Lesson 1 – NumPy Refresher
Topic: Arrays, creation, indexing, slicing, reshaping, operations

---

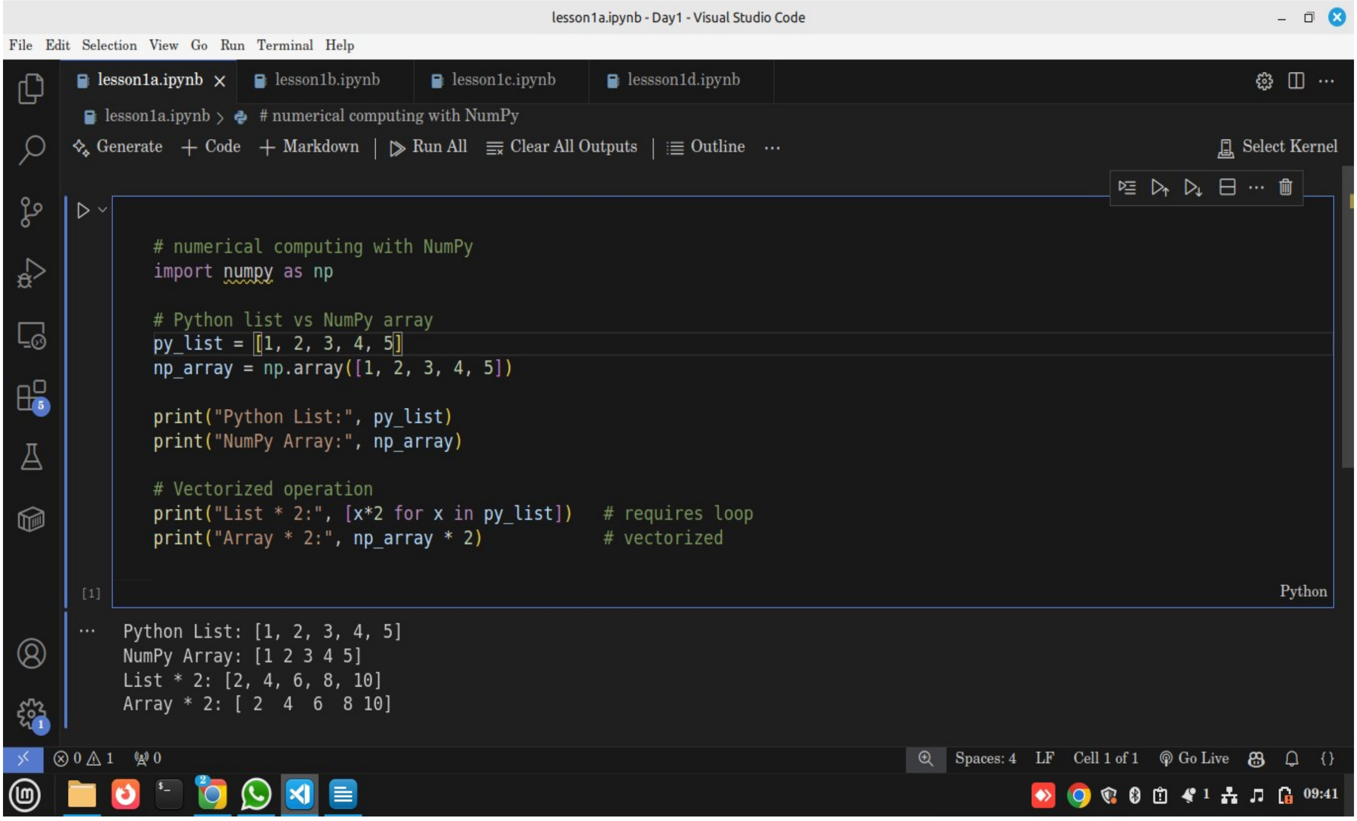
🔹 NumPy & Arrays

Theory Notes

- NumPy = *Numerical Python*, a library for numerical computing.
- Main object: **ndarray** (n-dimensional array).
- Advantages: **fast**, **efficient**, **vectorized operations**.

