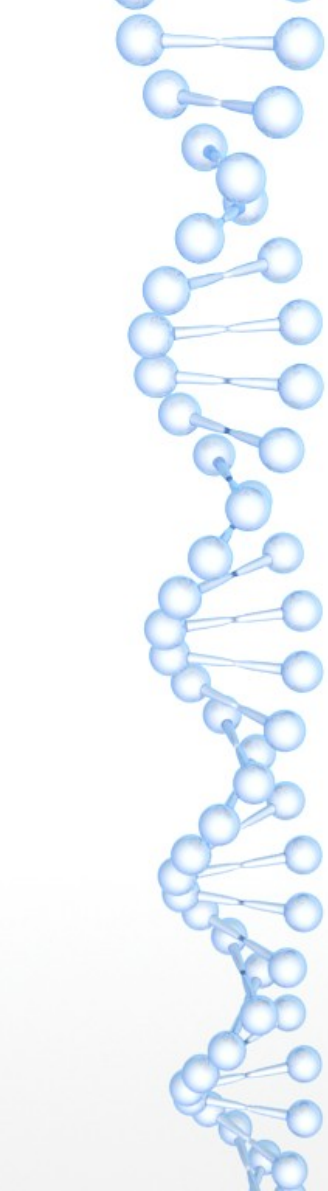
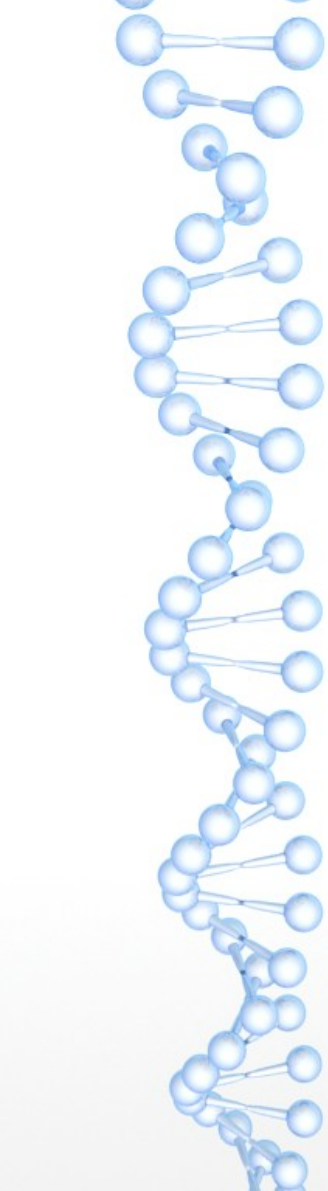
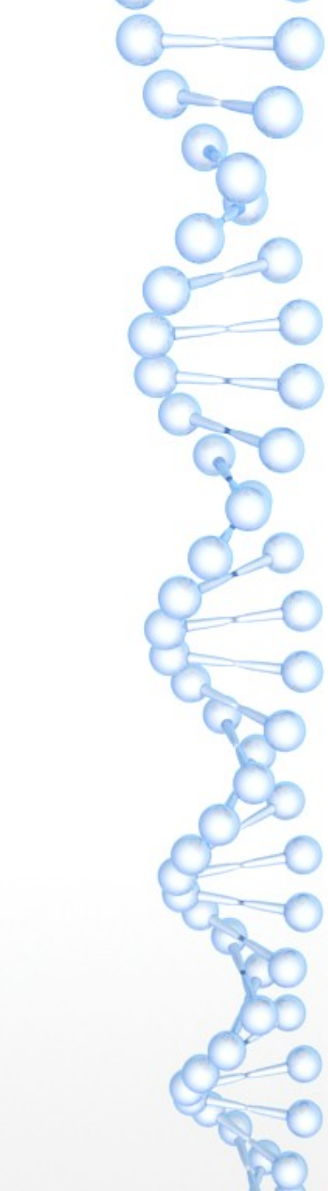


# Numpy



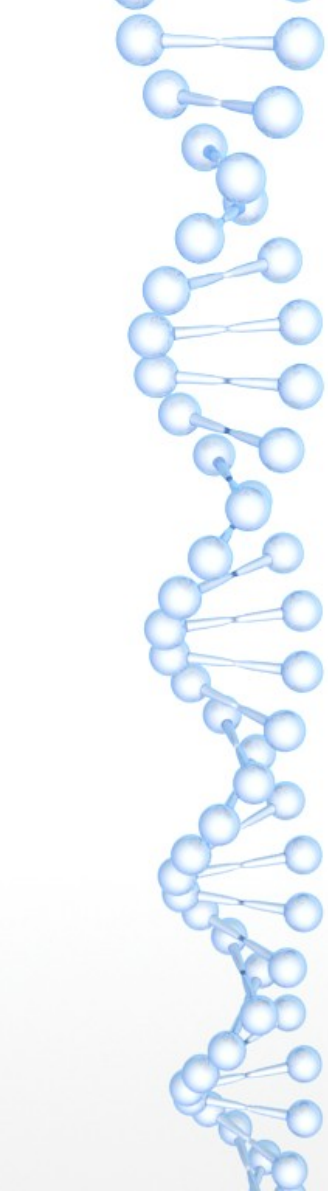
# Why Numpy ?

- What is Numpy ?
- Why Numpy ?
- How fast can it be? (Compared to Python)

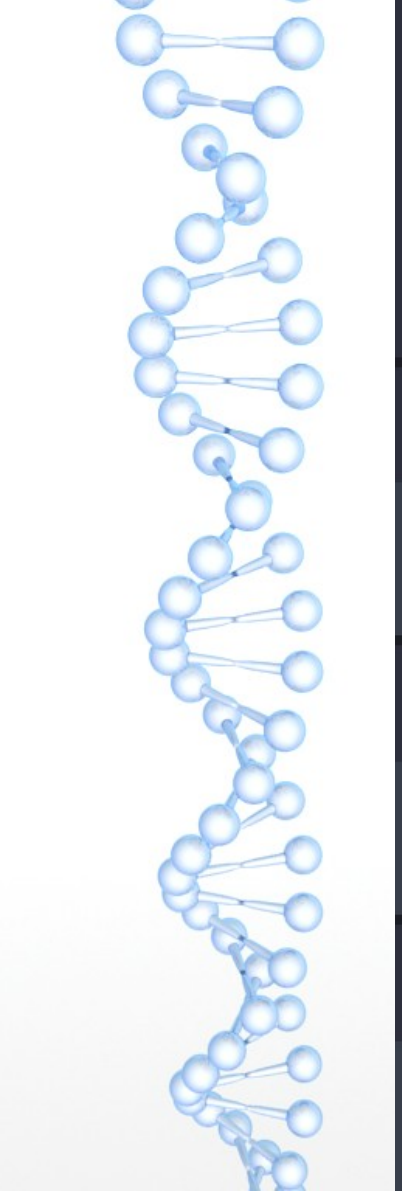


# Numerical Python

- ndarray
- functions
- tools
- C, C++, Fortran(API)



Scipy Matplotlib Scikits ...



