un **processo iterativo**.

2. **(Punti 5)** Scrivere una funzione che implementa l'algoritmo di *merge-sort* utilizzando le `pairlist`. Implementate una funzione `MergeSort(Ls, Compare)` che prende in input una lista di numeri interi (lista di tipo `pairlist`), e una predicato di confronto `Compare(x, y)` tra due numeri interi, e restituisce una nuova lista ordinata in base alla funzione di confronto data in input.

Seconda Parte

Nella seconda parte dell'esame potete usare qualsiasi libreria di python.

Si consideri un file di testo, chiamato `matrimoni.csv`, contenente i dati sui matrimoni del comune di Milano dal 2003 al 2016. Per ogni tipo di matrimonio, vengono date le seguenti informazioni:

```

Numerosità: numeri di matrimoni del tipo descritto dalla riga corrente
Anno: anno del matrimonio
Luogo: luogo del matrimonio
Rito: tipo di rito del matrimonio
ResidenzaMarito: residenza del marito
ResidenzaMoglie: residenza della moglie
StatoCivileMarito: stato civile del marito
StatoCivileMoglie: stato civile della moglie
CittadinanzaMarito: cittadinanza del marito
CittadinanzaMoglie: cittadinanza della moglie
EtaMarito: età del marito
EtaMoglie: età della moglie

```

3. **(Punti 4)** Si scriva una classe `Marriage` per rappresentare un oggetto di tipo matrimonio, che ha un attributo per ogni colonna del file di input. Il costruttore della classe `__init__(self, row)` prende in input una lista di stringhe corrispondente ad una riga letta dal file, con una stringa per attributo del file .csv, e inizializza tutti gli attributi della classe. Il metodo `__str__(self)` stampa una riga del tipo:

Anno: 2003, Rito: Cattolico, Eta Marito: 27, Eta Moglie: 32

4. **(Punti 4)** Si scriva una funzione `ParseInput(filename)` che legge il file `matrimoni.csv` che trovate nella vostra cartella d'esame. Le informazioni lette dal file .csv devono essere memorizzate in una lista di oggetti di tipo `Marriage`, un oggetto per ogni riga. **ATTENZIONE:** Per le righe in cui l'età del marito o l'età della moglie non è specificata, si usi un valore pari a 30.
5. **(Punti 4)** Scrivere una funzione `Percentage(Ls, name, value)` che prende in input la lista restituita dalla funzione `ParseInput`, il nome di uno degli attributi della classe `Marriage`, e un dato valore per quell'attributo, e calcola la percentuale di matrimoni con quel dato attributo. Per esempio, i matrimoni di rito cattolico sono 20.14%.
ATTENZIONE: Si consiglia di usare la funzione `getattr(object, name)` per recuperare l'attributo di nome `name` da un dato oggetto. Si ricorda che si può sempre usare l'help in linea, scrivendo `help(getattr)` dalla shell di python.
6. **(Punti 4)** Scrivere una funzione `MarriagesByYear(Ls)` che prende in input la lista restituita dalla funzione `ParseInput` e restituisce un dizionario, con una chiave per ogni anno, e come valore il numero di matrimoni celebrati in quell'anno. Richiamare la funzione nel `main` e stampare a video i tre anni con il maggior numero di matrimoni.
7. **(Punti 4)** Scrivere una funzione `PlotByYear(Ls)` che prende in input la lista restituita dalla funzione `ParseInput` e disegna un plot con l'andamento del numero di matrimoni, anno per anno (in ascissa si hanno gli anni, in ordinata il numero di matrimoni).
8. **(Punti 5)** Scrivere una funzione `Average(Ls)` che prende in input una lista di oggetti di tipo `Marriage`, e restituisce una coppia di valori che corrispondono all'età media degli uomini e delle donne. Richiamare questa funzione nel `__main__` per stampare a video l'età media degli uomini francesi, delle donne venezuelane, e degli uomini italiani che non sono cattolici.

Cognome:
Nome:
Matricola:
Postazione:

Prima Parte

Nella prima parte dell'esame dovete usare esclusivamente le funzioni ricorsive e le `pairlists`, come studiate nella prima parte del corso. Dovete importare la libreria `pairlists` che viene fornita nella vostra cartella d'esame. Dovete svolgere i seguenti esercizi, completando il file `soluzione_template.py` e utilizzando le funzioni di test che trovate nel blocco main in coda al file.

