Programmazione 1

Esami del 2019

STEFANO GUALANDI

March 18, 2020

Premessa

Questo documento presenta la raccolta dei testi di esame degli appelli del corso di Programmazione 1 tenuto nel 2019 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/Raccolta2019.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.

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 `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 2)** Scrivere una funzione `Filter(P, Ls)` che prende in input un predicato `P` e una lista `Ls`, e restituisce una nuova lista che contiene solo gli elementi di `Ls` che soddisfano il predicato `P`.
3. **(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.

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.

Si consideri un file di testo, chiamato `consumi.csv`, contenente i dati di consumo di energia di una casa. Per ogni riga del file, tranne la prima, vengono date le seguenti informazioni:

```
Data: formato dd/mm/yyyy
Ora: formato hh:mm:ss
GlobalActivePower: household global minute-averaged active power (in kilowatt) => gap
GlobalReactivePower: household global minute-averaged reactive power (in kilowatt) => grp
Voltage: minute-averaged voltage (in volt) => v
GlobalIntensity: household global minute-averaged current intensity (in ampere) => gi
SubMetering1: energy sub-metering No. 1 => sm1
SubMetering2: energy sub-metering No. 2 => sm2
SubMetering3: energy sub-metering No. 3 => sm3
```

1. **(Punti 3)** Si scriva una classe `DateTime` per rappresentare una data e ora, e che abbia sei attributi: anno, mese, giorni, ora, minuti, secondi. Il costruttore della classe `__init__(self, date, time)` prende in input due stringhe, ad esempio `'12/11/2010'` e `'23:43:00'`, e inizializza tutti gli attributi della classe in modo opportuno.

Il metodo `__str__(self)` deve stampare una riga del tipo:

```
23h43m 12/11/2010
```

2. **(Punti 4)** Si scriva una classe `Measure` per rappresentare la misura di consumo di un singolo minuto, e che abbia i seguenti attributi: `datetime`, `gap`, `grp`, `v`, `gi`, `sm1`, `sm2`, `sm3`, `sm4`. L'attributo `datetime` contiene un oggetto di tipo `DateTime` costruito in modo da riflettere la data e ora della misura, e l'attributo `sm4` viene impostato attraverso la formula seguente:

$$gap * 1000/60 - sm1 - sm2 - sm3$$

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:

```
datetime: 23h43m 12/11/2010, SubMeters: 0.0 0.0 0.0
```

3. **(Punti 3)** Si scriva una funzione `ParseInput(filename)` che legge il file `consumi.csv` che trovate nella vostra cartella d'esame. Le informazioni lette dal file `.csv` devono essere memorizzate in una lista di oggetti di tipo `Measure`, un oggetto per ogni riga. La funzione restituisce tale lista.
4. **(Punti 4)** Scrivere una funzione `TopGAP(Ls, n=5)` che ordina le misure lette dal file. Per ordinare la lista di misure si deve usare la funzione native delle liste `sort`, aggiungendo alla classe `Measure` il predicato `__lt__(self, b)` che implementa la logica seguente: il confronto tra due misure viene fatto in modo lessicografico sui tre attributi `sm1`, `sm2`, `sm3`. La funzione restituisce i primi n dei valori della lista ordinata.
5. **(Punti 4)** Scrivere una funzione `TopAVG(Ls, n=3)` che prende in input la lista restituita dalla funzione `ParseInput` e ordina in modo decrescente le misure usando la funzione nativa `sorted`, convertendo ciascuna misura nella media dei tre valori `sm1`, `sm2`, `sm3`. La funzione restituisce i primi n dei valori della lista ordinata.
6. **(Punti 4)** Scrivere una funzione `Plot(Ls, a)` che prende in input la lista di misure e l'attributo `a` e crea un plot con in ascissa le ore del giorno (da 0 a 24) e in ordinata i valori totali dei consumi corrispondenti all'attributo `a` passato in input.

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 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. **(Punti 4)** Scrivere una funzione `Map(F, Ls)` che prende in input una funzione `F` e una lista `Ls`, e restituisce una nuova lista che contiene il risultato di aver applicato la funzione `F` a ciascun elemento di `Ls`. Implementare la funzione `Map` in termini di una generica funzione `Reduce`.
3. **(Punti 3)** 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. Ad esempio, data una lista di numeri casuali, la funzione può essere usata nel modo seguente:

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

Seconda Parte

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

Si consideri un file di testo, chiamato `heart.csv`, contenente i dati di alcuni pazienti di un ospedale, di cui si vuole studiare la correlazione tra malattie cardiache e una serie di dati biomedici. Per ogni riga del file, tranne la prima che contiene le intestazioni, vengono riportate le seguenti informazioni:

```
age      (int):  età
sex      (str):  sesso                                (1='M' e 0='F')
cp       (int):  tipo di dolore al petto                (valori possibili: 0,1,2,3)
trestbps (int):  pressione del sangue a riposo
chol     (int):  livello di colesterolo in mg/dl
fbs      (bool): glicemia a digiuno, pari a 1 se > 120 mg/dl, pari a 0 altrimenti
restecg  (int):  elettrocardiogramma a riposo            (valori possibili: 0,1,2)
```

```
thalach (int): massimo battito cardiaco
exang (bool): dolore al petto (1=True, 0=False)
target (bool): malato (1=True, 0=False)
```

Esempio:

```
age;sex;cp;trestbps;chol;fbs;restecg;thalach;exang;target
63;1;3;145;233;1;0;150;0;1
37;1;2;130;250;0;1;187;0;1
```

4. **(Punti 4)** Si scriva una classe `Patient` per rappresentare i dati di un paziente, con un attributo per ogni colonna del file `.csv`. Usare i nomi degli attributi indicati sopra, e convertire i dati usando il tipo indicato tra parentesi (`int`, `str`, `bool`). Il costruttore della classe `__init__(self, row)` prende in input una lista di stringhe corrispondente ad una riga letta dal file e inizializza tutti gli attributi della classe in modo opportuno. Se uno qualsiasi dei valori di input non può essere convertito correttamente, il costruttore della classe solleva un'eccezione di tipo `ValueError` tramite il comando `raise`.

Il metodo `__str__(self)` deve stampare una riga del tipo:

```
età: 56, sesso: F, malato: False, chol: 409, thalach: 1
```

5. **(Punti 4)** Si scriva una funzione `ParseInput(filename)` che legge il file `heart.csv` che trovate nella vostra cartella d'esame. Le informazioni lette dal file `.csv` devono essere memorizzate in una lista di oggetti di tipo `Patient`, un oggetto per ogni riga. La funzione restituisce tale lista. Se una riga del file di input contiene dei dati mancanti o errati (ovvero, il costruttore della classe `Patient` solleva un'eccezione), allora deve essere ignorata in fase di lettura.
6. **(Punti 3)** Scrivere una funzione `Percentage(Ls)` che prende in input una lista di oggetti di tipo `Patient` letti dal file, e restituisce una coppia di valori: il primo valore indica la percentuale di maschi con problemi cardiaci (`target=True`), e il secondo valore indica la percentuale di donne con problemi cardiaci.
7. **(Punti 4)** Scrivere una funzione `TopRatio(Ls, n=3)` che prende in input la lista restituita dalla funzione `ParseInput` e restituisce i primi n pazienti, tra quelli che hanno dolore al petto, con il più alto rapporto tra il massimo battito cardiaco (`thalach`) e il livello di colesterolo (`chol`).
8. **(Punti 4)** Scrivere una funzione `CholMeanAge(Ls, As)` che prende in input due liste, la prima è la lista `Ls` dei dati letti dal file, la seconda è una lista `As` di età di interesse (ovvero, è una lista di numeri interi). La funzione restituisce un dizionario, con una chiave per ogni elemento di `As`, a cui corrisponde come valore il valore medio di colesterolo per tutti i pazienti in `Ls` dell'età corrispondente alla chiave considerata. Nel `main` scrivere il ciclo che stampa il chiave e valore del dizionario restituito dalla funzione, in ordine di età decrescente.
9. **(Punti 3)** Scrivere una funzione `Plot(Ls)` che prende in input la lista di dati dei pazienti letti dal file e produce un plot che abbia in ascissa l'età e in ordinata il valore medio di colesterolo corrispondente all'età. Il plot produce tre curve lineari a tratti che rappresentano il valore medio di colesterolo in funzione dell'età. La prima curva rappresenta la media per i pazienti che hanno un valore dell'elettrocardiogramma a riposo pari a 0, la seconda per un valore pari a 1, la terza per un valore pari a 2. Rappresentare il plot per i pazienti con età compresa tra i 40 e i 70 anni. Si consiglia di usare la funzione `CholMeanAge(Ls, As)` dell'esercizio precedente.

