Corso Pratico Python

[Modulo 1- 2026]

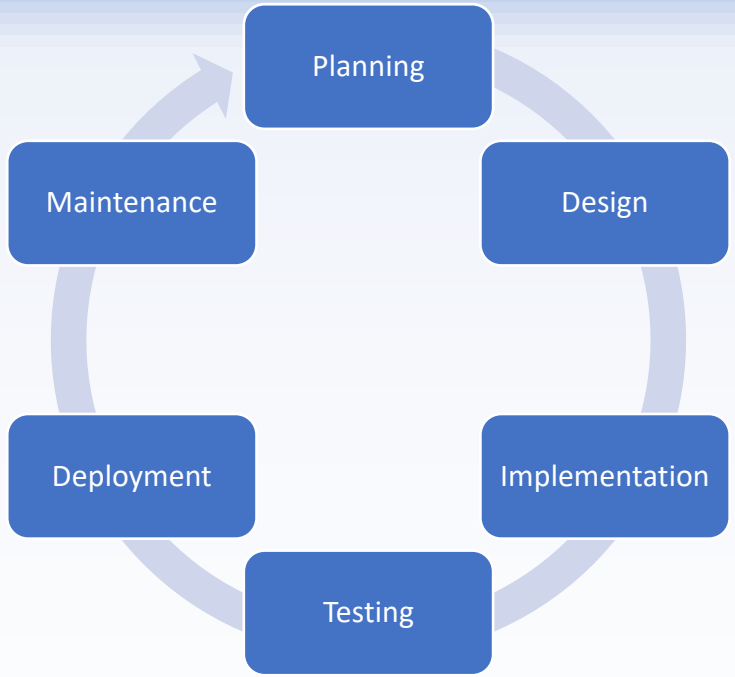
1. Introduzione all'ambiente Python
1.1. La sintassi : motivazioni al disegno del linguaggio

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

1. Introduzione all'ambiente Python

Il ciclo di Sviluppo : Software Development Life Cycle (SDLC)

Planning	: Identifying project goals and requirements.
Design	: Creating architecture and design specifications.
Implementation	: Writing and testing the code.
Testing	: Ensuring the software meets quality standards.
Deployment	: Releasing the software to users.
Maintenance	: Ongoing support and updates to the software



Python è stato progettato per massimizzare la produttività del programmatore.

"Il codice viene letto molto più spesso di quanto venga scritto".

1.1. La sintassi: Motivazioni al disegno del linguaggio

La sintassi di Python si basa sulla **PEP 20 (The Zen of Python)**.

Le scelte stilistiche che vedremo derivano da principi cardine:

Identazione Significativa: A differenza di C++ o Java che usano le graffe {}, Python usa gli spazi. Questo obbliga ogni sviluppatore a scrivere codice ordinato e visivamente coerente.

Minimalismo: Meno simboli (niente punti e virgola ; a fine riga) per ridurre il rumore visivo.

Tipizzazione Dinamica ma Forte: Non serve dichiarare se una variabile è un numero o un testo, ma Python non ti permetterà mai di sommare "2" (testo) con 2 (numero) senza una conversione esplicita.

1.1. La sintassi: Motivazioni al disegno del linguaggio

La sintassi di Python si basa sulla **PEP 20 (The Zen of Python)**.

https://peps.python.org/pep-0020/

1. Beautiful is better than ugly.
2. Explicit is better than implicit.
3. Simple is better than complex.
4. Complex is better than complicated.
5. Flat is better than nested.
6. Sparse is better than dense.
7. Readability counts.
8. Special cases aren't special enough to break the rules.
9. Although practicality beats purity.
10. Errors should never pass silently.
11. Unless explicitly silenced.
12. In the face of ambiguity, refuse the temptation to guess.
13. There should be one-- and preferably only one --obvious way to do it.
14. Although that way may not be obvious at first unless you're Dutch.
15. Now is better than never.
16. Although never is often better than *right* now.
17. If the implementation is hard to explain, it's a bad idea.
18. If the implementation is easy to explain, it may be a good idea.
19. Namespaces are one honking great idea -- let's do more of those!

>>> import this

1.2. Corpus, Moduli e Cartelle

Organizzare un algoritmo non significa solo scrivere una funzione, ma strutturare il progetto perché sia scalabile e manutenibile.

Il Corpus (Script):

input diretto dell'interprete

input da file '.py' dove risiede la logica.

I Moduli:

gli input a Python [diretti o file] che contengono definizioni e istruzioni. Permettono di "spacchettare" la complessità.

