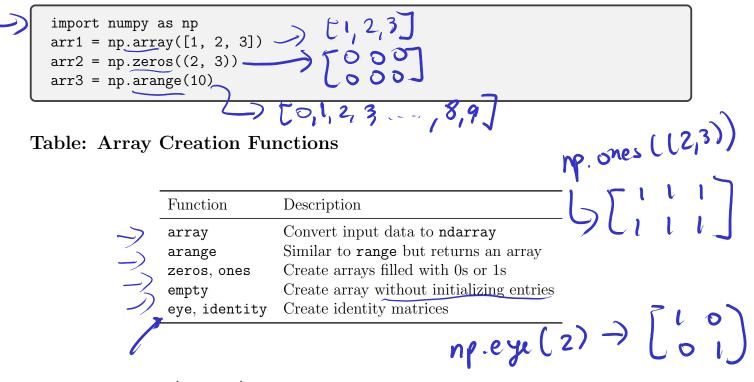
# **Numpy Basics**

#### Overview

NumPy is the foundational package for numerical computing in Python. It provides the ndarray, a fast and space-efficient multidimensional array, along with capabilities for vectorized computation, broadcasting, mathematical operations, file I/O, and linear algebra.

### **Creating Arrays**

Arrays are created using functions like np.array, np.zeros, np.ones, and np.arange.



### Data Types (dtype)

NumPy supports several data types, which can be set using the dtype parameter.

arr = np.array([1.5, 2.3, 3.1], dtype=np.float32)
$$X = \text{np.array}([1, 5, 2.3, 3.1], dtype = \text{np. int } 32)$$

Type	Description
int32, int64	Integer types
float32, float64	Floating point
bool	Boolean
object	Generic Python objects

"one" o spect

### **Array Operations and Indexing**

Vectorized operations eliminate the need for explicit loops.

```
arr = np.array([1, 2, 3])
arr * 2 # array([2, 4, 6])
```

#### Indexing and slicing:

```
arr = np.arange(10) t_0, t_1, t_2, t_3, t_4, t_5, arr [5:8] = 100 # modifies elements at indices 5, 6, 7
```

[0,1,2,3,100,100,100,8,9

## قریاری Boolean Indexing

Boolean indexing allows you to filter or select data from an array using boolean conditions. It is a powerful feature for subsetting arrays without the need for loops. Example: selecting rows based on a condition.

```
names = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'])
data = np.random.randn(7, 4) # 7x4 matrix of standard normal values

# Get rows where names == 'Bob'
data[names == 'Bob']

TRUE FALSE FALSE TRUE FALSE FALSE
```

# Fancy Indexing

Fancy indexing allows indexing using arrays of integers. It creates copies rather than views, and supports powerful reordering and extraction capabilities. Example, reassign values

Using negative indices:

arr[[-1, -3]]

Selecting multiple elements (advanced):

arr = np.arange(32).reshape((8, 4))
# Select elements at (1, 0), (5, 3), and (7, 1)
arr[[1, 5, 7], [0, 3, 1]]

Transposing  $\begin{bmatrix} 123 \\ 456 \end{bmatrix}$   $\begin{bmatrix} 14 \\ 25 \\ 36 \end{bmatrix}$ 

Transposing is a special form of reshaping that similarly returns a view on the underlying data without copying anything.

arr.T # Transpose

### Universal Functions (ufuncs)

Universal functions (ufuncs) are vectorized wrappers for simple functions. They perform element-wise operations and are much faster than Python loops.

## Unary ufuncs

Operate on a single array.

[-1, 25, -2, 4) ->	np. aus Cl., Co,
ray.	J [1,25,2,4]

Function	Description
np.abs, np.fabs	Absolute value
np.sqrt	Square root
np.square	Element-wise square
np.exp	Exponential
np.log, np.log10	Natural / base-10 logarithm
np.sign	Sign of number $(+1, 0, -1)$

Table 1: Common unary ufuncs

arr = np.array([1, 2, 3, -4])
np.sqrt(np.abs(arr)) # Safe square root

5 [1, 2, 3,4] 5 [1, 12, 13, 2]

#### Binary ufuncs

Operate on two arrays (or array and scalar).

	Function	Description
7777	np.add np.subtract np.multiply np.divide np.maximum np.minimum np.mod	Addition Subtraction Multiplication Division Element-wise maximum Element-wise minimum Modulo operation

Table 2: Common binary ufuncs

**Example:** element-wise operations

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

#### Performance

Ufuncs are implemented in C, enabling fast execution without explicit Python loops. This makes them ideal for large-scale numerical computation. Comparison with loop:

```
# Slower with loop
result = [math.sqrt(abs(x)) for x in arr]

# Faster with ufunc
np.sqrt(np.abs(arr))
```