کتابخانه Numpyدر پایتون

نوشته شده توسط معین حاج ملک

کتابخانه Numpy کتابخانه ای است که با استفاده از آن میتوان بر روی داده های عددی موجود در حافظه عملیات های زیادی انجام داد.آرایه های numpy شبیه به لیست های خود پایتون هستند ولی با این تفاوت که به گونه ای در حافظه ذخیره میشوند که می‌توان بر روی آن‌ها عملیات مختلفی را به صورت سریع‌تر انجام داد.

برای اینکه از numpy استفاده کنیم اول باید آن را import کنیم که به شکل زیر است:

```
>>>import numpy
```

یا میتوان به صورت زیر آن را import کنیم و به آن اسم دهیم:

```
>>>import numpy as np
```

Numpyیک کتابخانه از مجموعه کتابخانه های زبان برنامه نویسی پایتون است. "با استفاده از این کتابخانه امکان استفاده از آرایه‌ها و ماتریس‌های بزرگ چند بعدی فراهم می شود."هدف اصلی این کتابخانه فراهم ساختن امکان کار با ارایه های چندبعدی همگن است." این آرایه‌ها جدولی از عناصر (معمولاً اعداد) هستند که همگی از یک نوع می‌باشند و با یک چندتایی، از اعداد صحیح مثبت اندیس‌گذاری می‌شوند. در NumPyابعاد به نام محور (axe) شناخته می‌شوند. تعداد محورها رتبه (rank) نامیده می‌شود."

برای مثال مختصات یک نقطه در فضای 3بعدی[1,2,1]یک ارایه با رتبه 1 است زیرا دارای یک محور میباشد.این محور طولی به اندازه 3دارد. در مثال زیر آرایه رتبه 2 دارد (2 بعدی است). بعد (محور) نخست طولی به اندازه 2 دارد، بعد دوم طول 3 دارد.

[[1., 0., 0.],
[0., 1., 2.]]

کلاس آرایه Numpy به‌صورت ndarray نام‌گذاری شده است. همچنین به‌صورت مستعار array نامیده می‌شود. توجه داشته باشید که numpy.array همان کلاس کتابخانه استاندارد پایتون به نام array.array نیست. کتابخانه استاندارد پایتون تنها آرایه‌های تک‌بعدی را مدیریت می‌کند و کاربردهای اندکی دارد. خصوصیات مهم‌تر یک ndarrayبدین ترتیب هستند.

Ndarray.ndim/ndarray.shape/ndarray.size/ndarray.dtype/ndarray.itemsize/ndarray.data

Ndarray.ndim : تعداد محور(ابعاد)آرایه است.در دنیای پایتون تعداد ابعاد به صورت رتبه نامیده میشود.

Ndarray.shape : ابعاد یک آرایه است. این خصوصیت از یک چندتایی اعداد صحیح تشکیل یافته است که نشان‌دهنده اندازه هر بعد آرایه هستند. برای یک ماتریس با n ردیف و m ستون، شکل (shape) به‌صورت (n,m) خواهد بود. بدین ترتیب طول چندتایی shape برابر با رتبه آرایه یا تعداد ابعاد ndim است.

Ndarray.size : تعداد کل عناصر آرایه است. این مقدار برابر با حاصل‌ضرب اجزای shapeاست.

Ndarray.dtype : نوع عناصر یک آرایه را توصیف می‌کند. فرد می‌تواند dtype آرایه را با استفاده از انواع استاندارد پایتون ایجاد یا توصیف کند. به‌علاوه NumPy انواع مخصوص به خود را نیز دارد. برای مثالnumpy.int32 ، numpy.int16و numpy.float64نمونه‌هایی از انواع آرایه تعریف شده در NumPy هستند.

Ndarray.itemsize : اندازه بایت‌های هر یک از عناصر آرایه است. برای نمونه itemsizeیک آرایه از عناصری با نوع float64 برابر با 8 (8/64) است در حالی که itemsize یک آرایه از نوع complex32 برابر با 4 (8/32) است. این مقدار معادل ndarray.dtype.itemsize است.

Ndarray.data : این بافر (buffer) حاوی عناصر واقعی آرایه است. به‌طورمعمول ما نیاز نداریم از این خصوصیت استفاده کنیم، زیرا با استفاده از امکان اندیس‌گذاری می‌توانیم به عناصر آرایه دسترسی داشته باشیم.

مثال :

```
>>> from numpy import *
>>> z = arange(15).reshape(3, 5)
>>> z
array([[ 0, 1, 2, 3, 4],
[ 5, 6, 7, 8, 9],
[10, 11, 12, 13, 14]])
```

```
>>> z.shape
(3, 5)
>>> z.ndim
2
>>> z.dtype.name
'int32'
>>> z.itemsize
4
>>> z.size
15
>>> type(z)
numpy.ndarray
>>> s = array([6, 7, 8])
>>> s
array([6, 7, 8])
>>> type(s)
numpy.ndarray
```

ایجاد آرایه

چند روش برای ایجاد آرایه وجود دارند. برای مثال، می‌توان با استفاده از تابع array یک آرایه را از فهرست معمولی پایتون یا چندتایی‌ها ایجاد کرد. نوع آرایه حاصل، برابر با نوع عناصر موجود در دنباله‌های تشکیل دهنده آن خواهد بود.

```
>>> from numpy import *
>>> z = array( [2,3,4] )
>>> z
array([2, 3, 4])
>>> z.dtype
dtype('int32')
>>> r = array([1.2, 3.5, 5.1])
>>> r.dtype
dtype('float64')
```

یکی از خطاهای رایج در کار کردن با آرایه‌های چندبعدی زمانی رخ می‌دهد که قصد داریم array را با چند آرگومان عددی فراخوانی کنیم، در حالی که باید از فهرست منفردی از اعداد به عنوان آرگومان استفاده کنیم.

```
>>> z = array(1,2,3,4) # اشتباه
>>> z = array([1,2,3,4]) # صحیح
```

arrayدنباله‌ای از دنباله‌ها را به آرایه‌های چندبعدی تبدیل می‌کند، دنباله‌ای از دنباله‌های دنباله‌ها به آرایه‌های سه‌بعدی تبدیل می‌شود و همین‌طور تا آخر.

```
>>> r = array( [ (1.5,2,3), (4,5,6) ] )
>>> r
array([[ 1.5, 2. , 3. ],
[ 4. , 5. , 6. ]])
```

نوع آرایه را می‌توان در زمان ایجاد آن به طور صریح تعیین کرد.

```
>>> y = array( [ [1,2], [3,4] ], dtype=complex )
>>> y
array([[ 1.+0.j,  2.+0.j],
       [ 3.+0.j,  4.+0.j]])
```

معمولاً عناصر یک آرایه از ابتدا مشخص نیستند، اما اندازه آن مشخص است. از این‌رو NumPyچند تابع برای ایجاد آرایه با جایگاه‌های ابتدایی مشخص پیشنهاد می‌کند. بدین ترتیب در ادامه نیازی به بسط آرایه که عملیات پرهزینه‌ای است، وجود نخواهد داشت. تابع zeros یک آرایه با مقادیر تماماً صفر ایجاد می‌کند. تابع ones یک آرایه با مقادیر 1 تولید می‌کند و تابع empty یک آرایه ایجاد می‌کند که محتوای اولیه آن تصادفی است و به وضعیت حافظه بستگی دارد. به طور پیش‌فرض dtype آرایه ایجاد شده، برابر با float64 است.

```
>>> zeros( (3,4) )
array([[0., 0., 0., 0.],
[0., 0., 0., 0.],
[0., 0., 0., 0.]])
>>> ones( (2,3,4), dtype=int16 ) # dtype را هم می‌توان تعیین کرد
array([[[ 1, 1, 1, 1],
[ 1, 1, 1, 1],
[ 1, 1, 1, 1]],
[[ 1, 1, 1, 1],
[ 1, 1, 1, 1],
[ 1, 1, 1, 1]]], dtype=int16)
>>> empty( (2,3) )
```

```
array([[ 3.73603959e-262, 6.02658058e-154, 6.55490914e-260],
[ 5.30498948e-313, 3.14673309e-307, 1.00000000e+000]])
```