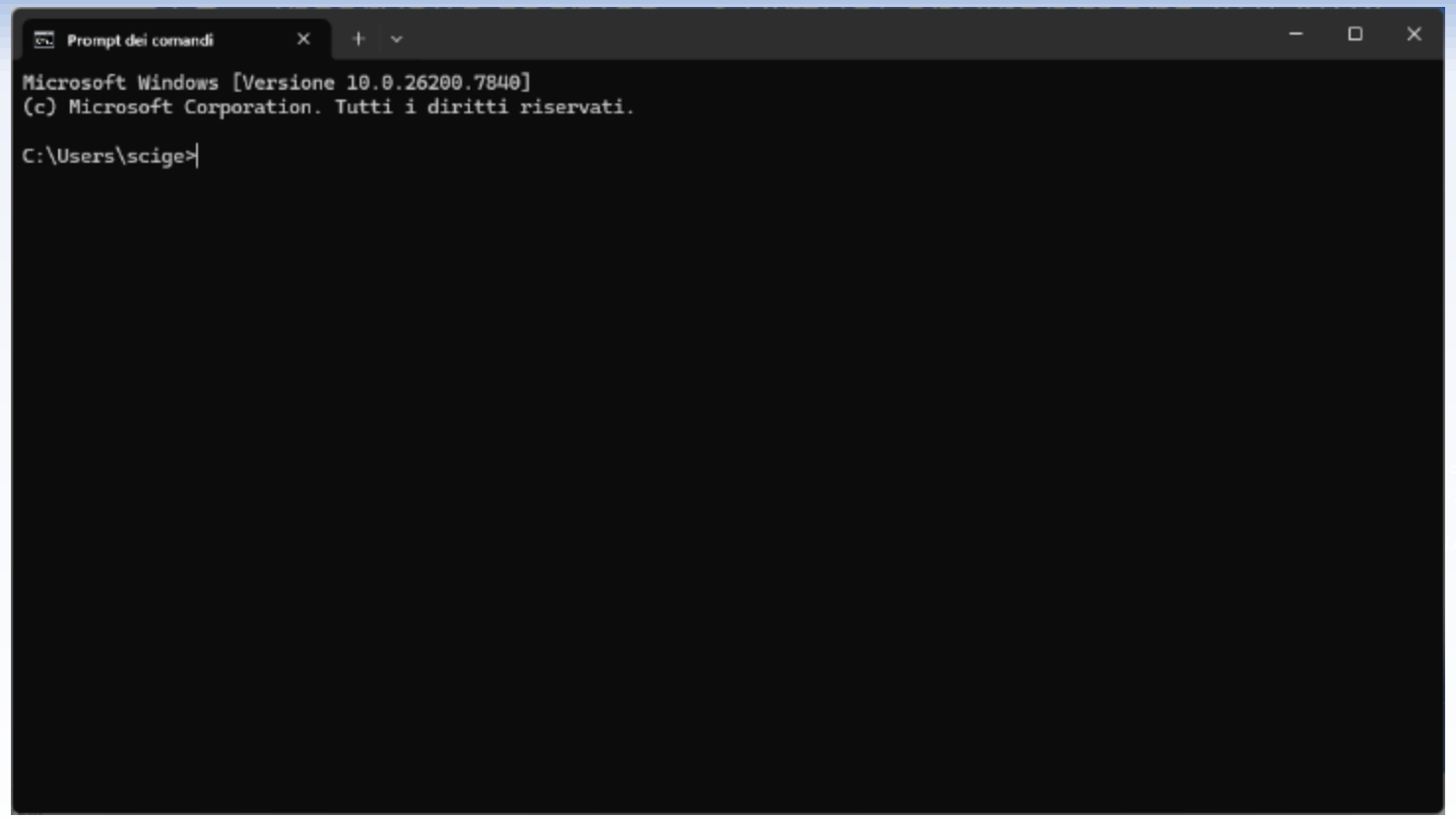
I Package (Cartelle):

Una cartella diventa un pacchetto Python quando contiene un file __init__.py.
Questo permette la sintassi import cartella.modulo ; che importa funzionalità dei moduli all'interno della cartella

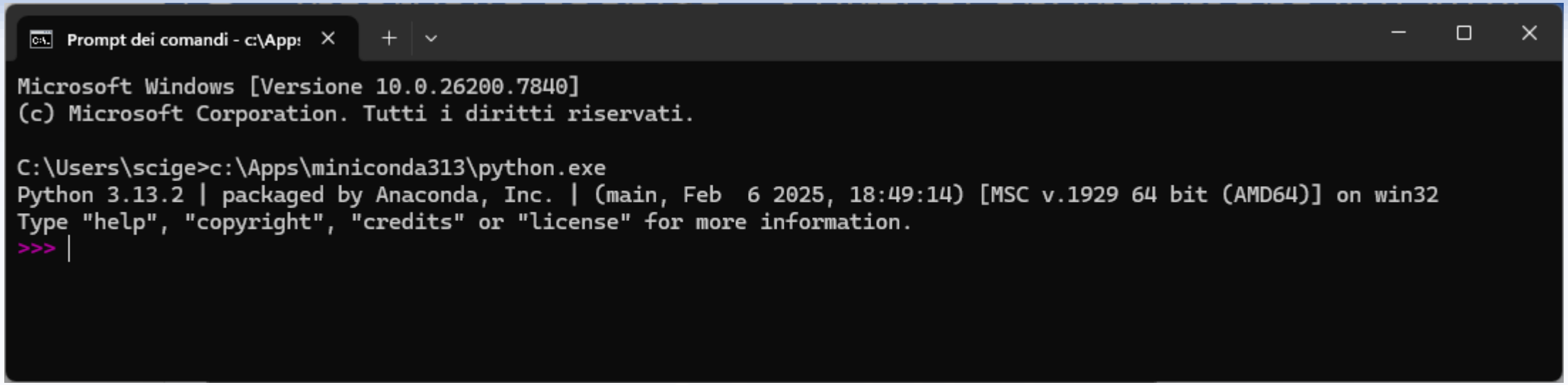
E le classi????

→ Abbiamo pianificato come farci-più-male più avanti ←

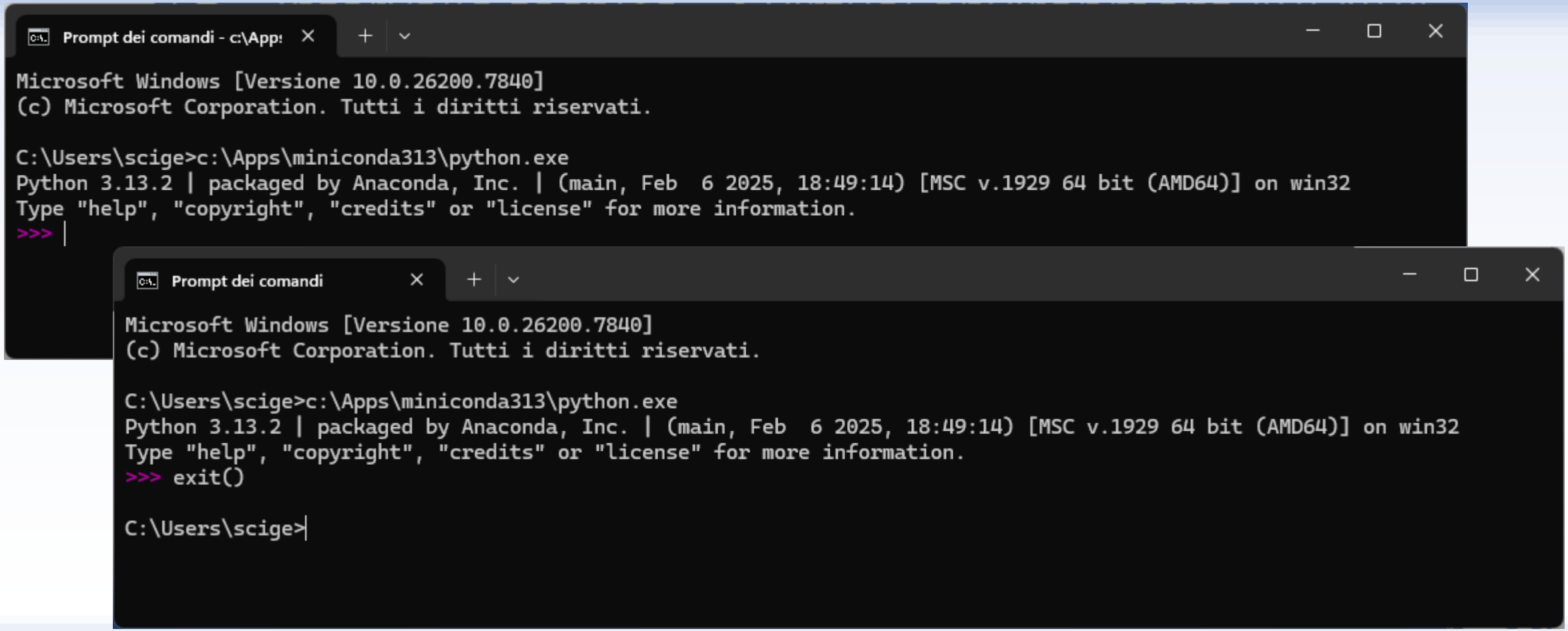
Intervallo tecnico → virtual environment «VENV»



Intervallo tecnico → virtual environment «VENV»



Intervallo tecnico → virtual environment «VENV»



Intervallo tecnico → virtual environment «VENV»

