

Corso di STATISTICA, INFORMATICA, ELABORAZIONE DELLE INFORMAZIONI

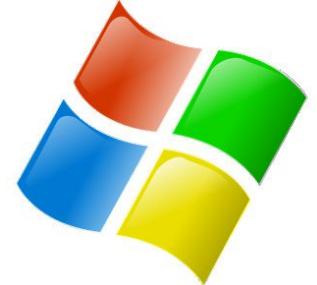
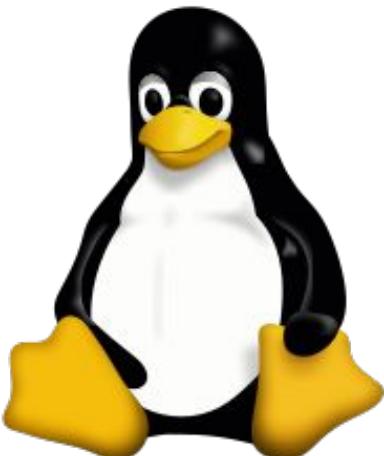
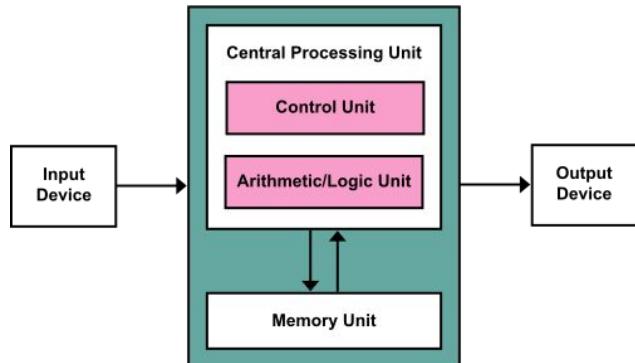
*Modulo di Sistemi di Elaborazione delle
Informazioni*



**UNIVERSITÀ DEGLI STUDI
DELLA BASILICATA**

Liste e Tuple

Docente:
**Domenico Daniele
Bloisi**



Domenico Daniele Bloisi

- Professore Associato

Dipartimento di Matematica, Informatica
ed Economia

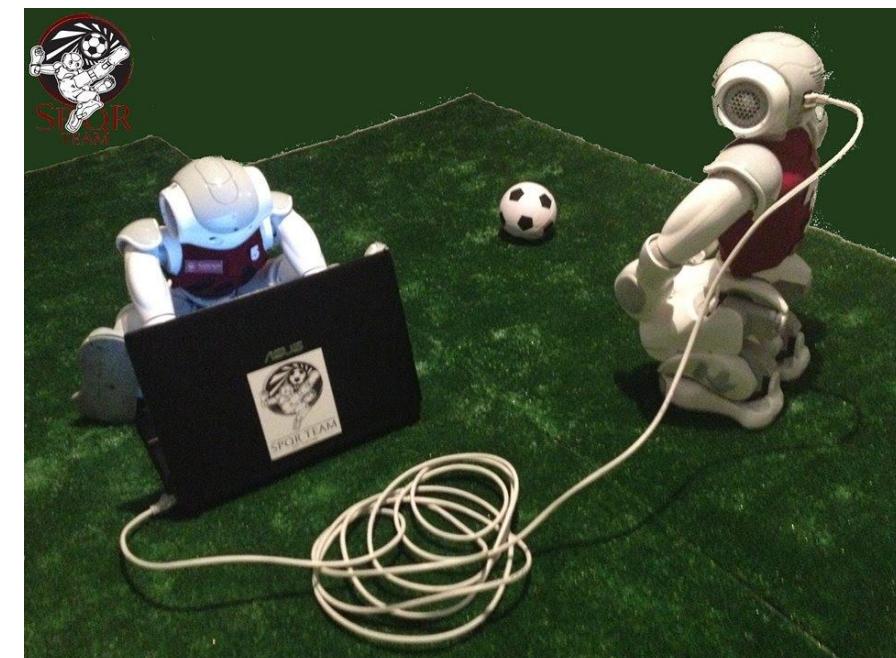
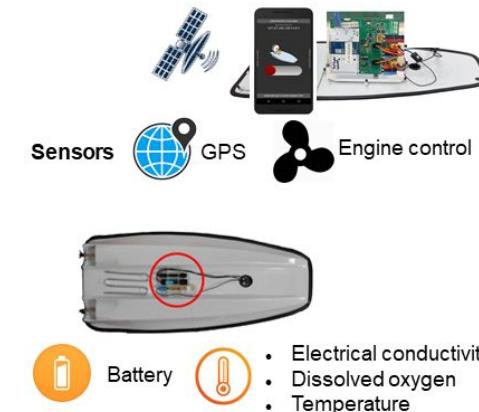
Università degli studi della Basilicata

