

Lecture 2: Variables and Types

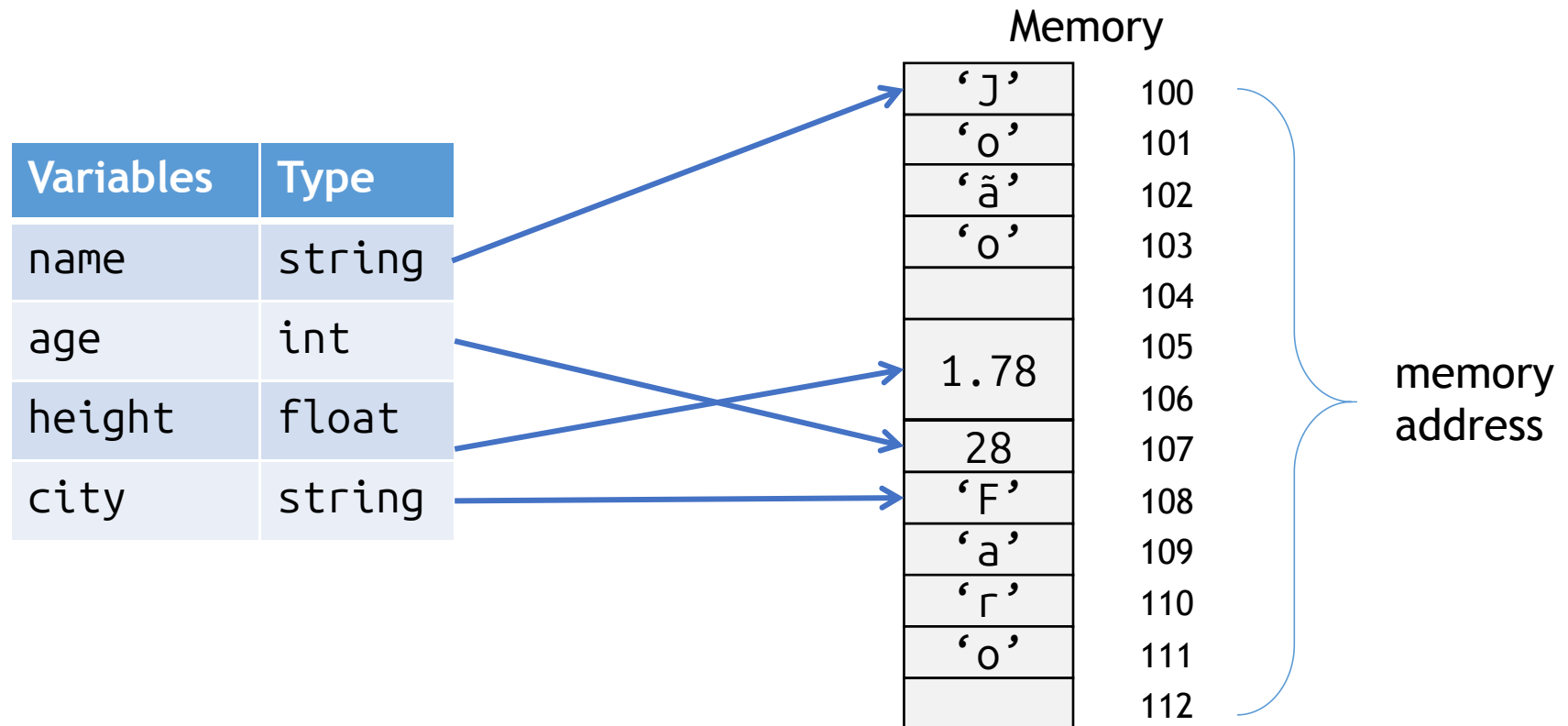
Introduction to numbers, strings and expressions

Lecture 2: Variables and Types

- In this section:
 - Basic elements of the language: **numbers** and **strings**
 - Fundamental structures:
 - Variables and Constants
 - Operators (math and instructions)

Lecture 2: Variables and Types

- Variable: a reference, with **name**, to an area of the memory containing a specific **value**
- Values can be of different **types**, allowing to memorize different things: integer/real numbers, text...



