



Corso Pratico Python

[Modulo 1- 2026]

1. Introduzione all'ambiente Python
- 1.1. La sintassi : motivazioni al disegno del linguaggio

Richiedente: Dott. Roberto Felici [CNR-ISM] • Insegnante: Dott. Scigè J. Liu [INAF-IAPS]

1.1. La sintassi ... le stringhe

Le stringhe [str()] sono aggregati di caratteri – parenti dei byte-array

str() con " ":

```
str1 = "L'intelligenza artificiale non sostituirà l'estro umano."
```

str() con ' ':

```
str2 = 'L\'intelligenza artificiale non sostituirà l\'estro umano.'
```

```
str3 = 'ma con gli apici posso scrivere "citazioni"'
```

str() con """ ... """:

```
str4 = """
```

Con le triplici potrei:

- Andare a capo

```
"""
```

1.1. La sintassi ... le stringhe

Le stringhe [str()] sono aggregati di caratteri – parenti dei byte-array

str() con " ":

```
str1 = "L'intelligenza artificiale non sostituirà l'estro umano."
```

str() con ' ':

```
str2 = 'L\'intelligenza artificiale non sostituirà l\'estro umano.'
```

```
str3 = 'ma con gli apici posso scrivere "citazioni"'
```

str() con """ ... """:

```
str4 = """
```

Con le triplici potrei:

- Andare a capo

```
"""
```

str() con r' ':

```
str5 = r'c:\cartella\windows.dir'
```

str() con f' ':

```
str6 = f'apotema : {apotema:3f}'
```

1.1. La sintassi ... le stringhe

Le stringhe [str()] sono aggregati di caratteri – parenti dei byte-array

str() con " ":

```
str1 = "L'intelligenza artificiale non sostituirà l'estro umano."
```

str() con ' ':

```
str2 = 'L\'intelligenza artificiale non sostituirà l\'estro umano.'
```

```
str3 = 'ma con gli apici posso scrivere "citazioni"'
```

str() con """ ... """:

```
str4 = """
```

Con le triplici potrei:

str() con r' ':

```
str5 = r'c:\cartella\windows.dir'
```

```
- apotema = 12.3456789
```

```
"""
```

```
# Formattazione: 15 caratter, 6 decimali, tipo float (f)
```

```
percorso_file = fr"C:\Risultati\Dati_{apotema:15.6f}.csv"
```

1.1. La sintassi ... i commenti

I commenti sono parti dei corpus non fungibili

Commenti con `#` (detto in-linea):

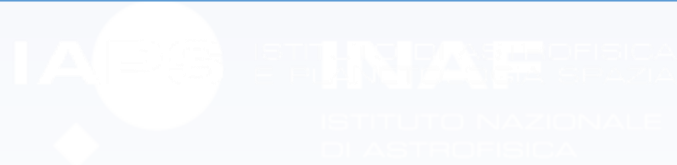
```
str1 = "testo di prova" # il commento non partecipa all'assegnazione
```

Commenti con `##` (detto documentato, alle volte con `!` o `<`):

```
mioPiGreco = np.arctan(1.0) * 4 ##< Calcolo di Pi Greco dinamicamente
```

```
##! Calcolo di Pi Greco dinamicamente  
mioPiGreco = np.arctan(1.0) * 4
```

– o interpretati dai plug-in



1.1. La sintassi ... i commenti

I commenti sono parti dei corpus non fungibili

Commenti con `#` (detto in-linea):

```
str1 = "testo di prova" # il commento non partecipa all'assegnazione
```

Commenti con `##` (detto documentato, alle volte con `!` o `<`):

```
mioPiGreco = np.arctan(1.0) * 4 ##< Calcolo di Pi Greco dinamicamente
```

```
##! Calcolo di Pi Greco dinamicamente  
mioPiGreco = np.arctan(1.0) * 4
```

– o interpretati dai plug-in

Separatore di sezione `###`:

usati nelle versioni «stream» per segmentare le parti del corpus,

utili per esecuzione report-in-linea, a-blocchi o debug

`###` sezione degli import

```
import numpy as np
```

`###` sezione delle definizioni di funzioni

```
def areaPoligono(lato, numLati, rapportoApotema):
```

```
    return lato*numLato*(lato*rapportoApotema)/2
```

1.1. La sintassi ... i commenti

I commenti sono parti dei corpus non fungibili

Commenti con `#` (detto in-linea):

```
str1 = "testo di prova" # il commento non partecipa all'assegnazione
```

Commenti con `##` (detto documentato, alle volte con `!` o `<`):

```
mioPiGreco = np.arctan(1.0) * 4 ##< Calcolo di Pi Greco dinamicamente
```

```
##! Calcolo di Pi Greco dinamicamente  
mioPiGreco = np.arctan(1.0) * 4
```

– o interpretati dai plug-in

Separatore di sezione `###`:

```
usati nelle versioni «stream» per segmentare le parti del  
corpus,
```

```
utili per esecuzione report-in-linea, a-blocchi o debug
```

```
### sezione degli import
```

```
import numpy as np
```

```
### sezione delle definizioni di funzioni
```

Dichiarazione Interprete (`#!`)

```
#!/usr/bin/env python3
```

Fattivamente interpretato dalla shell e non da python

Apotema):

otema)/2

1.1. La sintassi ... i commenti

I commenti sono parti dei corpus non fungibili

Commenti con `#` (detto in-linea):

```
str1 = "testo di prova" # il commento non partecipa all'assegnazione
```

Commenti con `##` (detto documentato, alle volte con `!` o `<`):

```
mioPiGreco = np.arctan(1.0) * 4 ##< Calcolo di Pi Greco dinamicamente
```

```
##! Calcolo di Pi Greco dinamicamente  
mioPiGreco = np.arctan(1.0) * 4
```

– o interpretati dai plug-in

Separatore di sezione `###`:

usati nelle versioni «stream» per segmentare le parti del corpus,

utili per esecuzione report-in-linea, a-blocchi o debug

`###` sezione degli import

```
import numpy as np
```

`###` sezione delle definizioni di funzioni

Dichiarazione Interprete (`#!`)

```
#!/usr/bin/env python3
```

Fattivamente interpretato dalla shell e non da python

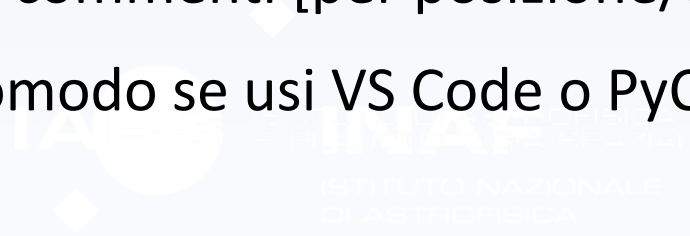
Apotema):

otema)/2

1.1. La sintassi ... i commenti

Riepilogo Tecnico

- **#!:** Inserito all'inizio per rendere lo script eseguibile su sistemi Unix.
- **#:** Usato per brevi spiegazioni del calcolo.
- **##:** Usato per evidenziare commenti [per posizione/enfasi].
- **###:** Usato per dividere (comodo se usi VS Code o PyCharm).





Corso Pratico Python

[Modulo 1- 2026]

1. Introduzione all'ambiente Python
- 1.2. Corpus, Moduli e Cartelle

Richiedente: Dott. Roberto Felici [CNR-ISM] • Insegnante: Dott. Scigè J. Liu [INAF-IAPS]

1.2. Corpus, Moduli e Cartelle

Corpus: elemento aggregato di elementi sintattici

```
C:\Windows\System32\cmd.e X + v
Microsoft Windows [Versione 10.0.26200.7840]
(c) Microsoft Corporation. Tutti i diritti riservati.

C:\P_Python>VirtualEnv\Scripts\activate

(VirtualEnv) C:\P_Python>
```

INAF
ISTITUTO NAZIONALE
DI ASTRONOMIA

Intervallo Tecnico 1.2.1 – Command-Line 1

Python

```
import os

# 1. Assicuriamoci che la cartella esista
os.makedirs("package1", exist_ok=True)

# 2. Scriviamo il contenuto nel file __init__.py
with open("package1/__init__.py", "w") as package1_init_file:
    # Definiamo il contenuto del file come stringa
    content = """
__version__ = '1.0.0'

if __name__ != '__main__':
    print("Esecuzione in forma import")
else:
    print("Esecuzione come script principale")
"""
    package1_init_file.write(content.strip())

print("File package1/__init__.py aggiornato!")
```

Intervallo Tecnico 1.2.1 – Command-Line 1

```

Python 3.13.2 | packaged by Anaconda, Inc. | (main, Feb 6 2025, 18:49:14)
[MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> content = """
... __version__ = '1.0.0'
... if __name__ != '__main__':
...     print("Esecuzione in forma import")
... else:
...     print("Esecuzione come script principale")
... """
>>> import os
>>> with open("package1/__init__.py", "w") as package1_init_file:
...     outFile=package1_init_file
...     print(content,file=outFile)
>>> print("File package1/__init__.py aggiornato!")
File package1/__init__.py aggiornato!
>>>
  
```



Intervallo Tecnico 1.2.1 – Command-Line 1

```
Python 3.13.2 | packaged by Anaconda, Inc. | (main, Feb 6 2025, 18:49:14)
[MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> content = """
... __version__ = '1.0.0'
... if __name__ != '__main__':
...     print("Esecuzione in forma import")
... else:
...     print("Esecuzione c
... """
>>> import os
>>> with open("package1/___
...     outFile=package1_in
...     print(content,file=
>>> print("File package1/___
File package1/___init__.py a
>>> |
```

```
(VirtualEnv) C:\P_Python\prova>python
Python 3.13.2 | packaged by Anaconda, Inc. | (main, Feb 6 2025, 18:49:14)
[MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import package1
Esecuzione in forma import
>>> |
```





Intervallo Tecnico 1.2.1 – Command-Line 1

The image displays three overlapping terminal windows. The top-left window shows a Python script with comments in Italian and code for version handling and file output. The top-right window shows the output of running this script, indicating it was packaged by Anaconda. The bottom window shows the execution of the script as a command-line program, resulting in the output 'Esecuzione come script principale'.

```
Python 3.13.2 | packaged by Anaconda, Inc. | (main, Feb 6 2025, 18:49:14)
[MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> content = """
... __version__ = '1.0.0'
... if __name__ != '__main__':
...     print("Esecuzione in forma import")
... else:
...     print("Esecuzione c
... """
>>> import os
>>> with open("package1/___
...     outFile=package1_in
...     print(content,file=
>>> print("File package1/___
File package1/___init__.py a
>>> |

(VirtualEnv) C:\P_Python\prova>python
Python 3.1
[MSC v.19
Type "help
>>> import
Esecuzione
>>> |

(VirtualEnv) C:\P_Python\prova>python package1\__init__.py
Esecuzione come script principale

(VirtualEnv) C:\P_Python\prova>
```



Intervallo Tecnico 1.2.2 – Command-Line 2

```

C:\P_Python\prova\package1\__init__.py - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
__init__.py
1 #
2 __version__ = '1.2.3'
3 if __name__ != '__main__':
4     print("Esecuzione in forma import [__init__]")
5 else:
6     print("Esecuzione come script principale [__init__]")
7 #

__main__.py
1 #if inside package1:
2 from . import __version__ as package1_version
3
4 __version__ = '4.5.6'
5 if __name__ == '__main__':
6     print("Esecuzione come script principale [__main__]")
7     print(f"package1_main : {__version__}")
8     print(f"package1_version : {package1_version}")
9 #
    
```

```

C:\Windows\System32\cmd.e
(VirtualEnv) C:\P_Python\prova>python -m package1
Esecuzione in forma import [__init__]
Esecuzione come script principale [__main__]
package1_main      : 4.5.6
package1_version   : 1.2.3

(VirtualEnv) C:\P_Python\prova>
    
```

1.2. Corpus, Moduli e Cartelle - esempio import 1

Cartella/File: ./geometria/formule.py e ./main.py

```
# cartella /geometria/
```

```
# formule.py
```

```
def area_rettangolo(base, altezza):  
    return base * altezza
```

```
# cartella .
```

```
# main.py
```

```
from geometria.formule import area_rettangolo as ar  
base = 10  
altezza = 5  
print(f"L'area calcolata è: {ar(base, altezza)}")
```

```
#
```

INAF
ISTITUTO NAZIONALE
DI ASTRONOMIA
E FISICA

1.2. Corpus, Moduli e Cartelle - esempio import 2

Cartella/File: ./geometria/formule.py e ./main.py

```
# cartella /geometria/  
# formule.py  
def area_rettangolo(base, altezza):  
    return base * altezza
```

```
# cartella .  
# main.py  
from geometria.formule import area_rettangolo as ar  
base = '10'  
altezza = 5  
print(f"L'area calcolata è: {ar(base, altezza)}")  
  
#
```

INAF
ISTITUTO NAZIONALE
DI ASTRONOMIA

1.2. Corpus, Moduli e Cartelle - esempio import 3 + tipi

Cartella/File: ./geometria/formule.py e ./main.py

```
# cartella /geometria/
```

```
# formule.py
def area_rettangolo(base: float, altezza: float) ->
float:
    """
    Calcola l'area di un rettangolo.
    :param base: La base del rettangolo (float)
    :param altezza: L'altezza del rettangolo (float)
    :return: L'area calcolata (float)
    """
    return base * altezza
```

```
# cartella .
```

```
# main.py
from geometria.formule import
    area_rettangolo as ar

base = '10'
altezza = 5
print(f"L'area calcolata è:
      {ar(base, altezza)}")

#
```

1.2. Corpus, Moduli e Cartelle - esempio import 3 + casting

• Arricchimento Contenuti: Gestione degli Errori

```
# cartella /geometria/
```

```
# formule.py
```

```
def area_rettangolo(base, altezza):  
    return base * altezza
```

def FUNZIONE ():
 pass

Passaggio ARGOMENTI

Valore di RITORNO

```
# cartella .
```

```
# main.py
```

```
from geometria.formule import area_rettangolo as ar
```

```
base = 10
```

```
altezza = 5
```

```
try:
```

```
    base = float(input("Inserisci base : "))
```

```
    altezza = float(input("Inserisci altezza : "))
```

```
except ValueError:
```

```
    print("Errore: devi inserire valori numerici!")
```

```
print(f"L'area calcolata è: {ar(base, altezza)}")
```

```
#
```

1.2. Corpus, Moduli e Cartelle - esempio



Spyder (Python 3.13)

File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\P_Python\prova

C:\P_Python\prova\geometria\formule.py

temp.py x formule.py x main.py x

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Sun Feb 22 23:05:55 2026
4
5 @author: scige
6 """
7
8 def area_rettangolo(base, altezza):
9     return base * altezza
10
```

C:\P_Python\prova\main.py

temp.py x formule.py x main.py x

```
4
5 @author: scige
6 """
7
8 # main.py
9 from geometria.formule import area_rettangolo as ar
10 base = 10
11 altezza = 5
12 print(f"L'area calcolata è: {ar(base, altezza)}")
13
14
```

File Explorer

Name	Date Modified
> __pycache__	19/02/2026 00:50
> _avast_	22/02/2026 23:05
> geometria	22/02/2026 23:06
> __pycache__	22/02/2026 23:06
formule.py	22/02/2026 23:06
> package1	22/02/2026 22:53
main.py	22/02/2026 23:06
main1.py	22/02/2026 23:06

Help Variable Explorer Debugger Plots Files

Console 1/A x