<http://web.unibas.it/bloisi>

- SPQR Robot Soccer Team

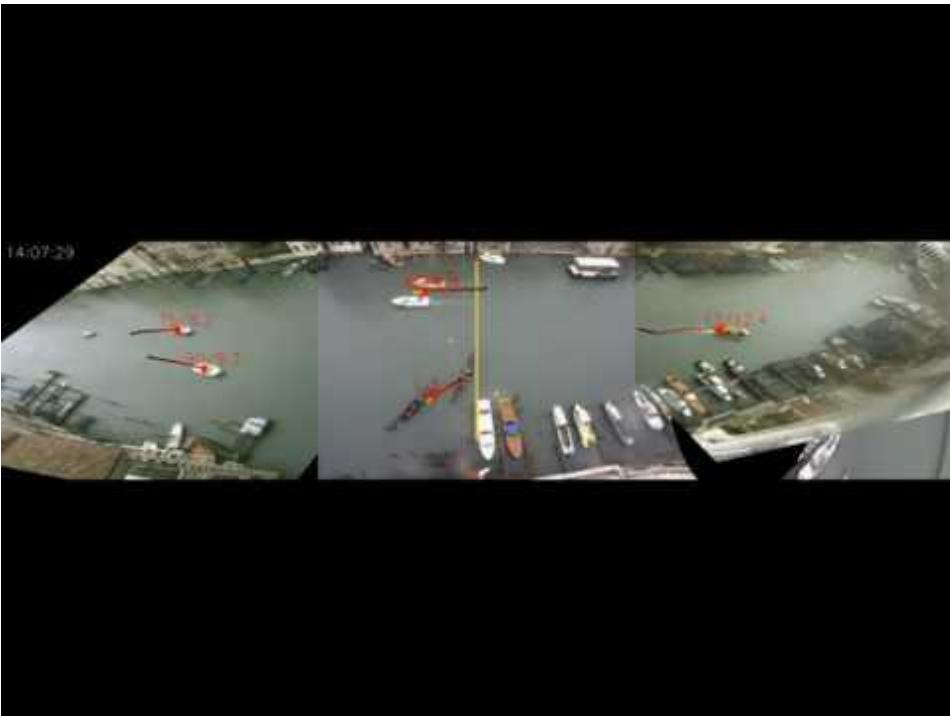
Dipartimento di Informatica, Automatica
e Gestionale Università degli studi di
Roma “La Sapienza”

<http://spqr.diag.uniroma1.it>



Interessi di ricerca

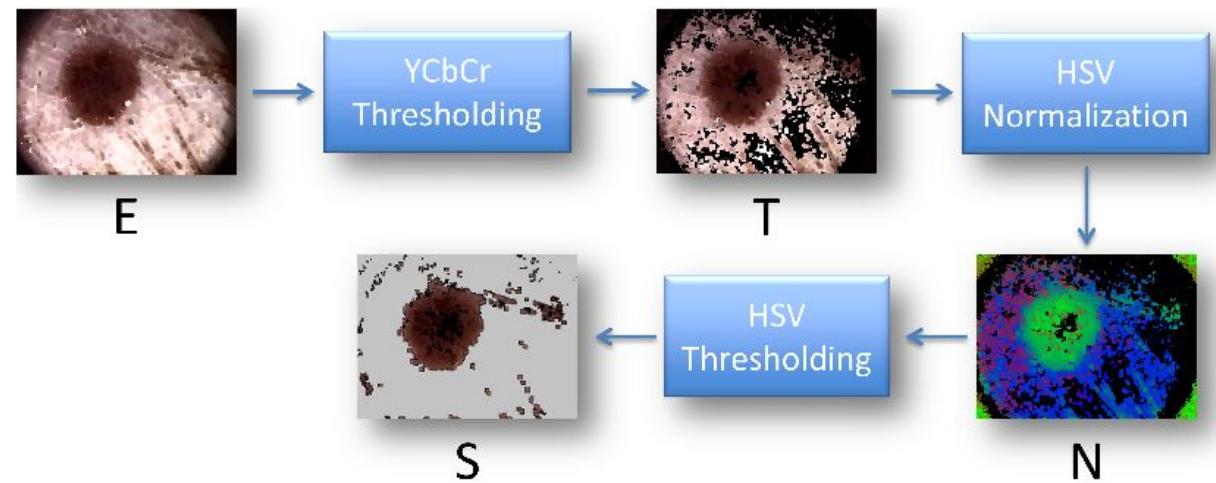
- Intelligent surveillance
- Robot vision
- Medical image analysis