Python lists vs NumPy Array

Practical



```python
# numerical computing with NumPy
import numpy as np

# Python list vs NumPy array
py_list = [1, 2, 3, 4, 5]
np_array = np.array([1, 2, 3, 4, 5])

print("Python List:", py_list)
print("NumPy Array:", np_array)

# Vectorized operation
print("List * 2:", [x*2 for x in py_list])  # requires loop
print("Array * 2:", np_array * 2)           # vectorized
```

```
Python List: [1, 2, 3, 4, 5]
NumPy Array: [1 2 3 4 5]
List * 2: [2, 4, 6, 8, 10]
Array * 2: [ 2  4  6  8 10]
```

Explanation

- We imported NumPy as `np` (common alias).
- `py_list` is a normal Python list.
- `np_array` is a NumPy array created from the same numbers.
- When multiplying a Python list by 2 → it duplicates the list `[1,2,3,4,5,1,2,3,4,5]`.
- When multiplying a NumPy array by 2 → it multiplies **each element** (`[2,4,6,8,10]`).
- This is the **power of vectorization**.

Note:

- A Python list requires a loop to multiply each element.
- A NumPy array applies the operation automatically to all elements.
- This makes NumPy much faster when working with large datasets.

Use in Data Science

NumPy arrays are the foundation of **pandas DataFrames, scikit-learn, and deep learning libraries**. Almost every dataset you load is converted internally into arrays for speed.

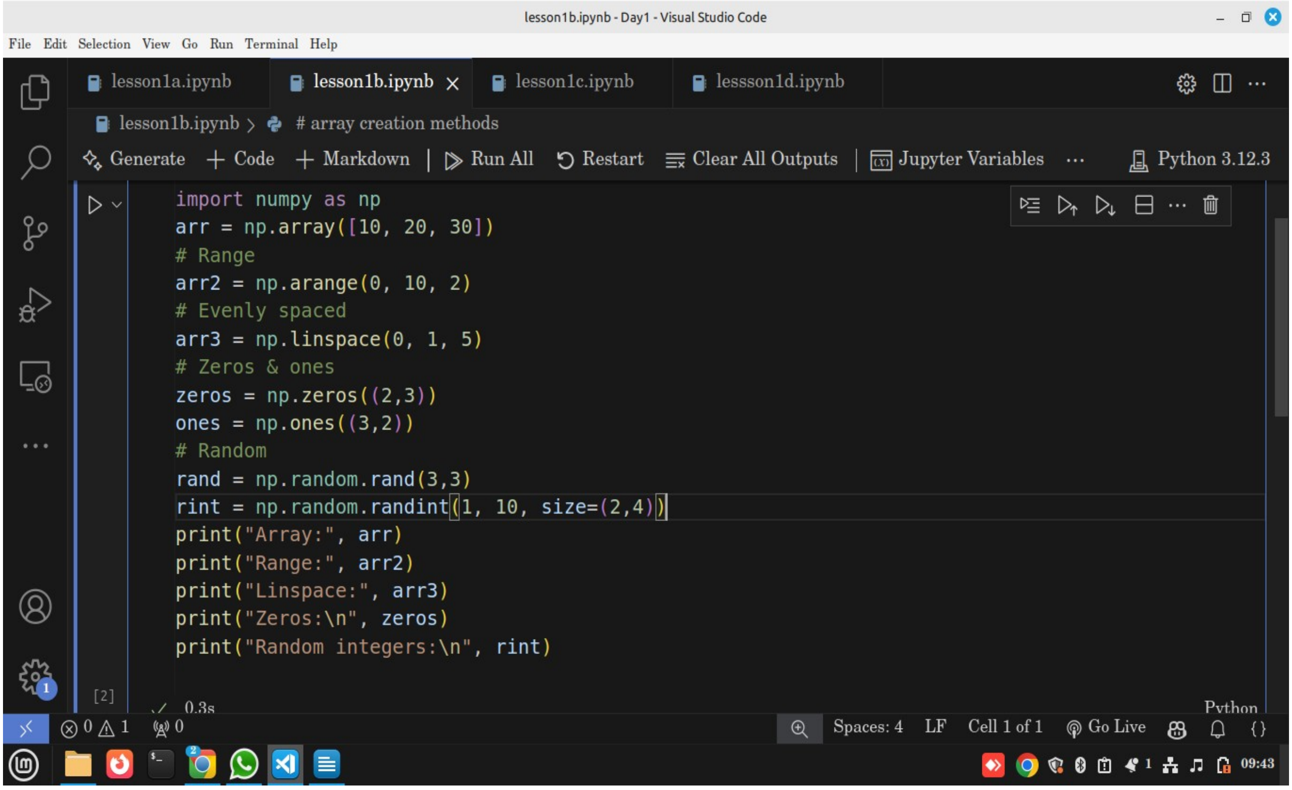
🔹 Array Creation Methods

Theory Notes

Ways to create arrays in NumPy:

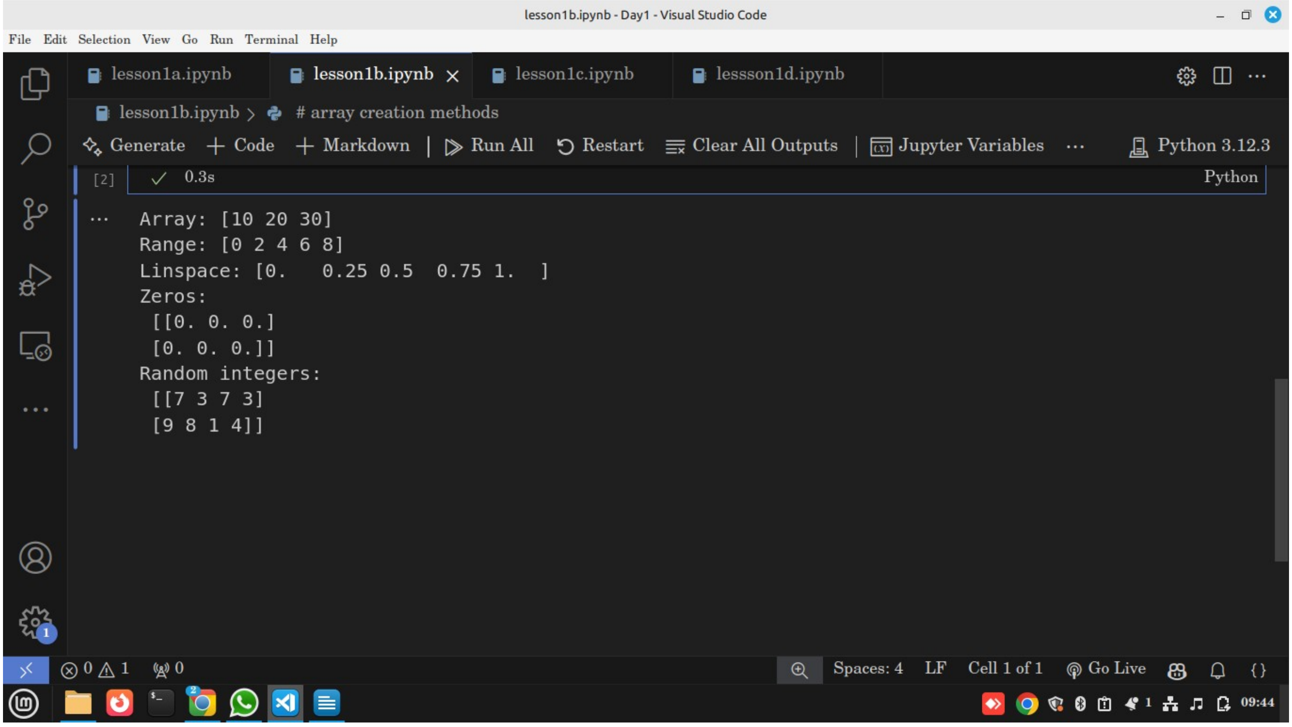
- From Python lists → `np.array()`
- Number ranges → `np.arange()`
- Even spacing → `np.linspace()`
- Special arrays → `np.zeros()`, `np.ones()`, `np.eye()`
- Random arrays → `np.random.rand()`, `np.random.randint()`

Practical



```python
import numpy as np
arr = np.array([10, 20, 30])
# Range
arr2 = np.arange(0, 10, 2)
# Evenly spaced
arr3 = np.linspace(0, 1, 5)
# Zeros & ones
zeros = np.zeros((2,3))
ones = np.ones((3,2))
# Random
rand = np.random.rand(3,3)
rint = np.random.randint(1, 10, size=(2,4))
print("Array:", arr)
print("Range:", arr2)
print("Linspace:", arr3)
print("Zeros:\n", zeros)
print("Random integers:\n", rint)
```

Output



lesson1b.ipynb - Day1 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
lesson1a.ipynb
lesson1b.ipynb
lesson1c.ipynb
lessson1d.ipynb
lesson1b.ipynb > # array creation methods
Generate + Code + Markdown | Run All Restart Clear All Outputs | Jupyter Variables
Python 3.12.3
[2] ✓ 0.3s
Python
Spaces: 4 LF Cell 1 of 1 Go Live

```
Array: [10 20 30]
Range: [0 2 4 6 8]
Linspace: [0.   0.25 0.5  0.75 1.  ]
Zeros:
 [[0. 0. 0.]
 [0. 0. 0.]]
Random integers:
 [[7 3 7 3]
 [9 8 1 4]]
```

Explanation

- `np.array([10,20,30])` → creates an array from a list.
- `np.arange(0,10,2)` → creates `[0,2,4,6,8]`.
- `np.linspace(0,1,5)` → creates 5 numbers evenly spaced between 0 and 1.
- `np.zeros((2,3))` → 2 rows, 3 columns, all zeros.
- `np.ones((3,2))` → 3 rows, 2 columns, all ones.
- `np.random.rand(3,3)` → 3×3 matrix of random floats between 0–1.
- `np.random.randint(1,10,size=(2,4))` → random integers 1–9 in a 2×4 grid.

Use in Data Science

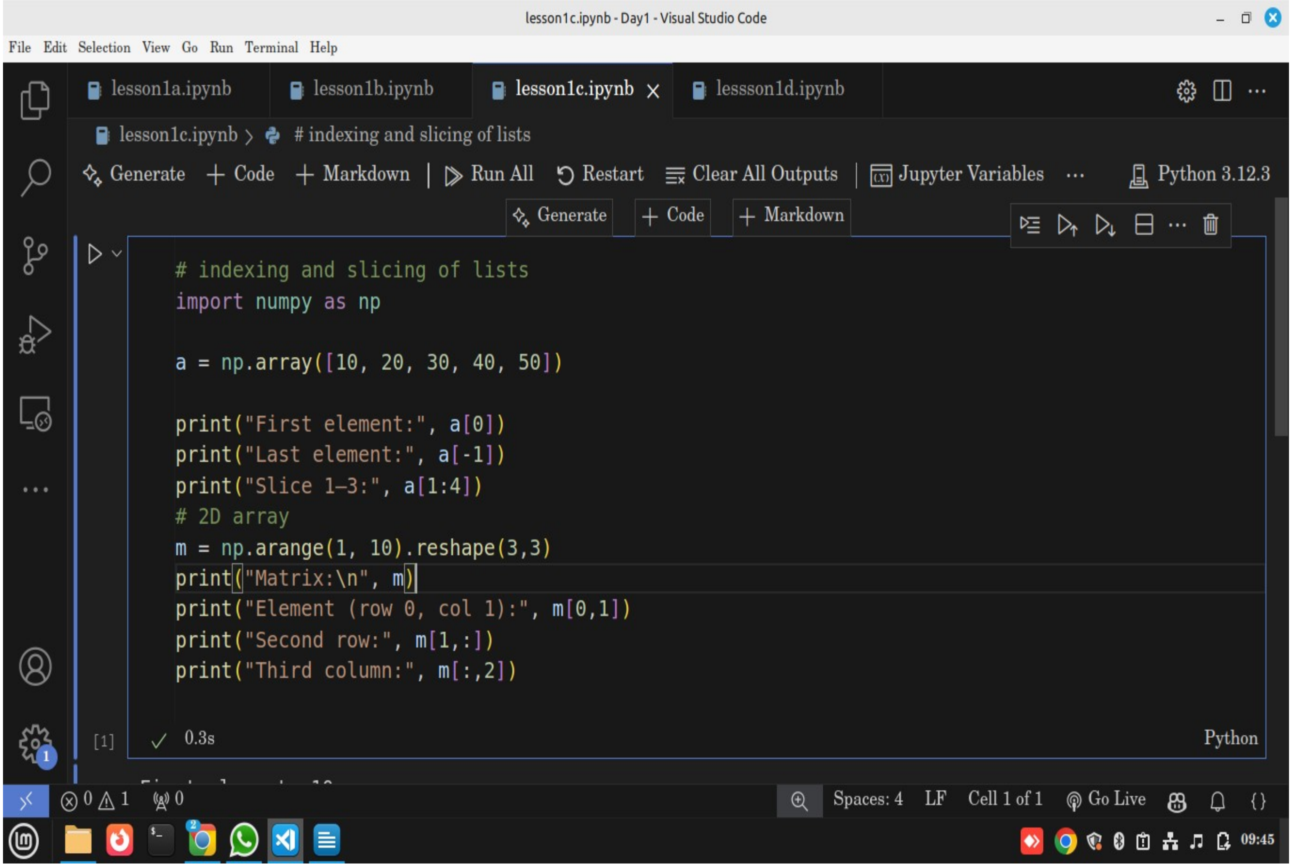
- `np.arange` and `np.linspace` are used to generate sequences of features (like time steps).
- Random arrays are essential in **machine learning** for initializing model weights, or for creating synthetic datasets.

🔹 Indexing & Slicing

Theory Notes

- Indexing = accessing elements.
- Works in 1D and 2D arrays.
- Uses **row, column** notation for 2D.

Practical



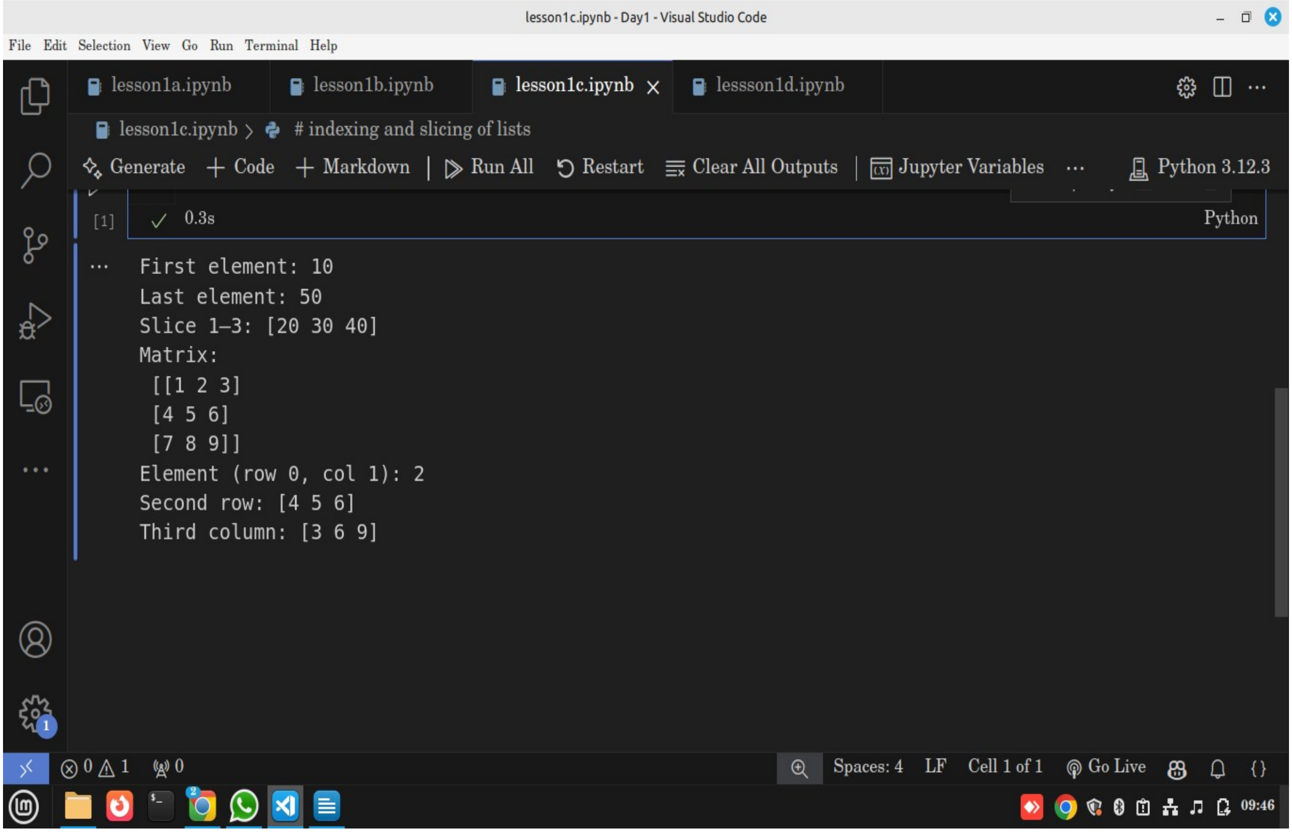
```
# indexing and slicing of lists
import numpy as np

a = np.array([10, 20, 30, 40, 50])

print("First element:", a[0])
print("Last element:", a[-1])
print("Slice 1–3:", a[1:4])
# 2D array
m = np.arange(1, 10).reshape(3,3)
print("Matrix:\n", m)
print("Element (row 0, col 1):", m[0,1])
print("Second row:", m[1,:])
print("Third column:", m[:,2])
```

lesson1c.ipynb - Day1 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
lesson1a.ipynb
lesson1b.ipynb
lesson1c.ipynb
lesssson1d.ipynb
lesson1c.ipynb > # indexing and slicing of lists
Generate + Code + Markdown | Run All Restart Clear All Outputs | Jupyter Variables ... Python 3.12.3
[1] ✓ 0.3s Python
Spaces: 4 LF Cell 1 of 1 Go Live
09:45

Output



```
First element: 10
Last element: 50
Slice 1–3: [20 30 40]
Matrix:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
Element (row 0, col 1): 2
Second row: [4 5 6]
Third column: [3 6 9]
```

Explanation

- `a[0]` → gives first element `10`.
- `a[-1]` → gives last element `50`.
- `a[1:4]` → extracts values from index 1 to 3 → `[20,30,40]`.
- `m = np.arange(1,10).reshape(3,3)` → makes a 3×3 matrix from numbers 1–9.
- `m[0,1]` → element in **row 0, col 1** (value = 2).
- `m[1,:]` → entire second row `[4,5,6]`.
- `m[:,2]` → entire third column `[3,6,9]`.

Use in Data Science

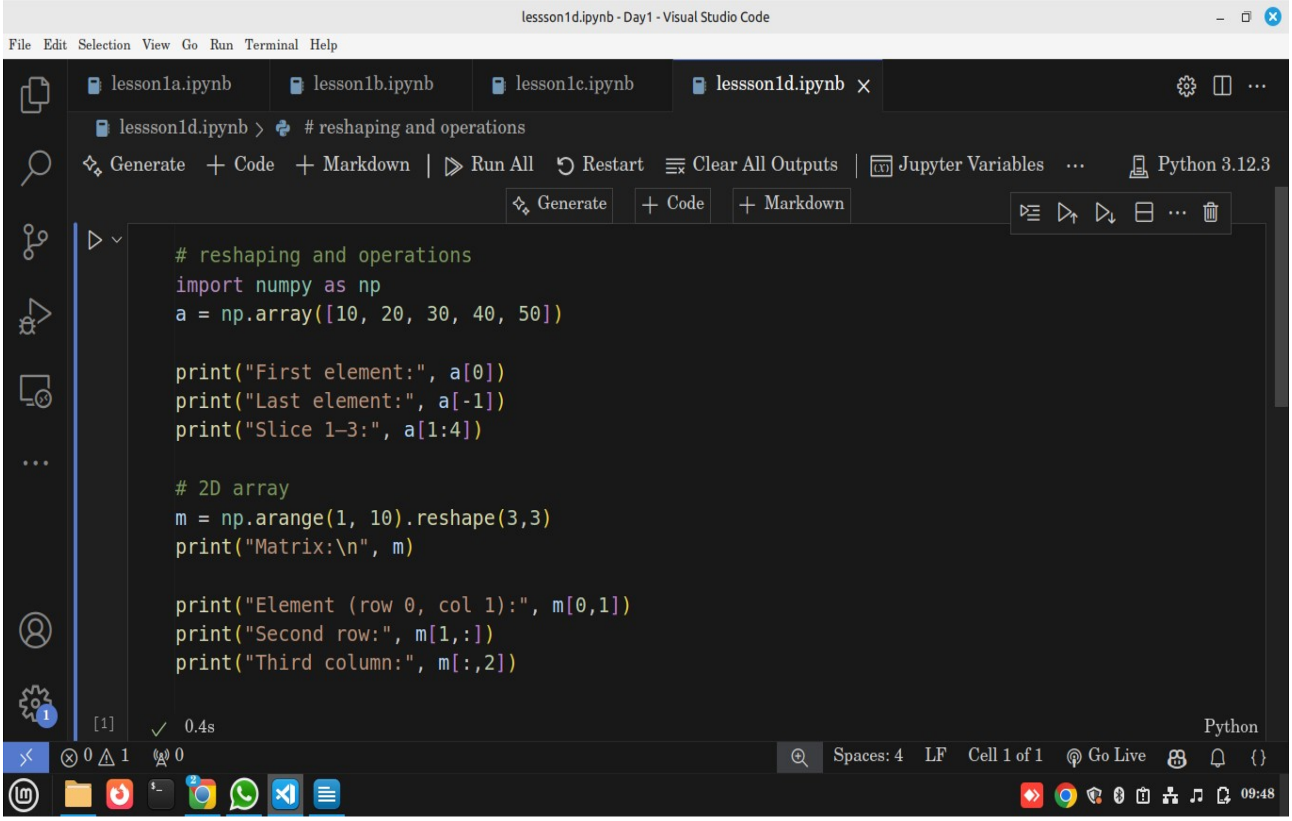
Indexing and slicing are key in **data wrangling**. For example, selecting a column of features (like “age”) from a dataset or filtering specific rows (like “patients over 50”).

🔹 Reshaping & Operations

Theory Notes

- `.reshape()` changes array shape without changing data.
- `.T` transposes rows ↔ columns.
- Math/stat functions: `.sum()`, `.mean()`, `.std()`, `.max()`, `.min()`.
- Arithmetic is element-wise.

Practical



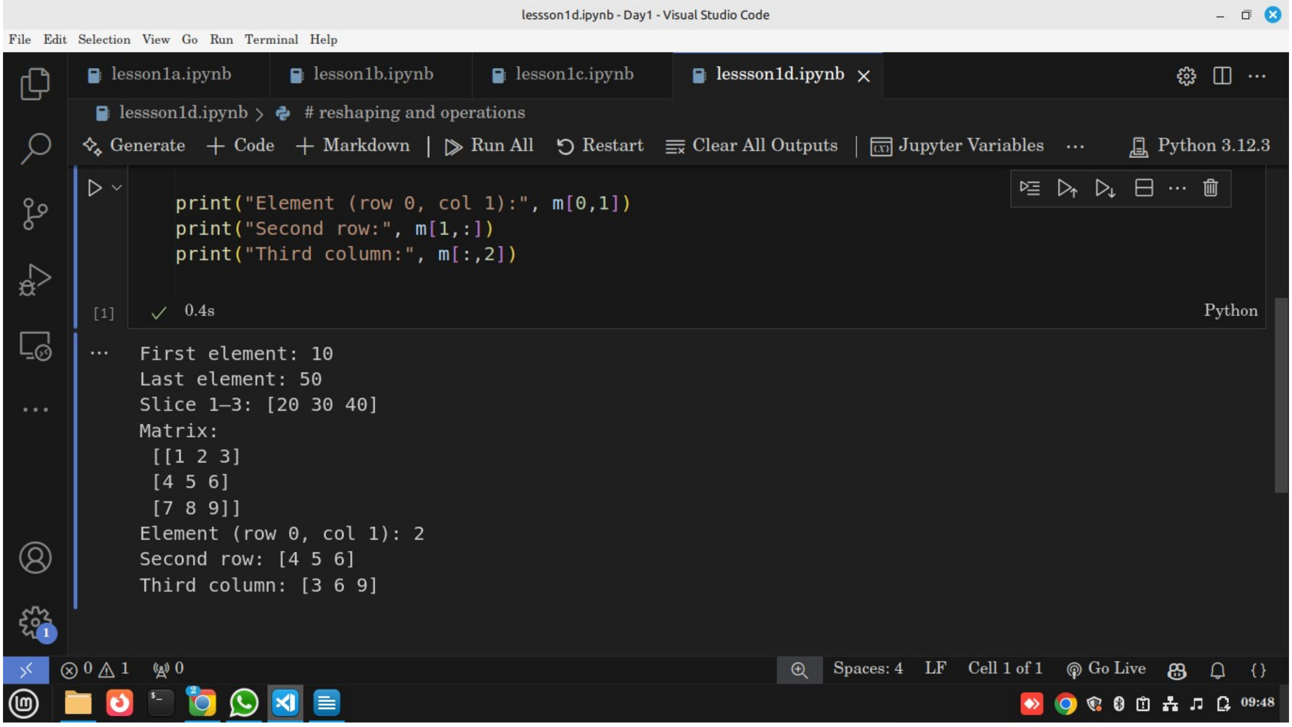
```python
# reshaping and operations
import numpy as np
a = np.array([10, 20, 30, 40, 50])

print("First element:", a[0])
print("Last element:", a[-1])
print("Slice 1–3:", a[1:4])

# 2D array
m = np.arange(1, 10).reshape(3,3)
print("Matrix:\n", m)

print("Element (row 0, col 1):", m[0,1])
print("Second row:", m[1,:])
print("Third column:", m[:,2])
```

Output



```
print("Element (row 0, col 1):", m[0,1])
print("Second row:", m[1,:])
print("Third column:", m[:,2])
```

```
First element: 10
Last element: 50
Slice 1–3: [20 30 40]
Matrix:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
Element (row 0, col 1): 2
Second row: [4 5 6]
Third column: [3 6 9]
```

Explanation

- `np.arange(1,13)` → creates numbers 1–12.
- `.reshape(3,4)` → reshapes into 3 rows × 4 columns.
- `.T` → flips rows & columns.
- `.sum()` → adds all numbers.
- `.mean()` → computes average.
- `.std()` → measures spread (standard deviation).
- Multiplication (`*2`) and addition (`+5`) apply to every element.

Use in Data Science

Reshaping is crucial when preparing data for machine learning models, where inputs must have a fixed size (e.g., reshape images into arrays). Stats functions help quickly summarize datasets (e.g., average income, standard deviation of ages).

📝 Assignment

🔍 Reflection

Discussion on **why NumPy is faster** and **real-world use cases**.