In [7]:

```
1 import numpy as np
2 import cupy as cp
3 np_arr = np.arange(100000000)
4 cp_arr = cp.arange(100000000)
5 py_list = list(range(100000000))
```

In [8]:

```
1 %time for _ in range(10): np_arr2 = np_arr * 2
```

```
CPU times: user 928 ms, sys: 713 ms, total: 1.64 s
Wall time: 1.64 s
```

In [9]:

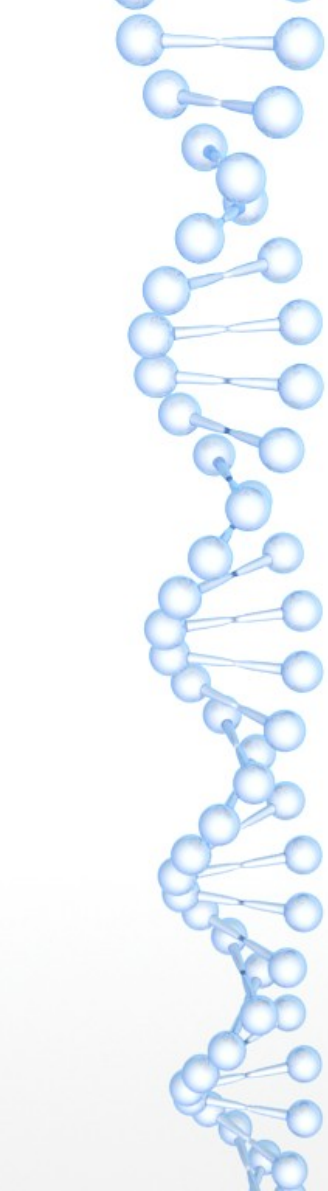
```
1 %time for _ in range(10): py_list2 = [x * 2 for x in py_list]
```

```
CPU times: user 36.7 s, sys: 7.53 s, total: 44.3 s
Wall time: 44.3 s
```

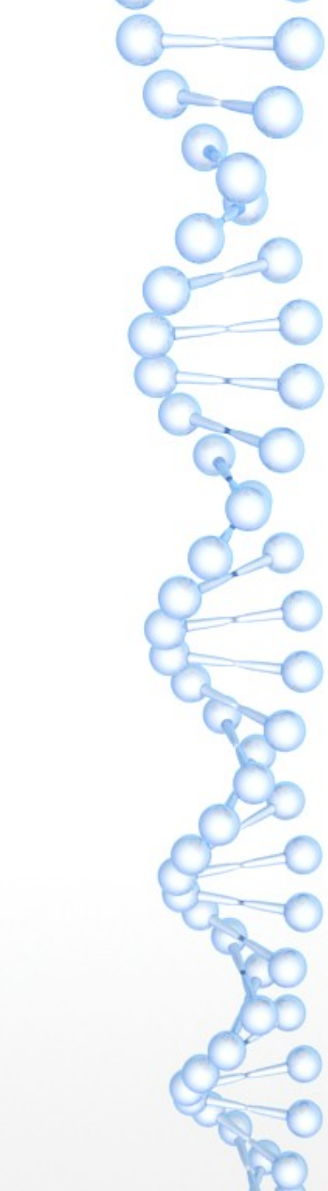
In [10]:

```
1 %time for _ in range(10): cp_arr2 = cp_arr * 2
```

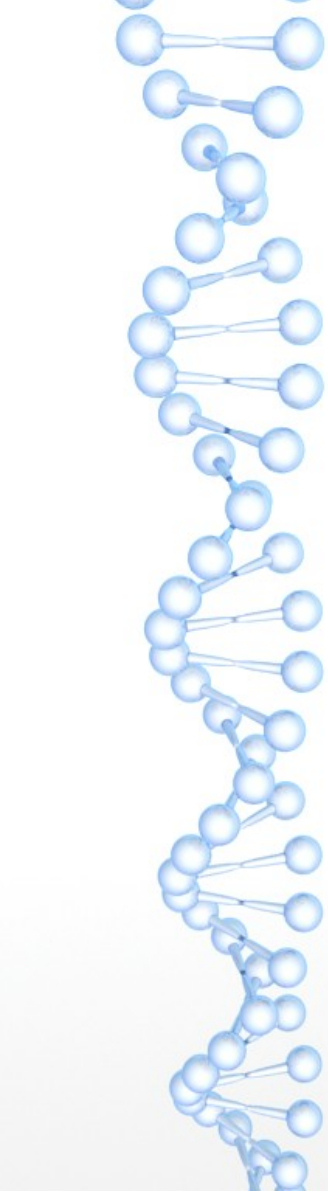
```
CPU times: user 0 ns, sys: 8.08 ms, total: 8.08 ms
Wall time: 7.41 ms
```



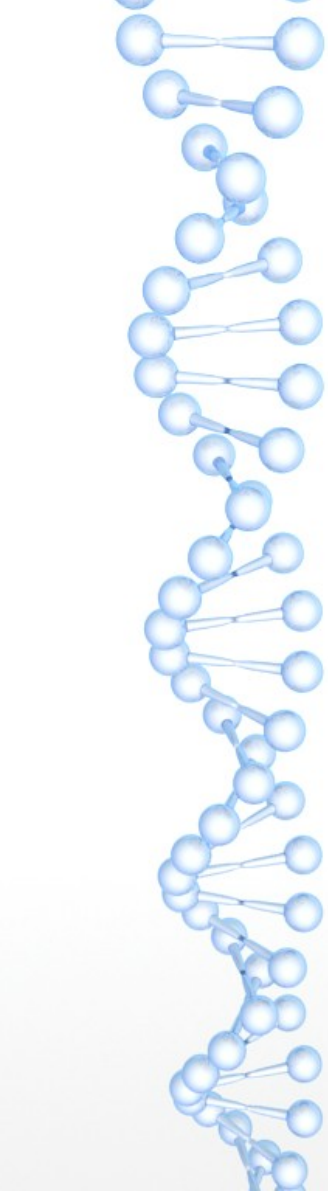
- Interpreted language
- Compiled language