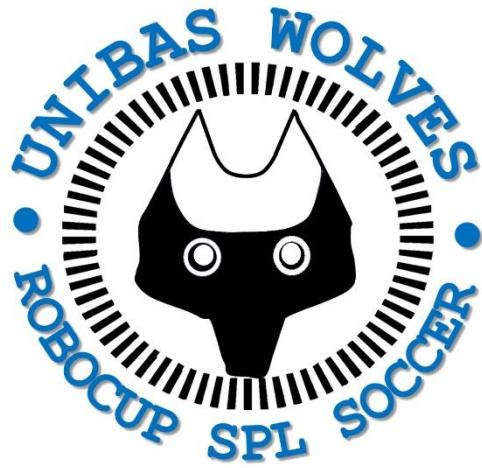
https://youtu.be/9a70Ucgbi_U



<https://youtu.be/2KHNZX7UIWQ>



UNIBAS Wolves <https://sites.google.com/unibas.it/wolves>



- UNIBAS WOLVES is the robot soccer team of the University of Basilicata. Established in 2019, it is focussed on developing software for NAO soccer robots participating in RoboCup competitions.
- UNIBAS WOLVES team is twinned with SPQR Team at Sapienza University of Rome



<https://youtu.be/ji0OmkaWh20>

Informazioni sul corso

Il corso di STATISTICA, INFORMATICA, ELABORAZIONE DELLE INFORMAZIONI

- include 3 moduli:
 - SISTEMI DI ELABORAZIONE DELLE INFORMAZIONI
(il martedì - docente: Domenico Bloisi)
 - INFORMATICA
(il mercoledì - docente: Enzo Veltri)
 - PROBABILITÀ E STATISTICA MATEMATICA
(il giovedì - docente: Antonella Iuliano)
- Periodo: I semestre ottobre 2022 – gennaio 2023

Ricevimento Bloisi

- In presenza, durante il periodo delle lezioni:
Lunedì dalle 17:00 alle 18:00
presso Edificio 3D, II piano, stanza 15
Si invitano gli studenti a controllare regolarmente la bacheca degli avvisi per eventuali variazioni
- Tramite google Meet e al di fuori del periodo delle lezioni:
da concordare con il docente tramite email

Per prenotare un appuntamento inviare
una email a
domenico.bloisi@unibas.it



Recap



recap-file.ipynb ☆

File Modifica Visualizza Inserisci Runtime Strumenti Guida Tutte le modifiche sono state salvate



+ Codice + Testo



[1] `ls`

0 s



{x} sample_data



0 s `ls sample_data`

▶

anscombe.json	mnist_test.csv
california_housing_test.csv	mnist_train_small.csv
california_housing_train.csv	README.md

```
f = open('sample_data/README.md', 'r')
data = f.read()
print(data)
f.close()
```

- ↳ This directory includes a few sample datasets to get you started.

- * `california_housing_data*.csv` is California housing data from the 1990 US Census; more information is available at:
<https://developers.google.com/machine-learning/crash-course/california-housing-data-description>
- * `mnist_*.csv` is a small sample of the [MNIST database](https://en.wikipedia.org/wiki/MNIST_database), which is described at: <http://yann.lecun.com/exdb/mnist/>
- * `anscombe.json` contains a copy of [Anscombe's quartet](https://en.wikipedia.org/wiki/Anscombe%27s_quartet); it was originally described in

Anscombe, F. J. (1973). 'Graphs in Statistical Analysis'. American Statistician. 27 (1): 17-21. JSTOR 2682899.

and our copy was prepared by the
[vega_datasets library](https://github.com/altair-viz/vega_datasets/blob/4f67bdaad10f45e3549984e17e)



▶ with open('sample_data/README.md', 'r') as f2:
{x} riga = f2.readline()
 print(riga)



⇨ This directory includes a few sample datasets to get you started.

✓ [13] print(f2.closed)
0 s

True

✓
0 s



```
with open('data.txt', 'w') as f3:  
    da_scrivere = 'dati che saranno scritti nel file'  
    f3.write(da_scrivere)
```

The screenshot shows a Python development environment with a script running and its output displayed.

Left Sidebar:

- File:** Shows a tree view with a folder named "sample_data" containing a file named "data.txt".
- Search:** A magnifying glass icon.
- Code Editor:** A toolbar with icons for file operations (New, Open, Save, etc.) and a search bar.

