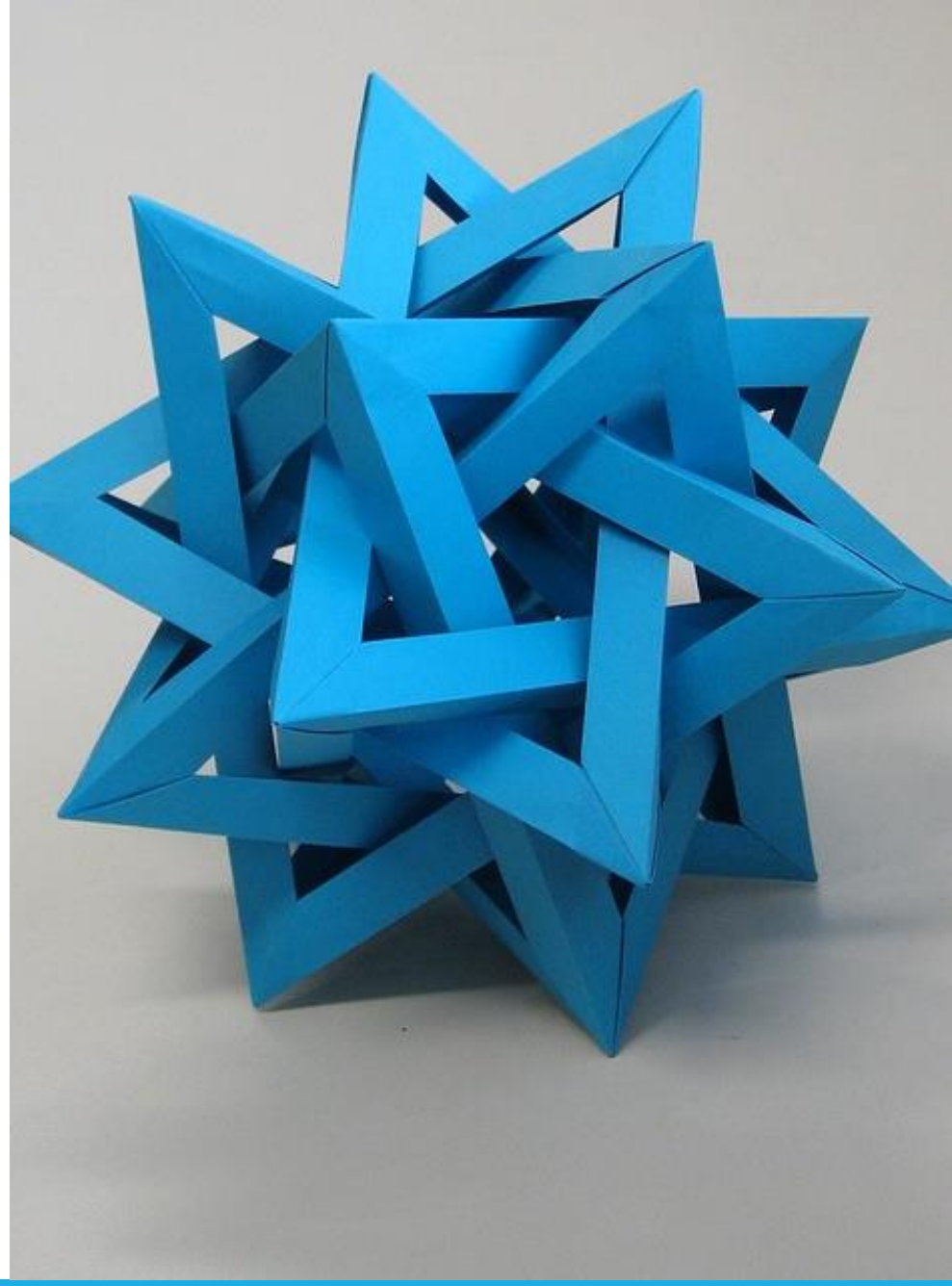
Politecnico di Torino

Dipartimento
di Automatica e Informatica

Laboratorio 09

RIPASSO DI COSTRUTTI CONDIZIONALI, CICLI, LISTE E TABELLE



Esercizio 1

Esercizio 1. Scrivete funzioni che risolvano i problemi seguenti per liste di numeri interi, fornendo un programma di collaudo per ciascuna funzione.