```
Python 3.13.2 | packaged by Anaconda, Inc. | (main, Feb 6 2025, 18:49:14) [MSC
v.1929 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.
IPython 8.30.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: %runfile C:/P_Python/prova/main.py --wdir
L'area calcolata è: 50

In [2]:
```

IPython Console History

Update available Conda: miniconda313 (Python 3.13.2) LSP: Python Line 8, Col 10 UTF-8 CRLF RW Mem 42%



1.2. Corpus, Moduli e Cartelle - esempio



Spyder (Python 3.13)

File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\P_Python\prova\geometria\formule.py

temp.py x formule.py x main.py x

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Sun Feb 22 23:05:55 2026
4
5 @author: scige
6 """
7
8 def area_rettangolo(base, altezza):
9     return base * altezza
10
```

C:\P_Python\prova\main.py

temp.py x formule.py x main.py x

```
4
5 @author: scige
6 """
7
8 # main.py
9 from geometria.formule import area_rettangolo as ar
10 base = "10"
11 altezza = 5
12 print(f"L'area calcolata è: {ar(base, altezza)}")
13
14
```

File Explorer

Name	Date Modified
> __pycache__	19/02/2026 00:50
> _avast_	22/02/2026 23:05
> geometria	22/02/2026 23:06
> __pycache__	22/02/2026 23:06
formule.py	22/02/2026 23:06
> package1	22/02/2026 22:53
main.py	22/02/2026 23:09
main1.py	22/02/2026 23:06

Help Variable Explorer Debugger Plots Files

Console 1/A x

IPython 8.30.0 -- An enhanced Interactive Python. Type '?' for help.

```
In [1]: %runfile C:/P_Python/prova/main.py --wdir
L'area calcolata è: 50

In [2]: %runfile C:/P_Python/prova/main.py --wdir
Reloaded modules: geometria.formule
L'area calcolata è: 1010101010

In [3]:
```

IPython Console History

Update available Conda: miniconda313 (Python 3.13.2) LSP: Python Line 14, Col 1 UTF-8 CRLF RW Mem 42%



1.2. Corpus, Moduli e Cartelle - esempio



The screenshot displays the Spyder Python IDE interface. The top menu bar includes File, Edit, Search, Source, Run, Debug, Consoles, Projects, Tools, View, and Help. The toolbar contains icons for file operations, running, and debugging. The main editor area is divided into two panes. The top pane shows the file explorer for the project 'C:\P_Python\prova', listing directories like __pycache__ and __avast__, and files like formule.py, main.py, and main1.py. The bottom pane shows the code editor for 'formule.py', containing a function definition for 'area Rettangolo'. The console at the bottom shows the execution of the code, displaying the output 'L'area calcolata è: 1010101010'.

```
@author: scige
"""
def area_Rettangolo(base: float, altezza: float) -> float:
    """
    Calcola l'area di un rettangolo.
    :param base: La base del rettangolo (float)
    :param altezza: L'altezza del rettangolo (float)
    :return: L'area calcolata (float)
    """
    return base * altezza
```

```
# main.py
from geometria.formule import area_Rettangolo as ar
base = "10"
altezza = 5
print(f"L'area calcolata è: {ar(base, altezza)}")
```

Console 1/A X

```
In [2]: %runfile C:/P_Python/prova/main.py --wdir
Reloaded modules: geometria.formule
L'area calcolata è: 1010101010

In [3]: %runfile C:/P_Python/prova/geometria/formule.py --wdir
Reloaded modules: geometria.formule

In [4]: %runfile C:/P_Python/prova/main.py --wdir
L'area calcolata è: 1010101010

In [5]:
```