Central Area:

- Code View:** A code editor window titled "+ Codice" showing the following Python code:

```
with open('data.txt', 'w') as f3:
    da_scrivere = 'dati che saranno scritti nel file'
    f3.write(da_scrivere)
```
- Output View:** A terminal window titled "data.txt" showing the output:

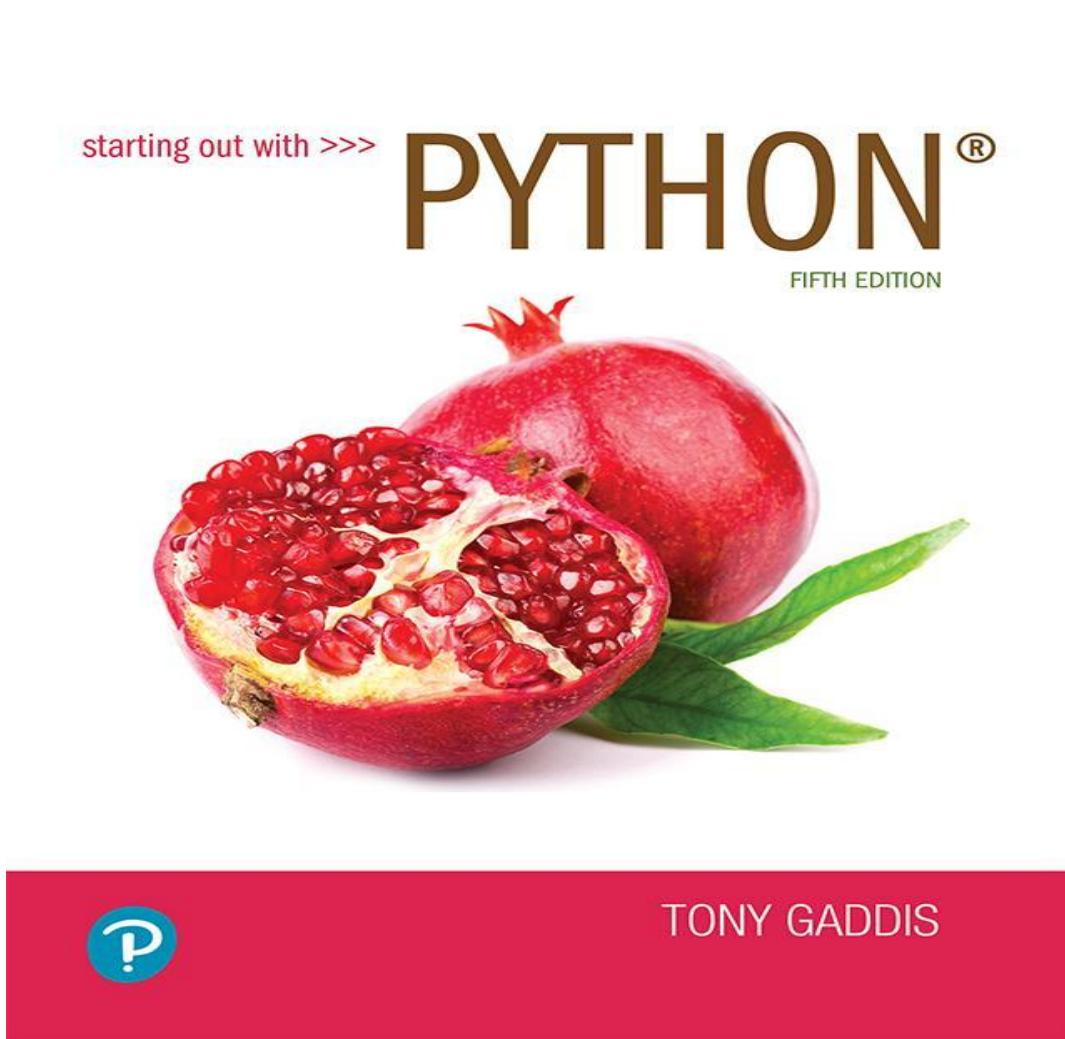
```
1 dati che saranno scritti nel file
```

Top Right:

- Performance indicators: RAM (green checkmark) and Disco (grey).
- A settings gear icon.
- A pen icon for editing.

Starting out with Python

Fifth Edition



Chapter 7

Lists and Tuples

Topics (1 of 2)

- Sequences
- Introduction to Lists
- List Slicing
- Finding Items in Lists with the in Operator
- List Methods and Useful Built-in Functions

Topics (2 of 2)

- Copying Lists
- Processing Lists
- List Comprehensions
- Two-Dimensional Lists
- Tuples
- Plotting List Data with the `matplotlib` Package

Sequences

- Sequence: an object that contains multiple items of data
 - The items are stored in sequence one after another
- Python provides different types of sequences, including lists and tuples
 - The difference between these is that **a list is mutable** and **a tuple is immutable**

Introduction to Lists (1 of 2)

