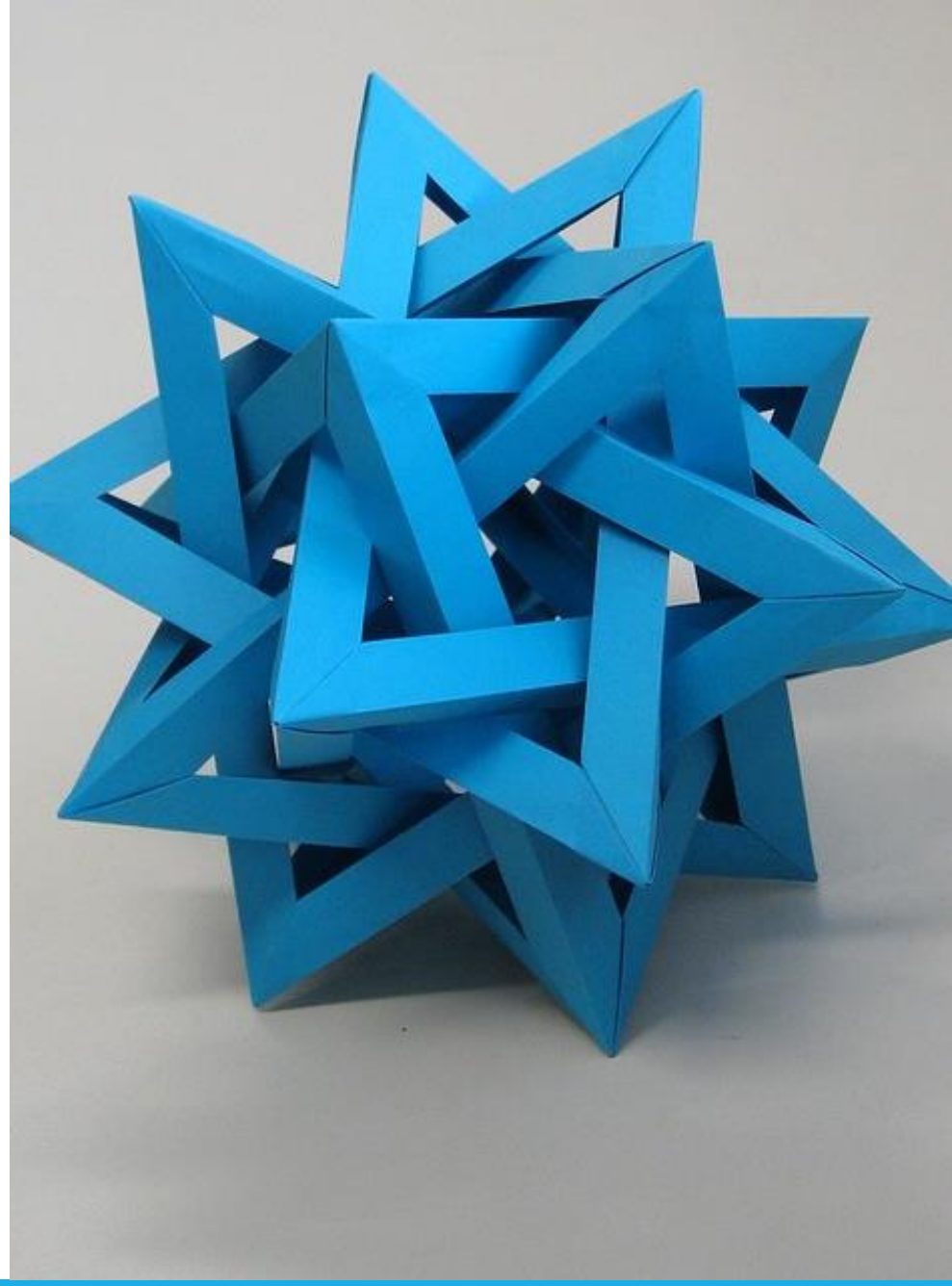
Politecnico di Torino
Dipartimento di Automatica e Informatica