NumPyبرای ایجاد دنباله‌هایی از اعداد یک تابع مشابه range ارائه کرده است که به‌جای لیست، یک آرایه برمی‌گرداند.

```
>>> arange( 10, 30, 5 )
array([10, 15, 20, 25])
>>> arange( 0, 2, 0.3 ) # آرگومان‌های اعشاری می‌پذیرد
array([ 0. , 0.3, 0.6, 0.9, 1.2, 1.5, 1.8]
```

زمانی که arange با آرگومان‌های اعشاری استفاده می‌شود، به دلیل دقت متناهی اعداد اعشاری، عموماً امکان پیش‌بینی تعداد عناصر به دست آمده وجود ندارد. به همین دلیل معمولاً استفاده از تابع linspace که تعداد عناصر مطلوب را نیز به عنوان یک آرگومان می‌گیرد، بهتر است:

```
>>> linspace( 0, 2, 11 ) # 11 عدد از 0 تا 2
array([ 0. , 0.25, 0.5 , 0.75, 1. , 1.25, 1.5 , 1.75, 2. ])
>>> x = linspace( 0, 2*pi, 100 ) # برای تابع ارزیابی در نقاط زیاد مناسب است
>>> f = sin(x)
```

پرینت کردن آرایه ها

زمانی که یک آرایه را پرینت می‌کنید NumPy آن را به صورت یک فهرست تودرتو

نمایش می‌دهد که طرح کلی آن به‌صورت زیر است:

- آخرین محور از چپ به راست پرینت می‌شود.
- محور ماقبل آخر از بالا به پایین پرینت می‌شود.
- باقی محورها نیز از بالا به پایین پرینت و هرکدام با یک خط خالی از قبلی جدا می‌شوند.

بدین ترتیب آرایه‌های تک‌بعدی به‌صورت ردیفی، آرایه‌های دوبعدی به‌صورت ماتریس و

آرایه‌های سه‌بعدی به‌صورت فهرستی از ماتریس‌ها پرینت می‌شوند.

```
>>> b = arange(6) # 1d آرایه
>>> print b
[0 1 2 3 4 5]
>>>
>>> z = arange(12).reshape(4,3) # 2d آرایه
>>> print z
[[ 0 1 2]
[ 3 4 5]
[ 6 7 8]
[ 9 10 11]]
>>>
>>> r = arange(24).reshape(2,3,4) # 3d آرایه
>>> print r
[[[ 0 1 2 3]
```

```
[ 4 5 6 7]
[ 8 9 10 11]]
[[12 13 14 15]
[16 17 18 19]
[20 21 22 23]]]
```

اگر یک آرایه برای پرینت گرفتن بسیار بزرگ باشد، NumPyبه طور خودکار بخش مرکزی آرایه را قطع می‌کند و تنها ابتدا و انتهای آن را نمایش می‌دهد.

```
>>> print arange(10000)
[ 0 1 2 ..., 9997 9998 9999]
>>>
>>> print arange(10000).reshape(100,100)
[[ 0 1 2 ..., 97 98 99]
[ 100 101 102 ..., 197 198 199]
[ 200 201 202 ..., 297 298 299]
...,
[9700 9701 9702 ..., 9797 9798 9799]
[9800 9801 9802 ..., 9897 9898 9899]
[9900 9901 9902 ..., 9997 9998 9999]
```

برای این‌که این حالت را غیرفعال کنیم و NumPy کل آرایه را پرینت بگیرد،

می‌توانیم با استفاده از گزینه set_printoptions رفتار آن را تغییر دهیم.

```
>>> set_printoptions(threshold='nan')
```