- List: an object that contains multiple data items
 - Element: An item in a list
 - Format: *list* = [item1, item2, etc.]
 - Can hold items of different types
- `print` function can be used to display an entire list
- `list()` function can convert certain types of objects to lists

Esempi List



```
even_numbers = [2, 4, 6, 8, 10]  
  
names = ["Molly", "Steven", "Will", "Alicia", "Adriana"]  
  
info = ["Alicia", 27, 1550.87]
```

Introduction to Lists (2 of 2)



Figure 7-1 A list of integers



Figure 7-2 A list of strings



Figure 7-3 A list holding different types

```
[2] numbers = [5, 10, 15, 20]
    print(numbers)
```

[5, 10, 15, 20]



```
numbers2 = list(range(5, 21, 5))
print(numbers2)
```

[5, 10, 15, 20]

The Repetition Operator and Iterating over a List

- Repetition operator: makes multiple copies of a list and joins them together
 - The * symbol is a repetition operator when applied to a sequence and an integer
 - Sequence is left operand, number is right
 - General format: *list* * *n*
- You can iterate over a list using a `for` loop
 - Format: `for x in list:`

```
[7] numbers = [1, 2, 3]
     print(numbers)
     new_numbers = numbers * 3
     print(new_numbers)
```

```
[1, 2, 3]
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```



```
for i in new_numbers:
    print(i)
```

```
1
2
3
1
2
3
1
2
3
```

Indexing

- Index: a number specifying the position of an element in a list
 - Enables access to individual element in list
 - Index of first element in the list is 0, second element is 1, and n'th element is n-1
 - Negative indexes identify positions relative to the end of the list
 - The index -1 identifies the last element, -2 identifies the next to last element, etc.



```
my_list = [10, 20, 30, 40]

print(my_list[1])

print(my_list[-2])

print(my_list[5])
```

20
30

```
-----
IndexError                                Traceback (most recent call last)
<ipython-input-11-623b113a715a> in <module>
      5 print(my_list[-2])
      6
----> 7 print(my_list[5])
      8
```

IndexError: list index out of range

SEARCH STACK OVERFLOW

The `len` function

- An `IndexError` exception is raised if an invalid index is used
- `len` function: returns the length of a sequence such as a list
 - Example: `size = len(my_list)`
 - Returns the number of elements in the list, so the index of last element is `len(list) - 1`
 - Can be used to prevent an `IndexError` exception when iterating over a list with a loop

```
▶ my_list = [10, 20, 30, 40, 50]
    size = len(my_list)
    print(size)
```

```
⇨ 5
```

```
▶ for i in range(len(my_list)):
    print(my_list[i])
```

```
10
20
30
40
50
```

Lists Are Mutable

- Mutable sequence: the items in the sequence can be changed
 - Lists are mutable, and so their elements can be changed
- An expression such as
- `list[1] = new_value` can be used to assign a new value to a list element
 - Must use a valid index to prevent raising of an `IndexError` exception

```
[16] numbers = [1, 2, 3, 4, 5]
      print(numbers)
      numbers[0] = 99
      print(numbers)
```

```
[1, 2, 3, 4, 5]
[99, 2, 3, 4, 5]
```



```
numbers[5] = 99
print(numbers)
```



```
-----
IndexError                                Traceback (most recent call last)
<ipython-input-17-3cda4899c1ce> in <module>
----> 1 numbers[5] = 99
      2 print(numbers)
```

IndexError: list assignment index out of range

SEARCH STACK OVERFLOW

Concatenating Lists

- Concatenate: join two things together
- The + operator can be used to concatenate two lists
 - Cannot concatenate a list with another data type, such as a number
- The += augmented assignment operator can also be used to concatenate lists



```
rettili = ["serpente", "lucertola", "coccodrillo", "tartaruga"]
pesci = ["merluzzo", "anguilla", "scorfano", "tonno"]

mix = rettili + pesci
print(mix)
```

```
['serpente', 'lucertola', 'coccodrillo', 'tartaruga', 'merluzzo', 'anguilla', 'scorfano', 'tonno']
```

List Slicing

- Slice: a span of items that are taken from a sequence
 - List slicing format: $list[start : end]$
 - Span is a list containing copies of elements from $start$ up to, but not including, end
 - If $start$ not specified, 0 is used for start index
 - If end not specified, $\text{len}(list)$ is used for end index
 - Slicing expressions can include a step value and negative indexes relative to end of list

