

# *Corso di STATISTICA, INFORMATICA, ELABORAZIONE DELLE INFORMAZIONI*

*Modulo di Sistemi di Elaborazione delle  
Informazioni*

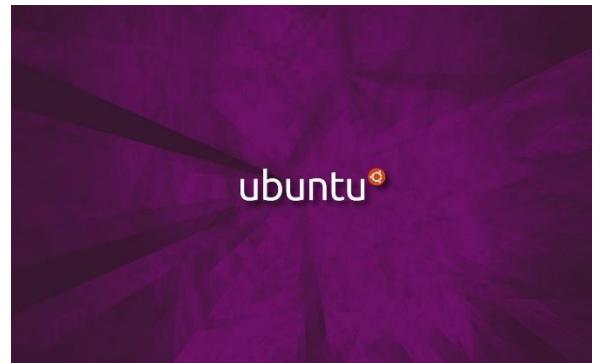
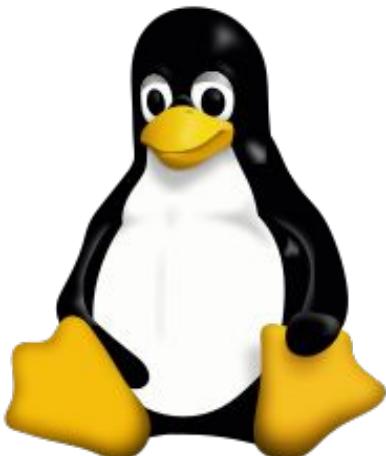
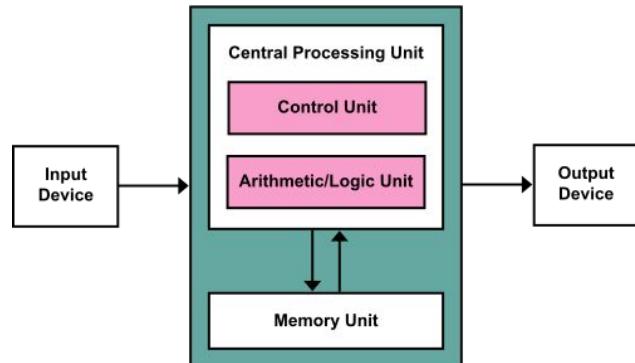
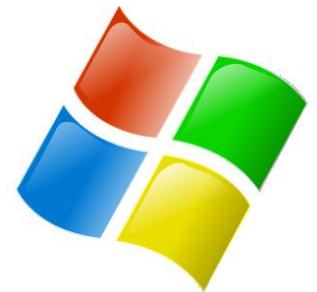
# File ed Eccezioni



**UNIVERSITÀ DEGLI STUDI  
DELLA BASILICATA**



Docente:  
**Domenico Daniele  
Bloisi**



# Domenico Daniele Bloisi

---

- Professore Associato

Dipartimento di Matematica, Informatica  
ed Economia

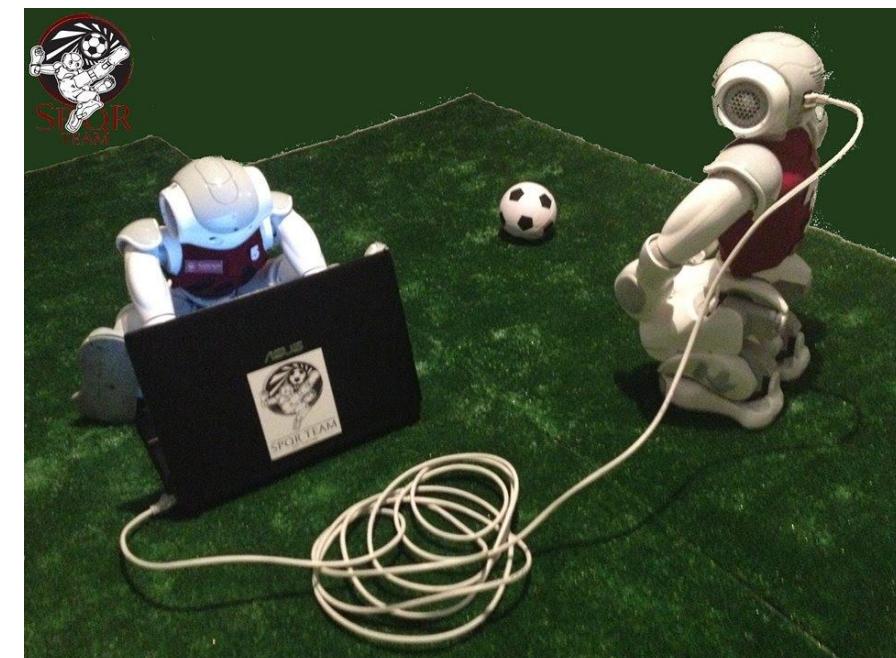
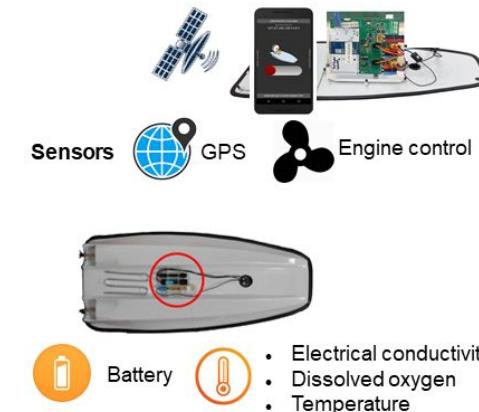
Università degli studi della Basilicata

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

- SPQR Robot Soccer Team

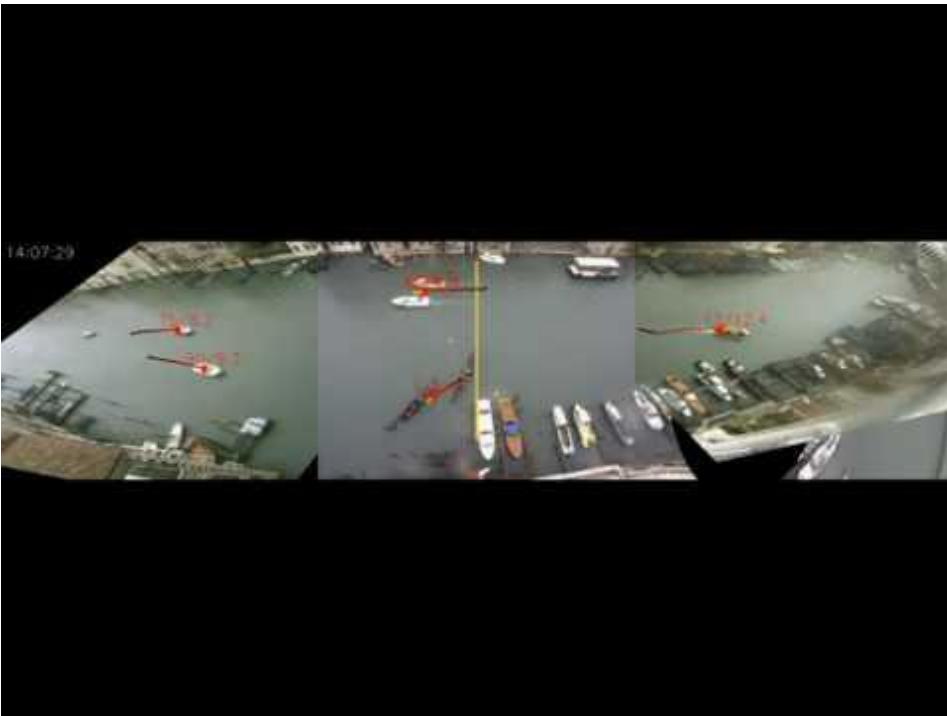
Dipartimento di Informatica, Automatica  
e Gestionale Università degli studi di  
Roma “La Sapienza”

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



# Interessi di ricerca

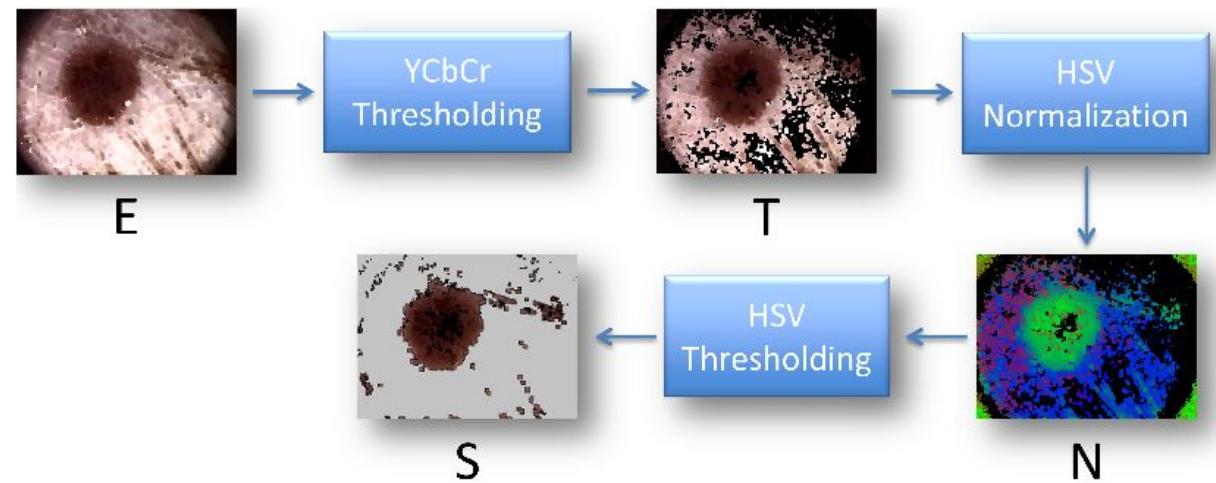
- Intelligent surveillance
- Robot vision
- Medical image analysis



[https://youtu.be/9a70Ucgbi\\_U](https://youtu.be/9a70Ucgbi_U)

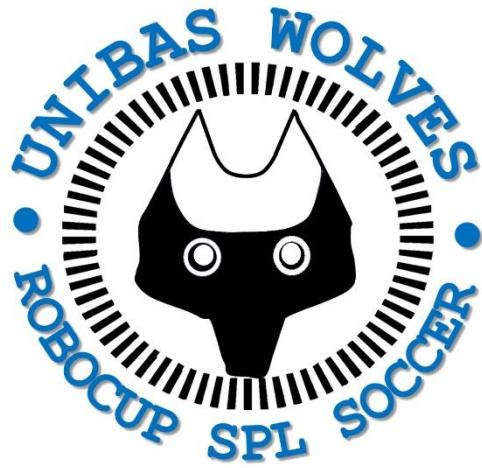


<https://youtu.be/2KHNZX7UIWQ>



# 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

# Informazioni sul modulo

---

- Home page del modulo:  
<https://web.unibas.it/bloisi/corsi/sei.html>
- Martedì dalle 11:30 alle 13:30

# 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](mailto:domenico.bloisi@unibas.it)



# Recap

# Variabili locali

---

I parametri di una funzione e le eventuali altre variabili alle quali viene assegnato un valore all'interno di essa sono dette locali, cioè vengono create dall'interprete nel momento in cui la funzione viene eseguita (con una chiamata) e vengono distrutte quando l'esecuzione della funzione termina.

# Variabili locali: esempio

---

The screenshot shows a code editor interface with a light gray background. On the left, there's a vertical toolbar with a green checkmark icon and a play button icon. The main area contains the following Python code:

```
def stampa_quadrato(x):
    a = x ** 2
    print(a)

def main():
    a = 5
    stampa_quadrato(a)
    print(a)

main()
```

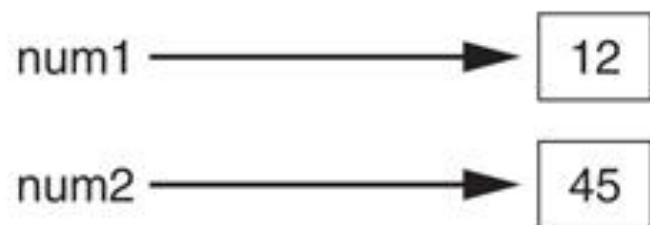
25

5

# Passing Multiple Arguments (2 of 2)

```
def main():
    print('The sum of 12 and 45 is')
    show_sum(12, 45)

def show_sum(num1, num2):
    result = num1 + num2
    print(result)
```



**Figure 5-16** Two arguments passed to two parameters

# Keyword Arguments

```
✓ [6] def stampa_data(giorno, mese, anno):  
0 s     print("La data di oggi è",  
           "giorno:", giorno,  
           "mese:", mese,  
           "anno:", anno)  
  
       stampa_data(6,12,2022)
```

La data di oggi è giorno: 6 mese: 12 anno: 2022

```
✓ [7] stampa_data(12,6,2022)  
0 s
```

La data di oggi è giorno: 12 mese: 6 anno: 2022

```
✓ [8] stampa_data(mese=12, giorno=6, anno=2022)  
0 s
```

La data di oggi è giorno: 6 mese: 12 anno: 2022

# Variabili globali

---

Se invece all'interno di una funzione il nome di una variabile (che non sia uno dei parametri) compare in una espressione senza che in precedenza nella funzione sia stato assegnato a essa alcun valore, tale variabile è considerata globale, cioè l'interprete assume che il suo valore sia stato definito nelle istruzioni precedenti la chiamata della funzione.

In questo modo, le istruzioni di una funzione possono accedere al valore di variabile definita nel programma chiamante (se tale variabile non esiste si ottiene un messaggio di errore).

# Variabili globali

---



```
# Crea una variabile globale.  
my_value = 10  
  
# La funzione show_value stampa  
# il valore della variabile globale.  
def show_value():  
    my_value = 15  
    print(my_value)  
  
# Chiama la funzione show_value.  
show_value()  
print(my_value)
```

→ 15  
10

# Variabili globali

---



```
# Crea una variabile globale.  
my_value = 10  
  
# La funzione show_value stampa  
# il valore della variabile globale.  
def show_value():  
    global my_value  
    my_value = 15  
    print(my_value)  
  
# Chiama la funzione show_value.  
show_value()  
print(my_value)
```

15

15

# Standard Library Functions and the `import` Statement (1 of 3)

- Standard library: library of pre-written functions that comes with Python
  - *Library functions* perform tasks that programmers commonly need
    - Example: `print`, `input`, `range`
      - Viewed by programmers as a “black box”
- Some library functions built into Python interpreter
  - To use, just call the function

# Esempi di funzioni built-in

---

`len(stringa)`

restituisce il numero di caratteri di una stringa

`abs(numero)`

restituisce il valore assoluto di un numero

`str(espressione)`

restituisce una stringa composta dalla sequenza di caratteri

corrispondenti alla rappresentazione del valore di espressione (che può essere di un qualsiasi tipo: numero, stringa, valore logico, ecc.)

# Esempi di funzioni built-in

---

`int (numero)`

restituisce la parte intera di un numero

`float (numero)`

restituisce il valore di numero come numero frazionario (floating point); può essere usata per evitare che la divisione tra interi produca la sola parte intera del quoziente, per es.: `float (2) / 3`

`int (stringa)`

Se stringa contiene la rappresentazione di un numero intero, restituisce il numero corrispondente a tale valore; in caso contrario produce un errore

`float (stringa)`

Se stringa contiene la rappresentazione di un numero qualsiasi (sia intero che frazionario), restituisce il suo valore espresso come numero frazionario; in caso contrario produce un errore

# from import

---

Per poter chiamare una funzione di librerie come `math` e `random` è necessario utilizzare la combinazione `from import`

Sintassi:

```
from nome_libreria import nome_funzione
```

- `nome_libreria` è il nome simbolico di una libreria
- `nome_funzione` può essere:
  - il nome di una specifica funzione di tale libreria (questo consentirà di usare solo tale funzione)
  - il simbolo `*` indicante tutte le funzioni di tale libreria

Se la combinazione `from import` non viene usata correttamente, la chiamata di funzione produrrà un errore, come mostrato negli esempi seguenti.

# from import

---



```
from random import randint  
randint(1,100)
```

→ 35

# Returning Strings

- You can write functions that return strings
- For example:

```
def get_name():  
    # Get the user's name.  
    name = input('Enter your name: ')  
    # Return the name.  
    return name
```

# Esercizio 13

---

Scrivere un programma che

- chieda all'utente di inserire un numero
- utilizzi una funzione che calcoli il fattoriale di tale numero
- stampi il risultato

# Possibile soluzione Esercizio 13

---

definizione di fattoriale

$$n! = n * (n - 1) * (n - 2) * \dots * 1$$

per esempio, con  $5!$  avremo

$$5! = 5 * 4 * 3 * 2 * 1$$

che possiamo riscrivere come

$$5! = 1 * 2 * 3 * 4 * 5$$

quindi

```
▶ n = int(input("Inserisci un intero: "))

print(f"{n}! = ", end="")

fattoriale = 1
for i in range(1,n+1):
    fattoriale *= i

print(fattoriale)
```

```
⇨ Inserisci un intero: 5
5! = 120
```

# soluzione usando import math

---

✓ 28

import math

n = int(input("Inserisci un intero: "))

print(f"{n}! = ", end="")

fattoriale = math.factorial(n)

print(fattoriale)

↳ Inserisci un intero: 5  
5! = 120

# soluzione usando from import

---

✓ 2 s

```
from math import factorial

n = int(input("Inserisci un intero: "))

print(f"{n}! = ", end="")

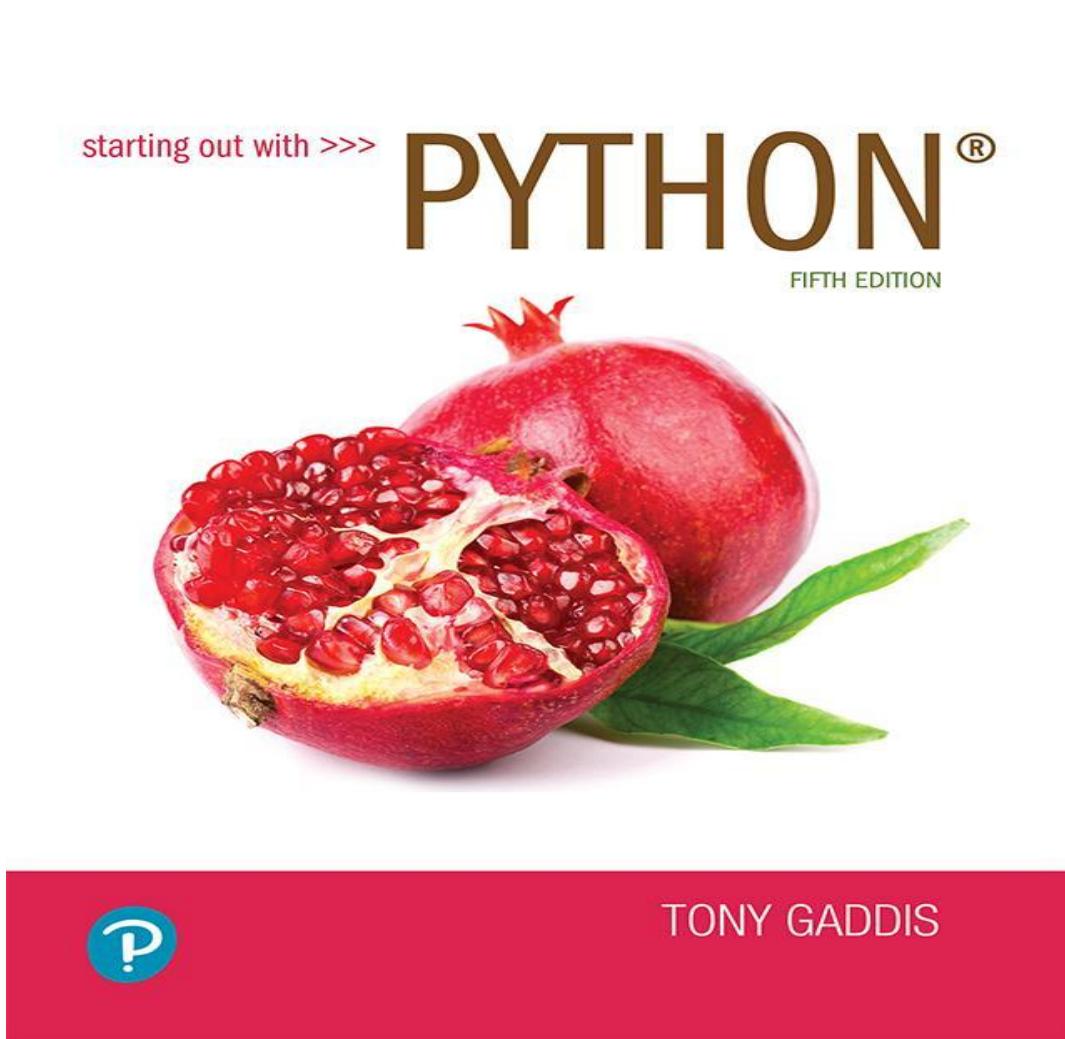
fattoriale = factorial(n)

print(fattoriale)
```