Laboratorio 07

LISTE



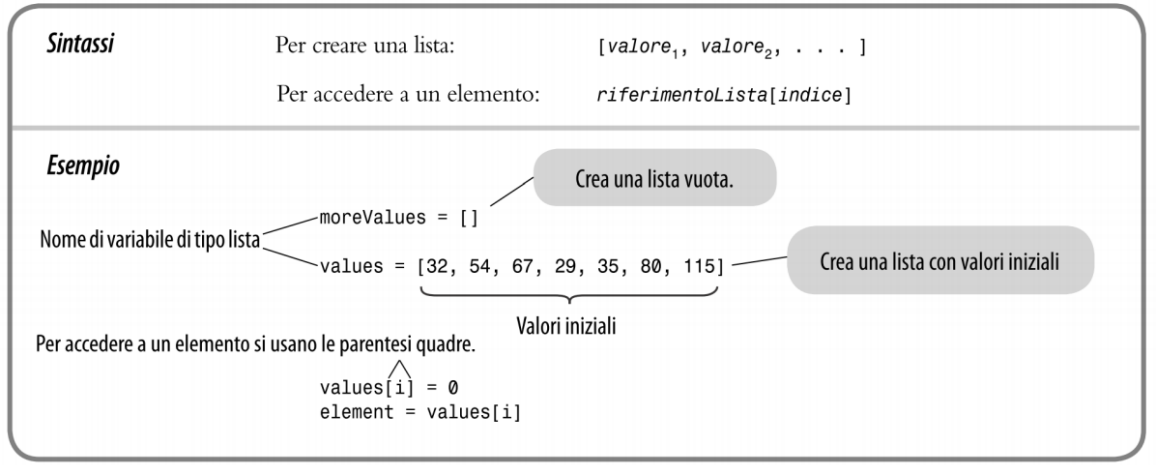
Dalla teoria…

Cos'è una lista?

- Una lista è una struttura dati versatile e dinamica, che contiene un numero variabile di elementi, di qualunque tipo, a cui si può avere accesso tramite la loro posizione (indice)

Creare una lista

- Assegnare ad una variabile una nuova lista con l'operatore di indicizzazione [ ]



Dalla teoria…

Operazioni predefinite per liste

- Il metodo `.insert()` inserisce un elemento in una data posizione della lista
- L'operatore `in` verifica se un elemento sia contenuto in una lista
- Il metodo `.pop()` rimuove un elemento da una data posizione della lista
- Il metodo `.remove()` toglie un elemento da una lista sulla base del suo valore
- Due liste possono essere concatenate usando l'operatore (`+`)
- La funzione `list()` crea la copia di una lista esistente
- L'operatore di slice (`:`) estrae una sottolista o una sottostringa

Esercizio 1

Esercizio 1. Scrivete un programma che inizializzi una lista con dieci numeri interi casuali tra 1 e 100 e, poi, visualizzi quattro righe di informazioni, contenenti:

a. Tutti gli elementi di *indice* pari.
b. Tutti gli elementi di *valore* pari.
c. Tutti gli elementi in ordine inverso.
d. Il primo e l'ultimo elemento. [P6.1]

Esercizio 1.a – *il codice Python*

```python
from random import randint

# Define constants.
NUM_INTEGERS = 10
MAX_VALUE = 100

# Generate the random integers from 1 to MAX_VALUE
values = []
for i in range(0, NUM_INTEGERS):
    values.append(randint(1, MAX_VALUE))

#
# a.
#

# Display the elements at even index
print("Elements at even index: ", end="")
for i in range(0, len(values), 2):
    print(values[i], end=" ")
print()

# Alternative (using a slice the list)
print("Elements at even index: ", end="")
for val in values[::2]:
    print(val, end=" ")
print()
```

Esercizio 1.b – *il codice Python*

```python
# Display the even elements.
# Alternative 1: use a for..range loop
print("The even elements are: ", end="")
for i in range(len(values)):
    if values[i] % 2 == 0:
        print(values[i], end=" ")
print()

# Alternative 2: usa a for..in statement
print("The even elements are: ", end="")
for num in values:
    if num % 2 == 0:
        print(num, end=" ")
print()
```