```
giorni = ["lunedì", "martedì", "mercoledì", "giovedì",
          "venerdì", "sabato", "domenica"]

turno = giorni[2:5]
print(turno)

turno2 = giorni[:5]
print(turno2)

turno3 = giorni[2:]
print(turno3)

turno4 = giorni[:]
print(turno4)

turno5 = giorni[-2:]
print(turno5)
```

```
['mercoledì', 'giovedì', 'venerdì']
['lunedì', 'martedì', 'mercoledì', 'giovedì', 'venerdì']
['mercoledì', 'giovedì', 'venerdì', 'sabato', 'domenica']
['lunedì', 'martedì', 'mercoledì', 'giovedì', 'venerdì', 'sabato', 'domenica']
['sabato', 'domenica']
```

Finding Items in Lists with the `in` Operator

- You can use the `in` operator to determine whether an item is contained in a list
 - General format: `item in list`
 - Returns `True` if the item is in the list, or `False` if it is not in the list
- Similarly you can use the `not in` operator to determine whether an item is not in a list



```
# Questo programma dimostra l'uso dell'operatore in
# all'interno di una lista.

def main():
    # Crea una lista di codici prodotto.
    prod_nums = ['V475', 'F987', 'Q143', 'R688']

    # Ottiene un codice prodotto da cercare.
    search = input('Inserire un codice prodotto: ')

    # Determina se il codice prodotto si trova nella lista.
    if search in prod_nums:
        print(f'{search} è stato trovato nella lista.')
    else:
        print(f'{search} non è stato trovato nella lista.')

    # Chiama la funzione main.
if __name__ == '__main__':
    main()
```

➡ Inserire un codice prodotto: V475
V475 è stato trovato nella lista.

List Methods and Useful Built-in Functions (1 of 4)

- append(*item*): used to add items to a list – *item* is appended to the end of the existing list
- index(*item*): used to determine where an item is located in a list
 - Returns the index of the first element in the list containing *item*
 - Raises `ValueError` exception if *item* not in the list



14 s

```
# Questo programma illustra come usare il metodo
# append per aggiungere elementi a una lista.

def main():
    # Per prima cosa crea una lista vuota.
    name_list = []
    # Crea una variabile per controllare il ciclo.
    again = 's'
    # Aggiunge nomi alla lista.
    while again == 's':
        # Ottiene un nome dall'utente.
        name = input('Inserisci un nome: ')
        # Aggiunge il nome alla lista.
        name_list.append(name)
        # Aggiungi un altro nome?
        print('Desideri aggiungere un altro nome?')
        again = input('s = sì, qualsiasi altra cosa = no: ')
        print()
    # Visualizza i nomi inseriti.
    print('Ecco i nomi che hai inserito.')
    for name in name_list:
        print(name)

# Chiama la funzione main.
if __name__ == '__main__':
    main()
```

☞ Inserisci un nome: aldo
Desideri aggiungere un altro nome?
s = sì, qualsiasi altra cosa = no: s

Inserisci un nome: michele
Desideri aggiungere un altro nome?
s = sì, qualsiasi altra cosa = no: n

<>

Ecco i nomi che hai inserito.

☰

aldo
michele

→-



```
oggetti = ["martello", "chiodo", "vite", "pinza"]
try:
    i = oggetti.index("vite")
    print(i)
except ValueError:
    print("oggetto non presente")
```

2

List Methods and Useful Built-in Functions (2 of 4)

- insert(index, item): used to insert *item* at position *index* in the list
- sort(): used to sort the elements of the list in ascending order
- remove(item): removes the first occurrence of *item* in the list
- reverse(): reverses the order of the elements in the list

```
# Questo programma mostra come ottenere
# l'indice di un elemento di una lista e sostituire
# quindi quell'elemento con un altro.

def main():
    # Crea una lista con alcuni elementi.
    food = ['Pizza', 'Hamburger', 'Patatine']
    # Visualizza la lista.
    print('Ecco gli elementi della lista food:')
    print(food)

    # Ottiene l'elemento da modificare.
    item = input('Quale elemento devo modificare? ')
    try:
        # Ottiene l'indice dell'elemento nella lista.
        item_index = food.index(item)
        # Ottiene il valore con cui modificare l'elemento.
        new_item = input('Inserisci il nuovo valore: ')
        # Sostituisce il vecchio elemento con quello nuovo.
        food[item_index] = new_item
        # Visualizza la lista.
        print('Ecco la lista modificata:')
        print(food)
    except ValueError:
        print('Non è possibile trovare questo elemento nella lista.')