- Dynamically Typed Language
- Statically Typed Language

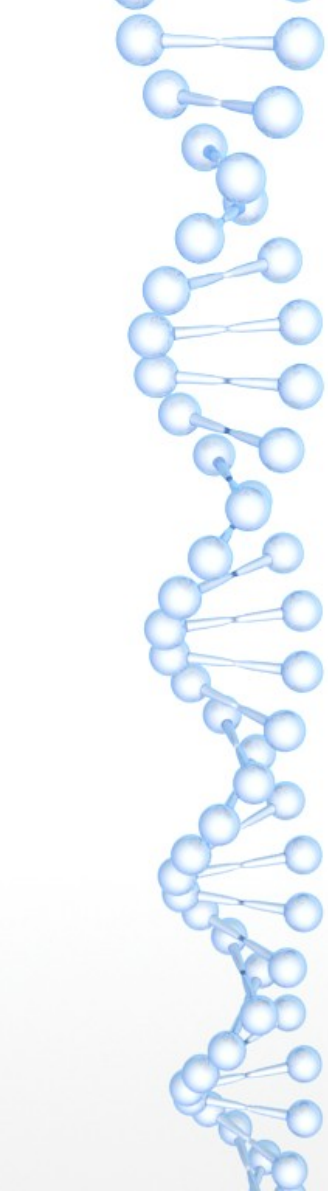


# Global Interpreter Lock



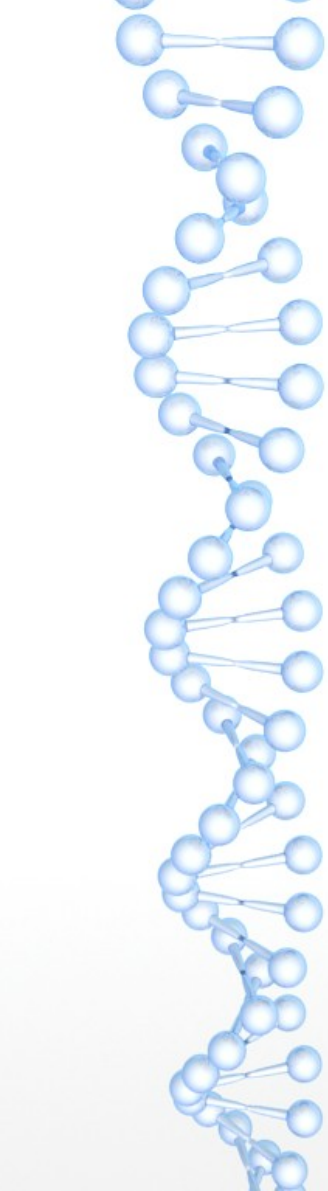
Numba/Pypy
Efficiency in time

Cython
Performance
Static variable

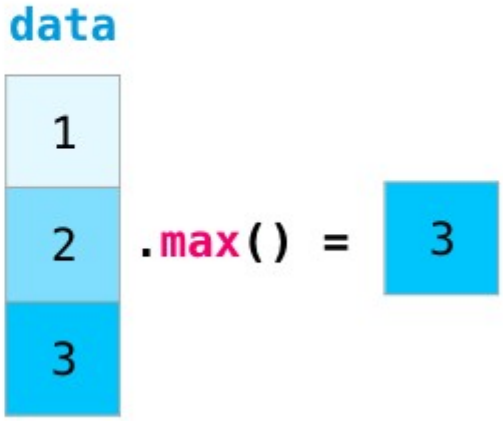


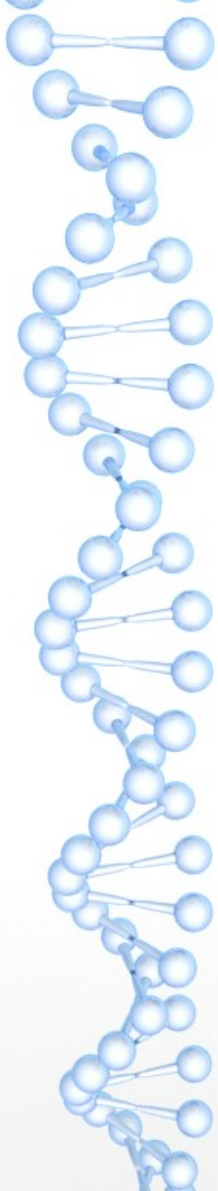
# Numpy

- C(Compiled language)
- BLAS
- Opitimization



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

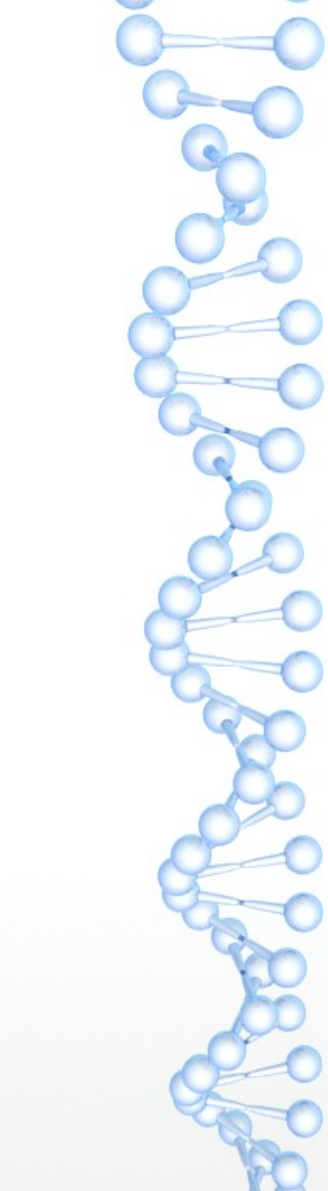





Command

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

NumPy Array



`np.ones(3)`

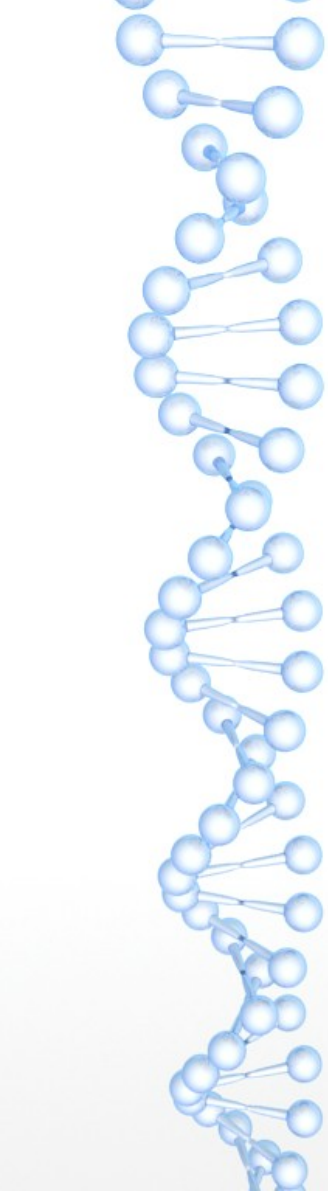
| 1 |
| --- |
| 1 |
| 1 |

`np.zeros(3)`

| 0 |
| --- |
| 0 |
| 0 |

`np.random.random(3)`

| 0.5967 |
| --- |
| 0.0606 |
| 0.2223 |



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

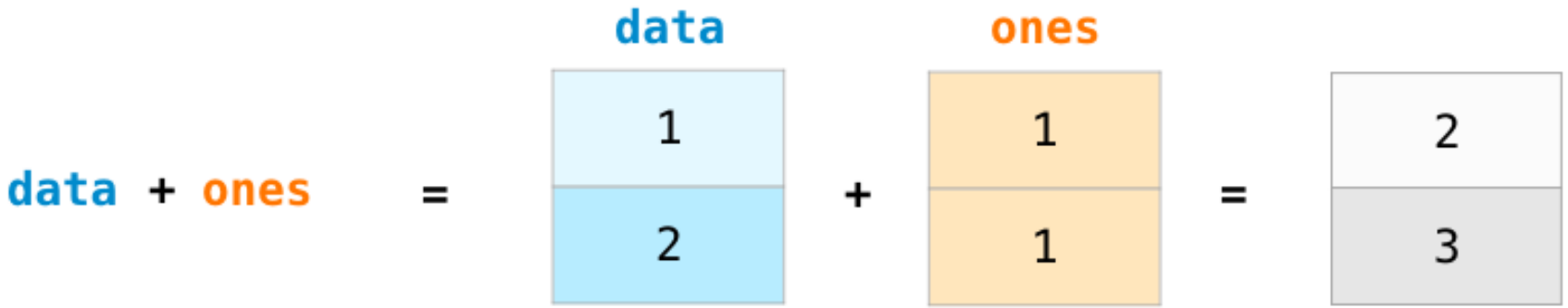
data

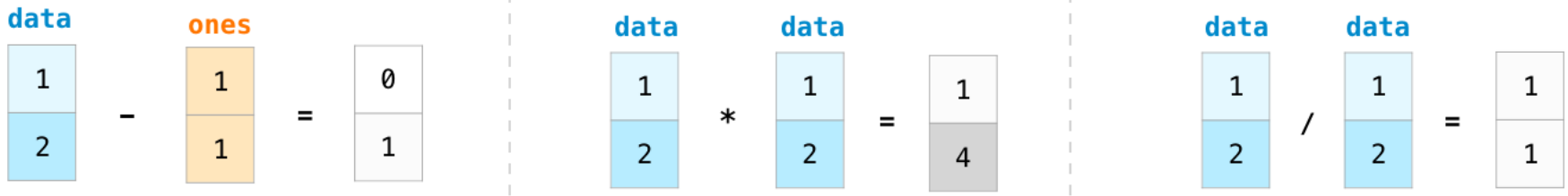
| 1 |
|---|
| 2 |

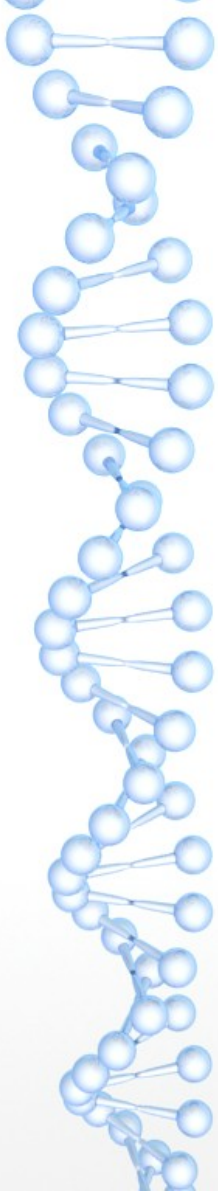
```
ones = np.ones(2)
```

ones

| 1 |
|---|
| 1 |


data + ones
=
data
1
2
+
ones
1
1
=
2
3


data
1
2
-
ones
1
1
=
0
1
data
1
2
*
data
1
2
=
1
4
data
1
2
/
data
1
2
=
1
1



| 1 |
| --- |
| 2 |

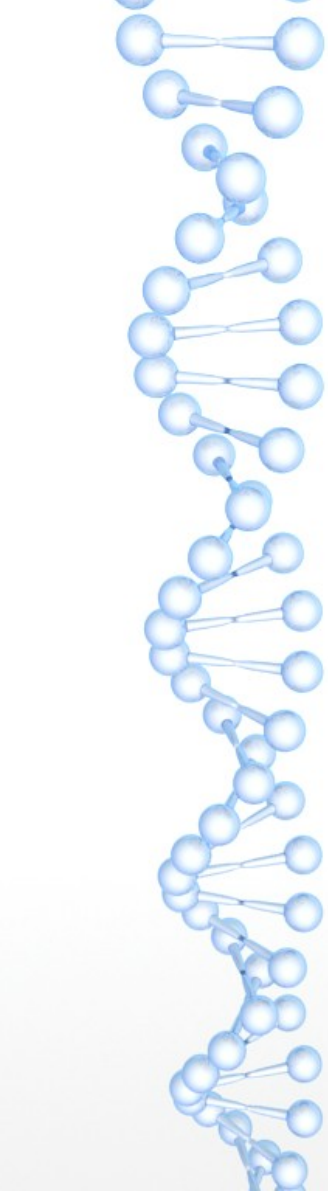
* 1.6 =

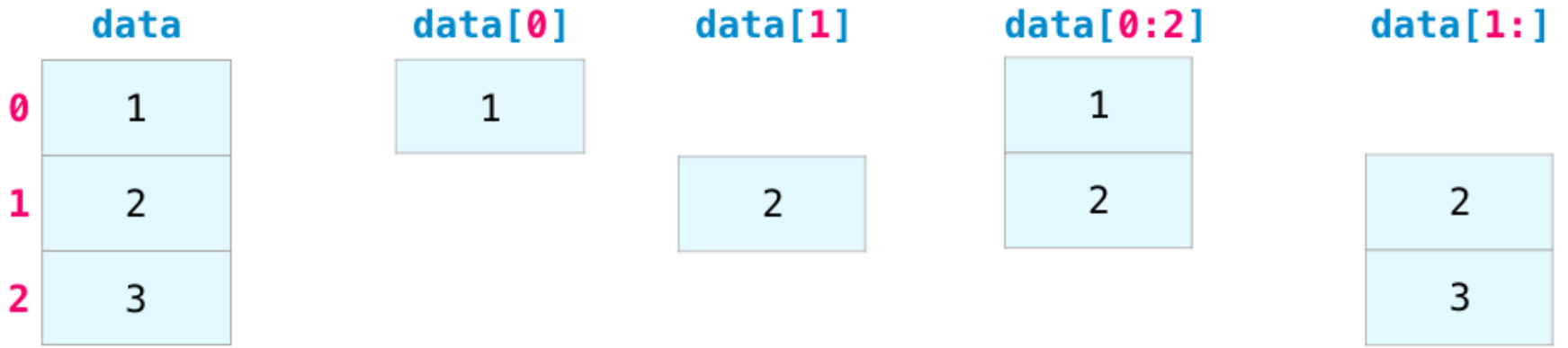
| 1 |
| --- |
| 2 |

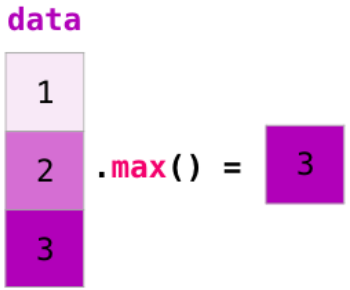
* | 1.6 |
| --- |
| 1.6 |

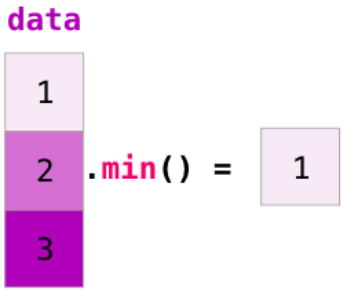
=

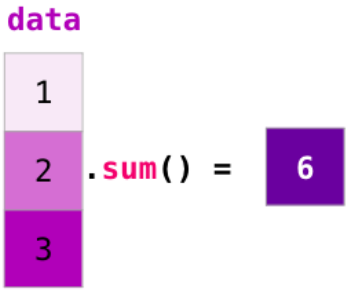
| 1.6 |
| --- |
| 3.2 |

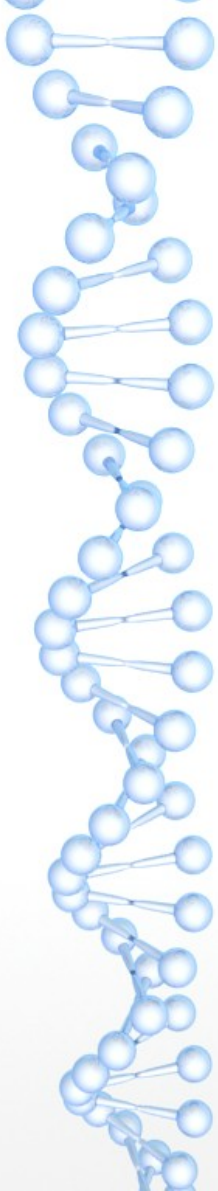



data
0
1
2
1
2
3
data[0]
1
data[1]
2
data[0:2]
1
2
data[1:]
2
3


data
1
2
3
.max() =
3

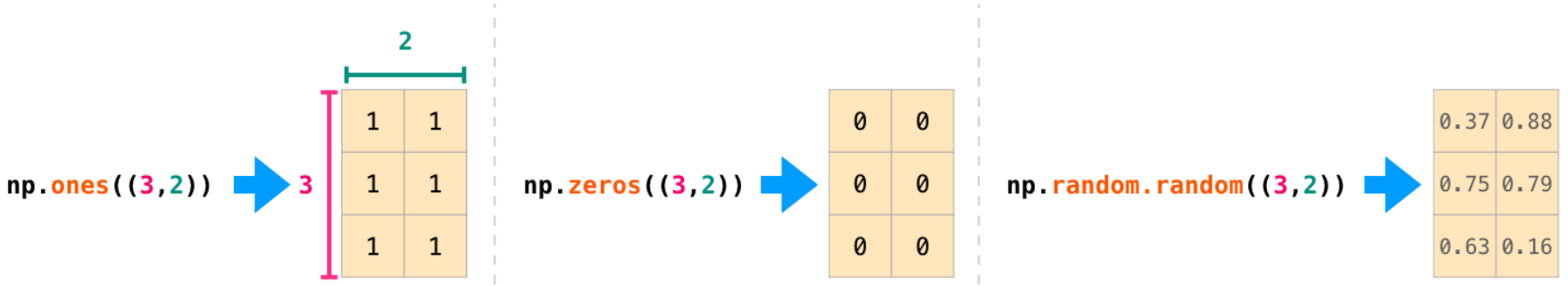

data
1
2
3
.min() =
1

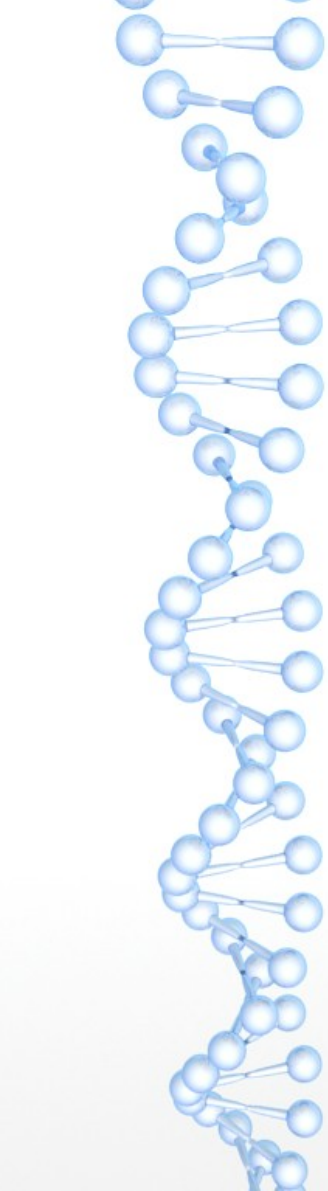

data
1
2
3
.sum() =
6

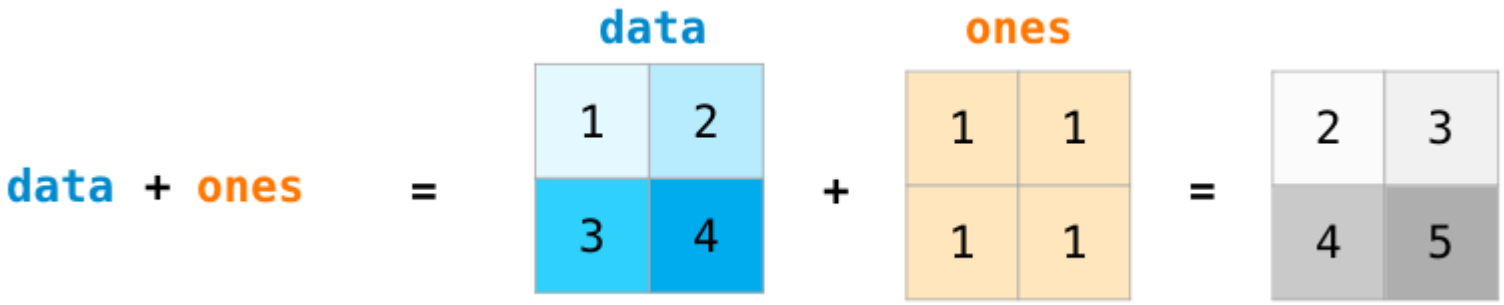


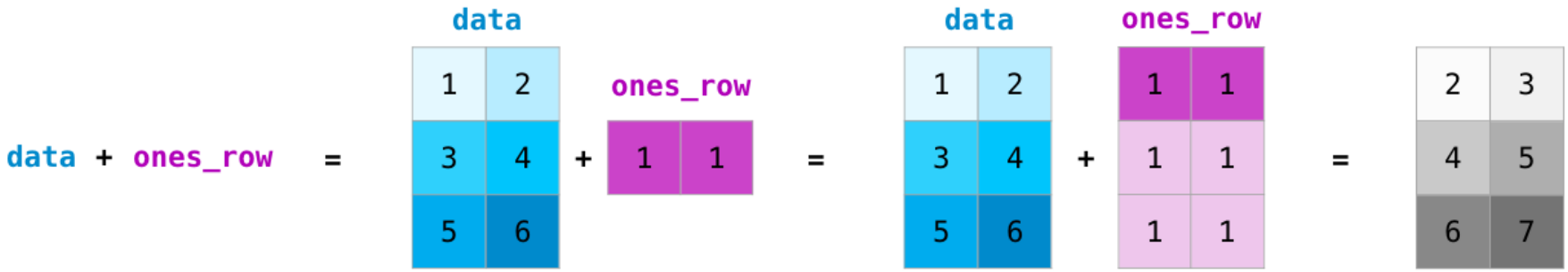
```
np.array([[1,2],[3,4]])
```

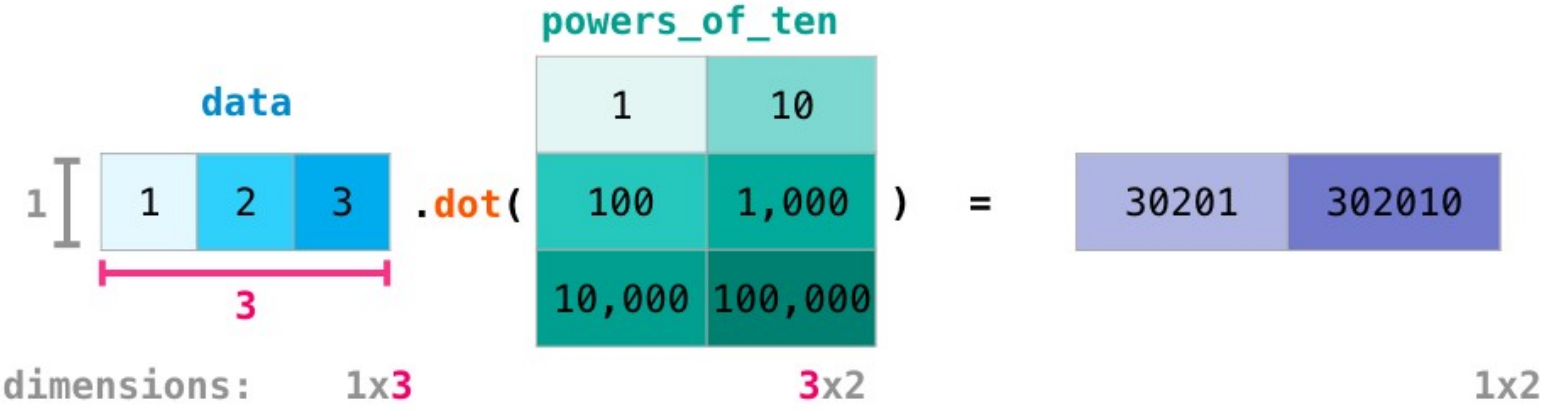
| | |
|---|---|
| 1 | 2 |
| 3 | 4 |


np.ones((3,2))
2
3
1 1
1 1
1 1
np.zeros((3,2))
0 0
0 0
0 0
np.random.random((3,2))
0.37 0.88
0.75 0.79
0.63 0.16



