NumPy Intermediate Cheat Sheet

1. Views vs. Copies

A critical concept. Slicing creates a **view** (shares memory with original). Use `.copy()` for a new, independent array.

```
a = np.arange(10)
# Create a view (a slice is a view)
b = a[0:5]
b[0] = 99 # Modify the view
# The original array 'a' is also changed!
# a is now: array([99, 1, 2, 3, 4, 5, 6, 7, 8, 9])

# Create a copy
c = a.copy()
c[0] = 0 # Modify the copy
# The original array 'a' is NOT changed.
```

2. Advanced Indexing

Use arrays of indices to select elements. This always creates a **copy**, not a view.

Integer Array Indexing

Select specific elements using lists or arrays of integers.

```
x = np.array([10, 20, 30, 40, 50, 60])
indices = [1, 3, 4]
x[indices] # Returns: array([20, 40, 50])

y = np.array([[1,2,3], [4,5,6], [7,8,9]])
# Select specific rows
y[[0, 2]] # Returns rows 0 and 2
# [[1, 2, 3],
#  [7, 8, 9]]
```

Indexing with Index Pairs

Select elements at specific (row, col) coordinates.

```
# Select elements at (0,1), (1,2), and (2,0)
z = y[[0, 1, 2], [1, 2, 0]]
# Returns: array([2, 6, 7])
```

3. Broadcasting

How NumPy treats arrays with different shapes during arithmetic operations.

Broadcasting Rules

1. If arrays have different numbers of dimensions, prepend 1s to the shape of the smaller array.
2. If the shape of two arrays does not match in any dimension, the array with shape equal to 1 in that dimension is stretched to match the other shape.
3. If in any dimension the sizes disagree and neither is 1, an error is raised.

Example

Add a 1D array to each row of a 2D array.

```
# a: (3, 3) matrix, b: (3,) vector
a = np.array([[1,2,3], [4,5,6], [7,8,9]])
b = np.array([10, 20, 30])

# b is "broadcast" across each row of a
c = a + b
# c becomes:
# [[11, 22, 33],
#  [14, 25, 36],
#  [17, 28, 39]]
```

To broadcast a column vector, reshape it first.

```
col = np.array([[10],[20],[30]]) # shape (3, 1)
a + col # Adds column to each column of a
```

4. Advanced Manipulation

Splitting Arrays

The opposite of stacking.

```
a = np.arange(9).reshape(3, 3)
# Split into 3 sub-arrays vertically
np.vsplit(a, 3)
# [array([[0,1,2]]), array([[3,4,5]]), ...]

# Split into 3 sub-arrays horizontally
np.hsplit(a, 3)
# [array([[0],[3],[6]]), array([[1],[4],[7]]), ...]
```

Inserting and Deleting

```
a = np.array([[1,2], [3,4]])

# Insert row [5,6] at index 1
np.insert(a, 1, [5, 6], axis=0)
# [[1, 2],
#  [5, 6],
#  [3, 4]]

# Delete column at index 1
np.delete(a, 1, axis=1)
# [[1],
#  [3]]
```

Finding Unique Elements

```
x = np.array([1, 2, 1, 3, 5, 2, 1])
np.unique(x) # Returns: array([1, 2, 3, 5])

# Get unique elements and their counts
vals, counts = np.unique(x, return_counts=True)
# vals:   [1, 2, 3, 5]
# counts: [3, 2, 1, 1]
```

5. Conditional Logic & UFuncs

Vectorized Conditional Logic

Use `np.where()` as a vectorized if-else statement.

```
# np.where(condition, value_if_true, value_if_false)
a = np.arange(10)

# Replace all odd numbers with -1
np.where(a % 2 == 1, -1, a)
# Returns: [0, -1, 2, -1, 4, -1, 6, -1, 8, -1]
```

NumPy Intermediate Cheat Sheet

UFunc Methods

Universal functions have useful methods.

```
x = np.array([1, 2, 3, 4])

# .reduce() applies op until 1 value is left
# Equivalent to x.sum()
np.add.reduce(x) # 1+2+3+4 = 10

# .accumulate() stores intermediate results
# Equivalent to np.cumsum()
np.add.accumulate(x) # [1, 1+2, 1+2+3, ...]
# Returns: [ 1,  3,  6, 10]
```

6. Structured Arrays

Arrays whose elements are like C structs or database rows, with named fields of different data types.

```
# Define a structured data type
dt = np.dtype([('name', 'S10'), ('age', 'i4')])

# Create a structured array
data = np.array([('Alice', 25), ('Bob', 30)],
                dtype=dt)

# Access data by field name
data['name'] # Returns: array([b'Alice', b'Bob'],
    ...)
data['age']  # Returns: array([25, 30], dtype=int32)

# Access a single record
data[0] # Returns: (b'Alice', 25)
```

7. Linear Algebra ('numpy.linalg')

Core linear algebra operations.

```
A = np.array([[1, 2], [3, 4]])
b = np.array([5, 11])

# Matrix determinant
np.linalg.det(A) # Returns: -2.0

# Matrix inverse
A_inv = np.linalg.inv(A)
# [[-2. ,  1. ],
#  [ 1.5, -0.5]]

# Solve a linear system: Ax = b
x = np.linalg.solve(A, b) # Returns: [1., 2.]

# Eigenvalues and eigenvectors
eigvals, eigvecs = np.linalg.eig(A)
```

8. File I/O

Saving and loading NumPy data.

Binary '.npy' format

Efficient way to save/load a single NumPy array.

```
a = np.arange(10)
np.save('my_array.npy', a)
b = np.load('my_array.npy')
```

Text format

Human-readable, for CSV-like files.

```
x = np.array([[1,2,3], [4,5,6]])

# Save to a text file
np.savetxt('my_data.csv', x, delimiter=',')

# Load from a text file
y = np.loadtxt('my_data.csv', delimiter=',')
```