➡ Inserisci un intero: 5  
5! = 120

# Starting out with Python

Fifth Edition



## Chapter 6

### Files and Exceptions

# Topics

- Introduction to File Input and Output
- Using Loops to Process Files
- Processing Records
- Exceptions

# File System

---

- Il **file system** fornisce il meccanismo per la **memorizzazione in linea** di dati e programmi appartenenti al sistema operativo
- Il **file system** è composto da:
  - un insieme di **file** (contenenti dati)
  - una struttura di **directory** (per organizzare i file)
- Il **file system** risiede, nella maggior parte dei casi, in **memoria secondaria**

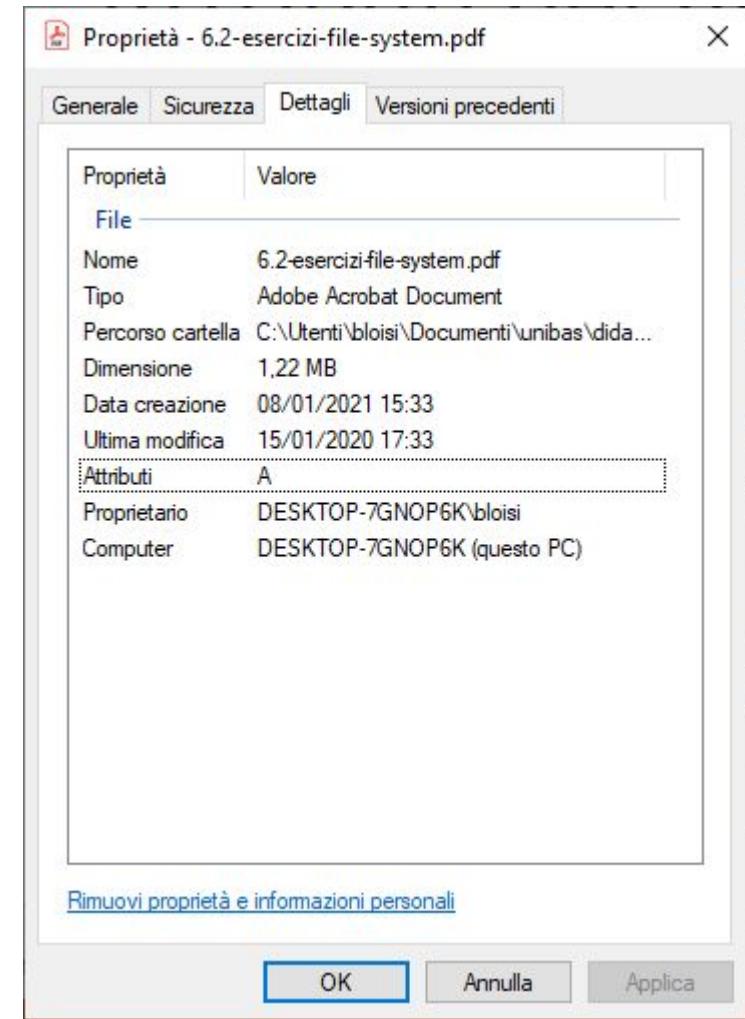
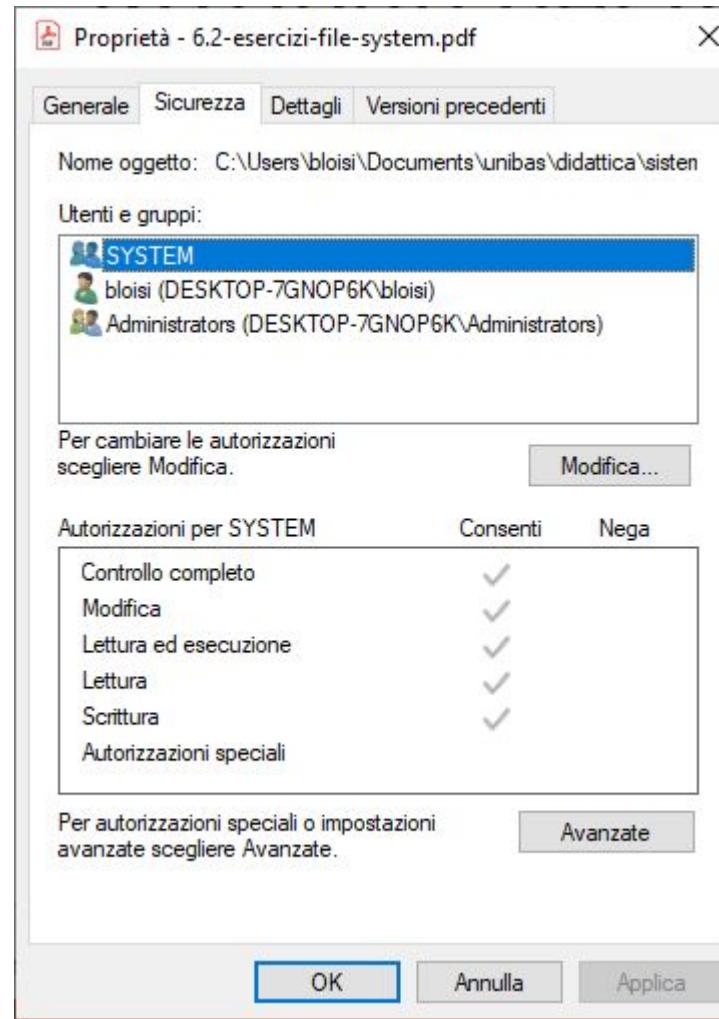
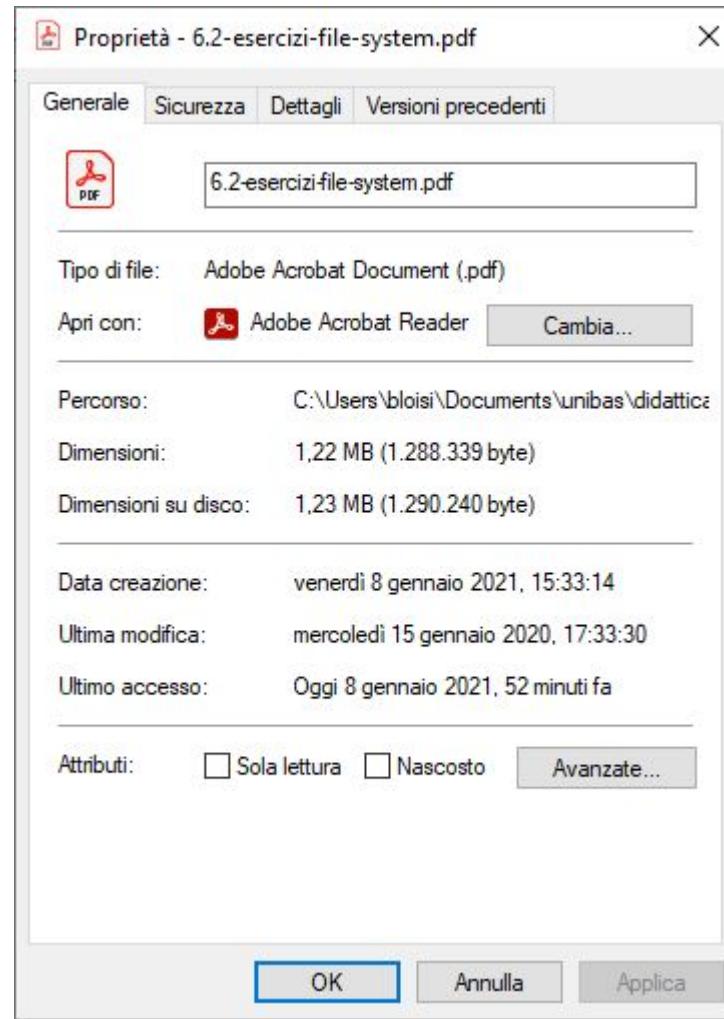
# File

---

- Un **file** è un insieme di informazioni correlate, registrate in memoria secondaria, cui è stato assegnato un **nome**.
- Un file **ha attributi** che possono variare secondo il sistema operativo, ma che tipicamente comprendono i seguenti:



# Attributi dei file – Windows 10



# Directory

---

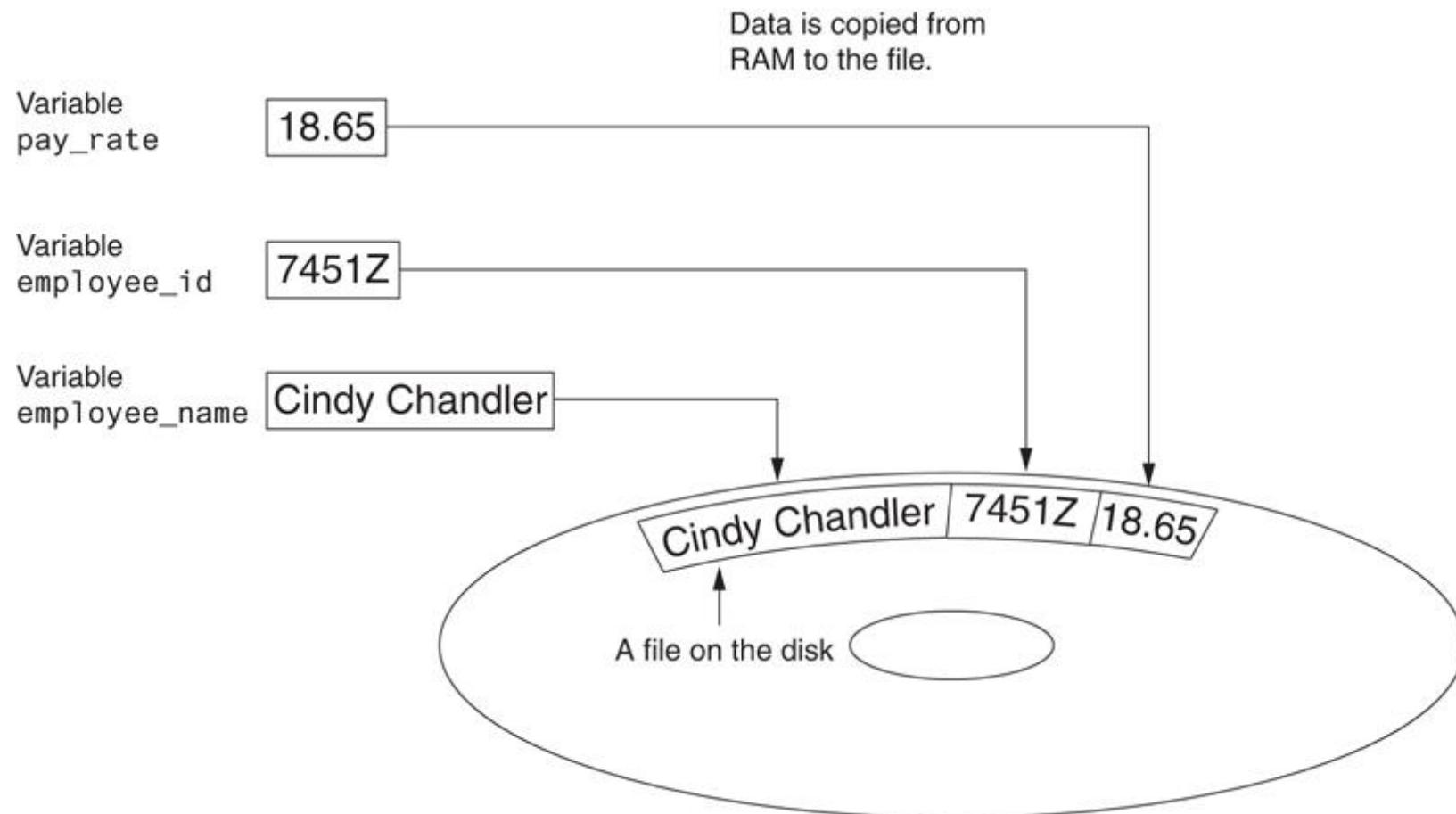
La directory è una sorta di **tabella di simboli** che viene impiegata per tradurre i nomi dei file negli elementi in essa contenuti.

```
training_demo/
└── annotations/
└── exported-models/
└── images/
    ├── test/
    │   └── 0.png
    └── train/
        └── 234.png
└── models/
└── pre-trained-models/
└── README.md
```

# Introduction to File Input and Output

- For program to retain data between the times it is run, you must save the data
  - Data is saved to a file, typically on computer disk
  - Saved data can be retrieved and used at a later time
- “Writing data to”: saving data on a file
- Output file: a file that data is written to

# Writing data to a file

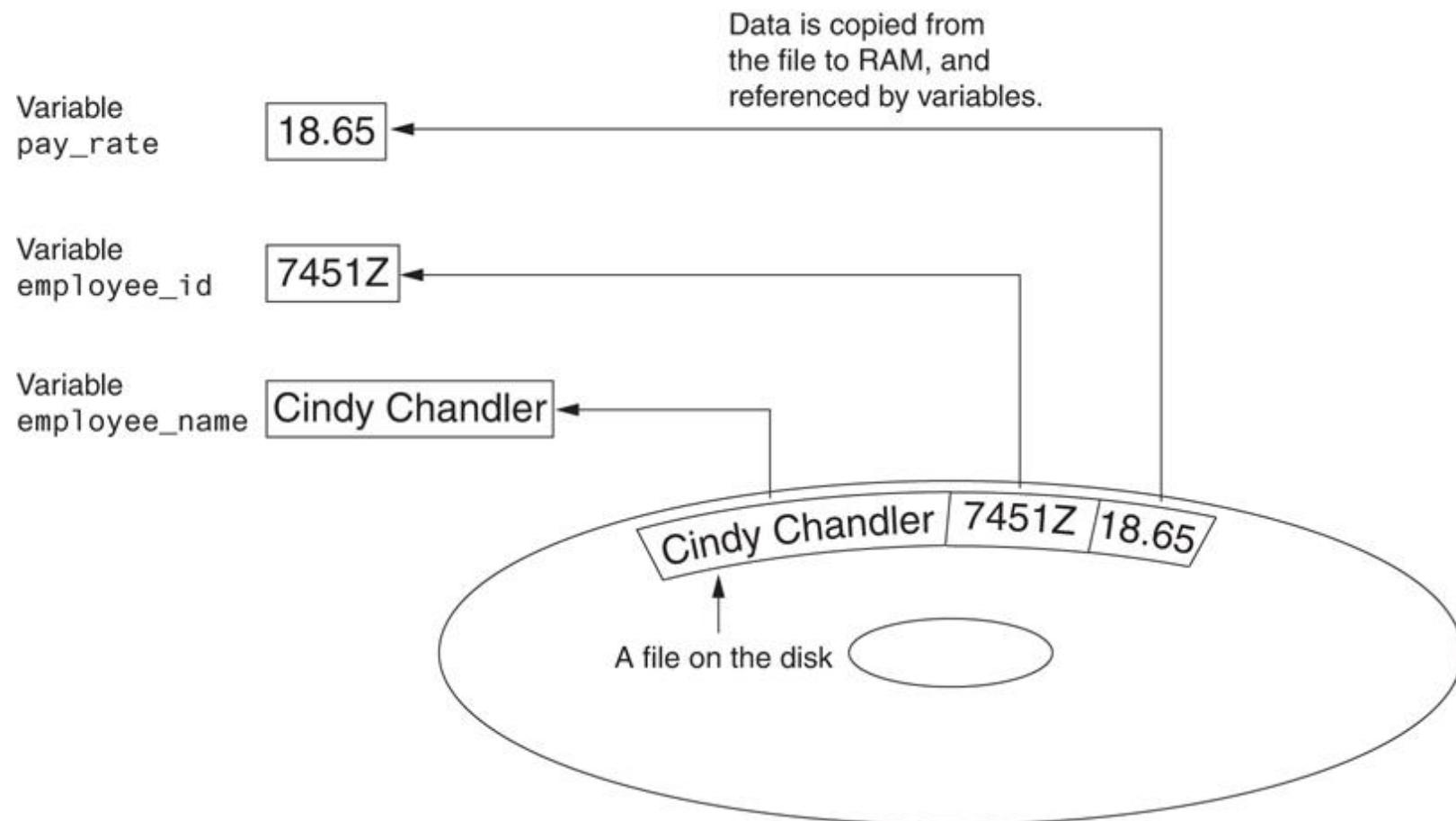


**Figure 6-1** Writing data to a file

# Reading data from a file (1 of 2)

- “Reading data from”: process of retrieving data from a file
- Input file: a file from which data is read
- Three steps when a program uses a file
  - Open the file
  - Process the file
  - Close the file

# Reading data from a file (2 of 2)



**Figure 6-2** Reading data from a file

# Types of Files and File Access Methods

- In general, two types of files
  - Text file: contains data that has been encoded as text
  - Binary file: contains data that has not been converted to text
- Two ways to access data stored in file
  - Sequential access: file read sequentially from beginning to end, can't skip ahead
  - Direct access: can jump directly to any piece of data in the file

# Tipi di file

---

Tipo di file	Estensione usuale	Funzione
Esegibile	exe, com, bin, o nessuna	Programma eseguibile, in linguaggio macchina
Oggetto	obj, o	Compilato, in linguaggio di macchina, non linkato
Codice sorgente	c, cc, java, perl, asm	Codice sorgente in vari linguaggi di programmazione
Batch	bat, sh	Comandi per l'interprete dei comandi
Markup	xml, html, tex	Dati testuali, documenti
Word processor	xml, rtf, docx	Vari formati di word processor
Libreria	lib, a, so, dll	Librerie di procedure per la programmazione
Stampa o visualizzazione	gif, pdf, jpg	File ASCII o binari in formato per la stampa o la visualizzazione
Archivio	rar, zip, tar	File contenenti più file tra loro correlati, talvolta compressi, per archiviazione o memorizzazione
Multimediali	mpeg, mov, mp3, mp4, avi	File binari contenenti informazioni audio o A/V