    # Chiama la funzione main.
if __name__ == '__main__':
    main()
```

- Ecco gli elementi della lista food:
['Pizza', 'Hamburger', 'Patatine']
Quale elemento devo modificare? Patatine
Inserisci il nuovo valore: Insalata
Ecco la lista modificata:
['Pizza', 'Hamburger', 'Insalata']

```
# Questo programma illustra il metodo insert.

def main():
    # Crea una lista contenente alcuni nomi.
    names = ['James', 'Paolo', 'Francesca']

    # Visualizza la lista.
    print("Ecco la lista prima dell'inserimento:")
    print(names)

    # Inserisce un nuovo nome come elemento 0.
    names.insert(0, 'Rocco')

    # Visualizza nuovamente la lista.
    print("Ecco la lista dopo l'inserimento:")
    print(names)

# Chiama la funzione main.
if __name__ == '__main__':
    main()
```

➡ Ecco la lista prima dell'inserimento:
['James', 'Paolo', 'Francesca']
Ecco la lista dopo l'inserimento:
['Rocco', 'James', 'Paolo', 'Francesca']

List Methods and Useful Built-in Functions (3 of 4)

Table 7-1 A few of the list methods

Method	Description
<code>append(item)</code>	Adds <i>item</i> to the end of the list.
<code>index(item)</code>	Returns the index of the first element whose value is equal to <i>item</i> . A <code>ValueError</code> exception is raised if <i>item</i> is not found in the list.
<code>insert(index, item)</code>	Inserts <i>item</i> into the list at the specified <i>index</i> . When an item is inserted into a list, the list is expanded in size to accommodate the new item. The item that was previously at the specified index, and all the items after it, are shifted by one position toward the end of the list. No exceptions will occur if you specify an invalid index. If you specify an index beyond the end of the list, the item will be added to the end of the list. If you use a negative index that specifies an invalid position, the item will be inserted at the beginning of the list.
<code>sort()</code>	Sorts the items in the list so they appear in ascending order (from the lowest value to the highest value).
<code>remove(item)</code>	Removes the first occurrence of <i>item</i> from the list. A <code>ValueError</code> exception is raised if <i>item</i> is not found in the list.
<code>reverse()</code>	Reverses the order of the items in the list.

List Methods and Useful Built-in Functions (4 of 4)

- del statement: removes an element from a specific index in a list
 - General format: `del list[i]`
- min and max functions: built-in functions that returns the item that has the lowest or highest value in a sequence
 - The sequence is passed as an argument



```
my_list = [5, 4, 3, 2, 50, 40, 30]
print('Il valore più basso è', min(my_list))
```

Il valore più basso è 2

Copying Lists (1 of 2)

- To make a copy of a list you must copy each element of the list
 - Two methods to do this:
 - Creating a new empty list and using a `for` loop to add a copy of each element from the original list to the new list
 - Creating a new empty list and concatenating the old list to the new empty list

```
[8] # Crea una lista contenente dei valori.  
list1 = [1, 2, 3, 4]  
# Crea una lista vuota.  
list2 = []  
# Copia gli elementi di list1 in list2.  
for item in list1:  
    list2.append(item)
```



```
# Crea una lista contenente dei valori.  
list1 = [1, 2, 3, 4]  
# Crea una copia di list1.  
list2 = [] + list1
```

Copying Lists (2 of 2)

```
# Crea una lista.  
list1 = [1, 2, 3, 4]  
# Assegna la lista alla variabile list2.  
list2 = list1
```

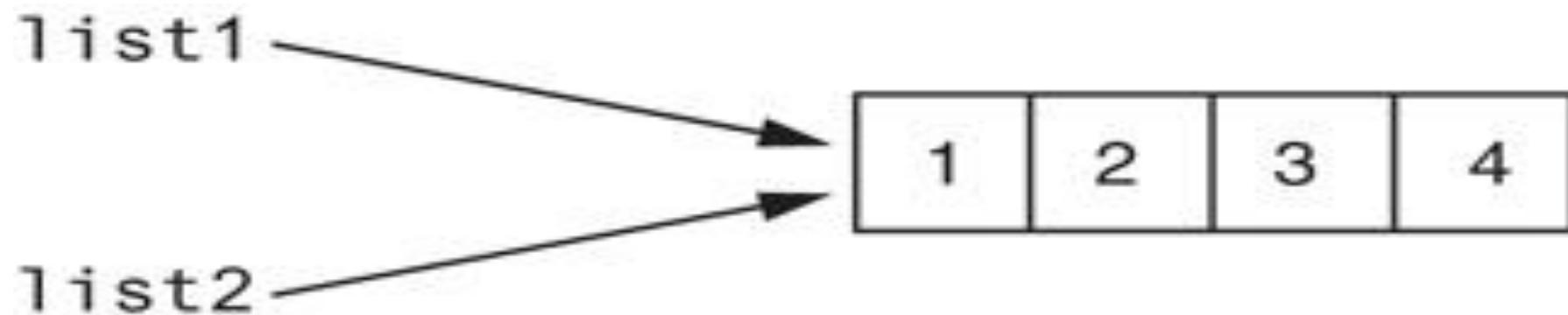


Figure 7-5 list1 and list2 reference the same list

Processing Lists (1 of 2)

- List elements can be used in calculations
- To calculate total of numeric values in a list use loop with accumulator variable
- To average numeric values in a list:
 - Calculate total of the values
 - Divide total of the values by `len(list)`
- List can be passed as an argument to a function



```
# Questo programma calcola il totale dei valori
# presenti in una lista.

def main():
    # Crea una lista.
    numbers = [2, 4, 6, 8, 10]

    # Crea una variabile da utilizzare come accumulatore.
    total = 0

    # Calcola il totale degli elementi della lista.
    for value in numbers:
        total += value

    # Visualizza il totale degli elementi della lista.
    print(f'Il totale degli elementi è {total}.')

# Chiama la funzione main.
if __name__ == '__main__':
    main()
```

➡ Il totale degli elementi è 30.

Processing Lists (2 of 2)

- A function can return a reference to a list
- To save the contents of a list to a file:
 - Use the file object's `writelines` method
 - Does not automatically write `\n` at the end of each item
 - Use a `for` loop to write each element and `\n`
- To read data from a file use the file object's `readlines` method



```
# Questo programma utilizza il metodo writelines per salvare
# in un file una lista di stringhe.

def main():
    # Crea una lista di stringhe.
    cities = ['New York', 'Boston', 'Atlanta', 'Dallas']

    # Apre un file in scrittura.
    outfile = open('cities.txt', 'w')

    # Scrive la lista nel file.
    outfile.writelines(cities)

    # Chiude il file.
    outfile.close()

# Chiama la funzione main.
if __name__ == '__main__':
    main()
```

The screenshot shows a Python code editor interface. On the left, the file tree displays a folder named 'sample_data' containing a file named 'cities.txt'. The main workspace shows a Python script named 'cities.py' with the following content:

```
# Questo programma utilizza il metodo writelines per salvare
# in un file una lista di stringhe.

def main():
    # Crea una lista di stringhe.
    cities = ['New York', 'Boston', 'Atlanta', 'Dallas']

    # Apre un file in scrittura.
    outfile = open('cities.txt', 'w')

    # Scrive la lista nel file.
    outfile.writelines(cities)

    # Chiude il file.
    outfile.close()

    # Chiama la funzione main.
if __name__ == '__main__':
    main()
```

The status bar at the top indicates 'RAM' and 'Disco' usage. The bottom right corner shows the contents of the 'cities.txt' file: '1 New YorkBostonAtlantaDallas'.

Two-Dimensional Lists (1 of 3)

- Two-dimensional list: a list that contains other lists as its elements
 - Also known as nested list
 - Common to think of two-dimensional lists as having rows and columns
 - Useful for working with multiple sets of data
- To process data in a two-dimensional list need to use two indexes
- Typically use nested loops to process



```
students = [['Joe', 'Kim'], ['Sam', 'Sue'], ['Kelly', 'Chris']]  
print(students)
```

```
[['Joe', 'Kim'], ['Sam', 'Sue'], ['Kelly', 'Chris']]
```

Two-Dimensional Lists (2 of 3)

	Column 0	Column 1
Row 0	'Joe'	'Kim'
Row 1	'Sam'	'Sue'
Row 2	'Kelly'	'Chris'

Figure 7-8 A two-dimensional list

Tuples (1 of 3)

- Tuple: an immutable sequence
 - Very similar to a list
 - Once it is created it cannot be changed
 - Format: `tuple_name = (item1, item2)`
 - Tuples support operations as lists
 - Subscript indexing for retrieving elements
 - Methods such as `index`
 - Built in functions such as `len`, `min`, `max`
 - Slicing expressions
 - The `in`, `+`, and `*` operators



```
my_tuple = (1, 2, 3, 4, 5)  
print(my_tuple)
```

```
↳ (1, 2, 3, 4, 5)
```

Tuples (2 of 3)

- Tuples do not support the methods:

- append
- remove
- insert
- reverse
- sort

Tuples (3 of 3)

- Advantages for using tuples over lists:
 - Processing tuples is faster than processing lists
 - Tuples are safe
 - Some operations in Python require use of tuples
- list () function: converts tuple to list
- tuple () function: converts list to tuple

Corso di STATISTICA, INFORMATICA, ELABORAZIONE DELLE INFORMAZIONI

*Modulo di Sistemi di Elaborazione delle
Informazioni*



**UNIVERSITÀ DEGLI STUDI
DELLA BASILICATA**

Liste e Tuple

Docente:
**Domenico Daniele
Bloisi**