Prompt dei co

Microsoft Wi
(c) Microsof

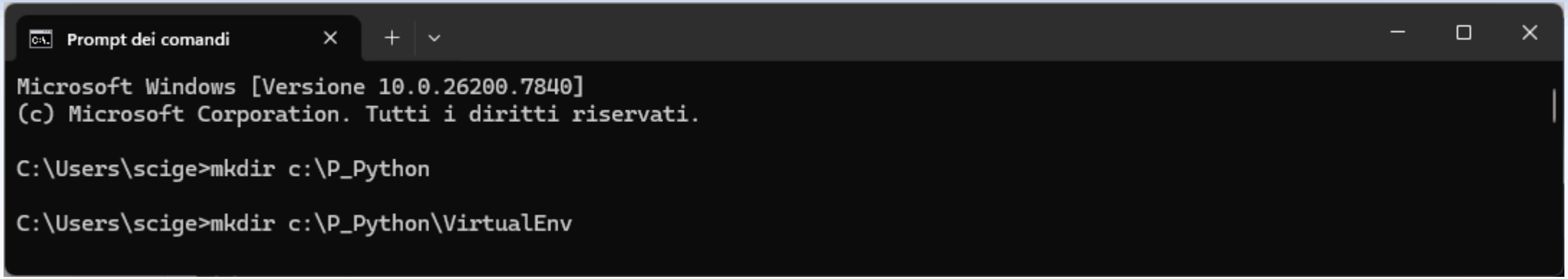
C:\Users\sci

C:\Users\sci

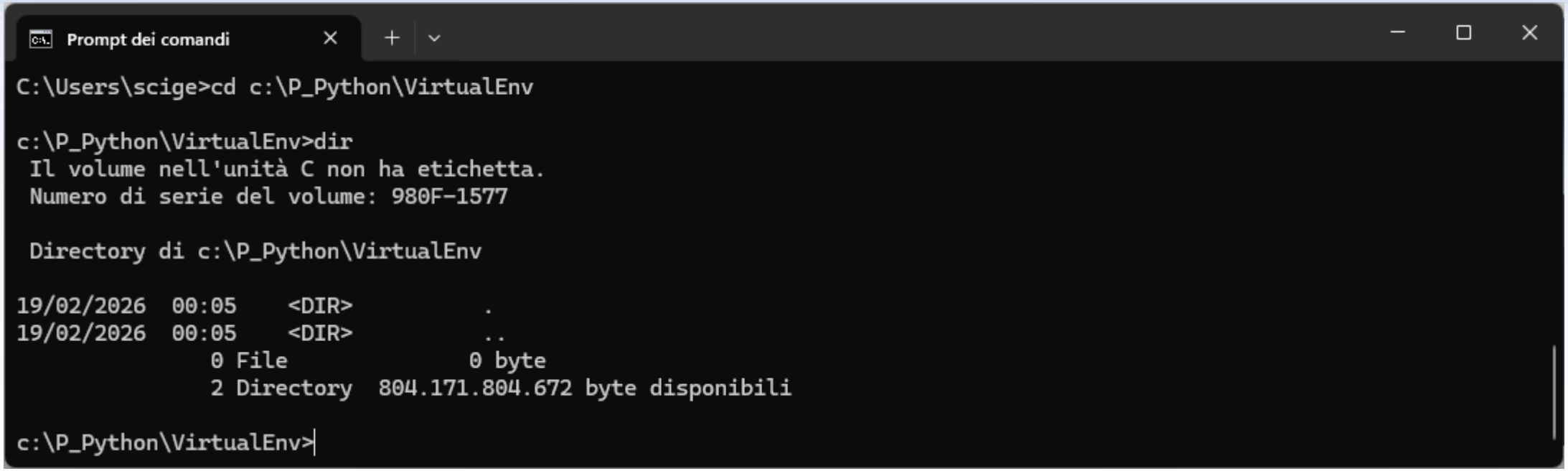
Prompt dei comandi

```
C:\Users\scige>c:\Apps\miniconda313\python.exe -m venv --help
usage: venv [-h] [--system-site-packages] [--symlinks | --copies] [--clear] [--upgrade] [--without-pip]
            [--prompt PROMPT] [--upgrade-deps] [--without-scm-ignore-files]
            ENV_DIR [ENV_DIR ...]

Creates virtual Python environments in one or more target directories.

positional arguments:
  ENV_DIR               A directory to create the environment in.

options:
  -h, --help            show this help message and exit
  --system-site-packages
                        Give the virtual environment access to the system site-packages dir.
  --symlinks            Try to use symlinks rather than copies, when symlinks are not the default for the platform.
  --copies              Try to use copies rather than symlinks, even when symlinks are the default for the platform.
  --clear               Delete the contents of the environment directory if it already exists, before environment
                        creation.
  --upgrade             Upgrade the environment directory to use this version of Python, assuming Python has been
                        upgraded in-place.
  --without-pip         Skips installing or upgrading pip in the virtual environment (pip is bootstrapped by
                        default)
  --prompt PROMPT       Provides an alternative prompt prefix for this environment.
  --upgrade-deps        Upgrade core dependencies (pip) to the latest version in PyPI
  --without-scm-ignore-files
                        Skips adding SCM ignore files to the environment directory (Git is supported by default).

Once an environment has been created, you may wish to activate it, e.g. by sourcing an activate script in its bin
directory.

C:\Users\scige>
```

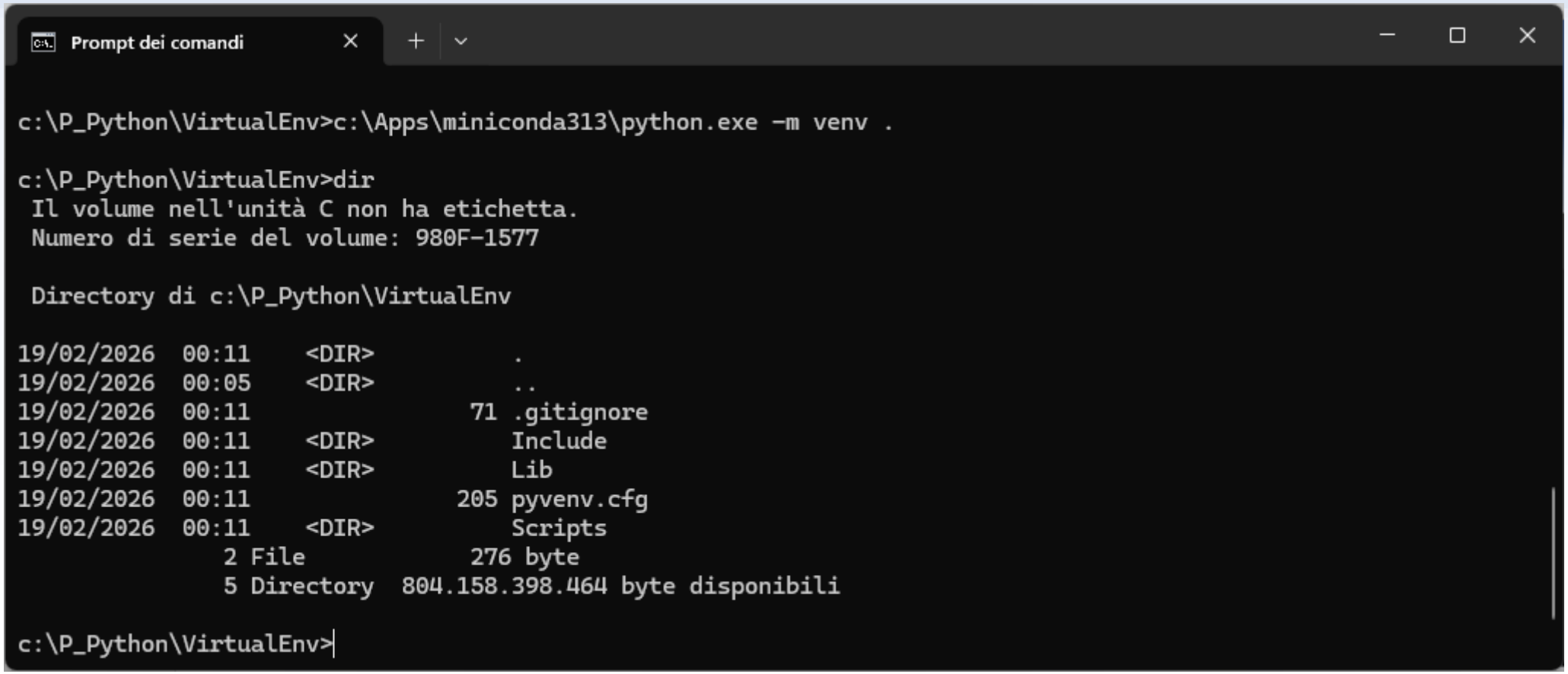
Intervallo tecnico → virtual environment «VENV»



