Lecture 5: NumPy (Arrays)

Core Concepts

Array metadata: a.dtype, a.shape, a.ndim, a.size, a.strides. Homogeneous dtypes for speed and vectorization.
Creation: np.array, np.zeros/ones/full, np.arange/linspace, np.random.rand/randn, np.eye, pd.Series.to_numpy().
Casting: a.astype(np.float32); beware precision tradeoffs.
Stacking: np.stack, np.concatenate, np.column_stack, np.hstack/vstack.
Axis semantics: Many reductions accept axis=0/1/... controlling dimension collapsed.
Aggregations: sum, mean, std, var, min, max, argmin, argmax (global or by axis).
Elementwise ufuncs: np.exp, log, sqrt, sin, cos, clip.
Boolean logic: (a>t) & (b<c), use &, |, ^, ~ (not/and/or).

Indexing/Slicing

- Basic: a[i], a[i,j], a[:,1:3], steps: a[::-1].
- Ex: axis 0: row, axis 1: column, axis 2: depth
- Boolean mask: a[a>0]; combine masks with &, |.
- Fancy indexing: a[[0,3,5]]; per-axis lists.
- Views vs copies: slices are often views; copy() when needed.
- Shape: np.transpose(), np.reshape(), np.flatten() copies, np.ravel() views

Broadcasting (Rules)

Cleaner and faster than loops. Compare dims from rightmost; dims equal or 1 are compatible. E.g., (3,1) + (1,4) → (3,4).
Tip: Add axes with None/np.newaxis: a[:,None].

Linear Algebra (Quick)

np.dot, @, np.linalg.inv, solve, eig, svd, norm. Shapes must agree.

I/O & Perf

np.save/load, np.savetxt/loadtxt; prefer vectorization; minimize Python loops; use numexpr/numba when applicable.

Lecture 6: Data Types & Control Flow (Updated)

Core Types & Ops

Immutable: int, float, bool, NoneType, str, tuple	Mutable: list, dict, set
Arithmetic: + - * / // % **	Comparison: == != < > <= >=
Identity: is, is not	Membership: in, not in

Use np.isclose(x, y, 1e-9) for comparing floating point (0.1 + 0.2 == 0.3)
NaN =/= None. NaN is Not a Number, numerical context like 0/0 and is float. None is it's own type with 1 value, None.

Strings (Handy)

s.lower(), upper(), strip(), split(), join(), find(), f-strings: f"v={v:.3f}".

Dict/Set Patterns

Dict: d.get(k, default), d.setdefault, d.update, del d[k], views: keys/values/items. d.pop(k), d.clear(), d.copy()
Set: {1,2}, ops: union |, inter &, diff -, xor ^.

Control Flow

If/Elif/Else: indentation defines blocks.
Loops: for x in iterable, while cond; break/continue.
Helpers: enumerate(seq, start=0), zip(a,b).

Comprehensions

List: [f(x) for x in xs if cond]
Dict: {k: f(v) for k,v in d.items() if cond}
Set: {g(x) for x in xs} Gen: (h(x) for x in xs).
Nesting: [(i,j) for i in I for j in J if filt].

Lecture 7: Functions, Scope & Testing

Function Mechanics

Def: def f(a, b=0, *args, **kwargs): return a+b
Lambda: square = lambda x: x*x
Args: positional vs keyword; defaults; keyword-only via *.
Docstrings: brief summary; args; returns; examples.
Pure vs impure: side effects (I/O, mutate external state) complicate testing.

Scope

LEGB rule: Local → Enclosing → Global → Builtins.
global/nonlocal when necessary (use sparingly).

Errors & Testing

Exceptions: try/except/else/finally; raise with raise ValueError("msg").
EAFP vs LBYL: try/except (Pythonic) vs pre-checks.
Assertions: assert f(2)==4, "square broken".
Design tests: small deterministic inputs; edge cases; property-based where useful.

Lecture 8: Classes & Style

OOP Essentials

Pattern:

```
class C:
    shared = 0                  # class attribute
    def __init__(self, x):     # instance init
        self.x = x
    def inc(self):              # instance method
        self.x += 1
    @classmethod
    def from_str(cls, s):      # alt constructor
        return cls(int(s))
    @staticmethod
    def add(a, b):
        return a + b
```

Inheritance/MRO

class D(C): ...; call super().__init__(...).
Method Resolution Order (MRO) defines lookup in multiple inheritance.

Dunder Methods (Common)

__repr__/ __str__ (debug/user text), __len__, __eq__/__lt__ (rich comparisons), __hash__, context manager: __enter__, __exit__.

Style & Tooling

PEP 8: naming, whitespace, line length, imports order.
Linters: flake8, pylint; Formatters: black, isort.
Pre-commit hooks automate checks on save/commit.

Quick Reference Tables

Truthiness (Common)

Falsey: 0, 0.0, 0j, "" (empty), [], {}, set()	Truthy: most other values

Common Pitfalls

- Floating-point: use math.isclose/np.isclose for comparisons.
- Mutable defaults: avoid def f(x, acc={}): ⇒ use None and create inside.
- Identity vs equality: is checks object identity, not value.
- Copy vs view in NumPy: modify views affects original.

Handy Snippets

```
# dict comprehension with condition
{w: len(w) for w in words if w.isalpha() and len(w)>3}

# safe get with default
score = d.get("score", 0)

# enumerate + zip
for i, (a,b) in enumerate(zip(A, B), 1):
    ...

# numpy: standardize columns by axis
X_std = (X - X.mean(axis=0)) / X.std(axis=0)

# context manager
with open("f.txt") as f:
    for line in f: ...
```

Best Practices

Vectorize with NumPy; manage mutability; use comprehensions/generators; keep classes small; document with docstrings; lint/autoformat; write modular tests; prefer pure functions for core logic; avoid deep inheritance; be explicit about side effects.