Lecture 2: Variables and Types

- To “create” a new variable, it must be **declared** and **initialized**, that means:
 - Give a **name**, so that it can be used in the program
 - Give an initial **value**
- To give or change the value, use the assignment operator =

```
triangle_area = 4  #variable initialization  
# some code here...  
triangle_area = 7  #change value
```

Lecture 2: Variables and Types

-  A variable is defined the first time it is assigned a value
 - After the declaration, it can be used for other operations
 - (otherwise: error; which type?)
 - Assignment syntax:
 - `variable_name = expression`
 - No spaces allowed for the name, and other limitations (see next slides)
 - expressions must define a value, it can be:

```
x = -3.2
messaggio = "Buongiorno"
y = x + 1
```

Lecture 2: Variables and Types

- Rules for variables naming:

- Contain letters or numbers or ‘_’
- Cannot start with a number
- Case sensitive
- Cannot coincide with Python keywords

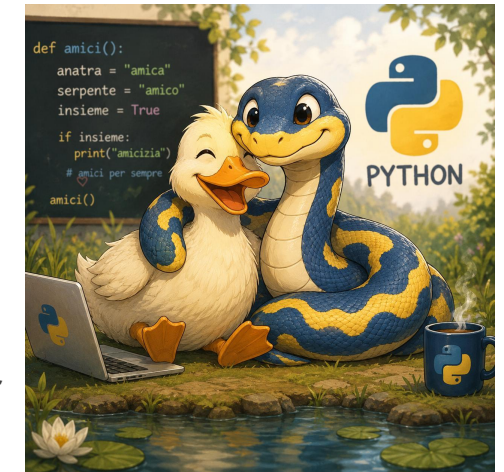
False	None	True	and	as	assert
async	await	break	class	continue	def
del	elif	else	except	finally	for
from	global	if	import	in	is
lambda	nonlocal	not	or	pass	raise
return	try	while	with	yield	

- Max 256 characters

Lecture 2: Variables and Types

- What about types?
 - Duck Typing in Python:
 - "If it looks like a duck and quacks like a duck, it's a duck"
 - type of an object less important than its structure
 - compilers don't do type-checking. It only matters if the object has particular attributes at run time
 - Type is not associated to variables, **only to the value** → We can assign different types to the same variable (*dynamic typing*)
 - Basic Python types:

	Explicit numbers (literals)
• integers (int):	0, 12, -9
• fractions (float)	3.14, -4.8, 0.05, 1.0, 1.99E33 ($1,99 \times 10^{33}$), -42.3e-4
• strings of characters (str)	"Hello World!"
• booleans (bool)	True, False



Lecture 2: Variables and Types

- function `type()`
 - Python provides a built-in function to retrieve the type of an object

```
>>> type(3.14)
<class 'float'>
>>> type(4)
<class 'int'>
>>> type(1.0)
<class 'float'>
>>> a = "type of string"
>>> type(a)
<class 'str'>
>>> type(True)
<class 'bool'>
>>>
>>> type_of_pi = type(3.14)
>>> type_of_pi
<class 'float'>
```

Lecture 2: Variables and Types

- function `type()`
 - Python provides a built-in function to retrieve the type of an object

Note: the value returned by a function can be (and normally is) stored in a variable or used in expressions

```
>>> type(3.14)
<class 'float'>
>>> type(4)
<class 'int'>
>>> type(1.0)
<class 'float'>
>>> a = "type of string"
>>> type(a)
<class 'str'>
>>> type(True)
<class 'bool'>
>>>
>>> type_of_pi = type(3.14)
>>> type_of_pi
<class 'float'>
```

Lecture 2: Constants

- Constants are variables whose value cannot be changed after initialization.
- In Python...

Lecture 2: Constants

- Constants are variables whose value cannot be changed after initialization.



- In Python... **they do not exist**

Lecture 2: Constants

- Constants are variables whose value cannot be changed after initialization.



- In Python... they do not exist
- Python follows *conventions*
 - Most rules are only guidelines
 - you are allowed not to follow them
 - even to redefine basic elements and behaviour
 - flexibility, but at your own risk
 - Python guarantees that everything will work if you follow the convention



Lecture 2: Constants

- Constants in Python:
 - The convention says that variables using CAPITAL LETTERS should not be modified
- Why use constants?
 - mainly readability, and maintainability of the code:

```
BOTTLE_CAPACITY = 1.5

bottles = 5 + 7
tot_capacity = bottles * BOTTLE_CAPACITY
# instead of
tot_capacity = bottles * 1.5
```

Lecture 2: Expressions and Operators

- The expressions produce values of these types
 - Arithmetic expressions → numeric types
 - String expressions → strings
 - Boolean expressions → boolean
- Arithmetic expressions may be simple numbers or formulated using so called **operators**:

```
a = 7
b = 3

# ARITHMETIC OPERATORS
a + b      # addition      -> 10      int
a - b      # subtraction   -> 4       int
a * b      # multiplication -> 21      int
a / b      # division      -> 2.333... float
a // b     # floor division -> 2       int
a % b      # remainder (modulo) -> 1      int
a ** b     # exponentiation -> 343     int
```

- `+`, `-`, `*`, `**`: If one of the two operands is float, we obtain a float
- Normal division `/` produces always float
- If you want an integer, you should use integer division `//`