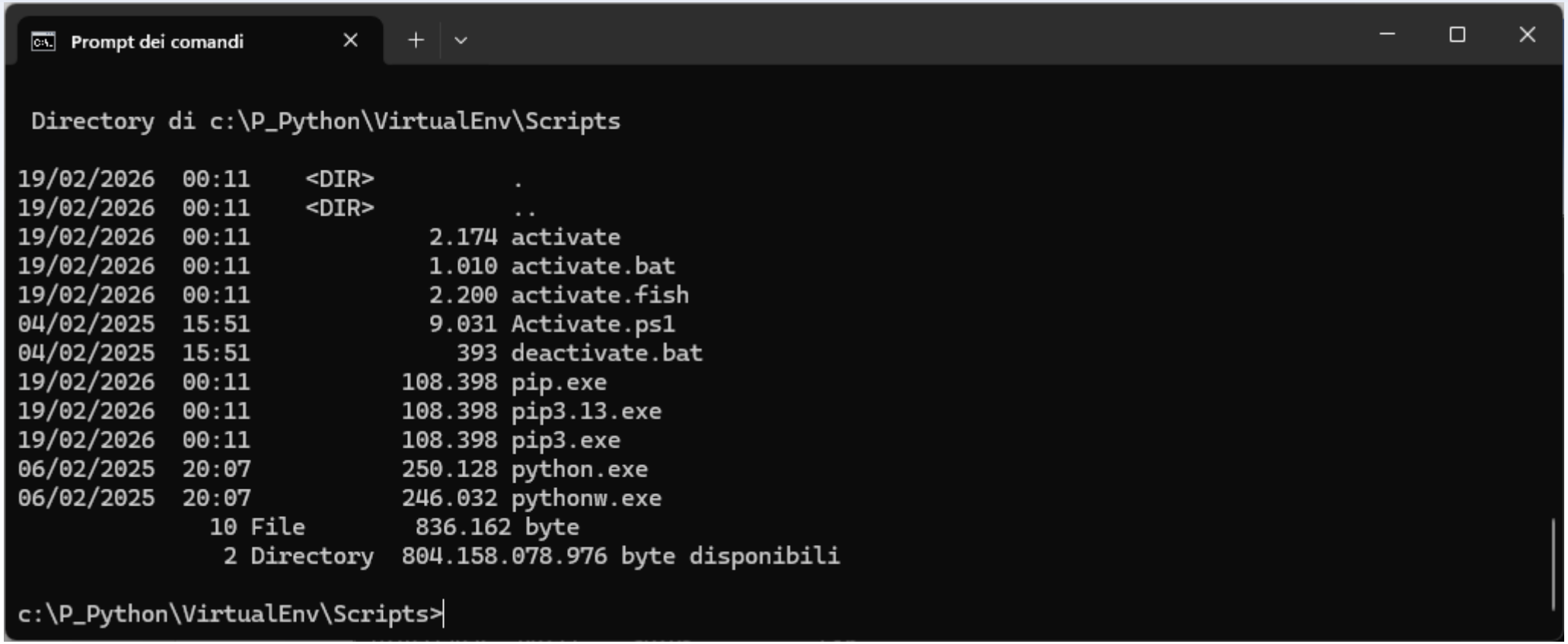
Intervallo tecnico → virtual environment «VENV»



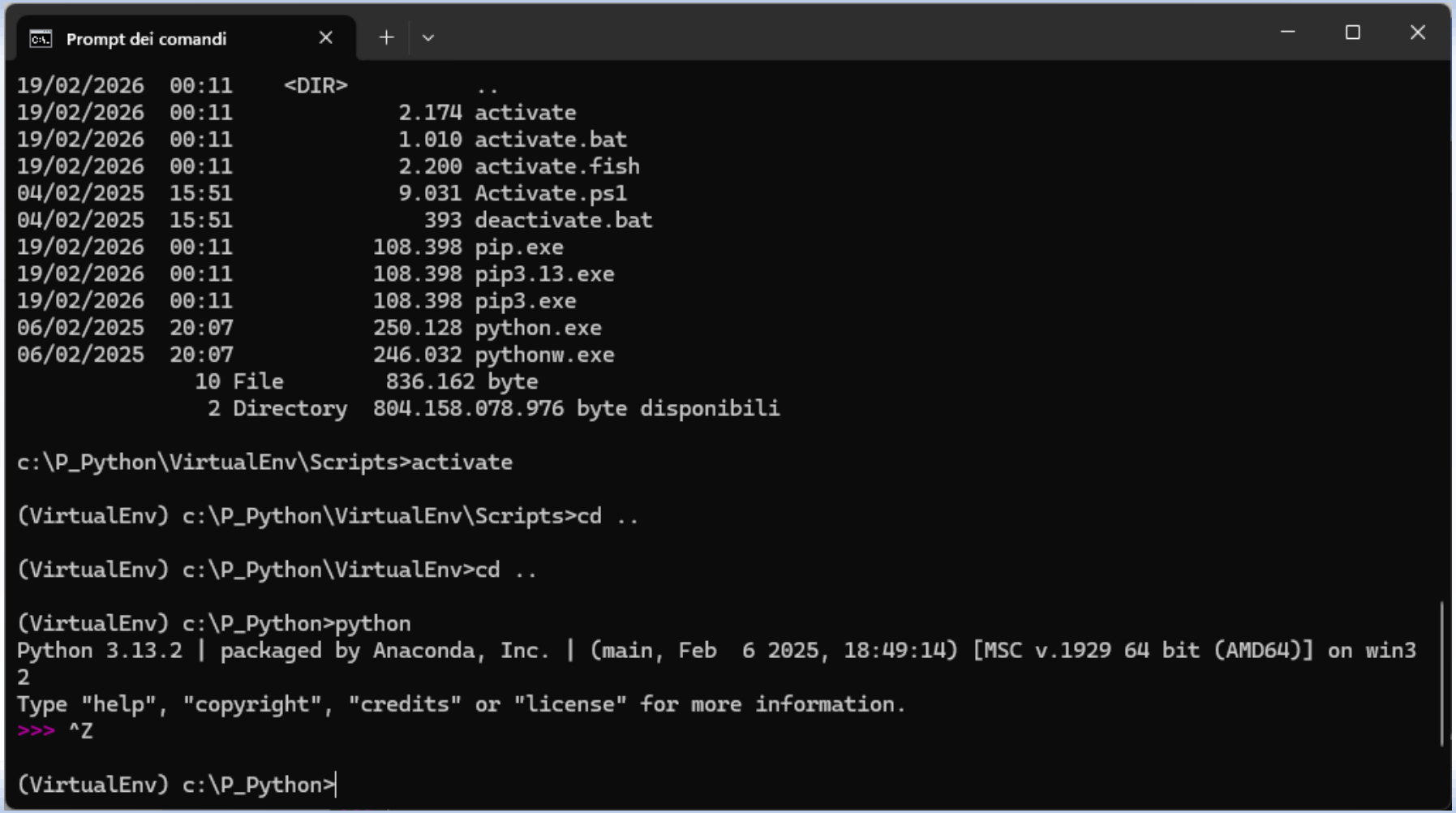
Intervallo tecnico → virtual environment «VENV»