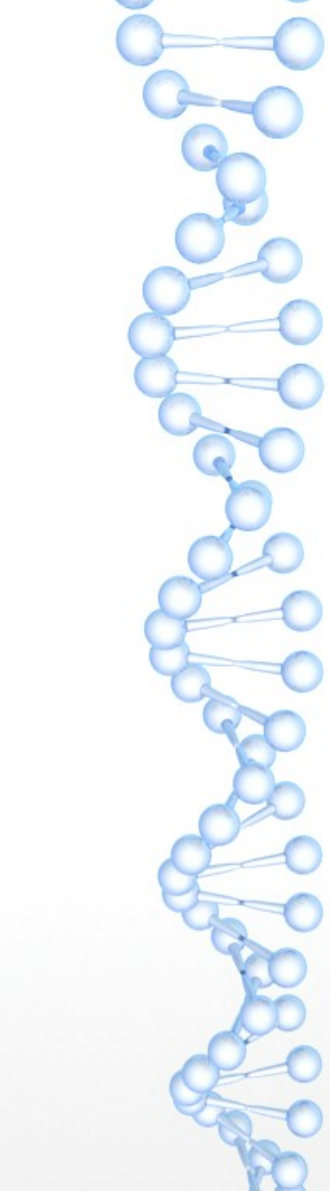

data + ones
=
data
1
2
3
4
+
ones
1
1
1
1
=
2
3
4
5

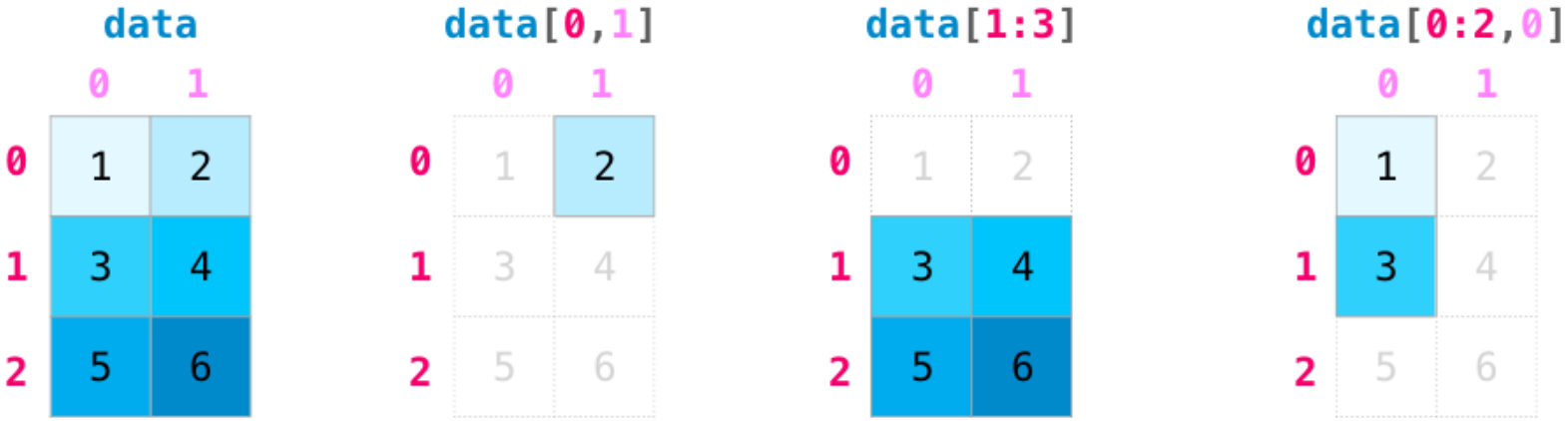

data + ones_row =
data
1 2
3 4
5 6
+
ones_row
1 1
=
data
1 2
3 4
5 6
+
ones_row
1 1
1 1
1 1
=
2 3
4 5
6 7

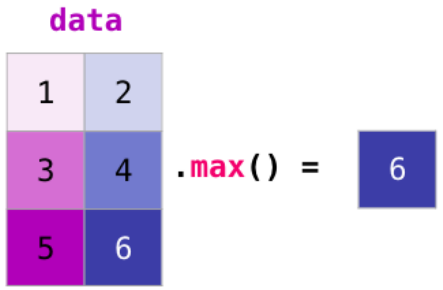

powers_of_ten
data
1
1
2
3
.dot(
100
1,000
10
10,000
100,000
)
=
30201
302010
3
Matrix dimensions:
1x3
3x2
1x2

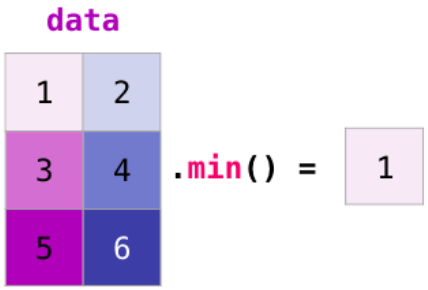

1x2

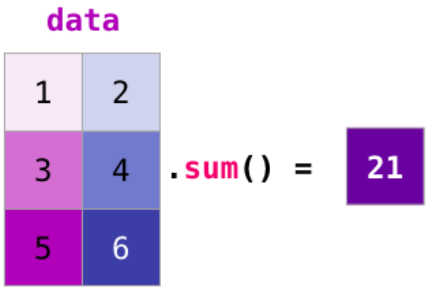
| 1*1 + 2*100 + 3*10,000 | 1*10 + 2*1,000 + 3*100,000 | = | 30201 | 302010 |
|---|---|---|---|---|

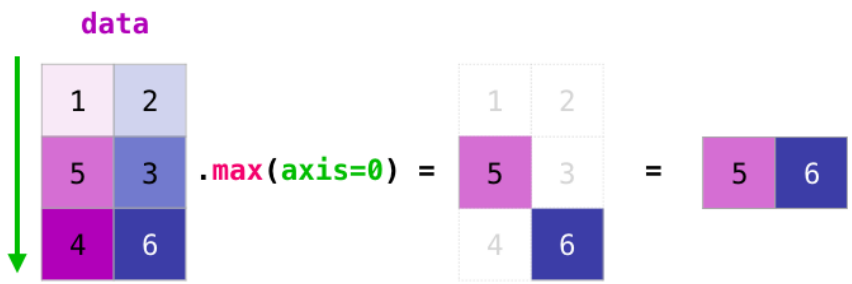


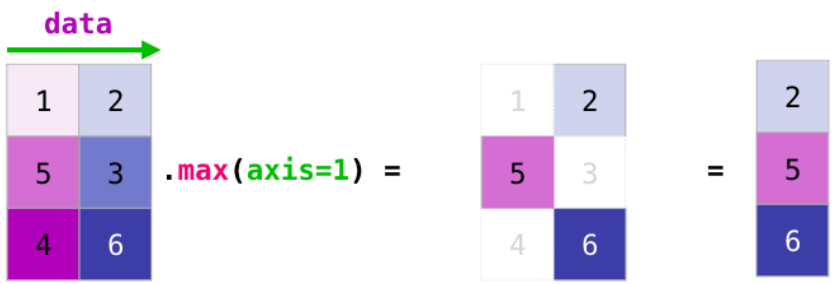

data
0 1
0 1 2
1 3 4
2 5 6
data[0,1]
0 1
0 1 2
1 3 4
2 5 6
data[1:3]
0 1
0 1 2
1 3 4
2 5 6
data[0:2,0]
0 1
0 1 2
1 3 4
2 5 6