Lecture 2: Expressions and Operators

- Math operations in python follow the normal precedence:
 1. (...) Parentheses evaluated first
 2. ** Exponentiation
 3. +x, -x Unary plus, unary minus
 4. *, /, //, % Multiplication, division, floor division, modulo
 5. +, - Addition, subtraction

Operator precedence examples

```
2 + 3 * 4      # 14
10 - 6 / 2     # 7.0
2 + 3 ** 2     # 11
2 * 3 ** 2     # 18
```

```
(2 + 3) * 4    # 20
(2 * 3) ** 2   # 36
```

Special cases with **

```
-2 ** 2        # -4    -> -(2 ** 2)
(-2) ** 2      # 4
```

```
2 ** 3 ** 2    # 512   -> 2 ** (3 ** 2)
(2 ** 3) ** 2  # 64
```

Lecture 2: Expressions and Operators

- String Expressions:

- Explicit string (*literal*): “This is a string”, ‘So is this, 2nd string!’
- variable associated with a string: `first_name = “Manuel”`
- String concatenation (use operator “+”)

```
firstName = "Manuel"
lastName  = "Pereira"
name = firstName + lastName # "ManuelPereira"
name_with_space = firstName + " " + lastName # "Manuel Pereira"
```

- String repetition (use operator “*”)

```
dashed = "-." * 10
# -.-.-.-.-.-.-.-.-.-
```

Lecture 2: Expressions and Operators

- Common error:
 - We cannot concatenate string with other types
 - to transform into a string a different type, use function `str()`
 - It returns a string containing a “copy” of the original value.

```
>>> a = 10
>>> "variable a has value " + a
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
>>>
>>>
>>> "variable a has value " + str(a)
'variable a has value 10'
>>> █
```

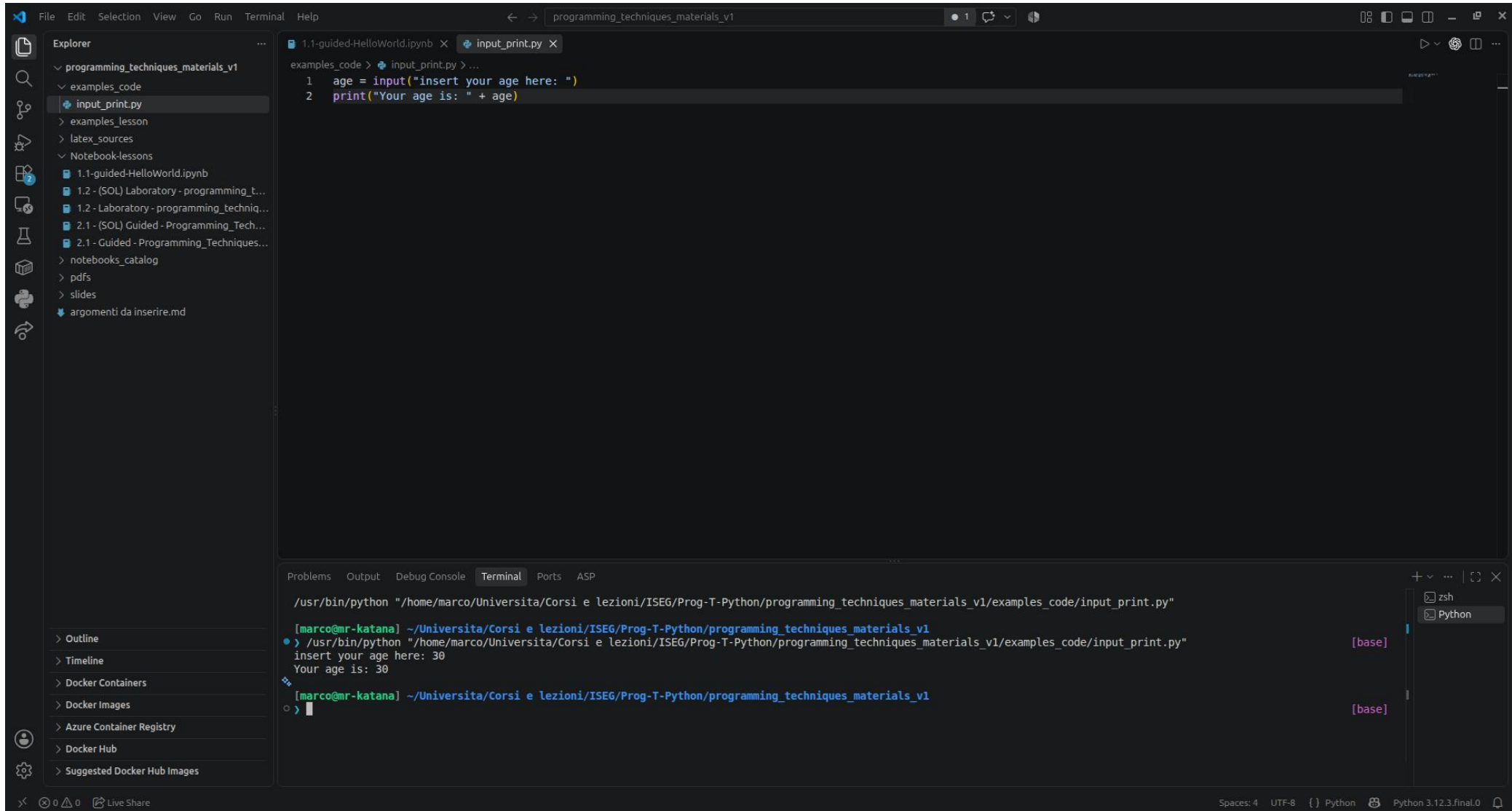
Lecture 2: Expressions and Operators

- Common error:
 - We cannot concatenate string with other types
 - to transform into a string a different type, use function `str()`
 - It returns a string containing a “copy” of the original value.

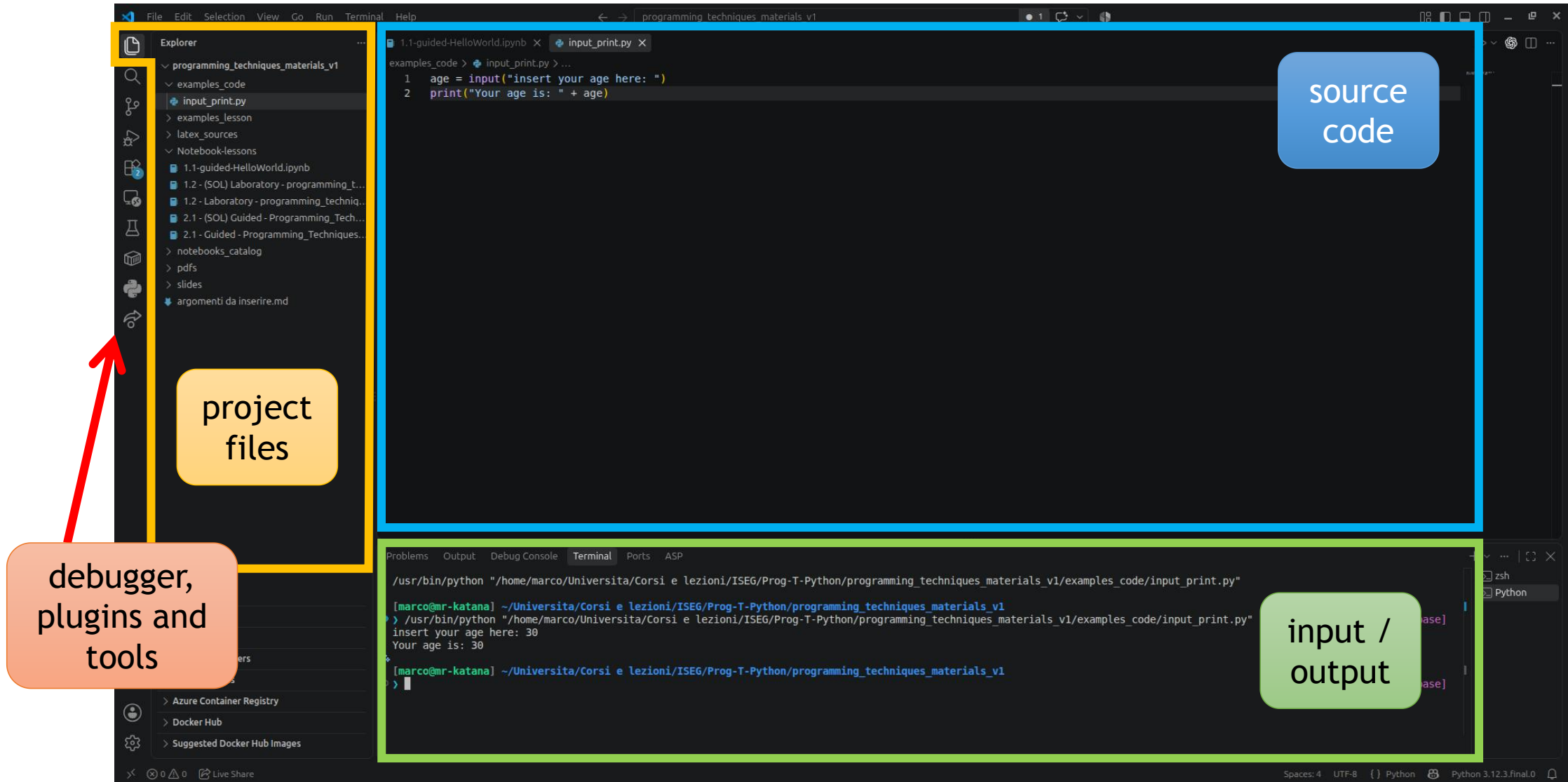
```
>>> a = 10
>>> "variable a has value " + a
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
>>>
>>>
>>> "variable a has value " + str(a)
'variable a has value 10'
>>> █
```