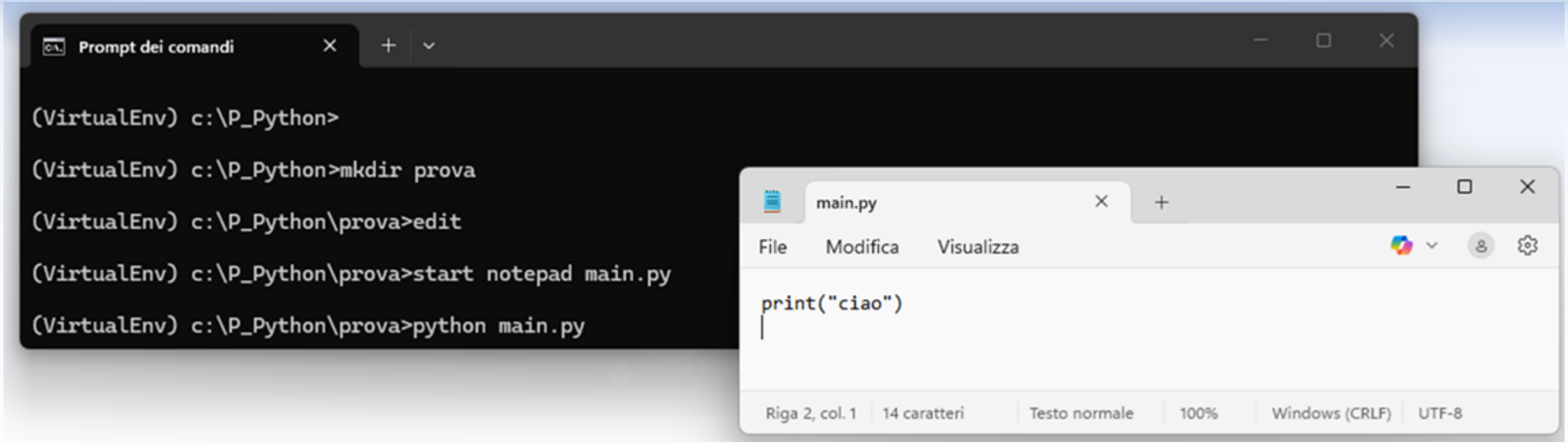
Intervallo tecnico → virtual environment «VENV»



Intervallo tecnico → virtual environment «VENV»