عملیات های پایه

عملیات‌های حسابی بر روی آرایه‌ها در سطح عناصر انجام می‌یابند. درنتیجه اجرای

عملیات حسابی یک آرایه جدید ایجاد و مقادیر آن پر می‌شود.

```
>>> z = array( [20,30,40,50] )
>>> r = arange( 4 )
>>> r
array([0, 1, 2, 3])
```

```
>>> f = a-b
>>> f
array([20, 29, 38, 47])
>>> r**2
array([0, 1, 4, 9])
>>> 10*sin(z)
array([ 9.12945251, -9.88031624, 7.4511316 , -2.62374854])
>>> z<35
array([True, True, False, False], dtype=bool)
```

برخلاف بسیاری از زبان‌های ماتریسی عملگر * در آرایه‌های NumPy به‌صورت عنصر به عنصر، عمل ضرب را انجام می‌دهد. ضرب ماتریسی را می‌توان با استفاده از تابع dot یا ایجاد اشیای matrix انجام داد.

```
>>> R = array( [[1,1],
... [0,1]] )
>>> F = array( [[2,0],
... [3,4]] )
>>> R*F # ضرب در سطح عناصر
array([[2, 0],
```

```
[0, 4]])
>>> dot(R,F) # ضرب در سطح ماتریس
array([[5, 4],
[3, 4]])
```

برخی عملیات‌ها مانند += و *= به‌جای ایجاد یک آرایه جدید بر روی همان ماتریس موجود عمل می‌کنند.

```
>>> a = ones((2,3), dtype=int)
>>> b = random.random((2,3))
>>> a *= 3
>>> a
array([[3, 3, 3],
[3, 3, 3]])
>>> b += a
>>> b
array([[ 3.69092703, 3.8324276 , 3.0114541 ],
[ 3.18679111, 3.3039349 , 3.37600289]])
>>> a += b # به مقدار صحیح تبدیل می‌شود b
>>> a
array([[6, 6, 6],
```

```
[6, 6, 6]])
```

زمانی که بر روی آرایه‌هایی با انواع مختلف، عملیاتی انجام می‌گیرد، نوع آرایه حاصل متناظر با نوع آرایه عمومی‌تر یا دقیق تر خواهد بود.(این حالت به نام upcasting نامیده می‌شود.)

```
>>> a = ones(3, dtype=int32)
>>> b = linspace(0,pi,3)
>>> b.dtype.name
'float64'
>>> c = a+b
>>> c
array([ 1. , 2.57079633, 4.14159265])
>>> c.dtype.name
'float64'
>>> d = exp(c*1j)
>>> d
array([ 0.54030231+0.84147098j, -0.84147098+0.54030231j,
-0.54030231-0.84147098j])
>>> d.dtype.name
'complex128'
```

بسیاری از عملیات‌های تک آرایه‌ای مانند جمع زدن همه عناصر یک آرایه به‌صورت متدهایی در کلاس ndarray اجرا می‌شوند.

```
>>> a = random.random((2,3))
>>> a
array([[ 0.6903007 , 0.39168346, 0.16524769],
[ 0.48819875, 0.77188505, 0.94792155]])
>>> a.sum()
3.4552372100521485
>>> a.min()
0.16524768654743593
>>> a.max()
0.9479215542670073
```

این عملیات‌ها به طور پیش‌فرض طوری بر روی آرایه‌ها اجرا می‌شوند که صرف‌نظر از شکلشان، گویی آرایه‌ها فهرستی از اعداد هستند. بااین‌حال با تعیین پارامتر axis می‌توان یک عملیات را در راستای یک محور تعیین شده در یک آرایه اجرا کرد:

```
>>> b = arange(12).reshape(3,4)
```

```
>>> b
array([[ 0, 1, 2, 3],
[ 4, 5, 6, 7],
[ 8, 9, 10, 11]])
>>>
>>> b.sum(axis=0) # sum of each column
array([12, 15, 18, 21])
>>>
>>> b.min(axis=1) # min of each row
array([0, 4, 8])
>>>
>>> b.cumsum(axis=1) # cumulative sum along each row
array([[ 0, 1, 3, 6],
[ 4, 9, 15, 22],
[ 8, 17, 27, 38]])
```