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)** Si scrive un predicato `IsPrime(n)` che prende in input un numero naturale n , e che restituisce `True` se il numero n è 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.

Dire quali dei seguenti numeri è un numero primo: 13, 15, 1317, 3719, 57143.

2. **(Punti 3)** Scrivere una funzione che implementa l'algoritmo di *selection sort* utilizzando le `pairlist`. La funzione `SelectionSort(Ls, Compare)` prende in input una lista di numeri interi (lista di tipo `pairlist`), e una predicato di confronto `Compare(x, y)` tra due numeri, 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(SelectionSort(Ls, lambda x,y: x > y))
```

3. **(Punti 4)** Si scrive una funzione `Integrale(F, a, b, dx)` che prende in input una funzione `F`, e che calcola l'integrale definito tra a e b di F , usando rettangoli con base di lunghezza dx . Per un esempio su come deve essere usata, si veda il blocco al codice interno al `main`.

Seconda Parte

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

Si consideri un file di testo, chiamato `alcohol.csv`, contenente i dati sui consumi di alcolici rilevati (misurati in litri) dalla organizzazione mondiale della sanità (*Global Health Observatory data repository*). Per ogni riga del file, tranne la prima che contiene le intestazioni, vengono riportate le seguenti informazioni:

```
country          (str) : nazione a cui si riferisce la riga
beverageTypes   (str) : tipo di bevanda alcolica: AllTypes, Beer, Wine, Spirit, Other
2016            (float): consumo medio annuale per persona nel 2016 (se rilevato)
2015            (float): consumo medio annuale per persona nel 2015 (se rilevato)
2014            (float): consumo medio annuale per persona nel 2014 (se rilevato)
2013            (float): consumo medio annuale per persona nel 2013 (se rilevato)
2012            (float): consumo medio annuale per persona nel 2012 (se rilevato)
2011            (float): consumo medio annuale per persona nel 2011 (se rilevato)
2010            (float): consumo medio annuale per persona nel 2010 (se rilevato)
```

Esempio di prime righe del file:

```
Country;Beverage Types;2016;2015;2014;2013;2012;2011;2010
Afghanistan;AllTypes;;0.02;0.03;0.03;0.04;0.04;0.03
Afghanistan;Beer;;0.01;0.01;0.01;0.01;0.01;0.01
Afghanistan;Wine;;0;0;0;0;0;0
```

```
Afghanistan;Spirits;;0.02;0.02;0.02;0.03;0.03;0.02
Afghanistan;Other;;0;0;0;0;0;0
Albania;AllTypes;5.07;4.77;4.81;5.06;5.43;5.65;5.53
Albania;Beer;1.74;1.57;1.58;1.82;1.81;1.88;1.72
....
```

4. **(Punti 6)** Si scriva una funzione `ParseInput(filename)` che legge il file `alcohol.csv` che trovate nella vostra cartella d'esame. Le informazioni lette dal file `.csv` devono essere memorizzate in una lista di tuple, una tupla per ogni riga. La funzione restituisce la lista di tuple lette. Se una riga del file di input contiene dei dati mancanti per alcuni anni, allora il valore mancante deve essere sostituito con il valore medio dei dati presenti nella riga stessa.

Esempio: nella prima riga del file dopo l'intestazione, manca il dato per il 2016, che deve essere sostituito con la media dei dati rimanenti, ovvero la media di: `0.02;0.03;0.03;0.04;0.04;0.03`.

5. **(Punti 4)** Si scriva una funzione `MakeDict(Ls)` che prende in input la lista di tuple lette dal file, e restituisce un dizionario, con una chiave per ogni nazione. Per ogni chiave (ogni nazione), viene memorizzato un secondo dizionario con una chiave per ognuno dei seguenti attributi: `AllTypes`, `Beer`, `Wine`, `Spirit`, `Other`. Ad ogni attributo viene associata la corrispondente lista dei consumi medi per persona per anno.

Esempio: per la chiave `'Afghanistan'` avremo come valore il dizionario:

```
{ 'Spirits': (0.02, 0.02, 0.02, 0.02, 0.03, 0.03, 0.02),
  'AllTypes': (0.03, 0.02, 0.03, 0.03, 0.04, 0.04, 0.03),
  'Other': (0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0),
  'Beer': (0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01),
  'Wine': (0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0) }
```

6. **(Punti 3)** Scrivere una funzione `MaxAlcoholType(D, t)` che prende in input un dizionario `D` e una stringa `t`, e trova la nazione che ha consumato il maggior quantitativo di alcohol di tipo `t`. La funzione restituisce una coppia di valori: il primo elemento è il nome della nazione che ha realizzato il consumo massimo, e il secondo valore è il consumo massimo.
7. **(Punti 3)** Scrivere una funzione `TopAlcoholType(D, t, n=5)` che prende in input un dizionario `D` e una stringa `t`, e restituisce una lista di coppie, in cui il primo elemento è il nome di una nazione, e il secondo il relativo consumo totale di bevanda alcolica di tipo `t`. Vengono restituite le prime `n` coppie che hanno il consumo maggiore della bevanda di tipo `t`.
8. **(Punti 4)** Scrivere una funzione `Growing(D, t)` che prende in input un dizionario `D` e una stringa `t`, e trova le nazioni il cui consumo di bevande di tipo `t` è stato strettamente crescente negli anni. La funzione restituisce un dizionario con una chiave per ogni nazione, e come valore la lista con i consumi annui corrispondenti della bevanda di tipo `t`.
9. **(Punti 3)** Scrivere una funzione `Plot(Ls, c)` che prende in input la lista di tuple lette dal file, e il nome di una nazione `c`, e produce un unico plot in cui viene rappresentato l'andamento del consumo annuo di: `Beer`, `Wine`, `Spirit` (in ascissa si hanno gli anni dal 2010 al 2016 e in ordinata i relativi consumi annui).

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) $d = Rc$ 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 `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. 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.**

- (Punti 3)** Scrivere una funzione `Zip(As, Bs)` che prende in input due `pairlists` della stessa lunghezza, e restituisce una lista di coppie di elementi, in cui l'*n*-esimo elemento di `As` é accoppiato con l'*n*-esimo elemento di `Bs`.
- (Punti 2)** Modificare la funzione `Zip(As, Bs)` dell'esercizio precedente in modo tale che quando le due liste non hanno la stessa lunghezza, viene sollevata un'eccezione di tipo `Runtime`.
- (Punti 4)** Scrivere una funzione che implementa l'algoritmo di *quick sort* utilizzando le `pairlists`. Implementare la 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.

Usare `QuickSort` per ordinare una lista di coppie, con un ordine decrescente rispetto al secondo valore di ciascuna coppia.

Ad esempio, le due funzioni precedenti possono essere usate nel modo seguente:

```
As = (85, (18, (28, (82, (93, (23, (16, 'EL'))))))))
Bs = (9, (68, (27, (95, (37, (3, (55, 'EL'))))))))
Zs = Zip(As, Bs)
Dopo lo Zip:
Zs = ((85, 9), ((18, 68), ((28, 27), ((82, 95), ((93, 37), ((23, 3), ((16, 55), 'EL'))))))))
Lista di coppie ordinata in modo decrescente:
((82, 95), ((18, 68), ((16, 55), ((93, 37), ((28, 27), ((85, 9), ((23, 3), 'EL'))))))))
```

Seconda Parte

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

Si consideri un file di testo, chiamato `words.csv`, contenente un sotto campione di dati di un *word embedding*. Il file contiene, in ogni riga, una parola seguita da un vettore di $\mathbb{R}^{300}$, ovvero da una lista di 300 numeri. Per esempio, le prime sei righe del file sono (per ogni riga, dei 300 numeri, riportiamo solo i primi sei e gli ultimi due):