HANDS-ON! (Guided 2.1)

Lecture 2: VS code Overview



Lecture 2: VS code Overview



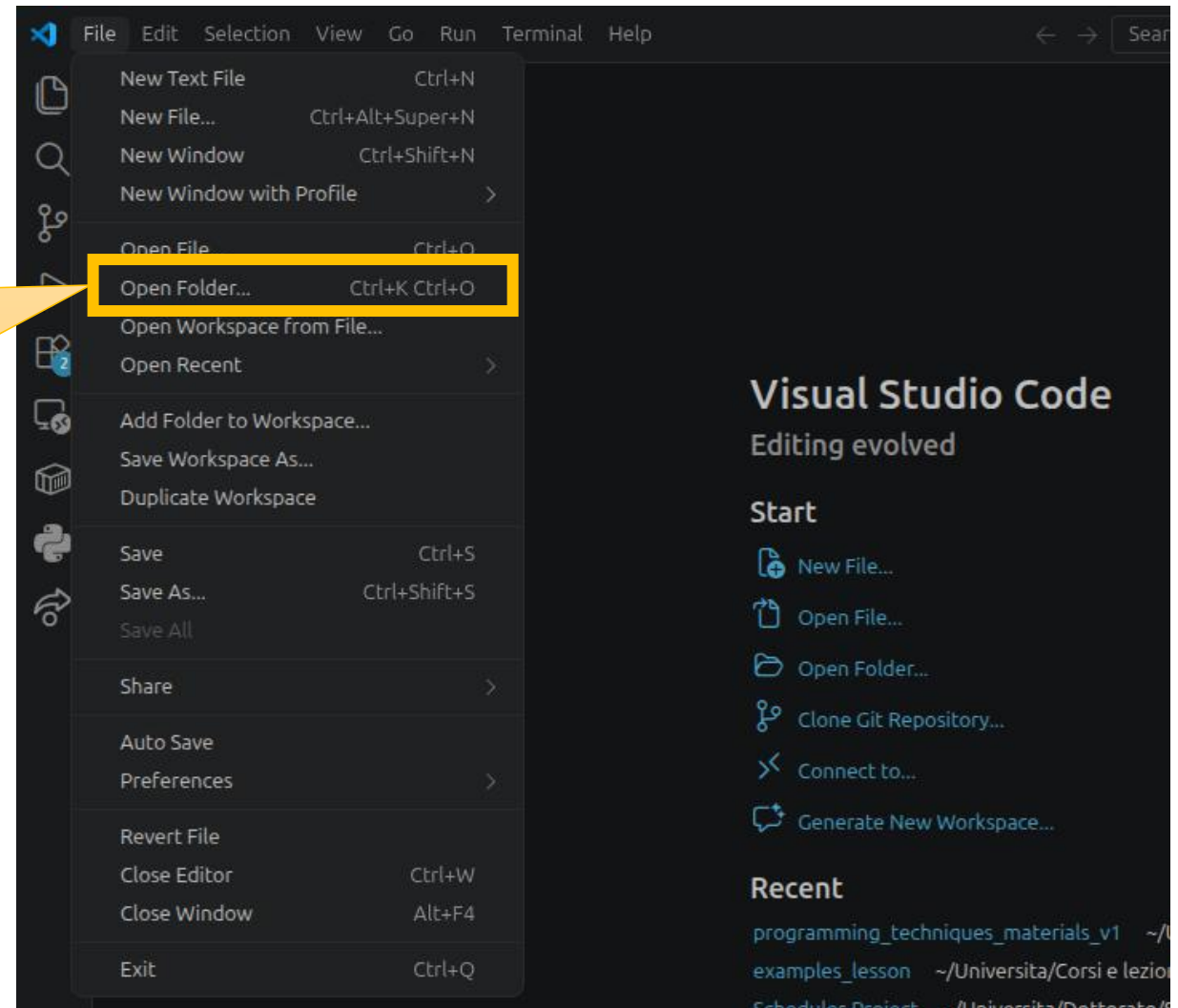
Lecture 2: VS code Overview

- Project, Program and file
 - A large program can be composed of many files
 - IDEs allow to group related files in a “project”
 - in VS Code, this corresponds to a folder with all its content
 - To create a new program:
 - new folder
 - create one or more python files (.py) in it

Lecture 2: VS code Overview

Open/Create new project (folder)

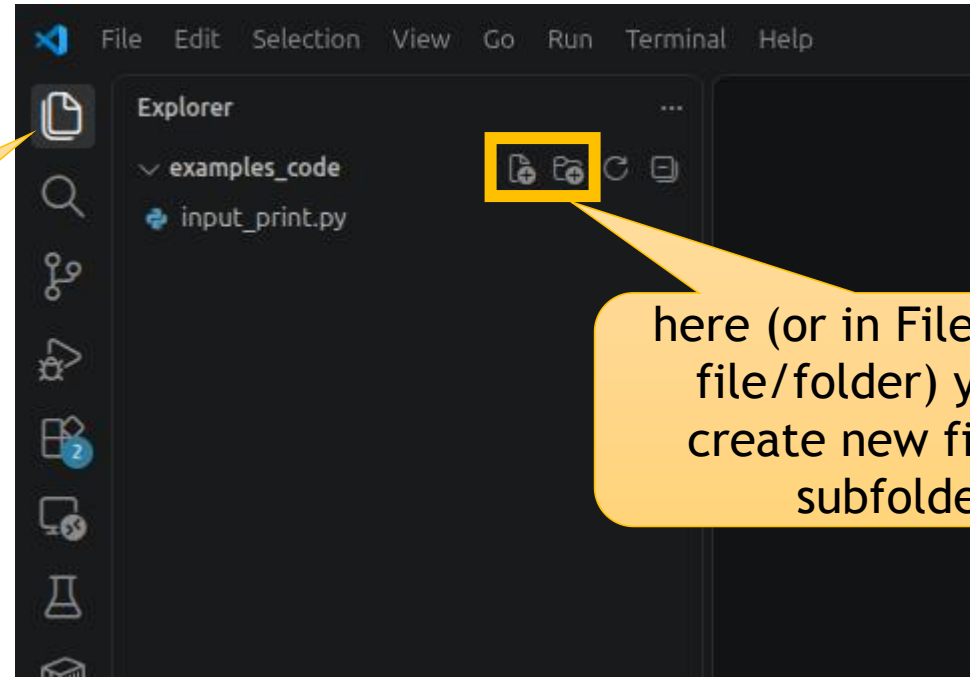
File >> Open Folder
select an existing folder or
create a new one



Lecture 2: VS code Overview

Open/Create new project (folder)

Explorer tab
here you can manage
your files and subfolders

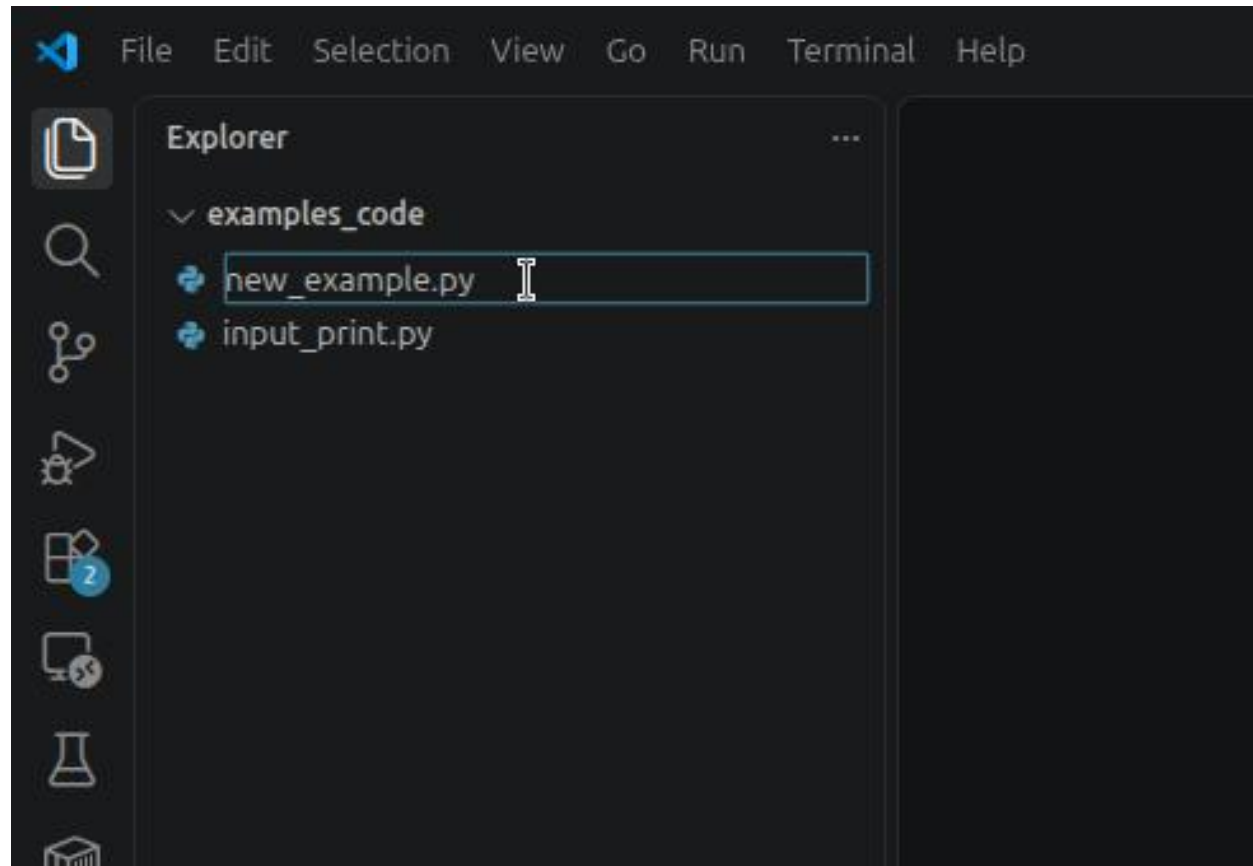


here (or in Files >> new
file/folder) you can
create new files and
subfolders

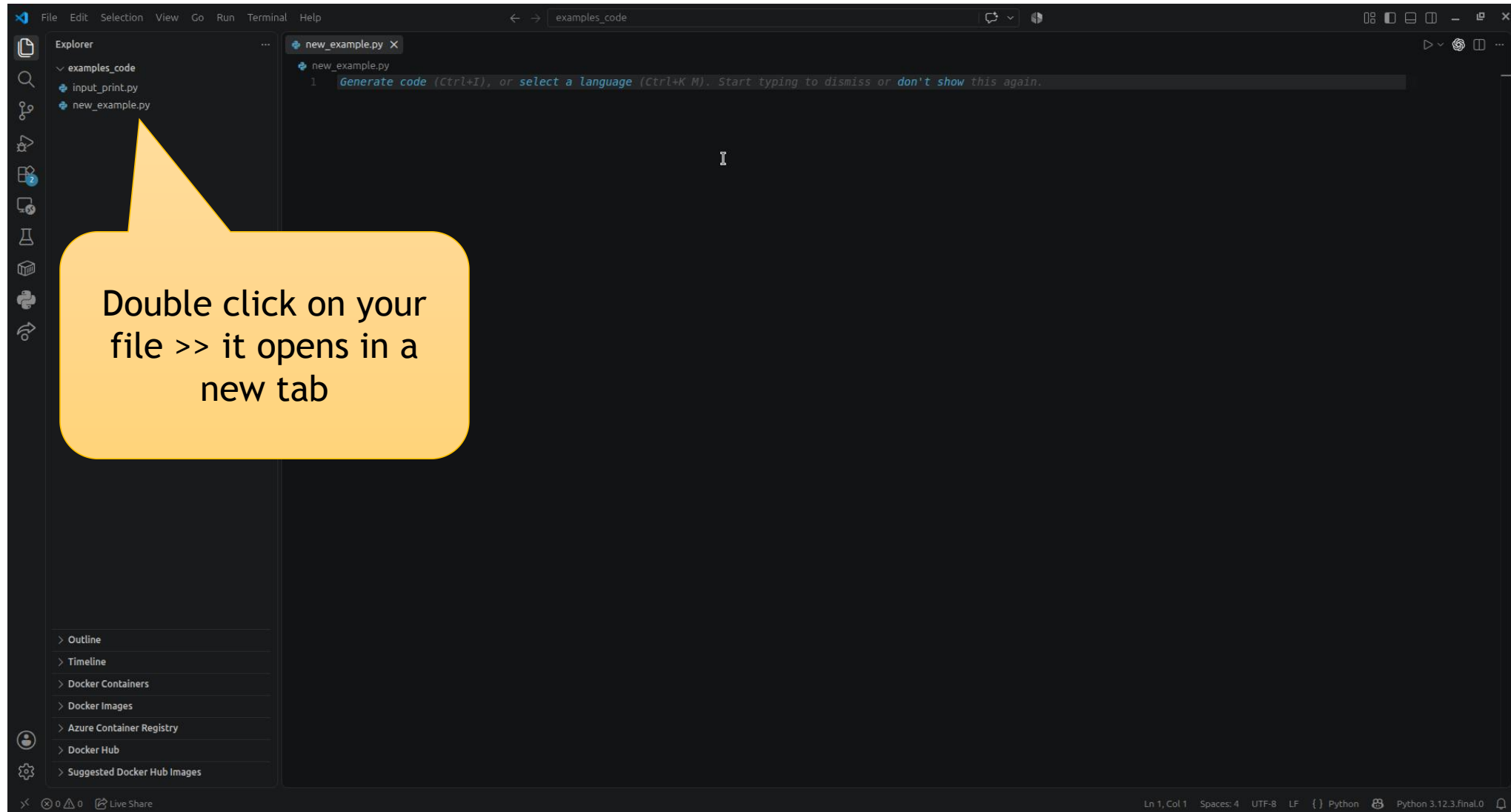
Lecture 2: VS code Overview

Open/Create new project (folder)

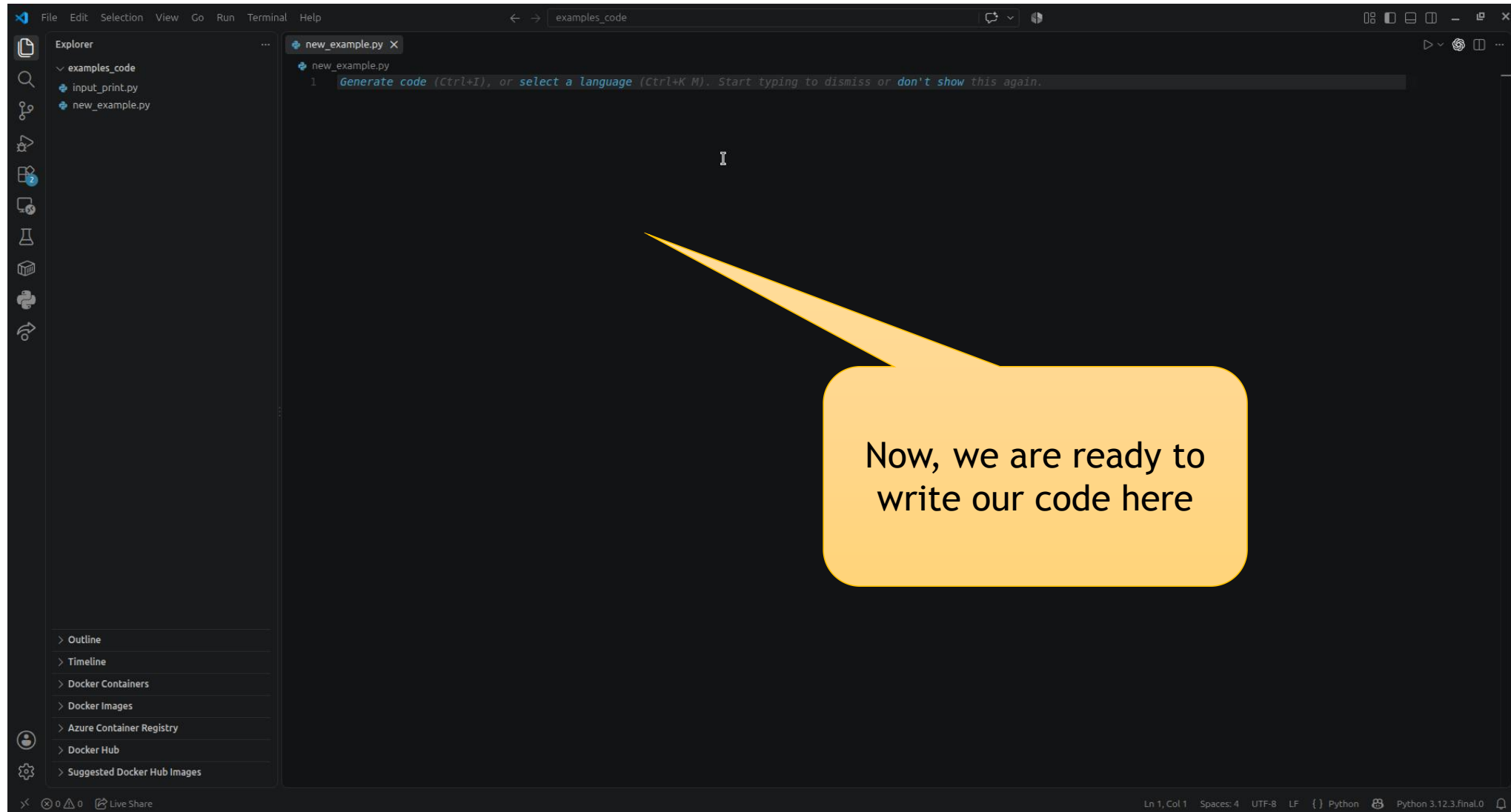
- Create a new file
- Enter a name, ending with `.py`
- Then, press enter
- Same for a notebook:
 - instead of `.py` use `.ipynb`
 - if jupyter extension is correctly installed in VS Code, it can manage immediately text and code cells exactly as in Colab



Lecture 2: VS code Overview



Lecture 2: VS code Overview



Lecture 2: VS code Overview

