🔥 Python Lists vs NumPy Arrays

What is NumPy?

NumPy is a Python library used for fast mathematical calculations on large amounts of data.

The main thing NumPy provides is the ndarray, which is a fast, multi-dimensional array.

[ ]:

Why NumPy is different from normal Python lists

- ✔ 1. Fixed size
- ✔ 2. Same data type
- ✔ 3. Very fast operations

✔ 1. Fixed size

Once we create a NumPy array, its size cannot change. (But Python lists can grow or shrink.)

For example:

✔ Python List → Size CAN change

Python lists are flexible.

[9]:
```
lst=[1,2,3,4,5]
lst.append(2)   # adding an element at the end ---- now the lst become [1,2,3,4,5,2]
lst.pop(3)      # removing an element at index 3 ---It removes 4 and lst become [1,2,3,5,2]
lst.pop()       # removing an element at the last -- removes 2 ---lst [1,2,3,5]
print(lst)
```

```
[1, 2, 3, 5]
```

Python lists can change size anytime — we can add items or remove items whenever we want.

[ ]:

✔ 2. NumPy Array → Size CANNOT change

NumPy arrays have a fixed size.

If we try to change the size of a NumPy array, Python creates a new array instead of changing the old one.

❌ We CANNOT append directly

For example:

[11]:
```
import numpy as np
arr = np.array([1,2,3,4])
arr.append(5)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[11], line 3
      1 import numpy as np
      2 arr = np.array([1,2,3,4])
----> 3 arr.append(5)

AttributeError: 'numpy.ndarray' object has no attribute 'append'
```

[ ]:

✔ If we "append", NumPy secretly creates a new array

[17]:
```
import numpy as np
arr = np.array([1,2,3,4])
new_arr = np.append(arr,5)  # In this line NumPy internally makes extra space, copies all
                            # elements from the old array, adds the new element, and returns this new array.
```

[18]:
```
print(arr)
print(new_arr)
```

```
[1 2 3 4]
[1 2 3 4 5]
```

[ ]:

⭐ When we add an element to a NumPy array, NumPy creates a new array because its size cannot change.

⭐ NumPy internally makes extra space, copies all elements from the old array, adds the new element, and returns this new array. ¶

[ ]:

[ ]:

✔ 2. Same data type

NumPy arrays require all elements to be the same data type for speed and memory efficiency.

If mixed types are given, NumPy automatically converts them into one common type

[ ]:

For example:

Example 1: All integers

[30]:
```
arr=np.array([1,2,3,4,5])
print(arr)
print(arr.dtype)
```

```
[1 2 3 4 5]
int64
```

[ ]:

✔ Example 2: Mix int + float → NumPy converts (upcasts)

NumPy automatically converts all elements into one common type that can fit every value safely. This automatic conversion is called upcasting.

➡️ NumPy converts everything to float, because float can hold more information.

[35]:
```
arr1 = np.array([1,2,3.5,4])
print(arr1)
print(arr1.dtype)
```

```
[1.  2.  3.5 4. ]
float64
```

[ ]:

✔ Example 3: Mix int + string → all become strings

[42]:
```
arr2 = np.array([1,2,"Hello",4])
print(arr2)
print(arr2.dtype)
```

```
['1' '2' 'Hello' '4']
<U21
```

Numbers can be turned into strings.But strings cannot be turned into numbers

➡️ So NumPy converts everything to string

❌ Why Python lists are different

Python lists can store different types easily: ¶

[28]:
```
lst = [1, 2.5, "hello", True]
```

[29]:
```
type(lst)
```

[29]:
```
list
```

but it is slower and not memory-efficient.

[ ]:
```

```

✔ 3. ⭐ Why NumPy Operations Are Fast

NumPy performs mathematical operations on arrays much faster than Python lists because:

✔ 1. NumPy uses C code internally

When we write:

c = a * b

NumPy does the multiplication using compiled C loops, not slow Python loops.

C is much faster, so operations run at near-machine speed.

[ ]:
```

```

✔ 2. Data is stored in one continuous memory block

Because all elements have the same data type, NumPy stores them like this:

| 1 | 2 | 3 | 4 | 5 |

This makes it super easy for the CPU to read and process the data.

You may get doubt like why Python list can't do c = a * 2 the same way NumPy does

when we write:

[49]:
```
a = np.array([1,2,3,4])
c = a * 2
print(c)
```

```
[2 4 6 8]
```

✔ NumPy meaning:

Multiply every element by 2

→ [1, 2, 3] * 2 = [2, 4, 6]

(Real math multiplication)

[ ]:

❌ Python list meaning is different:

[50]:
```
a = [1, 2, 3]
c = a * 2
print(c)
```

```
[1, 2, 3, 1, 2, 3]
```

Python does repetition, not multiplication.

Because lists are not made for math.

[ ]:

⭐ Why is NumPy fast?

✔ Reason 1: No Python loops

Python loops are slow.

NumPy removes loops — it does the whole operation at once.

[ ]:

✔ Reason 2: NumPy uses C inside (very fast language)

When you write:

c = a * b

NumPy actually runs fast C code behind the scenes. So we get Python-like simplicity + C-like speed.

[ ]:

✔ Reason 3: Data stored together in memory

NumPy keeps numbers in one straight line in memory:

1 2 3 4 5 6

[ ]:

Python lists store references to objects, not raw values.

The actual data lives in different memory locations, making lists flexible but slow for numerical operations.

[ ]: