Numpy Array Fonksiyonlar

Python'da mevcut olan bir fonksiyonu uygulamanın birkaç yolu vardır. En yaygın ve kullanışlı yöntemler şunlardır:

1. **Fonksiyonu Doğrudan Vektörel Olarak Uygulamak (Eğer Mümkünse):** Eğer fonksiyonunuz temel matematiksel işlemlerden veya zaten vektörel çalışabilen NumPy fonksiyonlarından (`np.sin` , `np.log` , `np.sqrt` vb.) oluşuyorsa, fonksiyonu doğrudan diziye uygulayabilirsiniz. NumPy bu işlemleri otomatik olarak her eleman için yapar. Bu genellikle **en hızlı ve en verimli** yöntemdir.Python

   ```
   import numpy as np

   def basit_islem(x):
    # Bu fonksiyonun içeriği NumPy'nin anladığı işlemlerden oluşuyor
    return x * 2 + 5

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

   # Fonksiyonu doğrudan çağırmak yerine, içindeki işlemi diziye uygulayın:
   sonuc = dizi * 2 + 5
   # Veya eğer fonksiyon sadece NumPy fonksiyonları kullanıyorsa:
   # sonuc = basit_islem(dizi) # Bu da çalışır çünkü işlem vektörel

   print(f"Orijinal Dizi: {dizi}")
   print(f"İşlem Sonucu: {sonuc}")
   ```

2. `np.vectorize()` **Kullanmak (Genel Amaçlı Fonksiyonlar İçin):** Eğer kendi yazdığınız ve sadece tek bir değere göre işlem yapan (skaler) bir Python fonksiyonunuz varsa (örneğin içinde `if/else` gibi doğrudan NumPy dizileriyle uyumlu olmayan yapılar varsa), bu fonksiyonu `np.vectorize()` ile "vektörel hale getirebilirsiniz". `np.vectorize()` , verdiğiniz fonksiyonu alıp, dizi üzerinde eleman eleman gezen ve her elemana fonksiyonu uygulayan yeni bir fonksiyon döndürür.Python

   Önemli Not: `np.vectorize()` temel olarak **kolaylık sağlamak** içindir. Arka planda genellikle bir döngü çalıştırdığı için, saf NumPy operasyonları (Yöntem 1) veya özel olarak optimize edilmiş C eklentileri kadar **hızlı olmayabilir**. Ancak karmaşık olmayan, standart Python fonksiyonlarını NumPy dizilerine uygulamak için çok pratiktir.

   ```
   import numpy as np

   # Sadece tek bir sayı alan standart bir Python fonksiyonu
   def karmasik_islem(x):
   ```

```
    if x % 2 == 0:
        return x / 2
    else:
        return x * 3 + 1

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

# Fonksiyonu vectorize et
vektorize_fonksiyon = np.vectorize(karmasik_islem)

# Vektorize edilmiş fonksiyonu diziye uygula
sonuc = vektorize_fonksiyon(dizi)

print(f"Orijinal Dizi: {dizi}")
print(f"np.vectorize ile Sonuç: {sonuc}")
# Çıktı: [ 4.  1. 10.  2. 16.  3.] (Sonuç float olabilir, isterseniz dtype belirtebilirsiniz)
```

3. **List Comprehension veya Döngü Kullanmak (Genellikle Önerilmez):** Python'un standart list comprehension yapısını veya `for` döngüsünü kullanarak dizi üzerinde gezebilir ve her elemana fonksiyonu uygulayabilirsiniz. Ancak bu, NumPy'nin optimize edilmiş C tabanlı operasyonlarından yararlanmaz ve genellikle **çok daha yavaştır**, bu yüzden büyük diziler için tavsiye edilmez.Python

```
import numpy as np

def benim_fonksiyonum(x):
    return x ** 2 + 1 # Örnek bir fonksiyon

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

# List comprehension ile
sonuc_liste = [benim_fonksiyonum(x) for x in dizi]
sonuc_dizi = np.array(sonuc_liste) # Tekrar NumPy dizisine çevirme

print(f"Orijinal Dizi: {dizi}")
print(f"List Comprehension ile Sonuç: {sonuc_dizi}")
```

Özet:

- Fonksiyonunuz basit NumPy işlemleriyle ifade edilebiliyorsa, **doğrudan dizi üzerinde işlem yapın (Yöntem 1)**. Bu en hızlısıdır.
- Tekil değerler üzerinde çalışan karmaşık bir Python fonksiyonunuz varsa ve kolayca uygulamak istiyorsanız, `np.vectorize()` **kullanın (Yöntem 2)**. Performans kritik değilse veya fonksiyon çok karmaşıksa iyi bir seçenektir.
- Performans çok önemliyse ve fonksiyon NumPy ile yazılamıyorsa, Numba veya Cython gibi kütüphanelerle fonksiyonu optimize etmeyi düşünebilirsiniz.

- **List comprehension veya Python döngülerinden (Yöntem 3)** genellikle NumPy dizileri için kaçının.