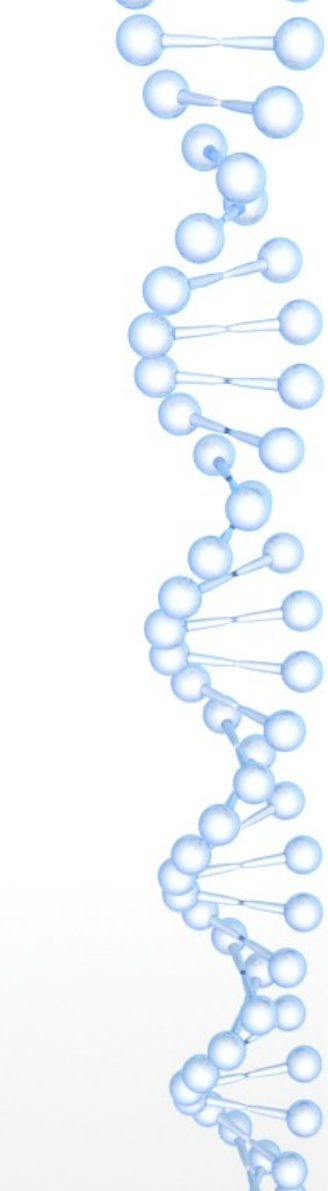

data
1
2
3
4
5
6
.max() =
6


data
1
2
3
4
5
6
.min() =
1


data
1
2
3
4
5
6
.sum() =
21


data
1 2
5 3
4 6
.max(axis=0) =
1 2
5 3
4 6
=
5 6


data
1 2
5 3
4 6
.max(axis=1) =
1 2
5 3
4 6
=
2
5
6

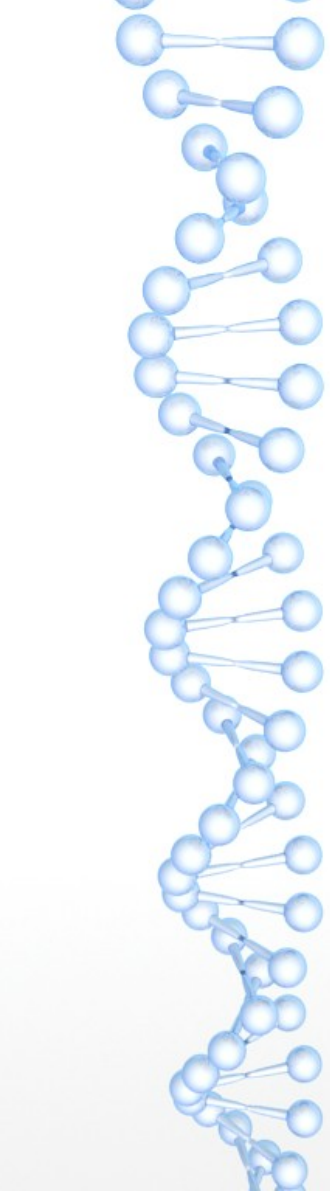


**data**

| | |
|---|---|
| 1 | 2 |
| 3 | 4 |
| 5 | 6 |

**data.T**

| | | |
|---|---|---|
| 1 | 3 | 5 |
| 2 | 4 | 6 |



**data**

| |
|---|
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |

**data.reshape(2,3)**

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |

2

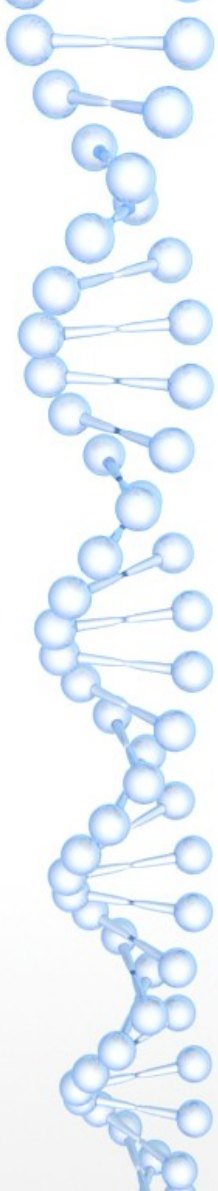
3

**data.reshape(3,2)**

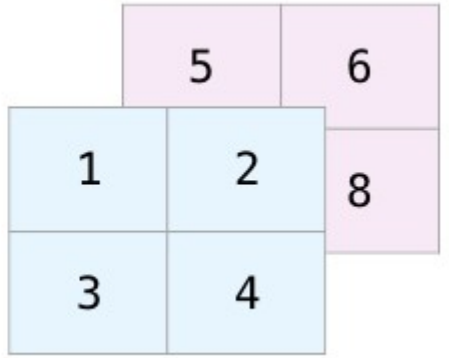
| | |
|---|---|
| 1 | 2 |
| 3 | 4 |
| 5 | 6 |

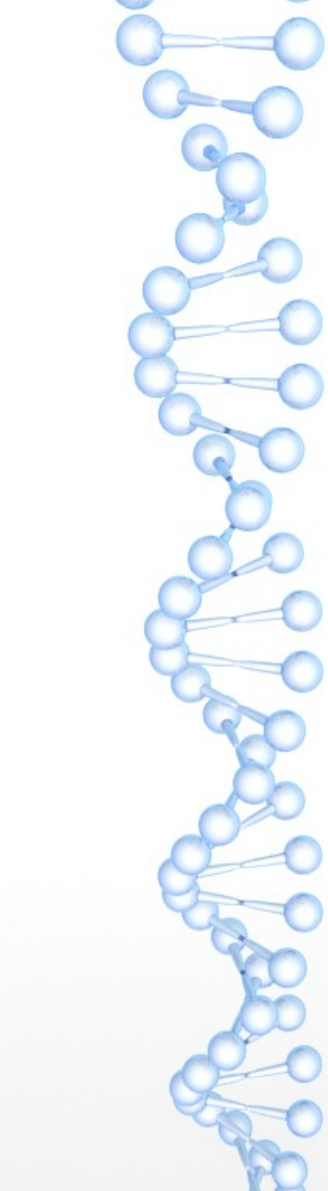
3

2

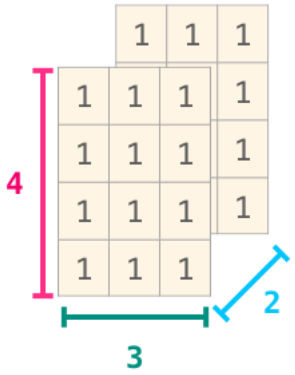


```
np.array([ [[1,2],[3,4]],
           [[5,6],[7,8]] ])
```

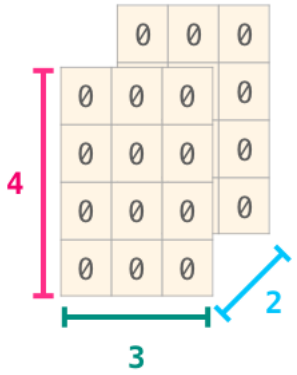




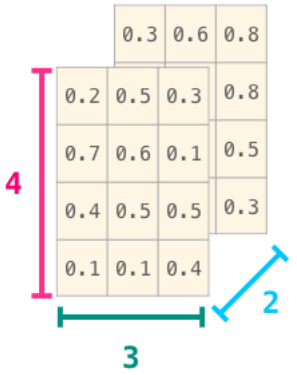
```
np.ones((4,3,2))
```




```
np.zeros((4,3,2))
```




```
np.random.random((4,3,2))
```



$$MeanSquareError = \frac{1}{n}\sum_{i=1}^{n}(Y\_prediction_i - Y_i)^2$$

```
error = (1/n) * np.sum(np.square(predictions - labels))
```

predictions labels

```
error = (1/3) * np.sum(np.square( [1, 1, 1] - [1, 2, 3] ))
```

```
error = (1/3) * np.sum(np.square( [0, -1, -2] ))
```

```
error = (1/3) * np.sum( [0, 1, 4] )
```

```
error = (1/3) * 5
```

# ndarray 对象的属性

| 属性 | 说明 |
| --- | --- |
| .ndim | 秩，即轴的数量或维度的数量 |
| .shape | ndarray对象的尺度，对于矩阵，n行m列 |
| .size | ndarray对象元素的个数，相当于.shape中n*m的值 |
| .dtype | ndarray对象的元素类型 |
| .itemsize | ndarray对象中每个元素的大小，以字节为单位 |



.data　存储地址

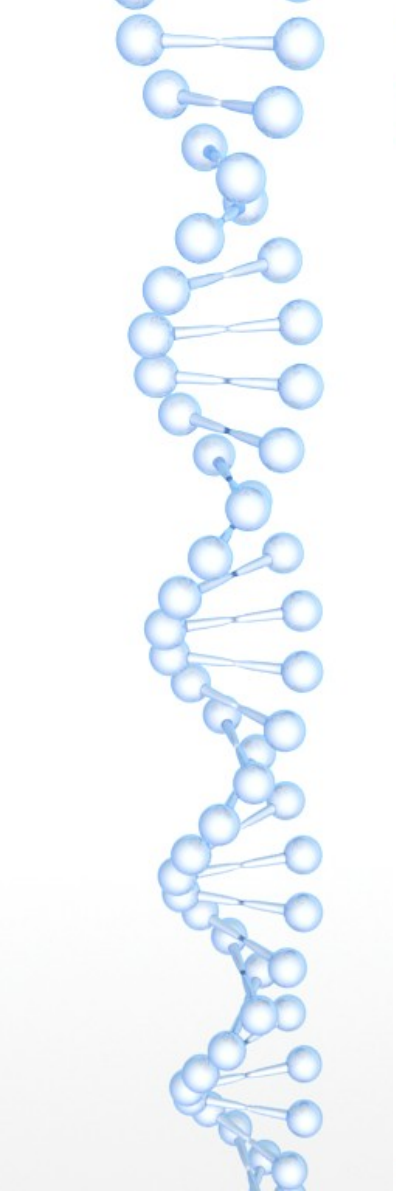


*Table 4-2. NumPy data types*

| Type | Type code | Description |
|---|---|---|
| `int8, uint8` | `i1, u1` | Signed and unsigned 8-bit (1 byte) integer types |
| `int16, uint16` | `i2, u2` | Signed and unsigned 16-bit integer types |
| `int32, uint32` | `i4, u4` | Signed and unsigned 32-bit integer types |
| `int64, uint64` | `i8, u8` | Signed and unsigned 64-bit integer types |
| `float16` | `f2` | Half-precision floating point |
| `float32` | `f4 or f` | Standard single-precision floating point; compatible with C float |
| `float64` | `f8 or d` | Standard double-precision floating point; compatible with C double and Python `float` object |
| `float128` | `f16 or g` | Extended-precision floating point |
| `complex64, complex128, complex256` | `c8, c16, c32` | Complex numbers represented by two 32, 64, or 128 floats, respectively |
| `bool` | ? | Boolean type storing `True` and `False` values |
| `object` | O | Python object type; a value can be any Python object |
| `string_` | S | Fixed-length ASCII string type (1 byte per character); for example, to create a string dtype with length 10, use `'S10'` |
| `unicode_` | U | Fixed-length Unicode type (number of bytes platform specific); same specification semantics as `string_` (e.g., `'U10'`) |

