NUMPY 빠른 참조

배열 생성, 수학 연산, 선형 대수 등

배열 생성

리스트로부터

```
import numpy as np
a = np.array([1, 2, 3])        # 1D
b = np.array([[1, 2], [3, 4]])  # 2D
```

내장 생성자

```
np.zeros((2, 3))     # 2x3 of zeros
np.ones((3, 3))      # 3x3 of ones
np.eye(4)            # 4x4 identity matrix
np.arange(0, 10, 2)  # [0, 2, 4, 6, 8]
np.linspace(0, 1, 5) # 5 evenly spaced
```

배열 속성

`a.shape` 튜플로 나타낸 차원: `(3, 4)`
`a.ndim` 차원 수
`a.size` 전체 원소 수
`a.dtype` 데이터 타입: `float64`, `int32` 등

인덱싱 및 슬라이싱

기본 인덱싱

```
a = np.array([[1, 2, 3], [4, 5, 6]])
a[0, 1]    # 2 (row 0, col 1)
a[1]       # [4, 5, 6] (row 1)
a[:, 0]    # [1, 4] (all rows, col 0)
```

슬라이싱

```
a[0, 1:]   # [2, 3] (row 0, col 1 onward)
a[:, :2]   # first 2 columns
a[::2]     # every other row
```

불리언 인덱싱

```
a = np.array([10, 20, 30, 40])
a[a > 15]   # [20, 30, 40]
a[a % 20 == 0]  # [20, 40]
```

배열 연산

원소별 연산

```
a = np.array([1, 2, 3])
a + 10    # [11, 12, 13]
a * 2     # [2, 4, 6]
a ** 2    # [1, 4, 9]
a + a     # [2, 4, 6]
```

비교

```
a = np.array([1, 2, 3, 4])
a > 2          # [False, False, True, True]
np.where(a > 2, a, 0)  # [0, 0, 3, 4]
```

집계

`a.sum()`	모든 원소의 합
`a.mean()`	산술평균
`a.std()`	표준 편차
`a.min()` / `a.max()`	최솟값 / 최댓값
`a.argmin()` / `a.argmax()`	최솟값 / 최댓값의 인덱스
`a.cumsum()`	누적 합

축별 결과를 위해 `axis=0` (열) 또는 `axis=1` (행) 추가

수학 함수

주요 함수

`np.sqrt(a)`	각 원소의 제곱근
`np.abs(a)`	절댓값
`np.exp(a)`	각 원소의 e^x
`np.log(a)`	자연로그 (ln)
`np.log10(a)`	상용로그 (밑 10)
`np.sin(a)` / `np.cos(a)`	삼각 함수 (라디안)
`np.round(a, 2)`	소수점 2자리로 반올림
`np.clip(a, lo, hi)`	값을 [lo, hi] 범위로 제한

선형 대수

행렬 연산

```
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])
A @ B             # matrix multiply
np.dot(A, B)      # same as A @ B
A.T               # transpose
```

분해 및 풀기

```
np.linalg.inv(A)       # inverse
np.linalg.det(A)       # determinant
np.linalg.eig(A)       # eigenvalues/vectors
np.linalg.solve(A, b)  # solve Ax = b
```

난수

난수 생성

```
rng = np.random.default_rng(42)  # seeded
rng.random((2, 3))     # uniform [0, 1)
rng.integers(1, 10, 5) # 5 ints in [1, 10)
rng.normal(0, 1, 100)  # 100 from N(0,1)
rng.choice([1, 2, 3], size=2)  # sample
```

레거시 API

```
np.random.seed(42)
np.random.rand(3, 3)     # uniform 3x3
np.random.randn(3, 3)    # standard normal
np.random.shuffle(arr)   # in-place shuffle
```

형태 변환

형태 조작

```
a = np.arange(12)
a.reshape(3, 4)    # 3x4 matrix
a.reshape(3, -1)   # infer columns
a.flatten()        # back to 1D (copy)
a.ravel()          # back to 1D (view)
```

스택 및 분할

```
np.vstack([a, b])  # stack vertically
np.hstack([a, b])  # stack horizontally
np.concatenate([a, b], axis=0)
np.split(a, 3)     # split into 3 parts
```

브로드캐스팅

브로드캐스팅 동작 방식

```
a = np.array([[1, 2, 3],
              [4, 5, 6]])  # shape (2,3)
b = np.array([10, 20, 30]) # shape (3,)
a + b  # b broadcasts to (2,3)
```

규칙

규칙 1 랭크가 맞을 때까지 짧은 형태에 1을 앞에 추가
규칙 2 크기가 같거나 하나가 1이면 차원 일치
규칙 3 크기 1인 차원을 상대 크기에 맞게 확장

파일 I/O

NumPy 바이너리

```
np.save("data.npy", arr)     # single array
arr = np.load("data.npy")
np.savez("data.npz", a=x, b=y) # multiple
d = np.load("data.npz"); d["a"]
```

텍스트 파일

```
np.savetxt("data.csv", arr, delimiter=",")
arr = np.loadtxt("data.csv", delimiter=",")
arr = np.genfromtxt("data.csv", delimiter=",",
                    skip_header=1)
```

일반 패턴

[0, 1]로 정규화

```
normalized = (a - a.min()) / (a.max() - a.min())
```

유클리드 거리

```
dist = np.sqrt(np.sum((a - b) ** 2))
# or: np.linalg.norm(a - b)
```

고유값 및 빈도

```
vals, counts = np.unique(a, return_counts=True)
dict(zip(vals, counts))
```

정렬

```
np.sort(a)             # sorted copy
idx = np.argsort(a)    # indices that sort
a[idx]                 # apply sort order
```