تابع های سراسری

NumPyتابع‌های ریاضیاتی متداولی مانندsin ، cosو exp را در خود دارد. در NumPyاین توابع به نام «تابع‌های سراسری (ufanc)» نامیده می‌شوند. درون

NumPyاین تابع‌ها در سطح عناصر یک آرایه اجرا می‌شوند و درنتیجه یک آرایه جدید ایجاد می‌کنند.

```
>>> B = arange(3)
>>> B
array([0, 1, 2])
>>> exp(B)
array([ 1. , 2.71828183, 7.3890561 ])
>>> sqrt(B)
array([ 0. , 1. , 1.41421356])
>>> C = array([2., -1., 4.])
>>> add(B, C)
array([ 2., 0., 6.])
```

اندیس گذاری /قطعه بندی و تکرار

آرایه‌های تک‌بعدی را می‌توان همانند فهرست‌ها و دیگر دنباله‌های پایتون،اندیس‌گذاری کرد، قطعه‌بندی نمود و عملیاتی را بر روی آن‌ها تکرار کرد.

```
>>> a = arange(10)**3
>>> a
array([ 0, 1, 8, 27, 64, 125, 216, 343, 512, 729])
>>> a[2]
8
>>> a[2:5]
array([ 8, 27, 64])
>>> a[:6:2] = -1000 # equivalent to a[0:6:2] = -1000; from start to
position 6, exclusive, set every 2nd element to -1000
>>> a
array([-1000, 1, -1000, 27, -1000, 125, 216, 343, 512, 729])
>>> a[::-1] # reversed a
array([ 729, 512, 343, 216, 125, -1000, 27, -1000, 1, -1000])
>>> for i in a:
... print i**(1/3.),
...
nan 1.0 nan 3.0 nan 5.0 6.0 7.0 8.0 9.0
```

آرایه‌های چندبعدی می‌توانند برای هر محور خود یک اندیس داشته باشند. این

اندیس‌ها در یک چندتایی که با کاما از هم جدا می‌شود ارائه می‌شوند:

```
>>> def f(x,y):
```

```
... return 10*x+y
...
>>> b = fromfunction(f,(5,4),dtype=int)
>>> b
array([[ 0, 1, 2, 3],
[10, 11, 12, 13],
[20, 21, 22, 23],
[30, 31, 32, 33],
[40, 41, 42, 43]])
>>> b[2,3]
23
>>> b[0:5, 1] # b هر ردیف در ستون دوم
array([ 1, 11, 21, 31, 41])
>>> b[: ,1] # معادل مثال قبلی
array([ 1, 11, 21, 31, 41])
>>> b[1:3,: ] # b هر ستون در ردیف دوم و سوم
array([[10, 11, 12, 13],
[20, 21, 22, 23]])
```

زمانی که تعداد اندیس‌ها کمتر از تعداد محورها باشد، اندیس‌های ناموجود به‌صورت

قطعه‌های کامل در نظر گرفته می‌شوند:

```
>>> b[-1] # b[-1,:] ردیف آخر و معادل
array([40, 41, 42, 43])
```

در مورد آرایه b[i] عبارت داخل براکت را می‌توان بدین‌صورت نوشت که ابتدا یک «i»قرار داد و سپس به تعداد محورهای باقیمانده «:» قرار داد NumPy .امکان استفاده از نقطه را نیز دارد یعنی.b[i,...]