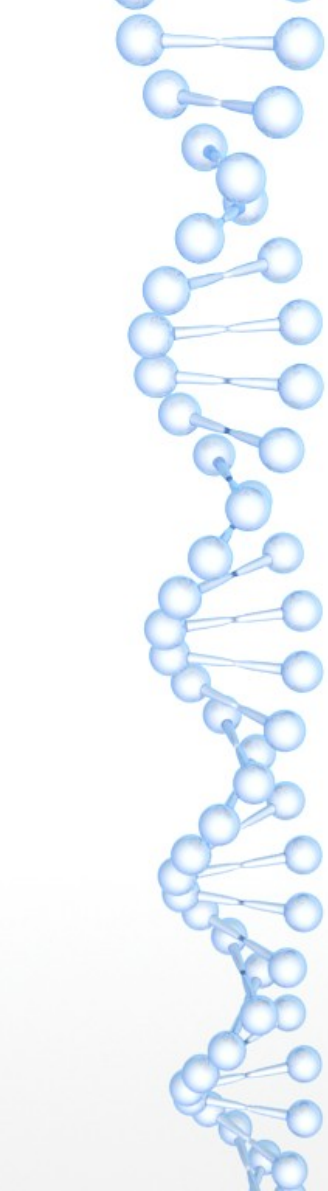
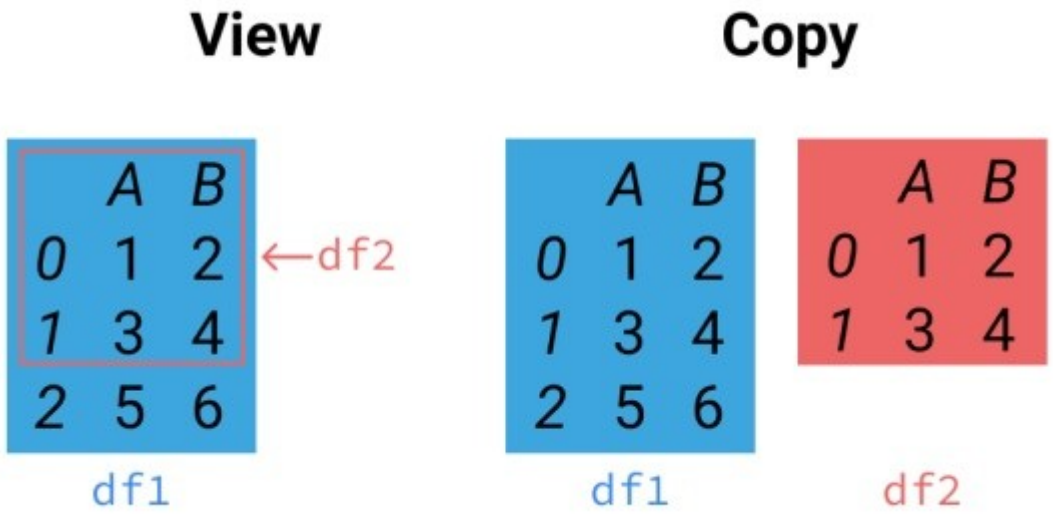


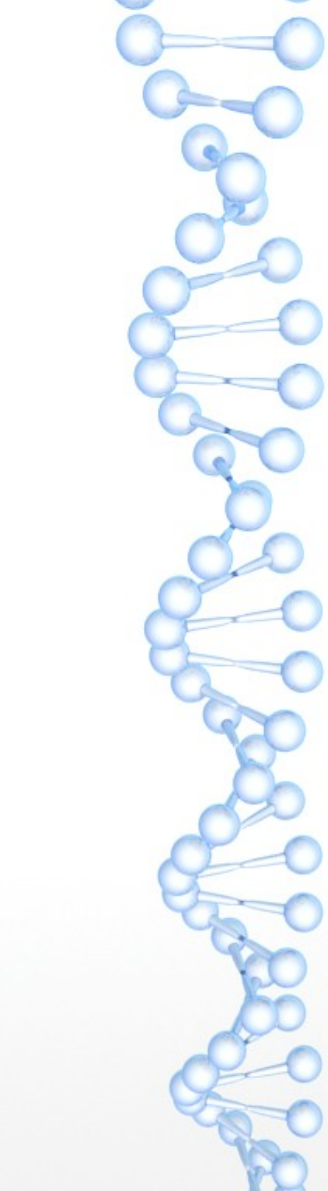
*Table 4-3. Unary ufuncs*

| Function | Description |
|---|---|
| `abs, fabs` | Compute the absolute value element-wise for integer, floating-point, or complex values |
| `sqrt` | Compute the square root of each element (equivalent to `arr ** 0.5`) |
| `square` | Compute the square of each element (equivalent to `arr ** 2`) |
| `exp` | Compute the exponent $e^x$ of each element |
| `log, log10, log2, log1p` | Natural logarithm (base *e*), log base 10, log base 2, and log(1 + x), respectively |
| `sign` | Compute the sign of each element: 1 (positive), 0 (zero), or –1 (negative) |
| `ceil` | Compute the ceiling of each element (i.e., the smallest integer greater than or equal to that number) |
| `floor` | Compute the floor of each element (i.e., the largest integer less than or equal to each element) |
| `rint` | Round elements to the nearest integer, preserving the `dtype` |
| `modf` | Return fractional and integral parts of array as a separate array |
| `isnan` | Return boolean array indicating whether each value is `NaN` (Not a Number) |
| `isfinite, isinf` | Return boolean array indicating whether each element is finite (non-`inf`, non-`NaN`) or infinite, respectively |
| `cos, cosh, sin, sinh, tan, tanh` | Regular and hyperbolic trigonometric functions |
| `arccos, arccosh, arcsin, arcsinh, arctan, arctanh` | Inverse trigonometric functions |
| `logical_not` | Compute truth value of `not x` element-wise (equivalent to `~arr`). |

*Table 4-4. Binary universal functions*

| Function | Description |
|---|---|
| `add` | Add corresponding elements in arrays |
| `subtract` | Subtract elements in second array from first array |
| `multiply` | Multiply array elements |
| `divide, floor_divide` | Divide or floor divide (truncating the remainder) |
| `power` | Raise elements in first array to powers indicated in second array |
| `maximum, fmax` | Element-wise maximum; `fmax` ignores `NaN` |
| `minimum, fmin` | Element-wise minimum; `fmin` ignores `NaN` |
| `mod` | Element-wise modulus (remainder of division) |
| `copysign` | Copy sign of values in second argument to values in first argument |


View
Copy
A B
0 1 2
1 3 4
2 5 6
←df2
df1
A B
0 1 2
1 3 4
2 5 6
df1
A B
0 1 2
1 3 4
df2



## flatten() 和 ravel() 的区别

数组的扁平化有两种常用的方法， flatten() 和 ravel() 。 flatten 处理后的数组是父数组的引用，因此新数组的任何变化也会改变父数组，因其未用复制的方式构建数组，内存使用效率高， ravel 通过复制的方式构建新数组。