Esercizio 1.c e 1.d – *il codice Python*

```
# Display the elements in reverse order.
print("In reverse order:")
for i in range(len(values) - 1, -1, -1):
    print(values[i], end=" ")
print()

# Alternative (reversing the list with a slice)
print("In reverse order:")
for val in values[::-1]:
    print(values[i], end=" ")
print()

#
# d.
#

# Display the first and last element.
print("First and last:", values[0], values[len(values) - 1])
print("First and last:", values[0], values[-1])  # shortcut: negative index
```

Esercizio 2

Esercizio 2. Scrivete un programma che acquisisca da tastiera una sequenza di numeri interi (terminata da una riga vuota), calcoli la *somma alternata* degli elementi di una lista. Ad esempio, se il programma legge i dati 1 4 9 16 9 7 4 9 11, deve calcolare e visualizzare 1 – 4 + 9 – 16 + 9 – 7 + 4 – 9 + 11 = –2. [P6.8]

Esercizio 2 – *il codice Python*

```python
def main():
    # Read values from the user and store them in a list.
    values = []
    input_str = input("Enter values (blank line to quit): ")
    while input_str != "":
        values.append(float(input_str))
        input_str = input("Enter values (blank line to quit): ")

    # Display the alternating sum of the entered values.
    print("The alternating sum is", alternating_sum(values))

    # Shortcut: use the "sum" function over the slice composed of even-position
    # and odd-position elements, separately.
    print('Another way of computing the alternating sum:', sum(values[0::2]) - sum(values[1::2]))
```

```python
def alternating_sum(data):
    """
    Compute the alternating sum of a list of values.

    :param data: the list of values to process
    :return: the alternating sum of the provided values
    """
    total = 0
    for i in range(len(data)):
        if i % 2 == 0:
            total = total + data[i]
        else:
            total = total - data[i]

    return total


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

Esercizio 3

Esercizio 3. Scrivete la funzione `def sameSet(a, b)` che verifichi se due liste contengono gli stessi elementi, indipendentemente dall'ordine e ignorando la presenza di duplicati. Ad esempio, le due liste `1 4 9 16 9 7 4 9 11` e `11 11 7 9 16 4 1` devono essere considerate uguali. La funzione non deve modificare le liste ricevute come parametri. [P6.12]

Esercizio 3

```
# Define constants.
LIST_1 = [1, 4, 9, 16, 9, 7, 4, 9, 11]
LIST_2 = [11, 11, 7, 9, 16, 4, 1]


def main():
    print("List 1 is", LIST_1)
    print("List 2 is", LIST_2)
    print("The lists contain the same elements: ", same_set(LIST_1, LIST_2))


def same_set(l1, l2):
    """
    Determine if two lists contain the same elements in any order, ignoring duplicates

    :param l1: the first list to consider
    :param l2: the second list to consider
    :return: True if the lists contain the same elements, False otherwise
    """
    for value in l1:
        if value not in l2:
            return False

    for value in l2:
        if value not in l1:
            return False

    return True


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

Esercizio 4

Esercizio 4. Scrivete un programma che generi una sequenza di 20 valori casuali compresi tra 0 e 99, poi visualizzi la sequenza generata, la ordini e la visualizzi di nuovo, ordinata. Usate il metodo `sort`. [P6.17]

Esercizio 4 – *il codice Python*

```
Generate a random sequence of values, print it, sort it, and print it again.
"""
from random import randint

# Define constants.
NUM_VALUES = 20
MIN_VALUE = 0
MAX_VALUE = 99

# Generate the list of random values.
values = []
for i in range(NUM_VALUES):
    values.append(randint(MIN_VALUE, MAX_VALUE))

# Display the original values.
print("The values are:")
print(values)

# Sort the values and display them again.
values.sort()
print("The values in sorted order are:")
print(values)
```

Esercizio 5

Esercizio 5. Leggere una sequenza di numeri interi conclusa da una riga vuota. Stampare la posizione dei massimi locali (numeri maggiori sia del valore precedente che di quello successivo) se ce ne sono, altrimenti stampare il messaggio “Non ci sono massimi locali”.

