***توابع Numpy***

numpy.ufunc.\_\_call(\*args,\*\*kwarg) -1\_\_

یک روش در کلاس numpy.ufunc است که برای اجرای تابع ufunc بر روی آرایه‌ها استفاده می‌شود. این روش با استفاده از آرگومان‌های ورودی مشخص شده، تابع ufunc را اعمال کرده و نتیجه را برمی‌گرداند.

در زیر، ما توضیحات مربوط به numpy.ufunc.\_\_call\_\_ را برای شما ارائه می‌دهیم:

آرگومان‌ها: \*args, \*\*kwarg:

توضیح: این روش تابع ufunc را بر روی آرایه‌ها اعمال می‌کند. آرگومان‌های ورودی می‌توانند تعداد متغیری از آرایه‌ها باشند که برای اعمال تابع ufunc استفاده می‌شوند.

می‌توانید آرگومان‌های ورودی را با استفاده از \*args یا \*\*kwargs به عنوان آرایه‌های مستقل برای تابع ufunc ارسال کنید.

برای مثال، فرض کنیدمی توانیم از ufunc دریک عملیات جمع از دو آرایه است.

numpy.ufunc.\_\_call\_\_ برای اعمال این تابع روی دو آرایه استفاده کنیم. به صورت زیر:

import numpy as np

# تعریف تابع ufunc

ufunc = np.add

# اعمال تابع بر روی آرایه‌ها با استفاده از \_\_call\_\_

array1 = np.array([1, 2, 3])

array2 = np.array([4, 5, 6])

result = ufunc.\_\_call\_\_(array1, array2)

print(result) # خروجی] 9و7و5]

2-numpy.ufunc.accumulate( a, axis=None, dtype=None, out=None) یک روش در کلاس numpy.ufunc است که برای اعمال تابع ufunc به صورت تجمعی روی آرایه‌ها استفاده می‌شود. این روش مقادیر تجمعی از عملکرد تابع ufunc را بر روی آرایه ورودی ایجاد می‌کند.

در زیر، ما توضیحات مربوط به numpy.ufunc.accumulate را برای شما ارائه می‌دهیم:

آرگومان‌ها: a, axis=None, dtype=None, out=None

توضیح: این روش تابع ufunc را به صورت تجمعی روی آرایه a اعمال می‌کند و مقادیر تجمعی را برمی‌گرداند.

آرگومان a آرایه ورودی است که تابع ufunc روی آن اعمال می‌شود.

آرگومان axis مشخص می‌کند که تابع ufunc بر روی کدام محور (بعد) آرایه تجمعی را ایجاد می‌کند.

آرگومان dtype نوع داده خروجی را مشخص می‌کند.

آرگومان out آرایه خروجی است که اگر مشخص شود، نتایج در آن ذخیره می‌شود

نحوه استفاده:

import numpy as np

# تعریف تابع ufunc

ufunc = np.add

# آرایه ورودی

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

# اعمال تابع به صورت تجمعی

result = ufunc.accumulate(array)

print(result) # خروجی] 15و10و6و3و1]

`numpy.ufunc.at` -3یک روش در کلاس `numpy.ufunc` است که برای اعمال تابع `ufunc` به عناصر مشخص آرایه استفاده می‌شود. این روش امکان تغییر مقادیر عناصر مشخص آرایه را با استفاده از تابع `ufunc` می‌دهد.

در زیر، ما توضیحات مربوط به `numpy.ufunc.at` را برای شما ارائه می‌دهیم:

1. `numpy.ufunc.at`(: `a, indices, b=None):

- آرگومان‌ها: `a, indices, b=None`

- توضیح: این روش تابع `ufunc` را بر روی عناصر مشخص آرایه `a` اعمال می‌کند.

آرگومان `a` آرایه ورودی است که تابع `ufunc` روی آن اعمال می‌شود.

آرگومان `indices` شاخص‌های عناصری است که تابع `ufunc` روی آنها اعمال می‌شود.

آرگومان `b` نیز می‌تواند یک آرایه ورودی دیگر باشد که برای عملیات `ufunc` استفاده می‌شود.

برای مثال، فرض کنید می‌خواهیم تابع `ufunc` جمع را بر روی عناصر مشخص آرایه ورودی اعمال کنیم. می‌توانیم از `numpy.ufunc.at` استفاده کنیم. به صورت زیر:

``` python

import numpy as np

# تعریف تابع ufunc

ufunc = np.add

# آرایه ورودی

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

# عناصر مشخص برای اعمال تابع

indices = [0, 2, 4]

# اعمال تابع بر روی عناصر مشخص

ufunc.at(array, indices, 10)

print(array) # خروجی: [11 2 13 4 15]

```

در این مثال، ما `ufunc` را با استفاده از تابع `np.add` تعریف کرده‌ایم. سپس با استفاده از `at` تابع `ufunc` را بر روی عناصر مشخص با شاخص‌های `[0, 2, 4]` در آرایه `array` اعمال می‌کنیم.

-`numpy.ufunc.outer`(`a, b`) -4یک روش در کلاس `numpy.ufunc` است که برای ایجاد آرایه‌ای جدید از نتایج اعمال تابع `ufunc` بر روی تمام جفت‌های عناصر دو آرایه ورودی استفاده می‌شود.

در زیر، ما توضیحات مربوط به `numpy.ufunc.outer` را برای شما ارائه می‌دهیم:

- آرگومان‌ها: `a, b`

- توضیح: این روش تابع `ufunc` را بر روی تمام جفت‌های عناصر دو آرایه ورودی `a` و `b` اعمال می‌کند و آرایه‌ی جدید حاصل از این عمل را برمی‌گرداند. طول آرایه خروجی برابر با ضرب طول آرایه `a` در طول آرایه `b` است.

برای مثال، فرض کنید می‌خواهیم تابع `ufunc` جمع را بر روی تمام جفت‌های عناصر دو آرایه ورودی اعمال کنیم و ماتریس خارجی (outer product) را بسازیم. می‌توانیم از `numpy.ufunc.outer` استفاده کنیم. به صورت زیر:

``` python

import numpy as np

# تعریف تابع ufunc

ufunc = np.add

# آرایه‌های ورودی

array1 = np.array([1, 2, 3])

array2 = np.array([4, 5, 6])

# اعمال تابع بر روی تمام جفت‌های عناصر

result = ufunc.outer(array1, array2)

print(result)

'''

خروجی:

[[ 5 6 7]

[ 6 7 8]

[ 7 8 9]]

'''

```

`numpy.ufunc.reduce`-5 یک روش در کلاس `numpy.ufunc` است که برای کاهش (reduce) آرایه ورودی به یک مقدار با استفاده از تابع `ufunc` استفاده می‌شود. این روش به صورت تکراری تابع `ufunc` را بر روی عناصر آرایه اعمال کرده و مقدار نهایی را برمی‌گرداند.

در زیر، ما توضیحات مربوط به `numpy.ufunc.reduce` را برای شما ارائه می‌دهیم:

1. `numpy.ufunc.reduce(`a, axis=None, dtype=None, out=None, keepdims=False`)`:

- آرگومان‌ها: `a, axis=None, dtype=None, out=None, keepdims=False`

- توضیح: این روش تابع `ufunc` را به صورت تکراری بر روی آرایه `a` اعمال کرده و تابع را به صورت نزولی بر روی آرایه ورودی اعمال میکند.

آرگومان `a` آرایه ورودی است که تابع `ufunc` روی آن اعمال می‌شود.

آرگومان `axis` مشخص می‌کند که تابع `ufunc` را بر روی کدام محور (بعد) آرایه اعمال کند. آرگومان `dtype` نوع داده خروجی را مشخص می‌کند.

آرگومان `out` آرایه خروجی است که اگر مشخص شود، نتیجه در آن ذخیره می‌شود. آرگومان `keepdims` مشخص می‌کند که ابعاد آرایه خروجی حفظ شود یا خیر.

برای مثال، فرض کنید می‌خواهیم تابع `ufunc` جمع را به صورت کاهشی (reduce) بر روی آرایه ورودی اعمال کنیم. می‌توانیم از `numpy.ufunc.reduce` استفاده کنیم. به صورت زیر:

``` python

import numpy as np

# تعریف تابع ufunc

ufunc = np.add

# آرایه ورودی

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

# اعمال تابع به صورت کاهشی

result = ufunc.reduce(array)

print(result) # خروجی: 15

```

`. `numpy.ufunc.reduceat(: `a, indices, axis=None, dtype=None, out=None):

-6. `numpy.ufunc.reduceat(: `a, indices, axis=None, dtype=None, out=None):.

یک روش در کلاس `numpy.ufunc` است که به صورت کاهشی با استفاده از تابع `ufunc` بر روی آرایه ورودی اعمال می شود.

در زیر، ما توضیحات مربوط به `numpy.ufunc.reduceat` را برای شما ارائه می‌دهیم:

- آرگومان‌ها: `a, indices, axis=None, dtype=None, out=None`

- توضیح: این روش تابع `ufunc` را به صورت تکراری بر روی عناصر مشخص شده در آرایه `a` اعمال کرده و تابع مورد نظر را به صورت نزولی بر روی آرایه اعمال می کند..

آرگومان `a` آرایه ورودی است که تابع `ufunc` روی آن اعمال می‌شود.

آرگومان `indices` آرایه ای از اندیس های عناصر مورد نظر برای اعمال تابع `ufunc` است.

آرگومان `axis` مشخص می‌کند که تابع `ufunc` را بر روی کدام محور (بعد) آرایه اعمال کند.

آرگومان `dtype` نوع داده خروجی را مشخص می‌کند. آرگومان `out` آرایه خروجی است که اگر مشخص شود، نتیجه در آن ذخیره می‌شود.

``` python

import numpy as np

# تعریف تابع ufunc

ufunc = np.add

# آرایه ورودی

array = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])

`numpy.ufunc.resolve\_dtypes()` -7یک روش در کلاس `numpy.ufunc` است که برای تعیین نوع داده خروجی تابع `ufunc` بر اساس نوع داده عناصر ورودی استفاده می‌شود. این روش، نوع داده خروجی را با توجه به نوع داده‌های ورودی و قوانین تعیین شده توسط `ufunc` مشخص می‌کند.

در زیر، ما توضیحات مربوط به `numpy.ufunc.resolve\_dtypes()` را برای شما ارائه می‌دهیم:

1. `numpy.ufunc.resolve\_dtypes`:

- آرگومان‌ها: ندارد.

- توضیح: این روش نوع داده خروجی تابع `ufunc` را بر اساس نوع داده‌های ورودی و قوانین تعیین شده توسط `ufunc` مشخص می‌کند. عموماً از این روش استفاده نمی‌شود، مگر اینکه بخواهید به صورت دستی نوع داده خروجی را تعیین کنید.

به عنوان مثال، در زیر یک کد ساده برای استفاده از `numpy.ufunc.resolve\_dtypes` آورده شده است:

``` python

import numpy as np

# تعریف تابع ufunc

ufunc = np.add

# نوع داده عناصر ورودی

input\_dtypes = [np.int32, np.float64]

# تعیین نوع داده خروجی با استفاده از resolve\_dtypes

output\_dtype = ufunc.resolve\_dtypes(input\_dtypes)

print(output\_dtype) # خروجی: float64

```

در این مثال، ما `ufunc` را با استفاده از تابع `np.add` تعریف می‌کنیم. سپس `input\_dtypes` را به عنوان لیستی از نوع داده‌های عناصر ورودی تعریف می‌کنیم. سپس با استفاده از `ufunc.resolve\_dtypes`، نوع داده خروجی تابع `ufunc` بر اساس نوع داده‌های ورودی تعیین می‌شود و در متغیر `output\_dtype` ذخیره می‌شود. در نهایت، نوع داده خروجی، در این مثال `float64`، نمایش داده می‌شود.

-8متد `numpy.ufunc.resolve\_dtypes(dtypes,signature,casting,reduction)` از NumPy 1.21.0 به بعد در دسترس است. این متد برای تعیین نوع داده خروجی تابع `ufunc` بر اساس نوع داده‌های ورودی و پارامترهای دیگر استفاده می‌شود.

توضیحات آرگومان‌ها:

- `dtypes`: لیستی از نوع داده‌های عناصر ورودی به تابع `ufunc`.

- `signature` (اختیاری): نوع داده‌های مورد انتظار برای هر عملیات در ورودی و خروجی. این پارامتر برای تعیین نوع داده خروجی در حالتی که تابع `ufunc` به صورت عمومی تعریف شده است و برای انواع داده‌های مختلف نتایج متفاوتی برمی‌گرداند، استفاده می‌شود.

- `casting` (اختیاری): حالت تبدیل داده (casting) بین نوع داده‌های ورودی و خروجی. مقادیر ممکن برای این پارامتر شامل `'no'` (عدم تبدیل)، `'equiv'` (تبدیل معادل)، `'safe'` (تبدیل ایمن) و `'same\_kind'` (تبدیل با همان نوع) هستند.

- `reduction` (اختیاری): اگر `True` باشد، نوع داده خروجی برای عملیات کاهش (reduction) تابع `ufunc` را مشخص می‌کند.

در صورت استفاده از این متد، نوع داده خروجی تابع `ufunc` بر اساس نوع داده‌های ورودی و پارامترهای دیگر تعیین می‌شود.

```python

import numpy as np

# تعریف تابع ufunc

ufunc = np.add

# نوع داده‌های عناصر ورودی

input\_dtypes = [np.int32, np.float64]

# تعیین نوع داده خروجی با استفاده از resolve\_dtypes

output\_dtype = ufunc.resolve\_dtypes(input\_dtypes, reduction=True)

print(output\_dtype) # خروجی: float64

```

در این مثال، ما `ufunc` را با استفاده از تابع `np.add` تعریف می‌کنیم. سپس `input\_dtypes` را به عنوان لیستی از نوع داده‌های عناصر ورودی مشخص می‌کنیم. سپس با استفاده از `ufunc.resolve\_dtypes` و با تنظیم پارامتر `reduction=True`، نوع داده خروجی تابع `ufunc` بر اساس نوع داده‌های ورودی تعیین می‌شود و در متغیر `output\_dtype` ذخیره می‌شود. در این مثال، نوع داده خروجی به عنوان `float64` تعیین می‌شود.

لطفاً توجه داشته باشید که در این مثال، من فرض کرده‌ام که تابع `ufunc` یک عملیات کاهش (reduction) است. اگر تابع `ufunc` شامل عملیات دیگری باشد، لازم است پارامتر `reduction` را در تابع `ufunc.resolve\_dtypes` متناسب با نوع عملیات مورد استفاده تنظیم کنید. همچنین، بررسی کنید که آرگومان‌ها و تنظیمات دیگر متناسب با نیازهای شما هستند.

-9تابع `numpy.ufunc.nin` از کلاس `numpy.ufunc` است. این تابع تعداد آرگومان‌های مورد نیازدر ورودی برای فراخوانی تابع `ufunc` را برمی‌گرداند.

در زیر، توضیحات مربوط به تابع `numpy.ufunc.nin` را برای شما آورده‌ام:

- `numpy.ufunc.nin()`:

- آرگومان‌ها: ندارد.

- توضیح: این تابع تعداد آرگومان‌های مورد نیاز برای فراخوانی تابع `ufunc` را برمی‌گرداند.

در زیر مثالی از استفاده از تابع `numpy.ufunc.nin` را برای تابع `numpy.add` ارائه داده‌ام:

```python

import numpy as np

# تابع ufunc

ufunc = np.add

# تعداد آرگومان‌های مورد نیاز برای تابع ufunc

num\_args = ufunc.nin

print(num\_args) # خروجی: 2

```

در این مثال، ما `numpy.add` را به عنوان تابع `ufunc` استفاده می‌کنیم. با استفاده از تابع `numpy.ufunc.nin` می‌توانیم تعداد آرگومان‌های مورد نیاز برای فراخوانی تابع `numpy.add` را دریافت کنیم، که در این حالت برابر با 2 است.

-10تابع `numpy.ufunc.nou()` از کلاس `numpy.ufunc` است. این تابع تعداد آرگومان‌های خروجی تابع `ufunc` را برمی‌گرداند.

در زیر، توضیحات مربوط به تابع `numpy.ufunc.nout` را برای شما آورده‌ام:

- `numpy.ufunc.nout`:

- آرگومان‌ها: ندارد.

- توضیح: این تابع تعداد آرگومان‌های خروجی مورد نیاز برای فراخوانی تابع `ufunc` را برمی‌گرداند.

در زیر مثالی از استفاده از تابع `numpy.ufunc.nout` را برای تابع `numpy.add` ارائه داده‌ام:

```python

import numpy as np

# تابع ufunc

ufunc = np.add

# تعداد آرگومان‌های خروجی مورد نیاز برای تابع ufunc

num\_outputs = ufunc.nout

print(num\_outputs) # خروجی: 1

```

در این مثال، ما `numpy.add` را به عنوان تابع `ufunc` استفاده می‌کنیم. با استفاده از تابع `numpy.ufunc.nout` می‌توانیم تعداد آرگومان‌های خروجی مورد نیاز برای فراخوانی تابع `numpy.add` را دریافت کنیم، که در این حالت برابر با 1 است.

-11تابع `numpy.ufunc.nargs()` از کلاس `numpy.ufunc` است. این تابع تعداد کل آرگومان‌های مورد نیاز (ورودی و خروجی) برای فراخوانی تابع `ufunc` را برمی‌گرداند.

در زیر، توضیحات مربوط به تابع `numpy.ufunc.nargs` را برای شما آورده‌ام:

- `numpy.ufunc.nargs`:

- آرگومان‌ها: ندارد.

- توضیح: این تابع تعداد کل آرگومان‌های مورد نیاز (ورودی و خروجی) برای فراخوانی تابع `ufunc` را برمی‌گرداند.

در زیر مثالی از استفاده از تابع `numpy.ufunc.nargs` را برای تابع `numpy.add` ارائه داده‌ام:

```python

import numpy as np

# تابع ufunc

ufunc = np.add

# تعداد کل آرگومان‌های مورد نیاز برای تابع ufunc

num\_args = ufunc.nargs

print(num\_args) # خروجی: 3

```

در این مثال، ما `numpy.add` را به عنوان تابع `ufunc` استفاده می‌کنیم. با استفاده از تابع `numpy.ufunc.nargs` می‌توانیم تعداد کل آرگومان‌های مورد نیاز (ورودی و خروجی) برای فراخوانی تابع `numpy.add` را دریافت کنیم، که در این حالت برابر با 3 است.

-12تابع `numpy.ufunc.ntypes()` از کلاس `numpy.ufunc` است. این تابع تعداد کل نوع‌های مختلف ممکن برای تابع `ufunc` را برمی‌گرداند.

در زیر، توضیحات مربوط به تابع `numpy.ufunc.ntypes` را برای شما آورده‌ام:

- `numpy.ufunc.ntypes`:

- آرگومان‌ها: ندارد.

- توضیح: این تابع تعداد کل نوع‌های مختلف ممکن برای تابع `ufunc` را برمی‌گرداند.

در زیر مثالی از استفاده از تابع `numpy.ufunc.ntypes` را برای تابع `numpy.add` ارائه داده‌ام:

```python

import numpy as np

# تابع ufunc

ufunc = np.add

# تعداد کل نوع‌های ممکن برای تابع ufunc

num\_types = ufunc.ntypes

print(num\_types) # خروجی: 21

```

در این مثال، ما `numpy.add` را به عنوان تابع `ufunc` استفاده می‌کنیم. با استفاده از تابع `numpy.ufunc.ntypes` می‌توانیم تعداد کل نوع‌های مختلف ممکن برای تابع `numpy.add` را دریافت کنیم، که در این حالت برابر با 21 است.

`numpy.ufunc.identity()`-13 یک ویژگی (attribute) از کلاس `numpy.ufunc` است. این ویژگی مقدار هویت عملگر (identity value) مرتبط با تابع `ufunc` را برمی‌گرداند.

در زیر، توضیحات مربوط به ویژگی `numpy.ufunc.identity` را برای شما آورده‌ام:

- `numpy.ufunc.identity`:

- آرگومان‌ها: ندارد.

- توضیح: این ویژگی مقدار هویت عملگر (identity value) مرتبط با تابع `ufunc` را برمی‌گرداند.

مقدار هویت عملگر بستگی به تابع `ufunc` دارد. به عنوان مثال، برای تابع `add`، مقدار هویت عملگر صفر است، و برای تابع `multiply`، مقدار هویت عملگر یک است.

در زیر مثالی از استفاده از ویژگی `numpy.ufunc.identity` را برای تابع `numpy.add` ارائه داده‌ام:

```python

import numpy as np

# تابع ufunc

ufunc = np.add

# مقدار هویت عملگر مرتبط با تابع ufunc

identity\_value = ufunc.identity

print(identity\_value) # خروجی: 0

```

در این مثال، ما `numpy.add` را به عنوان تابع `ufunc` استفاده می‌کنیم. با استفاده از ویژگی `numpy.ufunc.identity` می‌توانیم مقدار هویت عملگر مرتبط با تابع `numpy.add` را دریافت کنیم، که در این حالت برابر با 0 است.

`numpy.ufunc.signature()` -14یک ویژگی (attribute) از کلاس `numpy.ufunc` است. این ویژگی امضای (signature) تابع `ufunc` را برمی‌گرداند.

امضا تابع (`signature`) شامل نوع و ترتیب آرگومان‌های ورودی و خروجی تابع است. این امضا می‌تواند به شکل یک رشته (string) نمایش داده شود.

در زیر، مثالی از استفاده از ویژگی `numpy.ufunc.signature` را برای تابع `numpy.add` ارائه داده‌ام:

```python

import numpy as np

# تابع ufunc

ufunc = np.add

# امضای مرتبط با تابع ufunc

signature = ufunc.signature

print(signature) # خروجی: '(i)->()'

```

در این مثال، ما `numpy.add` را به عنوان تابع `ufunc` استفاده می‌کنیم. با استفاده از ویژگی `numpy.ufunc.signature` می‌توانیم امضای مرتبط با تابع `numpy.add` را دریافت کنیم، که در این حالت برابر با `'(i)->()'` است. این امضا بیانگر یک آرگومان ورودی به عنوان عدد صحیح (`i`) و خروجی بدون آرگومان است.

`numpy.ufunc.reduce(array,axis=None,dtype=None,out=None,keepdims)`-15 یک متد از کلاس `numpy.ufunc` است که برای کاهش (reduce) آرایه به یک عنصر استفاده می‌شود.

در زیر، توضیحات مربوط به متد `numpy.ufunc.reduce` را برای شما آورده‌ام:

- `numpy.ufunc.reduce`:

- آرگومان‌ها:

- `array`: آرایه ورودی که باید کاهش یابد.

- `axis=None`: محور یا محورهایی که بر روی آنها عمل کاهش انجام می‌شود. در صورت عدم تعیین، آرایه به صورت یک عدد کاهش یافته است.

- `dtype=None`: نوع داده مورد نظر برای خروجی کاهش یافته. در صورت عدم تعیین، نوع داده آرایه ورودی است.

- `out=None`: آرایه خروجی کاهش یافته. اگر مقدار دهی نشود، آرایه جدیدی ایجاد می‌شود.

- `keepdims`: آیا خروجی نگه داشته شود یا خیر؟

در زیر مثالی از استفاده از متد `numpy.ufunc.reduce` را برای تابع `numpy.add` ارائه داده‌ام:

```python

import numpy as np

# تابع ufunc

ufunc = np.add

# آرایه ورودی

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

# کاهش آرایه با تابع ufunc

result = ufunc.reduce(arr)

print(result) # خروجی: 15

```

در این مثال، ما `numpy.add` را به عنوان تابع `ufunc` استفاده می‌کنیم. با استفاده از متد `numpy.ufunc.reduce` و اعمال آن روی آرایه `arr`، آرایه کاهش یافته به عنوان خروجی بدست می‌آید، که در این حالت برابر با 15 است.

`numpy.empty(shape, dtype=float, order='C')`:-16یک تابع از ماژول `numpy` است که یک آرایه خالی با ابعاد مشخص می‌سازد، اما مقادیر آن را تعیین نمی‌کند.

توضیحات مربوط به تابع `numpy.empty` را برای شما آورده‌ام:

- `numpy.empty(shape, dtype=float, order='C')`:

- آرگومان‌ها:

- `shape`: ابعاد آرایه خروجی.

- `dtype`: نوع داده آرایه خروجی. (اختیاری، پیش‌فرض: float)

- `order`: الگوی مرتب‌سازی داده‌های آرایه در حافظه. (اختیاری، پیش‌فرض: 'C')

- خروجی: آرایه خالی با ابعاد مشخص.

در زیر مثالی از استفاده از تابع `numpy.empty` را برای ایجاد یک آرایه خالی ارائه داده‌ام:

```python

import numpy as np

# ایجاد آرایه خالی

arr = np.empty((3, 2))

print(arr)

```

در این مثال، با استفاده از تابع `numpy.empty` یک آرایه خالی با ابعاد `(3, 2)` ایجاد می‌شود و در متغیر `arr` ذخیره می‌شود. توجه داشته باشید که مقادیر آرایه در این حالت تعیین نشده‌اند و ممکن است با مقادیر ناخواسته پر شده باشند.

`numpy.empty\_like` -17یک تابع از ماژول `numpy` است که یک آرایه خالی با همان ابعاد و نوع داده یک آرایه موجود ایجاد می‌کند.

توضیحات مربوط به تابع `numpy.empty\_like` را برای شما آورده‌ام:

- `numpy.empty\_like(a, dtype=None, order='K', subok=True)`:

- آرگومان‌ها:

- `a`: آرایه ورودی، بر اساس ابعاد و نوع داده آن، آرایه خروجی ایجاد می‌شود.

- `dtype`: نوع داده آرایه خروجی. (اختیاری، پیش‌فرض: None)

- `order`: الگوی مرتب‌سازی داده‌های آرایه در حافظه. (اختیاری، پیش‌فرض: 'K')

- `subok`: در صورت True بودن، نوع آرایه خروجی همان نوعی است که `a` آن است. (اختیاری، پیش‌فرض: True)

- خروجی: آرایه خالی با همان ابعاد و نوع داده آرایه `a`.

در زیر مثالی از استفاده از تابع `numpy.empty\_like` را برای ایجاد یک آرایه خالی با همان ابعاد و نوع داده یک آرایه موجود ارائه داده‌ام:

```python

import numpy as np

# آرایه ورودی

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

# ایجاد آرایه خالی با همان ابعاد و نوع داده آرایه ورودی

empty\_arr = np.empty\_like(arr)

print(empty\_arr)

```

در این مثال، ما یک آرایه `arr` با ابعاد `(2, 3)` ایجاد کرده‌ایم. با استفاده از تابع `numpy.empty\_like` می‌توانیم یک آرایه خالی با همان ابعاد و نوع داده آرایه `arr` ایجاد کنیم و در متغیر `empty\_arr` ذخیره کنیم. توجه داشته باشید که مقادیر آرایه در این حالت تعیین نشده‌اند و ممکن است با مقادیر ناخواسته پر شده باشند.

`numpy.eye` -18یک تابع از ماژول `numpy` است که یک ماتریس واحد (identity matrix) را ایجاد می‌کند.

توضیحات مربوط به تابع `numpy.eye` را برای شما آورده‌ام:

- `numpy.eye(N, M=None, k=0, dtype=float, order='C')`:

- آرگومان‌ها:

- `N`: تعداد ردیف‌ها در ماتریس واحد.

- `M`: تعداد ستون‌ها در ماتریس واحد. اگر None باشد، مقدار آن برابر با `N` خواهد بود. (اختیاری)

- `k`: انتقال (shift) قطری ماتریس واحد. عدد صفر به معنای قطر اصلی است. عدد‌های مثبت به معنای ستون‌های بالای قطر و عدد‌های منفی به معنای ستون‌های پایین قطر هستند. (اختیاری، پیش‌فرض: 0)

- `dtype`: نوع داده ماتریس واحد. (اختیاری، پیش‌فرض: float)

- `order`: الگوی مرتب‌سازی داده‌های ماتریس در حافظه. (اختیاری، پیش‌فرض: 'C')

- خروجی: ماتریس واحد.

در زیر مثالی از استفاده از تابع `numpy.eye` را برای ایجاد یک ماتریس واحد ارائه داده‌ام:

```python

import numpy as np

# ایجاد ماتریس واحد 3x3

identity\_matrix = np.eye(3)

print(identity\_matrix)

```

در این مثال، با استفاده از تابع `numpy.eye` یک ماتریس واحد با ابعاد `3x3` ایجاد می‌شود و در متغیر `identity\_matrix` ذخیره می‌شود. ماتریس واحد در این حالت برابر است با:

```

[[1. 0. 0.]

[0. 1. 0.]

[0. 0. 1.]]

```

.

`numpy.ones` -19یک تابع از ماژول `numpy` است که یک آرایه با مقدار 1 به ابعاد و نوع داده مشخص ایجاد می‌کند.

توضیحات مربوط به تابع `numpy.ones` را برای شما آورده‌ام:

- `numpy.ones(shape, dtype=None, order='C')`:

- آرگومان‌ها:

- `shape`: ابعاد آرایه مورد نظر.

- `dtype`: نوع داده آرایه. (اختیاری، پیش‌فرض: None)

- `order`: الگوی مرتب‌سازی داده‌های آرایه در حافظه. (اختیاری، پیش‌فرض: 'C')

- خروجی: آرایه با مقدار 1 و ابعاد و نوع داده مشخص.

در زیر مثالی از استفاده از تابع `numpy.ones` را برای ایجاد یک آرایه با مقدار 1 ارائه داده‌ام:

```python

import numpy as np

# ایجاد آرایه 2x3 با مقدار 1

ones\_array = np.ones((2, 3))

print(ones\_array)

```

در این مثال، با استفاده از تابع `numpy.ones` یک آرایه به ابعاد `2x3` ایجاد می‌شود و در متغیر `ones\_array` ذخیره می‌شود. مقادیر تمامی عناصر این آرایه برابر با 1 است و خروجی به صورت زیر خواهد بود:

```

[[1. 1. 1.]

[1. 1. 1.]]

```

مقادیر در آرایه ایجاد شده با استفاده از تابع `numpy.ones` به صورت پیش‌فرض به نوع داده `float` تبدیل می‌شوند. اما با استفاده از پارامتر `dtype` می‌توانید نوع داده دلخواه خود را مشخص کنید.

`numpy.ones\_like` -20یک تابع از ماژول `numpy` است که یک آرایه جدید با مقدار 1 و با همان ابعاد و نوع داده یک آرایه موجود را ایجاد می‌کند.

توضیحات مربوط به تابع `numpy.ones\_like` را برای شما آورده‌ام:

- `numpy.ones\_like(a, dtype=None, order='K', subok=True, shape=None)`:

- آرگومان‌ها:

- `a`: آرایه ورودی به عنوان الگو برای ایجاد آرایه جدید.

- `dtype`: نوع داده آرایه جدید. (اختیاری، پیش‌فرض: None)

- `order`: الگوی مرتب‌سازی داده‌های آرایه جدید در حافظه. (اختیاری، پیش‌فرض: 'K')

- `subok`: مشخص می‌کند آیا آرایه جدید باید یک زیرکلاس از آرایه ورودی باشد یا نه. (اختیاری، پیش‌فرض: True)

- `shape`: ابعاد آرایه جدید. اگر این پارامتر مشخص نشود، ابعاد آرایه ورودی برداشت می‌شود. (اختیاری)

- خروجی: آرایه جدید با مقدار 1 و با همان ابعاد و نوع داده آرایه ورودی.

در زیر مثالی از استفاده از تابع `numpy.ones\_like` را برای ایجاد یک آرایه جدید با مقدار 1 و با همان ابعاد و نوع داده یک آرایه ورودی ارائه داده‌ام:

```python

import numpy as np

# آرایه ورودی

arr = np.array([[2, 3, 4], [5, 6, 7]])

# ایجاد آرایه جدید با مقدار 1 و با همان ابعاد و نوع داده آرایه ورودی

ones\_array = np.ones\_like(arr)

print(ones\_array)

```

`numpy.zeros` -21یک تابع از ماژول `numpy` است که یک آرایه با مقدار 0 به ابعاد و نوع داده مشخص ایجاد می‌کند.

توضیحات مربوط به تابع `numpy.zeros` را برای شما آورده‌ام:

- `numpy.zeros(shape, dtype=None, order='C')`:

- آرگومان‌ها:

- `shape`: ابعاد آرایه مورد نظر.

- `dtype`: نوع داده آرایه. (اختیاری، پیش‌فرض: None)

- `order`: الگوی مرتب‌سازی داده‌های آرایه در حافظه. (اختیاری، پیش‌فرض: 'C')

- خروجی: آرایه با مقدار 0 و ابعاد و نوع داده مشخص.

در زیر مثالی از استفاده از تابع `numpy.zeros` را برای ایجاد یک آرایه با مقدار 0 ارائه داده‌ام:

```python

import numpy as np

# ایجاد آرایه 2x3 با مقدار 0

zeros\_array = np.zeros((2, 3))

print(zeros\_array)

```

در این مثال، با استفاده از تابع `numpy.zeros` یک آرایه به ابعاد `2x3` ایجاد می‌شود و در متغیر `zeros\_array` ذخیره می‌شود. مقادیر تمامی عناصر این آرایه برابر با 0 است و خروجی به صورت زیر خواهد بود:

```

[[0. 0. 0.]

[0. 0. 0.]]

```

مقادیر در آرایه ایجاد شده با استفاده از تابع `numpy.zeros` به صورت پیش‌فرض به نوع داده `float` تبدیل می‌شوند. اما با استفاده از پارامتر `dtype` می‌توانید نوع داده دلخواه خود را مشخص کنید.

`numpy.zeros\_like` -22یک تابع از ماژول `numpy` است که یک آرایه جدید با مقدار 0 و با همان ابعاد و نوع داده یک آرایه ورودی را ایجاد می‌کند.

توضیحات مربوط به تابع `numpy.zeros\_like` را برای شما آورده‌ام:

- `numpy.zeros\_like(a, dtype=None, order='K', subok=True, shape=None)`:

- آرگومان‌ها:

- `a`: آرایه ورودی به عنوان الگو برای ایجاد آرایه جدید.

- `dtype`: نوع داده آرایه جدید. (اختیاری، پیش‌فرض: None)

- `order`: الگوی مرتب‌سازی داده‌های آرایه جدید در حافظه. (اختیاری، پیش‌فرض: 'K')

- `subok`: مشخص می‌کند آیا آرایه جدید باید یک زیرکلاس از آرایه ورودی باشد یا نه. (اختیاری، پیش‌فرض: True)

- `shape`: ابعاد آرایه جدید. اگر این پارامتر مشخص نشود، ابعاد آرایه ورودی برداشت می‌شود. (اختیاری)

- خروجی: آرایه جدید با مقدار 0 و با همان ابعاد و نوع داده آرایه ورودی.

در زیر مثالی از استفاده از تابع `numpy.zeros\_like` را برای ایجاد یک آرایه جدید با مقدار 0 و با همان ابعاد و نوع داده یک آرایه ورودی ارائه داده‌ام:

```python

import numpy as np

# آرایه ورودی

arr = np.array([[2, 3, 4], [5, 6, 7]])

# ایجاد آرایه جدید با مقدار 0 و با همان ابعاد و نوع داده آرایه ورودی

zeros\_array = np.zeros\_like(arr)

print(zeros\_array)

```

`numpy.full` -23یک تابع از ماژول `numpy` است که یک آرایه با مقدار تکراری و به ابعاد و نوع داده مشخص ایجاد می‌کند.

توضیحات مربوط به تابع `numpy.full` را برای شما آورده‌ام:

- `numpy.full(shape, fill\_value, dtype=None, order='C')`:

- آرگومان‌ها:

- `shape`: ابعاد آرایه مورد نظر.

- `fill\_value`: مقدار تکراری برای پر کردن آرایه.

- `dtype`: نوع داده آرایه. (اختیاری، پیش‌فرض: None)

- `order`: الگوی مرتب‌سازی داده‌های آرایه در حافظه. (اختیاری، پیش‌فرض: 'C')

- خروجی: آرایه با مقادیر تکراری و ابعاد و نوع داده مشخص.

در زیر مثالی از استفاده از تابع `numpy.full` را برای ایجاد یک آرایه با مقادیر تکراری ارائه داده‌ام:

```python

import numpy as np

# ایجاد آرایه 2x3 با مقدار تکراری 5

full\_array = np.full((2, 3), 5)

print(full\_array)

```

در این مثال، با استفاده از تابع `numpy.full` یک آرایه به ابعاد `2x3` ایجاد می‌شود و مقدار تکراری 5 در تمامی عناصر آرایه قرار می‌گیرد. خروجی به صورت زیر خواهد بود:

```

[[5 5 5]

[5 5 5]]

```

مقادیر در آرایه ایجاد شده با استفاده از تابع `numpy.full` به نوع داده `fill\_value` تبدیل می‌شوند. نوع داده پیش‌فرض برای `fill\_value` و `dtype` آرایه، نوع داده معمولی `int` است. اما می‌توانید با استفاده از پارامتر `dtype`، نوع داده دلخواه خود را مشخص کنید.

`numpy.full\_like` -24یک تابع از ماژول `numpy` است که یک آرایه جدید با مقادیر تکراری و با همان ابعاد و نوع داده یک آرایه ورودی ایجاد می‌کند.

توضیحات مربوط به تابع `numpy.full\_like` را برای شما آورده‌ام:

- `numpy.full\_like(a, fill\_value, dtype=None, order='K', subok=True, shape=None)`:

- آرگومان‌ها:

- `a`: آرایه ورودی به عنوان الگو برای ایجاد آرایه جدید.

- `fill\_value`: مقدار تکراری برای پر کردن آرایه جدید.

- `dtype`: نوع داده آرایه جدید. (اختیاری، پیش‌فرض: None)

- `order`: الگوی مرتب‌سازی داده‌های آرایه جدید در حافظه. (اختیاری، پیش‌فرض: 'K')

- `subok`: مشخص می‌کند آیا آرایه جدید باید یک زیرکلاس از آرایه ورودی باشد یا نه. (اختیاری، پیش‌فرض: True)

- `shape`: ابعاد آرایه جدید. اگر این پارامتر مشخص نشود، ابعاد آرایه ورودی برداشت می‌شود. (اختیاری)

- خروجی: آرایه جدید با مقادیر تکراری و با همان ابعاد و نوع داده آرایه ورودی.

در زیر مثالی از استفاده از تابع `numpy.full\_like` را برای ایجاد یک آرایه جدید با مقادیر تکراری و با همان ابعاد و نوع داده یک آرایه ورودی ارائه داده‌ام:

```python

import numpy as np

# آرایه ورودی

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

# ایجاد آرایه جدید با مقادیر تکراری 7 و با همان ابعاد و نوع داده آرایه ورودی

full\_array = np.full\_like(arr, 7)

print(full\_array)

```

`numpy.array’-25یک تابع از ماژول `numpy` است که یک آرایه جدید از داده‌های ورودی ایجاد می‌کند.

توضیحات مربوط به تابع `numpy.array` را برای شما آورده‌ام:

- `numpy.array(object, dtype=None, copy=True, order='K', subok=False, ndmin=0)`:

- آرگومان‌ها:

- `object`: داده‌های ورودی که به صورت آرایه تبدیل خواهند شد. می‌تواند یک لیست، تاپل، دنباله نمونه‌ای، یا شیء دیگری با رابط `\_\_array\_\_` باشد.

- `dtype`: نوع داده آرایه جدید. (اختیاری، پیش‌فرض: None)

- `copy`: مشخص می‌کند آیا یک کپی از داده ورودی باید ایجاد شود یا نه. (اختیاری، پیش‌فرض: True)

- `order`: الگوی مرتب‌سازی داده‌های آرایه در حافظه. (اختیاری، پیش‌فرض: 'K')

- `subok`: مشخص می‌کند آیا آرایه جدید باید یک زیرکلاس از `ndarray` باشد یا نه. (اختیاری، پیش‌فرض: False)

- `ndmin`: تعداد حداقل ابعاد آرایه جدید. (اختیاری، پیش‌فرض: 0)

- خروجی: آرایه جدید ایجاد شده با داده‌های ورودی.

در زیر مثالی از استفاده از تابع `numpy.array` را برای ایجاد یک آرایه جدید از داده‌های ورودی ارائه داده‌ام:

```python

import numpy as np

# لیست ورودی

lst = [1, 2, 3, 4, 5]

# ایجاد آرایه جدید از لیست ورودی

arr = np.array(lst)

print(arr)

```

`numpy.asarray` -26یک تابع از ماژول `numpy` است که ورودی را به یک آرایه نامپای تبدیل می‌کند، اگر ورودی از نوع آرایه باشد آن را کپی می‌کند و اگر ورودی از نوع ماتریس می‌باشد، آن را به یک آرایه نامپای تبدیل می‌کند.

توضیحات مربوط به تابع `numpy.asarray` را برای شما آورده‌ام:

- `numpy.asarray(a, dtype=None, order=None)`:

- آرگومان‌ها:

- `a`: داده ورودی که به آرایه تبدیل می‌شود.

- `dtype`: نوع داده آرایه جدید. (اختیاری، پیش‌فرض: None)

- `order`: الگوی مرتب‌سازی داده‌های آرایه در حافظه. (اختیاری)

- خروجی: آرایه نامپای حاوی داده‌های ورودی.

در زیر مثالی از استفاده از تابع `numpy.asarray` را برای تبدیل ورودی به آرایه نامپای ارائه داده‌ام:

```python

import numpy as np

# لیست ورودی

lst = [1, 2, 3, 4, 5]

# تبدیل لیست به آرایه نامپای

arr = np.asarray(lst)

print(arr)

```

در این مثال، یک لیست `lst` داریم که داده‌های ورودی را نمایش می‌دهد. با استفاده از تابع `numpy.asarray`، لیست را به یک آرایه نامپای تبدیل می‌کنیم و در متغیر `arr` ذخیره می‌کنیم. خروجی به صورت زیر خواهد بود:

```

[1 2 3 4 5]

```

تابع `numpy.asarray` می‌تواند برای تبدیل داده‌های متنوعی به آرایه نامپای استفاده شود، از جمله لیست‌ها، تاپل‌ها، دنباله‌های نمونه‌ای و سایر شیء‌هایی که متد `\_\_array\_\_` را پیاده‌سازی کرده‌اند.

`numpy.asanyarray` -27یک تابع از ماژول `numpy` است که ورودی را به یک آرایه نامپای تبدیل می‌کند، اما در صورتی که ورودی از قبل یک آرایه نامپای باشد، کپی از آن نمی‌سازد و آن را به عنوان خروجی برمی‌گرداند.

توضیحات مربوط به تابع `numpy.asanyarray` را برای شما آورده‌ام:

- `numpy.asanyarray(a, dtype=None)`:

- آرگومان‌ها:

- `a`: داده ورودی که به آرایه تبدیل می‌شود.

- `dtype`: نوع داده آرایه جدید. (اختیاری، پیش‌فرض: None)

- خروجی: آرایه نامپای حاوی داده‌های ورودی.

تفاوت اصلی بین `numpy.asarray` و `numpy.asanyarray` در این است که `numpy.asarray` همیشه یک کپی از ورودی را ایجاد می‌کند، در حالی که `numpy.asanyarray` در صورتی که ورودی از قبل یک آرایه نامپای باشد، کپی ایجاد نمی‌کند و خود آرایه ورودی را به عنوان خروجی برمی‌گرداند.

در زیر مثالی از استفاده از تابع `numpy.asanyarray` را برای تبدیل ورودی به آرایه نامپای ارائه داده‌ام:

```python

import numpy as np

# لیست ورودی

lst = [1, 2, 3, 4, 5]

# تبدیل لیست به آرایه نامپای

arr = np.asanyarray(lst)

print(arr)

```

در این مثال، یک لیست `lst` داریم که داده‌های ورودی را نمایش می‌دهد. با استفاده از تابع `numpy.asanyarray`، لیست را به یک آرایه نامپای تبدیل می‌کنیم و در متغیر `arr` ذخیره می‌کنیم. خروجی به صورت زیر خواهد بود:

```

[1 2 3 4 5]

```

`numpy.ascontiguousarray` -28یک تابع از ماژول `numpy` است که ورودی را به یک آرایه نامپای تبدیل می‌کند، اما در صورتی که آرایه ورودی مرتب بر روی حافظه نباشد، یک آرایه مرتب از ورودی ایجاد می‌کند.

توضیحات مربوط به تابع `numpy.ascontiguousarray` را برای شما آورده‌ام:

- `numpy.ascontiguousarray(a, dtype=None)`:

- آرگومان‌ها:

- `a`: داده ورودی که به آرایه تبدیل می‌شود.

- `dtype`: نوع داده آرایه جدید. (اختیاری، پیش‌فرض: None)

- خروجی: آرایه نامپای حاوی داده‌های ورودی.

تابع `numpy.ascontiguousarray` از آرگومان `a` به عنوان ورودی می‌گیرد و یک آرایه نامپای از آن ایجاد می‌کند. اگر آرایه ورودی در حالت مرتب بر روی حافظه قرار نداشته باشد، یعنی ابتدا بر روی آن عملیات مرتب‌سازی بایتی انجام شود تا به حافظه متوالی دسترسی شود.

در زیر مثالی از استفاده از تابع `numpy.ascontiguousarray` را برای تبدیل ورودی به آرایه نامپای ارائه داده‌ام:

```python

import numpy as np

# آرایه ورودی غیرمرتب

a = np.array([[1, 2, 3], [4, 5, 6]], order='F')

# تبدیل آرایه به حالت مرتب

b = np.ascontiguousarray(a)

print(b)

```

آرایه اصلی:

[[1 2 3]

[4 5 6]]

آرایه مرتب:

[[1 2 3]

[4 5 6]]

`numpy.asmatrix` -29یک تابع از ماژول `numpy` است که ورودی را به یک ماتریس نامپای تبدیل می‌کند. این تابع یک نمونه از کلاس `numpy.matrix` را ایجاد می‌کند که ویژگی‌های خاصی نسبت به آرایه نامپای است.

توضیحات مربوط به تابع `numpy.asmatrix` را برای شما آورده‌ام:

- `numpy.asmatrix(data, dtype=None)`:

- آرگومان‌ها:

- `data`: داده ورودی که به ماتریس تبدیل می‌شود.

- `dtype`: نوع داده ماتریس جدید. (اختیاری، پیش‌فرض: None)

- خروجی: ماتریس نامپای حاوی داده‌های ورودی.

تابع `numpy.asmatrix` از آرگومان `data` به عنوان ورودی می‌گیرد و یک ماتریس نامپای از آن ایجاد می‌کند. ورودی `data` می‌تواند یک آرایه نامپای، یک لیست، یک داده دوبعدی، یک رشته فضای نامپای و یا یک شیء نامپای باشد.

در زیر مثالی از استفاده از تابع `numpy.asmatrix` را برای تبدیل ورودی به یک ماتریس نامپای ارائه داده‌ام:

```python

import numpy as np

# آرایه ورودی

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

# تبدیل آرایه به ماتریس نامپای

mat = np.asmatrix(arr)

print(mat)

```

در این مثال، یک آرایه دوبعدی `arr` داریم که داده‌های ورودی را نمایش می‌دهد. با استفاده از تابع `numpy.asmatrix`، آرایه را به یک ماتریس نامپای تبدیل می‌کنیم و در متغیر `mat` ذخیره می‌کنیم. خروجی به صورت زیر خواهد بود:

```

[[1 2]

[3 4]]

```

در اینجا ماتریس نامپای حاوی داده‌های ورودی، به صورت یک ماتریس 2x2 نشان داده شده است.

`numpy.asfarray` -30یک تابع از ماژول `numpy` است که ورودی را به یک آرایه نامپای با نوع داده float تبدیل می‌کند. این تابع برخلاف تابع `numpy.asarray`، همیشه یک آرایه با نوع داده float ایجاد می‌کند.

توضیحات مربوط به تابع `numpy.asfarray` را برای شما آورده‌ام:

- `numpy.asfarray(a, dtype=<class 'float'>)`:

- آرگومان‌ها:

- `a`: داده ورودی که به آرایه تبدیل می‌شود.

- `dtype`: نوع داده آرایه جدید. (اختیاری، پیش‌فرض: float)

- خروجی: آرایه نامپای حاوی داده‌های ورودی با نوع float.

تابع `numpy.asfarray` از آرگومان `a` به عنوان ورودی می‌گیرد و یک آرایه نامپای با نوع داده float از آن ایجاد می‌کند. در صورتی که آرگومان `dtype` مشخص نشده باشد، آرایه با نوع داده پیش‌فرض float ایجاد می‌شود.

در زیر مثالی از استفاده از تابع `numpy.asfarray` را برای تبدیل ورودی به یک آرایه نامپای با نوع float ارائه داده‌ام:

```python

import numpy as np

# آرایه ورودی

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

# تبدیل آرایه به آرایه نامپای با نوع float

arr\_float = np.asfarray(arr)

print(arr\_float)

```

در این مثال، یک آرایه `arr` داریم که داده‌های ورودی را نمایش می‌دهد. با استفاده از تابع `numpy.asfarray`، آرایه را به یک آرایه نامپای با نوع float تبدیل می‌کنیم و در متغیر `arr\_float` ذخیره می‌کنیم. خروجی به صورت زیر خواهد بود:

```

[1. 2. 3.]

```

در اینجا آرایه نامپای حاوی داده‌های ورودی با نوع float نشان داده شده است.

`numpy.ascontiguousarray` -31یک تابع از ماژول `numpy` است که ورودی را به یک آرایه نامپای متوالی (contiguous) تبدیل می‌کند. این تابع یک آرایه جدید ایجاد می‌کند که حافظه مستمری را برای آن تخصیص می‌دهد.

توضیحات مربوط به تابع `numpy.ascontiguousarray` را برای شما آورده‌ام:

- `numpy.ascontiguousarray(a, dtype=None)`:

- آرگومان‌ها:

- `a`: داده ورودی که به آرایه تبدیل می‌شود.

- `dtype`: نوع داده آرایه جدید. (اختیاری، پیش‌فرض: None)

- خروجی: آرایه نامپای حاوی داده‌های ورودی با حالت متوالی.

تابع `numpy.ascontiguousarray` با گرفتن آرگومان `a` به عنوان ورودی، یک آرایه نامپای متوالی از آن ایجاد می‌کند. این تابع مفید است زمانی که می‌خواهید با داده‌هایی که در حافظه به صورت غیرمتوالی قرار دارند، کار کنید و نیاز به حافظه متوالی دارید.

در زیر مثالی از استفاده از تابع `numpy.ascontiguousarray` را برای تبدیل ورودی به یک آرایه نامپای متوالی ارائه داده‌ام:

```python

import numpy as np

# آرایه ورودی غیرمتوالی

arr = np.array([[1, 2, 3], [4, 5, 6]], order='F')

# تبدیل آرایه به آرایه نامپای متوالی

arr\_contiguous = np.ascontiguousarray(arr)

print(arr\_contiguous)

```

`numpy.asmatrix` -32یک تابع از ماژول `numpy` است که ورودی را به یک ماتریس نامپای تبدیل می‌کند. این تابع یک نمونه از کلاس `numpy.matrix` را ایجاد می‌کند که ویژگی‌های خاصی نسبت به آرایه نامپای دارد.

توضیحات مربوط به تابع `numpy.asmatrix` را برای شما آورده‌ام:

- `numpy.asmatrix(data, dtype=None)`:

- آرگومان‌ها:

- `data`: داده ورودی که به ماتریس تبدیل می‌شود.

- `dtype`: نوع داده ماتریس جدید. (اختیاری، پیش‌فرض: None)

- خروجی: ماتریس نامپای حاوی داده‌های ورودی.

تابع `numpy.asmatrix` از آرگومان `data` به عنوان ورودی می‌گیرد و یک ماتریس نامپای از آن ایجاد می‌کند. ورودی `data` می‌تواند یک آرایه نامپای، یک لیست، یک داده دوبعدی، یک رشته فضای نامپای و یا یک شیء نامپای باشد.

در زیر مثالی از استفاده از تابع `numpy.asmatrix` را برای تبدیل ورودی به یک ماتریس نامپای ارائه داده‌ام:

```python

import numpy as np

# آرایه ورودی

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

# تبدیل آرایه به ماتریس نامپای

mat = np.asmatrix(arr)

print(mat)

```

در این مثال، یک آرایه دوبعدی `arr` داریم که داده‌های ورودی را نمایش می‌دهد. با استفاده از تابع `numpy.asmatrix`، آرایه را به یک ماتریس نامپای تبدیل می‌کنیم و در متغیر `mat` ذخیره می‌کنیم. خروجی به صورت زیر خواهد بود:

```

[[1 2]

[3 4]]

```

`numpy.copy` -33یک تابع از ماژول `numpy` است که یک کپی از آرایه نامپای ورودی ایجاد می‌کند. این تابع برای ایجاد یک نسخه مستقل از آرایه استفاده می‌شود، به طوری که تغییرات در کپی، آرایه اصلی را تحت تأثیر قرار نمی‌دهد.

توضیحات مربوط به تابع `numpy.copy` را برای شما آورده‌ام:

- `numpy.copy(a, order='K')`:

- آرگومان‌ها:

- `a`: آرایه نامپای ورودی که کپی می‌شود.

- `order`: نحوه مرتب‌سازی داده‌ها در کپی. (اختیاری، پیش‌فرض: 'K')

- خروجی: کپی مستقل از آرایه ورودی.

تابع `numpy.copy` با گرفتن آرگومان `a` به عنوان ورودی، یک کپی مستقل از آرایه نامپای `a` ایجاد می‌کند. تغییرات در کپی بر روی آرایه اصلی `a` تأثیری نمی‌گذارد و برعکس.

در زیر مثالی از استفاده از تابع `numpy.copy` را برای ایجاد یک کپی مستقل از آرایه نامپای ارائه داده‌ام:

```python

import numpy as np

# آرایه ورودی

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

# ایجاد کپی مستقل از آرایه

arr\_copy = np.copy(arr)

# تغییر در کپی

arr\_copy[0] = 10

print(arr) # خروجی: [1 2 3]

print(arr\_copy) # خروجی: [10 2 3]

```

`numpy.copy` -34یک تابع از ماژول `numpy` است که یک کپی از آرایه نامپای ورودی ایجاد می‌کند. این تابع برای ایجاد یک نسخه مستقل از آرایه استفاده می‌شود، به طوری که تغییرات در کپی، آرایه اصلی را تحت تأثیر قرار نمی‌دهد.

توضیحات مربوط به تابع `numpy.copy` را برای شما آورده‌ام:

- `numpy.copy(a, order='K', subok=False)`:

- آرگومان‌ها:

- `a`: آرایه نامپای ورودی که کپی می‌شود.

- `order`: نحوه مرتب‌سازی داده‌ها در کپی. (اختیاری، پیش‌فرض: 'K')

- `subok`: اگر True باشد، نوع زیرین کپی ممکن است نسبت به نوع ورودی حاوی تغییرات کوچکی داشته باشد. (اختیاری، پیش‌فرض: False)

- خروجی: کپی مستقل از آرایه ورودی.

تابع `numpy.copy` با گرفتن آرگومان `a` به عنوان ورودی، یک کپی مستقل از آرایه نامپای `a` ایجاد می‌کند. تغییرات در کپی بر روی آرایه اصلی `a` تأثیری نمی‌گذارد و برعکس.

آرگومان `order` نحوه مرتب‌سازی داده‌ها در کپی را تعیین می‌کند. مقادیر ممکن برای `order` عبارتند از:

- `'C'`: مرتب‌سازی در حالت ردیفی (row-major).

- `'F'`: مرتب‌سازی در حالت ستونی (column-major).

- `'A'`: مرتب‌سازی بر اساس نوع داده آرایه و ورودی کپی.

آرگومان `subok` نشان می‌دهد که آیا نوع زیرین کپی ممکن است نسبت به نوع ورودی حاوی تغییرات کوچکی داشته باشد یا خیر.

import numpy as np

# آرایه ورودی

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

# ایجاد کپی مستقل از آرایه با تغییر نحوه مرتب‌سازی

arr\_copy = np.copy(arr, order='F')

# تغییر در کپی

arr\_copy[0] = 10

print(arr) # خروجی: [1 2 3]

print(arr\_copy) # خروجی: [10 2 3]

`numpy.frombuffer` -35یک تابع از ماژول `numpy` است که داده‌های آرایه را از بوفر داده شده ایجاد می‌کند. بوفر می‌تواند یک شیء با نوع `bytes` یا `bytearray` باشد که حاوی داده‌های مربوط به آرایه است. با استفاده از این تابع می‌توانید داده‌های آرایه را مستقیماً از بوفر موجود در حافظه بخوانید و به آرایه نامپای تبدیل کنید.

توضیحات مربوط به تابع `numpy.frombuffer` را برای شما آورده‌ام:

- `numpy.frombuffer(buffer, dtype=float, count=-1, offset=0, \*, like=None)`:

- آرگومان‌ها:

- `buffer`: بوفر داده که شامل داده‌های آرایه است.

- `dtype`: نوع داده مورد انتظار برای آرایه نامپای خروجی. (اختیاری، پیش‌فرض: float)

- `count`: تعداد عناصر مورد استفاده برای ساخت آرایه نامپای. اگر مقدار -1 داده شود، تمام داده‌ها استفاده خواهند شد. (اختیاری، پیش‌فرض: -1)

- `offset`: شیفت مورد استفاده در بوفر برای شروع خواندن داده‌ها. (اختیاری، پیش‌فرض: 0)

- `like`: آرایه نامپای یا شیء قابل تبدیل به آرایه که به عنوان قالب برای آرایه خروجی استفاده می‌شود. اگر مقدار داده نشود، آرایه خروجی با استفاده از `dtype` ساخته می‌شود. (اختیاری)

- خروجی: آرایه نامپای حاوی داده‌های موجود در بوفر.

تابع `numpy.frombuffer` با گرفتن آرگومان `buffer` به عنوان ورودی، داده‌های موجود در بوفر را خوانده و یک آرایه نامپای با استفاده از نوع داده مشخص شده (`dtype`) ایجاد می‌کند.

البته! در ادامه یک کد نمونه از استفاده از تابع `numpy.frombuffer` را برای شما آورده‌ام:

```python

import numpy as np

# بوفر داده

buffer = b'Hello, World!'

