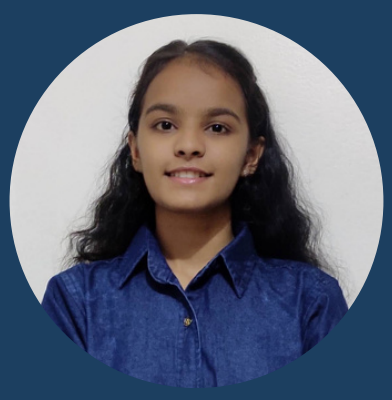
Fundamentals of Python

NUMPY



Shruti Bharat

@shrutibharat0105

Table of Contents

Numpy

- Numpy is written in c language.
- By default value of numpy array is float.
- At the core of the numy package it is the ndarray object. This encapsulates n-dimensional arrays of homogenous data types.
- Numpy array have a fixed sized at creation. Unlike python lists which can grow dynamically.
- Changing the size of ndarray will create a new arrat & delete the original.
- Numpy array are faster than python list.
- Numpy array occupies less memory than python list.
- Numpy array is c-array and static while python list is referential array and dyanamic.

Creating a Numpy array

```python
import numpy as np

# 1. Creating an array from a list (1D array/Vector)
array_from_list = np.array([1, 2, 3, 4, 5])

# 2. Creating an array with a range of values
array_with_range = np.arange(0, 10, 2)

# 3. Creating and reshaping an array with arange and reshape
array_arange_reshape = np.arange(1, 13).reshape((3, 4))

# 4. Creating a 2D array (matrix) from a list of lists
array_2d = np.array([[1, 2, 3], [4, 5, 6]])

# 5. Creating a 3D array (tensor)
array_3d = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])

# 6. Creating an array of zeros
array_of_zeros = np.zeros((3, 3))

# 7. Creating an array of ones
array_of_ones = np.ones((2, 2))

# 8. Creating an array of a constant value
array_full = np.full((2, 2), 7)

# 9. Creating an identity matrix
identity_matrix = np.identity(3)

# 10. Creating an array with evenly spaced values
array_with_linspace = np.linspace(0, 1, 5)

# 11. Creating an array with random values
array_with_random = np.random.random((3, 3))

# 12. Creating an empty array (uninitialized)
empty_array = np.empty((2, 2))

# 13. Creating an array with a specific data type
array_with_dtype = np.array([1, 2, 3, 4], dtype=np.float32)
```

Array attributes

```python
import numpy as np

# Creating a sample 3D array
array = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])

# Array attributes
array_ndim = array.ndim    # Number of dimensions
array_shape = array.shape  # Shape of the array (dimensions)
array_size = array.size    # Total number of elements in the array
array_dtype = array.dtype  # Data type of the elements
array_itemsize = array.itemsize  # Size in bytes of each element

# Changing the data type of the array elements using astype
array_float = array.astype(np.float64)
```

Array Operations

```python
import numpy as np

# Creating sample arrays
array1 = np.array([1, 2, 3, 4, 5])
array2 = np.array([5, 4, 3, 2, 1])

# Scalar operations
scalar_add = array1 + 10  # Adding a scalar to an array
scalar_mul = array1 * 2   # Multiplying an array by a scalar

# Relational operations
relational_gt = array1 > 2  # Checking which elements are greater than 2
relational_eq = array1 == array2  # Checking element-wise equality

# Vector operations (shape of array1 and array2 must be same)
vector_add = array1 + array2  # Element-wise addition
vector_mul = array1 * array2  # Element-wise multiplication
vector_dot = np.dot(array1, array2)  # Dot product
```

Reshaping

```python
import numpy as np

# Creating a sample array
array = np.arange(1, 13)  # Array with elements from 1 to 12
print("Original array:\n", array)

# Reshaping the array
reshaped_array = array.reshape(3, 4)  # Reshape to 3x4 matrix
print("\nReshaped array (3x4):\n", reshaped_array)

# Transposing the array
transposed_array = reshaped_array.transpose()  # Transpose the reshaped array
print("\nTransposed array (4x3):\n", transposed_array)

# Flattening the array using ravel
flattened_array = reshaped_array.ravel()  # Flatten the reshaped array
print("\nFlattened array (ravel):\n", flattened_array)


# Output
# Original array:
# [ 1  2  3  4  5  6  7  8  9 10 11 12]

# Reshaped array (3x4):
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]

# Transposed array (4x3):
# [[ 1  5  9]
#  [ 2  6 10]
#  [ 3  7 11]
#  [ 4  8 12]]

# Flattened array (ravel):
# [ 1  2  3  4  5  6  7  8  9 10 11 12]
```

Stacking & Splitting

```python
import numpy as np

# Creating a sample array
array = np.arange(1, 13)  # Array with elements from 1 to 12
print("Original array:\n", array)

# Reshaping the array
reshaped_array = array.reshape(3, 4)  # Reshape to 3x4 matrix
print("\nReshaped array (3x4):\n", reshaped_array)

# Transposing the array
transposed_array = reshaped_array.transpose()  # Transpose the reshaped array
print("\nTransposed array (4x3):\n", transposed_array)

# Flattening the array using ravel
flattened_array = reshaped_array.ravel()  # Flatten the reshaped array
print("\nFlattened array (ravel):\n", flattened_array)


# Output
# Original array:
# [ 1  2  3  4  5  6  7  8  9 10 11 12]

# Reshaped array (3x4):
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]

# Transposed array (4x3):
# [[ 1  5  9]
#  [ 2  6 10]
#  [ 3  7 11]
#  [ 4  8 12]]

# Flattened array (ravel):
# [ 1  2  3  4  5  6  7  8  9 10 11 12]
```

Stacking & Splitting

```
h_stack=[[1 2 5 6]
         [3 4 7 8]]

v_stack=[[1 2]
         [3 4]
         [5 6]
         [7 8]]

# Horizontal splitting
h_split = np.hsplit(h_stack, 2)
print("\nHorizontal Split:")

# Output
# Horizontal Split:
# [[1 2]
#  [3 4]]
# [[5 6]
#  [7 8]]

# Vertical splitting
v_split = np.vsplit(v_stack, 2)
print("\nVertical Split:")
for i, split in enumerate(v_split, start=1):
    print(f"Split {i}:\n{split}")

# Output
# Vertical Split:
# [[1 2]
#  [3 4]]
# [[5 6]
#  [7 8]]
```

Indexing & Slicing

```python
import numpy as np

# Create a sample 2D numpy array
array = np.array([[1, 2, 3, 4, 5],
                  [6, 7, 8, 9, 10],
                  [11, 12, 13, 14, 15],
                  [16, 17, 18, 19, 20]])
print("Original Array:")
print(array)


# Indexing a single element
element = array[1, 2]  # Accessing the element at 2nd row, 3rd column
print("\nElement at (1, 2):")
print(element)    # Element at (1, 2): 8


# Slicing a subarray
subarray = array[1:3, 2:5]  # Slicing rows 2 to 3 and columns 3 to 5
print("\nSubarray (rows 1-2, columns 2-4):")
print(subarray)   # Subarray (rows 1-2, columns 2-4):
                  # [[ 8  9 10]
                  #  [13 14 15]]


# Slicing using step
step_slice = array[::2, ::2]  # Slicing with a step of 2
print("\nArray with step slicing (every 2nd element):")
print(step_slice) # Array with step slicing (every 2nd element):
                  # [[ 1  3  5]
                  #  [11 13 15]]


# Boolean indexing
boolean_index = array[array > 10]  # Elements greater than 10
print("\nElements greater than 10:")
print(boolean_index)  # Elements greater than 10:
                      # [11 12 13 14 15 16 17 18 19 20]
```

Indexing & Slicing

```python
# Fancy indexing
fancy_index = array[[0, 2], [1, 3]]  # Accessing specific elements
print("\nFancy indexing (elements at (0, 1) and (2, 3)):")
print(fancy_index)  # Fancy indexing (elements at (0, 1) and (2, 3)): [ 2 14]

# Modifying elements using indexing
array[1, 2] = 99  # Changing element at (1, 2) to 99
print("\nArray after modifying element at (1, 2) to 99:")
print(array)
# Array after modifying element at (1, 2) to 99:
# [[ 1  2  3  4  5]
#  [ 6  7 99  9 10]
#  [11 12 13 14 15]
#  [16 17 18 19 20]]

# Modifying elements using slicing
array[1:3, 2:4] = [[55, 66], [77, 88]]  # Changing subarray elements
print("\nArray after modifying subarray:")
print(array)
# Array after modifying subarray:
# [[ 1  2  3  4  5]
#  [ 6  7 55 66 10]
#  [11 12 77 88 15]
#  [16 17 18 19 20]]
```

Array functions

```python
import numpy as np

# Creating sample arrays
array1 = np.array([1, 2, 3, 4, 5])
array2 = np.array([5, 4, 3, 2, 1])

# Array functions
array_max = array1.max()                # Maximum value in the array
print(f"Max: {array_max}")              # Max: 5

array_min = array1.min()                # Minimum value in the array
print(f"Min: {array_min}")              # Min: 1

array_sum = array1.sum()                # Sum of all elements in the array
print(f"Sum: {array_sum}")              # Sum: 15

array_mean = array1.mean()              # Mean of the array elements
print(f"Mean: {array_mean}")            # Mean: 3.0

array_std = array1.std()                # Standard deviation of the array elements
print(f"Std: {array_std}")              # Std: 1.4142135623730951

array_var = array1.var()                # Variance of the array elements
print(f"Var: {array_var}")              # Var: 2.0

array_argmax = array1.argmax()          # Index of the maximum value
print(f"Argmax: {array_argmax}")        # Argmax: 4

array_argmin = array1.argmin()          # Index of the minimum value
print(f"Argmin: {array_argmin}")        # Argmin: 0

array_cumsum = array1.cumsum()          # Cumulative sum of the elements
print(f"Cumsum: {array_cumsum}")        # Cumsum: [ 1  3  6 10 15]

array_cumprod = array1.cumprod()        # Cumulative product of the elements
print(f"Cumprod: {array_cumprod}")      # Cumprod: [ 1  2  6 24 120]
```

Array functions

```python
# Trigonometric functions
array_sin = np.sin(array1)                # Sine of the elements
print(f"Sin: {array_sin}")
# Sin: [ 0.84147098  0.90929743  0.14112001 -0.7568025  -0.95892427]

array_cos = np.cos(array1)                # Cosine of the elements
print(f"Cos: {array_cos}")
# Cos: [ 0.54030231 -0.41614684 -0.9899925  -0.65364362  0.28366219]

array_tan = np.tan(array1)                # Tangent of the elements
print(f"Tan: {array_tan}")
# Tan: [ 1.55740772 -2.18503986 -0.14254654  1.15782128 -3.38051501]

# Logarithmic and exponential functions
array_log = np.log(array1)                # Natural log of the elements
print(f"Log: {array_log}")
# Log: [0.         0.69314718 1.09861229 1.38629436 1.60943791]

array_exp = np.exp(array1)                # Exponential of the elements
print(f"Exp: {array_exp}")
# Exp: [  2.71828183   7.3890561   20.08553692  54.59815003 148.4131591 ]

# Rounding functions
array_round = np.round(array1)            # Round elements to nearest integer
print(f"Round: {array_round}")            # Round: [1 2 3 4 5]

array_floor = np.floor(array1)            # Floor of the elements
print(f"Floor: {array_floor}")            # Floor: [1. 2. 3. 4. 5.]

array_ceil = np.ceil(array1)              # Ceil of the elements
print(f"Ceil: {array_ceil}")              # Ceil: [1. 2. 3. 4. 5.]

# Sorting functions
array_sorted = np.sort(array2)            # Sort elements
print(f"Sorted: {array_sorted}")          # Sorted: [1 2 3 4 5]
```

Array functions

```python
import numpy as np

# Creating sample arrays
array1 = np.array([1, 2, 3, 4, 5])
array2 = np.array([5, 4, 3, 2, 1])
array3 = np.array([[1, 2], [3, 4]])
array4 = np.array([[5, 6], [7, 8]])

# Append and concatenate functions
array_appended = np.append(array1, array2)  # Append arrays
print(f"Appended: {array_appended}")        # Appended: [1 2 3 4 5 5 4 3 2 1]

array_concatenated = np.concatenate((array3, array4), axis=0)  # Concatenate arrays
print(f"Concatenated:\n{array_concatenated}")
# Concatenated: [[1 2] [3 4] [5 6] [7 8]]

# Unique function
array_unique = np.unique(array_appended)  # Unique elements
print(f"Unique: {array_unique}")          # Unique: [1 2 3 4 5]

# Expanding dimensions
array_expanded = np.expand_dims(array1, axis=0)  # Expand dimensions
print(f"Expanded:\n{array_expanded}")  # Expanded: [[1 2 3 4 5]]

# Conditional selection
array_where = np.where(array1 > 2, array1, -1)  # Where condition
print(f"Where: {array_where}")          # Where: [-1 -1  3  4  5]

# Percentile function
array_percentile = np.percentile(array1, 50)  # 50th percentile (median)
print(f"Percentile: {array_percentile}")  # Percentile: 3.0

# Histogram
array_histogram, bins = np.histogram(array1, bins=5)  # Histogram
print(f"Histogram: {array_histogram}, bins: {bins}")  # Histogram:
[1 1 1 1 1], bins: [1. 2. 3. 4. 5. 6.]
```

Array functions

```python
# Correlation coefficient
array_corrcoef = np.corrcoef(array1, array2)  # Correlation coefficient
print(f"Correlation coefficient:\n{array_corrcoef}")
# Correlation coefficient: [[ 1. -1.] [-1.  1.]]

# Check for elements in array
array_isin = np.isin(array1, [1, 2, 3])  # Check if elements are in array
print(f"Isin: {array_isin}")             # Isin: [ True  True  True False False]

# Flip array
array_flipped = np.flip(array1)  # Flip array
print(f"Flipped: {array_flipped}")  # Flipped: [5 4 3 2 1]

# Put values in array
array_put = array1.copy()
np.put(array_put, [0, 2], [-1, -2])  # Put values at specified indices
print(f"Put: {array_put}")           # Put: [-1  2 -2  4  5]

# Delete elements
array_deleted = np.delete(array1, [0, 2])  # Delete elements at specified indices
print(f"Deleted: {array_deleted}")         # Deleted: [2 4 5]

# Clip values
array_clipped = np.clip(array1, 2, 4)  # Clip values to the interval [2, 4]
print(f"Clipped: {array_clipped}")     # Clipped: [2 2 3 4 4]
```

Broadcasting

The smaller array is broadcast across the larger array so that they become compatible for further operations.

```python
import numpy as np

# Creating sample arrays
array1 = np.array([1, 2, 3])
array2 = np.array([[1], [2], [3]])

# Broadcasting addition
result_add = array1 + array2
print(f"Broadcasting addition:\n{result_add}")
# Broadcasting addition:
# [[2 3 4]
#  [3 4 5]
#  [4 5 6]]

# Broadcasting multiplication
result_mul = array1 * array2
print(f"Broadcasting multiplication:\n{result_mul}")
# Broadcasting multiplication:
# [[1 2 3]
#  [2 4 6]
#  [3 6 9]]

# Broadcasting with scalar
scalar = 10
result_scalar = array1 + scalar
print(f"Broadcasting with scalar:\n{result_scalar}")
# Broadcasting with scalar:
# [11 12 13]
```

Wroking with missing values

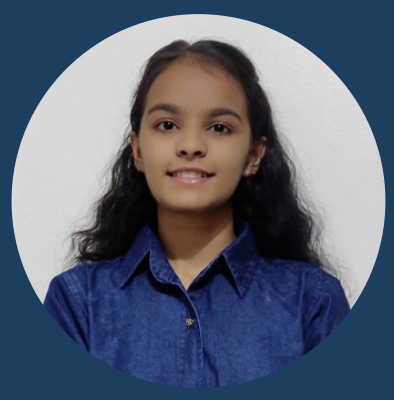
```python
import numpy as np

# Creating an array with missing values
array_with_nan = np.array([1, 2, np.nan, 4, np.nan, 6])

print(f"Original array:\n{array_with_nan}")
# Original array:
# [ 1.  2. nan  4. nan  6.]

# Checking for missing values
nan_mask = np.isnan(array_with_nan)
print(f"Missing values mask:\n{nan_mask}")
# Missing values mask:
# [False False  True False  True False]

# Counting missing values
num_missing = np.sum(nan_mask)
print(f"Number of missing values: {num_missing}")
# Number of missing values: 2

# Removing missing values
array_without_nan = array_with_nan[~nan_mask]
print(f"Array without missing values:\n{array_without_nan}")
# Array without missing values:
# [1. 2. 4. 6.]

# Replacing missing values with a specific value (e.g., 0)
array_filled_nan = np.where(nan_mask, 0, array_with_nan)
print(f"Array with missing values replaced by 0:\n{array_filled_nan}")
# Array with missing values replaced by 0:
# [1. 2. 0. 4. 0. 6.]

# Replacing missing values with the mean of the non-missing elements
mean_value = np.nanmean(array_with_nan)
array_filled_nan_mean = np.where(nan_mask, mean_value, array_with_nan)
print(f"Array with missing values replaced by the mean:\n{array_filled_nan_mean}")
# Array with missing values replaced by the mean:
# [1. 2. 3.25 4. 3.25 6.]
```

Wroking with random fun

```python
import numpy as np

# Set a seed for reproducibility
np.random.seed(42)


# Random array of specified shape with values uniformly distributed between 0 and 1
random_array_uniform = np.random.rand(3, 2)
print(f"Uniformly distributed random array:\n{random_array_uniform}")
# Uniformly distributed random array:
# [[0.37454012 0.95071431]
#  [0.73199394 0.59865848]
#  [0.15601864 0.15599452]]


# Random integers array with specified range and shape
random_array_integers = np.random.randint(1, 10, size=(3, 2))
print(f"Random integers array:\n{random_array_integers}")
# Random integers array:
# [[8 5]
#  [7 1]
#  [8 1]]


# Random choice from a given 1-D array
choices = np.random.choice([10, 20, 30, 40, 50], size=(3, 2))
print(f"Random choices array:\n{choices}")
# Random choices array:
# [[50 30]
#  [10 50]
#  [40 20]]


# Shuffle the array
array_to_shuffle = np.array([1, 2, 3, 4, 5])
np.random.shuffle(array_to_shuffle)
print(f"Shuffled array:\n{array_to_shuffle}")
# Shuffled array:
# [4 1 3 2 5]
```

🚀 Ready to Level Up?

CONNECT FOR MORE INSIGHTS AND FUTURE UPDATES!



Shruti Bharat

@shrutibharat0105