```
, -0.082752 0.67204 -0.14987 -0.064983 0.056491 0.40228 ... -0.37846 -0.06589
. 0.012001 0.20751 -0.12578 -0.59325 0.12525 0.15975 ... -0.36049 -0.035
the 0.27204 -0.06203 -0.1884 0.023225 -0.018158 0.0067192 ... -0.18317 0.1323
and -0.18567 0.066008 -0.25209 -0.11725 0.26513 0.064908 ... 0.12302 0.3312
to 0.31924 0.06316 -0.27858 0.2612 0.079248 -0.21462 ... 0.097801 0.25045
of 0.060216 0.21799 -0.04249 -0.38618 -0.15388 0.034635 ... -0.16692 -0.094085
```

- (Punti 4)** Si scriva una funzione `ParseWords(filename)` che legge il file `word.csv` che trovate nella vostra cartella d'esame. Le informazioni lette dal file `.csv` devono essere memorizzate in un dizionario `Ws`, con una coppia (*chiave*, *valore*) per ogni riga del file: la chiave é la prima parola della riga, e il valore é la lista dei 300 numeri seguenti. La funzione restituisce il dizionario `Ws`.

5. **(Punti 3)** Si scriva un predicato `IsWord(w)` che prende in input una stringa `w` e restituisce `True` se e solo se la string `w` contiene solo caratteri dell'alfabeto. Ad esempio, `IsWord('hello')` restituisce `True`, ma `IsWord('hello!')` restituisce `False`.
6. **(Punti 3)** Scrivere una funzione `Norm(x)` che prende in input un vettore di n elementi $x_1 \dots, x_n$ (ovvero una lista di n numeri), e ne calcola la norma euclidea:

$$\text{norm}(x) = \sqrt{\sum_{i=1, \dots, n} x_i^2}$$

Verrà valutato positivamente l'utilizzo della funzione high-order *reduce*.

7. **(Punti 3)** Scrivere una funzione `Distance(x, y)` che prende in input due vettori $x, y \in \mathbb{R}^n$, e calcola la distanza euclidea tra i due punti:

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

Verrà valutato positivamente l'utilizzo della funzione high-order *reduce*.

8. **(Punti 5)** Scrivere una funzione `SameDistance(WS, w1, w2, w3)` che prende in input il dizionario `WS` come letto dal file `.csv` e tre diverse parole `w1`, `w2` e `w3`. Sia $\bar{d}$ la distanza euclidea tra i vettori associati a `w1` e a `w2`. Trovare tra le chiavi del dizionario `WS` che soddisfano il predicato `IsWord(word)`, quella che ha la distanza da `w3` il più vicino possibile a $\bar{d}$.

Per testare la funzione usare le parole: `w1='Monday'`, `w2='day'`, `w3='President'`.

9. **(Punti 6)** Scrivere una funzione `NearestPairs(Ds)` che prende in input il dizionario `WS` come letto dal file `.csv` e calcola le prime cinque coppie di parole che hanno la distanza euclidea minima tra i rispettivi vettori. Si considerino solo le parole che soddisfano il predicato `IsWord(w)`. La funzione deve restituire la lista delle cinque coppie di parole che hanno distanza minima tra di loro.

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 `solution_template.py` e **utilizzando le funzioni di test che trovate nel blocco main in coda al file.**

1. **(Punti 3)** Il Massimo Comun Divisore (MCD) di due numeri intero a e b è definito come il più grande numero intero che divide sia a che b senza resto. L'algoritmo di Euclide si basa sull'osservazione che, se r è il resto di quando a è diviso per b , allora i divisori comuni di a e b sono esattamente gli stessi divisori comuni tra b e r . Quindi possiamo usare l'equazione

$$MCD(a, b) = MCD(b, r)$$

per ridurre il problema di trovare i divisori comuni calcolando il MCD tra coppie di numeri interi via via più piccoli. Per esempio:

$$MCD(206, 40) = MCD(40, 6) = MCD(6, 4) = MCD(4, 2) = MCD(2, 0) = 2$$

Scrivere una procedura che calcola il massimo comune divisore usando l'algoritmo di Euclide.

2. **(Punti 3)** Scrivere una funzione `RemoveAll(As, value)` che prende in input una lista `As` (lista di tipo `pairlist`) e un valore `value` e restituisce in output una lista con tutti i valori di `As` tranne tutte le occorrenze di `value`.
3. **(Punti 4)** Scrivere una funzione che implementa l'algoritmo di *quick sort* utilizzando le `pairlists`. Implementare la 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.

Usare `QuickSort` per ordinare una lista di numeri interi in ordine prima in ordine crescente e poi in ordine decrescente.

Seconda Parte

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

Si consideri un file di testo, chiamato `digits.csv`, contenente un sotto campione di immagini digitalizzate di alcune cifre, memorizzate come un **vettore** di $28 \times 28 = 784$ pixels. Il file contiene, in ogni riga, un numero che indica la cifra memorizzata, seguita da un vettore di 784 elementi. Ogni elemento è un numero compreso tra 0 e 255 che indica l'intensità di grigio del corrispondente pixel. Per esempio, le prime tre righe del file potrebbero essere (per ogni riga, dei 784 pixels, riportiamo solo i primi sei e gli ultimi due):