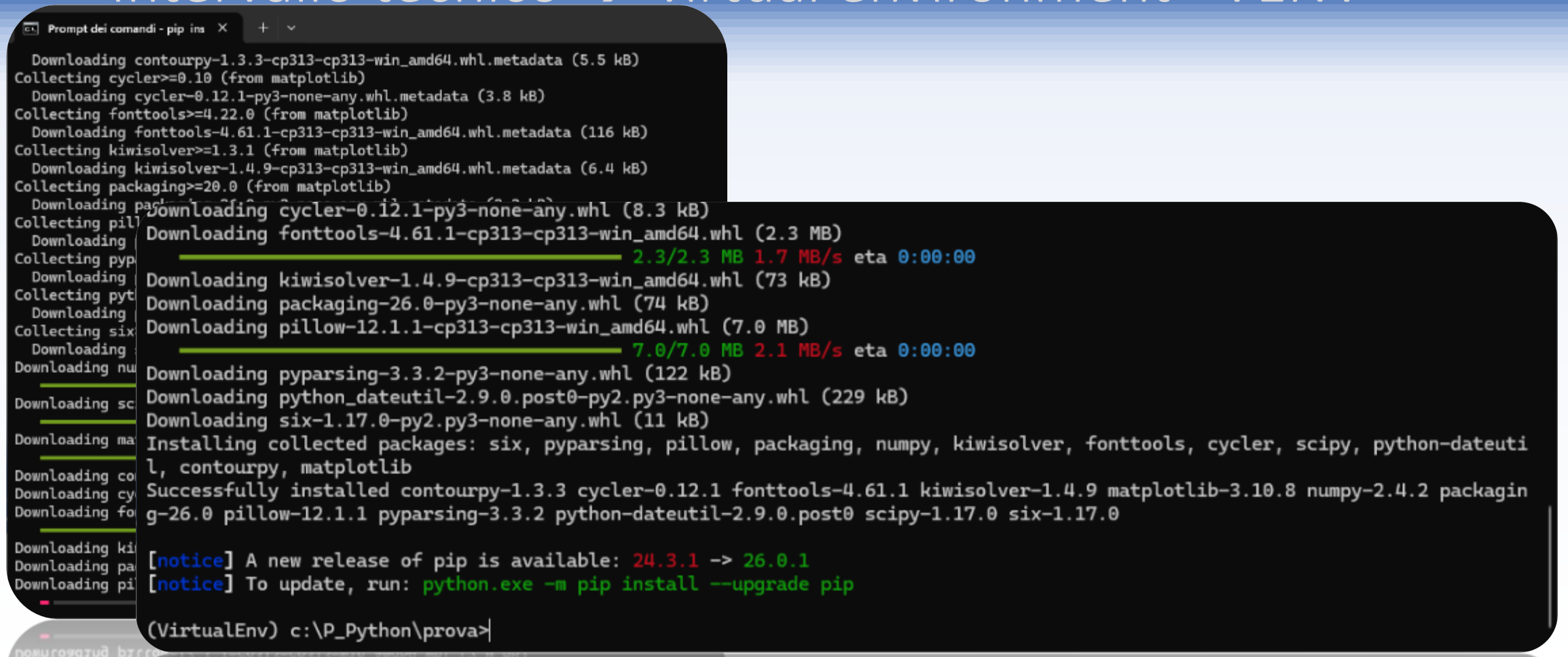


Intervallo tecnico → virtual environment «VENV»

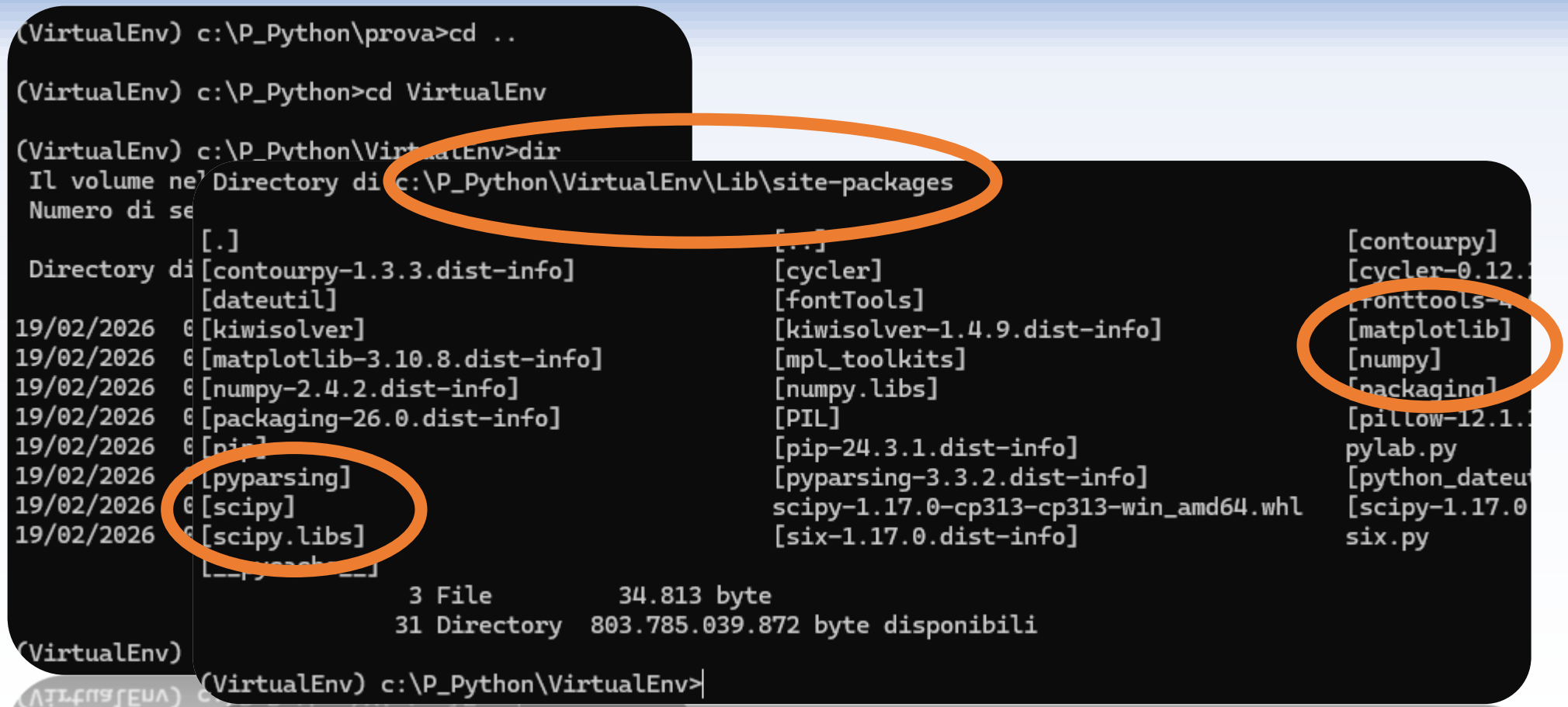



```
(VirtualEnv) c:\P_Python\prova>python main.py
ciao

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

Intervallo tecnico → virtual environment «VENV»



Intervallo tecnico → virtual environment «VENV»



Intervallo tecnico → virtual environment «VENV»