IPython Console History

Update available Conda: miniconda313 (Python 3.13.2) LSP: Python Line 13, Col 8 UTF-8 CRLF RW Mem 42%



1.2. Corpus, Moduli e Cartelle - esempio



Spyder (Python 3.13)

File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\P_Python\prova

C:\P_Python\prova\geometria\formule.py

```
4
5 @author: scige
6 """
7 def area_rettangolo(base: float, altezza: float) -> float:
8     """
9     Calcola l'area di un rettangolo.
10    :param base: La base del rettangolo (float)
11    :param altezza: L'altezza del rettangolo (float)
12    :return: L'area calcolata (float)
13    """
14    return base * altezza
15
```

C:\P_Python\prova\main.py

```
6 """
7 # main.py
8 from geometria.formule import area_rettangolo as ar
9 base = 10
10 altezza = 5
11 try:
12     base = float(input("Inserisci base : "))
13     altezza = float(input("Inserisci altezza : "))
14 except ValueError:
15     print("Errore: devi inserire valori numerici!")
16 print(f"L'area calcolata è: {ar(base, altezza)}")
17 #
```

Console 1/A

```
In [4]: %runfile C:/P_Python/prova/main.py --wdir
L'area calcolata è: 1010101010

In [5]: %runfile C:/P_Python/prova/main.py --wdir
Reloaded modules: geometria.formule
Inserisci base : "564654"
Errore: devi inserire valori numerici!
L'area calcolata è: 50

In [6]: |
```

IPython Console History

Update available Conda: miniconda313 (Python 3.13.2) LSP: Python Line 11, Col 5 UTF-8 CRLF RW Mem 43%



1.2. Corpus, Moduli e Cartelle - esempio



CORPUS IMMEDIATO : L'input immediato dell'interprete è assistito da interazioni di formattazioni

```
C:\Windows\System32\cmd.e x + v - □ x
Python 3.13.2 | packaged by Anaconda, Inc. | (main, Feb 6 2025, 18:49:14)
[MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> if True:
...     print("comanda già indentata")
...     #riga vuota
...
comanda già indentata
>>> |
```

è possibile interagire tramite «discovery» [TAB+TAB]

```
C:\Windows\System32\ x + v - □ x
>>> import os
>>> os.|
os.DirEntry() os.getcwd()
os.EX_OK os.getcwdb()
os.F_OK os.getenv()
os.GenericAlias() os.getlogin()
os.Mapping() os.getpid()
os.MutableMapping() os.getppid()
os.O_APPEND os.isatty(
132 more...
```

La discovery implementa una «auto-complete»

```
C:\Windows\System32\cmd.e x + v - □ x
>>> os.get_|
os.get_blocking( os.get_handle_inheritable( os.get_terminal_size(
os.get_exec_path( os.get_inheritable(
```



1.2. Corpus, Moduli e Cartelle

CORPUS IN FILE – script codificato «stoccato» in files

← Il MODULO è sinonimo di «FILE» [standard C]
 ↓ Le cartelle sono assimilate ai Packages

```
C:\P_Python\prova\geometria\formule.py

temp.py ×  formule.py ×  main.py ×

4
5  @author: scige
6  """
7  def area Rettangolo(base: float, altezza: float) -> float:
8      """
9      Calcola l'area di un rettangolo.
10     :param base: La base del rettangolo (float)
11     :param altezza: L'altezza del rettangolo (float)
12     :return: L'area calcolata (float)
13     """
14     return base * altezza
15
```

Name	Date Modified
> _avast_	22/02/2026 23:05
✓ geometria	22/02/2026 23:09
> _pycache_ formule.py	22/02/2026 23:09
✓ package 1	22/02/2026 22:53
> _pycache_ __init__.py __main__.py	22/02/2026 22:59
main.py	22/02/2026 23:11
main1.py	22/02/2026 23:06

```
my_project/
├── main_app.py           # Top-level script
├── parent_package/      # The "Module" or Parent Package
│   ├── __init__.py      # Initializer for parent
│   └── sub_package/     # The Nested Package
│       ├── __init__.py  # Initializer for sub-package
│       └── __main__.py  # Entry point for sub-package
```