Esercizio 5 – *il codice Python*

```
Find the local maxima of a series of numbers (read in input, terminated with an empty line)
"""

numbers = []
data = input("Number: ")
while data != '':
    numbers.append(int(data))
    data = input("Number: ")

pos_local_max = []

for i in range(1, len(numbers) - 1):
    if numbers[i - 1] < numbers[i] > numbers[i + 1]:
        pos_local_max.append(i)

if len(pos_local_max) > 0:
    print("Local maxima are in positions: ", pos_local_max)
else:
    print("There are no local maxima")
```

Esercizio 6

Esercizio 6. Scrivete la funzione `sum_without_smallest` che calcoli la somma di tutti i valori di una lista, escludendo il valore minimo. [P6.6]

Esercizio 6 – *il codice Python*

```
Sum values without including the smallest.
"""

# Define constants.
ONE_TEN = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]


def main():
    print("The sum without the smallest is", sum_without_smallest(ONE_TEN))


def sum_without_smallest(data):
    """
    Sum all elements in the list, excluding the smallest.

    :param data: the list of values to process
    :return: the sum of all of the values, excluding the smallest
    """
    smallest = min(data)
    total = sum(data)

    return total - smallest


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

Esercizio 7

Esercizio 7. Spesso i valori raccolti durante un esperimento vanno corretti, per togliere parte del rumore di misura. Un approccio semplice a questo problema prevede di sostituire, in una lista, ciascun valore con la media tra il valore stesso e i due valori adiacenti (o un unico adiacente se il valore in esame si trova a una delle due estremità della lista). Realizzate un programma che svolga tale operazione, senza creare un'altra lista. [P6.36]

Esercizio 7 – *il codice Python*

```python
def main():
    values = [1, 2, 3, 5, 3, 1, 4]
    values_copy = list(values)

    # Display the original values.
    print("The original values are:", values)

    # Smooth the values and display the result.
    smooth(values)
    print("The smoothed values are:", values)

    smoothed_alternative = smooth_on_new_list(values_copy)
    print("The smoothed values are:", smoothed_alternative)
```

```python
def smooth(data):
    """
    Smooth values in a list by averaging the value of each element with its neighbor(s).

    :param data: data the list of values to smooth
    :return: None (it modifies the list in-place)
    """
    if len(data) <= 1:
        return

    # Handle the first element in the list.
    old_left = data[0]
    data[0] = (data[0] + data[1]) / 2

    # Handle all values except for the first and last.
    for i in range(1, len(data) - 1):
        current = data[i]
        data[i] = (old_left + data[i] + data[i + 1]) / 3
        old_left = current

    # Handle the last element in the list.
    data[len(data) - 1] = (old_left + data[len(data) - 1]) / 2
```

Esercizio 7 – *il codice Python*

```python
def smooth_on_new_list(data):
    """
    This version creates a NEW list instead of modifying the current one, and returns it
    """
    result = []

    # element 0: average of elements 0 and 1
    result.append((data[0] + data[1]) / 2)

    # elements i = 1...len-1: average of 3 elements
    for i in range(1, len(data) - 1):
        value = (data[i - 1] + data[i] + data[i + 1]) / 3
        result.append(value)

    # last element: average of 2
    result.append((data[-2] + data[-1]) / 2)

    return result


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

Esercizio 8

Esercizio 8. Leggere una sequenza di numeri interi conclusa da una riga vuota. Individuare i due massimi locali (numeri maggiori sia del valore precedente che di quello successivo) *più vicini fra loro* come posizione e stampare la loro posizione.

Esercizio 8 – *il codice Python*

```python
numbers = []
data = input("Number: ")
while data != '':
    numbers.append(int(data))
    data = input("Number: ")

pos_local_max = []

for i in range(1, len(numbers) - 1):
    if numbers[i - 1] < numbers[i] > numbers[i + 1]:
        pos_local_max.append(i)

if len(pos_local_max) > 1:
    min_distance = pos_local_max[1] - pos_local_max[0]
    min_position = 1
    for i in range(2, len(pos_local_max)):
        if pos_local_max[i] - pos_local_max[i - 1] < min_distance:
            min_distance = pos_local_max[i] - pos_local_max[i - 1]
            min_position = i
    print("The closest local maxima are in positions: ", pos_local_max[min_position - 1], pos_local_max[min_position])
else:
    print("There are less than 2 local maxima")
```