# ایجاد آرایه نامپای از بوفر با نوع داده مشخص

arr = np.frombuffer(buffer, dtype='S1')

print(arr) # خروجی: [b'H' b'e' b'l' b'l' b'o' b',' b' ' b'W' b'o' b'r' b'l' b'd' b'!']

```

در این مثال، یک بوفر داده به نام `buffer` تعریف شده است که شامل رشته "Hello, World!" است. با استفاده از تابع `numpy.frombuffer` و مشخص کردن نوع داده `'S1'` (رشته بایت با طول 1)، یک آرایه نامپای به نام `arr` ایجاد می‌شود. داده‌های موجود در بوفر خوانده شده و به صورت رشته‌های بایتی در آرایه `arr` قرار می‌گیرند. سپس آرایه `arr` نمایش داده می‌شود که هر عنصر آن یک بایت از رشته اصلی را نشان می‌دهد.

-36تابع `numpy.from\_dlpack` یک تابع از ماژول `numpy` است که از یک DLPack capsule به عنوان ورودی استفاده می‌کند و آرایه نامپای معادل را ایجاد می‌کند. DLPack یک استاندارد مشترک برای تبادل داده‌های آرایه بین کتابخانه‌های عددی مختلف است.

توضیحات مربوط به تابع `numpy.from\_dlpack` را برای شما آورده‌ام:

- `numpy.from\_dlpack(x, /)`:

- آرگومان‌ها:

- `x`: DLPack capsule که حاوی اطلاعات آرایه است.

- خروجی: آرایه نامپای حاوی داده‌های موجود در DLPack capsule.

برای استفاده از این تابع، باید ابتدا یک DLPack capsule از کتابخانه‌ای که از آن استفاده می‌کنید بدست آورید، سپس این capsule را به عنوان ورودی به تابع `numpy.from\_dlpack` بدهید تا آرایه نامپای معادل ایجاد شود.

```python

import numpy as np

import torch

# تبدیل یک آرایه PyTorch به DLPack capsule

x = torch.randn(3, 4)

dlpack = x.to\_dlpack()

# ایجاد آرایه نامپای از DLPack capsule

arr = np.from\_dlpack(dlpack)

print(arr) # خروجی: محتوای آرایه نامپای معادل با آرایه PyTorch

```

در این مثال، یک آرایه PyTorch به نام `x` ایجاد شده است. ابتدا آن را به یک DLPack capsule تبدیل می‌کنیم با استفاده از `to\_dlpack()`، سپس از `numpy.from\_dlpack` برای ایجاد یک آرایه نامپای معادل استفاده می‌کنیم. در نهایت، محتوای آرایه نامپای ساخته شده را چاپ می‌کنیم.

خروجی:

[[ 1.3077087 0.5795277 -0.36383373 -0.2760452 ]

[-0.7689914 0.13130724 -0.777213 -0.1932758 ]

[ 0.3370659 -0.5403493 -0.43142557 0.6412978 ]]

در این مثال، خروجی نشان داده شده یک آرایه نامپای سه‌بعدی به ابعاد (3, 4) است که مقادیر تصادفی است.

38-تابع `numpy.fromfile` یک فایل را به عنوان ورودی دریافت می‌کند و محتوای آن را به عنوان آرایه نامپای بر می‌گرداند.

توضیحات مربوط به تابع `numpy.fromfile` را برای شما آورده‌ام:

- `numpy.fromfile(file, dtype=float, count=-1, sep='', offset=0)`:

- آرگومان‌ها:

- `file`: نام فایل یا شیء فایل باز برای خواندن داده‌ها.

- `dtype`: نوع داده مورد انتظار برای مقادیر در فایل. پیش‌فرض: `float`.

- `count`: تعداد مقادیری که باید از فایل خوانده شوند. اگر -1 باشد، تمام مقادیر فایل خوانده می‌شوند. پیش‌فرض: -1.

- `sep`: جداکننده بین مقادیر در فایل. این آرگومان تنها برای فایل‌های متنی استفاده می‌شود و برای فایل‌های دودویی نادیده گرفته می‌شود. پیش‌فرض: '' (جداکننده‌ای نداریم).

- `offset`: تعداد بایت‌هایی که باید در ابتدای فایل رد شوند قبل از خواندن داده‌ها. پیش‌فرض: 0.

- خروجی: آرایه نامپای حاوی مقادیر خوانده شده از فایل.

در زیر، یک مثال نمایش داده شده است:

```python

import numpy as np

# خواندن محتوای یک فایل به عنوان آرایه نامپای

arr = np.fromfile('data.txt', dtype=float, sep=',')

print(arr) # چاپ آرایه نامپای حاوی محتوای فایل

```

در این مثال، تابع `np.fromfile` برای خواندن محتوای فایل `data.txt` استفاده شده است. مقادیر فایل جداشده توسط کاما هستند و با استفاده از `dtype=float`، مقادیر به صورت اعشاری خوانده می‌شوند. سپس آرایه حاوی محتوای فایل خوانده شده، چاپ می‌شود.

```

[1.23 4.56 7.89 10.11]

```

در این مثال، فایل `data.txt` شامل چهار مقدار اعشاری است که با کاما جدا شده‌اند. با استفاده از `np.fromfile` و تنظیمات مشخص شده، این مقادیر از فایل خوانده می‌شوند و در آرایه نامپای ذخیره می‌شوند. بنابراین، خروجی بستگی به محتوای فایل `data.txt` خواهد داشت.

39-تابع `numpy.fromfunction` برای ایجاد یک آرایه نامپای با استفاده از یک تابع مشخص و ابعاد داده شده استفاده می‌شود.

توضیحات مربوط به تابع `numpy.fromfunction` را برای شما آورده‌ام:

- `numpy.fromfunction(function, shape, \*, dtype=float, like=None, \*\*kwargs)`:

- آرگومان‌ها:

- `function`: تابع که برای محاسبه مقادیر آرایه استفاده می‌شود.

- `shape`: ابعاد آرایه نامپای مورد نظر.

- `dtype`: نوع داده مورد انتظار برای مقادیر در آرایه. پیش‌فرض: `float`.

- `like`: شیء مرجع که به عنوان الگو برای ساختار آرایه استفاده می‌شود.

- `\*\*kwargs`: آرگومان‌های اضافی که به تابع `function` منتقل می‌شوند.

```python

import numpy as np

# تعریف تابع محاسبه

def my\_function(x, y):

return x + y

# ساخت آرایه نامپای با استفاده از تابع مشخص و ابعاد داده شده

arr = np.fromfunction(my\_function, (3, 3), dtype=int)

print(arr) # چاپ آرایه نامپای ساخته شده

```

در این مثال، تابع `my\_function` تعریف شده است که دو آرگومان `x` و `y` را دریافت کرده و جمع آنها را برمی‌گرداند. سپس با استفاده از `np.fromfunction` و تابع `my\_function` به همراه ابعاد `(3, 3)`، یک آرایه نامپای ساخته می‌شود که مقادیر آن با استفاده از تابع محاسبه شده است. در نهایت، آرایه ساخته شده چاپ می‌شود.

با توجه به کد نمونه ارائه شده، خروجی آرایه نامپای ایجاد شده با استفاده از `numpy.fromfunction` و تابع `my\_function` با ابعاد (3, 3) و نوع داده `int` به صورت زیر خواهد بود:

```

[[0 1 2]

[1 2 3]

[2 3 4]]

```

در این مثال، مقادیر آرایه نامپای ساخته شده با استفاده از تابع `my\_function` به این صورت محاسبه می‌شوند:

- در سطر اول آرایه، مقدار `my\_function(0, 0)` در عنصر اول، مقدار `my\_function(0, 1)` در عنصر دوم و مقدار `my\_function(0, 2)` در عنصر سوم قرار می‌گیرد.

- در سطر دوم آرایه، مقدار `my\_function(1, 0)` در عنصر اول، مقدار `my\_function(1, 1)` در عنصر دوم و مقدار `my\_function(1, 2)` در عنصر سوم قرار می‌گیرد.

- در سطر سوم آرایه، مقدار `my\_function(2, 0)` در عنصر اول، مقدار `my\_function(2, 1)` در عنصر دوم و مقدار `my\_function(2, 2)` در عنصر سوم قرار می‌گیرد.

بنابراین، خروجی نشان داده شده، آرایه نامپای حاوی مقادیر محاسبه شده است.

40-تابع `numpy.fromiter` برای ایجاد یک آرایه نامپای از یک تکرارکننده (iterable) با استفاده از نوع داده مشخص شده استفاده می‌شود.

توضیحات مربوط به تابع `numpy.fromiter` را برای شما آورده‌ام:

- `numpy.fromiter(iter, dtype, count=-1, \*, like=None)`:

- آرگومان‌ها:

- `iter`: یک تکرارکننده (iterable) که مقادیر آرایه را تولید می‌کند.

- `dtype`: نوع داده مورد انتظار برای مقادیر در آرایه.

- `count`: تعداد مقادیری که باید از تکرارکننده خوانده شوند. اگر `-1` باشد، تمام مقادیر خوانده می‌شوند.

- `like`: شیء مرجع که به عنوان الگو برای ساختار آرایه استفاده می‌شود.

- خروجی: آرایه نامپای حاوی مقادیر خوانده شده از تکرارکننده.

در زیر، یک مثال نمایش داده شده است:

```python

import numpy as np

# تعریف یک تکرارکننده

my\_iter = iter([1, 2, 3, 4, 5])

# ساخت آرایه نامپای با استفاده از تکرارکننده و نوع داده مشخص

arr = np.fromiter(my\_iter, dtype=int)

print(arr) # چاپ آرایه نامپای ساخته شده

```

در این مثال، یک تکرارکننده `my\_iter` با استفاده از لیست `[1, 2, 3, 4, 5]` تعریف شده است. سپس با استفاده از `np.fromiter` و تکرارکننده `my\_iter` و نوع داده `int`، یک آرایه نامپای ساخته می‌شود که مقادیر آن از تکرارکننده خوانده شده‌اند. در نهایت، آرایه ساخته شده چاپ می‌شود.

41-متد `ndarray.tofile` برای نوشتن محتوای یک آرایه به صورت دودویی در یک فایل استفاده می‌شود. توضیحات مربوط به متد `ndarray.tofile` به صورت زیر است:

```

ndarray.tofile(fid, sep='', format='%s')

پارامترها:

- `fid`: یک شیء فایل یا یک رشته حاوی نام فایل است. شیء فایل یا فایل مشخص شده با نام، به عنوان مقصد برای نوشتن داده‌ها استفاده می‌شود.

- `sep` (اختیاری): جداکننده بین عناصر آرایه برای خروجی متنی. این پارامتر فقط برای آرایه‌هایی که نوع داده آن‌ها به صورت رشته قابل نمایش هستند، قابل استفاده است.

- `format` (اختیاری): رشته فرمت استفاده شده برای نوشتن داده‌ها در فایل. این پارامتر فقط برای آرایه‌هایی که نوع داده آن‌ها به صورت رشته قابل نمایش هستند، قابل استفاده است.

مثال:

```python

import numpy as np

# ساخت یک آرایه

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

# نوشتن داده‌های آرایه به فایل به صورت دودویی

arr.tofile('output.bin')

```

در این مثال، یک آرایه `arr` با مقادیر `[1, 2, 3, 4, 5]` ساخته می‌شود. سپس از متد `tofile` برای نوشتن داده‌های آرایه به فایلی با نام 'output.bin' به صورت دودویی استفاده می‌شود.

لطفاً توجه داشته باشید که هنگام استفاده از متد `tofile`، داده به صورت دودویی نوشته می‌شود و قابل خواندن توسط انسان نیست.

42-متد `ndarray.tolist()` در آرایه‌های نامپای استفاده می‌شود تا محتوای آرایه را به صورت یک لیست پایتونی تبدیل کند.

در ادامه یک مثال نشان می‌دهد که چگونه از متد `tolist()` برای تبدیل یک آرایه نامپای به لیست پایتون استفاده کنیم:

```python

import numpy as np

# ساخت یک آرایه نامپای

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

# تبدیل آرایه به لیست پایتون

lst = arr.tolist()

# چاپ لیست پایتون

print(lst)

```

در این مثال، یک آرایه نامپای با مقادیر `[1, 2, 3, 4, 5]` ساخته می‌شود. سپس متد `tolist()` بر روی آرایه فراخوانی می‌شود تا آرایه به یک لیست پایتون تبدیل شود و نتیجه در متغیر `lst` ذخیره می‌شود. در نهایت، لیست پایتون به صورت `[1, 2, 3, 4, 5]` چاپ می‌شود.

لطفاً توجه داشته باشید که متد `tolist()` در نامپای باعث تبدیل آرایه به لیست می‌شود. این متد یک نسخه جدید از داده را ایجاد نمی‌کند و فقط یک مشاهده‌پذیری جدید از آرایه را در قالب لیست پایتون ارائه می‌دهد.

43-تابع `numpy.array2string()` در نامپای استفاده می‌شود تا یک آرایه را به صورت رشته‌ای نمایش دهد. این تابع به شما امکان می‌دهد تنظیمات مختلفی را برای نمایش آرایه استفاده کنید، مانند تعیین عرض خط حداکثر، تعیین دقت اعداد، قابلیت سرریزی اعداد کوچک و موارد دیگر.

در ادامه، یک مثال از استفاده از `numpy.array2string()` را نشان می‌دهم:

```python

import numpy as np

# ساخت یک آرایه نامپای

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

# تبدیل آرایه به رشته با استفاده از تابع array2string()

array\_string = np.array2string(arr)

# چاپ رشته حاوی آرایه

print(array\_string)

```

در این مثال، یک آرایه نامپای با مقادیر `[1, 2, 3, 4, 5]` ساخته می‌شود. سپس تابع `numpy.array2string()` بر روی آرایه فراخوانی می‌شود تا آرایه به یک رشته نمایش داده شود و نتیجه در متغیر `array\_string` ذخیره می‌شود. در نهایت، رشته حاوی آرایه به صورت `[1 2 3 4 5]` چاپ می‌شود.

مهم است به یاد داشته باشید که `numpy.array2string(max\_line\_width: , precision suppress\_small, separator, prefix) ,` به صورت پیش فرض از تنظیمات استاندارد برای نمایش آرایه استفاده می‌کند.

a: آرایه ورودی که قرار است به رشته تبدیل شود.

max\_line\_width: حداکثر عرض خط برای نمایش رشته. اگر طول رشته بیشتر از این حد باشد، خطوط جدید اضافه می‌شوند.

precision: دقت اعداد اعشاری در رشته نمایش داده شده.

suppress\_small: اگر برابر با True باشد، اعداد کوچک در رشته نمایش داده نمی‌شوند و به جای آن علامت "..." قرار می‌گیرد.

separator: جداکننده بین عناصر آرایه در رشته نمایش داده شده.

prefix: پیشوندی که به رشته نمایش داده شده اضافه می‌شود.

44-تابع `numpy.loadtxt()` در نامپای برای خواندن داده‌های عددی از یک فایل متنی استفاده می‌شود. این تابع به شما امکان می‌دهد فایل را بارگیری کنید و داده‌های عددی را به صورت آرایه نامپای دریافت کنید.

در ادامه، توضیح مختصری از آرگومان‌های `numpy.loadtxt(fname,dtype,comments,delimiter,converters,skiprows,usecols,

unpack,ndim,encoding,max\_rows, quotechar,like)` را ارائه می‌دهم:

- `fname`: نام فایل که می‌خواهید داده‌ها از آن خوانده شوند.

- `dtype`: نوع داده مورد انتظار برای آرایه خروجی. برای مقادیر عددی، می‌توانید از انواع داده نامپای مانند `float`, `int` و غیره استفاده کنید.

- `comments`: کاراکتر(های)ی که به عنوان نمادهای نظر در فایل تلقی می‌شوند و خطوط حاوی آن‌ها در نظر گرفته نمی‌شوند.

- `delimiter`: جداکننده بین داده‌ها در فایل متنی. برای جداکننده پیش فرض، از مقدار `None` استفاده می‌شود که هر فاصله خالی یا تب را به عنوان جداکننده در نظر می‌گیرد.

- `converters`: یک دیکشنری که به شما امکان می‌دهد تابع‌های تبدیل را برای تبدیل داده‌ها در هر ستون مشخص کنید.

- `skiprows`: تعداد خطوطی که در ابتدای فایل باید رد شوند.

- `usecols`: شماره ستون‌هایی که می‌خواهید خوانده شوند. می‌توانید از یک لیست یا تاپل استفاده کنید.

- `unpack`: اگر برابر با True باشد، داده‌ها به عنوان ستون‌های جداگانه دریافت می‌شوند.

- `ndmin`: حداقل تعداد ابعادی که آرایه خروجی باید داشته باشد.

- `encoding`: رمزگذاری مورد استفاده برای خواند.

ادامه توضیحات آرگومان‌های `numpy.loadtxt()` به شرح زیر است:

- `max\_rows`: تعداد حداکثر خطوطی که باید از فایل خوانده شوند. اگر برابر با `None` باشد، تمام خطوط خوانده می‌شوند.

- `quotechar`: کاراکتری که برای قرار دادن نمادهای نقل قول در داده‌ها استفاده می‌شود.

- `like`: آرایه‌ای که مشخص می‌کند خروجی باید به همان شکل و نوع داده آن آرایه باشد.

البته! در ادامه یک نمونه کد برای استفاده از تابع `numpy.loadtxt()` را ارائه می‌دهم:

```python

import numpy as np

# خواندن داده‌ها از فایل متنی

data = np.loadtxt('data.txt', delimiter=',', skiprows=1)

# چاپ آرایه خوانده شده

print(data)

```

در این نمونه کد، فرض می‌کنیم که داده‌ها در فایل متنی به نام `data.txt` قرار دارند. با استفاده از تابع `numpy.loadtxt()`، داده‌های موجود در فایل را می‌خوانیم و در آرایه `data` ذخیره می‌کنیم. سپس با استفاده از دستور `print(data)`، آرایه خوانده شده را نمایش می‌دهیم.

در این نمونه کد، مقادیر پیش‌فرض برای برخی از آرگومان‌ها مورد استفاده قرار گرفته است. شما می‌توانید آن‌ها را با مقادیر دلخواه خود جایگزین کنید تا نیازهای خاص خود را برآورده کنید.

45-تابع `numpy.core.records.array(obj,dtype,shape,offset,strides,names,copy)` در نامپای استفاده می‌شود تا آرایه‌ای از نوع رکورد را ایجاد کند. این تابع به شما امکان می‌دهد از مجموعه‌ای از اشیاء، نام‌ها و نوع‌ها، یک آرایه رکورد بسازید.

در ادامه، توضیح مختصری از آرگومان‌های `numpy.core.records.array()` را ارائه می‌دهم:

- `obj`: مجموعه‌ای از اشیاء و یا آرایه‌ها که می‌خواهید آن‌ها را به عنوان رکورد در آرایه بسازید.

- `dtype`: نوع داده مورد انتظار برای آرایه رکورد. این می‌تواند نوع رکورد تعریف شده با استفاده از `numpy.dtype` باشد یا یک نوع داده سفارشی.

- `shape`: شکل آرایه رکورد. این پارامتر مشخص می‌کند که آرایه چه تعداد رکورد دارد.

- `offset`: اندازه افست (offset) برای آرایه رکورد. این پارامتر مشخص می‌کند که چقدر از ابتدای بردار داده‌ها از آرایه رکورد رد شود.

- `strides`: اندازه گام (stride) برای هر بعد در آرایه رکورد. این پارامتر به شما امکان می‌دهد تا چگونگی قرارگیری داده‌ها در حافظه را مشخص کنید.

- `names`: نام‌های متناظر با هر رکورد در آرایه. این پارامتر باید با تعداد رکوردها هماهنگ باشد.

ادامه توضیحات آرگومان‌های `numpy.core.records.array()` به شرح زیر است:

- `- `copy`: مشخص می‌کند آیا داده‌های ورودی باید کپی شوند یا خیر.

برای استفاده از تابع `numpy.core.records.array()`، معمولاً ابتدا تعریف می‌کنید که هر رکورد چه نامی و چه نوع داده‌ای دارد، سپس مجموعه‌ای از اشیاء و یا آرایه‌ها را با توجه به این تعریف به عنوان آرگومان `obj` به تابع می‌دهید.

به عنوان یک نمونه، فرض کنید می‌خواهید آرایه‌ای رکوردی ایجاد کنید که شامل نام و سن دانش‌آموزان باشد. از `numpy.core.records.array()` برای ساختن آرایه رکورد استفاده می‌کنیم. در این مثال، دو دسته‌بندی نام و سن را تعریف می‌کنیم و سپس داده‌های مورد نظر را به عنوان آرگومان `obj` به تابع می‌دهیم. در انتها، آرایه رکورد حاصل را چاپ می‌کنیم.

```python

import numpy as np

# تعریف دسته‌بندی نام و سن

dtype = [('name', 'U20'), ('age', int)]

# داده‌های مورد نظر

data = [('John', 25), ('Alice', 30), ('Bob', 22)]

# ایجاد آرایه رکورد

arr = np.core.records.array(data, dtype=dtype)

# چاپ آرایه رکورد

print(arr)

```

خروجی:

```

[('John', 25) ('Alice', 30) ('Bob', 22)]

```

در این مثال، آرایه رکورد حاوی نام و سن سه دانش‌آموز به ترتیب جان، آلیس و باب ایجاد می‌شود و سپس چاپ می‌شود.

46-تابع `numpy.core.records.fromarrays(arrayList,dtype,shape,formats,names)` برای ساختن آرایه رکورد از آرایه‌های ورودی استفاده می‌شود. آرگومان‌های این تابع عبارتند از:

- `arrayList`: یک لیست از آرایه‌ها که برای ساخت آرایه رکورد استفاده می‌شوند. طول این لیست باید برابر با تعداد فیلدهای مورد نظر در آرایه رکورد باشد.

- `dtype`: نوع داده‌های رکورد. این پارامتر می‌تواند مقدار `None` داشته باشد، در این صورت نوع داده بر اساس نوع داده‌های ورودی تعیین می‌شود.

- `shape`: شکل آرایه رکورد. مقدار `None` به معنای استفاده از شکل خودکار است.

- `formats`: فرمت‌های داده‌های رکورد. این پارامتر می‌تواند مقدار `None` داشته باشد، در این صورت فرمت داده‌ها بر اساس نوع داده‌های ورودی تعیین می‌شود.

- `names`: نام‌های فیلدهای رکورد. این پارامتر می‌تواند مقدار `None` داشته باشد، در این صورت نام‌ها به صورت عددی تولید می‌شوند..

- ادامه مثال قبلی را با استفاده از `numpy.core.records.fromarrays()` بیان می‌کنم:

```python

import numpy as np

# داده‌های ورودی

names = np.array(['John', 'Alice', 'Bob'])

ages = np.array([25, 30, 22])

# ایجاد آرایه رکورد با استفاده از fromarrays

record\_array = np.core.records.fromarrays([names, ages], dtype=[('name', 'U20'), ('age', int)])

# چاپ آرایه رکورد

print(record\_array)

```

خروجی:

```

[('John', 25) ('Alice', 30) ('Bob', 22)]

```

در این مثال، دو آرایه `names` و `ages` را ایجاد می‌کنیم که به ترتیب نام و سن دانش‌آموزان را نشان می‌دهند. سپس از تابع `numpy.core.records.fromarrays()` برای ساخت آرایه رکورد استفاده می‌کنیم. در آرگومان `arrayList`، آرایه `names` و `ages` را به عنوان لیست ورودی به تابع می‌دهیم. در آرگومان `dtype` نوع داده‌های رکورد را مشخص می‌کنیم، در این حالت نوع داده‌ها به طور خودکار تشخیص داده می‌شود. در نهایت، آرایه رکورد حاصل را چاپ می‌کنیم و خروجی آن را مشاهده می‌کنیم که شامل نام و سن دانش‌آموزان است.

47-تابع `numpy.core.records.fromrecords(recList,dtype,shape,formats)` برای ساخت آرایه رکورد از لیست رکوردهای ورودی استفاده می‌شود. آرگومان‌های این تابع عبارتند از:

- `recList`: یک لیست از رکوردها که برای ساخت آرایه رکورد استفاده می‌شوند.

- `dtype`: نوع داده‌های رکورد. این پارامتر می‌تواند مقدار `None` داشته باشد، در این صورت نوع داده بر اساس نوع داده‌های ورودی تعیین می‌شود.

- `shape`: شکل آرایه رکورد. مقدار `None` به معنای استفاده از شکل خودکار است.

- `formats`: فرمت‌های داده‌های رکورد. این پارامتر می‌تواند مقدار `None` داشته باشد، در این صورت فرمت داده‌ها بر اساس نوع داده‌های ورودی تعیین می‌شود.

با استفاده از تابع `numpy.core.records.fromstring()`، می‌توانید یک رشته را به آرایه رکورد تبدیل کنید. در این مثال، فرض کنید رشته ورودی شامل اطلاعات مربوط به افراد باشد، هر رکورد شامل نام و سن فرد خواهد بود.

```python

import numpy as np

# رشته داده رکورد

datastring = b"John,25;Alice,30;Bob,22"

# تبدیل رشته به آرایه رکورد

record\_array = np.core.records.fromstring(datastring, dtype=[('name', 'U10'), ('age', int)])

# چاپ آرایه رکورد

print(record\_array)

```

خروجی:

```

[('John', 25) ('Alice', 30) ('Bob', 22)]

```

در این مثال، رشته داده رکورد را با نام `datastring` تعریف می‌کنیم. سپس از تابع `numpy.core.records.fromstring()` برای تبدیل رشته به آرایه رکورد استفاده می‌کنیم. در آرگومان `fromstring`، رشته `datastring` را به عنوان رشته ورودی به تابع می‌دهیم. در آرگومان `dtype`، نوع داده‌های رکورد را مشخص می‌کنیم، در این حالت نوع داده برای نام افراد `'U10'` (رشته با حداکثر طول 10 کاراکتر) و برای سن آن‌ها `int` تعیین شده است. در نهایت، آرایه رکورد حاصل را چاپ می‌کنیم و خروجی آن را مشاهده می‌کنیم که شامل نام و سن افراد است.

48-تابع `numpy.core.records.fromfile(fd,dtype,shape,offset,formats,)` برای خواندن داده‌ها از یک فایل و تبدیل آنها به آرایه رکورد استفاده می‌شود. آرگومان‌های این تابع عبارتند از:

- `fd`: فایل دسکریپتور فایلی که از آن می‌خواهید داده‌ها را بخوانید.

- `dtype`: نوع داده‌های رکورد. این پارامتر می‌تواند مقدار `None` داشته باشد، در این صورت نوع داده بر اساس نوع داده‌های ورودی تعیین می‌شود.

- `shape`: شکل آرایه رکورد. مقدار `None` به معنای استفاده از شکل خودکار است.

- `offset`: آفست برای خواندن داده‌ها از فایل.

- `formats`: فرمت‌های داده‌های رکورد. این پارامتر می‌تواند مقدار `None` داشته باشد، در این صورت فرمت داده‌ها بر اساس نوع داده‌های ورودی تعیین می‌شود.- `

به عنوان مثال، یک فایل به نام "data.bin" داریم که حاوی داده‌های رکورد است. می‌توانیم از تابع `numpy.core.records.fromfile()` برای خواندن این داده‌ها و تبدیل آنها به آرایه رکورد استفاده کنیم. در ادامه یک نمونه کد برای این منظور آمده است:

```python

import numpy as np

# تعریف نوع داده‌های رکورد

dtype = np.dtype([('name', 'S20'), ('age', 'int32'), ('height', 'float64')])

# باز کردن فایل برای خواندن داده‌ها

with open('data.bin', 'rb') as file:

# خواندن داده‌ها و تبدیل به آرایه رکورد

records = np.core.records.fromfile(file, dtype=dtype)

# چاپ آرایه رکورد

print(records)

```

در این مثال، فایل "data.bin" شامل داده‌های رکورد است که هر رکورد شامل سه فیلد "name"، "age" و "height" است. نوع داده‌های رکورد با استفاده از `np.dtype` تعریف شده است. سپس فایل با استفاده از `open()` باز می‌شود و داده‌ها با استفاده از `np.core.records.fromfile()` خوانده و به آرایه رکورد تبدیل می‌شوند. سپس آرایه رکورد چاپ می‌شود.

49-تابع `numpy.core.defchararray.array(obj,itemsize,copy,order)` برای ایجاد یک آرایه حرفی (string array) از یک شیء قابل تبدیل به رشته (convertible object) استفاده می‌شود. این تابع آرگومان‌های زیر را می‌پذیرد:

- `obj`: شیء قابل تبدیل به رشته که برای ساخت آرایه استفاده می‌شود.

- `itemsize`: اندازه هر عنصر آرایه حرفی. این پارامتر می‌تواند مقدار `None` داشته باشد، در این صورت اندازه هر عنصر بر اساس طول بزرگترین رشته در `obj` تعیین می‌شود.

- `copy`: مشخص می‌کند آیا یک کپی از `obj` برای ساخت آرایه استفاده شود یا نه. مقدار پیش‌فرض آن `True` است.

- `order`: ترتیب ذخیره سازی آرایه. مقادیر ممکن برای این پارامتر `'C'` (ذخیره سازی در حالت سطری، پیش‌فرض) یا `'F'` (ذخیره سازی در حالت ستونی) هستند.

در ادامه یک نمونه کد برای استفاده از تابع `numpy.core.defchararray.array()` آمده است:

```python

import numpy as np

# شیء قابل تبدیل به رشته

data = ['apple', 'banana', 'cherry']

# ساخت آرایه حرفی از شیء

string\_array = np.core.defchararray.array(data)

# چاپ آرایه حرفی

print(string\_array)

```

در این مثال، `data` یک شیء قابل تبدیل به رشته است که می‌تواند لیست رشته‌ها، آرایه رشته‌ها و غیره باشد. با استفاده از `np.core.defchararray.array()`، `data` به یک آرایه حرفی تبدیل می‌شود و در متغیر `string\_array` ذخیره می‌شود.

خروجی کد فوق در مثال فرضی به صورت زیر خواهد بود:

```

['apple' 'banana' 'cherry']

```

در اینجا، یک آرایه حرفی سه عنصری ساخته شده است. هر عنصر در آرایه یک رشته است که به ترتیب 'apple'، 'banana' و 'cherry' است.

50-تابع `numpy.core.defchararray.asarray(obj,itemsize,order)` برای تبدیل یک شیء قابل تبدیل به رشته به آرایه حرفی (string array) استفاده می‌شود. این تابع آرگومان‌های زیر را می‌پذیرد:

- `obj`: شیء قابل تبدیل به رشته که برای تبدیل به آرایه استفاده می‌شود.

- `itemsize`: اندازه هر عنصر آرایه حرفی. این پارامتر می‌تواند مقدار `None` داشته باشد، در این صورت اندازه هر عنصر بر اساس طول بزرگترین رشته در `obj` تعیین می‌شود.

- `order`: ترتیب ذخیره سازی آرایه. مقادیر ممکن برای این پارامتر `'C'` (ذخیره سازی در حالت سطری، پیش‌فرض) یا `'F'` (ذخیره سازی در حالت ستونی) هستند.

در ادامه یک نمونه کد برای استفاده از تابع `numpy.core.defchararray.asarray()` آمده است:

```python

import numpy as np

# شیء قابل تبدیل به رشته

data = ['apple', 'banana', 'cherry']

# تبدیل به آرایه حرفی

string\_array = np.core.defchararray.asarray(data)

# چاپ آرایه حرفی

print(string\_array)

```

در این مثال، `data` یک شیء قابل تبدیل به رشته است که می‌تواند لیست رشته‌ها، آرایه رشته‌ها و غیره باشد. با استفاده از `np.core.defchararray.asarray()`، `data` به یک آرایه حرفی تبدیل می‌شود و در متغیر `string\_array` ذخیره می‌شود. سپس آرایه حرفی چاپ می‌شود.

خروجی کد فوق در مثال فرضی به صورت زیر خواهد بود:

```

['apple' 'banana' 'cherry']

```

51-تابع `numpy.arange(start,stop,step,dtype,like)` برای ایجاد یک آرایه یک بعدی با اعداد پیوسته استفاده می‌شود. آرگومان‌های این تابع عبارتند از:

- `start` (اختیاری): مقدار شروع دنباله. پیش‌فرض آن 0 است.

- `stop`: مقدار پایان دنباله (قبل از آن مقداری نخواهد بود). این مقدار وارد شده به عنوان آخرین عنصر آرایه خواهد بود.

- `step` (اختیاری): بینهایت کوچکترین فاصله بین دو عنصر متوالی در آرایه. پیش‌فرض آن 1 است.

- `dtype` (اختیاری): نوع داده‌ای آرایه. این پارامتر به صورت خودکار براساس مقادیر `start`، `stop` و `step` تعیین می‌شود. به طور پیش‌فرض، از نوع داده integer استفاده می‌شود.

- `like` (اختیاری): آرایه مرجع برای تعیین نوع داده و ساختار آرایه جدید. این پارامتر به صورت خودکار مقدار دهی نمی‌شود و در صورت وارد نشدن، از پارامتر `dtype` استفاده می‌شود.

در ادامه یک نمونه کد برای استفاده از تابع `numpy.arange()` آمده است:

```python

import numpy as np

# ساخت آرایه اعداد صحیح از 0 تا 9

arr = np.arange(10)

# چاپ آرایه

print(arr)

```

در این مثال، با استفاده از `np.arange(10)`، یک آرایه اعداد صحیح از 0 تا 9 ساخته می‌شود و در متغیر `arr` ذخیره می‌شود. سپس آرایه چاپ می‌شود.

خروجی کد فوق در مثال فرضی به صورت زیر خواهد بود:

```

[0 1 2 3 4 5 6 7 8 9]

```

در اینجا، آرایه اعداد صحیح 0 تا 9 ساخته شده است.

52-تابع `numpy.linspace(start,stop,num,endpoint,retstep,dtype,axis)` برای ایجاد یک آرایه یک بعدی با اعداد پیوسته با فاصله یکسان بین دو مقدار `start` و `stop` استفاده می‌شود. آرگومان‌های این تابع عبارتند از:

- `start`: مقدار شروع دنباله.

- `stop`: مقدار پایان دنباله.

- `num` (اختیاری): تعداد عناصر مورد نظر در آرایه. پیش‌فرض آن 50 است.

- `endpoint` (اختیاری): نشان دهنده این است که آیا مقدار `stop` نیز در آرایه باید حضور داشته باشد یا خیر. پیش‌فرض آن True است.

- `retstep` (اختیاری): نشان دهنده این است که آیا قدم (فاصله بین دو عنصر) نیز باید در خروجی برگردانده شود یا خیر. پیش‌فرض آن False است.

- `dtype` (اختیاری): نوع داده‌ای آرایه. این پارامتر به صورت خودکار براساس نوع داده `start` و `stop` تعیین می‌شود.

- `axis` (اختیاری): محور در نظر گرفته شده برای ساخت آرایه. پیش‌فرض آن 0 است.

در ادامه یک نمونه کد برای استفاده از تابع `numpy.linspace()` آمده است:

```python

import numpy as np

# ساخت آرایه اعداد پیوسته بین 1 و 10 با 5 عنصر

arr = np.linspace(1, 10, num=5)

# چاپ آرایه

print(arr)

```

در این مثال، با استفاده از `np.linspace(1, 10, num=5)`، یک آرایه با 5 عنصر از 1 تا 10 (شامل اعداد 1 و 10) ساخته می‌شود و در متغیر `arr` ذخیره می‌شود. سپس آرایه چاپ می‌شود.

خروجی کد فوق در مثال فرضی به صورت زیر خواهد بود:

```

[ 1. 3.25 5.5 7.75 10. ]

```

. ادامه کد نمونه برای استفاده از تابع `numpy.linspace()` با پارامترهای دیگر نیز به شرح زیر است:

```python

import numpy as np

# ساخت آرایه اعداد پیوسته بین -1 و 1 با 10 عنصر و بدون انتها

arr = np.linspace(-1, 1, num=10, endpoint=False)

# چاپ آرایه

print(arr)

```

در این مثال، تابع `np.linspace(-1, 1, num=10, endpoint=False)` برای ساختن آرایه‌ای با 10 عنصر اعداد پیوسته بین -1 و 1 (بدون شامل شدن عدد 1) استفاده می‌شود. نتیجه در متغیر `arr` ذخیره شده و چاپ می‌شود.

خروجی کد فوق در مثال فرضی به صورت زیر خواهد بود:

```

[-1. -0.8 -0.6 -0.4 -0.2 0. 0.2 0.4 0.6 0.8]

```

در این مثال، آرایه اعداد پیوسته بین -1 و 1 (بدون شامل شدن عدد 1) با فاصله یکسان بین عناصر ساخته شده است.

53-تابع `numpy.logspace(start,stop,num,endpoint,base,dtype,axis)` برای ساختن یک آرایه از اعداد با پیشرفت لگاریتمی استفاده می‌شود. آرگومان‌های این تابع به شرح زیر هستند:

- `start`: مقدار شروع لگاریتمی (به صورت اعداد حقیقی یا صحیح قابل قبول است).

- `stop`: مقدار پایان لگاریتمی (به صورت اعداد حقیقی یا صحیح قابل قبول است).

- `num`: تعداد عناصر در آرایه خروجی (به صورت عدد صحیح است).

- `endpoint`: مشخص می‌کند آیا مقدار پایانی (stop) در آرایه خروجی شامل شود یا خیر (به صورت مقدار بولین است).

- `base`: پایه لگاریتم (به صورت اعداد حقیقی یا صحیح قابل قبول است).

- `dtype`: نوع داده خروجی (اختیاری، به صورت `None` به خودی خود نوع داده‌ها تعیین می‌شود).

- `axis`: محور مورد نظر برای ساخت آرایه (اختیاری، به صورت عدد صحیح مثبت مشخص می‌شود).

نمونه کد زیر نحوه استفاده از تابع `numpy.logspace()` را نشان می‌دهد:

```python

import numpy as np

# ساخت آرایه اعداد با پیشرفت لگاریتمی بین 1 تا 1000 با 10 عنصر

arr = np.logspace(1, 3, num=10)

# چاپ آرایه

print(arr)

```

در این مثال، تابع `np.logspace(1, 3, num=10)` برای ساختن یک آرایه اعداد با پیشرفت لگاریتمی بین 1 تا 1000 (شامل هر دو مقدار) با 10 عنصر استفاده شده است. نتیجه در متغیر `arr` ذخیره شده و چاپ می‌شود.

خروجی کد فوق در مثال فرضی به صورت زیر خواهد بود:

```

[ 10. 21.5443469 46.41588834 100. 215.443469 464.15888336 1000. 2154.434

54-تابع `numpy.geomspace(start,stop,num,endpoint,dtype,axis)` برای ساختن یک آرایه با پیشرفت هندسی استفاده می‌شود. آرگومان‌های این تابع به شرح زیر هستند:

- `start`: مقدار شروع هندسی (باید عددی مثبت باشد).

- `stop`: مقدار پایان هندسی (باید عددی مثبت باشد).

- `num`: تعداد عناصر در آرایه خروجی (به صورت عدد صحیح است).

- `endpoint`: مشخص می‌کند آیا مقدار پایانی (stop) در آرایه خروجی شامل شود یا خیر (به صورت مقدار بولین است).

- `dtype`: نوع داده خروجی (اختیاری، به صورت `None` به خودی خود نوع داده‌ها تعیین می‌شود).

- `axis`: محور مورد نظر برای ساخت آرایه (اختیاری، به صورت عدد صحیح مثبت مشخص می‌شود).

نمونه کد زیر نحوه استفاده از تابع `numpy.geomspace()` را نشان می‌دهد:

```python

import numpy as np

# ساخت آرایه با پیشرفت هندسی بین 1 تا 1000 با 10 عنصر

arr = np.geomspace(1, 1000, num=10)

# چاپ آرایه

print(arr)

```

در این مثال، تابع `np.geomspace(1, 1000, num=10)` برای ساختن یک آرایه با پیشرفت هندسی بین 1 تا 1000 (شامل هر دو مقدار) با 10 عنصر استفاده شده است. نتیجه در متغیر `arr` ذخیره شده و چاپ می‌شود.

خروجی کد فوق در مثال فرضی به صورت زیر خواهد بود:

```

[ 1. 2.15443469 4.64158883 10. 21.5443469 46.41588834 100. 215.443469 464.15888336 1000. ]

```

در این مثال، آرایه خروجی شامل 10 عنصر با پیشرفت هندسی بین 1 تا 1000 است.

55-تابع `numpy.meshgrid(\*xi,copy,sparse,indexing)` برای ساختن آرایه‌های مشترک از مبدلهایی استفاده می‌شود که مبدلهای مشترک به همراه آرایه‌های مبدلهای ورودی را برمی‌گرداند.

وقتی شما مبدلهای ورودی را به عنوان آرگومان‌ها به `numpy.meshgrid()` می‌دهید، این تابع آرایه‌های مشترک را برمی‌گرداند

ادامه توضیحات در مورد آرگومان‌های `numpy.meshgrid()`:

- `\*xi`: آرگومان‌های مبدلها. مبدلها می‌توانند به صورت هر تعداد و به هر شکل باشند. شما می‌توانید مبدلهای مورد نیاز خود را به عنوان آرگومان‌ها به `numpy.meshgrid()` ارسال کنید.

- `copy` (به صورت پیش‌فرض True): اگر مقدار `copy` برابر با True باشد، آرایه‌های مبدلهای خروجی به طور مستقل از آرایه‌های ورودی ایجاد می‌شوند. در غیر این صورت، آرایه‌های خروجی به آرایه‌های ورودی ارجاع می‌دهند.

- `sparse` (به صورت پیش‌فرض False): اگر `sparse` برابر با True باشد، آرایه‌های مبدلهای خروجی به صورت یک ماتریس مفصل از مبدلها ایجاد می‌شوند. در غیر این صورت، آرایه‌های خروجی به صورت ماتریس پر از مبدلها ایجاد می‌شوند.

- `indexing` (به صورت پیش‌فرض 'xy'): نحوه ایندکس‌گذاری آرایه‌های مبدلها. مقدار 'xy' به معنی استفاده از ایندکس‌گذاری کارتزی است که مبدلها بر اساس مختصات x و y مرتب می‌شوند. مقدار 'ij' به معنی استفاده از ایندکس‌گذاری ماتریسی است که مبدلها بر اساس مختصات i و j مرتب می‌شوند.

خروجی تابع `numpy.meshgrid()` شامل آرایه‌های مبدلهایی است که بر اساس مبدلهای ورودی ایجاد شده‌اند. این آرایه‌ها با استفاده از مبدلها محورهای جدیدی را ایجاد می‌کنند و ابعاد مشترکی را دارند که به عنوان خروجی برای محاسبات بیشتر استفاده می‌شوند.

ادامه توضیحات در مورد `numpy.meshgrid()`:

تابع `numpy.meshgrid()` برای ایجاد آرایه‌های مبدلها در دو بعد (یا بیشتر) مورد استفاده قرار می‌گیرد. به طور کلی، اگر تعداد مبدلها در ورودی برابر با N باشد، آرایه‌های مبدلهای خروجی ابعادی به اندازه N خواهند داشت، به طوری که هر آرایه مبدله با استفاده از مبدل مرتبط به یکی از محورها پر شده است.

مبدلهای ورودی می‌توانند به عنوان آرگومان‌های جداگانه به `numpy.meshgrid()` ارسال شوند یا به صورت یک لیست از آرایه‌ها با استفاده از عملگر '\*' به آن ارسال شوند. در هر صورت، آرایه‌های مبدلها در ترتیب ورودی ایجاد می‌شوند.

با استفاده از `numpy.meshgrid()`، می‌توانید یک شبکه از نقاط را در فضای مبدلها ایجاد کنید. این برای محاسباتی که نیاز به مبدلها برای محورهای مختلف دارند، مانند نمودارسازی 3D، تجسم داده‌ها در فضای سه بعدی و غیره، بسیار مفید است.

در ادامه مثالی از استفاده از `numpy.meshgrid()` را نشان می‌دهم:

``` python

import numpy as np

x = np.linspace(0, 1, 5)

y = np.linspace(0, 1, 3)

X, Y = np.meshgrid(x, y)

print(X)

print(Y)

```

Output:

```

[[0. 0.25 0.5 0.75 1. ]

[0. 0.25 0.5 0.75 1. ]

[0. 0.25 0.5 0.75 1. ]]

[[0. 0. 0. 0. 0.]

[0.5 0.5 0.5 0.5 0.5]

[1. 1. 1. 1. 1.]]

```

`numpy.mgrid` -56یک شیء از کلاس `numpy.lib.index\_tricks.MGridClass` است که برای ایجاد آرایه‌های مبدلها (مشابه `numpy.meshgrid()`) با استفاده از نحوه نوشتن مختصات مبدلها به صورت فشرده استفاده می‌شود.

با استفاده از `numpy.mgrid` می‌توانید آرایه‌های مبدلها را با استفاده از تعدادی خواص خاص ایجاد کنید. این خواص شامل استفاده از نقاط ابتدا و انتها به عنوان مرزهای تعیین شده، تعداد نقاط در هر مبدل، مقدار گام بین نقاط و ... است.

در ادامه یک مثال از استفاده از `numpy.mgrid` را نشان می‌دهم:

``` python

import numpy as np

x, y = np.mgrid[0:1:0.25, 0:1:0.5]

print(x)

print(y)

```

Output:

```

[[0. 0. 0. 0. ]

[0.25 0.25 0.25 0.25]

[0.5 0.5 0.5 0.5 ]

[0.75 0.75 0.75 0.75]

[1. 1. 1. 1. ]]

[[0. 0.5 1. ]

[0. 0.5 1. ]

[0. 0.5 1. ]

[0. 0.5 1. ]

[0. 0.5 1. ]]

```

در این مثال، با استفاده از `numpy.mgrid`، دو آرایه `x` و `y` ایجاد شده است که نشان دهنده مبدلهای مختلف است. از نظر عملکردی، `numpy.mgrid[0:1:0.25, 0:1:0.5]` معادل با استفاده از `numpy.meshgrid` با استفاده از `numpy.linspace` است.

-57در NumPy، ما به عنوان یکی دیگر از ابزارهای آرایه مبدل، `numpy.ogrid` را داریم. `numpy.ogrid` مشابه `numpy.mgrid` است، با این تفاوت که به جای ایجاد آرایه‌های کامل، مبدلهای کامل را بازگردانی می‌کند.

`numpy.ogrid` به صورت معمول در دو بعد استفاده می‌شود، یک بعد برای محور عمودی و بعد دیگر برای محور افقی. همچنین، مقادیر در هر مبدل به صورت پیوسته توصیف می‌شوند.

به عنوان مثال، با استفاده از `numpy.ogrid` می‌توانید آرایه‌هایی را بسازید که مبدلهای کامل در آن قرار دارند. به طور معمول، برای دسترسی به المان‌ها از عملگر ایندکسینگ استفاده می‌شود.

در زیر یک نمونه کد با استفاده از `numpy.ogrid` را مشاهده می‌کنید:

``` python

import numpy as np

x, y = np.ogrid[0:5, 0:3]

print(x)

print(y)

print(x.shape) # (5, 1)

print(y.shape) # (1, 3)

```

خروجی:

```

[[0]

[1]

[2]

[3]

[4]]

[[0 1 2]]

(5, 1)

(1, 3)

```

در این مثال، `x` مبدل عمودی و `y` مبدل افقی است. با استفاده از این مبدلها، می‌توانید به صورت سهولت بیشتری با المان‌های مبدل تعامل کنید.

58-تابع `numpy.diag(v,k)` در NumPy برای ساخت یک آرایه قطری یا استخراج قطر آرایه از ورودی استفاده می‌شود.

ورودی `v` می‌تواند یک آرایه یک بعدی باشد که المان‌های آن بر روی قطر آرایه قرار می‌گیرند، یا می‌تواند یک عدد صحیح باشد که برای ساخت آرایه قطری استفاده می‌شود.

پارامتر `k` نیز مشخص می‌کند که المان‌ها بر روی قطر اصلی (`k=0`) یا قطر‌های فرعی (`k>0` یا `k<0`) قرار بگیرند. با افزایش `k`، قطر آرایه به سمت بالا متحرک می‌شود و با کاهش `k`، قطر آرایه به سمت پایین متحرک می‌شود.

در زیر چند نمونه کد با استفاده از `numpy.diag` را مشاهده می‌کنید:

``` python

import numpy as np

# ساخت یک آرایه قطری از آرایه ورودی

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

result = np.diag(a)

print(result)

# خروجی [[1 0 0]

# [0 2 0]

# [0 0 3]

# استخراج قطر از یک آرایه

b = np.array([[1, 2, 3],

[4, 5, 6],

[7, 8, 9]])

result = np.diag(b)

print(result)

# خروجی: [1 5 9]

# ساخت یک آرایه قطری با عدد ورودی

result = np.diag(4, k=1)

print(result)

# خروجی: [[0 4 0 0]

# [0 0 4 0]

# [0 0 0 4]

# [0 0 0 0]]

```

59-تابع `numpy.diagflat(v,k)` در NumPy برای ساخت یک آرایه قطری با المان‌های ورودی `v` استفاده می‌شود.

ورودی `v` می‌تواند یک آرایه یک بعدی باشد که المان‌های آن بر روی قطر آرایه قرار می‌گیرند، یا می‌تواند یک عدد صحیح باشد که برای ساخت آرایه قطری استفاده می‌شود.

پارامتر `k` نیز مشخص می‌کند که المان‌ها بر روی قطر اصلی (`k=0`) یا قطر‌های فرعی (`k>0` یا `k<0`) قرار بگیرند. با افزایش `k`، قطر آرایه به سمت بالا متحرک می‌شود و با کاهش `k`، قطر آرایه به سمت پایین متحرک می‌شود.

در زیر چند نمونه کد با استفاده از `numpy.diagflat` را مشاهده می‌کنید:

``` python

import numpy as np

# ساخت یک آرایه قطری با المان‌های ورودی

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

result = np.diagflat(a)

print(result)

# خروجی: [[1 0 0]

# [0 2 0]

# [0 0 3]]

# ساخت یک آرایه قطری با عدد ورودی

result = np.diagflat(4, k=1)

print(result)

# خروجی: [[0 4 0 0]

# [0 0 4 0]

# [0 0 0 4]

# [0 0 0 0]]

```

در نمونه‌های بالا، تابع `numpy.diagflat` در حالت اول آرایه قطری با المان‌های آرایه `a` را ساخته است. در حالت دوم، یک آرایه قطری با عدد `4` ساخته می‌شود که المان‌های آن بر روی قطر اصلی و قطر فرعی قرار می‌گیرند.

دو تابع `numpy.diag` و `numpy.diagflat` هر دو برای ساخت آرایه‌های قطری در NumPy استفاده می‌شوند، اما تفاوتی در نحوه ورودی و استفاده از آنها وجود دارد.

1. `numpy.diag(v, k=0)`:

- `v` می‌تواند یک آرایه یک بعدی باشد که المان‌های آن بر روی قطر آرایه قرار می‌گیرند، یا می‌تواند یک عدد صحیح باشد که برای ساخت آرایه قطری استفاده می‌شود.

- پارامتر `k` مشخص می‌کند که المان‌ها بر روی قطر اصلی (`k=0`) یا قطر‌های فرعی (`k>0` یا `k<0`) قرار بگیرند. با افزایش `k`، قطر آرایه به سمت بالا متحرک می‌شود و با کاهش `k`، قطر آرایه به سمت پایین متحرک می‌شود.

2. `numpy.diagflat(v, k=0)`:

- `v` می‌تواند یک آرایه یک بعدی باشد که المان‌های آن بر روی قطر آرایه قرار می‌گیرند، یا می‌تواند یک عدد صحیح باشد که برای ساخت آرایه قطری استفاده می‌شود.

- پارامتر `k` نیز مشخص می‌کند که المان‌ها بر روی قطر اصلی (`k=0`) یا قطر‌های فرعی (`k>0` یا `k<0`) قرار بگیرند. با افزایش `k`، قطر آرایه به سمت بالا متحرک می‌شود و با کاهش `k`، قطر آرایه به سمت پایین متحرک می‌شود.

- با این تفاوت که تابع `numpy.diagflat` همواره یک آرایه دو بعدی را برمی‌گرداند، حتی اگر ورودی `v` یک عدد صحیح باشد. المان‌ها در این آرایه بر روی قطر اصلی قرار می‌گیرند .

60-تابع `numpy.tri(N,M,k,dtype)` در NumPy برای ساختن یک آرایه مثلثی (triangle) با مقادیر یکی یا صفر استفاده می‌شود. در ادامه یک نمونه کد با استفاده از تابع `numpy.tri` را ارائه می‌دهم و خروجی آن را نیز نشان می‌دهم:

```python

import numpy as np

N = 5 # تعداد ردیف‌ها

M = 5 # تعداد ستون‌ها

arr = np.tri(N, M, k=0, dtype=float)

print(arr)

```

خروجی:

```

[[1. 0. 0. 0. 0.]

[1. 1. 0. 0. 0.]

[1. 1. 1. 0. 0.]

[1. 1. 1. 1. 0.]

[1. 1. 1. 1. 1.]]

```

در این مثال، با استفاده از تابع `numpy.tri`، آرایه‌ای مثلثی با ابعاد `N` و `M` ساخته می‌شود، که تمام مقادیر بالای قطر اصلی (به جز قطر اصلی) برابر صفر است و مقادیر قطر اصلی برابر یک هستند. خروجی نهایی نشان داده شده است.

تابع `numpy.tri` دارای تعدادی آرگومان است. در ادامه توضیح مختصری از هر آرگومان آمده است:

- `N`: تعداد ردیف‌های آرایه مثلثی.

- `M` (اختیاری): تعداد ستون‌های آرایه مثلثی. اگر مقدار این آرگومان تعیین نشود، به طور پیش‌فرض برابر با `N` است و یک آرایه مربعی ساخته می‌شود.

- `k` (اختیاری): انتقال (shift) مقادیر مثلث در آرایه. مقدار صفر به معنی مثلث اصلی است و مقادیر مثبت برای انتقال به بالا و مقادیر منفی برای انتقال به پایین است.

- `dtype` (اختیاری): نوع داده مقادیر آرایه مثلثی. این آرگومان به طور پیش‌فرض برابر با `float` است.

- `like` (اختیاری): آرایه مبنا برای تعیین نوع داده و ساختار آرایه مثلثی. این آرگومان اجازه می‌دهد تا آرایه مثلثی ساخته شده همان نوع داده و ساختار آرایه مبنا را داشته باشد.

در نمونه کدی که قبلاً ارائه دادم، `N` و `M` به ترتیب برابر با 5 قرار داده شده است. سایر آرگومان‌ها با مقادیر پیش‌فرض استفاده شده‌اند.

61-تابع `numpy.tril(m,k)` یک آرایه ماتریسی را به آرایه مثلثی پایین تبدیل می‌کند. مشخصات آرگومان‌های این تابع عبارتند از:

- `m`: آرایه ماتریسی ورودی که به آرایه مثلثی تبدیل می‌شود.

- `k` (اختیاری): پارامتر انتقال (shift) مقادیر مثلث در آرایه خروجی. با تعیین مقدار `k` می‌توانید مقادیر مثلث را به بالا یا پایین انتقال دهید. مقدار پیش‌فرض برابر با 0 است که به معنای مثلث اصلی است.

در نمونه کد زیر، یک آرایه ماتریسی 3x3 تعریف شده است. سپس تابع `numpy.tril` بر روی آرایه اعمال شده است:

```python

import numpy as np

m = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

result = np.tril(m)

print(result)

```

خروجی به صورت زیر خواهد بود:

```

[[1 0 0]

[4 5 0]

[7 8 9]]

```

همانطور که مشاهده می‌شود، تابع `numpy.tril` مقادیر مثلث پایین آرایه ماتریسی را نگه می‌دارد و مابقی مقادیر را به صفر تبدیل می‌کند.

62-تابع `numpy.triu(m,k)` یک آرایه ماتریسی را به آرایه مثلثی بالا تبدیل می‌کند. مشخصات آرگومان‌های این تابع عبارتند از:

- `m`: آرایه ماتریسی ورودی که به آرایه مثلثی تبدیل می‌شود.

- `k` (اختیاری): پارامتر انتقال (shift) مقادیر مثلث در آرایه خروجی. با تعیین مقدار `k` می‌توانید مقادیر مثلث را به بالا یا پایین انتقال دهید. مقدار پیش‌فرض برابر با 0 است که به معنای مثلث اصلی است.

در نمونه کد زیر، یک آرایه ماتریسی 3x3 تعریف شده است. سپس تابع `numpy.triu` بر روی آرایه اعمال شده است:

```python

import numpy as np

m = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

result = np.triu(m)

print(result)

```

خروجی به صورت زیر خواهد بود:

```

[[1 2 3]

[0 5 6]

[0 0 9]]

```

همانطور که مشاهده می‌شود، تابع `numpy.triu` مقادیر مثلث بالای آرایه ماتریسی را نگه می‌دارد و مابقی مقادیر را به صفر تبدیل می‌کند.

63-تابع `numpy.vander(x,n,increasing)` یک آرایه وکتور را به آرایه واندرموند (Vandermonde) تبدیل می‌کند. واندرموند یک ماتریس مربعی است که در آن هر ستون توان‌های مختلف یک وکتور ورودی را نشان می‌دهد. مشخصات آرگومان‌های این تابع عبارتند از:

- `x`: وکتور ورودی.

- `N` (اختیاری): ابعاد ماتریس واندرموند خروجی. اگر مقدار `N` تعیین نشود، ماتریس خروجی به اندازه وکتور ورودی ایجاد می‌شود.

- `increasing` (اختیاری): مشخص می‌کند که آیا مقادیر توان‌ها در هر ستون باید افزایشی باشند یا خیر. مقدار پیش‌فرض `False` است که به معنای توان‌های نزولی است.

در نمونه کد زیر، یک وکتور ورودی به عنوان آرگومان `x` تعریف شده است. سپس تابع `numpy.vander` بر روی این وکتور اعمال شده است:

```python

import numpy as np

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

result = np.vander(x)

print(result)

```

خروجی به صورت زیر خواهد بود:

```

[[ 1 1 1 1]

[ 8 4 2 1]

[27 9 3 1]

[64 16 4 1]]

```

همانطور که مشاهده می‌شود، هر ستون ماتریس خروجی شامل توان‌های مختلف وکتور ورودی است. ستون اول توان ۰، ستون دوم توان ۱، ستون سوم توان ۲ و ستون چهارم توان ۳ را نشان می‌دهد.

64-تابع `numpy.mat(data,dtype)` یک آرایه را به ماتریس تبدیل می‌کند. این تابع با یک آرگومان اصلی `data` و یک آرگومان اختیاری `dtype` فراخوانی می‌شود.

- `data`: آرایه ورودی که به ماتریس تبدیل می‌شود. می‌تواند نوع داده‌های گوناگون و شکل‌های مختلفی داشته باشد.

- `dtype` (اختیاری): نوع داده ماتریس خروجی. اگر مقدار `dtype` مشخص نشود، نوع داده به طور خودکار بر اساس داده‌های ورودی تعیین می‌شود.

تابع `numpy.mat` ورودی را به یک نوع داده ماتریسی تبدیل می‌کند. در نتیجه، آرایه ورودی به یک شیء `matrix` تبدیل می‌شود که قابلیت‌های مختلفی برای عملیات ماتریسی فراهم می‌کند.

نمونه کد زیر نحوه استفاده از تابع `numpy.mat` را نشان می‌دهد:

```python

import numpy as np

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

matrix = np.mat(data)

print(matrix)

```

خروجی به صورت زیر خواهد بود:

```

[[1 2]

[3 4]]

```

همانطور که مشاهده می‌شود، آرایه ورودی به یک ماتریس تبدیل شده است. شیء ماتریس حاوی داده‌ها و قابلیت‌های مختلف ماتریسی است که می‌توان از آن برای انجام عملیات ریاضی بر روی ماتریس استفاده کرد.

65-تابع `numpy.bmat(obj)` برای ساختن یک ماتریس بلوکی از ماتریس‌ها استفاده می‌شود.

- `obj`: یک رشته یا یک توالی از ماتریس‌ها که برای ساختن ماتریس بلوکی استفاده می‌شود.

تابع `numpy.bmat` ماتریس بلوکی را با توجه به رشته `obj` و متغیرهای محلی و سراسری مشخص شده، ساخته و برمیگرداند.

نمونه کد زیر نحوه استفاده از تابع `numpy.bmat` را نشان می‌دهد:

```python

import numpy as np

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

B = np.array([[5], [6]])

C = np.array([[7, 8]])

result = np.bmat([[A, B], C])

print(result)

```

خروجی به صورت زیر خواهد بود:

```

[[1 2 5]

[3 4 6]

[7 8 0]]

```

در این مثال، ما سه ماتریس A، B و C را تعریف کرده و با استفاده از تابع `numpy.bmat` آن‌ها را به صورت ماتریس بلوکی به هم متصل کرده‌ایم. ماتریس B به سمت راست ماتریس A آمده و ماتریس C در سطر بعدی قرار دارد.

66-تابع `numpy.copyto(dst,,src,casting,where)` برای کپی کردن محتویات یک آرایه (منبع) به آرایه دیگر (مقصد) در NumPy استفاده می‌شود. این تابع با سه آرگومان اصلی `dst`، `src` و `casting` فراخوانی می‌شود و امکان استفاده از آرگومان اختیاری `where` نیز وجود دارد.

- `dst`: آرایه مقصد که محتویات منبع در آن کپی می‌شود.

- `src`: آرایه منبع که محتویات آن به آرایه مقصد کپی می‌شود.

- `casting` (اختیاری): نوع تبدیلی که برای کپی انجام می‌شود. مقادیر ممکن برای این آرگومان عبارتند از `'no'`، `'equiv'`، `'safe'` و `'same\_kind'`.

- `where` (اختیاری): شرطی که تعیین می‌کند کدام عناصر از منبع باید به مقصد کپی شوند. این آرگومان می‌تواند با یک آرایه بولی یا یک شرط قابل ارزیابی تعیین شود.

نمونه کد زیر نحوه استفاده از تابع `numpy.copyto` را نشان می‌دهد:

```python

import numpy as np

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

dst = np.zeros(3)

np.copyto(dst, src)

print(dst)

```

خروجی به صورت زیر خواهد بود:

```

[1. 2. 3.]

```

در این مثال، ما یک آرایه منبع `src` با مقادیر `[1, 2, 3]` و یک آرایه مقصد `dst` که همان `np.zeros(3)` است، تعریف کرده‌ایم. با استفاده از تابع `numpy.copyto`، محتویات آرایه منبع به آرایه مقصد کپی شده و خروجی نهایی در `dst` قرار می‌گیرد.

67-تابع `numpy.shape(arr)` برای دریافت ابعاد یک آرایه در NumPy استفاده می‌شود. این تابع بر روی یک آرایه فراخوانی می‌شود و توپلی از اعداد که نشان‌دهنده ابعاد آرایه است را برمی‌گرداند.

مثال زیر نحوه استفاده از تابع `numpy.shape` را نشان می‌دهد:

```python

import numpy as np

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

shape = np.shape(arr)

print(shape)

```

خروجی به صورت زیر خواهد بود:

```

(2, 3)

```

در این مثال، ما یک آرایه دوبعدی `arr` تعریف کرده‌ایم. با استفاده از تابع `numpy.shape`، ابعاد آرایه را دریافت کرده و در متغیر `shape` قرار می‌دهیم. در اینجا آرایه `arr` دارای ابعاد 2 در 3 است، بنابراین خروجی توپل `(2, 3)` خواهد بود.

68-تابع `numpy.reshape(a, newshape, order='C')` برای تغییر شکل یک آرایه به شکل موردنظر استفاده می‌شود.

- `a`: آرایه اصلی که می‌خواهید شکل آن را تغییر دهید.

- `newshape`: شکل جدید موردنظر برای آرایه. می‌تواند به صورت یک tuple یا یک عدد صحیح باشد.

- `order` (اختیاری): نحوه مرتب‌سازی داده‌ها در حافظه. مقادیر ممکن برای این آرگومان عبارت‌اند از 'C' (سطر به سطر)، 'F' (ستون به ستون) و 'A' (به‌ترتیب محلی).

تابع `reshape` یک نمونه از آرایه اصلی با شکل جدید برمی‌گرداند، اما در صورت امکان از همان داده‌های آرایه اصلی استفاده می‌کند و کپی جدیدی از داده‌ها را ایجاد نمی‌کند.

تابع `numpy.reshape` برای تغییر شکل یک آرایه استفاده می‌شود. آرگومان `a` آرایه اصلی است و `newshape` یک tuple یا یک عدد صحیح است که ابعاد جدید آرایه را مشخص می‌کند. آرگومان اختیاری `order` نیز نحوه مرتب‌سازی داده‌ها در حافظه را مشخص می‌کند. مقادیر ممکن برای `order` شامل 'C' (مرتب‌سازی ردیفی، به طور پیش‌فرض) و 'F' (مرتب‌سازی ستونی) هستند.

این تابع یک آرایه با شکل جدید برمی‌گرداند که از داده‌های آرایه اصلی استفاده می‌کند. اگر شکل جدید غیرمعتبر باشد (به طور مثال، اگر تعداد عناصر آرایه اصلی و شکل جدید با هم مطابقت نداشته باشد)، یک خطا ایجاد می‌شود.

در ادامه یک نمونه کد با استفاده از تابع `numpy.reshape` را مشاهده می‌کنید:

```python

import numpy as np

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

b = np.reshape(a, (2, 3)) # تغییر شکل آرایه a به شکل (2, 3)

print(b)

```

خروجی:

```

[[1 2 3]

[4 5 6]]

```

در این مثال، آرایه `a` با شکل اولیه (1, 6) داریم. با استفاده از `numpy.reshape`، آرایه `a` را به شکل (2, 3) تغییر دادیم و نتیجه را در آرایه `b` ذخیره کردیم. خروجی نشان می‌دهد که داده‌های آرایه `a` به شکل جدید با مرتب‌سازی ردیفی در آرایه `b` قرار گرفته‌اند.

69-تابع `numpy.ravel(a,order=C)` برای تبدیل یک آرایه چندبعدی به یک آرایه یک بعدی (یک بردار) استفاده می‌شود. آرگومان `a` آرایه اصلی است و آرگومان اختیاری `order` نیز نحوه مرتب‌سازی داده‌ها در آرایه یک بعدی را مشخص می‌کند. مقادیر ممکن برای `order` شامل 'C' (مرتب‌سازی ردیفی، به طور پیش‌فرض) و 'F' (مرتب‌سازی ستونی) هستند.

این تابع یک آرایه یک بعدی را با استفاده از داده‌های آرایه اصلی برمی‌گرداند. توجه داشته باشید که این تابع فقط شکل آرایه را تغییر نمی‌دهد و فقط داده‌ها را باقی می‌گذارد.

در ادامه یک نمونه کد با استفاده از تابع `numpy.ravel` را مشاهده می‌کنید:

```python

import numpy as np

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

b = np.ravel(a) # تبدیل آرایه a به آرایه یک بعدی

print(b)

```

خروجی:

```

[1 2 3 4 5 6]

```

در این مثال، آرایه `a` با شکل (2, 3) داریم. با استفاده از `numpy.ravel`، آرایه `a` را به یک آرایه یک بعدی تبدیل کردیم و نتیجه را در آرایه `b` ذخیره کردیم. خروجی نشان می‌دهد که داده‌های آرایه `a` در آرایه یک بعدی `b` قرار گرفته‌اند.

70-ویژگی `.flat()` یک ویژگی مربوط به آرایه‌ها در NumPy است. زمانی که از ویژگی `.flat` بر روی یک آرایه فراخوانی شود، یک ایتریتور یا یک تکرارکننده (iterator) برای دسترسی به همه عناصر آرایه بصورت متوالی (به صورت یک بعدی) ایجاد می‌شود.

به طور کلی، می‌توانید بر روی یک آرایه با ویژگی `.flat` حلقه `for` بگذارید و به صورت ترتیبی به همه عناصر آرایه دسترسی پیدا کنید. این ویژگی مفید است زمانی که شما به صورت ترتیبی عناصر آرایه را مورد استفاده قرار می‌دهید یا زمانی که نیاز دارید به صورت یک بعدی از عناصر آرایه استفاده کنید.

در ادامه یک نمونه کد برای استفاده از ویژگی `.flat` را مشاهده می‌کنید:

```python

import numpy as np

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

# دسترسی به عناصر آرایه به صورت ترتیبی با استفاده از ویژگی .flat

for element in a.flat:

print(element)

```

خروجی:

```

1

2

3

4

5

6

```

در این مثال، با استفاده از ویژگی `.flat`، به ترتیب به همه عناصر آرایه `a` دسترسی پیدا می‌کنیم و آنها را در حلقه `for` چاپ می‌کنیم.

`ndarray.flatten(order='C')` -71یک تابع است که بر روی یک آرایه NumPy فراخوانی می‌شود. این تابع آرایه چند بعدی را به یک آرایه یک بعدی تغییر شکل می‌دهد.

آرگومان `order` در این تابع مشخص می‌کند که آیا عناصر در آرایه یک بعدی به ترتیب ردیفی (`'C'`) یا به ترتیب ستونی (`'F'`) قرار داده شوند.

در ادامه یک نمونه کد برای استفاده از تابع `.flatten()` را مشاهده می‌کنید:

```python

import numpy as np

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

# تغییر شکل آرایه به یک بعد با استفاده از تابع .flatten()

b = a.flatten()

print(b)

```

خروجی:

```

[1 2 3 4 5 6]

```

در این مثال، با استفاده از تابع `.flatten()`، آرایه `a` را به یک بعد تغییر شکل می‌دهیم و نتیجه را در آرایه `b` ذخیره می‌کنیم.

`numpy.moveaxis(a, source, destination)` -72یک تابع است که بر روی یک آرایه NumPy فراخوانی می‌شود. این تابع محورهای یک آرایه را به محورهای دیگر جابجا می‌کند.

آرگومان `a` نشان دهنده آرایه ورودی است. آرگومان `source` نشان دهنده محورهایی است که قصد داریم به محورهای دیگر جابجا شوند. آرگومان `destination` نشان دهنده محورهایی است که قصد داریم محورهای `source` به آنها جابجا شوند.

در ادامه یک نمونه کد برای استفاده از تابع `moveaxis()` را مشاهده می‌کنید:

```python

import numpy as np

a = np.ones((2, 3, 4))

# جابجایی محورهای 0 و 2 با استفاده از تابع moveaxis()

b = np.moveaxis(a, [0, 2], [2, 0])

print(b.shape)

```

خروجی:

```

(4, 3, 2)

```

در این مثال، آرایه `a` یک آرایه سه بعدی با ابعاد (2, 3, 4) است. با استفاده از تابع `moveaxis()`، محور 0 به محور 2 جابجا شده و محور 2 به محور 0 جابجا می‌شود. نتیجه در آرایه `b` با ابعاد (4, 3, 2) ذخیره می‌شود.

`numpy.rollaxis(a, axis, start=0)` -73یک تابع است که بر روی یک آرایه NumPy فراخوانی می‌شود. این تابع محور مورد نظر را به مکان مشخص شده در آرگومان `start` جابجا می‌کند.

آرگومان `a` نشان دهنده آرایه ورودی است. آرگومان `axis` نشان دهنده محوری است که قصد داریم به مکان `start` جابجا شود. آرگومان `start` نشان دهنده مکان جدیدی است که محور `axis` به آن جابجا می‌شود.

در ادامه یک نمونه کد برای استفاده از تابع `rollaxis()` را مشاهده می‌کنید:

```python

import numpy as np

a = np.ones((3, 4, 5))

# جابجایی محور 2 به مکان 0 با استفاده از تابع rollaxis()

b = np.rollaxis(a, 2, 0)

print(b.shape)

```

خروجی:

```

(5, 3, 4)

```

در این مثال، آرایه `a` یک آرایه سه بعدی با ابعاد (3, 4, 5) است. با استفاده از تابع `rollaxis()`، محور 2 به مکان 0 جابجا می‌شود. نتیجه در آرایه `b` با ابعاد (5, 3, 4) ذخیره می‌شود.

`numpy.swapaxes(a, axis1, axis2)` -74یک تابع است که بر روی یک آرایه NumPy فراخوانی می‌شود و محورهای مشخص شده را جابجا می‌کند.

آرگومان `a` نشان دهنده آرایه ورودی است.

آرگومان `axis1` و `axis2` نشان دهنده دو محور است که قصد داریم آن‌ها را جابجا کنیم.

در ادامه یک نمونه کد برای استفاده از تابع `swapaxes()` را مشاهده می‌کنید:

```python

import numpy as np

a = np.ones((3, 4, 5))

# جابجایی محور 0 با محور 2 با استفاده از تابع swapaxes()

b = np.swapaxes(a, 0, 2)

print(b.shape)

```

خروجی:

```

(5, 4, 3)

```

در این مثال، آرایه `a` یک آرایه سه بعدی با ابعاد (3, 4, 5) است. با استفاده از تابع `swapaxes()`، محور 0 با محور 2 جابجا می‌شود. نتیجه در آرایه `b` با ابعاد (5, 4, 3) ذخیره می‌شود.

`ndarray.T` -75یک ویژگی است که روی آرایه‌های NumPy فراخوانی می‌شود و نقش ترانهاده ماتریس را ایفا می‌کند. این ویژگی ماتریس را جابجا می‌کند، به عبارت دیگر، سطرها را به ستون‌ها و ستون‌ها را به سطرها تبدیل می‌کند.

در ادامه یک نمونه کد برای استفاده از ویژگی `T` را مشاهده می‌کنید:

```python

import numpy as np

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

# ترانهاده ماتریس a با استفاده از ویژگی T

b = a.T

print(b)

```

خروجی:

```

[[1 4]

[2 5]

[3 6]]

```

در این مثال، آرایه `a` یک آرایه دو بعدی با ابعاد (2, 3) است. با استفاده از ویژگی `T`، ماتریس `a` ترانهاده می‌شود و در آرایه `b` با ابعاد (3, 2) ذخیره می‌شود.

`numpy.transpose(a, axes=None)` -76تابعی است که برای تغییر ترتیب ابعاد یک آرایه در NumPy استفاده می‌شود. با استفاده از این تابع، می‌توانید ترتیب ابعاد آرایه را به دلخواه تغییر دهید.

آرگومان‌های تابع `transpose` عبارتند از:

- `a`: آرایه اصلی که می‌خواهید ترتیب ابعاد آن را تغییر دهید.

- `axes` (اختیاری): ترتیب مورد نظر برای ابعاد آرایه. این پارامتر یک ترتیب جدید برای ابعاد را تعیین می‌کند. اگر این پارامتر داده نشود، ترتیب ابعاد آرایه به صورت معکوس تغییر می‌کند.

در ادامه یک نمونه کد برای استفاده از تابع `transpose` را مشاهده می‌کنید:

```python

import numpy as np

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

# تغییر ترتیب ابعاد آرایه a با استفاده از تابع transpose

b = np.transpose(a)

print(b)

```

خروجی:

```

[[1 4]

[2 5]

[3 6]]

```

در این مثال، آرایه `a` یک آرایه دو بعدی با ابعاد (2, 3) است. با استفاده از تابع `transpose`، ترتیب ابعاد آرایه `a` به صورت معکوس تغییر می‌یابد و در آرایه `b` با ابعاد (3, 2) ذخیره می‌شود.

`numpy.atleast\_1d(\*arys)` -77تابعی است که یک یا چند آرایه را به حداقل یک بعد تبدیل می‌کند. اگر ورودی یک آرایه چند بعدی باشد، آن را بدون تغییر برمی‌گرداند. اما اگر ورودی یک آرایه یک بعدی باشد، آن را به یک آرایه یک بعدی تبدیل می‌کند.

آرگومان `arys` در این تابع می‌تواند شامل یک یا چند آرایه باشد که می‌خواهید به حداقل یک بعد تبدیل شوند.

در ادامه یک نمونه کد برای استفاده از تابع `atleast\_1d` را مشاهده می‌کنید:

```python

import numpy as np

a = np.array(5)

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

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

# تبدیل a به یک آرایه یک بعدی

a\_1d = np.atleast\_1d(a)

# تبدیل b به یک آرایه یک بعدی

b\_1d = np.atleast\_1d(b)

# تغییری در آرایه c صورت نمی‌گیرد زیرا از قبل یک آرایه دو بعدی است

c\_1d = np.atleast\_1d(c)

print(a\_1d)

print(b\_1d)

print(c\_1d)

```

خروجی:

```

[5]

[1 2 3]

[[1 2]

[3 4]]

```

`numpy.atleast\_2d(\*arys)` -78تابعی است که یک یا چند آرایه را به حداقل دو بعد تبدیل می‌کند. اگر ورودی یک آرایه یک بعدی باشد، آن را به یک آرایه دو بعدی با یک ردیف تبدیل می‌کند. اگر ورودی یک آرایه دو بعدی باشد، آن را بدون تغییر برمی‌گرداند.

آرگومان `arys` در این تابع می‌تواند شامل یک یا چند آرایه باشد که می‌خواهید به حداقل دو بعد تبدیل شوند.

در ادامه یک نمونه کد برای استفاده از تابع `atleast\_2d` را مشاهده می‌کنید:

```python

import numpy as np

a = np.array(5)

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

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

# تبدیل a به یک آرایه دو بعدی

a\_2d = np.atleast\_2d(a)

# تبدیل b به یک آرایه دو بعدی

b\_2d = np.atleast\_2d(b)

# تغییری در آرایه c صورت نمی‌گیرد زیرا از قبل یک آرایه دو بعدی است

c\_2d = np.atleast\_2d(c)

print(a\_2d)

print(b\_2d)

print(c\_2d)

```

خروجی:

```

[[5]]

[[1 2 3]]

[[1 2]

[3 4]]

```

در این مثال، ما سه آرایه `a`، `b` و `c` داریم. با استفاده از تابع `atleast\_2d`، آرایه `a` به یک آرایه دو بعدی با یک ردیف تبدیل می‌شود و در `a\_2d` ذخیره می‌شود. آرایه `b` از قبل یک آرایه یک بعدی است، بنابراین به یک آرایه دو بعدی با یک ردیف تبدیل می‌شود و در `b\_2d` ذخیره می‌شود.

`numpy.atleast\_3d(\*arys)` -79تابعی است که یک یا چند آرایه را به حداقل سه بعد تبدیل می‌کند. اگر ورودی یک آرایه یک بعدی باشد، آن را به یک آرایه سه بعدی با یک ردیف و یک ستون تبدیل می‌کند. اگر ورودی یک آرایه دو بعدی باشد، آن را به یک آرایه سه بعدی با تنظیم سایز بعد سوم به 1 تبدیل می‌کند. اگر ورودی یک آرایه سه بعدی باشد، آن را بدون تغییر برمی‌گرداند.

آرگومان `arys` در این تابع می‌تواند شامل یک یا چند آرایه باشد که می‌خواهید به حداقل سه بعد تبدیل شوند.

در ادامه یک نمونه کد برای استفاده از تابع `atleast\_3d` را مشاهده می‌کنید:

```python

import numpy as np

a = np.array(5)

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

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

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

# تبدیل a به یک آرایه سه بعدی

a\_3d = np.atleast\_3d(a)

# تبدیل b به یک آرایه سه بعدی

b\_3d = np.atleast\_3d(b)

# تبدیل c به یک آرایه سه بعدی

c\_3d = np.atleast\_3d(c)

# تغییری در آرایه d صورت نمی‌گیرد زیرا از قبل یک آرایه سه بعدی است

d\_3d = np.atleast\_3d(d)

print(a\_3d)

print(b\_3d)

print(c\_3d)

print(d\_3d)

```

خروجی:

```

[[[5]]]

[[[1]

[2]

[3]]]

[[[1]

[2]]

[[3]

[4]]]

[[[1 2]

[3 4]]]

```

`numpy.broadcast` یک کلاس در NumPy است که برای انتشار (broadcasting) آرایه‌ها به همان شکلی که نیاز است، استفاده می‌شود. انتشار یا برادکستینگ یک مکانیزم در NumPy است که به شما امکان می‌دهد آرایه‌ها را به صورت خودکار به همان ابعاد و اندازه‌های مورد نیاز برای اجرای عملیات محاسباتی مورد نظر تغییر دهید.

-80تنظیمات و پارامترهای `broadcast` در کلاس `numpy.broadcast(\*arr)` ذخیره می‌شوند و می‌توانید با استفاده از متدهایی مانند `shape`, `nd` و `index` به اطلاعات مربوطه دسترسی پیدا کنید. این کلاس در اصل یک iterator است و می‌توانید از آن برای تکرار از طریق ابعاد مختلف آرایه‌ها استفاده کنید.

در مثال زیر، با استفاده از کلاس `numpy.broadcast`، ابعاد دو آرایه مختلف را بررسی می‌کنیم:

```python

import numpy as np

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

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

broadcasted = np.broadcast(a, b)

print('Shape:', broadcasted.shape)

print('Number of dimensions:', broadcasted.nd)

print('Current index:', broadcasted.index)

```

خروجی:

```

Shape: (3, 3)

Number of dimensions: 2

Current index: (0, 0)

```

در این مثال، دو آرایه `a` و `b` را ایجاد کرده و سپس با استفاده از `np.broadcast`، یک شیء `broadcasted` ایجاد می‌شود. با استفاده از متدهای `shape`, `nd` و `index` به ترتیب شکل (ابعاد)، تعداد بعد و شاخص فعلی از شیء `broadcasted` دسترسی پیدا می‌کنیم.

استفاده از `numpy.broadcast` به شما امکان می‌دهد به طور پویا و بهینه آرایه‌های با ابعاد مختلف را در محاسبات NumPy بکار ببرید.

-81تابع `numpy.broadcast\_to(array,shape,subok)` برای گسترش یک آرایه به یک شکل مشخص استفاده می‌شود. آرگومان‌های این تابع عبارتند از:

- `array`: آرایه اصلی که قصد گسترش آن را داریم.

- `shape`: شکل جدیدی که می‌خواهیم آرایه را به آن گسترش دهیم.

- `subok`: یک پارامتر بولی که در صورت True بودن، اگر آرایه اصلی نیز نوعی زیرکلاس (subclass) از ndarray باشد، نوع آرایه گسترش یافته نیز نوعی زیرکلاس از آن خواهد بود.

با استفاده از تابع `numpy.broadcast\_to`، می‌توانیم آرایه `array` را به شکل جدید `shape` گسترش داده و یک نسخه جدید از آن بسازیم. اگر `shape` از شکل اصلی آرایه `array` بزرگتر باشد، آرایه تکرار خواهد شد تا به شکل مورد نظر برسد. اگر `shape` از شکل اصلی آرایه کوچکتر یا مساوی باشد، آرایه اصلی بدون تغییر برگردانده می‌شود.

در ادامه، یک نمونه کد از استفاده از تابع `numpy.broadcast\_to` را به همراه خروجی آن نشان می‌دهم:

``` python

import numpy as np

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

shape = (3, 4)

result = np.broadcast\_to(array, shape)

print(result)

```

خروجی:

```

[[1 2 3 1]

[2 3 1 2]

[3 1 2 3]]

```

در این مثال، آرایه `array` با شکل (3,) به شکل جدید (3, 4) گسترش داده شده است. به دلیل کوچکتر بودن `shape` نسبت به شکل اصلی آرایه، آرایه `array` تکرار شده است تا به شکل جدید برسد.

-`numpy.broadcast\_arrays(\*args,subok)` -82یک تابع است که آرایه‌ها را به صورت همزمان پخش می‌کند و آرایه‌های پخش شده را برمی‌گرداند. این تابع می‌تواند آرایه‌هایی با اشکال مختلف را به صورت خودکار به یک شکل مشترک پخش کند، به طوری که بتوانید آن‌ها را به صورت مستقل به عنوان آرایه‌های جداگانه مورد استفاده قرار دهید.

آرگومان `\*args` به مجموعه‌ای از آرایه‌ها اشاره می‌کند که می‌خواهید آن‌ها را پخش کنید. آرایه‌ها می‌توانند تعدادی آرایه با اشکال و ابعاد مختلف باشند.

آرگومان `subok` که اختیاری است، تعیین می‌کند که آرایه‌های پخش شده باید زیرکلاس کلاس اصلی آرایه‌ها باشند (`subok=True`) یا آیا آرایه‌های پخش شده باید نمونه جدیدی باشند (`subok=False`).

خروجی این تابع یک لیست از آرایه‌های پخش شده است که هر کدام از آرایه‌ها یک نمونه جداگانه از داده‌های آرایه اصلی هستند. تعداد و اشکال آرایه‌های پخش شده با توجه به ابعاد و اشکال آرایه‌های اصلی تعیین می‌شود.

این تابع بسیار مفید است زمانی که می‌خواهید آرایه‌های مختلفی را به صورت خودکار به یک شکل مشترک پخش کنید و در عملیاتی مانند جمع، ضرب و غیره بر روی آن‌ها اعمال کنید.

در زیر یک نمونه کد از استفاده از `numpy.broadcast\_arrays` را مشاهده می‌کنید:

``` python

import numpy as np

a = np.array([1, 2, 3]) # ابعاد: (3,)

b = np.array([[4, 5, 6], [7

``` python

import numpy as np

a = np.array([1, 2, 3]) # ابعاد: (3,)

b = np.array([[4, 5, 6], [7, 8, 9]]) # ابعاد: (2, 3)

broadcasted\_arrays = np.broadcast\_arrays(a, b)

# چاپ آرایه‌های پخش شده

for arr in broadcasted\_arrays:

print(arr)

# نتیجه:

# [1 2 3]

# [[4 5 6]

# [7 8 9]]

```

در این مثال، ما دو آرایه `a` و `b` را تعریف کرده‌ایم. سپس از تابع `np.broadcast\_arrays` استفاده کرده‌ایم تا آرایه‌های پخش شده متناظر با `a` و `b` را بدست آوریم. در این حالت، آرایه `a` با ابعاد (3,) به طور خودکار به شکل (1, 3) پخش شده است و آرایه `b` با ابعاد (2, 3) به صورت خودکار پخش شده است. سپس آرایه‌های پخش شده را چاپ کرده‌ایم که خروجی به شکل زیر خواهد بود:

```

[1 2 3]

[[4 5 6]

[7 8 9]]

```

به این ترتیب، ما دو آرایه با ابعاد مختلف را به صورت خودکار به یک شکل مشترک پخش کردیم و آن‌ها را به صورت آرایه‌های مستقل قابل استفاده قرار دادیم.

-83تابع `numpy.expand\_dims(a, axis)` برای افزودن یک بُعد به آرایه `a` در محور مشخص شده با `axis` استفاده می‌شود. این تابع باعث تغییر ابعاد آرایه می‌شود و آرایه جدیدی با ابعاد بزرگتر از آرایه اصلی برمی‌گرداند.

در ادامه یک نمونه کد را به همراه توضیح آن ارائه می‌دهم:

``` python

import numpy as np

a = np.array([1, 2, 3]) # ابعاد: (3,)

expanded\_a = np.expand\_dims(a, axis=1)

print(expanded\_a.shape) # خروجی: (3, 1)

print(expanded\_a)

# نتیجه:

# [[1]

# [2]

# [3]]

```

در این مثال، ما آرایه `a` را با ابعاد (3,) تعریف کرده‌ایم. با استفاده از `np.expand\_dims(a, axis=1)`، یک بُعد به آرایه `a` در محور 1 اضافه می‌شود. خروجی این عملیات یک آرایه با ابعاد (3, 1) است که شکل آن به شکل زیر است:

```

[[1]

[2]

[3]]

```

با افزودن بُعد در محور 1، طول آرایه به 3 تغییر نکرده است، اما عرض آرایه به 1 تغییر کرده است. این عمل باعث می‌شود که آرایه `expanded\_a` به صورت آرایه دوبعدی با 3 سطر و 1 ستون نمایش داده شود.

-84تابع `numpy.squeeze(a, axis=None)` برای حذف ابعاد یک به یک از آرایه `a` استفاده می‌شود. این تابع باعث تغییر ابعاد آرایه می‌شود و آرایه جدیدی با ابعاد کوچکتر از آرایه اصلی برمی‌گرداند.

در صورت استفاده از پارامتر `axis`، فقط ابعادی که دارای طول 1 هستند و در محور مشخص شده با `axis` قرار دارند، حذف می‌شوند. در صورت عدم تعیین مقدار `axis`، تمام ابعادی که دارای طول 1 هستند حذف می‌شوند.

در ادامه یک نمونه کد را به همراه توضیح آن ارائه می‌دهم:

``` python

import numpy as np

a = np.array([[[1, 2, 3]]]) # ابعاد: (1, 1, 3)

squeezed\_a = np.squeeze(a)

print(squeezed\_a.shape) # خروجی: (3,)

print(squeezed\_a)

# نتیجه:

# [1 2 3]

```

در این مثال، ما آرایه `a` را با ابعاد (1, 1, 3) تعریف کرده‌ایم. با استفاده از `np.squeeze(a)`، ابعادی که دارای طول 1 هستند حذف می‌شوند. خروجی این عملیات یک آرایه با ابعاد (3,) است که شکل آن به شکل زیر است:

```

[1 2 3]

```

با حذف ابعاد با طول 1، آرایه `squeezed\_a` به صورت یک بردار یک بعدی با طول 3 نمایش داده می‌شود.

-85تابع `numpy.asarray(a, dtype=None, order=None, \*)` به منظور تبدیل ورودی `a` به آرایه NumPy استفاده می‌شود. این تابع ورودی `a` را به صورت آرایه NumPy با نوع داده مشخص `dtype` و ترتیب ذخیره‌سازی مشخص `order` تبدیل می‌کند.

- پارامتر `a`: ورودی که به آرایه تبدیل می‌شود.

- پارامتر `dtype`: نوع داده مورد نظر برای آرایه تولید شده. اگر مقدار `dtype` مشخص نشود، آرایه تولید شده همان نوع داده ورودی `a` خواهد داشت.

- پارامتر `order`: ترتیب ذخیره‌سازی داده‌ها در آرایه تولید شده. مقادیر ممکن برای این پارامتر عبارتند از `"C"` برای ترتیب ذخیره‌سازی روی سطرها (ترتیب معمول در NumPy) و `"F"` برای ترتیب ذخیره‌سازی روی ستون‌ها.

در ادامه یک نمونه کد را به همراه توضیح آن ارائه می‌دهم:

```python

import numpy as np

# تبدیل لیست به آرایه

list\_a = [1, 2, 3, 4, 5]

arr\_a = np.asarray(list\_a)

print(arr\_a) # خروجی: [1 2 3 4 5]

print(arr\_a.dtype) # خروجی: int64

# تبدیل تاپل به آرایه با نوع داده مشخص

tuple\_a = (1.1, 2.2, 3.3)

arr\_a = np.asarray(tuple\_a, dtype=float)

print(arr\_a) # خروجی: [1.1 2.2 3.3]

print(arr\_a.dtype) # خروجی: float64

```

در این مثال، ما ابتدا لیست `list\_a` را به آرایه `arr\_a` تبدیل کردیم. تابع `np.asarray` به طور خودکار نوع داده لیست را تشخیص داد و آرایه تولید شده نیز همان نوع داده لیست (`int64`) را داراست.

-86تابع `numpy.asanyarray(a, dtype=None, order=None, \*)` برای تبدیل ورودی `a` به یک آرایه NumPy استفاده می‌شود. این تابع عملکردی شبیه به `numpy.asarray` دارد، با این تفاوت که وقتی `a` یک آرایه NumPy باشد، آرایه ورودی را به صورت غیردسته‌ای (unstructured) برمی‌گرداند. اگر `a` غیر از آرایه باشد، یک آرایه NumPy با تنظیمات مشخص شده توسط پارامترهای `dtype` و `order` تولید می‌شود.

- پارامتر `a`: ورودی که به آرایه تبدیل می‌شود.

- پارامتر `dtype`: نوع داده مورد نظر برای آرایه تولید شده. اگر مقدار `dtype` مشخص نشود، آرایه تولید شده همان نوع داده ورودی `a` خواهد داشت.

- پارامتر `order`: ترتیب ذخیره‌سازی داده‌ها در آرایه تولید شده. مقادیر ممکن برای این پارامتر عبارتند از `"C"` برای ترتیب ذخیره‌سازی روی سطرها (ترتیب معمول در NumPy) و `"F"` برای ترتیب ذخیره‌سازی روی ستون‌ها.

در ادامه یک نمونه کد به همراه توضیح آن ارائه می‌شود:

```python

import numpy as np

# تبدیل لیست به آرایه

list\_a = [1, 2, 3, 4, 5]

arr\_a = np.asanyarray(list\_a)

print(arr\_a) # خروجی: [1 2 3 4 5]

print(arr\_a.dtype) # خروجی: int64

# تبدیل تاپل به آرایه با نوع داده مشخص

tuple\_a = (1.1, 2.2, 3.3)

arr\_a = np.asanyarray(tuple\_a, dtype=float)

print(arr\_a) # خروجی: [1.1 2.2 3.3]

print(arr\_a.dtype) # خروجی: float64

-87تابع `numpy.asmatrix(data, dtype=None)` برای تبدیل ورودی `data` به یک ماتریس در NumPy استفاده می‌شود. این تابع یک نمونه از کلاس `numpy.matrix` را برمی‌گرداند که به صورت ماتریس دوبعدی نمایش داده می‌شود.

- پارامتر `data`: ورودی که به ماتریس تبدیل می‌شود. می‌تواند یک آرایه NumPy، لیست، تاپل یا شیء قابل تبدیل به ماتریس باشد.

- پارامتر `dtype`: نوع داده مورد نظر برای ماتریس تولید شده. اگر مقدار `dtype` مشخص نشود، نوع داده ورودی `data` استفاده خواهد شد.

در ادامه یک نمونه کد به همراه توضیح آن ارائه می‌شود:

```python

import numpy as np

# تبدیل آرایه NumPy به ماتریس

arr\_a = np.array([[1, 2, 3], [4, 5, 6]])

mat\_a = np.asmatrix(arr\_a)

print(mat\_a) # خروجی: [[1 2 3]

# [4 5 6]]

print(type(mat\_a)) # خروجی: <class 'numpy.matrix'>

# تبدیل لیست به ماتریس

list\_a = [[1, 2, 3], [4, 5, 6]]

mat\_a = np.asmatrix(list\_a)

print(mat\_a) # خروجی: [[1 2 3]

# [4 5 6]]

print(type(mat\_a)) # خروجی: <class 'numpy.matrix'>

```

در این مثال، ابتدا آرایه NumPy `arr\_a` را به ماتریس `mat\_a` تبدیل کردیم. سپس لیست `list\_a` را به ماتریس `mat\_a` تبدیل کردیم.

-88تابع `numpy.asfarray(a, dtype=numpy.double)` به یک آرایه با داده‌های حقیقی در NumPy تبدیل ورودی `a` را تبدیل می‌کند. این تابع یک نسخه با داده‌های ممکن است کمی پیشرفته‌تر از ورودی اولیه است.

- پارامتر `a`: ورودی که به آرایه با داده‌های حقیقی تبدیل می‌شود. می‌تواند یک آرایه NumPy، لیست، تاپل یا شیء قابل تبدیل به آرایه با داده‌های حقیقی باشد.

- پارامتر `dtype`: نوع داده مورد نظر برای آرایه تولید شده. اگر مقدار `dtype` مشخص نشود، نوع داده `numpy.double` استفاده خواهد شد.

در ادامه یک نمونه کد به همراه توضیح آن ارائه می‌شود:

```python

import numpy as np

# تبدیل آرایه NumPy به آرایه با داده‌های حقیقی

arr\_a = np.array([1, 2, 3])

asfarr\_a = np.asfarray(arr\_a)

print(asfarr\_a) # خروجی: [1. 2. 3.]

print(asfarr\_a.dtype) # خروجی: float64

# تبدیل لیست به آرایه با داده‌های حقیقی

list\_a = [1, 2, 3]

asfarr\_a = np.asfarray(list\_a)

print(asfarr\_a) # خروجی: [1. 2. 3.]

print(asfarr\_a.dtype) # خروجی: float64

```

در این مثال، ابتدا آرایه NumPy `arr\_a` را به آرایه با داده‌های حقیقی `asfarr\_a` تبدیل کردیم. سپس لیست `list\_a` را به آرایه با داده‌های حقیقی `asfarr\_a` تبدیل کردیم.

-89تابع `numpy.asfortranarray(a, dtype=None)` بازنمایی ورودی `a` را به یک آرایه NumPy با نوع داده و ترتیب ذخیره‌سازی "Fortran" تبدیل می‌کند. در ترتیب ذخیره‌سازی "Fortran"، داده‌ها به صورت ستونی در حافظه ذخیره می‌شوند.

- پارامتر `a`: ورودی که به آرایه با ترتیب ذخیره‌سازی "Fortran" تبدیل می‌شود. می‌تواند یک آرایه NumPy، لیست، تاپل و یا شیء قابل تبدیل به آرایه باشد.

- پارامتر `dtype`: نوع داده مورد نظر برای آرایه تولید شده. اگر مقدار `dtype` مشخص نشود، نوع داده ورودی `a` استفاده خواهد شد.

در ادامه، یک نمونه کد به همراه توضیح آن نشان داده می‌شود:

```python

import numpy as np

# تبدیل آرایه NumPy به آرایه با ترتیب ذخیره‌سازی "Fortran"

arr\_a = np.array([[1, 2, 3], [4, 5, 6]])

asfortranarr\_a = np.asfortranarray(arr\_a)

print(asfortranarr\_a)

# Output:

# [[1 2 3]

# [4 5 6]]

print(asfortranarr\_a.flags['F\_CONTIGUOUS']) # True

# تبدیل لیست به آرایه با ترتیب ذخیره‌سازی "Fortran"

list\_a = [[1, 2, 3], [4, 5, 6]]

asfortranarr\_a = np.asfortranarray(list\_a)

print(asfortranarr\_a)

# Output:

# [[1 2 3]

# [4 5 6]]

print(asfortranarr\_a.flags['F\_CONTIGUOUS']) # True

```

در این مثال، ابتدا آرایه NumPy `arr\_a` را به یک آرایه با ترتیب ذخیره‌سازی "Fortran" تبدیل کردیم و نتیجه را در `asfortranarr\_a` ذخیره کردیم. سپس لیست `list\_a` را به یک آرایه با ترتیب ذخیره‌سازی "Fortran" تبدیل کردیم.

-90تابع `numpy.ascontiguousarray(a, dtype=None)` ورودی `a` را به یک آرایه NumPy با نوع داده و نوع حافظه پیوسته تبدیل می‌کند. اگر آرایه ورودی `a` در حافظه پیوسته باشد، آرایه ورودی برگردانده می‌شود. در غیر این صورت، یک نسخه پیوسته از آرایه ورودی ایجاد می‌شود و برگردانده می‌شود.

- پارامتر `a`: ورودی که به آرایه با حافظه پیوسته تبدیل می‌شود. می‌تواند یک آرایه NumPy، لیست، تاپل و یا شیء قابل تبدیل به آرایه باشد.

- پارامتر `dtype`: نوع داده مورد نظر برای آرایه تولید شده. اگر مقدار `dtype` مشخص نشود، نوع داده ورودی `a` استفاده خواهد شد.

در ادامه، یک نمونه کد به همراه توضیح آن نشان داده می‌شود:

```python

import numpy as np

# تبدیل آرایه NumPy به آرایه با حافظه پیوسته

arr\_a = np.array([[1, 2, 3], [4, 5, 6]])

ascontiguousarr\_a = np.ascontiguousarray(arr\_a)

print(ascontiguousarr\_a)

# Output:

# [[1 2 3]

# [4 5 6]]

print(ascontiguousarr\_a.flags['C\_CONTIGUOUS']) # True

# تبدیل لیست به آرایه با حافظه پیوسته

list\_a = [[1, 2, 3], [4, 5, 6]]

ascontiguousarr\_a = np.ascontiguousarray(list\_a)

print(ascontiguousarr\_a)

# Output:

# [[1 2 3]

# [4 5 6]]

print(ascontiguousarr\_a.flags['C\_CONTIGUOUS']) # True

```

در این مثال، ابتدا آرایه NumPy `arr\_a` را به یک آرایه با حافظه پیوسته تبدیل کردیم و نتیجه را در `ascontiguousarr\_a` ذخیره کردیم. سپس لیست `list\_a` را به یک آرایه با حافظه پیوسته تبدیل کردیم.

-91تابع `numpy.asarray\_chkfinite(a, dtype=None, order=None)` ورودی `a` را به یک آرایه NumPy تبدیل می‌کند و در عین حال بررسی می‌کند که آیا تمام عناصر آرایه مقادیر نامتناهی ندارند. در صورت وجود عنصر نامتناهی در آرایه، یک `ValueError` ایجاد می‌شود.

آرگومان‌های تابع عبارتند از:

- `a`: ورودی که به یک آرایه تبدیل می‌شود.

- `dtype` (اختیاری): نوع داده مورد نظر برای آرایه خروجی. اگر این آرگومان مشخص نشود، نوع داده ورودی حفظ می‌شود.

- `order` (اختیاری): نحوه ترتیب داده‌ها در حافظه. این آرگومان تنها برای آرایه‌های دوبعدی یا بیشتر قابل استفاده است.

در صورتی که آرایه از پیش یک آرایه NumPy باشد و همه عناصر آن نامتناهی باشند، تغییری اعمال نمی‌شود و همان آرایه به عنوان خروجی برگردانده می‌شود.

برای مثال، در نمونه زیر ابتدا یک لیست با داده‌های عددی و یک عنصر نامتناهی تعریف می‌کنیم، سپس تابع `numpy.asarray\_chkfinite` را برروی این لیست اعمال می‌کنیم:

``` python

import numpy as np

data = [1, 2, 3, float('inf')]

arr = np.asarray\_chkfinite(data)

print(arr)

```

خروجی:

```

ValueError: array contains NaN or infinity

```

با توجه به وجود عنصر نامتناهی در لیست، خطا به دلیل اعمال تابع `numpy.asarray\_chkfinite` رخ می‌دهد.

-92تابع `numpy.require(a, dtype=None, requirements=None, \*, like=None)` ورودی `a` را به یک آرایه NumPy تبدیل می‌کند و اطمینان حاصل می‌کند که نوع داده و سایر ویژگی‌های مشخص شده برای آرایه ارضا شوند. اگر آرایه ورودی از پیش نوع و سایر ویژگی‌های مورد نیاز را داشته باشد، هیچ تغییری اعمال نمی‌شود و همان آرایه به عنوان خروجی برگردانده می‌شود. در غیر این صورت، آرایه ورودی به نوع و سایر ویژگی‌های مورد نیاز تبدیل می‌شود.

آرگومان‌های تابع عبارتند از:

- `a`: ورودی که به یک آرایه تبدیل می‌شود.

- `dtype` (اختیاری): نوع داده مورد نظر برای آرایه خروجی. اگر این آرگومان مشخص نشود، نوع داده ورودی حفظ می‌شود.

- `requirements` (اختیاری): لیستی از ویژگی‌های مورد نیاز برای آرایه خروجی. این ویژگی‌ها می‌توانند مانند `'C'` یا `'F'` برای تعیین ترتیب داده‌ها در حافظه، یا `contiguous` برای بررسی اینکه آیا آرایه باید داده‌های متوالی داشته باشد، باشند.

برای مثال، در کد زیر یک آرایه NumPy تعریف می‌کنیم و سپس تابع `numpy.require` را برروی آن اعمال می‌کنیم:

``` python

import numpy as np

a = np.array([1, 2, 3], dtype=np.float32)

required\_array = np.require(a, dtype=np.float64, requirements=['C', 'aligned'])

print(required\_array)

print(required\_array.dtype)

print(required\_array.flags)

```

خروجی:

```

[1. 2. 3.]

float64

C\_CONTIGUOUS : True

F\_CONTIGUOUS : False

OWNDATA : True

...

```

دستورات زیر راهنمایی می‌کنند که چه ویژگی‌هایی را می‌توان در پارامتر `requirements` تابع `numpy.require` استفاده کرد:

- `'F\_CONTIGUOUS'` یا `'F'`: اطمینان حاصل می‌کند که آرایه به صورت Fortran-contiguous باشد، به این معنی که داده‌ها به ترتیب ستونی در حافظه قرار گیرند.

- `'C\_CONTIGUOUS'` یا `'C'`: اطمینان حاصل می‌کند که آرایه به صورت C-contiguous باشد، به این معنی که داده‌ها به ترتیب ردیفی در حافظه قرار گیرند.

- `'ALIGNED'` یا `'A'`: اطمینان حاصل می‌کند که آرایه داده‌های آن به ترتیب خاصی در حافظه قرار گیرند که با نوع داده سازگار باشند.

- `'WRITEABLE'` یا `'W'`: اطمینان حاصل می‌کند که آرایه قابل ویرایش باشد.

- `'OWNDATA'` یا `'O'`: اطمینان حاصل می‌کند که آرایه داده خود را مالک خود باشد.

- `'ENSUREARRAY'` یا `'E'`: اطمینان حاصل می‌کند که آرایه‌ای که برگشت داده می‌شود، آرایه پایه باشد و نه زیرکلاسی از آرایه.

مثال زیر نشان می‌دهد چگونه از برخی از این ویژگی‌ها در `requirements` استفاده می‌شود:

``` python

import numpy as np

a = np.array([1, 2, 3], dtype=np.float32)

required\_array = np.require(a, requirements=['C\_CONTIGUOUS', 'ALIGNED', 'WRITEABLE'])

print(required\_array)

print(required\_array.flags)

```

خروجی:

```

[1. 2. 3.]

C\_CONTIGUOUS : True

F\_CONTIGUOUS : False

OWNDATA : False

...

```

در این مثال، آرایه `a` به یک آرایه با ترتیب C-contiguous، داده‌های متوالی و قابل ویرایش تبدیل می‌شود.

-93تابع `numpy.concatenate(a1,a2,axis,out,dtype,casting)` برای ادغام (concatenation) یا اتصال چندین آرایه به یکدیگر در امتداد یک محور مشخص استفاده می‌شود. آرگومان‌های این تابع عبارتند از:

- `a1, a2, ...`: آرایه‌ها که می‌خواهید به یکدیگر اتصال دهید.

- `axis`: محوری که در آن اتصال انجام می‌شود. برای مقدار پیش‌فرض `axis=0`، اتصال به طول عمودی (ستونی) صورت می‌گیرد. برای مقدار `axis=1`، اتصال به طول افقی (سطری) انجام می‌شود.

- `out`: آرایه خروجی (output array) که در آن نتیجه اتصال قرار می‌گیرد. اگر `out` مقدار دهید، نتیجه در آن ذخیره می‌شود. در غیر این صورت، یک آرایه جدید ایجاد می‌شود.

- `dtype`: نوع داده خروجی. اگر مشخص نشود، نوع داده ترکیبی (combined dtype) آرایه‌های ورودی استفاده می‌شود.

- `casting`: روش تبدیل داده‌ها در صورت نیاز. مقادیر مجاز عبارتند از `"no"`، `"equiv"`، `"safe"`، `"same\_kind"`، `"unsafe"`.

در زیر یک نمونه کد برای استفاده از `numpy.concatenate` آورده شده است:

``` python

import numpy as np

a1 = np.array([1, 2, 3])

a2 = np.array([4, 5, 6])

a3 = np.array([7, 8, 9])

result = np.concatenate((a1, a2, a3), axis=1)

print(result)

```

خروجی:

```

[1 2 3 4 5 6 7 8 9]

```

در این مثال، آرایه‌های `a1`، `a2` و `a3` به صورت عمودی به هم اتصال داده شده‌اند.

-94تابع `numpy.stack(arrays,axis,out,dtype,casting)` برای تراکم کردن (stacking) چندین آرایه به هم در یک محور مشخص استفاده می‌شود. آرگومان‌های این تابع عبارتند از:

- `arrays`: آرایه‌هایی که می‌خواهید تراکم کنید. آرایه‌ها باید همان اندازه باشند.

- `axis`: محوری که در آن تراکم صورت می‌گیرد. برای مقدار پیش‌فرض `axis=0`، تراکم در امتداد عمودی (ستونی) صورت می‌گیرد. برای مقدار `axis=1`، تراکم در امتداد افقی (سطری) انجام می‌شود.

- `out`: آرایه خروجی (output array) که در آن نتیجه تراکم قرار می‌گیرد. اگر `out` مقدار دهید، نتیجه در آن ذخیره می‌شود. در غیر این صورت، یک آرایه جدید ایجاد می‌شود.

- `dtype`: نوع داده خروجی. اگر مشخص نشود، نوع داده ترکیبی (combined dtype) آرایه‌های ورودی استفاده می‌شود.

- `casting`: روش تبدیل داده‌ها در صورت نیاز. مقادیر مجاز عبارتند از `"no"`، `"equiv"`، `"safe"`، `"same\_kind"`، `"unsafe"`.

در زیر یک نمونه کد برای استفاده از `numpy.stack` آورده شده است:

``` python

import numpy as np

a1 = np.array([1, 2, 3])

a2 = np.array([4, 5, 6])

a3 = np.array([7, 8, 9])

result = np.stack((a1, a2, a3), axis=0)

print(result)

```

خروجی:

```

[[1 2 3]

[4 5 6]

[7 8 9]]

```

در این مثال، آرایه‌های `a1`، `a2` و `a3` در امتداد محور صفحه (عمودی) به هم تراکم شده‌اند.

-95تابع `numpy.block(arrays)` برای ساخت یک آرایه چندبعدی از آرایه‌های فرعی با ابعاد مختلف استفاده می‌شود. این تابع می‌تواند به صورت یکپارچه آرایه‌های فرعی را با هم ترکیب کند. آرگومان `arrays` یک لیست از آرایه‌ها است که باید از نوع ndarray باشند.

در حالت ساده، آرایه‌های فرعی باید با ابعاد یکسان باشند، با استفاده از `numpy.block` می‌توان آن‌ها را در یک صفحه (در یک سطر) یا در یک ستون قرار داد. همچنین، با استفاده از ابعاد مختلف می‌توان نواحی مختلفی از آرایه‌های فرعی را ترکیب کرد.

در زیر یک نمونه کد برای استفاده از `numpy.block` آورده شده است:

``` python

import numpy as np

a1 = np.array([[1, 2], [3, 4]])

a2 = np.array([[5, 6], [7, 8]])

a3 = np.array([[9, 10], [11, 12]])

result = np.block([[a1, a2], [a3]])

print(result)

```

خروجی:

```

[[ 1 2 5 6]

[ 3 4 7 8]

[ 9 10 11 12]]

```

در این مثال، آرایه‌های `a1`، `a2` و `a3` در یک سطر و ستون قرار گرفته‌اند و توسط `numpy.block` ترکیب شده‌اند.

-96تابع `numpy.vstack(tup)` برای ادغام آرایه‌ها به صورت عمودی (به اضافه کردن سطرها) استفاده می‌شود. آرگومان `tup` یک tuple از آرایه‌ها است که باید از نوع ndarray باشند.

تابع `numpy.vstack` سطرهای آرایه‌ها را به صورت پشت سر هم در یک آرایه جدید قرار می‌دهد. ابعاد ستونهای آرایه‌ها باید یکسان باشند. اگر ابعاد آرایه‌ها متفاوت باشد، خطا خواهد رخ داد.

در زیر یک نمونه کد برای استفاده از `numpy.vstack` آورده شده است:

``` python

import numpy as np

a1 = np.array([1, 2, 3])

a2 = np.array([4, 5, 6])

a3 = np.array([7, 8, 9])

result = np.vstack((a1, a2, a3))

print(result)

```

خروجی:

```

[[1 2 3]

[4 5 6]

[7 8 9]]

```

در این مثال، آرایه‌های `a1`، `a2` و `a3` عمودی به هم متصل شده‌اند و توسط `numpy.vstack` ترکیب شده‌اند.

-97تابع `numpy.hstack(tup)` برای ادغام آرایه‌ها به صورت افقی (به اضافه کردن ستون‌ها) استفاده می‌شود. آرگومان `tup` یک tuple از آرایه‌ها است که باید از نوع ndarray باشند.

تابع `numpy.hstack` ستون‌های آرایه‌ها را به صورت پشت سر هم در یک آرایه جدید قرار می‌دهد. ابعاد سطرهای آرایه‌ها باید یکسان باشند. اگر ابعاد آرایه‌ها متفاوت باشد، خطا خواهد رخ داد.

در زیر یک نمونه کد برای استفاده از `numpy.hstack` آورده شده است:

``` python

import numpy as np

a1 = np.array([1, 2, 3])

a2 = np.array([4, 5, 6])

a3 = np.array([7, 8, 9])

result = np.hstack((a1, a2, a3))

print(result)

```

خروجی:

```

[1 2 3 4 5 6 7 8 9]

```

در این مثال، آرایه‌های `a1`، `a2` و `a3` افقی به هم متصل شده‌اند و توسط `numpy.hstack` ترکیب شده‌اند.

-98تابع `numpy.dstack(tup)` برای ادغام آرایه‌ها به صورت عمودی (به اضافه کردن صفحات) استفاده می‌شود. آرگومان `tup` یک tuple از آرایه‌ها است که باید از نوع ndarray باشند.

تابع `numpy.dstack` صفحات آرایه‌ها را به صورت پشت سر هم در یک آرایه جدید قرار می‌دهد. ابعاد ستون‌ها و سطرهای آرایه‌ها باید یکسان باشند. اگر ابعاد آرایه‌ها متفاوت باشد، خطا خواهد رخ داد.

در زیر یک نمونه کد برای استفاده از `numpy.dstack` آورده شده است:

``` python

import numpy as np

a1 = np.array([[1, 2], [3, 4]])

a2 = np.array([[5, 6], [7, 8]])

a3 = np.array([[9, 10], [11, 12]])

result = np.dstack((a1, a2, a3))

print(result)

```

خروجی:

```

[[[ 1 5 9]

[ 2 6 10]]

[[ 3 7 11]

[ 4 8 12]]]

```

در این مثال، آرایه‌های `a1`، `a2` و `a3` عمودی به هم متصل شده‌اند و توسط `numpy.dstack` ترکیب شده‌اند.

-99تابع `numpy.column\_stack(tup)` برای ادغام آرایه‌ها به صورت عمودی (به اضافه کردن ستون‌ها) استفاده می‌شود. آرگومان `tup` یک tuple از آرایه‌ها است که باید از نوع ndarray باشند.

تابع `numpy.column\_stack` ستون‌های آرایه‌ها را به صورت پشت سر هم در یک آرایه جدید قرار می‌دهد. تعداد سطرهای آرایه‌ها باید یکسان باشد. اگر تعداد سطرهای آرایه‌ها متفاوت باشد، خطا خواهد رخ داد.

در زیر یک نمونه کد برای استفاده از `numpy.column\_stack` آورده شده است:

``` python

import numpy as np

a1 = np.array([1, 2, 3])

a2 = np.array([4, 5, 6])

a3 = np.array([7, 8, 9])

result = np.column\_stack((a1, a2, a3))

print(result)

```

خروجی:

```

[[1 4 7]

[2 5 8]

[3 6 9]]

```

در این مثال، آرایه‌های `a1`، `a2` و `a3` افقی به هم متصل شده‌اند و توسط `numpy.column\_stack` ترکیب شده‌اند.

-100تابع `numpy.row\_stack(tup)` برای ادغام آرایه‌ها به صورت عمودی (به اضافه کردن سطرها) استفاده می‌شود. آرگومان `tup` یک tuple از آرایه‌ها است که باید از نوع ndarray باشند.

تابع `numpy.row\_stack` سطرهای آرایه‌ها را به صورت پشت سر هم در یک آرایه جدید قرار می‌دهد. تعداد ستون‌های آرایه‌ها باید یکسان باشد. اگر تعداد ستون‌های آرایه‌ها متفاوت باشد، خطا خواهد رخ داد.

در زیر یک نمونه کد برای استفاده از `numpy.row\_stack` آورده شده است:

``` python

import numpy as np

a1 = np.array([1, 2, 3])

a2 = np.array([4, 5, 6])

a3 = np.array([7, 8, 9])

result = np.row\_stack((a1, a2, a3))

print(result)

```

خروجی:

```

[[1 2 3]

[4 5 6]

[7 8 9]]

```

در این مثال، آرایه‌های `a1`، `a2` و `a3` عمودی به هم متصل شده‌اند و توسط `numpy.row\_stack` ترکیب شده‌اند.

-101تابع `numpy.split(ary,indices\_or\_sections,axis)` برای تقسیم یک آرایه به بخش‌های مختلف استفاده می‌شود.

آرگومان `ary` آرایه‌ای است که باید تقسیم شود.

آرگومان `indices\_or\_sections` مکان‌های تقسیم را مشخص می‌کند و می‌تواند یک عدد صحیح یا یک آرایه از اعداد صحیح باشد.

آرگومان `axis` نیز مشخص می‌کند که تقسیم بر روی کدام محور انجام شود (به طور پیش‌فرض مقدار 0 است و تقسیم عمودی انجام می‌شود).

تابع `numpy.split` آرایه اصلی را بر اساس مکان‌های تقسیم مشخص شده تقسیم می‌کند و یک لیست از آرایه‌ها به عنوان خروجی برمی‌گرداند. تعداد بخش‌ها به تعداد مقادیر موجود در `indices\_or\_sections` وابسته است.

در زیر یک نمونه کد برای استفاده از `numpy.split` آورده شده است:

``` python

import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

# تقسیم آرایه به 3 بخش مساوی

result = np.split(arr, 3)

print(result)

```

خروجی:

```

[array([1, 2, 3]), array([4, 5, 6]), array([7, 8, 9, 10])]

```

در این مثال، آرایه `arr` به سه بخش تقسیم شده است و هر بخش به عنوان یک آرایه جداگانه در لیست `result` قرار دارد.

-102تابع `numpy.array\_split(arr,indices\_or\_sections,axis)` نیز برای تقسیم یک آرایه به بخش‌های مختلف استفاده می‌شود، اما به عکس `numpy.split` اجازه می‌دهد تا آرایه را به تعداد بخش‌های مشخص شده تقسیم کند، حتی اگر تعداد بخش‌ها به صورت مساوی نباشد.

آرگومان `ary` آرایه‌ای است که باید تقسیم شود. آ

رگومان `indices\_or\_sections` مکان‌های تقسیم را مشخص می‌کند و می‌تواند یک عدد صحیح یا یک آرایه از اعداد صحیح باشد.

آرگومان `axis` نیز مشخص می‌کند که تقسیم بر روی کدام محور انجام شود (به طور پیش‌فرض مقدار 0 است و تقسیم عمودی انجام می‌شود).

تابع `numpy.array\_split` آرایه اصلی را بر اساس مکان‌های تقسیم مشخص شده تقسیم می‌کند و یک لیست از آرایه‌ها به عنوان خروجی برمی‌گرداند. تعداد بخش‌ها می‌تواند متفاوت باشد، به عنوان مثال اگر تعداد مقادیر موجود در `indices\_or\_sections` بیشتر از تعداد بخش‌هایی باشد که ممکن است تقسیم شود، آرایه اصلی به تعداد بخش‌های مشخص شده تقسیم می‌شود و بخش‌های بیشتر خالی خواهند بود.

در زیر یک نمونه کد برای استفاده از `numpy.array\_split` آورده شده است:

``` python

import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

# تقسیم آرایه به 3 بخش با تعداد مختلف

result = np.array\_split(arr, [2, 5, 8])

print(result)

```

خروجی:

```

[array([1, 2]), array([3, 4, 5]), array([6, 7, 8]), array([9, 10])]

```

-103تابع `numpy.dsplit(ary, indices\_or\_sections )` برای تقسیم یک آرایه چند بعدی بر اساس محور سوم (عمودی) استفاده می‌شود.

آرگومان `ary` آرایه‌ای است که باید تقسیم شود.

`indices\_or\_sections مکان‌های تقسیم را مشخص می‌کند و می‌تواند یک عدد صحیح یا یک آرایه از اعداد صحیح باشد.

تابع `numpy.dsplit` آرایه اصلی را بر اساس مکان‌های تقسیم مشخص شده در محور سوم تقسیم می‌کند و یک لیست از آرایه‌ها به عنوان خروجی برمی‌گرداند. تعداد بخش‌ها می‌تواند متفاوت باشد، به عنوان مثال اگر تعداد مقادیر موجود در `indices\_or\_sections` بیشتر از تعداد بخش‌هایی باشد که ممکن است تقسیم شود، آرایه اصلی به تعداد بخش‌های مشخص شده تقسیم می‌شود و بخش‌های بیشتر خالی خواهند بود.

در زیر یک نمونه کد برای استفاده از `numpy.dsplit` آورده شده است:

``` python

import numpy as np

arr = np.array([[[1, 2, 3],

[4, 5, 6]],

[[7, 8, 9],

[10, 11, 12]]])

# تقسیم آرایه بر اساس محور سوم

result = np.dsplit(arr, 3)

print(result)

```

خروجی:

```

[array([[[1],

[4]],

[[7],

[10]]]),

array([[[2],

[5]],

[[8],

[11]]]),

array([[[3],

[6]],

[[9],

[12]]])]

```

در این مثال، آرایه `arr` یک آرایه سه بعدی است و با استفاده از `np.dsplit` آن را بر اساس محور سوم (عمودی) به سه بخش تقسیم کرده‌ایم. هر بخش حاوی یک ستون از آرایه اصلی است.

-104تابع `numpy.vsplit(ary, indices\_or\_sections )` برای تقسیم یک آرایه چند بعدی بر اساس محور اول (سطری) استفاده می‌شود.

آرگومان `ary` آرایه‌ای است که باید تقسیم شود

`indices\_or\_sections` مکان‌های تقسیم را مشخص می‌کند و می‌تواند یک عدد صحیح یا یک آرایه از اعداد صحیح باشد.

تابع `numpy.vsplit` آرایه اصلی را بر اساس مکان‌های تقسیم مشخص شده در محور اول تقسیم می‌کند و یک لیست از آرایه‌ها به عنوان خروجی برمی‌گرداند. تعداد بخش‌ها می‌تواند متفاوت باشد، به عنوان مثال اگر تعداد مقادیر موجود در `indices\_or\_sections` بیشتر از تعداد بخش‌هایی باشد که ممکن است تقسیم شود، آرایه اصلی به تعداد بخش‌های مشخص شده تقسیم می‌شود و بخش‌های بیشتر خالی خواهند بود.

در زیر یک نمونه کد برای استفاده از `numpy.vsplit` آورده شده است:

``` python

import numpy as np

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

[5, 6, 7, 8],

[9, 10, 11, 12]])

# تقسیم آرایه بر اساس محور اول

result = np.vsplit(arr, 3)

print(result)

```

خروجی:

```

[array([[1, 2, 3, 4]]),

array([[5, 6, 7, 8]]),

array([[ 9, 10, 11, 12]])]

```

در این مثال، آرایه `arr` یک آرایه سه بعدی است و با استفاده از `np.vsplit` آن را بر اساس محور اول (عمودی) به سه بخش تقسیم کرده‌ایم. هر بخش حاوی یک بخش عمودی از آرایه اصلی است.

-105تابع `numpy.tile(A, reps)` برای تکرار یک آرایه یا یک ماتریس در ابعاد مختلف استفاده می‌شود.

آرگومان `A` آرایه اصلی است که می‌خواهید تکرار کنید .`

repsتعداد تکرارها را در هر بعد مشخص می‌کند.

تابع `numpy.tile` آرایه `A` را با توجه به `reps` تکرار می‌کند و یک آرایه جدید برمی‌گرداند. `reps` می‌تواند یک عدد صحیح یا یک tuple از اعداد صحیح باشد. اگر `reps` یک عدد صحیح باشد، آرایه `A` در هر بعد به تعداد مشخص شده تکرار می‌شود. اگر `reps` یک tuple از اعداد صحیح باشد، هر عنصر تکرارها متناسب با بعد مربوطه است.

در زیر یک نمونه کد برای استفاده از `numpy.tile` آورده شده است:

``` python

import numpy as np

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

# تکرار آرایه در ابعاد مختلف

result = np.tile(arr, (2, 3))

print(result)

```

خروجی:

```

[[1 2 3 1 2 3 1 2 3]

[1 2 3 1 2 3 1 2 3]]

```

در این مثال، آرایه `arr` با استفاده از `np.tile` در ابعاد مختلف تکرار شده است. آرایه جدید `result` شامل تکرارهای متناسب با تعداد مشخص شده در `reps` است. در اینجا آرایه `arr` به صورت `[1, 2, 3]` در ابعاد 2 و 3 تکرار شده است.

-106تابع `numpy.repeat(a, repeats, axis=None)` برای تکرار عناصر یک آرایه در طول یک محور مشخص استفاده می‌شود.

آرگومان `a` آرایه اصلی است که می‌خواهید عناصر آن را تکرارکنید.

. `repeats` نشان دهنده تعداد تکرارها برای هر عنصر است و می‌تواند یک عدد صحیح یا آرایه از اعداد صحیح باشد.

آرگومان `axis` نشان می‌دهد کدام محور را برای تکرار عناصر انتخاب کنید، و اگر آن را خالی بگذارید، عمل تکرار بر روی تمام عناصر آرایه انجام می‌شود.

تابع `numpy.repeat` عناصر آرایه `a` را با توجه به `repeats` تکرار می‌کند و یک آرایه جدید برمی‌گرداند. تعداد تکرارها بر روی محور مشخص شده توسط `axis` اعمال می‌شود. اگر `repeats` یک عدد صحیح باشد، همه عناصر آرایه `a` در طول محور مشخص شده تکرار می‌شوند. اگر `repeats` یک آرایه از اعداد صحیح باشد، هر عنصر آرایه `a` متناسب با تعداد تکرارها در آرایه `repeats` تکرار می‌شود.

در زیر یک نمونه کد برای استفاده از `numpy.repeat` آورده شده است:

```python

import numpy as np

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

# تکرار عناصر آرایه

result = np.repeat(arr, 3)

print(result)

```

خروجی:

```

[1 1 1 2 2 2 3 3 3]

```

در این مثال، آرایه `arr` با استفاده از `np.repeat` تکرار شده است. تکرارهای متناسب با تعداد 3 در هر عنصر اعمال شده و آرایه جدید `result` حاوی عناصر تکرار شده است.

-107تابع `numpy.delete(arr, obj, axis=None)` برای حذف عناصر مشخص از یک آرایه استفاده می‌شود.

آرگومان `arr` آرایه اصلی است که می‌خواهید عناصر از آن حذف شوند.

آرگومان `obj` نشان دهنده عناصری است که می‌خواهید حذف شوند. می‌توانید این آرگومان را به صورت یک عدد صحیح یا یک آرایه از اعداد صحیح ارائه دهید.

آرگومان `axis` نشان می‌دهد کدام محور را برای حذف عناصر انتخاب کنید، و اگر آن را خالی بگذارید، آرایه به صورت یک بردار یک بعدی روی محور صفر مرتب می‌شود.

تابع `numpy.delete` عناصر مشخص شده توسط `obj` را از آرایه `arr` حذف می‌کند و یک آرایه جدید بدون این عناصر را برمی‌گرداند.

در زیر یک نمونه کد برای استفاده از `numpy.delete` آورده شده است:

```python

import numpy as np

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

# حذف عنصر در ایندکس 2

result = np.delete(arr, 2)

print(result)

```

خروجی:

```

[1 2 4 5]

```

در این مثال، عنصر با ایندکس 2 در آرایه `arr` با استفاده از `np.delete` حذف شده است و آرایه جدید `result` بدون این عنصر حاوی عناصر مابقی است.

-108تابع `numpy.insert(arr, obj, values, axis=None)` برای درج عناصر در یک آرایه استفاده می‌شود.

آرگومان `arr` آرایه اصلی است که می‌خواهید عناصر در آن درج شوند.

آرگومان `obj` نشان می‌دهد در کدام محور و اندیس‌ها عناصر درج شوند. می‌توانید این آرگومان را به صورت یک عدد صحیح یا یک آرایه از اعداد صحیح ارائه دهید.

آرگومان `values` نشانگر عناصری است که می‌خواهید درج کنید.

و آرگومان `axis` نشان می‌دهد در کدام محور عناصر درج شوند، و اگر آن را خالی بگذارید، آرایه به صورت یک بردار یک بعدی روی محور صفر مرتب می‌شود.

تابع `numpy.insert(arr,obj,values)` عناصر مشخص شده توسط `values` را در آرایه `arr` در مکان‌های مشخص شده توسط `obj` درج می‌کند و یک آرایه جدید با این عناصر درج شده را برمی‌گرداند.

در زیر یک نمونه کد برای استفاده از `numpy.insert` آورده شده است:

```python

import numpy as np

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

# درج عنصر با مقدار 100 در ایندکس 2

result = np.insert(arr, 2, 100)

print(result)

```

خروجی:

```

[ 1 2 100 3 4 5]

```

در این مثال، عنصر با مقدار 100 در ایندکس 2 در آرایه `arr` با استفاده از `np.insert` درج شده است و آرایه جدید `result` حاوی عناصر مابقی به همراه عنصر درج شده است.

-109تابع `numpy.append(arr, values, axis=None)` برای افزودن عناصر به انتهای یک آرایه استفاده می‌شود.

آرگومان `arr` نشان می‌دهد آرایه اصلی است که می‌خواهید عناصر به آن اضافه شوند.

آرگومان `values` نشانگر عناصری است که می‌خواهید به آرایه اضافه کنید.

و آرگومان `axis` نشان می‌دهد در کدام محور عناصر اضافه شوند. اگر آن را خالی بگذارید، آرایه `values` به صورت یک بردار یک بعدی روی محور صفر اضافه می‌شود.

تابع `numpy.append` عناصر `values` را به آرایه `arr` در انتها اضافه می‌کند و یک آرایه جدید حاوی همه عناصر قبلی و عناصر جدید را برمی‌گرداند.

در زیر یک نمونه کد برای استفاده از `numpy.append` آورده شده است:

```python

import numpy as np

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

# اضافه کردن عنصر با مقدار 4 به آرایه

result = np.append(arr, 4)

print(result)

```

خروجی:

```

[1 2 3 4]

```

در این مثال، عنصر با مقدار 4 به آرایه `arr` با استفاده از `np.append` اضافه شده است و آرایه جدید `result` حاوی عناصر قبلی به همراه عنصر جدید است.

-110تابع `numpy.resize(a, new\_shape)` برای تغییر اندازه یک آرایه استفاده می‌شود.

آرگومان `a` نشان می‌دهد آرایه‌ای است که می‌خواهید اندازه آن را تغییر دهید

و `new\_shape` نشانگر ابعاد جدید آرایه است.

در صورتی که `new\_shape` کوچکتر از اندازه اصلی آرایه باشد، آرایه اصلی برش داده می‌شود و فقط بخشی از عناصر حفظ می‌شود. در صورتی که `new\_shape` بزرگتر از اندازه اصلی آرایه باشد، آرایه اصلی با استفاده از روش پر کردن تکرار عناصر موجود تا اندازه جدید گسترش می‌یابد.

در زیر یک نمونه کد برای استفاده از `numpy.resize` آورده شده است:

```python

import numpy as np

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

# تغییر اندازه آرایه به شکل (3, 3)

resized\_arr = np.resize(arr, (3, 3))

print(resized\_arr)

```

خروجی:

```

[[1 2 3]

[4 5 1]

[2 3 4]]

```

در این مثال، آرایه `arr` اصلی شامل اعداد 1 تا 5 است. با استفاده از `np.resize`، آرایه با ابعاد جدید (3، 3) تولید شده است. برای پر کردن این آرایه جدید، عناصر آرایه اصلی به صورت تکراری استفاده شده‌اند.

-111تابع `numpy.trim\_zeros(filt, trim='fb')` برای حذف صفرهای اضافی در ابتدا و انتهای یک آرایه استفاده می‌شود.

آرگومان `filt` نشانگر آرایه‌ای است که می‌خواهید صفرهای اضافی آن را حذف کنید

و آرگومان `trim` نشان می‌دهد که صفرها در کدام قسمت‌های آرایه باید حذف شوند.

مقدار `trim` می‌تواند یکی از موارد زیر را داشته باشد:

- `'f'`: حذف صفرها از ابتدای آرایه

- `'b'`: حذف صفرها از انتهای آرایه

- `'fb'` (پیش‌فرض): حذف صفرها هم از ابتدا و هم از انتهای آرایه

در زیر یک نمونه کد برای استفاده از `numpy.trim\_zeros` آورده شده است:

```python

import numpy as np

filt = np.array([0, 0, 1, 2, 3, 0, 0])

# حذف صفرها از ابتدا و انتهای آرایه

trimmed = np.trim\_zeros(filt, trim='fb')

print(trimmed)

```

خروجی:

```

[1 2 3]

```

در این مثال، آرایه `filt` شامل برخی از صفرها و اعداد غیرصفر است. با استفاده از `np.trim\_zeros`، صفرهای اضافی در ابتدا و انتهای آرایه حذف شده‌اند و آرایه جدید `trimmed` حاوی اعداد غیرصفر می‌باشد.

-112تابع `numpy.unique(ar, return\_index=False, return\_inverse=False, return\_counts=False, axis=None, \*, equal\_nan=True)` برای بازگرداندن مقادیر منحصر به فرد در یک آرایه ویژگی راهنمایی می‌کند.

- آرگومان `ar` نشانگر آرایه ورودی است که می‌خواهید مقادیر منحصر به فرد آن را بدست آورید.

- آرگومان `return\_index` یک مقدار بولی است که در صورت تنظیم برابر True، ایندکس‌های مقادیر منحصر به فرد در آرایه اولیه را برمی‌گرداند.

- آرگومان `return\_inverse` نیز یک مقدار بولی است که اگر برابر True قرار داده شود، معکوس مقادیر منحصر به فرد را بازگردانده و برای هر عنصر آرایه ورودی، ایندکس متناظر آن در آرایه منحصر به فرد را نشان می‌دهد.

- آرگومان `return\_counts` نیز یک مقدار بولی است که در صورت تنظیم برابر True، تعداد تکرار هر مقدار منحصر به فرد را در آرایه ورودی برمی‌گرداند.

- آرگومان `axis` نیز برای تعیین محوری که روی آن عملیات یکتاپذیری انجام می‌شود، استفاده می‌شود.

- آرگومان `equal\_nan` یک مقدار بولی است که در صورت True بودن، NaN (Not a Number) را به عنوان مقدار منحصر به فرد در نظر می‌گیرد.

در زیر یک نمونه کد برای استفاده از `numpy.unique` آورده شده است:

```python

import numpy as np

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

# بدست آوردن مقادیر منحصر به فرد در آرایه

unique\_values = np.unique(arr)

print(unique\_values)

# بدست آوردن مقادیر منحصر به فرد همراه با ایندکس‌ها

unique\_values, indices = np.unique(arr, return\_index=True)

print(unique\_values)

print(indices)

-113تابع `numpy.flip(m, axis=None)` برای برعکس کردن محتوای یک آرایه در یک یا چند محور (axis) مشخص استفاده می‌شود.

آرگومان `m` نشان دهنده آرایه ورودی است

و آرگومان `axis` نشان دهنده محور (محورها)ی که می‌خواهید برعکس شود، می‌باشد.

در صورت عدم مشخص کردن آرگومان `axis`، آرایه `m` به صورت کامل برعکس می‌شود.

در ادامه یک نمونه کد برای استفاده از تابع `numpy.flip` را بررسی می‌کنیم:

```python

import numpy as np

m = np.array([[1, 2, 3],

[4, 5, 6],

[7, 8, 9]])

# برعکس کردن آرایه در محور افقی (محور 0)

result = np.flip(m, axis=0)

print(result)

# Output:

# [[7 8 9]

# [4 5 6]

# [1 2 3]]

# برعکس کردن آرایه در محور عمودی (محور 1)

result = np.flip(m, axis=1)

print(result)

# Output:

# [[3 2 1]

# [6 5 4]

# [9 8 7]]

# برعکس کردن آرایه در هر دو محور (محور 0 و 1)

result = np.flip(m, axis=(0, 1))

print(result)

# Output:

# [[9 8 7]

# [6 5 4]

# [3 2 1]]

```

در نمونه کد بالا، ابتدا یک آرایه 2D به نام `m` تعریف می‌کنیم. سپس با استفاده از تابع `np.flip`، این آرایه را در محور افقی، عمودی و هر دو محور برعکس می‌کنیم و نتایج را چاپ می‌کنیم.

-114تابع `numpy.fliplr(m)` برای برعکس کردن آرایه `m` در محور افقی (از چپ به راست) استفاده می‌شود.

در ادامه یک نمونه کد برای استفاده از تابع `numpy.fliplr` را بررسی می‌کنیم:

```python

import numpy as np

m = np.array([[1, 2, 3],

[4, 5, 6],

[7, 8, 9]])

result = np.fliplr(m)

print(result)

```

Output:

```

[[3 2 1]

[6 5 4]

[9 8 7]]

```

در نمونه کد بالا، ما یک آرایه 2D به نام `m` داریم و با استفاده از تابع `np.fliplr` آن را در محور افقی برعکس می‌کنیم. نتیجه به صورت آرایه جدید چاپ می‌شود. در این حالت، ستون‌های آرایه `m` به صورت برعکس در آرایه نتیجه ظاهر می‌شوند.

-115تابع `numpy.flipud(m)` برای برعکس کردن آرایه `m` در محور عمودی (از بالا به پایین) استفاده می‌شود.

در ادامه یک نمونه کد برای استفاده از تابع `numpy.flipud` را بررسی می‌کنیم:

```python

import numpy as np

m = np.array([[1, 2, 3],

[4, 5, 6],

[7, 8, 9]])

result = np.flipud(m)

print(result)

```

Output:

```

[[7 8 9]

[4 5 6]

[1 2 3]]

```

در نمونه کد بالا، ما یک آرایه 2D به نام `m` داریم و با استفاده از تابع `np.flipud` آن را در محور عمودی برعکس می‌کنیم. نتیجه به صورت آرایه جدید چاپ می‌شود. در این حالت، ردیف‌های آرایه `m` به صورت برعکس در آرایه نتیجه ظاهر می‌شوند.

-116تابع `numpy.reshape(a, newshape, order='C')` برای تغییر ابعاد یک آرایه به شکل مشخص استفاده می‌شود.

این تابع آرایه `a` را با شکل جدیدی که با استفاده از پارامتر `newshape` مشخص می‌شود، بازسازی می‌کند. اختیاری بودن پارامتر `order` نیز اجازه می‌دهد ترتیب عناصر در آرایه نهایی را مشخص کنید.

در ادامه یک نمونه کد برای استفاده از تابع `numpy.reshape` را بررسی می‌کنیم:

```python

import numpy as np

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

# تغییر ابعاد آرایه به (2, 3)

result = np.reshape(a, (2, 3))

print(result)

```

Output:

```

[[1 2 3]

[4 5 6]]

```

در نمونه کد بالا، ما یک آرایه 1D به نام `a` داریم و با استفاده از تابع `np.reshape` آن را به شکل (2, 3) بازسازی می‌کنیم. نتیجه به صورت یک آرایه جدید 2D با شکل مشخص چاپ می‌شود.

-117تابع `numpy.roll(a, shift, axis=None)` برای انجام جابجایی مدور بر روی آرایه `a` استفاده می‌شود. این تابع عناصر آرایه را در جهت مشخص شده تعداد مشخصی با استفاده از پارامتر `shift` می‌چرخاند.

آرگومان `shift` تعیین می‌کند که چند عنصر باید به سمت جلو یا عقب منتقل شوند. مقدار مثبت برای `shift` به معنی جابجایی به سمت جلو و مقدار منفی به معنی جابجایی به سمت عقب است.

آرگومان `axis` نیز تعیین می‌کند که جابجایی بر روی کدام محور انجام شود. اگر `axis` مشخص نشود، آرایه به صورت مسطح در نظر گرفته می‌شود.

در ادامه یک نمونه کد برای استفاده از تابع `numpy.roll` را بررسی می‌کنیم:

```python

import numpy as np

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

# جابجایی به سمت جلو با shift=2

result = np.roll(a, shift=2)

print(result)

# جابجایی به سمت عقب با shift=-1

result = np.roll(a, shift=-1)

print(result)

# جابجایی در محور دوم با shift=1

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

result = np.roll(b, shift=1, axis=1)

print(result)

```

Output:

```

[4 5 1 2 3]

[2 3 4 5 1]

[[3 1 2]

[6 4 5]]

```

در نمونه کد بالا، ما ابتدا یک آرایه 1D به نام `a` داریم و از تابع `np.roll` برای جابجایی عناصر به سمت جلو و عقب استفاده می‌کنیم. سپس یک آرایه 2D به نام `b` را ایجاد می‌کنیم و با تعیین محور دوم، عناصر را در آن محور جابجا می‌کنیم.

-118تابع `numpy.rot90(m, k=1, axes=(0, 1))` برای انجام چرخش ۹۰ درجه در آرایه `m` استفاده می‌شود. این تابع به طور پیش‌فرض آرایه را در جهت مخالف ساعتگرد (counter-clockwise) چرخانده ولی می‌توان با استفاده از پارامتر `k` تعداد چرخش‌های بیشتر را تنظیم کرد.

پارامتر `k` تعیین می‌کند که آرایه چند بار چرخش می‌کند. اگر مقدار `k` مثبت باشد، چرخش در جهت مخالف ساعتگرد انجام می‌شود و اگر مقدار `k` منفی باشد، چرخش در جهت ساعتگرد انجام می‌شود.

پارامتر `axes` نیز تعیین می‌کند که چرخش بر روی کدام محورها اعمال شود. به طور پیش‌فرض، مقدار `(0, 1)` به معنی چرخش بر روی محورهای ۰ و ۱ است.

در ادامه یک نمونه کد برای استفاده از تابع `numpy.rot90` را بررسی می‌کنیم:

```python

import numpy as np

# آرایه 2D

m = np.array([[1, 2, 3],

[4, 5, 6],

[7, 8, 9]])

# چرخش ۹۰ درجه

result = np.rot90(m)

print(result)

# چرخش ۱۸۰ درجه

result = np.rot90(m, k=2)

print(result)

# چرخش ۲۷۰ درجه بر روی محورهای ۱ و ۲

result = np.rot90(m, k=3, axes=(1, 2))

print(result)

```

Output:

```

[[3 6 9]

[2 5 8]

[1 4 7]]

[[9 8 7]

[6 5 4]

[3 2 1]]

[[[3 6 9]

[2 5 8]

[1 4 7]]]

```

-119تابع `numpy.matlib.empty(shape, dtype=None, order='C')` در ماژول `numpy.matlib` برای ایجاد یک آرایه خالی با ابعاد مشخص استفاده می‌شود. این آرایه خالی برای استفاده در محاسبات ماتریسی و عملیات ماتریسی مفید است.

پارامترهای ورودی تابع عبارتند از:

- `shape`: ابعاد مطلوب برای آرایه خروجی. این پارامتر می‌تواند یک عدد صحیح یا یک tuple از اعداد صحیح باشد.

- `dtype` (اختیاری): نوع داده موردنظر برای آرایه خروجی. اگر این پارامتر مشخص نشود، نوع داده پیش‌فرض `float` است.

- `order` (اختیاری): ترتیب ذخیره‌سازی آرایه خروجی. اگر `order` برابر با `'C'` باشد، آرایه به صورت سطر به سطر در حافظه ذخیره می‌شود. اگر `order` برابر با `'F'` باشد، آرایه به صورت ستون به ستون در حافظه ذخیره می‌شود.

در ادامه یک نمونه کد برای استفاده از تابع `numpy.matlib.empty` را بررسی می‌کنیم:

```python

import numpy as np

import numpy.matlib as matlib

# ایجاد یک آرایه خالی با ابعاد 2x3

empty\_array = matlib.empty((2, 3))

print(empty\_array)

# ایجاد یک آرایه خالی با ابعاد 3x3 و نوع داده integer

empty\_array\_int = matlib.empty((3, 3), dtype=int)

print(empty\_array\_int)

# ایجاد یک آرایه خالی با ابعاد 2x2 و ترتیب ذخیره‌سازی ستون به ستون

empty\_array\_column = matlib.empty((2, 2), order='F')

print(empty\_array\_column)

```

Output:

```

[[1. 1. 1.]

[1. 1. 1.]]

[[0 0 0]

[0 0 0]

[0 0 0]]

[[1. 1.]

[1. 1.]]

```-120تابع `numpy.matlib.zeros(shape, dtype=None, order='C')` در ماژول `numpy.matlib` برای ایجاد یک آرایه پر از صفر با ابعاد مشخص استفاده می‌شود. این آرایه پر از صفر برای استفاده در محاسبات ماتریسی و عملیات ماتریسی مفید است.

پارامترهای ورودی تابع عبارتند از:

- `shape`: ابعاد مطلوب برای آرایه خروجی. این پارامتر می‌تواند یک عدد صحیح یا یک tuple از اعداد صحیح باشد.

- `dtype` (اختیاری): نوع داده موردنظر برای آرایه خروجی. اگر این پارامتر مشخص نشود، نوع داده پیش‌فرض `float` است.

- `order` (اختیاری): ترتیب ذخیره‌سازی آرایه خروجی. اگر `order` برابر با `'C'` باشد، آرایه به صورت سطر به سطر در حافظه ذخیره می‌شود. اگر `order` برابر با `'F'` باشد، آرایه به صورت ستون به ستون در حافظه ذخیره می‌شود.

در ادامه یک نمونه کد برای استفاده از تابع `numpy.matlib.zeros` را بررسی می‌کنیم:

```python

import numpy as np

import numpy.matlib as matlib

# ایجاد یک آرایه پر از صفر با ابعاد 2x3

zeros\_array = matlib.zeros((2, 3))

print(zeros\_array)

# ایجاد یک آرایه پر از صفر با ابعاد 3x3 و نوع داده integer

zeros\_array\_int = matlib.zeros((3, 3), dtype=int)

print(zeros\_array\_int)

# ایجاد یک آرایه پر از صفر با ابعاد 2x2 و ترتیب ذخیره‌سازی ستون به ستون

zeros\_array\_column = matlib.zeros((2, 2), order='F')

print(zeros\_array\_column)

```

Output:

```

[[0. 0. 0.]

[0. 0. 0.]]

[[0 0 0]

[0 0 0]

[0 0 0]]

[[0. 0.]

[0. 0.]]

```

-121تابع `numpy.matlib.ones(shape, dtype=None, order='C')` در ماژول `numpy.matlib` برای ایجاد یک آرایه پر از یک با ابعاد مشخص استفاده می‌شود. این آرایه پر از یک برای استفاده در محاسبات ماتریسی و عملیات ماتریسی مفید است.

پارامترهای ورودی تابع عبارتند از:

- `shape`: ابعاد مطلوب برای آرایه خروجی. این پارامتر می‌تواند یک عدد صحیح یا یک tuple از اعداد صحیح باشد.

- `dtype` (اختیاری): نوع داده موردنظر برای آرایه خروجی. اگر این پارامتر مشخص نشود، نوع داده پیش‌فرض `float` است.

- `order` (اختیاری): ترتیب ذخیره‌سازی آرایه خروجی. اگر `order` برابر با `'C'` باشد، آرایه به صورت سطر به سطر در حافظه ذخیره می‌شود. اگر `order` برابر با `'F'` باشد، آرایه به صورت ستون به ستون در حافظه ذخیره می‌شود.

در ادامه یک نمونه کد برای استفاده از تابع `numpy.matlib.ones` را بررسی می‌کنیم:

```python

import numpy as np

import numpy.matlib as matlib

# ایجاد یک آرایه پر از یک با ابعاد 2x3

ones\_array = matlib.ones((2, 3))

print(ones\_array)

# ایجاد یک آرایه پر از یک با ابعاد 3x3 و نوع داده integer

ones\_array\_int = matlib.ones((3, 3), dtype=int)

print(ones\_array\_int)

# ایجاد یک آرایه پر از یک با ابعاد 2x2 و ترتیب ذخیره‌سازی ستون به ستون

ones\_array\_column = matlib.ones((2, 2), order='F')

print(ones\_array\_column)

```

Output:

```

[[1. 1. 1.]

[1. 1. 1.]]

[[1 1 1]

[1 1 1]

[1 1 1]]

[[1. 1.]

[1. 1.]]

```

-122تابع `numpy.matlib.eye(n, M=None, k=0, dtype=float, order='C')` در ماژول `numpy.matlib` برای ایجاد یک آرایه ماتریس همانی (identity matrix) با اندازه و مشخصات مشخص استفاده می‌شود. آرایه ماتریس همانی یک ماتریس مربع است که قطر آن پر از عناصر یک و سایر عناصر آن صفر است.

پارامترهای ورودی تابع عبارتند از:

- `n`: اندازه ماتریس همانی در جهت عمودی و افقی.

- `M` (اختیاری): اندازه ماتریس همانی در جهت افقی. اگر این پارامتر مشخص نشود، به صورت پیش‌فرض برابر با `n` در نظر گرفته می‌شود.

- `k` (اختیاری): انتقال ماتریس همانی در رابطه با قطر اصلی. مقدار صفر برای ماتریس همانی اصلی است و مقادیر مثبت و منفی برای جا به جایی ماتریس به صورت قطری به سمت بالا و پایین انتقال می‌دهد.

- `dtype` (اختیاری): نوع داده موردنظر برای آرایه خروجی. نوع داده پیش‌فرض `float` است.

- `order` (اختیاری): ترتیب ذخیره‌سازی آرایه خروجی. اگر `order` برابر با `'C'` باشد، آرایه به صورت سطر به سطر در حافظه ذخیره می‌شود. اگر `order` برابر با `'F'` باشد، آرایه به صورت ستون به ستون در حافظه ذخیره می‌شود.

در ادامه یک نمونه کد برای استفاده از تابع `numpy.matlib.eye` را بررسی می‌کنیم:

```python

import numpy as np

import numpy.matlib as matlib

# ایجاد یک ماتریس همانی 3x3

eye\_matrix = matlib.eye(3)

print(eye\_matrix)

# ایجاد یک ماتریس همانی 4x5

eye\_matrix\_custom = matlib.eye(4, 5)

print(eye\_matrix\_custom)

ماتریس همانی یک ماتریس مربع است که قطر اصلی آن پر از یک و سایر عناصر آن صفر هستند. با استفاده از تابع `numpy.matlib.eye` می‌توانید ماتریس همانی با ابعاد و خصوصیات دلخواه ایجاد کنید.

ادامه نمونه کد:

```python

import numpy as np

import numpy.matlib as matlib

# ایجاد یک ماتریس همانی 2x2 با انتقال قطر به بالا (k=1)

eye\_matrix\_shifted = matlib.eye(2, k=1)

print(eye\_matrix\_shifted)

# ایجاد یک ماتریس همانی 3x3 با نوع داده integer

eye\_matrix\_int = matlib.eye(3, dtype=int)

print(eye\_matrix\_int)

# ایجاد یک ماتریس همانی 4x4 با ترتیب ذخیره‌سازی ستون به ستون

eye\_matrix\_column = matlib.eye(4, order='F')

print(eye\_matrix\_column)

```

Output:

```

[[0. 1. 0.]

[0. 0. 1.]

[0. 0. 0.]]

[[1 0 0]

[0 1 0]

[0 0 1]]

[[1. 0. 0. 0.]

[0. 1. 0. 0.]

[0. 0. 1. 0.]

[0. 0. 0. 1.]]

```

در نمونه کد بالا، ما ماتریس همانی را با خصوصیات مختلف ایجاد کرده‌ایم. همچنین، توجه داشته باشید که این تابع در ماژول `numpy.matlib` وجود دارد و برای استفاده از آن باید ابتدا ماژول `numpy.matlib` را وارد کنید.

-123تابع `numpy.matlib.identity(n)` برای ایجاد یک ماتریس همانی با ابعاد n استفاده می‌شود. ماتریس همانی یک ماتریس مربع است که قطر اصلی آن پر از یک و سایر عناصر آن صفر هستند.

نمونه کد:

```python

import numpy as np

import numpy.matlib as matlib

# ایجاد یک ماتریس همانی 3x3

identity\_matrix = matlib.identity(3)

print(identity\_matrix)

```

خروجی:

```

[[1. 0. 0.]

[0. 1. 0.]

[0. 0. 1.]]

```

-124تابع `numpy.matlib.repmat(a,m,n)` برای تکرار یک آرایه `a` در یک الگوی ماتریسی مشخص استفاده می‌شود. آرگومان `a` نشان دهنده آرایه اولیه است که قرار است تکرار شود. آرگومان `m` نشان دهنده تعداد تکرار در جهت ردیف‌ها (محور اول) است و آرگومان `n` نشان دهنده تعداد تکرار در جهت ستون‌ها (محور دوم) است.

خروجی تابع `numpy.matlib.repmat` نیز یک ماتریس با ابعاد `m` برابر تعداد ردیف‌ها و `n` برابر تعداد ستون‌ها خواهد بود. المان‌های ماتریس خروجی برابر المان‌های آرایه `a` خواهند بود.

-125با استفاده از تابع `numpy.matlib.empty(shape,dtype,order)` می‌توانید یک آرایه‌ی خالی (بدون مقدار) با ابعاد مشخص ایجاد کنید. آرگومان‌های تابع عبارتند از:

- `shape`: ابعاد آرایه مورد نظر به صورت یک tuple.

- `dtype` (اختیاری): نوع داده‌های آرایه (برای مثال، `int`، `float` و غیره).

- `order` (اختیاری): ترتیب محل ذخیره‌سازی داده‌ها در حافظه (برای مثال، 'C' برای نگاشت به حافظه به صورت پیاپی و 'F' برای نگاشت به حافظه به صورت ستونی).

این تابع یک آرایه خالی به ابعاد مشخص ایجاد می‌کند. داده‌های آرایه به صورت تصادفی در حافظه قرار می‌گیرند و مقادیر آن‌ها تعیین نمی‌شوند. برای مقداردهی به آرایه، شما باید به صورت دستی مقادیر را وارد کنید.

در زیر یک مثال از استفاده از `numpy.matlib.empty` را می‌بینید:

```python

import numpy as np

import numpy.matlib as matlib

shape = (3, 4)

dtype = np.int

order = 'C'

arr = matlib.empty(shape, dtype, order)

print(arr)

```

این کد یک آرایه ۳ در ۴ خالی با نوع داده `int` و ترتیب `C` ایجاد می‌کند و آن را چاپ می‌کند. مقادیر داخل آرایه تعیین نشده‌اند و به صورت تصادفی در حافظه قرار می‌گیرند.

-126با استفاده از تابع `numpy.matlib.rand(\*shape)` می‌توانید یک آرایه تصادفی با ابعاد مشخص ایجاد کنید. این تابع مقادیر تصادفی از توزیع یکنواخت بین 0 و 1 را تولید می‌کند. در زیر یک مثال از استفاده از این تابع را می‌بینید:

```python

import numpy as np

import numpy.matlib as matlib

shape = (2, 3)

arr = matlib.rand(\*shape)

print(arr)

```

خروجی:

```

[[0.60371703 0.34812719 0.47629607]

[0.24788283 0.10870317 0.79163366]]

```

در این مثال، یک آرایه تصادفی به ابعاد 2 در 3 ایجاد می‌شود و مقادیر آن بین 0 و 1 قرار دارند. توجه کنید که مقادیر تصادفی هر بار که برنامه اجرا می‌شود، ممکن است متفاوت باشند.

-127تابع `numpy.matlib.randn(\*shape)` به شما امکان می‌دهد یک آرایه تصادفی با ابعاد مشخص را با استفاده از توزیع نرمال (گاوسی) ایجاد کنید. این تابع مقادیر تصادفی با توزیع نرمال با میانگین صفر و واریانس یک تولید می‌کند. در زیر یک مثال از استفاده از این تابع را می‌بینید:

```python

import numpy as np

import numpy.matlib as matlib

shape = (2, 3)

arr = matlib.randn(\*shape)

print(arr)

```

خروجی:

```

[[-1.12011443 0.70420353 -1.16735406]

[-0.20464079 -0.49203553 0.27874858]]

```

در این مثال، یک آرایه تصادفی با ابعاد 2 در 3 ایجاد می‌شود و مقادیر آن با توزیع نرمال تولید می‌شوند. توجه کنید که مقادیر تصادفی هر بار که برنامه اجرا می‌شود، ممکن است متفاوت باشند.

توابع زیر از کتابخانه NumPy جزو توابع عملیات باینری (Binary operations) و عملیات بیتی (Elementwise bit operations) می‌باشند:

. `bitwise\_and(x1, x2, /[, out, where, casting, ...])`:-128

این تابع بیتی AND بین دو آرایه `x1` و `x2` را انجام می‌دهد. برای هر جفت عنصر در هر دو آرایه، تابع این عملیات را انجام می‌دهد و نتیجه را در آرایه خروجی `out` ذخیره می‌کند. اگر آرایه خروجی `out` ارائه نشده باشد، یک آرایه جدید برای نتیجه ایجاد می‌شود. مکان آرایه خروجی در صورت ارائه، در پارامتر `out` مشخص می‌شود. علاوه بر این، می‌توانید شرایطی را برای انجام عملیات مشخص کنید با استفاده از پارامترهای `where`, `casting` و سایر پارامترهای قابل تنظیم دیگر.

-. `bitwise\_or(x1, x2, /[, out, where, casting, ...])`:-129

این تابع بیتی OR بین دو آرایه `x1` و `x2` را انجام می‌دهد. برای هر جفت عنصر در هر دو آرایه، تابع این عملیات را انجام می‌دهد و نتیجه را در آرایه خروجی `out` ذخیره می‌کند. مکان آرایه خروجی در صورت ارائه، در پارامتر `out` مشخص می‌شود. همچنین، می‌توانید با استفاده از پارامترهای `where`, `casting` و سایر پارامترهای قابل تنظیم دیگر شرایط عملیات را مشخص کنید.

. `bitwise\_xor(x1, x2, /[, out, where, casting, ...])`:-130

این تابع بیتی XOR (OR منحصر به فرد) بین دو آرایه `x1` و `x2` را انجام می‌دهد. برای هر جفت عنصر در هر دو آرایه، تابع این عملیات را انجام می‌دهد و نتیجه را در آرایه خروجی `out` ذخیره می‌کند. مکان آرایه خروجی در صورت ارائه، در پارامتر `out` مشخص می‌شود. شرایط عملیات نیز می‌تواند با استفاده از پارامترهای `where`, `casting` و سایر پارامترهای قابل تنظیم دیگر مشخص شود.

. `invert(x, /[, out, where, casting, order, ...])`:-131

این تابع بیتی NOT (منطقی عملگر NOT) را بر روی آرایه `x` اعمال می‌کند. برای هر عنصر در آرایه، تابع این عملیات را انجام می‌دهد و نتیجه را در آرایه خروجی `out` ذخیره می‌کند. مکان آرایه خروجی در صورت ارائه، در پارامتر `out` مشخص می‌شود. شرایط عملیات نیز می‌تواند با استفاده از پارامترهای `where`, `casting` و سایر پارامترهای قابل تنظیم دیگر مشخص شود.

. `left\_shift(x1, x2, /[, out, where, casting, ...])`:-132

این تابع عملیات شیفت به چپ بیت‌های آرایه `x1` را با تعداد بیت‌های مشخص شده در آرایه `x2` انجام می‌دهد. برای هر جفت عنصر در هر دو آرایه، تابع این عملیات را انجام می‌دهد و نتیجه را در آرایه خروجی `out` ذخیره می‌کند. مکان آرایه خروجی در صورت ارائه، در پارامتر `out` مشخص می‌شود. شرایط عملیات نیز می‌تواند با استفاده از پارامترهای `where`, `casting` و سایر پارامترهای قابل تنظیم دیگر مشخص شود.

. `right\_shift(x1, x2, /[, out, where, ...])`:-133

این تابع عملیات شیفت به راست بیت‌های آرایه `x1` را با تعداد بیت‌های مشخص شده در آرایه `x2` انجام می‌دهد. برای هر جفت عنصر در هر دو آرایه، تابع این عملیات را انجام می‌دهد و نتیجه را در آرایه خروجی `out` ذخیره می‌کند. مکان آرایه خروجی در صورت ارائه، در پارامتر `out` مشخص می‌شود. شرایط عملیات نیز می‌تواند با استفاده از پارامترهای `where` و سایر پارامترهای قابل تنظیم دیگر مشخص شود.

این توابع برای انجام عملیات بیتی و منطقی بر روی آرایه‌های NumPy بسیار مفید هستند و قابلیت‌های گسترده‌ای برای تنظیم و کنترل عملیات را فراهم می‌کنند.

توابع زیر از کتابخانه NumPy نیز جزو توابع مربوط به بسته بندی بیت (Bit packing) و فرمت‌بندی خروجی (Output formatting) می‌باشند:

. `packbits(a, /[, axis, bitorder])`:-134

این تابع بسته بندی بیت از آرایه `a` را انجام می‌دهد. برای هر عنصر در آرایه، تابع این عملیات را انجام می‌دهد و بیت‌های آرایه را به صورت فشرده بسته بندی می‌کند. این عملیات می‌تواند در راستای کاهش حجم حافظه مصرفی برای ذخیره‌سازی داده‌های بیتی مفید باشد. محور `axis` مشخص می‌کند که آیا عملیات بسته بندی بیت بر روی همه ابعاد آرایه یا فقط روی یک محور خاص اعمال شود. پارامتر `bitorder` نیز شیوه مرتب‌سازی بیت‌ها را تعیین می‌کند.

. `unpackbits(a, /[, axis, count, bitorder])`:-135

این تابع بازگشت بیت از بسته بندی بیت را انجام می‌دهد. به طور کلی، این تابع بسته بندی بیت‌ها را باز می‌کند و بیت‌های فشرده شده را به بیت‌های اصلی تبدیل می‌کند. پارامتر `axis` مشخص می‌کند که عملیات بازگشت بیت بر روی کدام محور انجام شود. همچنین، می‌توان با استفاده از پارامتر `count` تعداد بیت‌های مورد استفاده در بازگشت بیت را محدود کرد و با استفاده از پارامتر `bitorder` نیز شیوه مرتب‌سازی بیت‌ها را تعیین کرد.

. `binary\_repr(num[, width])`:-136

این تابع یک عدد صحیح `num` را به رشته‌ای از نمایش دودویی آن تبدیل می‌کند. با استفاده از پارامتر `width` می‌توان طول رشته خروجی را تعیین کرد. این تابع مفید است زمانی که نیاز به نمایش یک عدد در فرمت دودویی با طول ثابت دارید.

137-

. `numpy.bitwise\_and(x1, x2, /, out=None, \*, where=True, casting='same\_kind', = order='K', dtype=None, subok=True) = <ufunc 'bitwise\_and'>`:

این تابع عملیات بیتی AND بین دو آرایه `x1` و `x2` را انجام می‌دهد. تفاوت این تابع با تابع `bitwise\_and` که قبلاً توضیح داده شد، در این است که `numpy.bitwise\_and` یک تابع یونیورسال (ufunc) است که بر روی آرایه‌ها اعمال می‌شود. این به معنی این است که می‌توان آن را بر روی آرایه‌های چندبعدی و با ابعاد مختلف نیز اعمال کرد. پارامترهای مختلفی مانند `out`، `where`، `casting`، `dtype` و غیره برای تنظیم و کنترل عملیات مشخص شده در تابع موجود هستند.

توابع زیر از کتابخانه NumPy جزو توابع بیتی (Bitwise functions) هستند:

138-

. `numpy.bitwise\_or(x1, x2, /, out=None, \*, where=True, casting='same\_kind',

order='K', dtype=None, subok=True(= <ufunc 'bitwise\_or'>`:

این تابع عملیات بیتی OR بین دو آرایه `x1` و `x2` را انجام می‌دهد. واژه OR در اینجا به معنای انجام یک عملیات بیتی بر روی هر بیت مربوطه در آرایه‌ها است. نتیجه این عملیات بیتی را در آرایه خروجی `out` ذخیره می‌کند. پارامترهای `where`، `casting`، `dtype` و سایر پارامترهای قابل تنظیم دیگر برای کنترل و تنظیم عملیات مورد استفاده قرار می‌گیرند.

139-

. `numpy.bitwise\_xor(x1, x2, /, out=None, \*, where=True, casting='same\_kind', order='K', dtype=None, subok=True) = <ufunc 'bitwise\_xor'>`:

این تابع عملیات بیتی XOR (OR انحصاری) بین دو آرایه `x1` و `x2` را انجام می‌دهد. عملیات XOR بر روی هر بیت مربوطه در آرایه‌ها اعمال می‌شود و نتیجه را در آرایه خروجی `out` ذخیره می‌کند. مشابه سایر توابع بیتی، می‌توان با استفاده از پارامترهای `where`، `casting`، `dtype` و سایر پارامترهای قابل تنظیم دیگر عملیات را شخصی‌سازی کرد.

140-

. `numpy.invert(x, /, out=None, \*, where=True, casting='same\_kind', order='K', dtype=None, subok=True[, signature]) = <ufunc 'invert'>`:

این تابع عملیات بیتی NOT (منطقی عملگر NOT) را بر روی آرایه `x` اعمال می‌کند. به طور کلی، عملگر NOT یک عملیات بیتی است که بیت‌های آرایه را در مقابله با یک تغییر می‌دهد. نتیجه این عملیات را درآرایه خروجی `out` ذخیره می‌کند. مانند سایر توابع بیتی، این تابع نیز از پارامترهای `where`، `casting`، `dtype` و سایر پارامترهای قابل تنظیم دیگر برای کنترل عملیات استفاده می‌کند.

141-

. `numpy.left\_shift(x1, x2, /, out=None, \*, where=True, casting='same\_kind', order='K', dtype=None, subok=True]) = <ufunc 'left\_shift'>`:

این تابع عملیات شیفت به چپ بیت‌های آرایه `x1` را با تعداد شیفت داده شده توسط `x2` انجام می‌دهد. به طور کلی، عملیات شیفت به چپ بیت‌ها یک تغییر مکان بیت‌ها به سمت چپ است. نتیجه این عملیات را در آرایه خروجی `out` ذخیره می‌کند. پارامترهای `where`، `casting`، `dtype` و سایر پارامترهای قابل تنظیم دیگر نیز برای تنظیم عملیات استفاده می‌شوند.

توابع زیر از کتابخانه NumPy جزو توابع بیتی (Bitwise functions) هستند:

142-

. `numpy.right\_shift(x1, x2, /, out=None, \*, where=True, casting='same\_kind', order='K', dtype=None, subok=True]) = <ufunc 'right\_shift'>`:

این تابع عملیات شیفت به راست بیت‌های آرایه `x1` را با تعداد شیفت داده شده توسط `x2` انجام می‌دهد. عملیات شیفت به راست بیت‌ها معادل با تقسیم عدد بیتی بر ۲ به توان تعداد شیفت می‌باشد. نتیجه این عملیات را در آرایه خروجی `out` ذخیره می‌کند. پارامترهای `where`، `casting`، `dtype` و سایر پارامترهای قابل تنظیم دیگر نیز برای کنترل عملیات استفاده می‌شوند.

143-

. `numpy.packbits(a, /, axis=None, bitorder='big')`:

این تابع بسته‌بندی بیت‌های آرایه `a` را انجام می‌دهد و آنها را به صورت فشرده ذخیره می‌کند. هر بیت در آرایه ورودی `a` با یک المان در آرایه خروجی مطابقت دارد. پارامتر `axis` نشان می‌دهد که عملیات بسته‌بندی بیت‌ها بر روی کدام محور انجام شود. پارامتر `bitorder` نیز شیوه مرتب‌سازی بیت‌ها را تعیین می‌کند که می‌تواند "big" یا "little" باشد.

144-

. `numpy.unpackbits(a, /, axis=None, count=None, bitorder='big')`:

این تابع عمل برداشت بیت‌ها را از بسته‌بندی بیت انجام می‌دهد. یعنی بیت‌های فشرده شده را به بیت‌های اصلی تبدیل می‌کند. پارامتر `axis` نشان می‌دهد که عملیات برداشت بیت‌ها بر روی کدام محور انجام شود. با استفاده از پارامتر `count`، می‌توان تعداد بیت هایی که باید برداشت شوند را مشخص کرد. پارامتر `bitorder` نیز شیوه مرتب‌سازی بیت‌ها را تعیین می‌کند که می‌تواند "big" یا "little" باشد.

145-

. `numpy.binary\_repr(num, width=None)`:

این تابع عدد صحیح `num` را به رشته دودویی تبدیل می‌کند. اختیاری، می‌توان طول رشته دودویی مورد نظر را با استفاده از پارامتر `width` مشخص کرد. این تابع مفید است زمانی که نیاز به نمایش یک عدد صحیح در فرمت دودویی با طول ثابت دارید.

در زیر توضیحات و آرگومان‌های ورودی توابع مربوط به عملیات روی رشته‌ها در کتابخانه NumPy آمده است:

146-

. `add(x1, x2)`: این تابع دو رشته `x1` و `x2` را به صورت عضو به عضو جمع می‌کند.

147-

. `multiply(a, i)`: این تابع رشته `a` را `i` بار تکرار می‌کند.

148-

. `mod(a, values)`: این تابع هر کاراکتر در رشته `a` را به عنصر متناظر در آرایه `values` می‌ماند.

149-

. `capitalize(a)`: این تابع اولین حرف رشته `a` را به حرف بزرگ تبدیل می‌کند و بقیه حروف را به حروف کوچک.

150-

. `center(a, width[, fillchar])`: این تابع رشته `a` را در وسط یک رشته با عرض `width` قرار می‌دهد. حرف `fillchar` به عنوان حرف پر کننده مورد استفاده قرار می‌گیرد (پیش‌فرض خالی).

151-

. `decode(a[, encoding, errors])`: این تابع رشته `a` را با استفاده از کدگذاری `encoding` به رشته اصلی تبدیل می‌کند. پارامتر `errors` مشخص می‌کند چگونه با خطاهای کدگذاری برخورد کند (پیش‌فرض "strict").

152-

. `encode(a[, encoding, errors])`: این تابع رشته `a` را با استفاده از کدگذاری `encoding` به رشته کد شده تبدیل می‌کند. پارامتر `errors` مشخص می‌کند چگونه با خطاهای کدگذاری برخورد کند (پیش‌فرض "strict").

153-. `expandtabs(a[, tabsize])`: این تابع تمام تب‌ها در رشته `a` را با فاصله‌های مناسب جایگزین می‌کند. `tabsize` تعداد فاصله‌هایی است که هر تب نمایش داده می‌شود (پیش‌فرض 8).

154-. `join(sep, seq)`: این تابع عنصرهای `seq` را به کمک جداکننده `sep`

به یک رشته متصل می‌کند.

157-. `ljust(a, width[, fillchar])`: این تابع رشته `a` را در سمت چپ با حرف `fillchar` (پیش‌فرض خالی) به عرض `width` قرار می‌دهد.

156-. `lower(a)`: این تابع رشته `a` را به حروف کوچک تبدیل می‌کند.

157-. `lstrip(a[, chars])`: این تابع حروف را از سمت چپ رشته `a` حذف می‌کند. حروف قابل حذف با استفاده از `chars` مشخص می‌شوند (پیش‌فرض فاصله).

158-. `partition(a, sep)`: این تابع رشته `a` را براساس جداکننده `sep` به سه بخش تقسیم می‌کند: بخش قبل از جداکننده، خود جداکنده و بخش بعد از جداکنده.

159-. `replace(a, old, new[, count])`: این تابع تمام زیررشته‌های `old` در رشته `a` را با رشته `new` جایگزین می‌کند. `count` تعداد زیررشته‌هایی است که قرار است جایگزین شوند (پیش‌فرض همه زیررشته‌ها).

160-. `rjust(a, width[, fillchar])`: این تابع رشته `a` را در سمت راست با حرف `fillchar` (پیش‌فرض خالی) به عرض `width` قرار می‌دهد.

161-. `rpartition(a, sep)`: این تابع رشته `a` را براساس جداکننده `sep` به سه بخش تقسیم می‌کند: بخش قبل از جداکننده، خود جداکنده و بخش بعد از جداکنده (شروع از سمت راست).

162-. `rsplit(a[, sep, maxsplit])`: این تابع رشته `a` را براساس جداکنده `sep` به لیستی از زیررشته‌ها تقسیم می‌کند (شروع از سمت راست). `maxsplit` حداکثر تعداد تقسیم‌بندی‌ها را مشخص می‌کند.

163-. `rstrip(a[, chars])`: این تابع حروف را از سمت راست رشته ‘a’حذف می‌کند. حروف قابل حذف با استفاده از `chars` مشخص می‌شوند (پیش‌فرض فاصله).

164-. `split(a[, sep, maxsplit])`: این تابع رشته `a` را براساس جداکنده `sep` به لیستی از زیررشته‌ها تقسیم می‌کند. `maxsplit` حداکثر تعداد تقسیم‌بندی‌ها را مشخص می‌کند.

165-. `splitlines(a[, keepends])`: این تابع رشته `a` را براساس خطوط تقسیم می‌کند و لیستی از خطوط را برمی‌گرداند. پارامتر `keepends` مشخص می‌کند آیا حرف پایان خط (`\n` یا `\r\n`) در خروجی نگه داشته شود یا خیر (پیش‌فرض False).

166-. `strip(a[, chars])`: این تابع حروف را از دو طرف رشته `a` حذف می‌کند. حروف قابل حذف با استفاده از `chars` مشخص می‌شوند (پیش‌فرض فاصله).

167-. `swapcase(a)`: این تابع حروف رشته `a` را برعکس کوچکی و بزرگی تبدیل می‌کند.

168-. `title(a)`: این تابع رشته `a` را با تبدیل کردن حروف اول هر کلمه به حرف بزرگ و بقیه حروف به حروف کوچک، به عنوان عنوان تغییر می‌دهد.

169-. `translate(a, table[, deletechars])`: این تابع رشته `a` را با استفاده از یک جدول ترجمه (`table`) به رشته جدید تبدیل می‌کند. حروف موجود در `deletechars` حذف می‌شوند (پیش‌فرض خالی).

170-. `upper(a)`: این تابع رشته `a` را به حروف بزرگ تبدیل می‌کند.

171-. `zfill(a, width)`: این تابع رشته `a` را با صفرها در سمت چپ تا رسیدن به طول `width` پر می‌کند.

172-. `equal(x1, x2)`:

- `x1`: آرایه ورودی اول

- `x2`: آرایه ورودی دوم

تابع `equal` بررسی می‌کند که آیا عناصر آرایه‌های `x1` و `x2` با هم برابر هستند. آرایه خروجی شامل مقادیر بولین است که نشان می‌دهد هر عنصر از `x1` با متناظر خود در `x2` برابر است یا خیر.

173-. `not\_equal(x1, x2)`:

- `x1`: آرایه ورودی اول

- `x2`: آرایه ورودی دوم

تابع `not\_equal` بررسی می‌کند که آیا عناصر آرایه‌های `x1` و `x2` با هم برابر نیستند. آرایه خروجی شامل مقادیر بولین است که نشان می‌دهد هر عنصر از `x1` با متناظر خود در `x2` برابر نیست یا خیر.

174-. `greater\_equal(x1, x2)`:

- `x1`: آرایه ورودی اول

- `x2`: آرایه ورودی دوم

تابع `greater\_equal` بررسی می‌کند که آیا عناصر آرایه `x1` بزرگتر یا مساوی عناصر آرایه `x2` هستند. آرایه خروجی شامل مقادیر بولین است که نشان می‌دهد هر عنصر از `x1` بزرگتر یا مساوی متناظر خود در `x2` است یا خیر.

175-. `less\_equal(x1, x2)`:

- `x1`: آرایه ورودی اول

- `x2`: آرایه ورودی دوم

تابع `less\_equal` بررسی می‌کند که آیا عناصر آرایه `x1` کوچکتر یا مساوی عناصر آرایه `x2` هستند. آرایه خروجی شامل مقادیر بولین است که نشان می‌دهد هر عنصر از `x1` کوچکتر یا مساوی متناظر خود در `x2` است یا خیر.

176-. `greater(x1, x2)`:

- `x1`: آرایه ورودی اول

- `x2`: آرایه ورودی دوم

تابع `greater` بررسی می‌کند که آیا عناصر آرایه `x1` بزرگتر از عناصر آرایه `x2` هستند. آرایه خروجی شامل مقادیر بولین است که نشان می‌دهد هر عنصر از `x1` بزرگتر از متناظر خود در `x2` است یا خیر.

177-. `compare\_chararrays(a1, a2, cmp, rstrip)`:

- `a1`: آرایه ورودی اول (رشته‌ها)

- `a2`: آرایه ورودی دوم (رشته‌ها)

- `cmp`: یک رشته که نحوه مقایسه را مشخص می‌کند. این پارامتر می‌تواند 'eq'، 'ne', 'lt'، 'gt'، 'le' یا 'ge' باشد.

- `rstrip`: یک بولین که مشخص می‌کند آیا کاراکترهای فاصله در انتهای رشته‌ها باید حذف شوند یا خیر.

تابع `compare\_chararrays` دو آرایه رشته را مقایسه می‌کند. نحوه مقایسه با استفاده از پارامتر `cmp` مشخص می‌شود (مثلاً برابری، عدم برابری، بزرگتر بودن و غیره). اگر `rstrip` برابر True باشد، کاراکترهای فاصله در انتهای رشته‌ها حذف می‌شوند. آرایه خروجی شامل مقادیر بولین است که نشان می‌دهد هر رشته از `a1` با متناظر خود در `a2` برابر است یا خیر.

178-. `count(a, sub[, start, end])`:

- `a`: رشته ورودی

- `sub`: رشته جستجو شده

- `start` (اختیاری): اندیس شروع جستجو در رشته `a`

- `end` (اختیاری): اندیس پایان جستجو در رشته `a`

تابع `count` تعداد ظهور رشته `sub` در رشته `a` را برمی‌گرداند. این تابع می‌تواند از اندیس `start` شروع و تا اندیس `end` به جستجوی رشته `sub` بپردازد. تعداد ظهورهای رشته `sub` در رشته `a` به عنوان خروجی برگشت داده می‌شود.

179-. `endswith(a, suffix[, start, end])`:

- `a`: رشته ورودی

- `suffix`: رشته مورد بررسی برای پایان رشته `a`

- `start` (اختیاری): اندیس شروع بررسی در رشته `a`

- `end` (اختیاری): اندیس پایان بررسی در رشته `a`

تابع `endswith` بررسی می‌کند که رشته `a` آیا با رشته `suffix` به پایان می‌رسد یا خیر. اگر رشته `a` با رشته `suffix` به پایان برسد، مقدار True برگردانده می‌شود؛ در غیر این صورت مقدار False برگردانده می‌شود. این تابع می‌تواند از اندیس `start` شروع و تا اندیس `end` بررسی را انجام دهد.

180. `find(a, sub[, start, end])`:

- `a`: رشته ورودی

- `sub`: رشته جستجو شده

- `start` (اختیاری): اندیس شروع جستجو در رشته `a`

- `end` (اختیاری): اندیس پایان جستجو در رشته `a`

تابع `find` اندیس اولین رخداد رشته `sub` در رشته `a` را برمی‌گرداند. اگر رشته `sub` در رشته `a` یافت نشود، مقدار -1 برگردانده می‌شود.

این تابع می‌تواند از اندیس `start` شروع و تا اندیس `end` جستجو را انجام دهد.

181-. `index(a, sub[, start, end])`:

- `a`: رشته ورودی

- `sub`: رشته جستجو شده

- `start` (اختیاری): اندیس شروع جستجو در رشته `a`

- `end` (اختیاری): اندیس پایان جستجو در رشته `a`

تابع `index` اندیس اولین رخداد رشته `sub` در رشته `a` را برمی‌گرداند. اگر رشته `sub` در رشته `a` یافت نشود، یک خطا از نوع `ValueError` بوجود می‌آید. این تابع می‌تواند از اندیس `start` شروع و تا اندیس `end` جستجو را انجام دهد.

182-. `isalpha(a)`:

- `a`: رشته ورودی

تابع `isalpha` بررسی می‌کند که تمام کاراکترهای رشته `a` از نوع حروف الفبا هستند یا خیر. اگر تمام کاراکترهای رشته `a` حروف الفبا باشند، مقدار True برگردانده می‌شود؛ در غیر این صورت مقدار False برگردانده می‌شود.

183-. `isalnum(a)`:

- `a`: رشته ورودی

تابع `isalnum` بررسی می‌کند که تمام کاراکترهای رشته `a` از نوع حروف الفبا و/یا اعداد هستند یا خیر. اگر تمام کاراکترهای رشته `a` حروف الفبا و/یا اعداد باشند، مقدار True برگردانده می‌شود؛ در غیر این صورت مقدار False برگردانده می‌شود.

184-. `isdecimal(a)`:

- `a`: رشته ورودی

تابع `isdecimal` بررسی می‌کند که تمام کاراکترهای رشته `a` از نوع اعداد اعشاری هستند یا خیر. اگر تمام کاراکترهای رشته `a` اعداد اعشاری باشند، مقدار True برگردانده می‌شود؛ در غیر این صورت مقدار False برگردانده می‌شود.

185-. `isdigit(a)`:

- `a`: رشته ورودی

تابع `isdigit` بررسی می‌کند که تمام کاراکترهای رشته `a` از نوع اعداد صحیح هستند یا خیر. اگر تمام کاراکترهای رشته `a` اعداد صحیح باشند، مقدار True برگردانده می‌شود؛ در غیر این صورت مقدار False برگردانده می‌شود.

186-. `islower(a)`:

- `a`: رشته ورودی

تابع `islower` بررسی می‌کند که تمام کاراکترهای رشته `a` از نوع حروف کوچک هستند یا خیر. اگر تمام کاراکترهای رشته `a` حروف کوچک باشند، مقدار True برگردانده می‌شود؛ در غیر این صورت مقدار False برگردانده می‌شود.

187-. `isnumeric(a)`:

- `a`: رشته ورودی

تابع `isnumeric` بررسی می‌کند که تمام کاراکترهای رشته `a` از نوع اعداد هستند یا خیر. اگر تمام کاراکترهای رشته `a` اعداد باشند، مقدار True برگردانده می‌شود؛ در غیر این صورت مقدار False برگردانده می‌شود.

188-. `isspace(a)`:

- `a`: رشته ورودی

تابع `isspace` بررسی می‌کند که تمام کاراکترهای رشته `a` فاصله‌های سفید (فاصله، تب، خط جدید و ...) هستند یا خیر. اگر تمام کاراکترهای رشته `a` فاصله‌های سفید باشند، مقدار True برگردانده می‌شود؛ در غیر این صورت مقدار False برگردانده می‌شود.

189-. `istitle(a)`:

- `a`: رشته ورودی

تابع `istitle` بررسی می‌کند که رشته `a` با استفاده از حروف اول کلمات به صورت عنوان نوشته شده است یا خیر. اگر رشته `a` به صورت عنوان نوشته شده باشد، مقدار True برگردانده می‌شود؛ در غیر این صورت مقدار False برگردانده می‌شود.

190-. `isupper(a)`:

- `a`: رشته ورودی

تابع `isupper` بررسی می‌کند که تمام کاراکترهای رشته `a` از نوع حروف بزرگ هستند یا خیر. اگر تمام کاراکترهای رشته `a` حروف بزرگ باشند، مقدار True برگردانده می‌شود؛ در غیر این صورت مقدار False برگردانده می‌شود.

191-. `rfind(a, sub[, start, end])`:

- `a`: رشته ورودی

- `sub`: رشته جستجو شده

- `start` (اختیاری): اندیس شروع جستجو در رشته `a`

- `end` (اختیاری): اندیس پایان جستجو در رشته `a`

تابع `rfind` آخرین رخداد رشته `sub` در رشته `a` را برمی‌گرداند. اگر رشته `sub` در رشته `a` یافت نشود، مقدار -1 برگردانده می‌شود. این تابع می‌تواند از اندیس `start` شروع و تا اندیس `end` جستجو را انجام دهد.

192-. `rindex(a, sub[, start, end])`:

- `a`: رشته ورودی

- `sub`: رشته جستجو شده

- `start` (اختیاری): اندیس شروع جستجو در رشته `a`

- `end` (اختیاری): اندیس پایان جستجو در رشته `a`

تابع `rindex` اندیس آخرین رخداد رشته `sub` در رشته `a` را برمی‌گرداند. اگر رشته `sub` در رشته `a` یافت نشود، یک خطا از نوع `ValueError` بوجود می‌آید. این تابع می‌تواند از اندیس `start` شروع و تا اندیس endجستجو را انجام دهد.

193-. `startswith(a, prefix[, start, end])`:

- `a`: رشته ورودی

- `prefix`: رشته پیشوند

- `start` (اختیاری): اندیس شروع جستجو در رشته `a`

- `end` (اختیاری): اندیس پایان جستجو در رشته `a`

تابع `startswith` بررسی می‌کند که رشته `a` با رشته `prefix` شروع می‌شود یا خیر. اگر رشته `a` با `prefix` شروع شود، مقدار True برگردانده می‌شود؛ در غیر این صورت مقدار False برگردانده می‌شود.

194-. `str\_len(a)`:

- `a`: رشته ورودی

تابع `str\_len` طول (تعداد کاراکترها) رشته `a` را برمی‌گرداند.

195-. `array(obj[, copy, order])`:

- `obj`: آرایه ورودی

- `copy` (اختیاری): نشان می‌دهد آیا آرایه باید کپی شود یا خیر

- `order` (اختیاری): نشان می‌دهد آیا المان‌های آرایه در حافظه به صورت C-contiguous (رویه خواندن متوالی در ردیف‌ها) یا F-contiguous (رویه خواندن متوالی در ستون‌ها) باشند

تابع `array` یک آرایه numpy را با استفاده از `obj` ایجاد می‌کند. اگر `obj` قبلاً یک آرایه numpy بود، آرایه جدید از `obj` کپی می‌شود مگر اینکه `copy=False` باشد. مقدار پیش‌فرض `copy` برابر True است. توابع `array` همچنین می‌توانند آرایه‌های یونیکد و با ترتیب‌دهی C یا F ایجاد کنند.

196-. `asarray(obj[, order])`:

- `obj`: آرایه ورودی

- `order` (اختیاری): نشان می‌دهد آیا المان‌های آرایه در حافظه به صورت C-contiguous (رویه خواندن متوالی در ردیف‌ها) یا F-contiguous (رویه خواندن متوالی در ستون‌ها) باشند

تابع `asarray` یک آرایه numpy از `obj` ایجاد می‌کند. اگر `obj` قبلاً یک آرایه numpy بود، آرایه جدید از `obj` کپی می‌شود. تابع `asarray` از `obj` به عنوان ورودی دریافت می‌کند و آرایه numpy متناظر با `obj` را برمی‌گرداند.

197-. `chararray(shap, ...])`:

- `shape`: تاپل اندازه‌گیری آرایه

- سایر پارامترها (اختیاری): پارامترهای دیگری که به `numpy.ndarray` مربوط به آرایه اصلی ارسال می‌شوند.

تابع `chararray` یک آرایه numpy با اندازه و شکل مشخص شده از رشته‌ها ایجاد می‌کند. المان‌های این آرایه از نوع رشته هستند.

198-. `numpy.ctypeslib.as\_array(obj, shape=None)`:

- `obj`: آبجکت ctypes ورودی

- `shape` (اختیاری): تاپل اندازه‌گیری آرایه numpy مورد نظر

تابع `as\_array` یک آرایه numpy از آبجکت ctypes `obj` ایجاد می‌کند. شکل آرایه می‌تواند با استفاده از پارامتر `shape` مشخص شود. اگر `shape` مشخص نشود، آرایه به صورت یک بعدی ایجاد می‌شود.

199-. `numpy.ctypeslib.as\_ctypes(obj)`:

- `obj`: آرایه numpy ورودی

تابع `as\_ctypes` آبجکت ctypes متناظر با آرایه numpy `obj` را برمی‌گرداند.

200-. `numpy.ctypeslib.as\_ctypes\_type(dtype)`:

- `dtype`: نوع داده‌ی numpy

تابع `as\_ctypes\_type` آبجکت ctypes متناظر با نوع داده‌ی numpy `dtype` را برمی‌گرداند.

201-. `numpy.ctypeslib.load\_library(libname, loader\_path)`:

- `libname`: نام کتابخانه

- `loader\_path`: مسیر بارگیری

تابع `load\_library` یک کتابخانه ctypes را بارگیری می‌کند و آبجکت کتابخانه را برمی‌گرداند.

202-. `numpy.ctypeslib.ndpointer(dtype=None, ndim=None, shape=None, flags=None)`:

- `dtype` (اختیاری): نوع داده‌ی numpy

- `ndim` (اختیاری): تعداد ابعاد آرایه

- `shape` (اختیاری): تاپل اندازه‌گیری آرایه

- `flags` (اختیاری): پرچم‌های مربوط به آرایه

تابع `ndpointer` یک توصیف کننده‌ی آرایه numpy را برمی‌گرداند که می‌تواند به عنوان نوع آرگومان‌ها در توابع ctypes استفاده شود. این توصیف کننده، نوع داده، تعداد ابعاد، اندازه‌ها و پرچم‌های مربوط به آرایه را مشخص می‌کند.

203-. `numpy.ctypeslib.c\_intp`:

یک آبجکت ctypes معادل با نوع داده‌ی `numpy.intp` است که در ctypes برای اندازه‌گیری اندیس‌ها استفاده می‌شود.

موارد بالا به توابع مربوط به C-Types Foreign Function Interface (numpy.ctypeslib) مربوط هستند که امکان تعامل با کد C/C++ را در numpy فراهم می‌کنند. این توابع به کاربر امکان ایجاد و استفاده از آرایه‌ها و نوع‌های داده‌ی C را در numpy می‌دهند و امکان تبدیل بین آرایه‌ها و نوع‌های داده‌ی C و numpy را فراهم می‌کنند.

Datetime Support Functions:

204-. `datetime\_as\_string(arr[, unit, timezone, ...])`:

- `arr`: آرایه numpy حاوی اطلاعات تاریخ و زمان

- `unit` (اختیاری): واحد زمانی مورد استفاده برای تبدیل به رشته. مقادیر ممکن شامل `'auto'`، `'Y'`، `'M'`، `'W'`، `'D'`، `'h'`، `'m'`، `'s'`، `'ms'`، `'us'`، `'ns'` و `'ps'` هستند.

- `timezone` (اختیاری): منطقه زمانی مورد استفاده برای تبدیل تاریخ و زمان. مقادیر ممکن شامل `'naive'` (زمان بی‌نظم)، `'UTC'`، `'local'` و سایر مناطق زمانی قابل قبول هستند.

- سایر پارامترها (اختیاری): پارامترهای دیگری که به تابع `numpy.datetime\_as\_string` ارسال می‌شوند.

تابع `datetime\_as\_string` یک آرایه numpy حاوی اطلاعات تاریخ و زمان را به رشته‌ها تبدیل می‌کند. مقدار `unit` و `timezone` مشخص می‌کند که تبدیل به رشته به چه صورتی انجام شود.

205-. `datetime\_data(dtype, /)`:

- `dtype`: نوع داده‌ی numpy

تابع `datetime\_data` اطلاعات مربوط به نوع داده‌ی تاریخ و زمان `dtype` را برمی‌گرداند. این اطلاعات شامل نام واحد زمانی، شیفت زمان و...

Business Day Functions:

206-. `busdaycalendar([weekmask, holidays])`:

- `weekmask` (اختیاری): رشته‌ای که بیانگر روزهای کاری هفته است. در این رشته، روزهای کاری با حروف 'B' نمایش داده می‌شوند و روزهای غیرکاری با حروف دیگر. برای مثال، '1111100' بیانگر روزهای کاری از شنبه تا جمعه است.

- `holidays` (اختیاری): لیست تاریخ‌های تعطیل رسمی.

تابع `busdaycalendar` یک شیء تقویم روزهای کاری را برمی‌گرداند که براساس روزهای کاری هفته و تاریخ‌های تعطیل مشخص شده است.

207-. `is\_busday(dates[, weekmask, holidays, ...])`:

- `dates`: یک آرایه numpy حاوی تاریخ‌ها.

- `weekmask` (اختیاری): رشته‌ای که بیانگر روزهای کاری هفته است.

- `holidays` (اختیاری): لیست تاریخ‌های تعطیل رسمی.

- سایر پارامترها (اختیاری): پارامترهای دیگری که به تابع `numpy.is\_busday` ارسال می‌شوند.

تابع `is\_busday` بررسی می‌کند که آیا تاریخ‌های موجود در آرایه `dates` روزهای کاری هستند یا خیر و نتیجه را به صورت آرایه boolean برمی‌گرداند.

208-. `busday\_offset(dates, offsets[, roll, ...])`:

- `dates`: یک آرایه numpy حاوی تاریخ‌ها.

- `offsets`: یک عدد صحیح یا آرایه numpy حاوی اعداد صحیح به عنوان آفست مورد نظر.

- `roll` (اختیاری): روشی برای رفتار در صورتی که تاریخ جدید به خارج از دامنه تاریخ‌های مجاز رفته باشد.

- سایر پارامترها (اختیاری): پارامترهای دیگری که به تابع `numpy.busday\_offset` ارسال می‌شوند.

تابع `busday\_offset` تاریخ‌هایی را با استفاده از آفست مورد نظر از تاریخ‌های موجود در آرایه `dates` محاسبه می‌کند. این تابع تاریخ‌های جدید را به صورت یک آرایه numpy برمی‌گرداند.

209-. `busday\_count(begindates, enddates[, ...])`:

- `begindates`: یک آرایه numpy حاوی تاریخ‌های شروع.

- `enddates`: یک آرایه numpy حاوی تاریخ‌های پایان.

- سایر پارامترها (اختیاری): پارامترهای دیگری که به تابع `numpy.busday\_count` ارسال می‌شوند.

تابع `busday\_count` تعداد روزهای کاری بین هر جفت تاریخ شروع و پایان را محاسبه می‌کند و نتیجه را به صورت یک آرایه numpy برمی‌گرداند.

210. `numpy.datetime\_as\_string(arr, unit=None, casting='same\_kind')`:

- `arr`: یک آرایه numpy حاوی اعداد ممیز شناور نمایش‌دهنده تاریخ و زمان.

- `unit` (اختیاری): واحد زمانی که برای نمایش تاریخ و زمان استفاده می‌شود، مانند 'auto', 'Y', 'M', 'D', 'h', 'm', 's', 'ms', 'us', 'ns'.

- `casting` (اختیاری): روش تبدیل داده‌ها در صورت وجود نیمه‌داده‌ها.

تابع `numpy.datetime\_as\_string` تاریخ و زمان موجود در آرایه `arr` را به صورت رشته‌های قابل خواندن توسط انسان با استفاده از واحد زمانی و تایم‌ مشخص شده تبدیل می‌کند و نتیجه را به صورت یک آرایه numpy از رشته‌ها برمی‌گرداند.

Data type routines:

211-. `can\_cast(from\_, to[, casting])`:

- `from\_`: نوع داده‌ای مبدا.

- `to`: نوع داده‌ای مقصد.

- `casting` (اختیاری): روش تبدیل داده در صورت امکان یا عدم امکان تبدیل.

تابع `can\_cast` بررسی می‌کند که آیا تبدیل از نوع داده‌ای مبدا به نوع داده‌ای مقصد ممکن است یا خیر و نتیجه را به صورت یک مقدار boolean برمی‌گرداند.

212-. `promote\_types(type1, type2)`:

- `type1`: نوع داده‌ای اول.

- `type2`: نوع داده‌ای دوم.

تابع `promote\_types` نوع داده‌ای را که برای نگهداری مقادیر از دو نوع داده‌ای ورودی مناسب است، تعیین می‌کند و نوع داده‌ای به عنوان خروجی برمی گرداند.

213-. `min\_scalar\_type(a, /)`:

- `a`: یک عدد.

تابع `min\_scalar\_type` نوع داده‌ای کمینه را که برای نگهداری عدد ورودی لازم است، تعیین می‌کند و نوع داده‌ای به عنوان خروجی برمی‌گرداند.

214-. `result\_type(\*arrays\_and\_dtypes)`:

- `\*arrays\_and\_dtypes`: یک یا چند آرایه یا نوع داده‌ای.

تابع `result\_type` نوع داده‌ای را که برای نگهداری نتایج عملیات بین آرایه‌ها و نوع داده‌ها ورودی مناسب است، تعیین می‌کند و نوع داده‌ای به عنوان خروجی برمی‌گرداند.

215-. `common\_type(\*arrays)`:

- `\*arrays`: یک یا چند آرایه.

تابع `common\_type` نوع داده‌ای را که برای نگهداری مقادیر مشترک بین آرایه‌ها ورودی مناسب است، تعیین می‌کند و نوع داده‌ای به عنوان خروجی برمی‌گرداند.

216-. `obj2sctype(rep[, default])`:

- `rep`: یک شیء قابل نمایش به عنوان نوع داده.

- `default` (اختیاری): نوع داده‌ای پیشفرض در صورت عدم تطبیق.

تابع `obj2sctype` نوع داده‌ای معادل با یک شیء قابل نمایش را تعیین می‌کند و نوع داده‌ای به عنوان خروجی برمی‌گرداند.

Creating data types:

217-. `dtype(dtype[ copy])`:

- `dtype`: نوع داده‌ای مورد نظر که به صورت رشته یا نوع داده‌ای دیگری مشخص می‌شود.

- `copy` (اختیاری): مشخص می‌کند آیا یک نسخه جدید از نوع داده‌ای باید ساخته شود یا خیر.

تابع `dtype` از یک رشته یا نوع داده‌ای دیگر به عنوان ورودی گرفته و یک نوع داده‌ای NumPy متناظر با آن را ایجاد می‌کند. این تابع برای تعریف سفارشی نوع‌های داده‌ای و استفاده در آرایه‌های NumPy استفاده می‌شود.

218-. `format\_parser(formats, names, titles[, ...])`:

- `formats`: یک لیست از فرمت‌های رشته‌ای مورد نظر.

- `names`: یک لیست از نام‌های فیلدها.

- `titles` (اختیاری): یک لیست از عناوین فیلدها.

تابع `format\_parser` فرمت‌های رشته‌ای و متادیتای مربوط به آن‌ها را دریافت می‌کند و اطلاعات مربوط به نوع داده‌ای ایجاد شده را به عنوان خروجی برمی‌گرداند. این تابع برای ایجاد نوع داده‌های سفارشی با ساختارهای پیچیده استفاده می‌شود.

Data type information:

219-. `finfo(dtype)`:

- `dtype`: نوع داده‌ای مورد نظر.

تابع `finfo` اطلاعات مربوط به نوع داده‌ای را به عنوان خروجی برمی‌گرداند. این اطلاعات شامل مقادیر حداکثر و حداقل، دقت، نمایشگر قدرتی، و سایر ویژگی‌های مرتبط با نوع داده‌ای می‌باشد.

این تابع برای بررسی و کنترل دقت و ویژگی‌های مرتبط با نوع داده‌ها استفاده می‌شود.

220-. `iinfo(type)`:

- `type`: نوع داده‌ای مورد نظر.

تابع `iinfo` اطلاعات مربوط به نوع داده‌ای صحیح را به عنوان خروجی برمی‌گرداند. این اطلاعات شامل مقادیر حداکثر و حداقل ممکن برای نوع داده‌ای صحیح است. این تابع برای بررسی محدوده مقادیر قابل نمایش در نوع داده‌های صحیح استفاده می‌شود.

Data type testing:

221-. `issctype(rep)`:

- `rep`: یک نمایش‌پذیرنده برای نوع داده.

تابع `issctype` بررسی می‌کند که آیا یک شیء قابل نمایش معتبر برای نوع داده‌ای است یا خیر. اگر شیء قابل نمایش نوع داده‌ای معتبر باشد، مقدار `True` برگردانده می‌شود و در غیر این صورت `False` برگردانده می‌شود.

222-. `issubdtype(arg1, arg2)`:

- `arg1`: نوع داده‌ای اصلی.

- `arg2`: نوع داده‌ای فرعی.

تابع `issubdtype` بررسی می‌کند که آیا نوع داده‌ای فرعی یک زیرنوع از نوع داده‌ای اصلی است یا خیر. اگر نوع داده‌ای فرعی زیرنوع باشد، مقدار `True` برگردانده می‌شود و در غیر این صورت `False` برگردانده می‌شود.

223-. `issubsctype(arg1, arg2)`:

- `arg1`: نوع داده‌ای اصلی.

- `arg2`: نوع داده‌ای فرعی.

تابع `issubsctype` بررسی می‌کند که آیا نوع داده‌ای فرعی یک زیرنوع نامستقیم از نوع داده‌ای اصلی است یا خیر. اگر نوع داده‌ای فرعی زیرنوع نامستقیم باشد، مقدار `True` برگردانده می‌شود و در غیر این صورت `False` برگردانده می‌شود.

224-. `issubclass\_(arg1, arg2)`:

- `arg1`: کلاس اصلی.

- `arg2`: کلاس فرعی.

تابع `issubclass\_` بررسی می‌کند که آیا کلاس فرعی یک زیرکلاس از کلاس اصلی است یا خیر. اگر کلاس فرعی زیرکلاس باشد، مقدار `True` برگردانده می‌شود و در غیر این صورت `False` برگردانده می‌شود.

225-. `find\_common\_type(array\_types, scalar\_types)`:

- `array\_types`: یک لیست از نوع داده‌های آرایه.

- `scalar\_types`: یک لیست از نوع داده‌های مقدار مجزا.

تابع `find\_common\_type` نوع داده‌ای را که هماهنگ با تمام نوع داده‌های آرایه و مقادیر مجزا است، پیدا می‌کند و آن را به عنوان خروجی برمی‌گرداند.

Miscellaneous:

226-. `typename(char)`:

- `char`: یک کاراکتر یا رشته کاراکتری.

تابع `typename` نام نوع داده‌ای معادل با یک کاراکتر یا رشته کاراکتری را برمی‌گرداند. این تابع برای بررسی نوع داده‌ای متناظر با کاراکترهای نمایش‌پذیر در NumPy استفاده می‌شود.

227-. `sctype2char(sctype)`:

- `sctype`: نوع داده‌ای NumPy.

تابع `sctype2char` نمایش‌پذیرنده (کاراکتر) معادل با نوع داده‌ای NumPy را برمی‌گرداند. این تابع برای تبدیل نوع داده‌های NumPy به کاراکترهای نمایش‌پذیر در عملیات‌های مرتبط با C استفاده می‌شود.

228-. `mintypecode(typechars[, typeset, default])`:

- `typechars`: رشته حاوی کدهای نوع.

- `typeset`: مجموعه کدهای نوع مجاز (اختیاری).

- `default`: کد نوع پیش‌فرض (اختیاری).

تابع `mintypecode` کمترین کد نوع را برمی‌گرداند که در رشته کدهای نوع موجود می‌باشد. این تابع برای انتخاب کد نوع کمینه در عملیات‌های مرتبط با C استفاده می‌شود.

229-. `maximum\_sctype(t)`:

- `t`: یک رشته کاراکتری.

تابع `maximum\_sctype` نام ماکسیمم نوع داده‌ای را برمی‌گرداند که در رشته کاراکتری مشخص شده وجود دارد. این تابع برای دریافت نام ماکسیمم نوع داده‌ای در عملیات‌های مرتبط با C استفاده می‌شود..

Dual (SciPy-accelerated routines):

این بخش شامل توابعی است که از کتابخانه دومی (Dual) برای اجرای بهینه تر عملیات مرتبط با SciPy استفاده می‌کنند. این توابع از پیاده‌سازی‌های بهینه‌تر برخی عملیات عددی برای استفاده در کتابخانه‌های علمی و عددی استفاده می‌کنند.

230-. `cholesky(a)`:

- `a`: ماتریس متقارن و معین مثبت.

تابع `cholesky` ماتریس کوواریانس متقارن و معین مثبت `a` را به صورت ماتریس کوچولیسکی (Cholesky) بازگردانده و بردار فشرده کننده را برمی‌گرداند. این تابع برای حل سیستم‌های معادلات خطی و محاسبه ماتریس بالقوه استفاده می‌شود.

231-. `det(a)`:

- `a`: ماتریس داده شده.

تابع `det` مقدار تعین‌گر (determinant) ماتریس `a` را محاسبه کرده و آن را برمی‌گرداند. تعین‌گر ماتریس نشان می‌دهد که ماتریس چه تأثیری بر روی تغییر حجم و شکل بردارها دارد و در بسیاری از محاسبات ریاضی و هندسی استفاده می‌شود.

232-. `eig(a)`:

- `a`: ماتریس داده شده.

تابع `eig` بردارهای ویژه (eigenvectors) و مقادیر ویژه (eigenvalues) ماتریس `a` را محاسبه کرده و به صورت جفتی آن‌ها را برمی‌گرداند. بردارها و مقادیر ویژه در تحلیل خواص و ساختارهای ماتریسی مهم هستند و در بسیاری از مسائل علمی و مهندسی استفاده می‌شوند.

101. `e

igh(a[, UPLO])`:-233

- `a`: ماتریس داده شده.

- `UPLO` (اختیاری): مشخص می‌کند که ماتریس `a` چگونه ذخیره شده است.

تابع `eigh` بردارهای ویژه (eigenvectors) و مقادیر ویژه (eigenvalues) یک ماتریس تقارنی `a` را محاسبه می‌کند و به صورت جفتی آن‌ها را برمی‌گرداند. این تابع برای ماتریس‌های تقارنی بسیار بزرگ و چگال استفاده می‌شود.

234-. `inv(a)`:

- `a`: ماتریس معکوس‌پذیر.

تابع `inv` ماتریس `a` را معکوس می‌کند و ماتریس معکوس را برمی‌گرداند.

235-. `lstsq(a, b[, rcond])`:

- `a`: ماتریس ضرایب سیستم معادلات خطی.

- `b`: بردار مقادیر راست‌دست سیستم معادلات خطی.

- `rcond` (اختیاری): شرایط توقف برای تخمین بهتر جواب.

تابع `lstsq` با استفاده از روش کمترین مربعات، بهترین تخمین ممکن از جواب سیستم معادلات خطی `a @ x = b` را محاسبه می‌کند. این تابع برای حل مسائل رگرسیون و تحلیل داده‌ها استفاده می‌شود.

236-.

`norm(x[, ord, axis, keepdims])`:

- `x`: آرایه داده شده.

- `ord` (اختیاری): نوع نرم مورد نظر (مثبت، منفی یا بی‌نهایت).

- `axis` (اختیاری): محور‌های محاسبه نرم.

- `keepdims` (اختیاری): نگه‌داشتن ابعاد آرایه اصلی.

تابع `norm` نرم ماتریس یا بردار `x` را با توجه به پارامترهای ورودی محاسبه می‌کند و آن را برمی‌گرداند. نرم در تحلیل عددی یک مفهوم مهم است که در بسیاری از محاسبات و الگوریتم‌ها مورد استفاده قرار می‌گیرد.

237-. `pinv(a[, rcond, hermitian])`:

- `a`: ماتریس داده شده.

- `rcond` (اختیاری): شرایط توقف برای تخمین بهتر جواب.

- `hermitian` (اختیاری): مشخص می‌کند آیا ماتریس ورودی هرمیتی است یا خیر.

تابع `pinv` معکوس عمومی ماتریس `a` را با استفاده از روش تخمین معکوس معادل (Moore-Penrose) محاسبه می‌کند و آن را برمی‌گرداند. این تابع برای حل مسائل رگرسیون و تحلیل داده‌ها استفاده می‌شود.

238-. `solve(a, b)`:

- `a`: ماتریس ضرایب سیستم معادلات خطی.

- `b`: بردار مقادیر راست‌دست سیستم معادلات خطی.

تابع `solve` با استفاده از روش معکوس ماتریس یا روش الگویی، جواب سیستم معادلات خطی `a @ x = b` را محاسبه می‌کند. این تابع برای حل سیستم‌های معادلات خطی استفاده می‌شود.

239-. `svd(a[, full\_matrices, compute\_uv])`:

- `a`: ماتریس داده شده.

- `full\_matrices` (اختیاری): مشخص می‌کند آیا ماتریسهای U و V با ابعاد کامل برگردانده شوند یا خیر.

- `compute\_uv` (اختیاری): مشخص می‌کند آیا ماتریسهای U و V محاسبه شوند یا خیر.

تابع `svd` تجزیه مقدار واقعی ماتریس `a` را با استفاده از تجزیه مقادیر ویژه (Singular Value Decomposition) محاسبه کرده و ماتریسهای U ، S و V را برمی‌گرداند. این تابع در تحلیل داده‌ها و کاهش بعد مورد استفاده قرار می‌گیرد.

FFT:

240-. `fft(a[, n, axis, norm])`:

- `a`: آرایه داده شده.

- `n` (اختیاری): اندازه تبدیل.

- `axis` (اختیاری): محور محاسبه تبدیل.

- `norm` (اختیاری): نوع نرمال‌سازی.

تابع `fft` تبدیل فوریه سریع (FFT) از آرایه `a` را محاسبه می‌کند و نتیجه را برمی‌گرداند. تبدیل فوریه سریع یک الگوریتم مهم در پردازش سیگنال و تحلیل فرکانس است.

241-. `fft2(a[, s, axes, norm])`:

- `a`: آرایه داده شده.

- `s` (اختیاری): اندازه تبدیل برای هر محور.

- `axes` (اختیاری): محورهای محاسبه تبدیل.

- `norm` (اختیاری): نوع نرمال‌سازی.

تابع `fft2` تبدیل فوریه دوبعدی (FFT) از آرایه `a` را محاسبه می‌کند و نتیجه را برمی‌گرداند. این تابع در پردازش تصویر و تحلیل فرکانس دوبعدی استفاده می‌شود.

242-. `fftn(a[, s, axes, norm])`:

- `a`: آرایه داده شده.

- `s`

(اختیاری): اندازه تبدیل برای هر محور.

- `axes` (اختیاری): محورهای محاسبه تبدیل.

- `norm` (اختیاری): نوع نرمال‌سازی.

تابع `fftn` تبدیل فوریه چندبعدی (FFT) از آرایه `a` را محاسبه می‌کند و نتیجه را برمی‌گرداند. این تابع در پردازش سیگنال و تحلیل فرکانس چندبعدی استفاده می‌شود.

243-. `ifft(a[, n, axis, norm])`:

- `a`: آرایه داده شده.

- `n` (اختیاری): اندازه تبدیل.

- `axis` (اختیاری): محور محاسبه تبدیل.

- `norm` (اختیاری): نوع نرمال‌سازی.

تابع `ifft` تبدیل فوریه معکوس (IFFT) از آرایه `a` را محاسبه می‌کند و نتیجه را برمی‌گرداند. تبدیل فوریه معکوس برای بازگشت به دامنه زمانی از دامنه فرکانس استفاده می‌شود.

244-. `ifft2(a[, s, axes, norm])`:

- `a`: آرایه داده شده.

- `s` (اختیاری): اندازه تبدیل برای هر محور.

- `axes` (اختیاری): محورهای محاسبه تبدیل.

- `norm` (اختیاری): نوع نرمال‌سازی.

تابع `ifft2` تبدیل فوریه معکوس دوبعدی (IFFT) از آرایه `a` را محاسبه می‌کند و نتیجه را برمی‌گرداند. این تابع در بازگشت تصویر به دامنه زمانی استفاده می‌شود.

245-. `ifftn(a[, s, axes, norm])`:

- `a`: آرایه داده شده.

- `s` (اختیاری): اندازه تبدیل برای هر محور.

- `axes` (اختیاری): محورهای محاسبه تبدیل.

توابع نرمال سازی (norm):اختیاری(نوع نرمال سازی است )

تابع `ifftn` تبدیل فوریه معکوس چندبعدی (IFFT) از آرایه `a` را محاسبه می‌کند و نتیجه را برمی‌گرداند. این تابع در بازگشت سیگنال چندبعدی به دامنه زمانی استفاده می‌شود.

246-. `i0(x)`:

- `x`: مقدار ورودی.

تابع `i0` تابع تعریف‌نشده Bessel درجه صفر را از `x` محاسبه می‌کند و نتیجه را برمی‌گرداند.

Mathematical functions with automatic domain:

247-. `sqrt(x)`:

- `x`: مقدار ورودی.

تابع `sqrt` جذر مربعی از `x` را محاسبه می‌کند و نتیجه را برمی‌گرداند.

248-. `log(x)`:

- `x`: مقدار ورودی.

تابع `log` لگاریتم طبیعی (پایه `e`) از `x` را محاسبه می‌کند و نتیجه را برمی‌گرداند.

249-. `log2(x)`:

- `x`: مقدار ورودی.

تابع `log2` لگاریتم پایه ۲ از `x` را محاسبه می‌کند و نتیجه را برمی‌گرداند.

250-. `logn(n, x)`:

- `n`: پایه لگاریتم.

- `x`: مقدار ورودی.

تابع `logn` لگاریتم با پایه `n` از `x` را محاسبه می‌کند و نتیجه را برمی‌گرداند.

251-. `log10(x)`:

- `x`: مقدار ورودی.

تابع `log10` لگاریتم پایه ۱۰ از `x` را محاسبه می‌کند و نتیجه را برمی‌گرداند.

252-. `power(x, p)`:

- `x`: مقدار پایه.

- `p`: مقدار توان.

تابع `power` مقدار `x` را به توان `p` می‌رساند و نتیجه را برمی‌گرداند.

253. `arccos(x)`:

- `x`: مقدار ورودی.

تابع `arccos` کسینوس معکوس (آرک‌کسینوس) از `x` را محاسبه می‌کند و نتیجه را برمی‌گرداند.

254-. `arcsin(x)`:

- `x`: مقدار ورودی.

تابع `arcsin` سینوس معکوس (آرک‌سینوس) از `x` را محاسبه می‌کند و نتیجه را برمی‌گرداند.

255-. `arctanh(x)`:

- `x`: مقدار ورودی.

تابع `arctanh` تانژانت هاي معکوس هيپربوليک (آرک‌تانژانت هيپربوليک) از `x` را محاسبه می‌کند و نتیجه را برمی‌گرداند.

Setting and getting error handling:

256-. `seterr([all, over, under, invalid])`:

- `all`: مقدار برای تنظیم همه خطاها. (اختیاری)

- `over`: مقدار برای تنظیم خطاهای بزرگتر از حد مجاز. (اختیاری)

- `under`: مقدار برای تنظیم خطاهای کوچکتر از حد مجاز. (اختیاری)

- `invalid`: مقدار برای تنظیم خطاهای نامعتبر. (اختیاری)

تابع `seterr` برای تنظیم حالت خطاها در عملیات‌های محاسباتی از مقادیر ورودی استفاده می‌کند. با استفاده از آرگومان‌ها، می‌توان حالت‌های مختلف خطاها را تنظیم کرد.

257-. `geterr()`:

تابع `geterr` وضعیت فعلی خطاها را برمی‌گرداند.

258-. `seterrcall(func)`:

- `func`: یک تابع.

تابع `seterrcall` برای تنظیم تابعی که در صورت رخداد خطا فراخوانی می‌شود، استفاده می‌شود.

259-. `geterrcall()`:

تابع `geterrcall` تابع فعلی که در صورت رخداد خطا فراخوانی می‌شود را برمی‌گرداند.

260-. `errstate(\*\*kwargs)`:

- `\*\*kwargs`: آرگومان‌های کلیدواژه‌ای.

تابع `errstate` برای مدیریت حالت خطاها در محدوده‌ی یک بلاک کد استفاده می‌شود. با استفاده از آرگومان‌های کلیدواژه‌ای، می‌توان حالت‌های خطاها را تنظیم کند.

Setting and getting error handling:

261-. `seterrobj(errobj, /)`:

- `errobj`: یک شیء خطا.

تابع `seterrobj` برای تنظیم شیء خطا برای مدیریت خطاها در کد استفاده می‌شود.

262-. `geterrobj()`:

تابع `geterrobj` شیء خطا فعلی را برمی‌گرداند.

Discrete Fourier Transform (numpy.fft):

Standard FFTs:

263-. `fft(a[, n, axis, norm])`:

- `a`: آرایه ورودی.

- `n`: تعداد نقاط داده‌ها برای محاسبه FFT. (اختیاری)

- `axis`: محوری که بر روی آن FFT اعمال می‌شود. (اختیاری)

- `norm`: نوع نرمال‌سازی FFT. (اختیاری)

تابع `fft` تبدیل فوریه سریع (FFT) بر روی آرایه ورودی `a` انجام می‌دهد و نتیجه را برمی‌گرداند.

264-. `ifft(a[, n, axis, norm])`:

- `a`: آرایه ورودی.

- `n`: تعداد نقاط داده‌ها برای محاسبه IFFT. (اختیاری)

- `axis`: محوری که بر روی آن IFFT اعمال می‌شود. (اختیاری)

- `norm`: نوع نرمال‌سازی IFFT. (اختیاری)

تابع `ifft` تبدیل فوریه معکوس (IFFT) بر روی آرایه ورودی `a` انجام می‌دهد و نتیجه را برمی‌گرداند.

265-. `fft2(a[, s, axes, norm])`:

- `a`: آرایه ورودی.

- `s`: اندازه خروجی FFT در هر محور. (اختیاری)

- `axes`: محورهایی که بر روی آنها FFT اعمال می‌شود. (اختیاری)

- `norm`: نوع نرمال‌سازی FFT. (اختیاری

- تابع `fft2` تبدیل فوریه سریع دوبعدی (2D FFT) بر روی آرایه ورودی `a` انجام می‌دهد و نتیجه را برمی‌گرداند.

- `a`: آرایه ورودی.

- `s`: اندازه خروجی IFFT در هر محور. (اختیاری)

- `axes`: محورهایی که بر روی آنها IFFT اعمال می‌شود. (اختیاری)

- `norm`: نوع نرمال‌سازی IFFT. (اختیاری)

تابع `ifft2` تبدیل فوریه معکوس دوبعدی (2D IFFT) بر روی آرایه ورودی `a` انجام می‌دهد و نتیجه را برمی‌گرداند.

266-. `fftn(a[, s, axes, norm])`:

- `a`: آرایه ورودی.

- `s`: اندازه خروجی FFT در هر محور. (اختیاری)

- `axes`: محورهایی که بر روی آنها FFT اعمال می‌شود. (اختیاری)

- `norm`: نوع نرمال‌سازی FFT. (اختیاری)

تابع `fftn` تبدیل فوریه سریع چندبعدی (nD FFT) بر روی آرایه ورودی `a` انجام می‌دهد و نتیجه را برمی‌گرداند.

267-. `ifftn(a[, s, axes, norm])`:

- `a`: آرایه ورودی.

- `s`: اندازه خروجی IFFT در هر محور. (اختیاری)

- `axes`: محورهایی که بر روی آنها IFFT اعمال می‌شود. (اختیاری)

- `norm`: نوع نرمال‌سازی IFFT. (اختیاری)

تابع `ifftn` تبدیل فوریه معکوس چندبعدی (nD IFFT) بر روی آرایه ورودی `a` انجام می‌دهد و نتیجه را برمی‌گرداند.

این توابع مرتبط با تبدیل فوریه هستند و از آنها برای تبدیل سیگنال‌ها از دامنه زمان به دامنه فرکانس و بالعکس استفاده می‌شود.

268-. `rfft(a[, n, axis, norm])`:

- `a`: آرایه ورودی.

- `n`: طول تبدیل فوریه. (اختیاری)

- `axis`: محور تبدیل فوریه. (اختیاری)

- `norm`: نوع نرمال‌سازی تبدیل فوریه. (اختیاری)

تابع `rfft` تبدیل فوریه دیسکرت سریع بر روی آرایه ورودی `a` انجام می‌دهد و نتیجه تبدیل فوریه را برمی‌گرداند. تفاوت این تابع با تابع `fft` این است که آرایه ورودی `a` فقط شامل اعداد حقیقی است و نتایج تبدیل فوریه نصف طول عمومی تابع `fft` است.

269-. `irfft(a[, n, axis, norm])`:

- `a`: آرایه ورودی.

- `n`: طول تبدیل فوریه معکوس. (اختیاری)

- `axis`: محور تبدیل فوریه معکوس. (اختیاری)

- `norm`: نوع نرمال‌سازی تبدیل فوریه معکوس. (اختیاری)

تابع `irfft` تبدیل فوریه معکوس دیسکرت سریع بر روی آرایه ورودی `a` انجام می‌دهد و نتیجه تبدیل فوریه معکوس را برمی‌گرداند. این تابع برای تبدیل سیگنال‌های حقیقی از دامنه فرکانس به دامنه زمان استفاده می‌شود.

270-. `rfft2(a[, s, axes, norm])`:

- `a`: آرایه ورودی.

- `s`: اندازه خروجی تبدیل فوریه در هر محور. (اختیاری)

- `axes`: محورهایی که بر روی آنها تبدیل فوریه انجام می‌شود. (اختیاری)

- `norm`: نوع نرمال‌سازی تبدیل فوریه. (اختیاری)

تابع `rfft2` تبدیل فوریه دوبعدی دیسکرت سریع بر روی آرایه ورودی `a` انجام می‌دهد و نتیجه تبدیل فوریه را برمی‌گرداند. این تابع برای سیگنال‌های حقیقی دوبعدی استفاده می‌شود و نتایج تبدیل فوریه در نیمی از فضای فرکانس را بازمی‌گرداند.

271-. `irfft2(a[, s, axes, norm])`:

- `a`: آرایه ورودی.

- `s`: اندازه خروجی تبدیل فوریه معکوس در هر محور. (اختیاری)

- `axes`: محورهایی که بر روی آنها تبدیل فوریه معکوس انجام می‌شود. (اختیاری)

- `norm`: نوع نرمال‌سازی تبدیل فوریه معکوس. (اختیاری)

تابع `irfft2` تبدیل فوریه معکوس دوبعدی دیسکرت سریع بر روی آرایه ورودی `a` انجام می‌دهد و نتیجه تبدیل فوریه معکوس را برمی‌گرداند. این تابع برای بازیابی سیگنال‌های حقیقی دوبعدی از دامنه فرکانس استفاده می‌شود.

272-. `rfftn(a[, s, axes, norm])`:

- `a`: آرایه ورودی.

- `s`: اندازه خروجی تبدیل فوریه در هر محور. (اختیاری)

- `axes`: محورهایی که بر روی آنها تبدیل فوریه انجام می‌شود. (اختیاری)

- `norm`: نوع نرمال‌سازی تبدیل فوریه. (اختیاری)

تابع `rfftn` تبدیل فوریه دیسکرت سریع سه‌بعدی بر روی آرایه ورودی `a` انجام می‌دهد و نتیجه تبدیل فوریه را برمی‌گرداند. این تابع برای سیگنال‌های حقیقی سه‌بعدی استفاده می‌شود و نتایج تبدیل فوریه در نیمی از فضای فرکانس را بازمی‌گرداند.

273-. `irfftn(a[, s, axes, norm])`:

- `a`: آرایه ورودی.

- `s`: اندازه خروجی تبدیل فوریه معکوس در هر محور. (اختیاری)

- `axes`: محورهایی که بر روی آنها تبدیل فوریه معکوس انجام می‌شود. (اختیاری)

- `norm`: نوع نرمال‌سازی تبدیل فوریه معکوس. (اختیاری)

تابع `irfftn` تبدیل فوریه معکوس دیسکرت سریع سه‌بعدی بر روی آرایه ورودی `a` انجام می‌دهد و نتیجه تبدیل فوریه معکوس را برمی‌گرداند. این تابع برای بازیابی سیگنال‌های حقیقی سه‌بعدی از دامنه فرکانس استفاده می‌شود.

274-. `hfft(a[, n, axis, norm])`:

- `a`: آرایه ورودی.

- `n`: اندازه پردازش داده شده برای تبدیل فوریه نیمه سری (حتما یک عدد صحیح برابر با طول آرایه ورودی و یا بزرگتر از آن باشد). (اختیاری)

- `axis`: محوری که بر روی آن تبدیل فوریه نیمه سری انجام می‌شود. (اختیاری)

- `norm`: نوع نرمال‌سازی تبدیل فوریه نیمه سری. (اختیاری)

تابع `hfft` تبدیل فوریه نیمه سری دیسکرت هارمونیکی را بر روی آرایه ورودی `a` انجام می‌دهد و نتیجه تبدیل فوریه نیمه سری را برمی‌گرداند. این تابع برای سیگنال‌های حقیقی با خاصیت تقارن هارمونیکی استفاده می‌شود و نتایج تبدیل فوریه در دامنه فرکانس نیمه مثبت را بازمی‌گرداند.

275-. `ihfft(a[, n, axis, norm])`:

- `a`: آرایه ورودی.

- `n`: اندازه بازسازی تبدیل فوریه نیمه سری (حتما یک عدد صحیح بزرگتر از طول آرایه ورودی باشد). (اختیاری)

- `axis`: محوری که بر روی آن تبدیل فوریه نیمه سری معکوس انجام می‌شود. (اختیاری)

- `norm`: نوع نرمال‌سازی تبدیل فوریه نیمه سری معکوس. (اختیاری)

تابع `ihfft` تبدیل فوریه نیمه سری معکوس دیسکرت هارمونیکی را بر روی آرایه ورودی `a` انجام می‌دهد و نتیجه تبدیل فوریه نیمه سری معکوس را برمی‌گرداند.

276-. `fftfreq(n[, d])`:

- `n`: تعداد نقاط در تبدیل فوریه.

- `d`: فاصله بین نقاط در واحد‌های زمانی (اختیاری).

تابع `fftfreq` فرکانس‌های مربوط به نقاط تبدیل فوریه را برمی‌گرداند. طول تبدیل فوریه `n` را ورودی می‌گیرد و با استفاده از فاصله `d` بین نقاط در واحد‌های زمانی، فرکانس‌های مربوطه را محاسبه می‌کند.

277-. `rfftfreq(n[, d])`:

- `n`: تعداد نقاط در تبدیل فوریه نیمه سری.

- `d`: فاصله بین نقاط در واحد‌های زمانی (اختیاری).

تابع `rfftfreq` فرکانس‌های مربوط به نقاط تبدیل فوریه نیمه سری را برمی‌گرداند. طول تبدیل فوریه نیمه سری `n` را ورودی می‌گیرد و با استفاده از فاصله `d` بین نقاط در واحد‌های زمانی، فرکانس‌های مربوطه را محاسبه می‌کند.

278-. `fftshift(x[, axes])`:

- `x`: آرایه ورودی.

- `axes`: محورهایی که بر روی آن‌ها تغییرات محورها انجام می‌شود (اختیاری).

تابع `fftshift` محتوای آرایه ورودی `x` را به طور محوری جابجا می‌کند، به طوری که نقاط مرکزی در آرایه به مرکز آرایه منتقل شده و نقاط مرکزی جدید در قسمت‌های انتهایی آرایه قرار می‌گیرند. این تابع به طور معمول در تحلیل فوریه و پردازش سیگنال استفاده می‌شود.

279-. `apply\_along\_axis(func1d, axis, arr, \*args, ...)`:

- `func1d`: تابع یک متغیر که بر روی هر تکه یک بعدی از آرایه ورودی اعمال می‌شود.

- `axis`: محوری که بر روی آن تابع `func1d` اعمال می‌شود.

- `arr`: آرایه ورودی.

- `\*args`: آرگومان‌های اضافی برای تابع `func1d`.

تابع `apply\_along\_axis` تابع `func1d` را بر روی تکه‌های یک بعدی از آرایه `arr` با توجه به محور `axis` اعمال می‌کند. به عبارت دیگر، بر روی هر بعد از آرایه ورودی که متناظر با محور مشخص شده است، تابع `func1d` اعمال می‌شود.

280-. `apply\_over\_axes(func, a, axes)`:

- `func`: تابع که بر روی آرایه ورودی اعمال می‌شود.

- `a`: آرایه ورودی.

- `axes`: محورهایی که بر روی آن‌ها تابع `func` اعمال می‌شود.

تابع `apply\_over\_axes` تابع `func` را بر روی آرایه ورودی `a` با توجه به محورهای مشخص شده در `axes` اعمال می‌کند. برای هر محور، تابع `func` بر روی زیرآرایه‌های متناظر با آن محور اجرا می‌شود.

281-. `frompyfunc(func, /, nin, nout, \*[, identity])`:

- `func`: تابع پایتون که می‌خواهید به صورت تابع numpy اعمال شود.

- `nin`: تعداد پارامترهای ورودی تابع `func`.

- `nout`: تعداد خروجی‌های تابع `func`.

- `\*`: آرگومان‌های اضافی برای تابع `func`.

- `identity`: عدد یا آرایه برای تعیین نوع خروجی تابع. (اختیاری)

تابع `frompyfunc` یک تابع پایتون را به عنوان ورودی می‌گیرد و یک تابع numpy را بر اساس آن ایجاد می‌کند. تابع numpy حاصل، قابلیت اعمال تابع پایتون را بر روی آرایه‌های numpy دارد. تعداد و نوع پارامترها و خروجی‌های تابع ایجاد شده با استفاده از `frompyfunc` تعیین می‌شود.

از این توابع برای اعمال توابع پایتونی به صورت موثر بر روی آرایه‌های numpy استفاده می‌شود. به این ترتیب، توابع پایتونی را می‌توان به عنوان توابع numpy استفاده کرد و به صورت برداری بر روی آرایه‌ها اعمال نمود.

282-. `vectorize(pyfunc[, otypes, doc, excluded, ...])`:

- `pyfunc`: تابع پایتونی که می‌خواهید به صورت تابع numpy برداری شود.

- `otypes`: نوع‌های خروجی مورد انتظار برای تابع برداری شده. (اختیاری)

- `doc`: رشته‌ای که توضیحات مربوط به تابع برداری شده را مشخص می‌کند. (اختیاری)

- `excluded`: لیستی از نام‌های آرگومان‌هایی که نباید در ورودی تابع برداری شده قرار بگیرند. (اختیاری)

- `...`: آرگومان‌های اضافی برای تابع برداری شده.

تابع `vectorize` یک تابع پایتون را به عنوان ورودی می‌گیرد و یک تابع numpy برداری شده را ایجاد می‌کند. تابع numpy حاصل، قادر است بر روی آرایه‌های numpy با ابعاد مختلف عمل کند و به صورت برداری تابع پایتون را بر روی هر عنصر آرایه اعمال کند.

از تابع `vectorize` برای تبدیل توابع پایتونی به توابع numpy قابل استفاده برای آرایه‌های بزرگ و محاسبات برداری استفاده می‌شود. این تابع به صورت خودکار موارد مختلفی مانند پخش آرگومان‌ها و نوع خروجی را مدیریت می‌کند، بنابراین می‌توان آن را برای عملیات برداری ساده و سریع استفاده کرد.

283-. `info([object, maxwidth, output])`:

- `object`: شیء یا نام متغیری که می‌خواهید اطلاعات مربوط به آن را دریافت کنید. (اختیاری)

- `maxwidth`: حداکثر عرض خطوط خروجی. (اختیاری)

- `output`: نوع خروجی مورد نظر مانند stdout، stderr، html یا text. (اختیاری)

تابع `info` اطلاعات مفصلی در مورد یک شیء یا نام متغیر را ارائه می‌دهد. با فراخوانی این تابع، شما می‌توانید جزئیاتی مانند نام، نوع داده، متدها، توابع و ویژگی‌های مربوط به یک شیء را مشاهده کنید. اگر `object` مشخص نشود، تمام شیء موجود در فضای نام را نشان می‌دهد. تنظیمات دیگر مانند `maxwidth` و `output` نحوه نمایش و خروجی داده را تعیین می‌کنند. این تابع برای بررسی و کاوش در ساختارهای داده و کلاس‌ها بسیار مفید است.

284-. `source(object[, output])`:

- `object`: شیء یا نام متغیری که می‌خواهید منبع مربوط به آن را مشاهده کنید.

- `output`: نوع خروجی مورد نظر مانند stdout، stderr، html یا text. (اختیاری)

تابع `source` برای نمایش منبع (کد منبع) یک شیء یا نام متغیر استفاده می‌شود. با فراخوانی این تابع و ارائه نام یک شیء یا نام متغیر، می‌توانید کد منبع مربوط به آن را مشاهده کنید. این عمل برای مطالعه و فهمیدن نحوه پیاده‌سازی یک تابع یا کلاس مفید است.

285-. `load(file[, mmap\_mode, allow\_pickle, ...])`:

- `file`: مسیر فایل یا شیء file-like که باید بارگیری شود.

- `mmap\_mode`: حالت mmap برای خواندن داده ها. (اختیاری)

- `allow\_pickle`: مجوز بارگیری اشیاء pickled. (اختیاری)

- `fix\_imports`: در صورت تنظیم True، وارد کردن ماژول ها برای pickle های قدیمی تر را تنظیم می کند. (اختیاری)

تابع `load` بارگیری داده ها را از فایل ذخیره شده با استفاده از پروتکل pickle انجام می‌دهد. با ارائه مسیر فایل یا شیء file-like به عنوان ورودی، تابع فایل را می‌خواند و داده های موجود در آن را بازیابی می‌کند. می‌توانید با استفاده از پارامترهای اختیاری `mmap\_mode`، `allow\_pickle` و `fix\_imports` نحوه بارگیری داده ها را تنظیم کنید.

286-. `save(file, arr[, allow\_pickle, fix\_imports])`:

- `file`: مسیر فایل یا شیء file-like که باید داده ها در آن ذخیره شود.

- `arr`: آرایه یا داده هایی که می‌خواهید ذخیره کنید.

- `allow\_pickle`: مجوز ذخیره اشیاء pickled. (اختیاری)

- `fix\_imports`: در صورت تنظیم True، وارد کردن ماژول ها برای pickle های قدیمی تر را تنظیم می کند. (اختیاری)

تابع `save` داده ها را به فایل با استفاده از پروتکل pickle ذخیره می‌کند. با ارائه مسیر فایل یا شیء file-like به عنوان ورودی و همچنین آرایه یا داده هایی که می‌خواهید ذخیره کنید، تابع داده ها را در فایل ذخیره می‌کند. می‌توانید با استفاده از پارامترهای اختیاری `allow\_pickle` و `fix\_imports` نحوه ذخیره کردن داده ها را تنظیم می کند.

287-. `savez(file, \*args, kwds)`:

- `file`: مسیر فایل یا شیء file-like که باید داده ها در آن ذخیره شود.

- `\*args`: آرگومان‌های ترتیبی که شامل آرایه‌ها و داده‌هایی است که می‌خواهید ذخیره کنید.

- `kwds`: آرگومان‌های کلیدی که شامل آرایه‌ها و داده‌هایی است که می‌خواهید ذخیره کنید.

تابع `savez` داده ها را به فایل با فرمت فشرده NumPy `.npz` ذخیره می‌کند. با ارائه مسیر فایل یا شیء file-like به عنوان ورودی و همچنین آرایه‌ها و داده‌هایی که می‌خواهید ذخیره کنید، تابع داده ها را در فایل `.npz` ذخیره می‌کند. می‌توانید از آرگومان‌های ترتیبی `\*args` و آرگومان‌های کلیدی `kwds` برای ارائه داده ها استفاده کنید. فایل `.npz` حاوی یک باینری فشرده است که شامل یک مجموعه از آرایه‌ها و داده‌های متناظر با نام‌های آن‌ها است.

288-- `loadtxt(fname[, dtype, comments, delimiter, ...])`:

این تابع برای خواندن داده‌های موجود در یک فایل متنی به صورت ماتریس استفاده می‌شود. آرگومان‌های این تابع عبارتند از:

- `fname`: نام فایل یا مسیر فایل که باید از آن داده‌ها خوانده شوند.

- `dtype` (اختیاری): نوع داده مورد انتظار برای ذخیره داده‌ها. اگر این آرگومان مشخص نشود، نوع داده به صورت خودکار تشخیص داده می‌شود.

- `comments` (اختیاری): کاراکترهایی که به عنوان نشانگرهای کامنت در فایل در نظر گرفته می‌شوند.

- `delimiter` (اختیاری): کاراکتری که برای جدا کردن مقادیر در فایل متنی استفاده می‌شود. مقدار پیش‌فرض آن فاصله است.

- وجود آرگومان‌های دیگری نیز امکان‌پذیر است که مشخصات بیشتری برای خواندن داده‌ها فراهم می‌کنند.

289-تابع `ndarray.tofile(fid[, sep, format])` برای ذخیره کردن محتوای یک آرایه numpy در یک فایل باینری استفاده می‌شود. آرگومان `fid` نشانگر شیء فایلی است که باید از آن خروجی بگیریم و محتوای آرایه را در آن ذخیره کنیم. آرگومانهای `sep` و `format` به طور اختیاری هستند.

- `fid`: شیء فایل باینری که باید به آن داده‌ها را ذخیره کنیم. معمولاً می‌توان از توابع مانند `open` برای ایجاد یک فایل باینری استفاده کرد و سپس آن را به عنوان `fid` به تابع `tofile` پاس داد.

- `sep` (اختیاری): رشته جداکننده بین عناصر آرایه در فایل باینری. مقدار پیش‌فرض برابر با فاصله است.

- `format` (اختیاری): فرمت مشخص کننده نوع داده‌ها در فایل باینری. مقدار پیش‌فرض برابر با `"%s"` است که به معنای استفاده از فرمت پیش‌فرض برای نوع داده‌ها است.

این تابع با استفاده از `fid`، محتوای آرایه numpy را به فایل باینری ارسال می‌کند و آن را ذخیره می‌کند. این عملیات برای ذخیره سازی داده‌های بزرگ و چندبعدی با سرعت بالا مفید است.

290-تابع `fromfile(file[, dtype, count, sep, offset, like])` به کار می‌رود تا داده‌های موجود در یک فایل باینری را به صورت یک آرایه numpy بخواند. این تابع به طور معمول بعد از استفاده از تابع `tofile` برای ذخیره داده‌ها در فایل باینری، برای بازخوانی آن داده‌ها استفاده می‌شود. آرگومانهای `dtype`، `count`، `sep`، `offset` و `like` به طور اختیاری هستند.

- `file`: شیء فایل باینری که باید از آن خوانش انجام دهیم.

- `dtype` (اختیاری): نوع داده‌هایی که در فایل باینری وجود دارد و باید برای خوانش آنها استفاده شود. این پارامتر اختیاری است و اگر مقداردهی نشود، نوع داده پیش‌فرض برای آرایه numpy استفاده می‌شود.

- `count` (اختیاری): تعداد عناصری که باید از فایل بخوانیم. این پارامتر اختیاری است و اگر مقداردهی نشود، تمام عناصر موجود در فایل خوانده می‌شوند.

- `sep` (اختیاری): رشته جداکننده بین عناصر در فایل باینری. این پارامتر اختیاری است و مقدار پیش‌فرض برابر با فاصله است.

- `offset` (اختیاری): آفست (offset) بایتی که باید در فایل باینری جستجو شود تا از آنجا خواندن داده‌ها آغاز شود. این پارامتر اختیاری است و اگر مقداردهی نشود، خواندن داده‌ها از ابتدای فایل شروع می‌شود.

- `like` (اختیاری): یک آرایه numpy که به عنوان الگو برای تعیین نوع داده و شکل آرایه جدید استفاده می‌شود. این پارامتر اختیاری است .

291-

تابع `ndarray.tofile(fid[, sep, format])` به کار می‌رود تا داده‌های یک آرایه numpy را به صورت باینری در یک فایل ذخیره کند. آرگومانهای `fid`، `sep` و `format` اختیاری هستند.

- `fid`: شیء فایل که باید داده‌ها در آن ذخیره شوند. می‌توان از نوع فایل، رشته نام فایل یا شیء file-like برای این پارامتر استفاده کرد.

- `sep` (اختیاری): رشته جداکننده بین عناصر در فایل باینری. این پارامتر اختیاری است و مقدار پیش‌فرض برابر با فاصله است.

- `format` (اختیاری): فرمت باینری برای ذخیره داده‌ها. می‌توان از قالب‌های مشخص شده در دستور `struct` استفاده کرد. این پارامتر اختیاری است و مقدار پیش‌فرض برابر با `"%s"` است که به معنای نوشتن داده‌ها در فرمت پیش‌فرض نوع داده آرایه numpy است.

با استفاده از تابع `tofile`، می‌توانید داده‌های یک آرایه numpy را به صورت باینری در یک فایل ذخیره کنید. این تابع برای ذخیره سازی داده‌هایی که بعداً با استفاده از تابع `fromfile` قابل بازیابی هستند، بسیار مفید است.

-292تابع `fromfile(file[, dtype, count, sep, offset, like])` به کار می‌رود تا داده‌های موجود در یک فایل باینری را به صورت یک آرایه numpy بخواند. این تابع به طور معمول بعد از استفاده از تابع `tofile` برای ذخیره داده‌ها در فایل باینری، برای بازخوانی آن داده‌ها استفاده می‌شود. آرگومانهای `dtype`، `count`، `sep`، `offset` و `like` به طور اختیاری هستند.

- `file`: شیء فایل باینری که باید از آن خوانش انجام دهیم.

- `dtype` (اختیاری): نوع داده‌هایی که در فایل باینری وجود دارد و باید برای خوانش آنها استفاده شود. این پارامتر اختیاری است و اگر مقداردهی نشود، نوع داده پیش‌فرض برای آرایه numpy استفاده می‌شود.

- `count` (اختیاری): تعداد عناصری که باید از فایل بخوانیم. این پارامتر اختیاری است و اگر مقداردهی نشود، تمام عناصر موجود در فایل خوانده می‌شوند.

- `sep` (اختیاری): رشته جداکننده بین عناصر در فایل باینری. این پارامتر اختیاری است و مقدار پیش‌فرض برابر با فاصله است.

- `offset` (اختیاری): آفست (offset) بایتی که باید در فایل باینری جستجو شود تا از آنجا خواندن داده‌ها آغاز شود. این پارامتر اختیاری است و اگر مقداردهی نشود، خواندن داده‌ها از ابتدای فایل شروع می‌شود.

- `like` (اختیاری): یک آرایه numpy که به عنوان الگو برای تعیین نوع داده و شکل آرایه جدید استفاده می‌شود. این پارامتر اختیاری است .

-293تابع `ndarray.tofile(fid[, sep, format])` به کار می‌رود تا داده‌های یک آرایه numpy را به صورت باینری در یک فایل ذخیره کند. آرگومانهای `fid`، `sep` و `format` اختیاری هستند.

- `fid`: شیء فایل که باید داده‌ها در آن ذخیره شوند. می‌توان از نوع فایل، رشته نام فایل یا شیء file-like برای این پارامتر استفاده کرد.

- `sep` (اختیاری): رشته جداکننده بین عناصر در فایل باینری. این پارامتر اختیاری است و مقدار پیش‌فرض برابر با فاصله است.

- `format` (اختیاری): فرمت باینری برای ذخیره داده‌ها. می‌توان از قالب‌های مشخص شده در دستور `struct` استفاده کرد. این پارامتر اختیاری است و مقدار پیش‌فرض برابر با `"%s"` است که به معنای نوشتن داده‌ها در فرمت پیش‌فرض نوع داده آرایه numpy است.

با استفاده از تابع `tofile`، می‌توانید داده‌های یک آرایه numpy را به صورت باینری در یک فایل ذخیره کنید. این تابع برای ذخیره سازی داده‌هایی که بعداً با استفاده از تابع `fromfile` قابل بازیابی هستند، بسیار مفید است.

-294توابع `array2string(a[, max\_line\_width, precision, ...])` و `array\_repr(arr[, max\_line\_width, precision, ...])` در numpy برای تبدیل یک آرایه به رشته استفاده می‌شوند.

1. `array2string(a[ max\_line\_width, precision, ...])`:

- `a`: آرایه ورودی که باید به رشته تبدیل شود.

- `max\_line\_width` (اختیاری): عرض حداکثر خط در خروجی رشته. اگر طول خط بیشتر از این مقدار شود، خط بعدی شروع می‌شود. پیش‌فرض برابر با `75` است.

- `precision` (اختیاری): تعداد اعشار مجاز در اعداد. اگر مقدار `-1` باشد، همه اعداد با دقت کامل نمایش داده می‌شوند. پیش‌فرض برابر با `None` است که به معنای استفاده از تنظیمات دقت پیش‌فرض است.

تابع `array2string` یک رشته حاوی نمایش آرایه ورودی را برمی‌گرداند. این تابع می‌تواند به عنوان یک روش سفارشی برای نمایش داده‌های آرایه با استفاده از تنظیمات مانند محدودیت طول خط و دقت اعداد مورد استفاده قرار گیرد.

-295

2. `array\_repr(arr[, max\_line\_width, precision, ...])`:

- `arr`: آرایه ورودی که باید به رشته تبدیل شود.

- `max\_line\_width` (اختیاری): عرض حداکثر خط در خروجی رشته. اگر طول خط بیشتر از این مقدار شود، خط بعدی شروع می‌شود. پیش‌فرض برابر با `75` است.

- `precision` (اختیاری): تعداد اعشار مجاز در اعداد. اگر مقدار `-1` باشد، همه اعداد با دقت کامل نمایش داده می‌شوند. پیش‌فرض برابر با `None` است که به معنای استفاده از تنظیمات دقت پیش‌فرض است.

تابع `array\_repr` به صورت پیش‌فرض از تابع `repr` استفاده می‌کند تا نمایش رشته‌ای آرایه را ایجاد کند. تابع `repr` برای تبدیل یک شیء به رشته‌ای قابل قبول برای نمایش در کد Python استفاده می‌شود. نمایش ایجاد شده توسط `array\_repr` حاوی اطلاعات مختصری از آرایه است، از جمله ابعاد آرایه و مقادیر عناصر آن.

برای مثال، وقتی از تابع `array\_repr` برای یک آرایه دوبعدی استفاده می‌شود، خروجی ممکن است به شکل زیر باشد:

```

array([[1, 2, 3],

[4, 5, 6]])

```

در این خروجی، ابعاد آرایه به همراه مقادیر عناصر آن به صورت قالب بندی شده نمایش داده می‌شود. این نمایش می‌تواند به عنوان یک راه ساده برای مشاهده سریع محتوای آرایه در کد Python مورد استفاده قرار گیرد.

-296تابع `array\_str(arr,max-line-width,precision)` برای تبدیل یک آرایه به رشته‌ای قابل خواندن توسط انسان استفاده می‌شود. این تابع نمایشی خوانا از آرایه را ایجاد می‌کند و در آن مقادیر عناصر آرایه به صورت مناسب قالب بندی می‌شوند. می‌توانید از آرگومان‌های مختلفی مانند `max\_line\_width` و `precision` برای تنظیم نحوه نمایش و قالب بندی آرایه استفاده کنید.

به عنوان مثال، وقتی از تابع `array\_str` برای یک آرایه دوبعدی استفاده می‌شود، خروجی ممکن است به شکل زیر باشد:

```

[[1 2 3]

[4 5 6]]

```

در این خروجی، مقادیر عناصر آرایه به صورت قالب بندی شده و بدون استفاده از فاصله بین عناصر نمایش داده می‌شوند. این نمایش برای خوانایی بیشتر از آرایه در متن‌ها و خروجی‌های مشابه مناسب است.

-297تابع `format\_float\_scientific(values,precision)` برای قالب بندی اعداد به صورت علمی استفاده می‌شود.

این تابع یک عدد اعشاری را به صورت علمی با استفاده از نمایش علمی با پایه 10 به صورت رشته تبدیل می‌کند. می‌توانید با استفاده از آرگومان `precision` تعیین کنید که به چند رقم اعشاری نمایش داده شود. .

این تابع برای نمایش اعداد کوچک و بزرگ با دقت و قالب بندی صحیح در نمایش علمی مناسب است و به شما اجازه می‌دهد نمایش دقیق را براساس مقادیر ورودی تنظیم کنید.

-298تابع `set\_printoptions(precision,threshold,…)` در NumPy برای تنظیم گزینه‌های چاپ آرایه‌ها استفاده می‌شود. این تابع به شما امکان می‌دهد گزینه‌های مربوط به چاپ آرایه‌ها را تغییر دهید، مانند تعیین تعداد ارقام اعشار، تعداد عناصر قابل نمایش در یک آرایه و موارد مشابه.

آرگومان‌های این تابع عبارتند از:

- `precision` (اختیاری): تعداد ارقام اعشار برای نمایش اعداد. مقدار پیش‌فرض آن `8` است.

- `threshold` (اختیاری): حداقل تعداد عناصری که باید در یک آرایه باشد تا به صورت خلاصه چاپ شود. اگر آرایه بیشتر از این تعداد عنصر داشته باشد، نمایش خلاصه‌ای از آرایه ارائه نمی‌شود. مقدار پیش‌فرض آن `1000` است.

- سایر آرگومان‌ها (اختیاری): شامل `threshold` و سایر گزینه‌های دیگری که می‌تواند بر روی چاپ آرایه‌ها تأثیر بگذارد.

با استفاده از این تابع، شما می‌توانید تنظیمات مورد نظر خود را برای چاپ آرایه‌ها تعیین کنید تا خروجی چاپ شده با نیازهای خود هماهنگ باشد.

-299تابع `get\_printoptions()` در NumPy برای دریافت تنظیمات فعلی چاپ آرایه‌ها استفاده می‌شود. با فراخوانی این تابع، شما می‌توانید مقادیر فعلی گزینه‌های مربوط به چاپ را به عنوان خروجی دریافت کنید.

-300تابع `set\_string\_function` برای تعیین تابع نمایش رشته برای آرایه‌ها استفاده می‌شود. این تابع به شما امکان می‌دهد تابعی را برای نمایش هر عنصر از آرایه به شکل یک رشته تعیین کنید.

set\_string\_function(f[, repr])

f: یک تابع یا متد قابل فراخوانی (callable) که برای تبدیل عناصر آرایه به رشته استفاده می‌شود. این تابع باید یک پارامتر عنصر آرایه را دریافت کند و رشته متناظر را برگرداند.

repr (اختیاری): یک علامت بولین که نشان می‌دهد آیا تابع مشخص شده برای تبدیل عناصر به رشته باید به عنوان تابع repr استفاده شود یا خیر.

-301تابع `printoptions` به شما امکان می‌دهد تنظیمات چاپ را در یک فراخوانی تعیین کنید. شما می‌توانید مقادیر مورد نظر برای تنظیمات را به صورت آرگومان‌های متغیر (به صورت `key=value`) به این تابع ارسال کنید. این تابع به شما امکان می‌دهد تنظیمات چاپ را در یک محدوده محلی تغییر دهید و در پایان اجرای بلاک مربوطه به تنظیمات اصلی برگردید.

-printoptions(\*args, \*\*kwargs)

این تابع قابلیت تنظیم گزینه‌های چاپ را در یک فراخوانی تعیین می‌کند. آرگومان‌های متغیر (به صورت key=value) می‌توانند به عنوان ورودی به این تابع داده شوند تا گزینه‌های چاپ مورد نظر تعیین شود. برخی از آرگومان‌های مهم عبارتند از:

precision: تعیین تعداد ارقام اعشار برای نمایش اعداد ممیزشناور.

threshold: تعیین حد بزرگی برای نمایش آرایه‌ها به صورت خلاصه.

linewidth: تعیین طول حداکثر خطوط چاپ شده.

از این توابع برای کنترل نحوه نمایش آرایه‌ها در خروجی استفاده کنید تا نتایج چاپ شده با نیازهای خود هماهنگ باشند.

-302

- `binary\_repr(num[, width])`:

این تابع رشته‌ای متناظر با عدد صحیح غیر‌منفی در سیستم عدددهی دودویی (باینری) برمی‌گرداند. آرگومان `num` عدد ورودی است که باید غیرمنفی باشد. آرگومان اختیاری `width` نشان‌دهنده تعداد بیت‌های مورد استفاده برای نمایش عدد باینری است. اگر عرض تعیین نشده باشد، طول رشته خروجی برابر با طول باینری بدون صفرهای بی‌معنی عدد ورودی خواهد بود.

303-- `base\_repr(number[, base, padding])`:

این تابع رشته‌ای متناظر با عدد ورودی در سیستم عدددهی مبدأ دلخواه (با تعیین پایه) برمی‌گرداند.

آرگومان `number` عدد ورودی است که باید صحیح و غیرمنفی باشد.

آرگومان اختیاری `base` نشان‌دهنده پایه سیستم عدددهی است و به صورت پیش‌فرض برابر 10 است (سیستم دهدهی).

آرگومان اختیاری `padding` نشان‌دهنده تعداد رقم‌های حاشیه‌ای مورد استفاده در نمایش عدد است. اگر عرض تعیین نشده باشد، طول رشته خروجی برابر با طول عدد در سیستم عدددهی مبدأ خواهد بود و اگر طول عدد کوچکتر از `padding` باشد، صفرهای حاشیه‌ای به آن اضافه خواهند شد.

304-- `array\_repr(arr[, max\_line\_width, precision, suppress\_small])`:

این تابع نمایش رشته‌ای معرفی (repr) یک آرایه numpy را برمی‌گرداند.

آرگومان `arr` آرایه‌ای numpy است که باید نمایش داده شود.

آرگومان اختیاری `max\_line\_width` نشان‌دهنده حداکثر عرض خط در نمایش رشته است. اگر طول رشته خروجی بیشتر از `max\_line\_width` باشد، خطوط جدید به صورت خودکار درج می‌شوند.

آرگومان اختیاری `precision` نشان‌دهنده تعداد ارقام اعشاری در نمایش اعداد است.

آرگومان اختیاری `suppress\_small` کنترل می‌کند که اعداد بسیار کوچک نمایش داده شوند یا با عبارت "..." نشان داده شوند.

305-- `array\_str(a[, max\_line\_width, precision, suppress\_small, threshold])`:

این تابع رشته‌ای متناظر با یک آرایه numpy را برمی‌گرداند.

آرگومان `a` آرایه‌ای numpy است که باید نمایش داده شود.

آرگومان اختیاری `max\_line\_width` نشان‌دهنده حداکثر عرض خط در نمایش رشته است. اگر طول رشته خروجی بیشتر از `max\_line\_width` باشد، خطوط جدید به صورت خودکار درج می‌شوند.

آرگومان اختیاری `precision` نشان‌دهنده تعداد ارقام اعشاری در نمایش اعداد است. آرگومان اختیاری `suppress\_small` کنترل می‌کند که اعداد بسیار کوچک نمایش داده شوند یا با عبارت "..." نشان داده شوند. آرگومان اختیاری `threshold` تعیین می‌کند که آیا یک آرایه بزرگ به صورت خلاصه نمایش داده شود یا خیر.

306-- `DataSource([destpath])`:

این کلاس یک منبع داده را نمایش می‌دهد که به صورت ترکیبی از مسیر مقصد و پیش‌ویژه است.

آرگومان اختیاری `destpath` مسیر مقصد را نشان می‌دهد. اگر `destpath` مقدار نداشته باشد، مسیر مقصد خالی خواهد بود.

در نمونه‌ای از استفاده از این کلاس، می‌توان فایل‌های داده را از منبعی مشخص به صورت آنلاین یا آفلاین دریافت کرد.

307-- `lib.format()`:

این ماژول یک زیرماژول داخلی برای فرمت‌بندی داده‌های آرایه است. این زیرماژول حاوی توابعی است که برای فرمت‌بندی و نمایش داده‌های آرایه به کار می‌روند.

توابع موجود در این زیرماژول می‌توانند به عنوان واسطی برای تعیین روش نمایش و فرمت‌بندی داده‌های آرایه در حالت پیش‌فرض استفاده شوند. این توابع به عنوان توابع داخلی استفاده می‌شوند و معمولاً در دستورات دیگری مانند توابع `array2string`، `array\_repr` و `array\_str` بکار می‌روند.

منظور از فرمت‌بندی داده‌های آرایه، تعیین نحوه نمایش داده‌های آرایه در قالب رشته‌ها است. این فرمت‌بندی می‌تواند شامل تعیین تعداد ارقام اعشار، قواعد نمایش عددی، تعداد رقم‌ها و ... باشد.

308-تابع `dot(a,b)` یک عملگر ضرب داخلی (dot product) را بین دو آرایه یا بین یک بردار و یک آرایه انجام می‌دهد. ورودی‌های این تابع عبارتند از:

- `a`: آرایه ورودی اول، که می‌تواند یک بردار یا یک آرایه باشد.

- `b`: آرایه ورودی دوم، که می‌تواند یک بردار یا یک آرایه باشد.

- `out` (اختیاری): آرایه خروجی که در آن نتیجه عملگر ضرب داخلی قرار می‌گیرد. اگر این آرایه مشخص نشود، یک آرایه جدید برای ذخیره نتیجه ایجاد می‌شود.

تابع `dot` ضرب داخلی بین دو آرایه را محاسبه می‌کند. برای بردارها، ضرب داخلی به صورت محصول نقطه‌ای دو بردار تعریف می‌شود، و برای آرایه‌ها، ضرب داخلی به صورت جمع محصول عناصر متناظر دو آرایه تعریف می‌شود.

تابع `dot` برای انواع داده‌ها از جمله آرایه‌ها و بردارهای چند بعدی قابل استفاده است و به صورت خودکار اندازه‌ها را بررسی می‌کند تا ضرب داخلی معتبر انجام شود.

توابع `multi\_dot` و `vdot` همگی مربوط به عملیات خطی و ماتریسی در ماژول `linalg` از NumPy هستند.

309-تابع `multi\_dot(array,out)` برای ضرب ماتریس‌ها در یک زنجیره، از الگوریتم بهینه‌سازی شده‌ای استفاده می‌کند.

آرگومان `arrays` نشان‌دهنده آرایه‌های ورودی است که باید به ترتیب ضرب شوند. اختیاری استفاده می‌شود و در صورت مشخص نشدن، تمام آرگومان‌ها را به عنوان یک زنجیره ضرب ماتریسی در نظر می‌گیرد.

آرگومان `out` همچنین اختیاری است و نشان‌دهنده آرایه خروجی است که در آن نتیجه عملیات ضرب قرار می‌گیرد.

310-تابع `vdot(a,b)` نیز ضرب داخلی دو بردار را محاسبه می‌کند. برخلاف تابع `dot` که ضرب داخلی بین آرایه‌ها را محاسبه می‌کند، تابع `vdot` فقط برای بردارها قابل استفاده است. بردارهای ورودی `a` و `b` باید هر دو یک بعدی باشند. نتیجه عملیات ضرب داخلی به عنوان یک عدد صحیح یا ممیز شناور در خروجی بازگردانده می‌شود.

هر دو تابع `multi\_dot` و `vdot` برای عملیات خطی و ماتریسی مورد استفاده قرار می‌گیرند و در محاسبات عددی و ریاضیات ماتریسی مفید هستند.

تابع `inner` و `outer` نیز مربوط به عملیات خطی در ماژول `linalg` از NumPy هستند.

311-تابع `inner(a,b)` برای محاسبه ضرب داخلی دو بردار استفاده می‌شود. ضرب داخلی بین دو بردار `a` و `b` محاسبه می‌شود و یک عدد صحیح یا ممیز شناور به عنوان خروجی بازگردانده می‌شود.

312-تابع `outer(a,b)` برای محاسبه ضرب خارجی دو بردار استفاده می‌شود. ضرب خارجی بین دو بردار `a` و `b` محاسبه می‌شود و یک ماتریس خروجی بازگردانده می‌شود. ابعاد ماتریس خروجی برابر با طول بردار `a` در طول بردار `b` است. مقدار هر عنصر در ماتریس خروجی برابر با ضرب متناظر از عناصر متناظر در دو بردار است.

هر دو تابع `inner` و `outer` برای محاسبات خطی و ماتریسی مورد استفاده قرار می‌گیرند و در عملیات‌های آماری، ریاضیات ماتریسی و پردازش سیگنال مفید هستند.

313-تابع `matmul(x1,x2)` نیز برای ضرب ماتریسی دو آرای `x1` و `x2` در NumPy استفاده می‌شود.

برای ضرب ماتریسی، دو آرای ورودی باید ساختار ماتریسی داشته باشند. اگر `x1` یک آرای دوبعدی با ابعاد (n × m) باشد و `x2` یک آرای دوبعدی با ابعاد (m × p) باشد، آرای خروجی ماتریسی با ابعاد (n × p) به دست می‌آید.

ضرب ماتریسی در `matmul` به صورت دقیق انجام می‌شود و نیازی به تبدیل آرای ورودی به ماتریس قبل از انجام عملیات ندارد. همچنین، این تابع قابلیت پشتیبانی از آرای چند بعدی را نیز دارد و می‌تواند برای ضرب تانسورها نیز استفاده شود.

تابع `matmul` می‌تواند برای محاسبه ضرب ماتریسی در بسیاری از برنامه‌ها مورد استفاده قرار بگیرد، از جمله پردازش تصویر، محاسبات عددی، شبکه‌های عصبی و بسیاری دیگر از زمینه‌های علوم کامپیوتر و مهندسی.

314-تابع `tensordot(a,b,axes)` در NumPy برای محاسبه محصول تانسوری دو آرای چند بعدی `a` و `b` استفاده می‌شود.

محصول تانسوری دو آرای چند بعدی، یک عمل ترکیبی است که با استفاده از جمع و ضرب داخلی، یک تانسور جدید ایجاد می‌کند. برای محاسبه محصول تانسوری، محورهای مشخصی در آرای ورودی باید با یکدیگر ترکیب شوند.

آرگومان `axes` مشخص می‌کند که کدام محورها در هر آرای ورودی باید با یکدیگر ترکیب شوند. این آرگومان می‌تواند مقدارهای متنوعی داشته باشد:

- اگر `axes` یک عدد صحیح باشد، آنگاه محورهای `axes` در `a` و `b` با یکدیگر ترکیب می‌شوند.

- اگر `axes` یک زوج از لیست‌ها باشد، آنگاه هر زوج (ax, bx) در `axes` نشان دهنده ترکیب محورهای `ax` در `a` و `bx` در `b` است.

- اگر `axes` یک عدد صحیح و یک لیست باشد، آنگاه محور `axes` در `a` با همه محورهای `axes` در `b` ترکیب می‌شود.

- اگر `axes` یک لیست باشد، آنگاه محورهای مشخص شده در `axes` در هر آرای ورودی با یکدیگر ترکیب می‌شوند.

تابع `tensordot` نتیجه محصول تانسوری را بازگردانده و معمولاً در محاسبات عددی پیچیده و در زمینه‌هایی مانند یادگیری ماشین، شبکه‌های عصبی و پردازش تصویر مورد استفاده قرار می‌گیرد.

315-einsum(subscripts, \*operands[, out, dtype, ...])

تابع `einsum` در NumPy برای انجام عملیات سطری‌بندی (einsum-style) روی آرای چند بعدی استفاده می‌شود. سطری‌بندی یک نمایش کامپیوتری است که به کاربر امکان می‌دهد تا عملیات ریاضی را بر روی آرای چند بعدی به صورت خلاصه توصیف کند.

آرگومان اول `subscripts` رشته‌ای است که شامل سطری‌بندی است. سطری‌بندی از قواعد خاصی پیروی می‌کند که برای تعیین نحوه ترکیب عناصر آرای ورودی استفاده می‌شود. هر حرف در سطری‌بندی نمایانگر یک محور در آرای ورودی است و عملیات مورد نظر بر روی آن انجام می‌شود.

آرگومان دوم به بعد، `\*operands`، آرای ورودی است که روی آنها عملیات سطری‌بندی انجام می‌شود.

آرگومان اختیاری `out` آرای خروجی را مشخص می‌کند که در آن نتیجه عملیات سطری‌بندی ذخیره می‌شود. آرگومان `dtype` نوع داده خروجی را تعیین می‌کند و `order` نحوه مرتب‌سازی عناصر در حافظه را مشخص می‌کند.

تابع `einsum` بسیار قدرتمند است و به کاربر امکان می‌دهد تا تنظیمات مختلفی را برای عملیات سطری‌بندی انجام دهد. این تابع در بسیاری از بخش‌های علوم محاسباتی، جبر خطی و پردازش داده‌های چند بعدی استفاده می‌شود.

316-

einsum\_path(subscripts, \*operands[, optimize])

تابع `einsum\_path` در NumPy برای بهینه‌سازی عملیات سطری‌بندی (einsum-style) استفاده می‌شود. این تابع یک مسیر بهینه برای اجرای عملیات سطری‌بندی بر روی آرای ورودی را محاسبه می‌کند.

آرگومان اول `subscripts` رشته‌ای است که شامل سطری‌بندی است. سطری‌بندی از قواعد خاصی پیروی می‌کند که برای تعیین نحوه ترکیب عناصر آرایه ورودی استفاده می‌شود. هر حرف در سطری‌بندی نمایانگر یک محور در آرایه ورودی است و عملیات مورد نظر بر روی آن انجام می‌شود.

آرگومان دوم به بعد، `\*operands`، آرایه ورودی است که روی آنها عملیات سطری‌بندی انجام می‌شود.

آرگومان اختیاری `optimize` نشان می‌دهد که آیا تابع `einsum\_path` باید مسیر بهینه را محاسبه کند یا نه. با تنظیم این آرگومان به `True`، تابع تلاش می‌کند برای یافتن مسیر بهینه اجرای عملیات سطری‌بندی. با تنظیم به `False`، تابع فقط طرحی را برای اجرای عملیات ارائه می‌دهد.

تابع `einsum\_path` برای بهبود کارایی و عملکرد عملیات سطری‌بندی، به ویژه در مواردی که آرای ورودی بزرگ باشند، بسیار مفید است.

320-تابع `matrix\_power(a, n)` در ماژول `numpy.linalg` برای محاسبه توان ماتریس استفاده می‌شود. این تابع ماتریس وارد شده `a` را به توان `n` می‌برد و ماتریس حاصل را برمی‌گرداند. آرگومان `a` ماتریس ورودی است و آرگومان `n` نشان دهنده توان است که باید به `a` اعمال شود. توجه داشته باشید که `n` باید یک عدد صحیح باشد.

321-تابع `kron(a, b)` نیز در ماژول `numpy` برای محاسبه حاصلضرب کرونکر میان دو آرایه`a` و `b` استفاده می‌شود. حاصلضرب کرونکر دو آرای `a` و `b` یک ماتریس جدید تشکیل می‌دهد که ابعاد آن برابر با حاصلضرب ابعاد آرای اولیه می‌باشد. مقدار هر عنصر در ماتریس جدید برابر با حاصلضرب مقادیر متناظر عناصر ماتریس اولیه است.

Decompositions:

322-. linalg.cholesky(a):

- a: آرایهٔ ورودی. باید ماتریس معین مثبت (ماتریس چهارگانه) باشد.

تابع linalg.cholesky ماتریس ورودی a را به صورت چولسکی (Cholesky) فاکتوریزه می‌کند. در خروجی، ماتریس فوق مثلثی L برگردانده می‌شود که برآورد ماتریس a است.

323-. linalg.qr(a[, mode]):

- a: آرایهٔ ورودی. باید یک ماتریس باشد.

- mode (اختیاری): حالت محاسبهٔ فاکتوری QR. مقدار پیش‌فرض برابر "reduced" است.

تابع linalg.qr فاکتوری QR را برای ماتریس ورودی a محاسبه می‌کند. در خروجی، ماتریس Q و ماتریس فوق سه‌ضلعی R برگردانده می‌شود که برآوردهای ماتریس a هستند.

324-. linalg.svd(a[, full\_matrices, compute\_uv, ...]):

- a: آرایهٔ ورودی. باید یک ماتریس باشد.

- full\_matrices (اختیاری): آیا ماتریس‌های U و V با ابعاد کامل برگردانده شوند یا نه. مقدار پیش‌فرض برابر True است.

- compute\_uv (اختیاری): آیا ماتریس‌های U و V برگردانده شوند یا نه. مقدار پیش‌فرض برابر True است.

- overwrite\_a (اختیاری): آیا آرایهٔ ورودی a به صورت غیرقابل تغییر تغییر یابد یا نه. مقدار پیش‌فرض برابر False است.

Solving equations and inverting matrices:

325-. linalg.solve(a, b):

- a: آرایهٔ ماتریس واگرایی با ابعاد (n, n).

- b: آرایهٔ ماتریس مقدارهای وابسته با ابعاد (n, m).

تابع linalg.solve برای حل سیستم معادلات خطی a \* x = b، جواب x را محاسبه می‌کند. a نمایانگر ماتریس ضرایب سیستم معادلات است و b نشان‌دهندهٔ ماتریس مقدارهای وابسته است.

326-. linalg.tensorsolve(a, b[, axes]):

- a: آرایهٔ ماتریس واگرایی.

- b: آرایهٔ ماتریس مقدارهای وابسته.

- axes (اختیاری): محورهای جداول a و b که برای حل استفاده می‌شوند. اگر ارائه نشود، مقدار پیش‌فرض برابر None است.

تابع linalg.tensorsolve برای حل سیستم معادلات خطی با استفاده از روش تانسوری، جواب را محاسبه می‌کند. این تابع به طور کلی برای سیستم‌هایی با بعد بالاتر از دو بعد استفاده می‌شود.

327. linalg.lstsq(a, b[, rcond]):

- a: آرایهٔ ماتریس طراحی با ابعاد (m, n).

- b: آرایهٔ ماتریس مقدارهای وابسته با ابعاد (m,).

- rcond (اختیاری): مقدار آستانه برای صفربودن مقادیر تک‌مقداری ماتریس a. اگر مقدار داده نشود، از مقدار آستانه پیش‌فرض استفاده می‌شود.

تابع linalg.lstsq با استفاده از روش کمترین مربعات، بهینه‌ترین مقدار برای بردار x در معادلهٔ ax = b را پیدا می‌کند. این تابع با استفاده از روش نزول گرادیان برای حل معادله استفاده می‌کند.

328-. linalg.inv(a):

- a: آرایهٔ ماتریس قابل برگشت با ابعاد (n, n).

تابع linalg.inv ماتریس معکوس a را محاسبه می‌کند. اگر ماتریس ورودی واگرایی باشد یا دترمینان آن نزدیک به صفر باشد، خطای LinAlgError پرتاب می‌شود.

329-. linalg.pinv(a[, rcond, hermitian]):

- a: آرایهٔ ماتریس قابل برگشت با ابعاد (n, m).

- rcond (اختیاری): مقدار آستانه برای بررسی صفربودن مقادیر تک‌مقداری ماتریس a. اگر مقدار داده نشود، از مقدار آستانه پیش‌فرض استفاده می‌شود.

- hermitian (اختیاری): در صورت تنظیم برابر True، ماتریس a به عنوان یک ماتریس هرمیتی در نظر گرفته می‌شود. پیش‌فرض برابر False است.

تابع linalg.pinv معکوس تقریبی ماتریس a را محاسبه می‌کند. این تابع در حالتی که ماتریس a واگرا باشد یا دترمینان آن نزدیک به صفر باشد، استفاده می‌شود.

330-. linalg.tensorinv(a[, ind]):

- a: آرایهٔ ماتریس قابل برگشت.

- ind (اختیاری): مقدار نمایی که برای نیازهای ویژهٔ تانسوری استفاده می‌شود. پیش‌فرض برابر 2 است.

تابع linalg.tensorinv برای محاسبهٔ معکوس تانسور a با استفاده از روش تانسوری استفاده می‌شود.

Exceptions:

331-. linalg.LinAlgError:

linalg.LinAlgError یک استثناء است که در صورت بروز خطا در عملیات های جبرخطی در numpy.linalg پرتاب می‌شود. این استثناء برای مواردی مانند ماتریس واگرایی یا دترمینان نزدیک به صفر استفاده می‌شود.

Truth value testing:

332-. all(a[, axis, out, keepdims, where]):

- a: آرایهٔ ورودی.

- axis (اختیاری): محور یا محورهایی که روی آنها عملیات انجام می‌شود.

- out (اختیاری): آرایهٔ خروجی.

- keepdims (اختیاری): مشخص می‌کند که ابعاد آرایهٔ ورودی در آرایهٔ خروجی حفظ شود یا نه.

- where (اختیاری): شرطی که مشخص می‌کند کدام مقادیر آرایهٔ ورودی باید در نظر گرفته شود.

تابع all بررسی می‌کند که آیا تمام عناصر آرایهٔ ورودی a در شرطی که تعیین شده است، صحیح هستند یا نه. در صورت صحیح بودن شرط برای تمام عناصر، خروجی برابر True خواهد بود؛ در غیر این صورت، خروجی برابر False خواهد بود.

333-. any(a[, axis, out, keepdims, where]):

- a: آرایهٔ ورودی.

- axis (اختیاری): محور یا محورهایی که روی آنها عملیات انجام می‌شود.

- out (اختیاری): آرایهٔ خروجی.

- keepdims (اختیاری): مشخص می‌کند که ابعاد آرایهٔ ورودی در آرایهٔ خروجی حفظ شود یا نه.

- where (اختیاری): شرطی که مشخص می‌کند کدام مقادیر آرایهٔ ورودی باید در نظر گرفته شود.

تابع any بررسی می‌کند که آیا حداقل یکی از عناصر آرایهٔ ورودی a در شرطی که تعیین شده است، صحیح هستند یا نه. در صورت صحیح بودن شرط برای حداقل یک عنصر، خروجی برابر True خواهد بود؛ در غیر این صورت، خروجی برابر False خواهد بود.

Array contents:

334-. isfinite(x, /[, out, where, casting, order, ...]):

- x: آرایهٔ ورودی.

- out (اختیاری): آرایهٔ خروجی.

- where (اختیاری): شرطی که مشخص می‌کند کدام مقادیر آرایهٔ ورودی باید در نظر گرفته شود.

- casting (اختیاری): روش تبدیل مقادیر در صورت نیاز.

- order (اختیاری): حالت مرتب سازی آرایهٔ خروجی.

تابع isfinite بررسی می‌کند که آیا عناصر آرایهٔ ورودی x عدد متناهی (finite) هستند یا نه. عناصری که عدد متناهی هستند، trueبازگشت داده می‌شوند و در غیر این صورت بازگشت داده False خواهد بود.

335-. isinf(x, /[, out, where, casting, order, ...]):

- x: آرایهٔ ورودی.

- out (اختیاری): آرایهٔ خروجی.

- where (اختیاری): شرطی که مشخص می‌کند کدام مقادیر آرایهٔ ورودی باید در نظر گرفته شود.

- casting (اختیاری): روش تبدیل مقادیر در صورت نیاز.

- order (اختیاری): حالت مرتب سازی آرایهٔ خروجی.

تابع isinf بررسی می‌کند که آیا عناصر آرایهٔ ورودی x بی‌نهایت (infinity) هستند یا نه. عناصری که بی‌نهایت هستند، trueبازگشت داده می‌شوند و در غیر این صورت بازگشت داده False خواهد بود.

336-. isnan(x, /[, out, where, casting, order, ...]):

- x: آرایهٔ ورودی.

- out (اختیاری): آرایهٔ خروجی.

- where (اختیاری): شرطی که مشخص می‌کند کدام مقادیر آرایهٔ ورودی باید در نظر گرفته شود.

- casting (اختیاری): روش تبدیل مقادیر در صورت نیاز.

- order (اختیاری): حالت مرتب سازی آرایهٔ خروجی.

تابع isnan بررسی می‌کند که آیا عناصر آرایهٔ ورودی x ناعدد (NaN) هستند یا نه. عناصری که ناعدد هستند، بازگشت داده می‌شوند و در غیر این صورت بازگشت داده False خواهد بود.

337-. isnat(x, /[, out, where, casting, order, ...]):

- x: آرایهٔ ورودی.

- out (اختیاری): آرایهٔ خروجی.

- where (اختیاری): شرطی که مشخص می‌کند کدام مقادیر آرایهٔ ورودی باید در نظر گرفته شود.

- casting (اختیاری): روش تبدیل مقادیر در صورت نیاز.

- order (اختیاری): حالت مرتب سازی آرایهٔ خروجی.

تابع isnat بررسی می‌کند که آیا عناصر آرایهٔ ورودی x ناقص زمانی (NaT) هستند یا نه. عناصری که ناقص زمانی هستند، trueبازگشت داده می‌شوند و در غیر این صورت بازگشت داده False خواهد بود.

338-. isneginf(x[, out]):

- x: آرایهٔ ورودی.

- out (اختیاری): آرایهٔ خروجی.

تابع isneginf بررسی می‌کند که آیا عناصر آرایهٔ ورودی x بی‌نهایت منفی (negative infinity) هستند یا نه. عناصری که بی‌نهایت منفی هستند،true بازگشت داده می‌شوند و در غیر این صورت بازگشت داده False خواهد بود.

339-. isposinf(x[, out]):

- x: آرایهٔ ورودی.

- out (اختیاری): آرایهٔ خروجی.

تابع isposinf بررسی می‌کند که آیا عناصر آرایهٔ ورودی x بی‌نهایت مثبت (positive infinity) هستند یا نه. عناصری که بی‌نهایت مثبت هستند، trueبازگشت داده می‌شوند و در غیر این صورت بازگشت داده False خواهد بود.

Array type testing:

340-. iscomplex(x):

- x: آرایهٔ ورودی.

تابع iscomplex بررسی می‌کند که آیا عناصر آرایهٔ ورودی x اعداد مختلط هستند یا نه. در صورتی که عناصر حاوی اعداد مختلط باشند، خروجی برابر True خواهد بود؛ در غیر این صورت، خروجی برابر False خواهد بود.

341-

. iscomplexobj(x):

- xآرایهٔ ورودی.

تابع iscomplexobj بررسی می‌کند که آیا آرایهٔ ورودی x حاوی اعداد مختلط یا شیء مختلط (complex object) است یا نه. در صورتی که x حاوی اعداد مختلط یا شیء مختلط باشد، خروجی برابر True خواهد بود؛ در غیر این صورت، خروجی برابر False خواهد بود.

342-. isfortran(a):

- a: آرایهٔ ورودی.

تابع isfortran بررسی می‌کند که آیا آرایهٔ ورودی a به ترتیب فرتران (Fortran order) است یا نه. در صورتی که a به ترتیب فرتران باشد، خروجی برابر True خواهد بود؛ در غیر این صورت، خروجی برابر False خواهد بود.

343-. isreal(x):

- x: آرایهٔ ورودی.

تابع isreal بررسی می‌کند که آیا عناصر آرایهٔ ورودی x اعداد حقیقی هستند یا نه. در صورتی که عناصر حاوی اعداد حقیقی باشند، خروجی برابر True خواهد بود؛ در غیر این صورت، خروجی برابر False خواهد بود.

344-. isrealobj(x):

- x: آرایهٔ ورودی.

تابع isrealobj بررسی می‌کند که آیا آرایهٔ ورودی x حاوی اعداد حقیقی یا شیء حقیقی (real object) است یا نه. در صورتی که x حاوی اعداد حقیقی یا شیء حقیقی باشد، خروجی برابر True خواهد بود؛ در غیر این صورت، خروجی برابر False خواهد بود.

345-

. iscomplexobj(x):

- x: آرایهٔ ورودی.

تابع iscomplexobj بررسی می‌کند که آیا آرایهٔ ورودی x حاوی اعداد مختلط یا شیء مختلط (complex object) است یا نه. در صورتی که x حاوی اعداد مختلط یا شیء مختلط باشد، خروجی برابر True خواهد بود؛ در غیر این صورت، خروجی برابر False خواهد بود.

346-. isfortran(a):

- a: آرایهٔ ورودی.

تابع isfortran بررسی می‌کند که آیا آرایهٔ ورودی a به ترتیب فرتران (Fortran order) است یا نه. در صورتی که a به ترتیب فرتران باشد، خروجی برابر True خواهد بود؛ در غیر ا

ین صورت، خروجی برابر False خواهد بود.

347-. isreal(x):

- x: آرایهٔ ورودی.

تابع isreal بررسی می‌کند که آیا عناصر آرایهٔ ورودی x اعداد حقیقی هستند یا نه. در صورتی که عناصر حاوی اعداد حقیقی باشند، خروجی برابر True خواهد بود؛ در غیر این صورت، خروجی برابر False خواهد بود.

348-. isrealobj(x):

- x: آرایهٔ ورودی.

تابع isrealobj بررسی می‌کند که آیا آرایهٔ ورودی x حاوی اعداد حقیقی یا شیء حقیقی (real object) است یا نه. در صورتی که x حاوی اعداد حقیقی یا شیء حقیقی باشد، خروجی برابر True خواهد بود؛ در غیر این صورت، خروجی برابر False خواهد بود.

349-

. iscomplexobj(x):

- x: آرایهٔ ورودی.

تابع iscomplexobj بررسی می‌کند که آیا آرایهٔ ورودی x حاوی اعداد مختلط یا شیء مختلط (complex object) است یا نه. در صورتی که x حاوی اعداد مختلط یا شیء مختلط باشد، خروجی برابر True خواهد بود؛ در غیر این صورت، خروجی برابر False خواهد بود.

350-. isfortran(a):

- a: آرایهٔ ورودی.

تابع isfortran بررسی می‌کند که آیا آرایهٔ ورودی a به ترتیب فرتران (Fortran order) است یا نه. در صورتی که a به ترتیب فرتران باشد، خروجی برابر True خواهد بود؛ در غیر ا

ین صورت، خروجی برابر False خواهد بود.

351-. isreal(x):

- x: آرایهٔ ورودی.

تابع isreal بررسی می‌کند که آیا عناصر آرایهٔ ورودی x اعداد حقیقی هستند یا نه. در صورتی که عناصر حاوی اعداد حقیقی باشند، خروجی برابر True خواهد بود؛ در غیر این صورت، خروجی برابر False خواهد بود.

352-. isrealobj(x):

- x: آرایهٔ ورودی.

تابع isrealobj بررسی می‌کند که آیا آرایهٔ ورودی x حاوی اعداد حقیقی یا شیء حقیقی (real object) است یا نه. در صورتی که x حاوی اعداد حقیقی یا شیء حقیقی باشد، خروجی برابر True خواهد بود؛ در غیر این صورت، خروجی برابر False خواهد بود.

353-

. iscomplexobj(x):

- x: آرایهٔ ورودی.

تابع iscomplexobj بررسی می‌کند که آیا آرایهٔ ورودی x حاوی اعداد مختلط یا شیء مختلط (complex object) است یا نه. در صورتی که x حاوی اعداد مختلط یا شیء مختلط باشد، خروجی برابر True خواهد بود؛ در غیر این صورت، خروجی برابر False خواهد بود.

354-. isfortran(a):

- a: آرایهٔ ورودی.

تابع isfortran بررسی می‌کند که آیا آرایهٔ ورودی a به ترتیب فرتران (Fortran order) است یا نه. در صورتی که a به ترتیب فرتران باشد، خروجی برابر True خواهد بود؛ در غیر این صورت، خروجی برابر False خواهد بود.

355-. isreal(x):

- x: آرایهٔ ورودی.

تابع isreal بررسی می‌کند که آیا عناصر آرایهٔ ورودی x اعداد حقیقی هستند یا نه. در صورتی که عناصر حاوی اعداد حقیقی باشند، خروجی برابر True خواهد بود؛ در غیر این صورت، خروجی برابر False خواهد بود.

356-. isrealobj(x):

- x: آرایهٔ ورودی.

تابع isrealobj بررسی می‌کند که آیا آرایهٔ ورودی x حاوی اعداد حقیقی یا شیء حقیقی (real object) است یا نه. در صورتی که x حاوی اعداد حقیقی یا شیء حقیقی باشد، خروجی برابر True خواهد بود؛ در غیر این صورت، خروجی برابر False خواهد بود.

357-

. array\_equiv(a1, a2): این تابع بررسی می‌کند که آیا دو آرایه a1 و a2 در معنای عناصری با یکدیگر برابر هستند یا خیر. آرگومان‌های این تابع عبارتند از:

- a1: آرایه اول برای مقایسه.

- a2: آرایه دوم برای مقایسه.

358-. greater(x1, x2, /[, out, where, casting, ...]): این تابع دو آرایه x1 و x2 را به صورت عنصر به عنصر مقایسه کرده و آرایه‌ای بولیان برمی‌گرداند که در آن عناصری که مقدار متناظر در x1 بزرگتر از x2 است، با مقدار True و سایر عناصر با مقدار False نشان داده می‌شوند. آرگومان‌های این تابع عبارتند از:

- x1: آرایه اول برای مقایسه.

- x2: آرایه دوم برای مقایسه.

- out (اختیاری): آرایه خروجی که در آن نتیجه مقایسه ذخیره می‌شود.

- where (اختیاری): یک شرط بولی یا آرایه بولی که مشخص می‌کند کدام عناصر باید در مقایسه شرکت کنند.

359-. greater\_equal(x1, x2, /[, out, where, ...]): این تابع دو آرایه x1 و x2 را به صورت عنصر به عنصر مقایسه کرده و آرایه‌ای بولیان برمی‌گرداند که در آن عناصری که مقدار متناظر در x1 بزرگتر یا مساوی با x2 است، با مقدار True و سایر عناصر با مقدار False نشان داده می‌شوند. آرگومان‌های این تابع عبارتند از:

- x1: آرایه اول برای مقایسه.

- x2: آرایه دوم برای مقایسه.

- out (اختیاری):آرایه خروجی که در آن نتیجه مقایسه ذخیره می‌شود.

- where (اختیاری): یک شرط بولی یا آرایه بولی که مشخص می‌کند کدام عناصر باید در مقایسه شرکت کنند.

360-. less(x1, x2, /[, out, where, casting, ...]): این تابع دو آرایه x1 و x2 را به صورت عنصر به عنصر مقایسه کرده و آرایه‌ای بولیان برمی‌گرداند که در آن عناصری که مقدار متناظر در x1 کمتر از x2 است، با مقدار True و سایر عناصر با مقدار False نشان داده می‌شوند. آرگومان‌های این تابع عبارتند از:

- x1: آرایه اول برای مقایسه.

- x2: آرایه دوم برای مقایسه.

- out (اختیاری): آرایه خروجی که در آن نتیجه مقایسه ذخیره می‌شود.

- where (اختیاری): یک شرط بولی یا آرایه بولی که مشخص می‌کند کدام عناصر باید در مقایسه شرکت کنند.

361-. less\_equal(x1, x2, /[, out, where, casting, ...]): این تابع دو آرایه x1 و x2 را به صورت عنصر به عنصر مقایسه کرده و آرایه‌ای بولیان برمی‌گرداند که در آن عناصری که مقدار متناظر در x1 کمتر یا مساوی با x2 است، با مقدار True و سایر عناصر با مقدار False نشان داده می‌شوند. آرگومان‌های این تابع عبارتند از:

- x1: آرایه اول برای مقایسه.

- x2: آرایه دوم برای مقایسه.

- out (اختیاری): آرایه خروجی که در آن نتیجه مقایسه ذخیره می‌شود.

- where (اختیاری یک شرط بولی یا آرایه بولی که مشخص می‌کند کدام عناصر باید در مقایسه شرکت کنند.

362-. equal(x1, x2, /[, out, where, casting, ...]): این تابع دو آرایه x1 و x2 را به صورت عنصر به عنصر مقایسه کرده و آرایه‌ای بولیان برمی‌گرداند که در آن عناصری که مقدار متناظر در x1 با x2 برابر است، با مقدار True و سایر عناصر با مقدار False نشان داده می‌شوند. آرگومان‌های این تابع عبارتند از:

- x1: آرایه اول برای مقایسه.

- x2: آرایه دوم برای مقایسه.

- out (اختیاری): آرایه خروجی که در آن نتیجه مقایسه ذخیره می‌شود.

- where (اختیاری): یک شرط بولی یا آرایه بولی که مشخص می‌کند کدام عناصر باید در مقایسه شرکت کنند.

363-. not\_equal(x1, x2, /[, out, where, casting, ...]): این تابع دو آرایه x1 و x2 را به صورت عنصر به عنصر مقایسه کرده و آرایه‌ای بولیان برمی‌گرداند که در آن عناصری که مقدار متناظر در x1 با x2 برابر نیست، با مقدار True و سایر عناصر با مقدار False نشان داده می‌شوند. آرگومان‌های این تابع عبارتند از:

- x1: آرایه اول برای مقایسه.

- x2: آرایه دوم برای مقایسه.

- out (اختیاری): آرایه خروجی که در آن نتیجه مقایسه ذخیره می‌شود.

- where (اختیاری): یک شرط بولی یا آرایه بولی که مشخص می‌کند کدام عناصر باید در مقایسه شرکت کنند.

364-. ma.copy(self, \*args): این تابع یک کپی از آرایه ماسک شده موجود را ایجاد می‌کند. آرگومان‌های این تابع به طور مشابه با تابع copy در کلاس ndarray در NumPy است.

365-. ma.frombuffer(buffer[, dtype, count, ...]): این تابع یک آرایه ماسک شده را از بوفر (buffer) می‌سازد. آرگومان‌های این تابع عبارتند از:

- buffer: بوفری که برای ساختن آرایه استفاده می‌شود.

- dtype (اختیاری): نوع داده مورد استفاده برای آرایه خروجی.

- count (اختیاری): تعداد عناصر مورد استفاده از بوفر.

366-. ma.fromfunction(function, shape, dtype): این تابع با استفاده از یک تابع و یک شکل، یک آرایه ماسک شده را ایجاد می‌کند. آرگومان‌های این تابع عبارتند از:

- function: تابعی که برای تولید مقادیر آرایه استفاده می‌شود.

- shape: شکل آرایه .

- dtype (اختیاری): نوع داده مورد استفاده برای آرایه خروجی

367-. ma.all(self[, axis, out, keepdims]):

- self: آرایه ماسک شده.

- axis (اختیاری): محور یا محورهایی که بر روی آنها عملیات all انجام می‌شود.

- سایر آرگومان‌ها: شامل out (آرایه خروجی) و keepdims (مشخص می‌کند آیا ابعاد آرایه حاصل باید حفظ شوند یا خیر).

تابع ma.all بررسی می‌کند که آیا تمام عناصر غیرنامعتبر (مقادیر ترو) در آرایه ماسک شده وجود دارند یا خیر. در صورت وجود تمام عناصر غیرنامعتبر، مقدار True برگردانده می‌شود؛ در غیر این صورت مقدار False برگردانده می‌شود. آرگومان axis مشخص می‌کند که عملیات all بر روی کدام محورها انجام شود. سایر آرگومان‌ها اختیاری هستند و به کنترل خروجی و حفظ ابعاد آرایه کمک می‌کنند.

368-. ma.any(self[, axis, out, keepdims]):

- self: آرایه ماسک شده.

- axis (اختیاری): محور یا محورهایی که بر روی آنها عملیات any انجام می‌شود.

- سایر آرگومان‌ها: شامل out (آرایه خروجی) و keepdims (مشخص می‌کند آیا ابعاد آرایه حاصل باید حفظ شوند یا خیر).

تابع ma.any بررسی می‌کند که آیا حداقل یکی از عناصر غیرنامعتبر (مقادیر ترو) در آرایه ماسک شده وجود دارد یا خیر. در صورت وجود حداقل یکی از عناصر غیرنامعتبر، مقدار True برگردانده می‌شود؛ در غیر این صورت مقدار False برگردانده می‌شود. آرگومان axis مشخص می‌کند که عمیلات any بر روی کدام محورها انجام شود. سایر آرگومان‌ها اختیاری هستند و به کنترل خروجی و حفظ ابعاد آرایه کمک می‌کنند.

369-. ma.count(self[, axis, keepdims]):

- self: آرایه ماسک شده.

- axis (اختیاری): محور یا محورهایی که بر روی آنها تعداد عناصر شمرده می‌شود.

- keepdims (اختیاری): مشخص می‌کند آیا ابعاد آرایه حاصل باید حفظ شوند یا خیر.

تابع ma.count تعداد عناصر غیرنامعتبر (مقادیر ترو) در آرایه ماسک شده را محاسبه می‌کند. آرگومان axis مشخص می‌کند که عملیات شمارش بر روی کدام محورها انجام شود. آرگومان keepdims نشان می‌دهد آیا ابعاد آرایه حاصل باید حفظ شوند یا خیر.

370-. ma.count\_masked(arr[, axis]):

- arr: آرایه ماسک شده.

- axis (اختیاری): محور یا محورهایی که بر روی آنها تعداد عناصر غیرنامعتبر شمرده می‌شود.

تابع ma.count\_masked تعداد عناصر غیرنامعتبر (مقادیر ترو) در آرایه ماسک شده را محاسبه می‌کند. آرگومان axis مشخص می‌کند که عملیات شمارش بر روی کدام محورها انجام شود.

371-. ma.getmask(a):

- a: آرایه ماسک شده.

تابع ma.getmask ماسک آرایه را بازمی‌گرداند.

372-. ma.getmaskarray(arr):

- arr: آرایه ماسک شده.

تابع ma.getmaskarray ماسک آرایه را به عنوان یک آرایه بازمی‌گرداند.

373-. ma.getdata(a[, subok]):

- a: آرایه ماسک شده.

- subok (اختیاری): مشخص می‌کند آیا نوع آرایه بازگردانده شده باید با نوع آرایه ورودی یکسان باشد یا خیر.

تابع ma.getdata داده‌های آرایه را بدون ماسک بازمی‌گرداند. اگر آرگومان subok برابر با True باشد، آرایه بازگردانده شده با نوع آرایه ورودی همخوانی دارد.

374-. ma.nonzero(self):

- self: آرایه ماسک شده.

تابع ma.nonzero اندیس‌های عناصر غیرنامعتبر (مقادیر ترو) در آرایه ماسک شده را برمی‌گرداند.

375-. ma.shape(obj):

- obj: آرایه یا شیء قابل تبدیل به آرایه.

تابع ma.shape ابعاد آرایه یا شیء را برمی‌گرداند.

376-. ma.size(obj[, axis]):

- obj: آرایه یا شیء قابل تبدیل به آرایه.

- axis (اختیاری): محور یا محورهایی که بر روی آنها تعداد عناصر محاسبه می‌شود.

تابع ma.size تعداد کل عناصر آرایه یا شیء را محاسبه می‌کند. آرگومان axis مشخص می‌کند که تعداد عناصر بر روی کدام محورها محاسبه شود.

377-. ma.is\_masked(x):

- x: شیء قابل تبدیل به آرایه.

تابع ma.is\_masked بررسی می‌کند که آیا شیء ورودی ماسک شده است یا خیر. اگر شیء ورودی یک آرایه ماسک شده باشد، مقدار True برگردانده می‌شود؛ در غیر این صورت مقدار False برگردانده می‌شود.

378-. ma.is\_mask(m):

- m: شیء قابل تبدیل به آرایه.

تابع ma.is\_mask بررسی می‌کند که آیا شیء ورودی ماسک است یا خیر. اگر شیء ورودی یک ماسک باشد، مقدارtrueبرگردانده می‌شود؛ در غیر این صورت مقدار False برگردانده می‌شود.

379-. ma.isMaskedArray(x):

- x: شیء قابل تبدیل به آرایه.

تابع ma.isMaskedArray بررسی می‌کند که آیا شیء ورودی یک آرایه ماسک شده است یا خیر. اگر شیء ورودی یک آرایه ماسک شده باشد، مقدار True برگردانده می‌شود؛ در غیر این صورت مقدار False برگردانده می‌شود.

380-. ma.isMA(x):

- x: شیء قابل تبدیل به آرایه.

تابع ma.isMA بررسی می‌کند که آیا شیء ورودی یک آرایه ماسک شده است یا خیر. اگر شیء ورودی یک آرایه ماسک شده باشد، مقدار True برگردانده می‌شود؛ در غیر این صورت مقدار False برگردانده می‌شود.

381-. ma.isarray(x):

- x: شیء قابل تبدیل به آرایه.

تابع ma.isarray بررسی می‌کند که آیا شیء ورودی یک آرایه است یا خیر. اگر شیء ورودی یک آرایه باشد، مقدار True برگردانده می‌شود؛ در غیر این صورت مقدار False برگردانده می‌شود.

382-. ma.MaskedArray.all([axis, out, keepdims]):

- axis (اختیاری): محور یا محورهایی که بر روی آنها عملیات all انجام می‌شود.

- سایر آرگومان‌ها: شامل out (آرایه خروجی) و keepdims (مشخص می‌کند آیا ابعاد آرایه حاصل باید حفظ شوند یا خیر).

تابع ma.MaskedArray.all بررسی می‌کند که آیا تمام عناصر غیرنامعتبر (مقادیر ترو) در آرایه ماسک شده وجود دارند یا خیر. در صورت وجود تمام عناصر غیرنامعتبر، مقدار True برگردانده می‌شود؛ در غیر این صورت مقدار False برگردانده می‌شود.

383-. ma.MaskedArray.any([axis, out, keepdims]):

- axis (اختیاری): محور یا محورهایی که بر روی آنها عملیات any انجام می‌شود.

- سایر آرگومان‌ها: شامل out (آرایه خروجی) و keepdims (مشخص می‌کند آیا ابعاد آرایه حاصل باید حفظ شوند یا خیر).

تابع ma.MaskedArray.any بررسی می‌کند که آیا حداقل یک عنصر غیرنامعتبر (مقادیر ترو) در آرایه ماسک شده وجود دارد یا خیر. در صورت وجود حداقل یک عنصر غیرنامعتبر، مقدار True برگردانده می‌شود؛ در غیر این صورت مقدار False برگردانده می‌شود.

384-. ma.MaskedArray.count([axis, keepdims]):

- axis (اختیاری): محور یا محورهایی که بر روی آنها تعداد عناصر غیرنامعتبر شمرده می‌شود.

- keepdims (اختیاری): مشخص می‌کند آیا ابعاد آرایه حاصل باید حفظ شوند یا خیر.

تابع ma.MaskedArray.count تعداد عناصر غیرنامعتبر (مقادیر ترو) در آرایه ماسک شده را محاسبه می‌کند. آرگومان axis مشخص می‌کند که عملیات شمارش بر روی کدام محورها انجام شود. آرگومان keepdims نشان می‌دهد آیا ابعاد آرایه حاصل باید حفظ شوند یا خیر.

385-. ma.MaskedArray.nonzero():

تابع ma.MaskedArray.nonzero اندیس‌های عناصر غیرنامعتبر (مقادیر ترو) در آرایه ماسک شده را برمی‌گرداند.

386-. ma.MaskedArray.data:

ویژگی ma.MaskedArray.data بازگرداندن داده های ماسک شده.

.

387-. ma.MaskedArray.mask:

ویژگی ma.MaskedArray.mask بازگرداندن ماسک آرایه ماسک شده است.

388-

. ma.MaskedArray.recordmask

ویژگی ma.MaskedArray.recordmask بازگرداندن ماسک سابق آرایه ماسک شده است.

