Esercizi di Fondamenti di Informatica

Andrea Gussoni
andrea1.gussoni at polimi.it

Politecnico di Milano

December 19, 2019

Table of Contents

Section 1

Ripasso Esercitazione Precedente

Ripasso

Problemi con esercizi della precedente esercitazione?

- Gestione files
- Outer sort

Section 2

Ricorsione

Palindromi

Ideare un programma, che tramite l'utilizzo di funzioni ricorsive, verifichi che in una lista di numeri, il numero in posizione *i* sia minore del numero in posizione *n* - *i* (dove *n* è la lunghezza della lista).

Palindromi I

Una possibile soluzione

```
def main() :
    uno = [4,8,1,10,15,7]
    print(uno)
    print("Lista:", palindromo(uno))
    due = [6,1,8,9,4]
    print(due)
    print("Lista:", palindromo(due))

## Funzione che controlla se una stringa e'
## un palindromo
#  @param lista lista da controllare
#  @return True se la lista rispetta il criterio
#
def palindromo(lista) :
```

Palindromi II

```
lun = len(lista)

# Se una stringa e' formata da un carattere
# e' un palindromo.
if lun <= 1 :
    return True
else :
    # Ottieniamo il primo e l'ultimo carattere
    primo = lista[0]
    ultimo = lista[lun - 1]

    # Stabiliamo se entrambi sono caratteri
    # Due lettere
    if primo < ultimo:
        # Lettere uguali, togliamole e facciamo
        # il passo ricorsivo
```

Palindromi III

```
        sottostringa = lista[1 : lun - 1]
        return palindromo(sottostringa)
    else :
        return False

main()
```

Permutazioni

Ideare un programma, che tramite l'utilizzo di funzioni ricorsive, stampi tutte le permutazioni possibili di una stringa.

Permutazioni I

Una possibile soluzione

```
def main() :
    for permutazione in permuta("ciao") :
        print(permutazione)

## Funzione per generare tutte le permutazioni
#  @param parola input da permutare
#  @return lista delle parole permutate
#
def permuta(parola) :
    permutazioni = []

    # Se abbiamo una stringa vuota, siamo
    # arrivati in fondo
    if len(parola) == 0 :
```

Permutazioni II

```
        permutazioni.append(parola)
        return permutazioni
    else :
        # Ciclichiamo su ogni carattere delle stringa.
        for i in range(len(parola)) :
            # Rimuoviamo l'iesimo carattere
            ridotta = parola[ : i] + parola[i + 1 :]

            # Generate all permutations of the simpler word.
            permutazioni_ridotta = permuta(ridotta)

            # Aggiungiamo il carattere rimosso in testa ad
            # ogni permutazione ridotta
            for stringa in permutazioni_ridotta :
                permutazioni.append(parola[i] + stringa)
```

Permutazioni III

```
    # Restituisci tutte le permutazioni
    return permutazioni

main()
```

Ricerca Binaria

Ideare un programma, che tramite l'utilizzo di funzioni ricorsive, effettui la ricerca di un elemento in una lista di interi seguendo l'approccio della ricerca binaria.

Ricerca Binaria I

Una possibile soluzione

```
# Ritorniamo l'indice a cui l'elemento e'
# presente, o -1 se l'elemento non e' presente
def ricerca(lista, minimo, massimo, x):

    # Caso base
    if massimo >= minimo:

        centrale = minimo + (massimo - minimo)/2
        centrale = int(centrale)

        # Se l'elemento cercato e' proprio quello
        # a meta'
        if lista[centrale] == x:
            return centrale
```

Ricerca Binaria II

```
        # Se l'elemento e' piu' piccolo di quello
        # centrale, deve essere nella meta' sinistra
        elif lista[centrale] > x:
            return ricerca(lista, minimo, centrale - 1, x)

        # L'elemento deve essere presente nella
        # meta' a destea
        else:
            return ricerca(lista, centrale + 1, massimo, x)

    else:
        # l'elemento non ' presente
        return -1

def main():
```

Ricerca Binaria III

```
    lista = [1, 3, 4, 7, 10, 15, 21, 35, 99]
    print(ricerca(lista, 0, len(lista), 15))
    print(ricerca(lista, 0, len(lista), 25))

main()
```

Mergesort

Ideare un programma, che tramite l'utilizzo di funzioni ricorsive, implementi l'algoritmo di ordinamento mergesort.

Mergesort I

Una possibile soluzione

```
## Ordina una lista usando l'algoritmo di mergesort
#  @param lista da ordinare
#
def merge_sort(lista):
    if len(lista) <= 1:
        return lista
    centrale = len(lista) // 2
    meta_1 = lista[ : centrale]
    meta_2 = lista[centrale : ]
    meta_1 = merge_sort(meta_1)
    meta_2 = merge_sort(meta_2)
    ordinata = incolla_liste(meta_1, meta_2)
    return ordinata
```

Mergesort II

```
## Unisce due liste ordinate in una ordinata
#  @param prima lista ordinata
#  @param seconda lista ordinata
#  @return la lista ordinata
#
def incolla_liste(prima, seconda):
    ordinata = []
    indice_1 = 0   # Indice prima lista.
    indice_2 = 0  # Indice seconda lista

    # Copiamo il piu' piccolo degli elementi nella lista ordinata
    while indice_1 < len(prima) and indice_2 < len(seconda) :
        if prima[indice_1] < seconda[indice_2] :
            ordinata.append(prima[indice_1])
            indice_1 = indice_1 + 1
        else :
```

Mergesort III

```
            ordinata.append(seconda[indice_2])
            indice_2 = indice_2 + 1


    # Copia il resto della prima lista.
    while indice_1 < len(prima) :
        ordinata.append(prima[indice_1])
        indice_1 = indice_1 + 1

    # Copia il resto della seconda lista.
    while indice_2 < len(seconda) :
        ordinata.append(seconda[indice_2])
        indice_2 = indice_2 + 1

    return ordinata
```

Mergesort IV

```
def main():
    lista = [66, 98, 34, 56, 34, 12]
    print(merge_sort(lista))

main()
```

Section 3

Ripasso Generale

Cifrario di Cesare

Ideare un programma che prenda come input un file di testo, e ne effettui la cifratura con il cifrario di Cesare. La cifratura di Cesare opera sostituendo ciascun carattere con il carattere che segue 3 posizioni il carattere originale nell'ordine alfabetico.

Cifrario di Cesare I

Una possibile soluzione

```
def cifra(car, chiave):
    num_lettere = 26

    if car >= 'A' and car <= 'Z':
        base = ord('A')
    elif car >= 'a' and car <= 'z':
        base = ord('a')
    else:
        return car

    offset = ord(car) - base + chiave
    if offset > num_lettere:
        offset = offset - num_lettere
    elif offset < 0:
```

Cifrario di Cesare II

```
        offset = offset + num_letter

    return chr(base + offset)

def main():
    chiave = 3
    input_file = 'plaintext.txt'
    output_file = 'ciphertext.txt'

    ifile = open(input_file, 'r')
    ofile = open(output_file, 'w')

    # Read all the characters in the file.
    for linea in ifile:
        for car in linea:
            nuovo_car = cifra(car, chiave)
```

Cifrario di Cesare III

```
            ofile.write(nuovo_car)

    ifile.close()
    ofile.close()

main()
```

Togli Maiuscolo

Scrivere un programma che legga un file di testo, e che trasformi le parole scritte completamente in maiscolo nelle corrispondenti parole in minuscolo.

Togli Maiuscolo I

Una possibile soluzione

```
def main():
    # Leggi i nomi di input e output
    input_name = input("File da censurare: ")
    output_name = input("File sistemato: ")

    # Open the files.
    originale = open(input_name, "r")
    sistemato = open(output_name, "w")

    # Analizziamo tutte le righe
    for linea in originale:
        lista_linea = linea.split()
        nuova_lista_linea = []
        for parola in lista_linea:
```

Togli Maiuscolo II

```
            if parola.isupper():
                nuova_lista_linea.append(parola.lower())
            else:
                nuova_lista_linea.append(parola)

        # Write the line to the output file.
        nuova_linea = ''
        for parola in nuova_lista_linea:
            nuova_linea = nuova_linea + parola + ' '
        sistemato.write(nuova_linea +  '\n')

    # Close the files.
    originale.close()
    sistemato.close()

main()
```

Togli Maiuscolo III

Negativo Immagine

Ideare un programma che presa in input un'immagine salvata in formato BitMap, ne produca una copia che rappresenti il negativo dell'immagine originaria.

Negativo Immagine I

Una possibile soluzione

```
from io import SEEK_CUR
from sys import exit
from shutil import copyfile

def main() :
    filename_orig = 'queen_mary.bmp'
    filename = 'neg_' + filename_orig
    copyfile(filename_orig, filename)

    # Apriamo l'immagino come file binario
    file = open(filename, "rb+")

    # Leggiamo alcune informazioni dell'immagine
    dimensione = decodifica(file, 2)
```

Negativo Immagine II

```
inizio = decodifica(file, 10)
larghezza = decodifica(file, 18)
altezza = decodifica(file, 22)

# Ciascuna riga di pixel occupa un multiplo di 4 byte
riga_pixel = larghezza * 3
if riga_pixel % 4 == 0 :
    padding = 0
else :
    padding = 4 - riga_pixel % 4

# Controlliamo che sia una immagine valida
if dimensione != (inizio + (riga_pixel + padding) * altezza):
    sys.exit("Il file non e' un BMP a 24 bit.")

# Muoviamoci al primo pixel.
```

Negativo Immagine III

```
    file.seek(inizio)

    # Process the individual pixels.
    for row in range(altezza):
        for col in range(larghezza):
            processa_pixel(file)

        # Salta il padding a fine riga.
        file.seek(padding, SEEK_CUR)

    file.close()

def processa_pixel(file) :
    # Leggiamo i pixel come byte.
    pixel_bytes = file.read(3)
    blu = pixel_bytes[0]
```

Negativo Immagine IV

```
        verde = pixel_bytes[1]
        rosso = pixel_bytes[2]

        # Facciamo il negativo dei pixel.
        blu = 255 - blu
        verde = 255 - verde
        rosso = 255 - rosso

        # Torniamo all'inizio del pixel.
        file.seek(-3, SEEK_CUR)
        file.write(bytes([blu, verde, rosso]))

## Funzione che legge informazioni in little endian
def decodifica(file, offset) :
    # Muoviamoci all'inizio del byte.
    file.seek(offset)
```

Negativo Immagine V

```
    # Leggiamo 4 byte e costruiamo un intero.
    bytes = file.read(4)
    res = 0
    base = 1
    for i in range(4) :
        res = res + bytes[i] * base
        base = base * 256

    return res

main()
```

Robustezza Password

Ideare un programma che valuti la robustezza di una password inserita tramite i seguenti parametri:

- Una password per essere accettabile deve essere lunga almeno 10 caratteri.
- La password è debole se contiene solo caratteri minuscoli.
- La password è buona se contiene caratteri alfanumerici sia maiuscoli che minuscoli.
- La password è forte se contiene caratteri alfanumerici maiuscoli e minuscoli ed un carattere speciale.
- In tutti gli altri casi la password non è accettabile.

Robustezza Password I

Una possibile soluzione

```
def cerca_password(filename, utente):
    file = open(filename, 'r')
    lista_pass = []
    for riga in file:
        lista_riga = riga.split()
        if lista_riga[1].lower() == utente.lower():
            lista_pass.append(lista_riga[3])
    file.close()
    return lista_pass

def sicurezza(password):

    if len(password) >= 10 and (password.isupper() == True \
                                or password.islower() == True):
```

Robustezza Password II

```
        return 'Debole'
    elif len(password) >= 10 and password.isalnum() == True \
                             and password.isupper() == False \
                             and password.islower() == False \
                             and password.isdigit() == False:
        return "Media"
    elif len(password) >= 10 and password.isupper() == False \
                             and password.islower() == False \
                             and password.isdigit() == False \
                             and password.find('.,:;!?—'):
        return "Forte"
    else:
        return "Non Valida"

def main():
    nome = input("Utente: ")
```

Robustezza Password III

```
    password_db = 'pass_db.txt'

    lista = cerca_password(password_db, nome)

    for password in lista:
        valutazione = sicurezza(password)
        print('Password: ', password,
              ' ha livello sicurezza" ', valutazione)

main()
```

Correttore

Ideare un programma che esegua il compito di correttore ortografico, controllando che le parole di un certo testo siano tutte presenti in un altro file che useremo come dizionario di tutte le parole corrette presenti nel nostro alfabeto.

Correttore I

Una possibile soluzione

```
def main():
    dizionario_parole = leggi_parole('words', "")
    parole_testo = leggi_parole('testo.txt', "")

    parole_errate = parole_testo.difference(dizionario_parole)

    print('Parole errate:')
    for parola in parole_errate:
        print(parola)

def leggi_parole(nome_file, marcatore):
    set_parole = set()
    file = open(nome_file, "r")
    skip = True
```

Correttore II

```
for linea in file :
    linea = linea.strip()
    if not skip:
        # Dividiamo la parola in corrispondenza
        # di qualsiasi carattere che non sia una lettera
        parti = linea.split()
        for parola in parti:
            parola = parola.strip('.,?!')
            parola = pulisci(parola)
            if len(parola) > 0 :
                set_parole.add(parola.lower())
    elif linea.find(marcatore) >= 0 :
        skip = False

file.close()
```

Correttore III

```
    return set_parole

def pulisci(stringa) :
    risultato = ""
    for car in stringa :
        if car.isalpha() :
            risultato = risultato + car

    return risultato.lower()

main()
```

Filtro CSV

Ideare un programma che, preso in input un file del formato CSV, contenente un catalogo di film e avente per ognuno i seguenti campi: *Nome*, *Anno*, *Registi*, *Produttori*, *Attori*, crei un nuovo file dove verranno salvati sempre in formato csv i soli film prodotti negli anni ottanta, e mantenendo solo il *Nome*, l'*Anno*, e il campo degli *Attori*.

Filtro CSV I

Una possibile soluzione

```
from csv import reader, writer

# Apri il file contenente l'elenco dei film
input_file = open("film.csv")
csv_input = reader(input_file)

# Apri il file dove scriveremo i film filtrati
output_file = open("film-filtrati.csv", "w")
csv_output = writer(output_file)

# Aggiungi intestazione del file.
intestazione = ["Nome","Anno","Attori"]
csv_output.writerow(intestazione)
```

Filtro CSV II

```
# Skip the row of column headers in the reader.
next(csv_input)

# Filter the rows of data.
for riga in csv_input :
  anno = int(riga[1])
  if anno >= 1980 and anno <= 1989 :
    nuova_riga = [riga[0], riga[1], riga[4]]
    csv_output.writerow(nuova_riga)

input_file.close()
output_file.close()
```

Albo Studenti

Ideare un programma che permetta la gestione di un albo di studenti. In particolare dovremo offrire le seguenti funzionalità:

- L'albo dovrà contenere un numero arbitrario di studenti.
- Ogni studente dovrà avere i campi: nome, cognome, matricola e una serie di esami sostenuti con voto associato.
- Dovremo offrire la possibilità di inserire un nuovo studente (e le relative informazioni anagrafiche).
- Dovremo offrire la possibilità di inserire un nuovo esame sostenuto con relativo voto associato.
- Dovremo offrire la funzionalità di stampa delle informazioni relative all'intero albo degli studenti.
- Dovremo offrire di stampare la media delle votazioni conseguite da una certa matricola.

Nel risolvere l'esercizio si consiglia l'utilizzo di uno o più dizionari, e della realizzazione di un piccolo menù interattivo che guidi l'utente tra le opzioni indicate precedentemente.

Albo Studenti I

Una possibile soluzione

```
def inserisci_studente():
    studente = {}
    nome = input("Inserisci nome")
    cognome = input("Inserisci cognome")
    matricola = int(input("Inserisci matricola"))
    esami = {}
    studente["nome"] = nome
    studente["cognome"] = cognome
    studente["matricola"] = matricola
    studente["esami"] = esami
    return studente

def aggiungi_voto(studenti, matricola, esame, voto):
    if voto < 18 or voto > 30:
```

Albo Studenti II

```
            print("Il voto non e' valido")
            return
        for studente in studenti:
            if studente["matricola"] == matricola:
                if esame not in studente["esami"]:
                    studente["esami"][esame] = voto
                    return
                else:
                    print("Esame gia' presente")
                    return

        print("La matricola non e' presente nell'albo")

def stampa_informazioni(studenti):
    for studente in studenti:
        print("nome:", studente["nome"],
```

Albo Studenti III

```
                "cognome:", studente["cognome"],
                "matricola:", studente["matricola"])
            print("esami:", studente["esami"])

def media_carriera(studenti, matricola):
    for studente in studenti:
        if studente["matricola"] == matricola:
            esami = studente["esami"]
            totale = 0
            for esame in esami:
                totale = totale + esami[esame]
            return totale/len(esami)
        else:
            print("La matricola non e' presente nell'albo")
            return 0
```

Albo Studenti IV

```
def menu():
    print("Fai la tua scelta")
    print("1 Per inserire un nuovo studente")
    print("2 Per aggiungere un nuovo voto")
    print("3 Per stampare la media dei voti di uno studente")
    print("4 Per stampare le informazioni di tutti gli studenti")
    print("5 Per terminare il programma")
    scelta = int(input(""))
    return scelta

def main():
    albo_studenti = []
    scelta = menu()
    while scelta != 5:
        if scelta == 1:
            nuovo_studente = inserisci_studente()
```

Albo Studenti V

```
            albo_studenti.append(nuovo_studente)
        elif scelta == 2:
            matricola = int(input("Inserire la matricola:"))
            esame = input("Inserisci nome dell'esame")
            voto = int(input("Inserisci il voto"))
            aggiungi_voto(albo_studenti, matricola, esame, voto)
        elif scelta == 3:
            matricola = int(input("Inserisci la matricola:"))
            m = media_carriera(albo_studenti, matricola)
            print("Media voti:", m)
        elif scelta == 4:
            stampa_informazioni(albo_studenti)

        scelta = menu()

main()
```

Albo Studenti VI