**Figura 13.3** Comuni tipi di file.

# Esempio Markdown .md

The screenshot shows a GitHub repository page for 'detectball'. The URL is [github.com/dbloisi/detectball](https://github.com/dbloisi/detectball). The page displays the contents of the README.md file.

```
detectball
detectball is a software for detecting the RoboCup SPL black and white ball in images captured by NAO robots - Copyright 2017 Domenico Daniele Bloisi
detectball is part of the tutorial "How to Use OpenCV for Ball Detection" by Domenico D. Bloisi (http://web.unibas.it/bloisi/tutorial/balldetection.html) and it is distributed under the terms of the GNU Lesser General Public License (Lesser GPL)
Prerequisites
You need to have OpenCV 4.3 installed to compile and run the code
Build the code


```
mkdir build
cd build && cmake ..
make
```


Run the code


```
detectball test
```


```

The screenshot shows a GitHub edit session for the README.md file of the 'detectball' repository. The URL is [github.com/dbloisi/detectball/edit/master/README.md](https://github.com/dbloisi/detectball/edit/master/README.md). The interface includes tabs for Code, Issues, Pull requests, Actions, and Projects. The 'Code' tab is selected.

```
detectball
detectball is a software for detecting the RoboCup SPL black and white ball in images captured by NAO robots - Copyright 2017 Domenico Daniele Bloisi
detectball is part of the tutorial "How to Use OpenCV for Ball Detection" by Domenico D. Bloisi (http://web.unibas.it/bloisi/tutorial/balldetection.html) and it is distributed under the terms of the GNU Lesser General Public License (Lesser GPL)
### Prerequisites
You need to have OpenCV 4.3 installed to compile and run the code
### Build the code
```
mkdir build
cd build && cmake ..
make
```
### Run the code

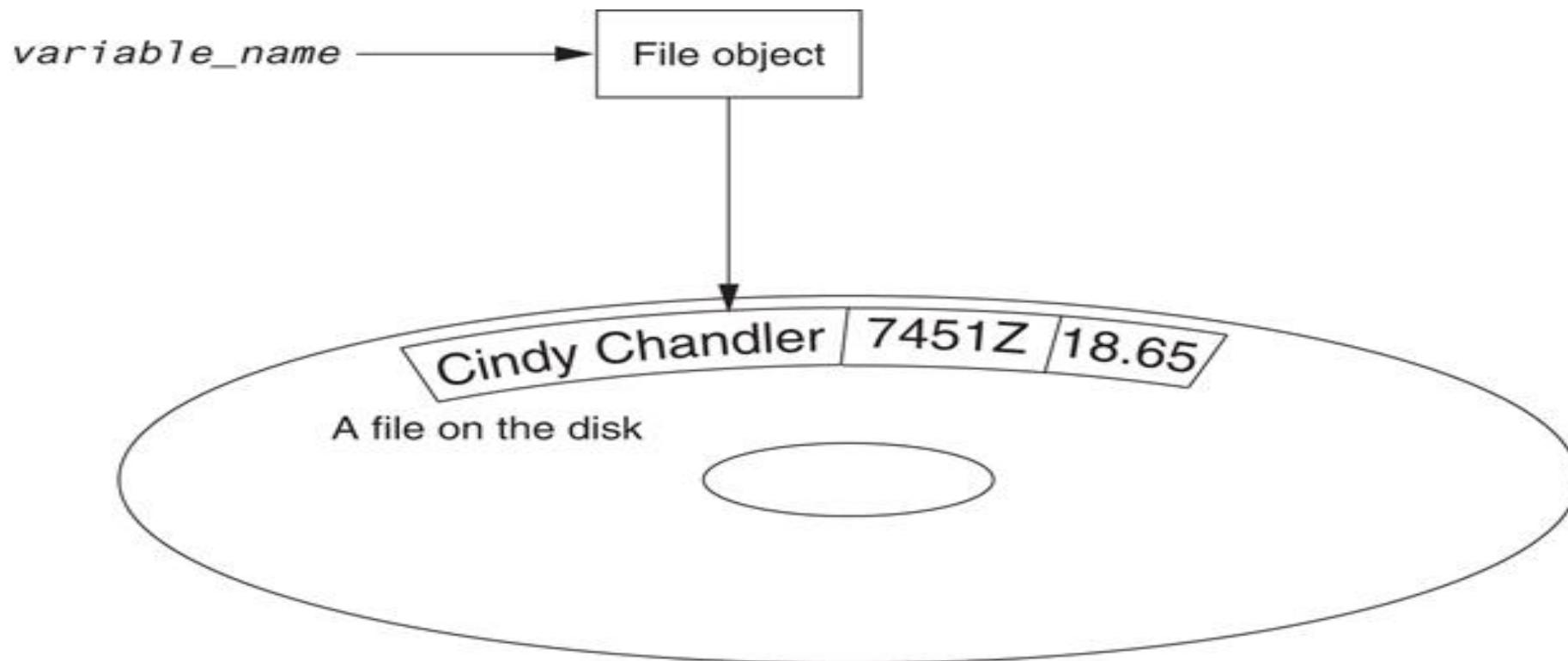

```
detectball test
```


```

# Filenames and File Objects (1 of 2)

- Filename extensions: short sequences of characters that appear at the end of a filename preceded by a period
  - Extension indicates type of data stored in the file
- File object: object associated with a specific file
  - Provides a way for a program to work with the file: file object referenced by a variable

# Filenames and File Objects (2 of 2)



**Figure 6-4** A variable name references a file object that is associated with a file

# Accesso al file

---

La lettura o la scrittura di dati su un file avvengono in tre fasi:

1. apertura del file, per mezzo della funzione built-in `open`
2. esecuzione di una o più operazioni di lettura o scrittura, per mezzo delle opportune funzioni built-in
3. chiusura del file, per mezzo della funzione built-in `close`

# Opening a File

- open function: used to open a file
  - Creates a file object and associates it with a file on the disk
  - General format:
    - *file\_object* = `open(filename, mode)`
- Mode: string specifying how the file will be opened
  - Example: reading only ('`r`'), writing ('`w`'), and appending ('`a`')

# open

---

La funzione `open` restituisce un valore strutturato contenente alcune informazioni sul file.

Sintassi:

```
variabile = open(nome_file, modalita)
```

- `variabile`: il nome della variabile che verrà associata al file
- `nome_file`: una stringa contenente il nome del *file*
- `modalita`: una stringa che indica la modalità di apertura (lettura o scrittura)

# open

---

Il nome del file che si desidera aprire deve essere passato come argomento della funzione `open` sotto forma di stringa.

Il nome del file può essere:

- **assoluto**, cioè preceduto dalla sequenza (detta anche path) dei nomi delle directory che lo contengono a partire dalla directory radice del file system, scritta secondo la sintassi prevista dal sistema operativo del proprio calcolatore
- **relativo**, cioè composto dal solo nome del file: questo è possibile solo se la funzione `open` è chiamata da un programma o da una funzione che si trovi nella stessa directory che contiene il file da aprire

# open

---

Nel caso di un file di nome dati.txt che si trovi nella directory C:\Users\Erika\ di un sistema operativo Windows:

- il nome relativo è dati.txt
- il nome assoluto è C:\Users\Erika\dati.txt

Se un file con lo stesso nome (dati.txt) è memorizzato nella directory /users/Erika/ di un sistema operativo Linux oppure Mac OS:

- il nome relativo è ancora dati.txt
- il nome assoluto è /users/Erika/dati.txt

# open

---

È possibile aprire in modalità di lettura solo un file esistente. Se il file non esiste si otterrà un errore.

La modalità di scrittura consente invece anche la creazione di un nuovo file. Più precisamente, attraverso la modalità di scrittura è possibile:

- creare un nuovo file
- aggiungere dati in coda a un file già esistente
- sovrascrivere (cancellare e sostituire) il contenuto di un file già esistente

# open

---

Nella chiamata di `open` la modalità di accesso è indicata (come secondo argomento) da una stringa composta da un singolo carattere:

- "r" (read ): lettura (se il file non esiste si ottiene un errore)
- "w" (write): (sovra) scrittura
  - se il file non esiste viene creato
  - se il file esiste viene sovrascritto, cancellando i dati contenuti in esso
- "a" (append ): scrittura (aggiunta)
  - se il file non esiste viene creato
  - se il file esiste i nuovi dati saranno aggiunti in coda a quelli già esistenti

# Opening a File

```
test_file = open('test.txt', 'w')
```

Modalità	Descrizione
'r'	Apre un file in sola lettura. Non è possibile modificare il file o scrivervi sopra.
'w'	Apre un file in scrittura. Se il file esiste già, ne elimina il contenuto; se non esiste, lo crea.
'a'	Apre un file in modalità aggiunta. Tutti i dati scritti nel file vengono aggiunti alla fine. Se il file non esiste, lo crea.