- (Punti 4)** Si scrive un predicato che dica se un dato numero n sia un numero primo.
Si può usare la definizione che n è un numero primo, se e solo se n è il suo più piccolo divisore maggiore di uno (suggerimento: scrivere prima una funzione che trova il più piccolo divisore di n).
È possibile scrivere questa procedura in modo tale che l'ordine di crescita per il numero di operazioni richieste sia $\Theta(\sqrt{n})$? Implementare tale versione.
- (Punti 4)** Scrivere una funzione che implementa l'algoritmo di *quick-sort* utilizzando le `pairlist`. Implementate una funzione `QuickSort(Ls, Compare)` che prende in input una lista di numeri interi (lista di tipo `pairlist`), e una predicato di confronto `Compare(x, y)` tra due numeri interi, e restituisce una nuova lista ordinata in base alla funzione di confronto data in input.

Seconda Parte

Nella seconda parte dell'esame potete usare qualsiasi libreria di python.

Si consideri un file di testo, chiamato `priceHouse.csv`, contenente i dati sui prezzi delle case vendute nello stato dello Iowa (US) dal 2006 al 2010. Per ogni casa venduta, vengono date le seguenti informazioni:

Id: identificativo unico della casa
LotArea: Dimensione della casa in piedi al quadrato
OverallCond: Un voto da uno a dieci sulle condizioni delle casa
YearBuilt: Anno di costruzione
GarageArea: Dimensione del garage in piedi al quadrato
MoSold: Mese dell'anno in cui è avvenuta la vendita (intero da 1 a 12)
YrSold: Anno in cui è avvenuta la vendita
SalePrice: Prezzo di vendita

- (Punti 4)** Si scriva una classe `House` per rappresentare la vendita di una casa, che ha un attributo per ogni colonna del file di input. Il costruttore della classe `__init__(self, row)` prende in input una lista di stringhe corrispondente ad una riga letta dal file, con una stringa per ogni attributo del file `.csv`, e inizializza tutti gli attributi della classe in modo opportuno.
Il metodo `__str__(self)` deve stampare una riga del tipo:

Id: 692, LotArea: 2000, MonthSold: 1, YearSold: 2007, Price: 755000

- (Punti 4)** Si scriva una funzione `ParseInput(filename)` che legge il file `priceHouse.csv` che trovate nella vostra cartella d'esame. Le informazioni lette dal file `.csv` devono essere memorizzate in una lista di oggetti di tipo `House`, un oggetto per ogni riga.

ATTENZIONE: Per la dimensione della casa e del garage si devono convertire le dimensioni in metri al quadrato, prima di memorizzare il dato nell'attributo corrispondente. Un piede al quadrato equivale a $\frac{1}{0.092903}$ metri al quadrato.

3. **(Punti 4)** Scrivere una funzione `PercentageQuality(Ls, Op, value)` che prende in input la lista restituita dalla funzione `ParseInput`, la stringa 'GE' o 'LE' per indicare rispettivamente maggiore o uguale, e minore o uguale, e calcola la percentuale di case vendute con un prezzo maggiore o uguale, oppure minore uguale, al valore indicato.
4. **(Punti 3)** Scrivere una funzione `SortBySale(Ls)` che prende in input la lista restituita dalla funzione `ParseInput` e restituisce una lista ordinata per la data di vendita. Per svolgere questo esercizio, si deve definire la funzione builtin `__le__(self, b)` per la classe `House`, e usare la funzione builtin `sorted`.
5. **(Punti 4)** Scrivere una funzione `PlotByYearSale(Ls)` che prende in input la lista restituita dalla funzione `ParseInput` e disegna un plot con l'andamento del numero di case vendute, anno per anno (in ascissa si hanno gli anni, in ordinata il numero di case vendute).
6. **(Punti 6)** Scrivere una funzione `MakeHistogram(Ls, breaks)` che prende in input una lista di oggetti di tipo `House` e restituisce un dizionario, che come chiave ha delle coppie di numeri interi (a, b) , e come valore il logaritmo del numero di case vendute ad un prezzo contenuto nell'intervallo $[a..b)$. Il numero di intervalli scelti deve essere pari a `breaks`, e viene determinato in modo automatico in base al valore minimo e massimo dei prezzi delle case.

Richiamare questa funzione nel `__main__`, e partire dal dizionario ottenuto si stampi a video un istogramma stilizzato, tipo quello seguente:

```
# Chiamando la funzione: H = MakeHistogram(Ls, 10)
# Si ottiene:

(34900, 106910) *****
(106910, 178920) *****
(178920, 250930) *****
(250930, 322940) *****
(322940, 394950) *****
(394950, 466960) *****
(466960, 538970) *****
(538970, 610980) ****
(610980, 682990) ***
(682990, 755000)
```