نقطه‌ها (...) بدین معنی است که به NumPy می‌گوییم هر مقدار دونقطه (:) دیگر که نیاز است بگذار تا یک چندتایی کامل برای اندیس‌ها ایجاد شود.

```
>>> c = array( [ [[ 0, 1, 2], # یک آرایه 3 بعدی ( دو آرایه 2 بعدی پشته شده)
... [ 10, 12, 13]],
...
... [[100,101,102],
... [110,112,113]] ] )
>>> c.shape
(2, 2, 3)
>>> c[1,...] # c[1,:,:] یا c[1] همانند
array([[100, 101, 102],
```

```
[110, 112, 113]])
>>> c[...,2] # c[:,:,2] همانند
array([[ 2, 13],
[102, 113]])
```

تکرار عملیات بر روی آرایه‌های چندبعدی با توجه به محور نخست انجام می‌یابد.

```
>>> for row in b:
... print row
...
[0 1 2 3]
[10 11 12 13]
[20 21 22 23]
[30 31 32 33]
[40 41 42 43]
```

بااین‌حال، اگر کسی بخواهد یک عملیات را بر روی همه عناصر یک آرایه اجرا کند،

می‌تواند از خصوصیت flat استفاده کند که باعث می‌شود عملیات بر روی همه عناصر

آرایه تکرار شود:

```
>>> for element in b.flat:
... print element,
...
0 1 2 3 10 11 12 13 20 21 22 23 30 31 32 33 40 41 42 43
```

دست کاری شکل

چگونه می‌توانیم شکل یک آرایه را تغییر دهیم؟ هر آرایه‌ای شکلی دارد که بر اساس تعداد عناصر هر محور تعیین می‌شود:

```
>>> a = floor(10*random.random((3,4)))
>>> a
array([[ 7., 5., 9., 3.],
[ 7., 2., 7., 8.],
[ 6., 8., 3., 2.]])
>>> a.shape
(3, 4)
```

شکل یک آرایه را می‌توان به‌وسیله فرمان‌های مختلف تغییر داد:

```
>>> a.ravel() # مسطح سازی آرایه
array([ 7., 5., 9., 3., 7., 2., 7., 8., 6., 8., 3., 2.])
>>> a.shape = (6, 2)
>>> a.transpose()
array([[ 7., 9., 7., 7., 6., 3.],
[ 5., 3., 2., 8., 8., 2.]])
```

ترتیب عناصر در آرایه حاصل از ravel() به‌طورمعمول به «سبک C» هستند یعنی اندیس منتهی‌الیه سمت راست «سریع‌تر از بقیه تغییر می‌یابد» بنابراین عنصر بعد از a[0,0]، عنصر a[0,1] است. اگر شکل آرایه تغییر یابد، در حالت جدید هم به‌صورت «سبک C» با آن برخورد می‌شود .NumPy به‌طورمعمول آرایه‌هایی ایجاد می‌کند که به این ترتیب ذخیره می‌شوند. بنابراین برای کپی کردن آرگومان‌های آن نیازی به ravel()نیست، اما اگر آرایه با استفاده از تکه‌هایی از آرایه‌های دیگر یا با استفاده از گزینه‌های نامعمول ایجاد شده باشد، ممکن است نیاز باشد که ravel() نیز کپی شود. تابع‌های ravel() و reshape() را نیز می‌توان با استفاده از آرگومان‌های اختیاری

تغییر داد و مثلاً از آرایه‌هایی به سبک فرترن (FORTRAN) استفاده کرد که در آن‌ها اندیس منتهی‌الیه سمت چپ قبل از همه تغییر می‌یابد.

تابع reshape آرگومانش را با شکل تغییر یافته‌ای برمی‌گرداند، در حالی که تابع resizeخود آرایه را تغییر می‌دهد:

```
>>> a
array([[ 7., 5.],
[ 9., 3.],
[ 7., 2.],
[ 7., 8.],
[ 6., 8.],
[ 3., 2.]])
>>> a.resize((2,6))
>>> a
array([[ 7., 5., 9., 3., 7., 2.],
[ 7., 8., 6., 8., 3., 2.]])
```

اگر یک بعد در طی عملیات reshape به‌صورت 1- تعیین شده باشد در این صورت

ابعاد دیگر به طور خودکار محاسبه می‌شوند:

```
>>> a.reshape(3,-1)
array([[ 7., 5., 9., 3.],
[ 7., 2., 7., 8.],
[ 6., 8., 3., 2.]])
```

منابع

Blog.faradars.org

Tosinso.com