# Specifying the Location of a File

- If `open` function receives a filename that does not contain a path, assumes that file is in same directory as program
- If program is running and file is created, it is created in the same directory as the program
  - Can specify alternative path and file name in the `open` function argument
    - Prefix the path string literal with the letter `r`

# open (nuovo file)

---

The screenshot shows a Jupyter Notebook interface with the following details:

- Header:** CO file.ipynb
- Menu Bar:** File Modifica Visualizza Inserisci Runtime Strumenti Guida
- Left Sidebar:** File, search, upload, folder icons, and a sample\_data folder.
- Code Cell:** + Codice + Testo
- Code Content:**

```
f = open("dati.txt","w")
print(f)
!ls
```

# open (nuovo file)

file.ipynb

File Modifica Visualizza Inserisci Runtime Strumenti Guida

File

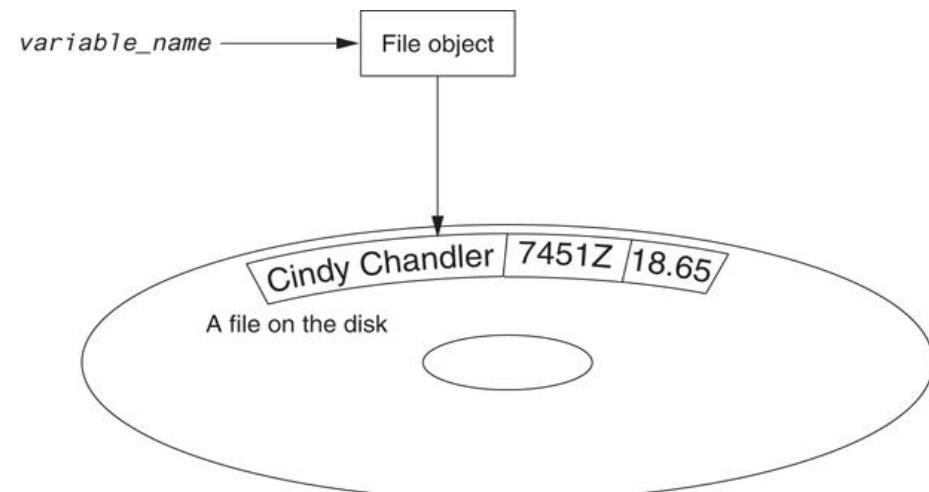
sample\_data  
dati.txt

```
+ Codice + Testo
```

✓ 0 s

f = open("dati.txt","w")  
print(f)  
!ls

<\_io.TextIOWrapper name='dati.txt' mode='w' encoding='UTF-8'>  
dati.txt sample\_data



# open (file esistente)

The screenshot shows a Jupyter Notebook interface with the following elements:

- File Bar:** CO file.ipynb ★
- Menu:** File Modifica Visualizza Inserisci Runtime Strumenti Guida
- File Browser:** Shows a tree view of files and folders:
  - sample\_data
    - README.md
    - anscombe.json
    - california\_housing\_test.csv
    - california\_housing\_train.csv
    - mnist\_test.csv
    - mnist\_train\_small.csv
  - dati.txt
- Code Editor:** + Codice + Testo
- Code Cells:**
  - [1] f = open("dati.txt","w")  
print(f)  
!ls
  - <\_io.TextIOWrapper name='dati.txt' mode='w' encoding='UTF-8'>  
dati.txt sample\_data
  - f = open("sample\_data/README.md","r")

# open (file NON esistente)

The screenshot shows a Jupyter Notebook interface with the following details:

- File Explorer:** On the left, there's a sidebar titled "File" showing the directory structure. It includes icons for search, upload, folder, and refresh. The "sample\_data" folder contains files: README.md, anscombe.json, california\_housing\_test.csv, california\_housing\_train.csv, mnist\_test.csv, mnist\_train\_small.csv, and dati.txt.
- Code Editor:** The main area has tabs for "+ Codice" and "+ Testo". The "Codice" tab is active, displaying the following code:

```
[1] f = open("dati.txt","w")
print(f)
!ls
```

Output:

```
<_io.TextIOWrapper name='dati.txt' mode='w' encoding='UTF-8'>
dati.txt  sample_data
```

```
[3] f = open("sample_data/README.md","r")
```

The line `f = open("sample_data/pippo.txt","r")` is highlighted in red, indicating an error. A red play button icon is positioned next to it.

**Traceback:**

```
FileNotFoundException                                     Traceback (most recent call last)
<ipython-input-4-fef62098ac9f> in <module>
----> 1 f = open("sample_data/pippo.txt","r")
```

**Message:**

```
FileNotFoundException: [Errno 2] No such file or directory: 'sample_data/pippo.txt'
```

**Buttons:** At the bottom right of the code editor, there are two buttons: "SEARCH STACK OVERFLOW" and a large red "RUN" button.

# Writing Data to a File

- Method: a function that belongs to an object
  - Performs operations using that object
- File object's `write` method used to write data to the file
  - Format: `file_variable.write(string)`
- File should be closed using file object `close` method
  - Format: `file_variable.close()`

# close

---

Quando le operazioni di lettura o scrittura su un file sono terminate, il file deve essere chiuso attraverso la funzione built-in `close`. Questo impedirà l'esecuzione di ulteriori operazioni su tale file, fino a che esso non venga eventualmente riaperto.

Sintassi:

`variabile.close()`

dove `variabile` deve essere la variabile usata nell'apertura dello stesso file attraverso la funzione `open`

Esempio

```
f = open ("dati.txt", "r")
#operazioni di I/O su dati.txt
f.close()
```

# close

---



```
!ls  
f = open("README.md", "r")  
print(f)  
f.close()
```

```
anscombe.json          dati.txt           README.md  
california_housing_test.csv  mnist_test.csv  
california_housing_train.csv  mnist_train_small.csv  
<_io.TextIOWrapper name='README.md' mode='r' encoding='UTF-8'>
```

# write

---

In un file di testo che sia stato aperto in scrittura (in modalità "w" oppure "a") è possibile scrivere dati sotto forma di stringhe (cioè sequenze di caratteri) attraverso la funzione built-in `write`

Sintassi:

```
variabile.write(stringa)
```

- `variabile` è la variabile associata al file
- `stringa` è una stringa contenente la sequenza di caratteri da scrivere nel file

La chiamata di `write` con un file aperto in lettura (in modalità "r") produce un messaggio di errore.

# write

The screenshot shows a Jupyter Notebook interface with the following components:

- Header:** Includes a logo, the file name "file.ipynb", and a star icon.
- Menu Bar:** File, Modifica, Visualizza, Inserisci, Runtime, Strumenti, Guida, and Salvataggio in corso... (Saving in progress).
- File Explorer:** Shows a tree view of files and folders. The "sample\_data" folder contains: README.md, anscombe.json, california\_housing\_test.csv, california\_housing\_train.csv, mnist\_test.csv, mnist\_train\_small.csv, and dati.txt.
- Code Cell:** Contains Python code to open a file in write mode, write content to it, and close it. The code is:

```
f = open("dati.txt","w")
f.write("contenuto da scrivere nel file")
f.close()
```
- Output Cell:** Shows the output of the code, which is the file "dati.txt" containing the number "1".

# write

The screenshot shows a Jupyter Notebook interface with the following components:

- Header:** CO file.ipynb ★ File Modifica Visualizza Inserisci Runtime Strumenti Guida Tutte le modifiche sono state salvate
- File Explorer:** Shows a tree structure with a root folder containing README.md, anscombe.json, california\_housing\_test.csv, california\_housing\_train.csv, mnist\_test.csv, mnist\_train\_small.csv, and dati.txt. A sample\_data folder is also present.
- Code Cell:** Contains the following Python code:

```
f = open("dati.txt", "w")
f.write("contenuto da scrivere nel file")
f.close()
```
- Output Cell:** Shows the result of the execution: "dati.txt" with the content "1 contenuto da scrivere nel file".

# write (errore)

The screenshot shows a Jupyter Notebook interface with the following components:

- Header:** CO file.ipynb
- File menu:** File Modifica Visualizza Inserisci Runtime Strumenti Guida Tutte le modifiche sono state salvate
- File pane:** Shows a tree structure with a sample\_data folder containing README.md, anscombe.json, california\_housing\_test.csv, california\_housing\_train.csv, mnist\_test.csv, mnist\_train\_small.csv, and dati.txt.
- Code cell:** Contains two code snippets:
  - [11] f = open("dati.txt","w")  
f.write("contenuto da scrivere nel file")  
f.close()
  - 0: ! f = open("dati.txt","r")  
f.write("contenuto da scrivere nel file")  
f.close()
- Output pane:** Shows the output of the second code cell, which failed with an error message:

```
UnsupportedOperation Traceback  
<ipython-input-10-8194041e3286> in <module>  
      1 f = open("dati.txt","r")  
----> 2 f.write("contenuto da scrivere nel file")  
      3 f.close()  
  
UnsupportedOperation: not writable
```
- Search bar:** SEARCH STACK OVERFLOW

# Appending Data to an Existing File

- When open file with 'w' mode, if the file already exists it is overwritten
- To append data to a file use the 'a' mode
  - If file exists, it is not erased, and if it does not exist it is created
  - Data is written to the file at the end of the current contents

# write (con append)

+ Codice + Testo

```
[11] f = open("dati.txt","w")
     f.write("contenuto da scrivere nel file")
     f.close()
```

```
[10] f = open("dati.txt","r")
     f.write("contenuto da scrivere nel file")
     f.close()
```

```
-----  
UnsupportedOperation           Traceback  
<ipython-input-10-8194041e3286> in <module>  
      1 f = open("dati.txt","r")  
----> 2 f.write("contenuto da scrivere nel file")  
      3 f.close()
```

UnsupportedOperation: not writable

SEARCH STACK OVERFLOW

```
✓ ▶ f = open("dati.txt","a")
     f.write("contenuto da scrivere nel file")
     f.close()
```

dati.txt X

1 contenuto da scrivere nel file

1 contenuto da scrivere nel file

# write (righe multiple)

The screenshot shows a Jupyter Notebook environment with the following components:

- File Explorer:** On the left, it shows a tree view of files and folders. The 'sample\_data' folder is expanded, containing 'README.md', 'anscombe.json', 'california\_housing\_test.csv', 'california\_housing\_train.csv', 'mnist\_test.csv', 'mnist\_train\_small.csv', 'dati.txt', and 'nuovo\_file.txt'.
- Code Cell:** The main area contains three code cells:
  - Cell 10:** An error message: `[10] UnsupportedOperation`.  
Traceback:  

```
<ipython-input-10-8194041e3286> in <module>
      1 f = open("dati.txt","r")
----> 2 f.write("contenuto da scrivere nel file")
      3 f.close()
```

UnsupportedOperation: not writable

[SEARCH STACK OVERFLOW](#)
  - Cell 12:** A successful execution:  

```
[12]: f = open("dati.txt","a")
       f.write("contenuto da scrivere nel file")
       f.close()
```
  - Cell 0:** A successful execution:  

```
✓ 0 s   f = open("nuovo_file.txt","w")
       f.write("prima riga\n")
       f.write("seconda riga\n")
       f.close()
```
- Output:** To the right of the code cells, the output pane shows the contents of the 'nuovo\_file.txt' file:

```
1 prima riga
2 seconda riga
3
```

# Lettura da file

---

Le operazioni di lettura da file possono essere eseguite attraverso tre diverse funzioni built-in:

- `read`
- `readline`
- `readlines`

Queste tre funzioni restituiscono una parte o l'intero contenuto del file sotto forma di una stringa oppure di una lista di stringhe.

Il valore restituito dalle funzioni di lettura viene di norma memorizzato in una variabile per poter essere successivamente elaborato.

# Reading Data From a File

- read method: file object method that reads entire file contents into memory
  - Only works if file has been opened for reading
  - Contents returned as a string
- readline method: file object method that reads a line from the file
  - Line returned as a string, including '\n'
- Read position: marks the location of the next item to be read from a file

# read

---

La funzione `read` acquisisce l'intero contenuto di un file, che viene restituito sotto forma di una stringa.

Le eventuali interruzioni di riga presenti nel file vengono codificate con il carattere newline "`\n`".

**Sintassi:**

`variabile.read()`

dove `variabile` è la variabile associata al file.

# read

The screenshot shows a Jupyter Notebook interface with the following elements:

- File Bar:** Includes icons for file operations like Open, Save, and New.
- File Explorer:** Shows a tree view of files and folders: .., sample\_data, dati.txt, and nuovo\_file.txt.
- Code Cell 1:** Displays an error message:

```
[10] UnsupportedOperation

```
- Code Cell 2:** Displays successful execution:

```
[12] f = open("dati.txt","a")
      f.write("contenuto da scrivere nel file")
      f.close()
```
- Code Cell 3:** Displays successful execution:

```
[13] f = open("nuovo_file.txt","w")
      f.write("prima riga\n")
      f.write("seconda riga\n")
      f.close()
```
- Code Cell 4 (Currently Executing):** Displays the code being run:

```
f2 = open("nuovo_file.txt","r")
tutto_il_contenuto = f2.read()
f2.close()
print(tutto_il_contenuto)
```
- Output:** Shows the printed results:

```
prima riga
seconda riga
```
- Right Panel:** Shows the contents of the "nuovo\_file.txt" file:

```
1 prima riga
2 seconda riga
3
```

# readline

---

La funzione `readline` acquisisce una singola riga di un file, restituendola sotto forma di una stringa.

Per “riga” di un file si intende una sequenza di caratteri fino alla prima interruzione di riga, oppure, se il file non contiene interruzioni di riga, l’intera sequenza di caratteri contenuta in esso.

Nel primo caso anche l’interruzione di riga, codificata con il carattere `newline`, farà parte della stringa restituita da `readline`.

Sintassi:

`variabile.readline()`

dove `variabile` indica come al solito la variabile associata al file.

# readline

The screenshot shows a Jupyter Notebook interface with the following structure:

- File Bar:** Includes icons for File, Cell, Run, Kernel, and Help.
- File Explorer:** Shows a tree view with a red folder icon selected, containing:
  - ..
  - sample\_data
  - dati.txt
  - nuovo\_file.txt
- Code Editor:** Displays three code cells with their execution results:
  - [13] f.write("seconda riga\n")  
f.close()  
Output: prima riga  
seconda riga
  - [15] f2 = open("nuovo\_file.txt","r")  
tutto\_il\_contenuto = f2.read()  
f2.close()  
print(tutto\_il\_contenuto)  
Output: 1 prima riga  
2 seconda riga  
3
  - f3 = open("nuovo\_file.txt","r")  
una\_riga = f3.readline()  
f3.close()  
print(una\_riga)  
Output: prima riga
- Output Area:** Shows the final output of the third cell: "prima riga".

# Concatenating a Newline to and Stripping it From a String

- In most cases, data items written to a file are values referenced by variables
  - Usually necessary to concatenate a '\n' to data before writing it
    - Carried out using the + operator in the argument of the `write` method
- In many cases need to remove '\n' from string after it is read from a file
  - `rstrip` method: string method that strips specific characters from end of the string

# Concatenating a Newline to and Stripping it From a String

The screenshot shows a Python code editor with the following interface elements:

- Toolbar:** Includes icons for file operations (New, Open, Save, Find), run, and refresh.
- File Explorer:** Shows a directory structure with files: .., sample\_data, dati.txt, friends.txt, and nuovo\_file.txt.
- Code Editor:** Displays the following Python script:

```
# Questo programma riceve dall'utente tre nomi
# e li scrive in un file.

# Ottiene i tre nomi.
print('Inserisci i nomi di tre amici.')
name1 = input('Amico n. 1: ')
name2 = input('Amico n. 2: ')
name3 = input('Amico n. 3: ')

# Apre un file denominato friends.txt.
myfile = open('friends.txt', 'w')

# Scrive i nomi sul file.
myfile.write(name1 + '\n')
myfile.write(name2 + '\n')
myfile.write(name3 + '\n')

# Chiude il file.
myfile.close()
print('I nomi sono stati scritti in friends.txt.)
```
- Output Panel:** Shows the terminal output of the script execution:

```
Inserisci i nomi di tre amici.
Amico n. 1: Antonio
Amico n. 2: Luigi
Amico n. 3: Maria
I nomi sono stati scritti in friends.txt.
```
- File Preview:** On the right, there is a preview of the contents of the 'friends.txt' file, which contains:

```
1 Antonio
2 Luigi
3 Maria
4
```

# Writing and Reading Numeric Data

- Numbers must be converted to strings before they are written to a file
- str function: converts value to string
- Numbers are read from a text file as strings
  - Must be converted to numeric type in order to perform mathematical operations
  - Use `int` and `float` functions to convert string to numeric value

# Writing and Reading Numeric Data

The screenshot shows a Python code editor interface. On the left is a sidebar with icons for search, file operations, and a file tree containing files like sample\_data, dati.txt, friends.txt, numbers.txt, and nuovo\_file.txt. The main area displays a Python script with comments in Italian. The script reads three numbers from the user, writes them to a file named 'numbers.txt', and then prints a message. To the right, the contents of 'numbers.txt' are shown, listing the numbers 1, 23; 2, 45; 3, 67; and 4.

```
# Questo programma mostra in che modo
# si possono convertire numeri in stringhe
# prima di scriverli in un file di testo.

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

# Ottiene tre numeri dall'utente.
num1 = int(input('Inserisci un numero: '))
num2 = int(input('Inserisci un altro numero: '))
num3 = int(input("Inserisci l'ultimo numero: "))

# Scrive i numeri in un file.
outfile.write(str(num1) + '\n')
outfile.write(str(num2) + '\n')
outfile.write(str(num3) + '\n')

# Chiude il file.
outfile.close()
print('Dati scritti in numbers.txt')

Inserisci un numero: 23
Inserisci un altro numero: 45
Inserisci l'ultimo numero: 67
Dati scritti in numbers.txt
```

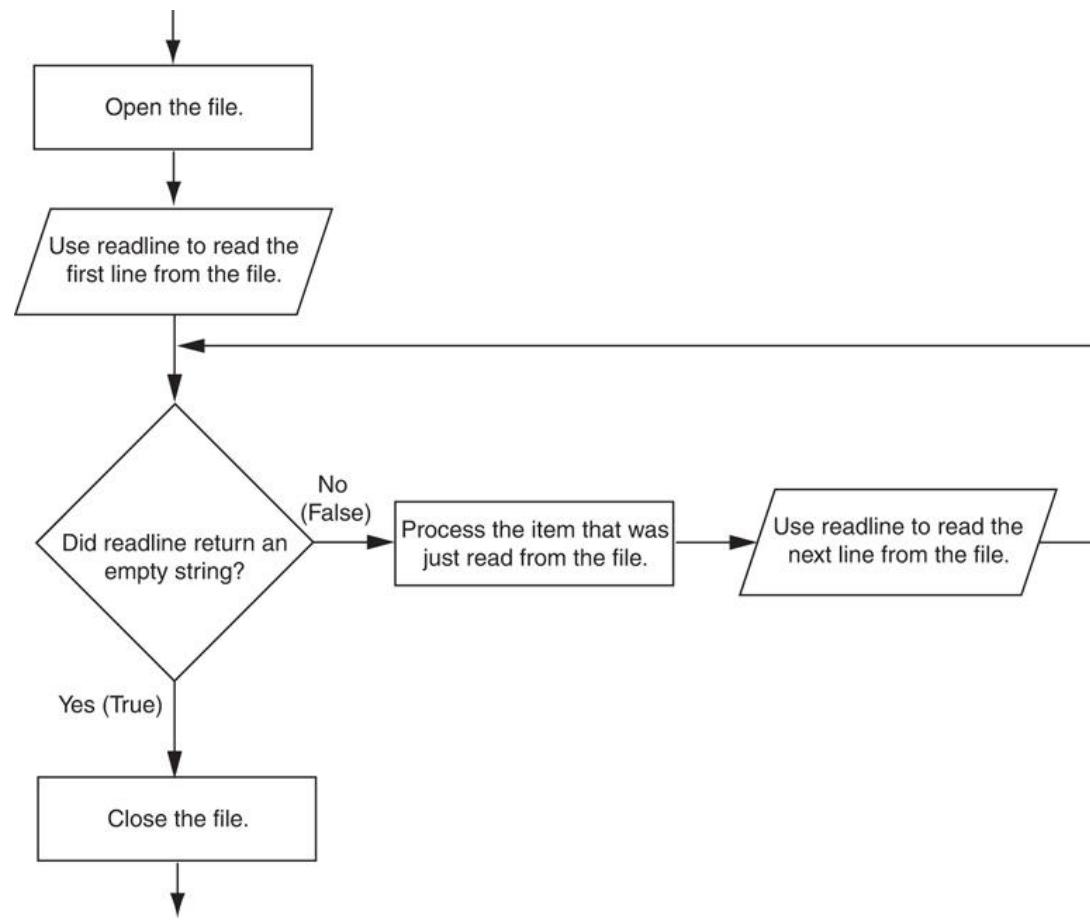
numbers.txt ×

1 23  
2 45  
3 67  
4

# Using Loops to Process Files (1 of 2)

- Files typically used to hold large amounts of data
    - Loop typically involved in reading from and writing to a file
  - Often the number of items stored in file is unknown
    - The `readline` method uses an empty string as a sentinel when end of file is reached
      - Can write a while loop with the condition
- ```
while line != ''
```

# Using Loops to Process Files (2 of 2)



**Figure 6-17** General logic for detecting the end of a file

# Using Loops to Process Files

The screenshot shows a Python code editor interface with two code cells and a file explorer sidebar.