a. Scambiare tra loro il primo e l'ultimo elemento della lista.
b. Far scorrere tutti gli elementi di una posizione "verso destra", spostando l'ultimo elemento nella prima posizione. Ad esempio, la lista `1 4 9 16 25` deve diventare `25 1 4 9 16`.
c. Sostituire con 0 tutti gli elementi pari.
d. Sostituire ciascun elemento, tranne il primo e l'ultimo, con il più grande dei due elementi adiacenti.
e. Eliminare l'elemento centrale della lista se questa ha dimensione dispari, altrimenti eliminare i due elementi centrali.
f. Spostare tutti gli elementi pari all'inizio della lista, preservando però l'ordinamento relativo tra gli elementi, se si esclude la condizione posta.
g. Restituire il secondo valore maggiore della lista.
h. Restituire `True` se e solo se la lista è ordinata in senso crescente.
i. Restituire `True` se e solo se la lista contiene due elementi adiacenti duplicati.
j. Restituire `True` se e solo se la lista contiene elementi duplicati (non necessariamente adiacenti). [P6.4]

Esercizio 1 – *l'impostazione del codice Python*

```
# definizione lista iniziale
NUMBERS = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Definirla come costante, non è necessario chiedere all'utente. Concentratevi sullo sviluppo delle funzioni!

```
# main
def main():
    # collaudo delle funzioni
    data = NUMBERS

    # invoca la prima funzione da collaudare
    funz1(data)
    print('…')
```

Nel main si collaudano le funzioni create.

```
# funzioni
def funz1():
    …

def funz2():
    …
```

Parte principale dell'esercizio: definizione delle funzioni!

```
# invocazione del main
main()
```

Invocazione del main()

Esercizio 1 – *il codice Python*

```python
from random import randint


def main():
    rand_list = [randint(1, 20) for _ in range(10)]  # generate a list of 10 random numbers from 1 to 20
    # note: by convention, we use the variable name '_' when the control variable of a for loop is not needed
    print("The original data for all functions is: ", rand_list)

    # a) Demonstrate swapping the first and last element.
    data = list(rand_list)
    swap_first_last(data)
    print("After swapping first and last: ", data)

    # b) Demonstrate shifting to the right.
    data = list(rand_list)
    shift_right(data)
    print("After shifting right: ", data)

    # c) Demonstrate replacing even elements with zero.
    data = list(rand_list)
    replace_even(data)
    print("After replacing even elements: ", data)

    # d) Demonstrate replacing values with the larger of their neighbors.
    data = list(rand_list)
    replace_neighbors(data)
    print("After replacing with neighbors: ", data)
```

Esercizio 1 – *il codice Python*

```
# e) Demonstrate removing the middle element.
data = list(rand_list)
remove_middle(data)
print("After removing the middle element(s): ", data)

# f) Demonstrate moving even elements to the front of the list.
data = list(rand_list)
even_to_front(data)
print("After moving even elements: ", data)

# g) Demonstrate finding the second largest value.
print("The second largest value is: ", second_largest(rand_list))

# h) Demonstrate testing if the list is in increasing order.
print("The list is in increasing order: ", is_increasing(rand_list))

# i) Demonstrate testing if the list contains adjacent duplicates.
print("The list has adjacent duplicates: ", has_adjacent_duplicate(rand_list))

# j) Demonstrate testing if the list contains duplicates.
print("The list has duplicates: ", has_duplicate(rand_list))
```

Esercizio 1 – *il codice Python*

```python
def swap_first_last(data):
    """
    Swap the first and last element in a list

    :param data: the list of values to process
    """
    if len(data) < 2:
        return
    (data[0], data[-1]) = (data[-1], data[0])

    # Alternate solution (without 'shortcuts')
    # temp = data[0]
    # data[0] = data[len(data) - 1]
    # data[len(data) - 1] = temp
def shift_right(data):
    """
    Shift the elements to the right

    :param data: the list of values to process
    """
    if len(data) == 0:
        return

    last = data[len(data) - 1]
    # Iteration starting from the last element in the list and ending at the second one (i = 1)
    for i in range(len(data) - 1, 0, -1):
        data[i] = data[i - 1]
    data[0] = last

    # shortcut using slices and tuples
    # (data[0], data[1:]) = (data[-1], data[:-1])

    # Alternate solution with pop and insert methods
    # last = data.pop(len(data) - 1)
    # data.insert(0, last)
```

Esercizio 1 – *il codice Python*

```python
def replace_even(data):
    """
    Replace all even elements in the list with 0

    :param data: the list of values to process
    """
    for i in range(0, len(data)):
        if data[i] % 2 == 0:
            data[i] = 0


def replace_neighbors(data):
    """
    Replace each value with the larger of its neighbors

    :param data:
    :return: the list of values to process
    """
    # Make a copy of the list
    old_values = list(data)
    for i in range(1, len(data) - 1):
        data[i] = max(old_values[i - 1], old_values[i + 1])
```

Esercizio 1 – *il codice Python*

```python
def remove_middle(data):
    """
    Remove the middle element or elements from a list

    :param data: the list of values to process
    """
    if len(data) == 0:
        return

    if len(data) % 2 == 1:
        data.pop(len(data) // 2)
    else:
        data.pop(len(data) // 2)
        data.pop(len(data) // 2)


def even_to_front(data):
    """
    Move even elements to the front of the list

    :param data: the list of values to process
    """
    even_pos = 0
    pos = 0
    while pos < len(data):
        if data[pos] % 2 == 0:
            temp = data.pop(pos)
            data.insert(even_pos, temp)
            even_pos = even_pos + 1
        pos = pos + 1
```

Esercizio 1 – *il codice Python*

```python
def second_largest(data):
    """
    Identify the second largest value in a list

    :param data: the list of values to process
    :return: the second largest value in the list
    """

    data.sort(reverse=True)
    second_large = data[1]

    # Alternate solution
    # largest = max(data)
    # second_large = min(data)  # just as an initial guess
    # for value in data:
    #    if value > second_large and value != largest:
    #        second_large = value

    return second_large


def is_increasing(data):
    """
    Determine whether or not the list is in increasing order

    :param data: the list of values to process
    :return: True if the list is in increasing order, False otherwise
    """
    increasing = True
    for i in range(0, len(data) - 1):
        if data[i] > data[i + 1]:
            increasing = False
    return increasing
```

Esercizio 1 – *il codice Python*

```python
def has_adjacent_duplicate(data):
    """
    Determine if the list contains adjacent duplicate elements

    :param data: the list of values to process
    :return: True if the list contains adjacent duplicates, False otherwise
    """
    has_dup = False
    for i in range(0, len(data) - 1):
        if data[i] == data[i + 1]:
            has_dup = True
    return has_dup


def has_duplicate(data):
    """
    Determine if the list contains duplicate elements

    :param data: the list of values to process
    :return: True if the list contains duplicates, False otherwise
    """
    has_dup = False
    for i in range(0, len(data) - 1):
        for j in range(i + 1, len(data)):  # avoid comparing data[i] with itself!!
            if data[i] == data[j]:
                has_dup = True
    return has_dup


# Call the main function.
main()
```

Esercizio 2

Esercizio 2. Lo schema dei posti a teatro è una tabella con i prezzi dei biglietti per ciascun posto, come questa. [P6.27]

```
10  10  10  10  10  10  10  10  10  10
10  10  10  10  10  10  10  10  10  10
10  10  10  10  10  10  10  10  10  10
10  10  20  20  20  20  20  20  10  10
10  10  20  20  20  20  20  20  10  10
10  10  20  20  20  20  20  20  10  10
20  20  30  30  40  40  30  30  20  20
20  30  30  40  50  50  40  30  30  20
30  40  50  50  50  50  50  50  40  30
```

Scrivete un programma che gestisca un menù che chieda all'utente di scegliere un posto, un prezzo o l'uscita dal programma. Contrassegnate con un prezzo uguale a 0 i posti già venduti. Quando l'utente specifica un posto, accertatevi che sia libero e che le coordinate siano all'interno della tabella. Quando, invece, specifica un prezzo, assegnategli un posto qualsiasi tra quelli disponibili a quel prezzo.

Esercizio 2 – *il codice Python*

```
def main():

    # Set up the price chart.
    price_chart = [[10, 10, 10, 10, 10, 10, 10, 10, 10, 10],
                   [10, 10, 10, 10, 10, 10, 10, 10, 10, 10],
                   [10, 10, 10, 10, 10, 10, 10, 10, 10, 10],
                   [10, 10, 20, 20, 20, 20, 20, 20, 10, 10],
                   [10, 10, 20, 20, 20, 20, 20, 20, 10, 10],
                   [10, 10, 20, 20, 20, 20, 20, 20, 10, 10],
                   [20, 20, 30, 30, 40, 40, 30, 30, 20, 20],
                   [20, 30, 30, 40, 50, 50, 40, 30, 30, 20],
                   [30, 40, 50, 50, 50, 50, 50, 50, 40, 30]]

    # Read the user's choice.
    display_map(price_chart)
    choice = input("Pick a (S)eat, Pick a (P)rice or (Q)uit? ").upper()
    while choice != "Q":
        # Handle a seat selection of by seat position.
        if choice == "S":
            row = int(input("Enter the row: "))
            col = int(input("Enter the column: "))
            # Verify that it is still available.
            if 0 <= row < len(price_chart) and 0 <= col < len(price_chart[row]) and price_chart[row][col] != 0:
                print("Sold, for %d dollars!" % price_chart[row][col])
                price_chart[row][col] = 0
            else:
                print("Sorry, that seat is not available.")
```

Esercizio 2 – *il codice Python*

```
# Handle a seat selection by price.
elif choice == "P":
    price = float(input("How much do you want to spend? "))
    found = False

    # Search for a seat with the desired price.
    for row in range(len(price_chart)):
        for col in range(len(price_chart[row])):
            if not found and price_chart[row][col] == price:
                print("You have seat (%d, %d)." % (row, col))
                price_chart[row][col] = 0
                found = True
    if not found:
        print("Sorry, no tickets available at that price.")
else:
    print("That wasn't a valid option.")

# Read the next choice.
display_map(price_chart)
choice = input("Pick a (S)eat, Pick a (P)rice or (Q)uit? ").upper()
```

Esercizio 2 – *il codice Python*

```
def display_map(price_chart):
    # Display the seating map.
    for row in range(len(price_chart)):
        for col in range(len(price_chart[row])):
            print("%3d" % price_chart[row][col], end="")
        print()

main()
```

Esercizio 3

Esercizio 3. Nei lunghi viaggi in auto, per ingannare il tempo, si può fare il gioco delle "parole concatenate". Il primo giocatore dice una parola iniziale, poi a turno ciascun giocatore dovrà dire una nuova parola (ossia mai detta prima) la cui *sillaba iniziale* sia uguale alla *sillaba finale* della parola precedente. (NOTA: Per semplicità, ipotizziamo che tutte le sillabe siano lunghe esattamente 2 caratteri, quindi per "figli" la 'sillaba' finale sarà "li" e non "gli").
Ad esempio: gatto - torino - notte - tela - lana …
Scrivere un programma per permettere di gestire una o più partite del gioco.
Ciascuna partita termina quando un giocatore inserisce una parola già detta nella stessa partita, quando inserisce una parola non correttamente concatenata, oppure quando non riesce a proseguire (per abbandonare, inserisce *).

Esercizio 3 – *il codice Python*

```python
def main():
    stop = False
    while not stop:
        partita()
        yn = input("Continue (y/n)? ")
        if yn.strip().lower().startswith("n"):
            stop = True
```

```python
def partita():
    words = []

    last = input('Insert the starting word: ').strip().lower()

    valid_game = True
    while valid_game:
        new = input('Insert a word: ').strip().lower()
        if new == '*':
            print('You abandon!')
            valid_game = False
        elif len(new) < 2 or last[-2:] != new[:2]:
            print('Invalid word!')
            valid_game = False
        elif new in words:
            print('Repeated word!')
            valid_game = False
        else:
            last = new
            words.append(new)


main()
```

Esercizio 4

Esercizio 4. Un supermercato vuole ricompensare il proprio miglior cliente del giorno, mostrandone il nome su uno schermo all'interno del negozio. A questo scopo, vengono memorizzati in una lista (customers) i nomi di tutti i clienti del giorno e, in un'altra lista (sales), il corrispondente importo della spesa effettuata.
Scrivete la funzione `nameOfBestCustomer(sales, customers)` che restituisca il nome del cliente che ha speso la cifra più alta.
Poi, scrivete un programma che chieda al cassiere di digitare tutti gli importi spesi e i nomi dei relativi clienti, aggiungendoli via via a due liste distinte, per poi invocare la funzione che avete progettato e visualizzare il risultato. Usate il prezzo 0 come sentinella. [P6.33]

Esercizio 4 – *il codice Python*

```python
def main():
    sales = []
    names = []

    # Read the names and sales amounts, storing them in two lists.
    amount = float(input("Enter the sale amount: "))
    while amount != 0:
        sales.append(amount)

        name = input("Enter the customer's name: ")
        names.append(name)

        amount = float(input("Enter the sale amount: "))

    # Compute and display the name of the best customer.
    best_name = name_of_best_customer(sales, names)
    print("The best customer was:", best_name)
```

```python
def name_of_best_customer(sales, customers):
    """
    Identify the name of the best customer

    :param sales: the list of sale amounts
    :param customers: the names of the customers
    :return: the name of the customer with the highest sale amount
    """
    # Find the largest sale.
    largest = max(sales)

    # Display the name of the best customer.
    # If more than one has the same sales, return the 1st one
    for i in range(len(sales)):
        if sales[i] == largest:
            return customers[i]

    # Alternate solution
    # return customers[sales.index(max(sales))]


# Call the main function.
main()
```

Esercizio 5

Esercizio 5. Scrivere un programma che acquisisca dall'utente un elenco di numeri interi positivi, scritti su un'unica riga e separati dal carattere ':'.

Esempio: `3:12:21:8:4:7`

Il programma dovrà stampare, sempre nello stesso formato:

- gli stessi numeri, ad eccezione del minimo e del massimo (es. `12:8:4:7`)
- i soli numeri pari (es. `12:8:4`)
- i soli numeri di 2 cifre (es. `12:21`)

Si suggerisce di lavorare costruendo una lista di numeri interi.

Esercizio 5 – *il codice Python*

```
sequence = input('Insert a list of numbers (separated by :) ')

numbers_as_strings = sequence.split(':')

numbers = []
for s in numbers_as_strings:
    n = int(s)
    numbers.append(n)

print(numbers)

# all except min and max
numbers_without_min_max = list(numbers)
numbers_without_min_max.remove(min(numbers_without_min_max))
numbers_without_min_max.remove(max(numbers_without_min_max))
for i in range(len(numbers_without_min_max)):
    numbers_without_min_max[i] = str(numbers_without_min_max[i])
print(':'.join(numbers_without_min_max))
```

```
# only even numbers
numbers_even = []
for n in numbers:
    if n % 2 == 0:
        numbers_even.append(n)
for i in range(len(numbers_even)):
    numbers_even[i] = str(numbers_even[i])
print(':'.join(numbers_even))

# only 2-digit numbers
# may work directly on the list of strings
numbers_2digits_as_string = []
for s in numbers_as_strings:
    if len(s) == 2:
        numbers_2digits_as_string.append(s)
print(':'.join(numbers_2digits_as_string))
```

Esercizio 6

Esercizio 6. Scrivere un programma che generi una “spirale” di numeri interi inserita in una matrice di NxN elementi, con N inserito dall’utente. I numeri saranno i valori compresi tra 1 e N^2.
Suggerimento: costruire prima la spirale numerica in una tabella NxN, e solo successivamente stamparla.
Ad esempio, se N=4, il programma dovrà stampare:

1	2	3	4
12	13	14	5
11	16	15	6
10	9	8	7

⊙	→	→	↴
↱	→	↴	↓
↑	⊗	↲	↓
↑	←	←	↲

Esercizio 6 – *il codice Python*

```python
from pprint import pprint


def main():
    n = int(input('Size of the spiral: '))

    # build an nxn matrix, filled with 0
    matrix = []
    for row in range(n):
        matrix.append([0] * n)

    # the spiral can be seen as a sequence of "rings".
    # - if N is even, we have N/2 rings
    # - if N is odd, we have N/2 rings, plus a "central" value
    # ring=0 -> the external ring

    start = 1
    for ring in range(n // 2):
        start = build_ring(matrix, ring, start)
    if n % 2 == 1:
        matrix[n // 2][n // 2] = start

    pprint(matrix)
```

```python
def build_ring(mat, ring, start):
    n = len(mat)
    # the ring starts at cell mat[ring][ring]
    # go right from [ring][ring]->[ring][n-1-ring]
    for col in range(ring, n - ring):
        mat[ring][col] = start
        start = start + 1
    # go down, on the same column [ring+1][n-1-ring]->[n-1-ring][n-1-ring]
    for row in range(ring + 1, n - ring):
        mat[row][n - ring - 1] = start
        start = start + 1
    # go left, on the same row [n-1-ring][n-1-ring-1]->[n-1-ring][ring]
    for col in range(n - ring - 2, ring - 1, -1):
        mat[n - ring - 1][col] = start
        start = start + 1
    # go up, on the same column [n-1-ring-1][ring] -> [ring+1][ring]
    for row in range(n - ring - 2, ring, -1):
        mat[row][ring] = start
        start = start + 1

    return start


main()
```