Esercizi di Laboratorio di Programmazione

Elena Buscaroli Federico Pigozzi
Gianluca Guglielmo Stefano A. Russo (supervisione)

18 Dicembre 2021, Foglio 2

Versione modificata del foglio esercizi preparato da Andrea Mecchina, Michele Rispoli, Pietro Morichetti e Nicolas Solomita per il laboratorio di programmazione a.a. 2020/2021.

1 Eccezioni e Sanitizzazione

Esercizio 1

Il centro di ricerca sullo sviluppo di intelligenze artificiali di Dublino vi ha assegnato il compito di sviluppare uno script in Python (aka un programma) per un automa: l'androide deve simulare il comportamento umano dell'atto di vestirsi. La normale procedura che tutti noi eseguiamo in maniera automatica può essere riassunta nei seguenti steps (funzioni):

1. `biancheria`, atto di indossare la biancheria intima;
2. `calzini`, atto di indossare i calzini;
3. `maglia`, atto di indossare una maglietta;
4. `pantaloni`, atto di indossare i pantaloni;
5. `calzatura`, atto di indossare le scarpe;

Ogni funzione restituisce 1 se ha avuto successo, altrimenti 0.

Sviluppare una classe **Automa** che presenta come **attributi** i capi di vestiario sopra citati, tutti inizializzati a `None`, ed un **metodo** per ogni azione richiesta dall'automa.
Si osservi che in caso di successo per un'azione, il corrispondente attributo è impostato a True (esempio: l'automa prova ad indossare la maglietta, l'azione ha avuto successo, e prima che il metodo ritorni 1, l'attributo `self.maglia` diventa `True`). Il corpo principale dello script non dovrà vestire l'automa in maniera procedurale, ma casuale sfruttando un `while`.
Di seguito vi viene proposto una porzione di pseudo-codice, non esaustiva:

```
import random

class Automa {
    ...
}

# funzione esterna
def esegui(automa, capo) {
    # in base al valore dell'input
    # richiama il metodo della classe
    # associata al capo di abbigliamento

    # ritorna quanto restituito dal metodo chiamato
}

...
capi_vestiario = [...]
vestito = True
...

while(vestito) {
    # seleziona a caso un capo
    # (G1) gestire in caso il capo selezionato
    # va contro la sequenza della procedura
    # chiama esegui per provare ad indossare il capo
    # (G2) gestire in caso di insuccesso
}

print("automa vestito correttamente")
```

Il concetto di *casualità* all'interno di un linguaggio di programmazione è espresso per mezzo di funzioni apposite, che sono state fornite in calce all'esercizio. Osserviamo che esistono due situazioni in cui si potrebbe avere un comportamento scorretto da parte dell'androide:

- **G1**: il capo selezionato (casualmente) non può essere indossato poichè non rispetta la naturale procedura (es: indossare pantaloni prima della biancheria). Questo tipo di anomalia può essere gestito attraverso il costrutto `if`: non sanitizza ma si limita a riportare l'errore ed il capo non viene indossato.
- **G2**: l'automa ha tentato di indossare il capo ma non ci è riuscito (il metodo ha restituito 0 in maniera casuale). Questo tipo di errore è per noi un errore "fatale" che non possiamo gestire; si utilizzi il costrutto raise riportando l'errore.

Nota: Importare la libreria `random` per poter utilizzare le seguenti funzioni di casualità:

- `random.choice(lista)`, la funzione estrae casualmente e restituisce un elemento della lista passata come argomento (la lista resta immutata).

- `random.randint(a, b)`, la funzione restituisce un intero compreso nell'intervallo `[a, b]`.
- altre funzioni al seguente link: `https://pynative.com/python/random/`.

Siete invitati a gestire anche casi di errore provenienti da queste funzioni.

2 Testing

Esercizio 2

Definire una classe **Calcolatrice** che realizza le principali funzioni del calcolo aritmetico (somma, sottrazione, moltiplicazione e divisione) e anche le funzioni di potenza, di modulo, di radice e conversione di base. In particolare si osservi che:

- non sono ammesse operazioni tra oggetti che che non siano parte delle note classi numeriche (Integer, Float, ...); a meno che i valori numerici siano passati in forma di stringa;
- la divisione per zero non è ammesso;
- l'operazione di potenza è ammessa solo con base e potenza interi;
- l'operazione di potenza ammette base negativa;
- l'operazione di radice è ammessa solo per valori strettamente positivi;
- l'operazione di conversione di base è esclusivamente dalla base 10 a base 2;
- sanitizzare dove possibile.

Oltre alla classe precedente, realizzare una classe **Test** che andrà a testare tutti i metodi della classe **Calcolatrice**; in caso di errori, la classe deve stampare su schermo un messaggio di avvertimento con tutte le informazioni necessarie. Provare a completare l'esercizio seguendo la filosofia del test-driven.

Esercizio 3

Abbiamo modificato l'oggetto **CSVFile** in modo che la funzione `get_data(self, start=None, end=None)` permetta di leggere un particolare intervallo di righe del file. Nel caso in cui non vengano passati i parametri `start` e `end`, vogliamo che la funzione ritorni tutte le righe del file. Scrivere una funzione di test che verifichi che il comportamento del codice modificato sia quello desiderato. Per esempio, un'idea potrebbe essere creare una funzione `test_sum(file)` che confronti:

- la lista contente i valori di tutte le righe ottenuta tramite `file.get_data()`,
- la lista contenente i valori di tutte le righe ottenuta fornendo esplicitamente gli estremi delle righe, tramite `file.get_data(1,size)` dove `size` è il numero di righe totali contenute nel file.

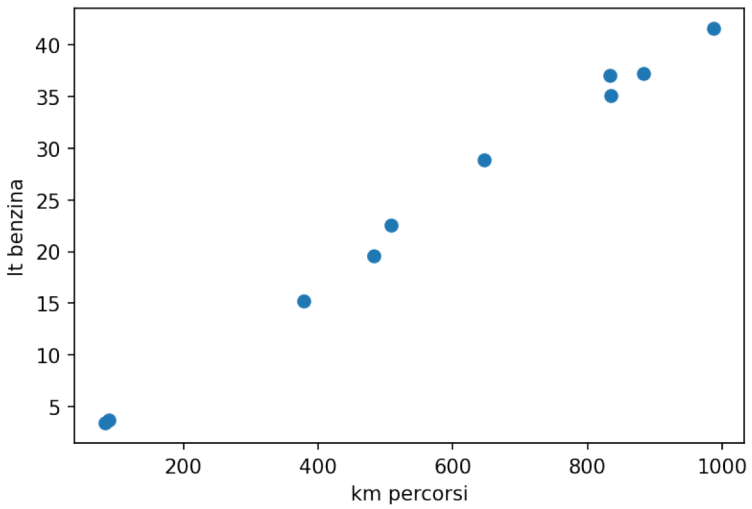
Un possibile criterio di confronto potrebbe essere verificare che la somma dei valori di entrambe le liste coincida. Nel caso in cui il criterio di confronto non sia rispettato, la funzione di test deve lanciare un'eccezione. Un ulteriore possibile controllo potrebbe essere verificare che la somma degli elementi della lista che seleziona solo un numero valido di righe sia minore o uguale alla somma di tutti gli elementi della lista.

3 Modelli

Esercizio 4 - opzionale

Tre studenti di statistica decidono di organizzare un road trip per festeggiare la prossima conclusione della sessione estiva. Scelte le tappe, calcolano che complessivamente dovranno percorrere circa 2500 km, e sarebbero interessati a stimare quanto spenderanno a testa in benzina. Siccome per l'ultimo esame della sessione gli è richiesto di sviluppare un modello statistico in python su dati da loro raccolti, decidono di unire l'utile al dilettevole, sfruttando l'occasione per risolvere il calcolo di loro interesse. Contattando amici e parenti che possiedono la stessa macchina riescono ad ottenere i seguenti dati:

km percorsi	lt di benzina
833	37
987	41.6
883	37.2
378	15.2
84	3.4
483	19.6
835	35.1
646	28.9
508	22.6
90	3.7



I punti sul grafico sembrano ragionevolmente approssimabili con una linea retta: il problema dunque è risolvibile con un modello lineare.
Implementare la classe **LinearModel** costituita dai seguenti membri:

- il costruttore `__init__(self)`, che inizializza a **None** i campi:
  - `self.angular coeff`
  - `self.intercept`
  - `self.train_data`
- il metodo `fit(self,train_data)` che, data una serie di n coppie di valori (x_i, y_i), effettui le seguenti operazioni:

  1. determini il coefficiente angolare della retta di fit, calcolato secondo la formula

  $$m = P_{x,y} * \frac{s_y}{s_x},$$

  dove:
  - $P_{x,y} = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{(y)})}{\sqrt{\sum_i (x_i - \bar{x})^2 \sum_i (y_i - \bar{y})^2}}$ è il coefficiente di correlazione di Pearson tra le x e le y. $\bar{x}$ e $\bar{y}$ nella formula indicano la media aritmetica delle x e delle y.
  - $s_x = \sqrt{\frac{\sum_i (x_i - \bar{x})^2}{n-1}}$ è la deviazione standard corretta delle x; s_y è definita analogamente.

  Si salvi il valore di m in `self.angular_coeff`.

  2. determini il valore dell'intercetta della retta di fit, calcolato secondo la formula

  $$q = \bar{y} + m * \bar{x},$$

  dove m è il coefficiente angolare appena determinato.
  Si salvi il valore di q in `self.intercept`.

  3. salvi la serie di dati ricevuta in input nel campo `self.train_data`;

  Il metodo non deve restituire nulla.

- il metodo `predict(self, xp)` che, dato un vettore dato un vettore di n valori x_i, calcoli e restituisca le corrispondenti y_i predette dal modello, secondo la formula

  $$y_i = m * x_i + q,$$

  dove m e q sono i parametri del modello calcolati con `fit()`. Qualora tali valori non fossero ancora stati calcolati (i.e. fossero ancora **None**), il metodo deve laciare un'eccezione.

Per effettuare i calcoli descritti e gestire i valori in ingresso ed in uscita dai vari metodi è opportuno utilizzare il modulo python `Numpy`, che fornisce classi e funzioni per effettuare facilmente operazioni su vettori e matrici di numeri. Per risolvere l'esercizio sono sufficienti le seguenti nozioni al riguardo:

- Inserire l'istruzione `import numpy as np` all'inizio del vostro script per poter utilizzare numpy;
- Il campo `self.train_data` ed i parametri `train_data` e `xp` passati rispettivamente ai metodi `fit` e `predict` possono essere rappresentati con oggetti della classe `np.ndarray`, che non sono altro che vettori o matrici (i.e. "griglie") multidimensionali di numeri. Funzionano in maniera analoga alle liste per quanto riguarda l'accesso ai singoli elementi, ma in più consentono di effettuare operazioni su tutti i numeri al loro interno in maniera molto intuitiva (vedi seguito). Ecco alcuni esempi:
  - `x = np.ndarray([[1,2],[3,4],[5,6]])` definisce un `ndarray` di dimensioni `(3,2)` contenente i numeri specificati;
  - `x[2,1]` restituisce il valore `6`;
  - `x[:,1]` restituisce il vettore `[2,4,6]` (anch'esso `ndarray`);
  - `x.shape` restituisce la tupla `(3,2)`, ovvero le dimensioni di `x`;
- `np.sum(x)` restituisce la somma di tutti gli elementi all'interno dell'ndarray `x` (in questo caso 21)
- `x*5` restituisce un nuovo ndarray ottenuto moltiplicando i valori di `x` per 5 (in questo caso `[[5,10],[15,20],[25,30]]`). Discorso analogo vale per le operazioni +,-,/ e ** (l'ultima è la potenza)
- Operazioni tra due ndarray delle stesse dimensioni vengono applicate elemento per elemento (ad esempio, se definisco `y = np.ndarray([[0,-1],[0,-2],[0,-3]])`, il comando `x+y` restituirà l'ndarray `[[1,1],[3,2],[5,3]]`)
- `np.mean(x)` restituisce la media aritmetica dei valori in `x` (in questo caso 3.5)
- `np.std(x,ddof=1))` restituisce la deviazione standard corretta dei valori in `x` (in questo caso 1.870...)

Una volta definita la classe, applicare il metodo fit sui dati riportati in tabella, dopodichè calcolare quanti litri di benzina saranno necessari per compiere il tragitto applicando il metodo predict. Infine, sapendo che la benzina costa mediamente 1.4€/l, stimare quanto dovrà spendere ciascuno dei tre ragazzi.

Qui trovate un template già pronto per svolgere l'esercizio: `https://github.com/drpOpZ/ModelloLineare`.