389-. ma.ravel(self[, order]):

- order (اختیاری): نحوه ترتیب داده‌ها در آرایه خروجی.

تابع ma.ravel آرایه ماسک شده را به یک آرایه یک بعدی تغییر شکل می‌دهد. آرگومان order نحوه ترتیب داده‌ها در آرایه خروجی را تعیین می‌کند.

390-. ma.reshape(a, new\_shape[, order]):

- a: آرایه ماسک شده اصلی.

- new\_shape: شکل جدید برای آرایه.

تابع ma.reshape آرایه ماسک شده را با شکل جدید مشخص شده تغییر شکل می‌دهد. آرگومان order نحوه ترتیب داده‌ها در آرایه خروجی را تعیین می‌کند.

391-. ma.resize(x, new\_shape):

- x: آرایه ماسک شده اصلی.

- new\_shape: شکل جدید برای آرایه.

تابع ma.resize آرایه ماسک شده را با شکل جدید مشخص شده تغییر شکل می‌دهد. در صورتی که شکل جدید بزرگتر از شکل اصلی باشد، آرایه به صورت تکراری گسترش پیدا می‌کند و در صورتی که شکل جدید کوچکتر از شکل اصلی باشد، برش می‌شود.

392-. ma.MaskedArray.flatten([order]):

- order (اختیاری): نحوه ترتیب داده‌ها در آرایه خروجی.

تابع ma.MaskedArray.flatten آرایه ماسک شده را به یک آرایه یک بعدی تغییر شکل می‌دهد. آرگومان order نحوه ترتیب داده‌ها در آرایه خروجی را تعیین می‌کند.

393-. ma.MaskedArray.ravel([order]):

- order (اختیاری): نحوه ترتیب داده‌ها در آرایه خروجی.

تابع ma.MaskedArray.ravel آرایه ماسک شده را به یک آرایه یک بعدی تغییر شکل می‌دهد. آرگومان order نحوه

ترتیب داده‌ها در آرایه خروجی را تعیین می‌کند.

394. ma.MaskedArray.reshape(\*s, kwargs):

- \*s: ابعاد جدید برای آرایه.

- kwargs: سایر آرگومان‌ها که شامل order نیز می‌شود.

تابع ma.MaskedArray.reshape آرایه ماسک شده را با ابعاد جدید مشخص شده تغییر شکل می‌دهد. می‌توان ابعاد جدید را به صورت واحدی یا به صورت آرایه ابعاد جدید مشخص کرد. سایر آرگومان‌ها، از جمله order که نحوه ترتیب داده‌ها در آرایه خروجی را تعیین می‌کند، نیز قابل استفاده هستند.

395-. ma.MaskedArray.resize(newshape[, refcheck, ...]):

- newshape: شکل جدید برای آرایه.

- refcheck (اختیاری): بررسی انتشار مجدد آرایه و اعمال تغییرات.

تابع ma.MaskedArray.resize آرایه ماسک شده را با شکل جدید مشخص شده تغییر شکل می‌دهد. در صورتی که شکل جدید بزرگتر از شکل اصلی باشد، آرایه به صورت تکراری گسترش پیدا می‌کند و در صورتی که شکل جدید کوچکتر از شکل اصلی باشد، برش می‌شود. آرگومان refcheck مشخص می‌کند که آیا باید بررسی انتشار مجدد آرایه صورت گیرد و تغییرات اعمال شود یا خیر.

396-. ma.swapaxes(self, \*args, ...):

- \*args: اندیس‌های محورهایی که باید جابه‌جا شوند.

تابع ma.swapaxes محورهای مشخص شده در آرایه ماسک شده را جابه‌جا می‌کند. با استفاده از آرگومان \*args، می‌توان چندین زوج اندیس محور را مشخص کرده و جابه‌جایی محورها را انجام داد.

397-. ma.transpose(a[, axes]):

- a: آرایه ماسک شده اصلی.

- axes (اختیاری): الگوی جدید محورها.

تابع ma.transpose ماتریس ترانهاده (تعویض ردیف و ستون) از آرایه ماسک شده را برمی‌گرداند. آرگومان axes نیز الگوی جدید محورها را مشخص می‌کند.

398-. ma.MaskedArray.swapaxes(axis1, axis2):

- axis1: اندیس محور اول.

- axis2: اندیس محور دوم.

تابع ma.MaskedArray.swapaxes محورهای مشخص شده در آرایه ماسک شده را جابه‌جا می‌کند. با استفاده از این تابع، می‌توان دو محور را جابه‌جا کرده و آرایه ماسک شده را به شکل جدیدی با محورهای جابه‌جا شده تغییر شکل داد.

399-. ma.MaskedArray.transpose(\*axes):

- \*axes: الگوی جدید محورها.

تابع ma.MaskedArray.transpose ماتریس ترانهاده (تعویض ردیف و ستون) از آرایه ماسک شده را برمی‌گرداند. با استفاده از آرگومان \*axes، می‌توان الگوی جدید محورها را مشخص کرده و تغییر شکل آرایه را انجام داد.

Changing the number of dimensions:

400-. ma.atleast\_1d(\*args, kwargs):

- \*args: آرگومان‌ها می‌توانند آرایه‌ها یا اشیاء قابل تبدیل به آرایه باشند.

- kwargs: آرگومان‌های نامگذاری شده دیگر.

تابع ma.atleast\_1d آرگومان‌ها را به صورت حداقل یک بعد تبدیل می‌کند. اگر آرگومان‌ها از قبل آرایه باشند، خود آرایه‌ها برگردانده می‌شوند. در غیر این صورت، آرگومان‌ها به صورت آرایه با حداقل یک بعد تبدیل می‌شوند.

401-. ma.atleast\_2d(\*args, kwargs):

- \*args: آرگومان‌ها می‌توانند آرایه‌ها یا اشیاء قابل تبدیل به آرایه باشند.

- kwargs: آرگومان‌های نامگذاری شده دیگر.

تابع ma.atleast\_2d آرگومان‌ها را به صورت حداقل دو بعد تبدیل می‌کند. اگر آرگومان‌ها از قبل دارای حداقل دو بعد هستند، خود آرایه‌ها برگردانده می‌شوند. در غیر این صورت، آرگومان‌ها به صورت آرایه با حداقل دو بعد تبدیل می‌شوند.

402-. ma.atleast\_3d(\*args, kwargs):

- \*args: آرگومان‌ها می‌توانند آرایه‌ها یا اشیاء قابل تبدیل به آرایه باشند.

- kwargs: آرگومان‌های نامگذاری شده دیگر.

تابع ma.atleast\_3d آرگومان‌ها را به صورت حداقل سه بعد تبدیل می‌کند. اگر آرگومان‌ها از قبل دارای حداقل سه بعد هستند، خود آرایه‌ها برگردانده می‌شوند. در غیر این صورت، آرگومان‌ها به صورت آرایه با حداقل سه بعد تبدیل می‌شوند.

403-. ma.expand\_dims(a, axis):

- a: آرایه مورد نظر.

- axis: محوری که باید در آرایه جدید اضافه شود.

تابع ma.expand\_dims آرایه را با اضافه کردن یک محور اضافی در محل مشخص شده تغییر شکل می‌دهد. با استفاده از این تابع، می‌توان تعداد بعد آرایه را افزایش داد.

404-. ma.squeeze(\*args, kwargs):

- \*args: آرگومان‌هایی که باید از آرایه برداشته شوند.

- kwargs: آرگومان‌های نامگذاری شده دیگر.

تابع ma.squeeze ابعاد یک بعدی غیرضروری را از آرایه برداشته و آرایه جدیدی با ابعاد کمتر ایجاد می‌کند. اگر آرگومان‌ها مشخص شوند، تنها ابعاد مشخص شده بررسی و برداشته می‌شوند.

405-. ma.MaskedArray.squeeze([axis]):

- axis (اختیاری): محوری که باید از آرایه برداشته شود.

تابع ma.MaskedArray.squeeze ابعاد یک بعدی غیرضروری را از آرایه ماسک شده برداشته و آرایه جدیدی با ابعاد کمتر ایجاد می‌کند. با مشخص کردن آرگومان axis، تنها ابعاد مشخص شده بررسی و برداشته می‌شوند.

406-. ma.stack(\*args, kwargs):

- \*args: آرگومان‌ها می‌توانند آرایه‌ها یا اشیاء قابل تبدیل به آرایه باشند.

- kwargs: آرگومان‌های نامگذاری شده دیگر.

تابع ma.stack آرایه‌ها را در یک محور جدید پشته‌ای می‌کند. ابعاد آرایه‌ها باید مشابه باشند تا قابل پشته‌بندی با

شند.

407-. ma.column\_stack(\*args, kwargs):

- \*args: آرگومان‌ها می‌توانند آرایه‌ها یا اشیاء قابل تبدیل به آرایه باشند.

- kwargs: آرگومان‌های نامگذاری شده دیگر.

تابع ma.column\_stack آرایه‌ها را در ستون‌های جدید پشته‌ای می‌کند. ابعاد آرایه‌ها باید سازگار باشند تا قابل پشته‌بندی باشند.

408-. ma.concatenate(arrays, axis):

- arrays: آرایه‌ها که باید به هم پیوسته شوند.

- axis: محوری که پیوسته‌سازی بر روی آن انجام می‌شود.

تابع ma.concatenate آرایه‌های موجود را در طول یک محور مشخص پیوسته می‌کند. ابعاد آرایه‌ها باید سازگار باشند تا قابل پیوسته‌سازی باشند.

409-. ma.dstack(\*args, kwargs):

- \*args: آرگومان‌ها می‌توانند آرایه‌ها یا اشیاء قابل تبدیل به آرایه باشند.

- kwargs: آرگومان‌های نامگذاری شده دیگر.

تابع ma.dstack آرایه‌ها را در محور سوم (عمق) پشته‌ای می‌کند. ابعاد آرایه‌ها باید سازگار باشند تا قابل پشته‌بندی باشند.

410-. ma.hstack(\*args, kwargs):

- \*args: آرگومان‌ها می‌توانند آرایه‌ها یا اشیاء قابل تبدیل به آرایه باشند.

- kwargs: آرگومان‌های نامگذاری شده دیگر.

تابع ma.hstack آرایه‌ها را در ستون‌های جدید به صورت افقی پشته‌ای می‌کند. ابعاد آرایه‌ها باید سازگار باشند تا قابل پشته‌بندی باشند.

411-. ma.hsplit(\*args, kwargs):

- \*args: آرگومان ها می‌توانند آرایه‌ها یا اشیاء قابل تبدیل به آرایه باشند.

- kwargs: آرگومان‌های نامگذاری شده دیگر.

تابع ma.hsplit آرایه را در نقاط تقسیم مشخص شده در محور افقی جدا می‌کند و آرایه‌های جدیدی ایجاد می‌کند.

412-. ma.mr\_:

- هیچ آرگومانی ندارد.

ma.mr\_ یک شیء اختصاصی است که امکان ایجاد آرایه با ترکیب ردیف‌ها و ستون‌ها را فراهم می‌کند.

413-. ma.row\_stack(\*args, kwargs):

- \*args: آرگومان‌ها می‌توانند آرایه‌ها یا اشیاء قابل تبدیل به آرایه باشند.

- kwargs: آرگومان‌های نامگذاری شده دیگر.

تابع ma.row\_stack آرایه‌ها را در ردیف‌های جدید عمودی پشته‌ای می‌کند. ابعاد آرایه‌ها باید سازگار باشند تا قابل پشته‌بندی باشند.

415. ma.vstack(\*args, kwargs):

- \*args: آرگومان‌ها می‌توانند آرایه‌ها یا اشیاء قابل تبدیل به آرایه باشند.

- kwargs: آرگومان‌های نامگذاری شده دیگر.

تابع ma.vstack آرایه‌ها را در ردیف‌های جدید عمودی پشته‌ای می‌کند. ابعاد آرایه‌ها باید سازگار باشند تا قابل پشته‌بندی باشند.

Joining arrays:

416. ma.concatenate(arrays, axis):

- arrays: آرایه‌ها که باید به هم پیوسته شوند.

- axis: محوری که پیوسته‌سازی بر روی آن انجام می‌شود.

تابع ma.concatenate آرایه‌های موجود را در طول یک محور مشخص پیوسته می‌کند. ابعاد آرایه‌

ها باید سازگار باشند تا قابل پیوسته‌سازی باشند.

417-. ma.stack(\*args, kwargs):

- \*args: آرگومان‌ها می‌توانند آرایه‌ها یا اشیاء قابل تبدیل به آرایه باشند.

- kwargs: آرگومان‌های نامگذاری شده دیگر.

تابع ma.stack آرایه‌ها را در یک محور جدید پشته‌ای می‌کند. ابعاد آرایه‌ها باید مشابه باشند تا قابل پشته‌بندی باشند.

418-. ma.vstack(\*args, kwargs):

- \*args: آرگومان‌ها می‌توانند آرایه‌ها یا اشیاء قابل تبدیل به آرایه باشند.

- kwargs: آرگومان‌های نامگذاری شده دیگر.

تابع ma.vstack آرایه‌ها را در ردیف‌های جدید عمودی پشته‌ای می‌کند. ابعاد آرایه‌ها باید سازگار باشند تا قابل پشته‌بندی باشند.

419-. ma.hstack(\*args, kwargs):

- \*args: آرگومان‌ها می‌توانند آرایه‌ها یا اشیاء قابل تبدیل به آرایه باشند.

- kwargs: آرگومان‌های نامگذاری شده دیگر.

تابع ma.hstack آرایه‌ها را در ستون‌های جدید به صورت افقی پشته‌ای می‌کند. ابعاد آرایه‌ها باید سازگار باشند تا قابل پشته‌بندی باشند.

420-. ma.dstack(\*args, kwargs):

- \*args: آرگومان‌ها می‌توانند آرایه‌ها یا اشیاء قابل تبدیل به آرایه باشند.

- kwargs: آرگومان‌های نامگذاری شده دیگر.

تابع ma.dstack آرایه‌ها را در محور سوم (عمق) پشته‌ای می‌کند. ابعاد آرایه‌ها باید سازگار باشند تا قابل پشته‌بندی باشند.

421-. ma.column\_stack(\*args, kwargs):

- \*args: آرگومان‌ها می‌توانند آرایه‌ها یا اشیاء قابل تبدیل به آرایه باشند.

- kwargs: آرگومان‌های نامگذاری شده دیگر.

تابع ma.column\_stack آرایه‌ها را در ستون‌های جدید پشته‌ای می‌کند. ابعاد آرایه‌ها باید سازگار باشند تا قابل پشته‌بندی باشند.

422-. ma.append(a, b, axis=None):

- a: آرایه اول.

- b: آرایه دوم.

- axis: محوری که آرایه دوم به آرایه اول پیوسته می‌شود.

تابع ma.append آرایه دوم را به آرایه اول پیوسته می‌کند. اگر محور مشخص نشده باشد، آرایه‌ها به صورت flatten شده و پیوسته می‌شوند. در غیر اینصورت، آرایه دوم در محور مشخص شده پیوسته می‌شود.

423-. ma.make\_mask(m, copy=True, shrink=True, dtype=bool):

- m: آرایه ورودی برای ساختن ماسک.

- copy (اختیاری): مشخص می‌کند آیا ماسک جدید باید کپی شود یا خیر. مقدار پیش‌فرض True است.

- shrink (اختیاری): مشخص می‌کند آیا ابعاد ماسک جدید باید بدون بعد‌های اضافی باشد. مقدار پیش‌فرض True است.

- dtype (اختیاری): نوع داده ماسک جدید.

تابع ma.make\_mask یک ماسک جدید با استفاده از آرایه ورودی می‌سازد.

424-. ma.make\_mask\_none(newshape, dtype=bool):

- newshape: شکل جدید ماسک.

- dtype (اختیاری): نوع داده ماسک جدید.

تابع ma.make\_mask\_none یک ماسک خالی جدید با ابعاد و نوع داده مشخص شده می‌سازد.

425-. ma.mask\_or(m1, m2, copy=True, shrink=True):

- m1: ماسک اول.

- m2: ماسک دوم.

- copy (اختیاری): مشخص می‌کند آیا ماسک جدید باید کپی شود یا خیر. مقدار پیش‌فرض True است.

- shrink (اختیاری): مشخص می‌کند آیا ابعاد ماسک جدید باید بدون بعد‌های اضافی باشد. مقدار پیش‌فرض True است.

تابع ma.mask\_or ماسک جدیدی را براساس ماسک‌های ورودی می‌سازد، که بیانگر مکان‌هایی است که در یکی از ماسک‌ها مقدار ماسک شده است.

426-. ma.make\_mask\_descr(ndtype):

- ndtype: نوع داده ماسک.

تابع ma.make\_mask\_descr یک توصیف‌گر ماسک براساس نوع داده ورودی می‌سازد.

427-. ma.getmask(a):

- a: آرایه ورودی.

تابع ma.getmask ماسک مرتبط با آرایه ورودی را بازمی‌گرداند.

428-. ma.getmaskarray(arr):

- arr: آرایه ورودی.

تابع ma.getmaskarray آرایه ماسک مرتبط با آرایه ورودی را بازمی‌گرداند.

429-. ma.masked\_array.mask:

ویژگی mask در آرایه ماسک شده را بازمی‌گرداند.

430-. ma.ndenumerate(a, compressed=False):

- a: آرایه ورودی.

- compressed (اختیاری): مشخص می‌کند آیا ایندکس‌ها باید فشرده شوند یا خیر. مقدار پیش‌فرض False است.

تابع ma.ndenumerate تمام اندیس‌ها و مقادیر غیر ماسک آرایه ورودی را بازمی‌گرداند.

431-. ma.flatnotmasked\_contiguous(a):

- a: آرایه ورودی.

تابع ma.flatnotmasked\_contiguous بخش‌های پیوسته از مقادیر غیر ماسک آرایه ورودی را در قالب یک تک بعد آرایه بازمی‌گرداند.

432-. ma.flatnotmasked\_edges(a):

- a: آرایه ورودی.

تابع ma.flatnotmasked\_edges اندیس‌های آغازین و پایانی بخش‌های پیوسته از مقادیر غیر ماسک آرایه ورودی را در قالب یک تک بعد آرایه بازمی‌گرداند.

433-. ma.notmasked\_contiguous(a, axis=None):

- a: آرایه ورودی.

- axis (اختیاری): محور مورد نظر.

تابع ma.notmasked\_contiguous بخش‌های پیوسته از مقادیر غیر ماسک آرایه ورودی را در محور مشخص شده بازمی‌گرداند.

434-. ma.notmasked\_edges(a, axis=None):

- a: آرایه ورودی.

- axis (اختیاری): محور مورد نظر.

تابع ma.notmasked\_edges اندیس‌های آغازین و پایانی بخش‌های پیوسته از مقادیر غیر ماسک آرایه ورودی را در محور مشخص شده بازمی‌گرداند.

435-. ma.clump\_masked(a):

- a: آرایه ورودی.

تابع ma.clump\_masked بخش‌های پیوسته از مقادیر ماسک شده آرایه ورودی را بازمی‌گرداند.

436-. ma.clump\_unmasked(a):

- a: آرایه ورودی.

تابع ma.clump\_unmasked بخش‌های پیوسته از مقادیر غیر ماسک آرایه ورودی را بازمی‌گرداند.

437-. ma.mask\_cols(a, axis=None):

- a: آرایه ورودی.

- axis (اختیاری): محور مورد نظر.

تابع ma.mask\_cols ستون‌های آرایه ورودی را ماسک می‌کند.

438-. ma.mask\_or(m1, m2, copy=True, shrink=True):

- m1: ماسک اول.

- m2: ماسک دوم.

- copy (اختیاری): مشخص می‌کند آیا ماسک جدید باید کپی شود یا خیر. مقدار پیش‌فرض True است.

- shrink (اختیاری): مشخص می‌کند آیا ابعاد ماسک جدید باید بدون بعد‌های اضافی باشد. مقدار پیش‌فرض True است.

تابع ma.mask\_or ماسک جدیدی را براساس ماسک‌های ورودی می‌سازد، که بیانگر مکان‌هایی است که در یکی از ماسک‌ها مقدار ماسک شده است.

439-. ma.mask\_rowcols(a, axis=None):

- a: آرایه ورودی.

- axis (اختیاری): محور مورد نظر.

تابع ma.mask\_rowcols سطرها و ستون‌های آرایه ورودی را ماسک می‌کند.

440-. ma.mask\_rows(a, axis=None):

- a: آرایه ورودی.

- axis (اختیاری): محور مورد نظر.

تابع ma.mask\_rows سطرهای آرایه ورودی را ماسک می‌کند.

441-. ma.harden\_mask(self):

- self: آرایه ورودی.

تابع ma.harden\_mask ماسک آرایه ورودی را قابل تغییر نمی‌کند.

442-. ma.soften\_mask(self):

- self: آرایه ورودی.

تابع ma.soften\_mask ماسک آرایه ورودی را قابل تغییر می‌کند.

443-. ma.MaskedArray.soften\_mask():

- self: آرایه ورودی.

تابع ma.MaskedArray.soften\_mask ماسک آرایه ورودی را قابل تغییر می‌کند.

444-. ma.MaskedArray.shrink\_mask():

- self: آرایه ورودی.

تابع ma.MaskedArray.shrink\_mask ابعاد ماسک آرایه ورودی را کاهش می‌دهد.

445-. ma.MaskedArray.unshare\_mask():

- self: آرایه ورودی.

تابع ma.MaskedArray.unshare\_mask ماسک آرایه ورودی را با سایر آرایه‌های ماسک مشترک نمی‌کند.

446-. ma.asarray(a, dtype=None, order=None):

- a: آرایه ورودی.

- dtype (اختیاری): نوع داده مطلوب برای آرایه خروجی.

- order (اختیاری): نحوه ذخیره‌سازی داده در حافظه.

تابع ma.asarray آرایه ورودی را به یک آرایه ماسک شده تبدیل می‌کند.

447-. ma.asanyarray(a, dtype=None):

- a: آرایه ورودی.

- dtype (اختیاری): نوع داده مطلوب برای آرایه خروجی.

تابع ma.asanyarray آرایه ورودی را به یک آرایه ماسک شده تبدیل می‌کند.

448-. ma.fix\_invalid(a, mask=None, copy=True, fill\_value=None):

- a: آرایه ورودی.

- mask (اختیاری): ماسک مربوط به آرایه ورودی.

- copy (اختیاری): مشخص می‌کند آیا آرایه خروجی باید کپی شود یا خیر. مقدار پیش‌فرض True است.

- fill\_value (اختیاری): مقداری که به جای مقادیر نامعتبر قرار می‌گیرد.

تابع ma.fix\_invalid مقادیر نامعتبر در آرایه ورودی را با مقدار دلخواه جایگزین می‌کند و یک

آرایه ماسک شده برمی‌گرداند.

449-. ma.masked\_equal(x, value, copy=True):

- x: آرایه ورودی.

- value: مقدار مورد نظر برای مقایسه.

- copy (اختیاری): مشخص می‌کند آیا آرایه خروجی باید کپی شود یا خیر. مقدار پیش‌فرض True است.

تابع ma.masked\_equal مقادیر مساوی با یک مقدار مشخص را ماسک می‌کند.

450-. ma.masked\_greater(x, value, copy=True):

- x: آرایه ورودی.

- value: مقدار مورد نظر برای مقایسه.

- copy (اختیاری): مشخص می‌کند آیا آرایه خروجی باید کپی شود یا خیر. مقدار پیش‌فرض True است.

تابع ma.masked\_greater مقادیر بزرگتر از یک مقدار مشخص را ماسک می‌کند.

451-. ma.masked\_greater\_equal(x, value, copy=True):

- x: آرایه ورودی.

- value: مقدار مورد نظر برای مقایسه.

- copy (اختیاری): مشخص می‌کند آیا آرایه خروجی باید کپی شود یا خیر. مقدار پیش‌فرض True است.

تابع ma.masked\_greater\_equal مقادیر بزرگتر یا مساوی با یک مقدار مشخص را ماسک می‌کند.

452-. ma.masked\_inside(x, v1, v2, copy=True):

- x: آرایه ورودی.

- v1: مقدار شروع محدوده.

- v2: مقدار پایان محدوده.

- copy (اختیاری): مشخص می‌کند آیا آرایه خروجی باید کپی شود یا خیر. مقدار پیش‌فرض True است.

تابع ma.masked\_inside مقادیر بین دو مقدار مشخص را ماسک می‌کند.

453-. ma.masked\_invalid(a, copy=True):

- a: آرایه ورودی.

- copy (اختیاری): مشخص می‌کند آیا آرایه خروجی باید کپی شود یا خیرمقدار پیش‌فرض True است.

تابع ma.masked\_invalid مقادیر نامعتبر (NaN و infinity) را ماسک می‌کند.

454-. ma.masked\_less(x, value, copy=True):

- x: آرایه ورودی.

- value: مقدار مورد نظر برای مقایسه.

- copy (اختیاری): مشخص می‌کند آیا آرایه خروجی باید کپی شود یا خیر. مقدار پیش‌فرض True است.

تابع ma.masked\_less مقادیر کوچکتر از یک مقدار مشخص را ماسک می‌کند.

455-. ma.masked\_less\_equal(x, value, copy=True):

- x: آرایه ورودی.

- value: مقدار مورد نظر برای مقایسه.

- copy (اختیاری): مشخص می‌کند آیا آرایه خروجی باید کپی شود یا خیر. مقدار پیش‌فرض True است.

تابع ma.masked\_less\_equal مقادیر کوچکتر یا مساوی با یک مقدار مشخص را ماسک می‌کند.

456-. ma.masked\_not\_equal(x, value, copy=True):

- x: آرایه ورودی.

- value: مقدار مورد نظر برای مقایسه.

- copy (اختیاری): مشخص می‌کند آیا آرایه خروجی باید کپی شود یا خیر. مقدار پیش‌فرض True است.

تابع ma.masked\_not\_equal مقادیر نامساوی با یک مقدار مشخص را ماسک می‌کند.

457-. ma.masked\_object(x, value, copy=True, shrink=True):

- x: آرایه ورودی.

- value: مقدار مورد نظر برای مقایسه.

- copy (اختیاری): مشخص می‌کند آیا آرایه خروجی باید کپی شود یا خیر. مقدار پیش‌فرض True است.

- shrink (اختیاری): مشخص می‌کند آیا ابعاد ماسک جدید باید بدون بعد‌های اضافی باشد. مقدار پیش‌فرض True است.

تابع ma.masked\_object مقادیر مشابه با یک مقدار مشخص را ماسک می‌کند.

458-. ma.masked\_outside(x, v1, v2, copy=True):

- x: آرایه ورودی.

- v1: مقدار شروع محدوده.

- v2: مقدار پایان محدوده.

- copy (اختیاری): مشخص می‌کند آیا آرایه خروجی باید کپی شود یا خیر. مقدار پیش‌فرض True است.

459-. ma.compress\_cols(a):

- a: آرایه ورودی.

تابع ma.compress\_cols ستون‌های آرایه ورودی را متمرکز می‌کند و یک آرایه جدید با تمام ستون‌های فشرده شده ایجاد می‌کند.

460-. ma.compress\_rowcols(x, axis=None):

- x: آرایه ورودی.

- axis (اختیاری): محور برای فشرده سازی. اگر مقدار دهی نشود، تمام ابعاد غیرماسک فشرده می‌شوند.

تابع ma.compress\_rowcols سطرها و ستون‌های آرایه ورودی را متمرکز می‌کند و یک آرایه جدید با تمام سطرها و ستون‌های فشرده شده ایجاد می‌کند.

461-. ma.compress\_rows(a):

- a: آرایه ورودی.

تابع ma.compress\_rows سطرهای آرایه ورودی را متمرکز می‌کند و یک آرایه جدید با تمام سطرهای فشرده شده ایجاد می‌کند.

462-. ma.compressed(x):

- x: آرایه ورودی.

تابع ma.compressed آرایه ورودی را فشرده کرده و یک آرایه جدید با مقادیر غیرماسک ایجاد می‌کند.

463-. ma.filled(a, fill\_value=None):

- a: آرایه ورودی.

- fill\_value (اختیاری): مقداری که برای جایگزینی مقادیر ماسک استفاده می‌شود. اگر مقدار دهی نشود، مقدار پر کردن پیش‌فرض آرایه ورودی است.

تابع ma.filled آرایه ورودی را با مقادیر ماسک جایگزین می‌کند و یک آرایه جدید بدون مقادیر ماسک ایجاد می‌کند.

464-. ma.MaskedArray.compressed():

تابع ma.MaskedArray.compressed آرایه را فشرده کرده و یک نسخه فشرده شده از آن را برمی‌گرداند.

465-. ma.MaskedArray.tofile(fid, sep='', format='%s'):

- fid: شیء فایل برای نوشتن داده‌های آرایه.

- sep (اختیاری): جداکننده بین عناصر آرایه در فایل.

- format (اختیاری): فرمت استفاده شده برای تبدیل هر عنصر به رشته.

تابع ma.MaskedArray.tofile آرایه را به فایل نوشته و ذخیره می‌کند.

466-. ma.MaskedArray.tolist(fill\_value=None):

- fill\_value (اختیاری): مقداری که برای جایگزینی مقادیر ماسک استفاده می‌شود. اگر مقدار دهی نشود، مقدار پر کردن پیش‌فرض آرایه است.

تابع ma.MaskedArray.tolist آرایه را به یک لیست تبدیل می‌کند و مقادیر ماسک را با مقدار پر کردن مشخص شده (یا مقدار پر کردن پیش‌فرض) جایگزین می‌کند.

467-. ma.MaskedArray.torecords():

تابع ma.MaskedArray.torecords آرایه را به رکوردهای مستقل تبدیل می‌کند و برای هر رکورد یک نسخه فشرده شده از آرایه را برمی‌گرداند.

468-. ma.MaskedArray.tobytes(fill\_value=None, order=None):

- fill\_value (اختیاری): مقداری که برای جایگزینی مقادیر ماسک استفاده می‌شود. اگر مقدار دهی نشود، مقدار پر کردن پیش‌فرض آرایه است.

- order (اختیاری): ترتیب ذخیره‌سازی برای داده‌های آرایه.

تابع ma.MaskedArray.tobytes آرایه را به رشته‌ای از بایت‌ها تبدیل می‌کند و مقادیر ماسک را با مقدار پر کردن مشخص شده (یا مقدار پر کردن پیش‌فرض) جایگزین می‌کند.

469-. ma.common\_fill\_value(a, b):

- a: آرایه اول

- b: آرایه دوم.

تابع ma.common\_fill\_value مقدار پر کردن مشترک بین دو آرایه را برمی‌گرداند. این مقدار مشخص می‌کند که برای پر کردن مقادیر ماسک در هر دو آرایه از چه مقداری استفاده شود.

470-. ma.default\_fill\_value(obj):

- obj: شیء مرتبط با کلاس ماسک شده.

تابع ma.default\_fill\_value مقدار پر کردن پیش‌فرض برای مقادیر ماسک در آرایه‌هایی با کلاس ماسک شده را برمی‌گرداند.

471-. ma.maximum\_fill\_value(obj):

- obj: شیء مرتبط با کلاس ماسک شده.

تابع ma.maximum\_fill\_value مقدار پر کردن حداکثر برای مقادیر ماسک در آرایه‌هایی با کلاس ماسک شده را برمی‌گرداند.

472-. ma.minimum\_fill\_value(obj):

- obj: شیء مرتبط با کلاس ماسک شده.

تابع ma.minimum\_fill\_value مقدار پر کردن حداقل برای مقادیر ماسک در آرایه‌هایی با کلاس ماسک شده را برمی‌گرداند.

473-. ma.set\_fill\_value(a, fill\_value=None):

- a: آرایه ورودی.

- fill\_value (اختیاری): مقداری که به عنوان مقدار پر کردن استفاده می‌شود. اگر مقدار دهی نشود، مقدار پر کردن پیش‌فرض آرایه است.

تابع ma.set\_fill\_value مقدار پر کردن آرایه ورودی را تنظیم می‌کند.

474-. ma.MaskedArray.get\_fill\_value():

تابع ma.MaskedArray.get\_fill\_value مقدار پر کردن مورد استفاده در آرایه را برمی‌گرداند.

475-. ma.MaskedArray.set\_fill\_value(value):

- value: مقداری که به عنوان مقدار پر کردن استفاده می‌شود.

تابع ma.MaskedArray.set\_fill\_value مقدار پر کردن آرایه را تنظیم می‌کند.

476-. ma.MaskedArray.fill\_value:

ویژگی ma.MaskedArray.fill\_value مقدار پر کردن آرایه را نشان می‌دهد.

477-. ma.anom(self, axis=None, dtype=None):

- self: آرایه صحیح)

Axis: (اختیاری): محور یا محورهایی که بر روی آنها محاسبات انجام می‌شود. اگر مقدار دهی نشود، محاسبات بر روی تمام عناصر آرایه انجام می‌شود.

- dtype (اختیاری): نوع داده محاسبه شده.

تابع ma.anom میانگین هر ستون یا سطر را از مقدارهای ماسک شده محاسبه کرده و تفاوت هر عنصر با میانگین محاسبه شده را برمی‌گرداند.

478-. ma.anomalies(self, axis=None, dtype=None):

- self: آرایه ماسک شده.

- axis (اختیاری): محور یا محورهایی که بر روی آنها محاسبات انجام می‌شود. اگر مقدار دهی نشود، محاسبات بر روی تمام عناصر آرایه انجام می‌شود.

- dtype (اختیاری): نوع داده محاسبه شده.

تابع ma.anomalies میانگین هر ستون یا سطر را از مقدارهای ماسک شده محاسبه کرده و تفاوت هر عنصر با میانگین محاسبه شده را برمی‌گرداند. نتیجه برگردانده شده همچنین شامل مقادیر ماسک باقی‌مانده است.

479-. ma.average(a, axis=None, weights=None, returned=False, keepdims=False):

- a: آرایه مورد استفاده برای محاسبه میانگین.

- axis (اختیاری): محور یا محورهایی که بر روی آنها محاسبات انجام می‌شود. اگر مقدار دهی نشود، محاسبات بر روی تمام عناصر آرایه انجام می‌شود.

- weights (اختیاری): آرایه وزن‌ها که در محاسبه میانگین استفاده می‌شود. باید ابعاد آن با آرایه a سازگار باشد.

- returned (اختیاری): در صورت True برگرداندن جفت مرتب شامل میانگین

و مجموع وزن‌ها.

- keepdims (اختیاری): در صورت True حفظ ابعاد محورها در نتیجه برگردانده شده.

تابع ma.average میانگین وزن‌دار عناصر آرایه را محاسبه می‌کند. در صورت استفاده از آرایه وزن‌ها، عناصر ماسک شده در محاسبه میانگین حذف می‌شوند.

480-. ma.conjugate(x, /, out=None, where=True, casting='same\_kind', order='K', dtype=None, subok=True):

- x: آرایه ورودی.

- out (اختیاری): آرایه خروجی.

- where (اختیاری): شرط برای انجام عملیات.

- casting (اختیاری): روش تبدیل نوع داده.

- order (اختیاری): ترتیب ارتباط حافظه ورودی و خروجی.

- dtype (اختیاری): نوع داده خروجی.

- subok (اختیاری): در صورت True حاصل عملیات یک نمونه از کلاس ماسک شده است.

تابع ma.conjugate مقدار مختلط هر عنصر آرایه را محاسبه کرده و خروجی را برمی‌گرداند.

481-. ma.corrcoef(x, y=None, rowvar=True, bias=<no value>, ddof=<no value>):

- x: آرایه ورودی اول.

- y (اختیاری): آرایه ورودی دوم. اگر داده نشود، آرایه x به عنوان ورودی دوم در نظر گرفته می‌شود.

- rowvar (اختیاری): در صورت True هر ردیف از آرایه ورودی به عنوان یک متغیر در نظر گرفته می‌شود. در صورت False هر ستون به عنوان یک متغیر در نظر گرفته می‌شود.

- bias (اختیاری): اگر داده نشود، مقدار پیش‌فرض آن استفاده می‌شود.

- ddof (اختیاری): اگر داده نشود، مقدار پیش‌فرض آن استفاده می‌شود.

تابع ma.corrcoef ضریب همبستگی بین دو آرایه را محاسبه می‌کند. آرایه‌ها می‌توانند شامل مقادیر ماسک باشند.

482-. ma.count(a, axis=None):

- a: آرایه ورودی.

- axis (اختیاری): محور یا محورهایی که بر روی آنها تعداد محاسبات انجام می‌شود. اگر مقدار دهی نشود، محاسبات بر روی تمام عناصر آرایه انجام می‌شود.

تابع ma.count تعداد عناصر غیرماسک در آرایه را محاسبه می‌کند.

483-. ma.cov(x, y=None, rowvar=True, bias=<no value>, ddof=<no value>):

- x: آرایه ورودی اول.

- y (اختیاری): آرایه ورودی دوم. اگر داده نشود، آرایه x به عنوان ورودی دوم در نظر گرفته می‌شود.

- rowvar (اختیاری): در صورت True هر ردیف از آرایه ورودی به عنوان یک متغیر در نظر گرفته می‌شود. در صورت False هر ستون به عنو

ان یک متغیر در نظر گرفته می‌شود.

- bias (اختیاری): اگر داده نشود، مقدار پیش‌فرض آن استفاده می‌شود.

- ddof (اختیاری): اگر داده نشود، مقدار پیش‌فرض آن استفاده می‌شود.

تابع ma.cov ماتریس کوواریانس بین دو آرایه را محاسبه می‌کند. آرایه‌ها می‌توانند شامل مقادیر ماسک باشند.

484-. ma.cumprod(a, axis=None, dtype=None, out=None):

- a: آرایه ورودی.

- axis (اختیاری): محوری که بر روی آن محاسبات انجام می‌شود. اگر مقدار دهی نشود، محاسبات در کل آرایه صورت می‌گیرد.

- dtype (اختیاری): نوع داده خروجی.

- out (اختیاری): آرایه خروجی.

تابع ma.cumprod محاسبه نتیجه‌ی تجمعی حاصلضرب عناصر آرایه را بر روی محور مشخص شده انجام می‌دهد.

485-. ma.cumsum(a, axis=None, dtype=None, out=None):

- a: آرایه ورودی.

- axis (اختیاری): محوری که بر روی آن محاسبات انجام می‌شود. اگر مقدار دهی نشود، محاسبات در کل آرایه صورت می‌گیرد.

- dtype (اختیاری): نوع داده خروجی.

- out (اختیاری): آرایه خروجی.

تابع ma.cumsum محاسبه نتیجه‌ی تجمعی مجموع عناصر آرایه را بر روی محور مشخص شده انجام می‌دهد.

486-. ma.diagonal(a, offset=0, axis1=0, axis2=1):

- a: آرایه ورودی.

- offset (اختیاری): آفست محورها.

- axis1 (اختیاری): محور اول برای محاسبه قطر آرایه.

- axis2 (اختیاری): محور دوم برای

محاسبه قطر آرایه. تابع ma.diagonal عناصر قطری آرایه را برمی‌گرداند. با استفاده از پارامترهای offset، axis1 و axis2 می‌توانید قطرهای دیگری را در آرایه محاسبه کنید.

487-. ma.dot(a, b, out=None):

- a: آرایه اول برای عملیات ضرب نقطه‌ای.

- b: آرایه دوم برای عملیات ضرب نقطه‌ای.

- out (اختیاری): آرایه خروجی.

تابع ma.dot ضرب نقطه‌ای دو آرایه را محاسبه می‌کند.

488-. ma.fill\_diagonal(a, val, wrap=False):

- a: آرایه ورودی.

- val: مقداری که در قطر آرایه قرار می‌گیرد.

- wrap (اختیاری): اگر True است، آرایه قطر را به یک نقطه محدود می‌کند. در غیر این صورت، قطر را به طور کامل پر می‌کند.

تابع ma.fill\_diagonal مقادیر داده شده را در قطر آرایه ورودی قرار می‌دهد.

489-. ma.inner(a, b):

- a: آرایه اول برای عملیات ضرب داخلی.

- b: آرایه دوم برای عملیات ضرب داخلی.

تابع ma.inner ضرب داخلی دو آرایه را محاسبه می‌کند.

490-. ma.invert(a):

- a: آرایه ورودی.

تابع ma.invert برعکس آرایه ورودی را محاسبه می‌کند.

491-. ma.outer(a, b):

- a: آرایه اول برای عملیات ضرب خارجی.

- b: آرایه دوم برای عملیات ضرب خارجی.

تابع ma.outer ضرب خارجی دو آرایه را محاسبه می‌کند.

492-. ma.linalg.pinv(a, rcond=1e-15, hermitian=False):

- a: آرایه ورودی.

- rcond (اختیاری): آستانه‌

ی مورد استفاده برای تعیین صفر بودن مقادیر تک‌مقداری.

- hermitian (اختیاری): در صورت True، آرایه ورودی فرض می‌شود وارون‌پذیر است.

تابع ma.linalg.pinv ماتریس وارون‌پذیر معکوس معتبر را برای آرایه ورودی محاسبه می‌کند.

493-. ma.linalg.matrix\_power(M, n):

- M: آرایه ورودی.

- n: توان.

تابع ma.linalg.matrix\_power توان ماتریس ورودی را محاسبه می‌کند.

494-. ma.linalg.tensorsolve(a, b, axes=None):

- a: آرایه نمایش‌دهنده‌ی مجموعهٔ عملگرها.

- b: آرایه نمایش‌دهنده‌ی یک یا چند بردار.

- axes (اختیاری): محورهایی که در آنها تنظیم شده است.

تابع ma.linalg.tensorsolve به طور عمومی رابطهٔ einsum(a, [\*,a\_axes],[\*,b\_axes],[\*,out\_axes], dtype) را محاسبه می‌کند، در حالی که \* نشان‌دهندهٔ همه محورها است.

495-. ma.linalg.det(a):

- a: آرایه ورودی.

تابع ma.linalg.det مقدار مقطع ماتریس ورودی را محاسبه می‌کند.

496-. ma.linalg.eigvals(a):

- a: آرایه ورودی.

تابع ma.linalg.eigvals مقادیر ویژهٔ ماتریس ورودی را محاسبه می‌کند.

497-. ma.linalg.eigvalsh(a, UPLO='L'):

- a: آرایه ورودی.

- UPLO (اختیاری): طرفین ماتریس ورودی که باید در آنها محاسبات صورت گیرد.

تابع ma.linalg.eigvalsh مقادیر ویژهٔ ماتریس هرمیتی ورودی را محاسبه می‌کند.

498-. ma.linalg.eig(a):

- a: آرایه ورودی.

تابع ma.linalg.eig بردارهای ویژهٔ ماتریس ورودی را محاسبه می‌کند

.

499-. ma.linalg.eigh(a, UPLO='L'):

- a: آرایه ورودی.

- UPLO (اختیاری): طرفین ماتریس ورودی که باید در آنها محاسبات صورت گیرد.

تابع ma.linalg.eigh بردارهای ویژهٔ ماتریس هرمیتی ورودی را محاسبه می‌کند.

500-. ma.linalg.norm(x, ord=None, axis=None, keepdims=False):

- x: آرایه ورودی.

- ord (اختیاری): نرم مورد نظر برای محاسبه. اگر مقدار دهی نشود، نرم 2 (نرم اقلیدسی) استفاده می‌شود.

- axis (اختیاری): محورها برای محاسبهٔ نرم.

- keepdims (اختیاری): در صورت True ابعاد آرایه حاصل نرم حفظ می‌شود.

تابع ma.linalg.norm مقدار نرم آرایه ورودی را محاسبه می‌کند.

501-. ma.linalg.matrix\_rank(M, tol=None):

- M: آرایه ورودی.

- tol (اختیاری): مقدار آستانه برای محاسبهٔ رتبهٔ ماتریس.

تابع ma.linalg.matrix\_rank رتبهٔ ماتریس ورودی را محاسبه می‌کند.

502-. ma.diff(\*args, kwargs):

تابع ma.diff تفاضل بین عناصر متوالی آرایهٔ ورودی را محاسبه می‌کند. می‌توانید آرگومان‌های \*args را برای ارائهٔ آرایه‌های متوالی به عنوان ورودی استفاده کنید. همچنین، می‌توانید آرگومان‌های کلیدی kwargs را برای تنظیمات اضافی مانند axis، استفاده کنید.

503-. ma.MaskedArray.argmax([axis, fill\_value, ...]):

- axis (اختیاری): محوری که بر روی آن حداکثر مقدار جستجو می‌شود.

- fill\_value (اختیاری): مقدار پر کننده برای عناصر نامعتبر.

تابع ma.MaskedArray.argmax اندیس عنصر با بزرگترین مقدار در آرایهٔ ماسک شده را برمی‌گرداند.

504-. ma.MaskedArray.argmin([axis, fill\_value, ...]):

- axis (اختیاری): محوری که بر روی آن حداقل مقدار جستجو می‌شود.

- fill\_value (اختیاری): مقدار پر کننده برای عناصر نامعتبر.

تابع ma.MaskedArray.argmin اندیس عنصر با کوچکترین مقدار در آرایهٔ ماسک شده را برمی‌گرداند.

505. ma.MaskedArray.max([axis, out, fill\_value, ...]):

- axis (اختیاری): محوری که بر روی آن حداکثر مقدار جستجو می‌شود.

- out (اختیاری): آرایهٔ خروجی.

- fill\_value (اختیاری): مقدار پر کننده برای عناصر نامعتبر.

تابع ma.MaskedArray.max مقدار بزرگترین عنصر در آرایهٔ ماسک شده را برمی‌گرداند.

506. ma.MaskedArray.min([axis, out, fill\_value, ...]):

- axis (اختیاری): محوری که بر روی آن حداقل مقدار جستجو می‌شود.

- out (اختیاری): آرایهٔ خروجی.

- fill\_value (اختیاری): مقدار پر کننده برای عناصر نامعتبر.

تابع ma.MaskedArray.min مقدار کوچکترین عنصر در آرایهٔ ماسک شده را برمی‌گرداند.

507. ma.MaskedArray.ptp([axis, out, fill\_value, ...]):

- axis (اختیاری): محوری که بر روی آن محدودهٔ دامنه جستجو می‌شود.

- out (اختیاری): آرایهٔ خروجی.

- fill\_value (اختیاری): مقدار پر کننده برای عناصر نامعتبر.

تابع ma.MaskedArray.ptp محدودهٔ دامنهٔ جستجو در آرایهٔ ماسک شده را برمی‌گرداند (با محدودهٔ بزرگترین و کوچکترین عناصر).

508. ma.argsort(a[, axis, kind, order, endwith, ...]):

تابع ma.argsort اندیس‌های مرتب‌سازی شدهٔ مقادیر آرایهٔ ورودی را برمی‌گرداند. می‌توانید آرگومان‌های کلیدی مانند axis، kind، order، endwith و fill\_value را برای تنظیمات مرتب‌سازی استفاده کنید.

509-. ma.sort(a[, axis, kind, order, endwith, ...]):

- axis (اختیاری): محوری که بر روی آن مقادیر آرایه مرتب شده می‌شوند.

- kind (اختیاری): نوع مرتب‌سازی مورد استفاده. از مقادیر ممکن مانند 'quicksort'، 'mergesort'، 'heapsort' استفاده کنید.

- order (اختیاری): نام ستون‌ها برای استفاده در مرتب‌سازی ساختار داده‌های رکوردی.

- endwith (اختیاری): اندیس انتهایی برای مرتب‌سازی.

- fill\_value (اختیاری): مقدار پر کننده برای عناصر نامعتبر.

تابع ma.sort آرایهٔ ورودی را مرتب می‌کند. مقادیر غیرمعتبر (عناصر ماسک) در نظر گرفته نمی‌شوند و در مرتب‌سازی حذف می‌شوند. علاوه بر این، می‌توانید آرگومان‌های کلیدی مانند axis، kind، order، endwith و fill\_value را برای تنظیمات مرتب‌سازی استفاده کنید.

510-. ma.MaskedArray.argsort([axis, kind, order, ...]):

- axis (اختیاری): محوری که بر روی آن مقادیر آرایه مرتب شده می‌شوند.

- kind (اختیاری): نوع مرتب‌سازی مورد استفاده. از مقادیر ممکن مانند 'quicksort'، 'mergesort'، 'heapsort' استفاده کنید.

- order (اختیاری): نام ستونها برای استفاده در مرتب‌سازی ساختار داده‌های رکوردی.

- endwith (اختیاری): اندیس انتهایی برای مرتب‌سازی.

- fill\_value (اختیاری): مقدار پر کننده برای عناصر نامعتبر.

تابع ma.MaskedArray.argsort اندیس‌های مرتب‌سازی شدهٔ مقادیر آرایهٔ ماسک شده را برمی‌گرداند. می‌توانید آرگومان‌های کلیدی مانند axis، kind، order، endwith و fill\_value را برای تنظیمات مرتب‌سازی استفاده کنید.

511-. ma.MaskedArray.sort([axis, kind, order, ...]):

- axis (اختیاری): محوری که بر روی آن مقادیر آرایه مرتب شده می‌شوند.

- kind (اختیاری): نوع مرتب‌سازی مورد استفاده. از مقادیر ممکن مانند 'quicksort'، 'mergesort'، 'heapsort' استفاده کنید.

- order (اختیاری): نام ستون‌ها برای استفاده در مرتب‌سازی ساختار داده‌های رکوردی.

- endwith (اختیاری): اندیس انتهایی برای مرتب‌سازی.

- fill\_value (اختیاری): مقدار پر کننده برای عناصر نامعتبر.

تابع ma.MaskedArray.sort آرایهٔ ماسک شده را مرتب می‌کند. عناصر غیرمعتبر (عناصر ماسک) در نظر گرفته نمی‌شوند و در مرتب‌سازی حذف می‌شوند. علاوه بر این، می‌توانید آرگومان‌های کلیدی مانند axis، kind، order، endwith و fill\_value را برای تنظیمات مرتب‌سازی استفاده کنید.

512-. ma.diag(v[, k]):

- v: آرایه ورودی.

- k (اختیاری): شیفت در اصطکاک قطری. اگر k > 0

، اصطکاک به سمت بالا است و اگر k < 0، اصطکاک به سمت پایین است.

تابع ma.diag یک ماتریس قطری را برمی‌گرداند. در صورتی که v یک بردار باشد، خروجی ماتریس قطری با مقادیر ورودی v در قطر خواهد بود. در صورتی که v یک ماتریس باشد، خروجی ماتریس قطری با مقادیر قطر v خواهد بود. با استفاده از آرگومان k، می‌توانید اصطکاک قطری را تغییر دهید.

513-. ma.dot(a, b[, strict, out]):

- a: آرایهٔ ورودی اول.

- b: آرایهٔ ورودی دوم.

- strict (اختیاری): در صورت True، اندازهٔ آرایه‌ها باید منطبق باشد.

- out (اختیاری): آرایهٔ خروجی.

تابع ma.dot ضرب داخلی دو آرایه را محاسبه می‌کند. اندازهٔ آرایه‌ها باید منطبق باشد، مگر اینکه strict را False قرار دهید. می‌توانید آرگومان out را برای تعیین آرایهٔ خروجی استفاده کنید.

514-. ma.identity(n[, dtype]):

- n: اندازهٔ ردیف‌ها و ستون‌های ماتریس هویز.

تابع ma.identity یک ماتریس هویز را با اندازهٔ n برمی‌گرداند. ماتریس هویز یک ماتریس مربعی است که در قطر آن تمامی عناصر غیرصفر و در سایر قسمت‌ها عناصر صفر قرار دارند.

515-. ma.inner(a, b, /):

- a: آرایهٔ ورودی اول.

- b: آرایهٔ ورودی دوم.

تابع ma.inner ضرب داخلی دو آرایه را محاسبه می‌کند. این تابع به‌طور پیش‌فرض برای آرایه‌های ۱ بعدی عمل می‌کند.

516-. ma.innerproduct(a, b, /):

- a: آرایهٔ ورودی اول.

- b: آرایهٔ ورودی دوم.

تابع ma.innerproduct ضرب داخلی دو آرایه را محاسبه می‌کند. این تابع به‌طور پیش‌فرض برای آرایه‌های ۱ بعدی عمل می‌کند و معادل با تابع ma.inner است.

517-. ma.outer(a, b):

- a: آرایهٔ ورودی اول.

- b: آرایهٔ ورودی دوم.

تابع ma.outer ضرب خارجی دو آرایه را محاسبه می‌کند. خروجی این تابع یک آرایه با اندازهٔ (len(a), len(b)) است که عناصر آن برابر حاصلضرب هر جفت عنصر از آرایه‌های a و b است.

518-. ma.outerproduct(a, b):

- a: آرایهٔ ورودی اول.

- b: آرایهٔ ورودی دوم.

تابع ma.outerproduct ضرب خارجی دو آرایه را محاسبه می‌کند. این تابع معادل با تابع ma.outer است.