```
(VirtualEnv) c:\P_Python\prova>python
Python 3.13.2 | packaged by Anaconda, Inc. | (main, Feb  6 2025, 18:49:14) [MSC
Type "help", "copyright", "credits" or "license" for more information.
>>> 3*85
255
>>> 654*3541684
2316261336
>>> ('a6s5df4a6sd4fasfd'+"           "+'asdfafsd  ')*2
'a6s5df4a6sd4fasfd           asdfafsd  a6s5df4a6sd4fasfd           asdfafsd  '
>>> ('123'+"."+'456  ')*2
'123.456  123.456  '
>>>
```

```
>>> "Questa è una Stringa".__len__()
20
>>> len("Questa è una Stringa")
20
>>>
```

```
>>> dir("Questa è una Stringa")
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribu
te__', '__getitem__', '__getnewargs__', '__getstate__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '_
_le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__'
, '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'encod
e', 'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit
', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip'
, 'maketrans', 'partition', 'removeprefix', 'removesuffix', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'r
strip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
>>>
```

Intervallo tecnico → virtual environment «VENV»

```
>>> mario=input("asdfasfd? ")
asdfasfd? {'key':'value'}
>>> mario
"{'key':'value'}"
>>> luigi=eval(mario)
>>> luigi
{'key': 'value'}
>>>
```

```
>>> ggg=globals().copy()
>>> for pp in ggg:
...     print(pp)
...
__name__
__doc__
__package__
__loader__
__spec__
__annotations__
__builtins__
__file__
__cached__
mario
luigi
pp
```

```
>>> for pp in ggg:
...     print(pp,ggg[pp])
...
__name__ __main__
__doc__ None
__package__ _pyrepl
__loader__ <_frozen_importlib_external.SourceFileLoader object at 0x000002888BA57A
__spec__ ModuleSpec(name='_pyrepl.__main__', loader=<_frozen_importlib_external.So
>, origin='c:\\Apps\\miniconda313\\Lib\\_pyrepl\\__main__.py')
__annotations__ {}
__builtins__ <module 'builtins' (built-in)>
__file__ c:\Apps\miniconda313\Lib\_pyrepl\__main__.py
__cached__ c:\Apps\miniconda313\Lib\_pyrepl\__pycache__\__main__.cpython-313.pyc
mario {'key':'value'}
luigi {'key': 'value'}
pp __name__
>>>
```

Intervallo tecnico → virtual environment «VENV»

Built-in Functions
The Python interpreter has functions and types built into it, always available.

Built-in Functions

A
abs()
aiter()
all()
anext()
any()
ascii()

B
bin()
bool()
breakpoint()
bytearray()
bytes()

C
callable()
chr()
classmethod()
compile()
complex()

D
delattr()
dict()
dir()
divmod()

E
enumerate()
eval()
exec()

F
filter()
float()
format()
frozenset()

G
getattr()
globals()

H
hasattr()
hash()
help()
hex()

I
id()
input()
int()
isinstance()
issubclass()
iter()

L
len()
list()
locals()

M
map()
max()
memoryview()
min()

N
next()

O
object()
oct()
open()
ord()

P
pow()
print()
property()

R
range()
repr()
reversed()
round()

S
set()
setattr()
slice()
sorted()
staticmethod()
str()
sum()
super()

T
tuple()
type()

V
vars()

Z
zip()

_
__import__()

https://docs.python.org/3/library/functions.html

Intervallo tecnico → virtual environment «VENV»

Built-in Functions
Python has functions
and types built into,
always available.

A	D	H	M	R	V
abs()	delattr()	hasattr()	map()	range()	vars()
aiter()	dict()	hash()	max()	repr()	Z
all()	dir()	help()	memoryview()	reversed()	zip()
anext()	divmod()	hex()	min()	round()	
any()	E	I	N	S	__import__()
ascii()	enumerate()	id()	next()	set()	
B	eval()	input()	O	setattr()	
bin()	exec()	int()	object()	slice()	
bool()	F	isinstance()	oct()	sorted()	
breakpoint()	filter()	issubclass()	open()	staticmethod()	
bytearray()	float()	iter()	ord()	str()	
bytes()	format()	L	P	sum()	
C	frozenset()	len()	pow()	super()	
callable()	G	list()	print()	T	
chr()	getattr()	locals()	property()	tuple()	
classmethod()	globals()			type()	
compile()					
complex()					

https://docs.python.org/3/library/functions.html

Intervallo tecnico → sys

```
(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 sys
>>> print(f"Sistema operativo: {sys.platform}")
Sistema operativo: win32
>>> print(f"Versione Python: {sys.version}")
Versione Python: 3.13.2 | packaged by Anaconda, Inc. | (main, Feb  6 2025, 18:49:14) [MSC v.1929 64 bit (AMD64)]
>>> if len(sys.argv) < 2:
...     print("Nessun argomento fornito")
...     sys.exit(1)
...
Nessun argomento fornito

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

```
(VirtualEnv) c:\P_Python\prova>python
Python 3.13.2 | packaged by Anaconda, Inc.
Type "help", "copyright", "credits" or "lice
>>> from main import Mario
ciao
>>> from main import Mario
>>> pppp=Mario()
>>> pppp.variabile1
123
>>>
```

main.py — Built-in Functions

File Modifica Visualizza

```
print("ciao")

class Mario():
        def __init__(self):
                self.variabile1=123
```

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)}")



#
```

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)}")



#
```

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)}")
#
```