Fix: ‘numpy.ndarray’ object has no attribute ‘append’

Authored by

Mohammed looti

November 3, 2025

When performing data manipulation or scientific calculations in Python, developers heavily rely on the capabilities of the NumPy library. A common point of confusion, particularly for users accustomed to standard Python data structures, arises when attempting to extend a NumPy array. One error you may encounter is the following **AttributeError**:

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

This error occurs when you attempt to append one or more values to the end of a NumPy array by using the instance-based **append()** function designed for regular Python lists. Since the internal structure of a **numpy.ndarray** differs significantly from a list, it does not possess this method, thus triggering the error.

Understanding the Structural Mismatch

To correctly resolve this issue, it is essential to understand the fundamental difference between standard Python lists and NumPy arrays. A native Python list is dynamic; it is designed to be easily mutable and can grow or shrink in memory using methods like **append()**, which modify the list in place.

Conversely, a NumPy array is typically a fixed-size structure when created. This fixed-size nature allows NumPy to manage memory efficiently and perform vectorized operations rapidly, making it the backbone of high-performance numerical computing in Python. Because array memory allocation is static, the array object itself cannot simply be modified in place using a list-style **append()** method.

Therefore, when you attempt to call the method `x.append()` on a **numpy.ndarray** object `x`, the interpreter correctly reports that the array object has no such attribute, forcing us to use a dedicated NumPy function instead.

How to Reproduce the Attribute Error

We can easily illustrate this problem by defining a simple NumPy array and then attempting to use the standard Python **append()** function on it. This example clearly shows why the error is thrown when the wrong method is applied to the NumPy array structure:

```
import numpy as np

#define NumPy array
x = np.array()

#attempt to append the value '25' to end of NumPy array using the list method
```

x.append(25)

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

We receive the **AttributeError** because, as established, NumPy arrays are designed differently from Python lists, and the instance method **append()** simply does not exist for this object type.

The Functional Solution: Using np.append()

To fix this error, we must replace the instance method `x.append()` with the dedicated NumPy function: **np.append()**. This function takes the original array and the values to be added as its arguments.

A key concept to remember is that **np.append()** does not modify the existing array in memory. Instead, it creates a brand new array object that incorporates the old data and the new elements, and then returns this new object. This means the result must be explicitly assigned back to a variable (often overwriting the original variable) for the change to take effect.

The following code demonstrates the correct way to append a single value, '25', to the end of the existing array `x`:

import numpy as np

#define NumPy array
x = np.array()

#append the value '25' to end of NumPy array and reassign the new array to x
x = np.append(x, 25)

#view updated array
x

array()

By utilizing the **np.append()** function, we successfully created and assigned the extended array, resolving the **AttributeError**.

Alternative: Concatenating Multiple Arrays with np.concatenate()

While **np.append()** works for adding individual values or sequences, if your intent is to combine two or more existing arrays, the more explicit and often preferred function is **np.concatenate()**.

This function is explicitly designed for joining arrays along a specified axis and is generally more readable and efficient when combining large data structures.

The primary distinction is that **np.concatenate()** takes a sequence (a tuple or list) of the arrays you wish to join. For one-dimensional arrays, as shown below, it simply joins them end-to-end. For multi-dimensional arrays, you can specify the `axis` parameter to control the direction of the merge.

This example shows how to define two separate NumPy arrays, `a` and `b`, and combine them into a single resulting array `c` using **np.concatenate()**:

import numpy as np

#define two NumPy arrays
a = np.array()
b = np.array()

#concatenate two arrays together
c = np.concatenate((a, b))

#view resulting array
c

array()

The use of **np.concatenate()** is the recommended method for combining multiple arrays, emphasizing clarity and performance when dealing with large datasets.

Performance Warning: Avoiding Repeated Appending

Although **np.append()** provides the technical fix for the **AttributeError**, it is crucial to understand its performance implications. Because every call to **np.append()** results in the creation of a new array and the copying of all existing data, repeatedly calling this function within a loop is highly inefficient.

If you are iterating and adding many elements sequentially, the time complexity increases rapidly, leading to significant slowdowns. For optimal performance in iterative data collection, consider these superior alternatives:

Use Python Lists for Collection: Collect all elements using a standard Python list's **append()** method (which is efficient for lists), and then convert the final list to a NumPy array just once, outside the loop.
Pre-allocate Memory: If the final size of the array is known or can be estimated, pre-allocate a

large NumPy array (e.g., using `np.zeros()`) and fill the array slots directly.

Adopting these practices ensures that you leverage NumPy's speed benefits rather than hindering performance through constant reallocation.

Key Takeaways for Array Manipulation

The **AttributeError: 'numpy.ndarray' object has no attribute 'append'** is a classic indication of mixing Python list logic with NumPy array logic. Always remember that NumPy arrays are fixed-size objects requiring functional methods for modification.

To summarize the correct approach for extending NumPy arrays:

Use **np.append()** when adding single elements or small sequences, ensuring the result is reassigned to capture the new array.
Use **np.concatenate()** when combining two or more existing NumPy arrays.
Avoid iterative use of **np.append()** in favor of list collection or pre-allocation for performance-critical code.

Additional Resources

For further learning on efficient array manipulation and detailed documentation of the functions discussed, please refer to the following resources:

The official NumPy reference for np.append().
The official NumPy reference for np.concatenate().