← La nidificazione di packages è per cartelle [tipo Java]



1.2. Corpus, Moduli e Cartelle – definizioni dinamiche



CORPUS DINAMICI – la definizione di funzioni/metodi è il risultato di interpretazioni del flusso di comandi in input. Conseguenza → è possibile definire funzioni diverse in base all'input e quindi variegare l'effettivo comportamento

Definizione-In-Selettore ↓

```
mode = "advanced"

if mode == "advanced":
    def logic(x):
        return x ** 2
else:
    def logic(x):
        return x + 1

# Ora puoi chiamare 'logic' normalmente
print(logic(5)) # Output: 25
```

```
print(logic(5)) # Output: 25
# Ora puoi chiamare 'logic' normalmente
```

Definizione-In-Selettore ↓

```
def make_multiplier(n):
    def multiplier(x):
        return x * n
    return multiplier

use_triple = True

if use_triple:
    my_func = make_multiplier(3)
else:
    my_func = make_multiplier(2)

print(my_func(10)) # Output: 30
```

```
print(my_func(10)) # Output: 30
my_func = make_multiplier(5)
```

Definizione-In-Selettore ↓

```
use_cloud = True

if use_cloud:
    class Storage:
        def save(self, data):
            print("Salvataggio su AWS S3...")
else:
    class Storage:
        def save(self, data):
            print("Salvataggio su disco locale...")

# L'utente usa sempre 'Storage' senza sapere quale sia stata scelta
db = Storage()
db.save("test_data")
```

```
db.save("test_data")
db = Storage()
# L'utente usa sempre 'Storage' senza sapere quale sia stata scelta
```

1.2. Corpus, Moduli e Cartelle - decoratori



DECORATORI sono strumenti più eleganti. Un decoratore è una funzione che "avvolge" un'altra per modificarne il comportamento, non cambiare il codice originale. La funzione risultante risulta con funzionalità aggiuntive.

```
# -*- coding: utf-8 -*-
"""
Created on Sun Feb 22 23:34:21 2026

@author: scige
"""

from functools import wraps

# 1. Definiamo il decoratore che accetta il parametro N
def ripeti(n):
    def decoratore(func):
        @wraps(func)
        def wrapper(*args, **kwargs):
            # Eseguiamo la funzione originale N volte
            for _ in range(n):
                func(*args, **kwargs)
            return wrapper
        return decoratore

# 2. Usiamo il decoratore con la sintassi @
@ripeti(5)
def mioPrint(testo):
    print(testo)

# 3. Test dell'esecuzione
mioPrint("Ciao! Questa riga apparirà 5 volte.")
```

> _avast_ 22/02/2026 23:05

▼ geometria 22/02/2026 23:09

> > _pycache_ 22/02/2026 23:09

> > > formule.py 22/02/2026 23:09

▼ package1 22/02/2026 22:53

> > _pycache_ 22/02/2026 22:59

> > > __init__.py 22/02/2026 22:57

> > > __main__.py 22/02/2026 22:58

> > main.py 22/02/2026 23:11

> > main1.py 22/02/2026 23:06

Help Variable Explorer Debugger Plots Fil

Console 1/A X

Ciao! Questa riga apparirà 5 volte.
Ciao! Questa riga apparirà 5 volte.
Ciao! Questa riga apparirà 5 volte.
In [13]: %runfile C:/P_Python/prova/untitled1.py --wdir
Ciao! Questa riga apparirà 5 volte.
Ciao! Questa riga apparirà 5 volte.
Ciao! Questa riga apparirà 5 volte.
Ciao! Questa riga apparirà 5 volte.
Ciao! Questa riga apparirà 5 volte.
In [14]:

