📘Complete NumPy Guide (Two Files)

Below are two Python files you can put in your GitHub repo:

- `numpy_basics.py` — Beginner to Intermediate concepts
- `numpy_advanced.py` — Advanced/Pro concepts

📄File 1: `numpy_basics.py`

```
"""
=====================================
 NumPy Beginner-Friendly Tutorial
-------------------------------------
 Author : Danish Khatri
 Date   : 31/07/2025 (updated)
 Desc   : Beginner to intermediate NumPy examples with explanations.
=====================================
"""

import numpy as np  # Import NumPy library

# ---------------------------
# 1) Array Creation
# ---------------------------
# Python list -> NumPy array (1D)
a = [1, 2, 3, 4, 5]
arr = np.array(a)
print("Python List:", a)
print("NumPy Array:", arr)

# 2D array from nested list
b = [[1, 2], [3, 4]]
brr = np.array(b)
print("2D NumPy Array:\n", brr)
print("Original Python Nested List:", b)

# ---------------------------
# 2) Range Functions
# ---------------------------
# np.arange(start, stop, step) -> stop is excluded
prr = np.arange(1, 20, 2)
print("np.arange(1, 20, 2):", prr)
```

```python
# np.linspace(start, stop, num_points) -> evenly spaced values including stop
c = np.linspace(1, 10, 6)
print("np.linspace(1, 10, 6):", c)

# ---------------------------
# 3) Zeros, Ones, Identity
# ---------------------------
# 1D zeros
d = np.zeros(10)
print("Zeros (1D):", d)

# 2D zeros (5x6)
e = np.zeros((5, 6))
print("Zeros (5x6):\n", e)

# 2D ones (5x6)
f = np.ones((5, 6))
print("Ones (5x6):\n", f)

# Identity matrix (diagonal 1s) of size 4x4
I = np.eye(4)
print("Identity (4x4):\n", I)

# ---------------------------
# 4) Random Number Generators
# ---------------------------
# Tip: Set seed for reproducibility
np.random.seed(42)

# Standard normal distribution (mean 0, std 1)
g = np.random.randn(5)
print("randn(5):", g)

# Uniform [0, 1) 3x3 matrix
u = np.random.rand(3, 3)
print("rand(3,3):\n", u)

# Random integers in [low, high) with specific count
h = np.random.randint(1, 10, 10)  # 10 integers from 1..9
print("randint(1,10,10):", h)

# Random choice from array
choices = np.random.choice([10, 20, 30], size=5, replace=True)
print("random choice:", choices)

# ---------------------------
# 5) Looping in Arrays (rarely needed; use vectorization when possible)
# ---------------------------
```

```
a1 = np.array([1, 2, 3, 4, 5])
print("Looping elements:")
for val in a1:
    print(val)

# ---------------------------
# 6) Join, Stack and Split
# ---------------------------
a2 = np.array([1, 2])
b2 = np.array([3, 4])

# Concatenate into 1D
c3 = np.concatenate((a2, b2))
print("Concatenate [1,2] + [3,4] ->", c3)

# Split into 2 nearly equal parts
d2 = np.array_split(c3, 2)
print("Split into 2 parts:", d2)

# Vertical stack (row-wise)
v_stack = np.vstack((a2, b2))
print("vstack ->\n", v_stack)

# Horizontal stack (col-wise for 1D -> concatenation)
h_stack = np.hstack((a2, b2))
print("hstack ->", h_stack)

# ---------------------------
# 7) Searching, Sorting, Filtering
# ---------------------------
h1 = np.array([23, 44, 33, 45, 6, 77, 12, 445])

# Where equals 44 (returns tuple of indices per axis)
idx_44 = np.where(h1 == 44)
print("Index of 44:", idx_44)

# Sorting (returns a new sorted array)
print("Sorted h1:", np.sort(h1))

# Boolean filtering (keep values > 30)
print("Values > 30:", h1[h1 > 30])

# ---------------------------
# 8) Basic Statistics & Math
# ---------------------------
n1 = np.array([10, 20, 30, 770, 660, 90])
print("Max:", n1.max())
print("Min:", n1.min())
```

```python
print("Argmax (index of max):", n1.argmax())
print("Argmin (index of min):", n1.argmin())

n2 = np.array([4, 9])
print("sqrt:", np.sqrt(n2))
print("exp:", np.exp(n2))
print("log (natural):", np.log(n2))

print("Mean:", np.mean(n1))
print("Median:", np.median(n1))
print("Std Dev:", np.std(n1))

# ----------------------------
# 9) Reshape & Slicing
# ----------------------------
arr2 = np.arange(1, 13)  # 1..12
print("Original:", arr2)

# Reshape to 3 rows x 4 cols
reshaped = arr2.reshape(3, 4)
print("Reshaped (3x4):\n", reshaped)

# Reshape with -1 lets NumPy infer dimension automatically
reshaped_auto = arr2.reshape(2, -1)  # 2 rows, infer columns
print("Reshaped (2x?):\n", reshaped_auto)

# Slicing examples
print("First 5:", arr2[:5])
print("Last 3:", arr2[-3:])
print("Every 2nd:", arr2[::2])

# ----------------------------
# 10) Broadcasting
# ----------------------------
arr3 = np.array([1, 2, 3])
print("arr3:", arr3)
print("arr3 + 5:", arr3 + 5)  # scalar adds to every element

M = np.ones((3, 3))
print("M + arr3 (row-wise broadcast):\n", M + arr3)

print("\n✅ NumPy BASICS completed successfully!")
```

📄 File 2: `numpy_advanced.py`

```
"""
=====================================
 NumPy Advanced Tutorial (Pro)
-------------------------------------
 Author : Danish Khatri
 Date   : 31/07/2025 (updated)
 Desc   : Advanced NumPy concepts for interviews and ML projects.
=====================================
"""

import numpy as np

# Keep results reproducible
np.random.seed(123)

# ---------------------------------
# 1) Dtypes & Type Casting
# ---------------------------------
arr = np.array([1, 2, 3], dtype=np.int32)  # specify dtype
print("dtype:", arr.dtype)

# Cast to another dtype (creates a copy)
arr_f = arr.astype(np.float64)
print("astype -> float64:", arr_f, arr_f.dtype)

# Mixed types auto-upcast
mixed = np.array([1, 2.5, True, 7])
print("mixed upcast ->", mixed, mixed.dtype)

# ---------------------------------
# 2) Array Attributes
# ---------------------------------
A = np.arange(24).reshape(4, 6)
print("A shape:", A.shape)     # (rows, cols)
print("A ndim:", A.ndim)       # number of dimensions
print("A size:", A.size)       # total elements
print("A itemsize:", A.itemsize)  # bytes per element
print("A nbytes (total bytes):", A.nbytes)  # itemsize * size

# ---------------------------------
# 3) Copy vs View
# ---------------------------------
base = np.array([10, 20, 30, 40])
view = base.view()          # view shares memory
```

```
copy = base.copy()          # copy is independent

view[0] = 999               # changes reflect in base
copy[1] = 777               # does NOT affect base
print("base after edits:", base)
print("view shares memory:", view)
print("copy independent:", copy)

# ---------------------------------
# 4) Flatten vs Ravel
# ---------------------------------
B = np.arange(12).reshape(3, 4)
flat_copy = B.flatten()     # returns a copy
flat_view = B.ravel()       # returns a view when possible

B[0, 0] = 111
print("B after change:\n", B)
print("flatten (copy unaffected):", flat_copy)
print("ravel (view may reflect):", flat_view)

# ---------------------------------
# 5) Fancy Indexing & Boolean Masks
# ---------------------------------
C = np.arange(10)
idx = [2, 5, 7]
print("Fancy index C[idx]:", C[idx])

mask = C % 2 == 0  # even numbers mask
print("Boolean mask:", mask)
print("C[mask] (even values):", C[mask])

# ---------------------------------
# 6) Matrix Operations (Linear Algebra)
# ---------------------------------
M = np.array([[1., 2.], [3., 4.]])
N = np.array([[5., 6.], [7., 8.]])

# Element-wise operations
print("M + N:\n", M + N)
print("M * N (element-wise):\n", M * N)

# Matrix multiplication
print("M @ N (matmul):\n", M @ N)           # same as np.matmul(M, N)
print("dot product (flattened):", np.dot(M.ravel(), N.ravel()))

# Transpose, inverse, determinant, trace, rank
tM = M.T
print("Transpose M^T:\n", tM)
print("det(M):", np.linalg.det(M))
print("inv(M):\n", np.linalg.inv(M))
print("trace(M):", np.trace(M))
print("matrix_rank(M):", np.linalg.matrix_rank(M))

# Solve linear system Mx = b
b = np.array([1., 0.])
x = np.linalg.solve(M, b)
print("Solve Mx=b -> x:", x)

# Eigenvalues & eigenvectors
e_vals, e_vecs = np.linalg.eig(M)
print("Eigenvalues:", e_vals)
print("Eigenvectors (columns):\n", e_vecs)

# ---------------------------------
# 7) Unique & Set Operations
# ---------------------------------
S = np.array([1, 2, 2, 3, 3, 3, 4])
print("unique:", np.unique(S))
print("return_index:", np.unique(S, return_index=True))
print("return_counts:", np.unique(S, return_counts=True))

A1 = np.array([1, 2, 3, 4])
B1 = np.array([3, 4, 5, 6])
print("intersect1d:", np.intersect1d(A1, B1))
print("union1d   :", np.union1d(A1, B1))
print("setdiff1d :", np.setdiff1d(A1, B1))  # in A1 not in B1
print("setxor1d  :", np.setxor1d(A1, B1))   # symmetric difference

# ---------------------------------
# 8) Clipping & Rounding
# ---------------------------------
D = np.array([-3.2, -1.5, 0.2, 1.4, 5.9])
print("clip to [0, 2]:", np.clip(D, 0, 2))
print("round :", np.round(D, 1))
print("floor :", np.floor(D))
print("ceil  :", np.ceil(D))

# ---------------------------------
# 9) Handling NaN / Inf & Masked Arrays
# ---------------------------------
E = np.array([1.0, np.nan, 2.5, np.inf, -np.inf, 3.0])
print("isnan:", np.isnan(E))
print("isfinite:", np.isfinite(E))

# nan-aware reductions
print("nanmean:", np.nanmean(E))
```

```python
print("nanmax :", np.nanmax(E))

# Replace NaN/Inf with numbers
E_clean = np.nan_to_num(E, nan=0.0, posinf=1e9, neginf=-1e9)
print("nan_to_num ->", E_clean)

# Masked arrays (mask bad values)
masked = np.ma.masked_invalid(E)
print("masked array:", masked)
print("masked mean (ignores masked):", masked.mean())

# ---------------------------------
# 10) Saving & Loading Arrays
# ---------------------------------
F = np.arange(6).reshape(2, 3)

# Binary .npy (fast, preserves dtype & shape)
np.save("array_F.npy", F)
loaded_F = np.load("array_F.npy")
print("Loaded from .npy:\n", loaded_F)

# Text format (human-readable; may lose precision)
np.savetxt("array_F.txt", F, fmt="%d")
loaded_txt = np.loadtxt("array_F.txt", dtype=int)
print("Loaded from .txt:\n", loaded_txt)

# ---------------------------------
# 11) Broadcasting Tricks
# ---------------------------------
# Add a column vector to each row using np.newaxis
row = np.array([1, 2, 3])      # shape (3,)
col = np.array([10, 20, 30])   # shape (3,)

# Make col into (3,1) then broadcast to (3,3)
result = col[:, np.newaxis] + row  # shapes: (3,1) + (3,) -> (3,3)
print("Broadcast add col->rows:\n", result)

# ---------------------------------
# 12) Tiling & Repeating
# ---------------------------------
base = np.array([1, 2, 3])
print("repeat each element 3x:", np.repeat(base, 3))
print("tile the array 2x:", np.tile(base, 2))

# ---------------------------------
# 13) Structured Arrays (quick glimpse)
# ---------------------------------
people = np.array([
    ("Alice", 25, 55.5),
    ("Bob",   30, 72.3),
], dtype=[("name", "U10"), ("age", "i4"), ("weight", "f4")])

print("structured names:", people["name"])   # field access
print("age > 26:", people[people["age"] > 26])

print("\n🚯 NumPy ADVANCED completed successfully!")
```

🔗How to Use

1. Create two files in your repo: `numpy_basics.py` and `numpy_advanced.py`.
2. Run them separately to see outputs:

   ```
   python numpy_basics.py
   python numpy_advanced.py
   ```

3. (Optional) Add a `README.md` explaining both files and topics covered.

📌Tip

- Keep screenshots of outputs to showcase on LinkedIn.
- Add a license (`MIT`) and a short project description for a professional touch.