**File Explorer:**

- File
- sample\_data
- dati.txt
- friends.txt
- numbers.txt
- nuovo\_file.txt

**Code Cell 1 (Top):**

```
[18] 11 s
# Chiude il file.
outfile.close()
print('Dati scritti in numbers.txt')

Inserisci un numero: 23
Inserisci un altro numero: 45
Inserisci l'ultimo numero: 67
Dati scritti in numbers.txt
```

**Code Cell 2 (Bottom):**

```
0 s
f = open("numbers.txt", "r")
riga = f.readline()
while riga != "":
    print(riga)
    riga = f.readline()

23
45
67
```

**Output Area:**

```
numbers.txt X
1 23
2 45
3 67
4
```

# Using Loops to Process Files

The screenshot shows a Jupyter Notebook interface with the following components:

- File Explorer:** On the left, it shows a tree view of files and folders. The 'sample\_data' folder is expanded, containing 'dati.txt', 'friends.txt', 'numbers.txt', and 'nuovo\_file.txt'. A red vertical bar highlights the 'numbers.txt' file.
- Codice Cell:** In the center, there is a code cell with a play button icon. The code reads:

```
f = open("numbers.txt", "r")
riga = f.readline()
while riga != "":
    print(riga.rstrip("\n"))
    riga = f.readline()
```

Below the code cell, the output is displayed:

```
23
45
67
```
- Testo Cell:** Above the code cell, there is a cell labeled '+ Testo' which contains the text 'numbers.txt' followed by its contents:

```
1 23
2 45
3 67
4
```

# Using Python's for Loop to Read Lines

- Python allows the programmer to write a `for` loop that automatically reads lines in a file and stops when end of file is reached
  - Format: `for line in file_object:`
  - statements
  - The loop iterates once over each line in the file

# Using Python's for Loop to Read Lines

```
[21] f = open("numbers.txt", "r")
     riga = f.readline()
     while riga != "":
         print(riga.rstrip("\n"))
         riga = f.readline()
```

```
23
45
67
```

```
f = open("numbers.txt", "r")
for riga in f:
    print(riga.rstrip("\n"))
```

```
23
45
67
```

numbers.txt X

```
1 23
2 45
3 67
4
```

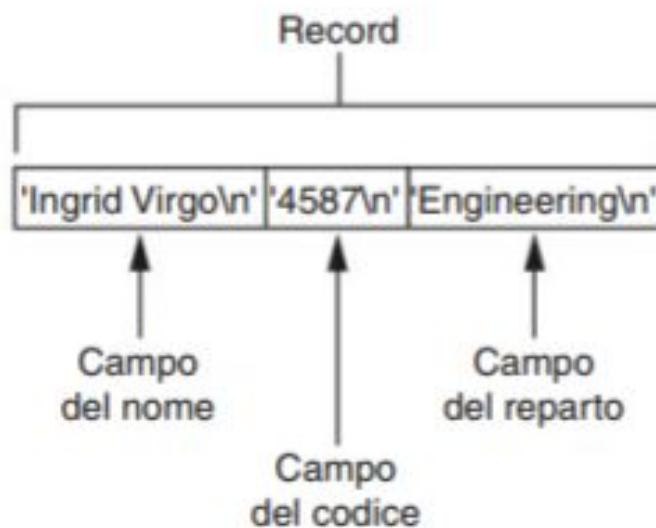
## Processing Records (1 of 2)

- Record: set of data that describes one item
- Field: single piece of data within a record
- Write record to sequential access file by writing the fields one after the other
- Read record from sequential access file by reading each field until record complete

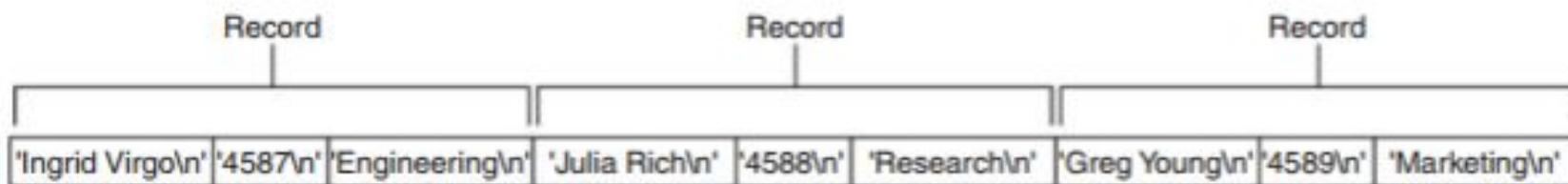
## Processing Records (2 of 2)

- When working with records, it is also important to be able to:
  - Add records
  - Display records
  - Search for a specific record
  - Modify records
  - Delete records

## Processing Records (2 of 2)



**Figura 6.18**  
I campi di un record.



**Figura 6.19**  
I record in un file.

## Exceptions (1 of 4)

- Exception: error that occurs while a program is running
  - Usually causes program to abruptly halt
- Traceback: error message that gives information regarding line numbers that caused the exception
  - Indicates the type of exception and brief description of the error that caused exception to be raised

# Eccezioni

---

Alcune possibili eccezioni per i programmi in Python sono:

- la divisione di un valore per zero, come:

```
print(55/0)
```

che produce

```
ZeroDivisionError: integer division or modulo
```

## Exceptions (2 of 4)

- Many exceptions can be prevented by careful coding
  - Example: input validation
  - Usually involve a simple decision construct
- Some exceptions cannot be avoided by careful coding
  - Examples
    - Trying to convert non-numeric string to an integer
    - Trying to open for reading a file that doesn't exist

## Exceptions (3 of 4)

- Exception handler: code that responds when exceptions are raised and prevents program from crashing

- In Python, written as `try/except` statement

- General format: `try:`

```
statements
except exceptionName:
    statements
```

- Try suite: statements that can potentially raise an exception
    - Handler: statements contained in `except` block

# FileNotFoundException

---



```
nome_file = input("Inserire il nome per il file da leggere: ")
try:
    f = open(nome_file, 'r')
except FileNotFoundError:
    print("Il file non esiste!")
```

⇨ Inserire il nome per il file da leggere: numeri.txt  
Il file non esiste!

## Exceptions (4 of 4)

- If statement in try suite raises exception:
  - Exception specified in except clause:
    - Handler immediately following except clause executes
    - Continue program after try/except statement
  - Other exceptions:
    - Program halts with traceback error message
- If no exception is raised, handlers are skipped

# Exceptions (4 of 4)

+ Codice + Testo

```
✓ ⏪ # Questo programma calcola la paga lorda.  
6 s  
  
try:  
    # Ottiene il numero di ore lavorate.  
    hours = int(input('Quante ore hai lavorato? '))  
  
    # Ottiene la retribuzione oraria.  
    pay_rate = float(input('Inserisci la retribuzione oraria: '))  
  
    # Calcola la paga lorda.  
    gross_pay = hours * pay_rate  
  
    # Visualizza la paga lorda.  
    print(f'Paga lorda: €{gross_pay:.2f}')  
  
except ValueError:  
    print('ERRORE: Ore lavorate e retribuzione oraria devono')  
    print('essere numeri validi.')  
  
↳ Quante ore hai lavorato? undici  
ERRORE: Ore lavorate e retribuzione oraria devono  
essere numeri validi.
```

# Handling Multiple Exceptions

- Often code in try suite can throw more than one type of exception
  - Need to write except clause for each type of exception that needs to be handled
- An except clause that does not list a specific exception will handle any exception that is raised in the try suite
  - Should always be last in a series of except clauses

# Displaying an Exception's Default Error Message

- Exception object: object created in memory when an exception is thrown
  - Usually contains default error message pertaining to the exception
  - Can assign the exception object to a variable in an `except` clause
    - Example: `except ValueError as err:`
  - Can pass exception object variable to `print` function to display the default error message

# The else Clause

- try/except statement may include an optional else clause, which appears after all the except clauses
  - Aligned with try and except clauses
  - Syntax similar to else clause in decision structure
  - Else suite: block of statements executed after statements in try suite, only if no exceptions were raised
    - If exception was raised, the else suite is skipped

# The finally Clause

- try/except statement may include an optional finally clause, which appears after all the except clauses
  - Aligned with try and except clauses
  - General format: finally:
    - statements
  - Finally suite: block of statements after the finally clause
    - Execute whether an exception occurs or not
    - Purpose is to perform cleanup before exiting

# What If an Exception Is Not Handled?

- Two ways for exception to go unhandled:
  - No except clause specifying exception of the right type
  - Exception raised outside a try suite
- In both cases, exception will cause the program to halt
  - Python documentation provides information about exceptions that can be raised by different functions

# Summary

- This chapter covered:
  - Types of files and file access methods
  - Filenames and file objects
  - Writing data to a file
  - Reading data from a file and determining when the end of the file is reached
  - Processing records
  - Exceptions, including:
    - Traceback messages
    - Handling exceptions

# *Corso di STATISTICA, INFORMATICA, ELABORAZIONE DELLE INFORMAZIONI*

*Modulo di Sistemi di Elaborazione delle  
Informazioni*

# File ed Eccezioni



**UNIVERSITÀ DEGLI STUDI  
DELLA BASILICATA**



Docente:  
**Domenico Daniele  
Bloisi**