```
5,255,248,248,230,191,168,...,214,198
5,240,250,252,250,248,254,...,207,203
6,255,250,255,252,250,254,...,232,241
```

4. **(Punti 2)** Si scriva un predicato `IsPosInteger(w)` che prende in input una stringa `w` e restituisce `True` se e solo se la stringa `w` contiene solo caratteri numerici, e può essere convertito in un numero intero positivo. Ad esempio, `IsPosInteger('17')` restituisce `True`, ma `IsPosInteger('-1')` restituisce `False`.

5. **(Punti 4)** Si scriva una funzione `ParseDigits(filename)` che legge il file `digits.csv` che trovate nella vostra cartella d'esame. Le informazioni lette dal file devono essere memorizzate in una lista di coppie `Ls`, in cui il primo elemento della coppia è valore della cifra e il secondo valore della coppia è il vettore corrispondente ad un'immagine digitalizzata della cifra corrispondente. La funzione restituisce la lista `Ls`. Per ogni singolo elemento letto file, si usi il predicato `IsPosInteger(w)` per controllare che possa essere convertito in un numero intero positivo. Se un elemento non può essere convertito in un numero intero positivo, il programma deve sollevare un'eccezione di tipo `TypeError`.
6. **(Punti 3)** Si scriva una funzione `Norm(x)` che prende in input un vettore di n elementi $x_1 \dots, x_n$ (ovvero una lista di n numeri), e ne calcola la norma euclidea:

$$\text{norm}(x) = \sqrt{\sum_{i=1, \dots, n} x_i^2}$$

Verrà valutato positivamente l'utilizzo della funzione high-order *reduce*.

7. **(Punti 4)** Scrivere una funzione `Distance(x, y)` che prende in input due vettori $x, y \in \mathbb{R}^n$, e calcola la distanza euclidea tra i due vettori:

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

Verrà valutato positivamente l'utilizzo della funzione high-order *reduce*.

8. **(Punti 4)** Scrivere una funzione `NearestDigit(digit, Ls)` che prende in input un vettore `digit` di un'immagine digitalizzata e una lista di coppie (`cifra, immagine`) come letta dalla funzione `ParseDigits(filename)`. La funzione deve trovare, tra tutte le immagini contenute in `Ls`, quella più simile a `digit`, dove la similitudine è data dalla distanza Euclidea definita al punto precedente (minore è la distanza, maggiore è la similitudine). Trovata l'immagine più simile, la funzione restituisce una tripla (una tupla di 3 elementi), contenente il valore di distanza minimo, il valore della cifra trovata, e il vettore dell'immagine corrispondente (si veda il corpo del `__main__` per un esempio).
9. **(Punti 6)** Scrivere una funzione `Baricentro(As, value)` che data una lista di coppie (`cifra, immagine`), e il valore di una cifra `value`, esegue le operazioni seguenti.

- (a) Seleziona dalla lista di coppie `As` solo gli elementi corrispondenti a immagini della cifra `value`. Sia `X` la lista così ottenuta e sia N la lunghezza di `X`.
- (b) Si implementi una funzione che calcoli il baricentro delle immagini contenute nella lista `X`. Il baricentro rappresenta una "media" di un'insieme di immagini date, ed è definito da una matrice B , che viene calcolata nel modo seguente:

$$b_{ij} = \frac{1}{N} \sum_{k=1, \dots, N} X(k, i, j)$$

in cui $X(k, i, j)$ rappresenta il pixel (i, j) dell'immagine k -esima.

- (c) Usare la funzione `DrawMatrix(A)` data nel template di soluzione per visualizzare l'immagine corrispondente al baricentro trovato.