NumPy

A library for numerical computing

ARRAY

similar to C same type array

intro

```
a = [[1, 2], 'abc',  3.14, True, None]  # all types possible
print(type(a))
print(a[0])  # element in first index
print(a[0][1])
```

```
<class 'list'>
[1, 2]
2
```

```
import numpy as np

# homogeneous list
a = np.array([0, 1, 1, 2, 3, 5, 8])  # only one type
print(type(a))
print(a)
```

```
<class 'numpy.ndarray'>
[0 1 1 2 3 5 8]
```

many

```
print([0] * 5, [1] * 5)
print([1, 2] * 5)
```

```
[0, 0, 0, 0, 0] [1, 1, 1, 1, 1]
[1, 2, 1, 2, 1, 2, 1, 2, 1, 2]
```

```
import numpy as np

print(zeros := np.zeros(10))
print(ones  := np.ones(10))
print(full  := np.full(10, 2))
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
[2 2 2 2 2 2 2 2 2 2]
```

```
print(np.tile(a, 2))
print(np.repeat(a, 3))
```

```
[1 2 1 2]
[1 1 1 2 2 2]
```

range

```
print(range(5))
print(*range(5))
```

```
range(0, 10)
0 1 2 3 4 5 6 7 8 9
```

```
import numpy as np

print(np.arange(5))
print(*np.arange(5))
```

```
[0 1 2 3 4]
```

benchmark

creating array

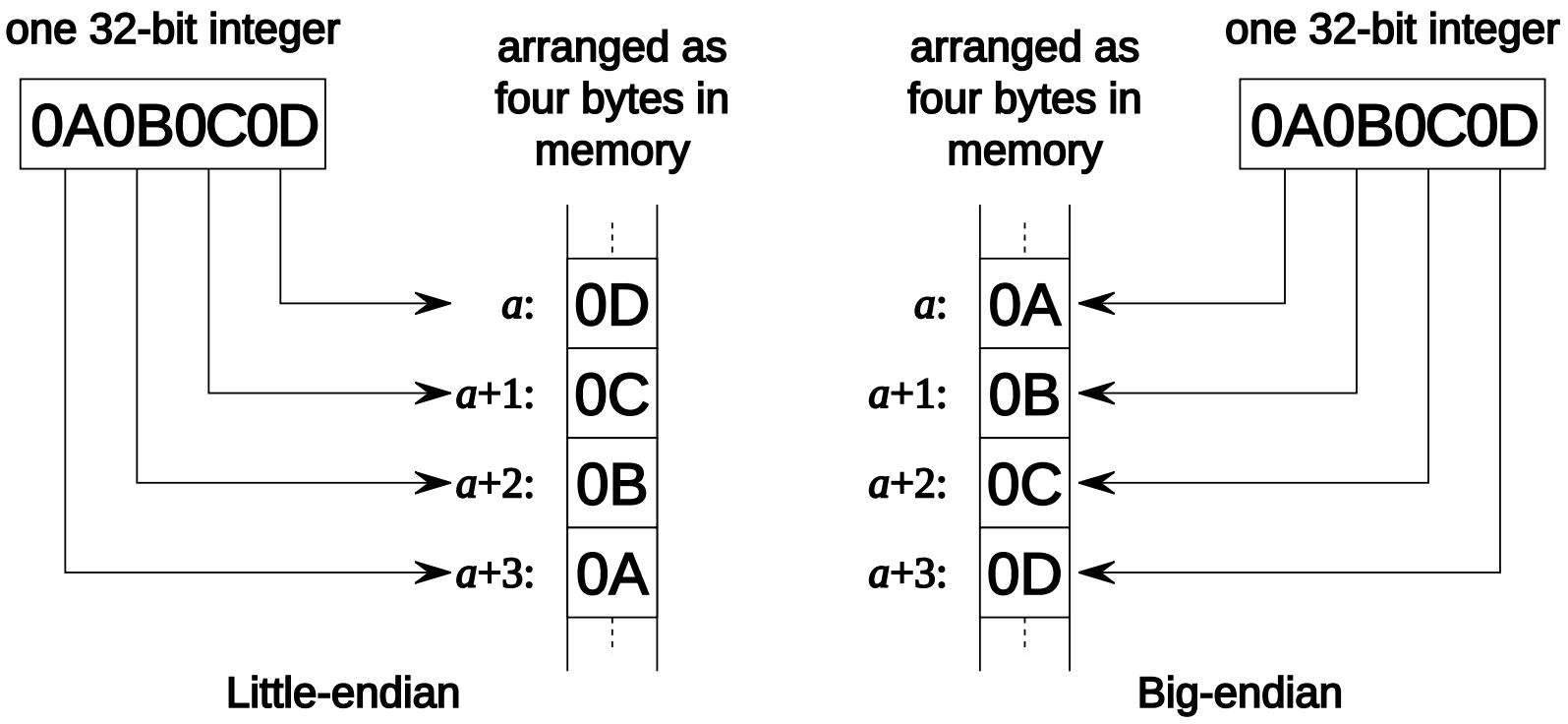
```
python -m timeit '[0] * 123456789'
python -m timeit 'list(range(123456789))'  # don't try it
```

```
2 loops, best of 5: 157 msec per loop
1 loop, best of 5: 3.39 sec per loop
```

```
python -m timeit -s 'import numpy as np' 'np.full(123456789, 0)'
python -m timeit -s 'import numpy as np' 'np.zeros(123456789)'
python -m timeit -s 'import numpy as np' 'np.arange(123456789)'
```

```
2 loops, best of 5: 102 msec per loop
20000 loops, best of 5: 13.4 usec per loop
2 loops, best of 5: 110 msec per loop
```

DTYPE


one 32-bit integer
0A0B0C0D
arranged as four bytes in memory
a: 0D
a+1: 0C
a+2: 0B
a+3: 0A
Little-endian
arranged as four bytes in memory
one 32-bit integer
0A0B0C0D
a: 0A
a+1: 0B
a+2: 0C
a+3: 0D
Big-endian

types

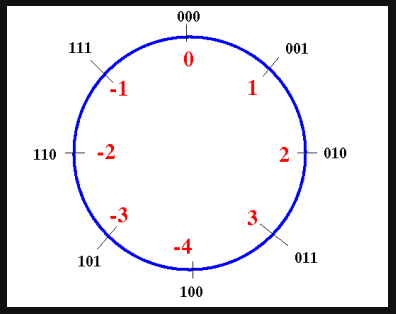
```python
import numpy as np

print(z := np.zeros(10), z.dtype)   # default is float64
print(np.zeros(10, dtype=np.int8))  # change to int8
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.] float64
[0 0 0 0 0 0 0 0 0 0]
```

```python
print(a := np.arange(10), a.dtype)       # default is int64
print(np.arange(10, dtype=np.float16))  # change to float16
```

```
[0 1 2 3 4 5 6 7 8 9] int64
[0. 1. 2. 3. 4. 5. 6. 7. 8. 9.]
```

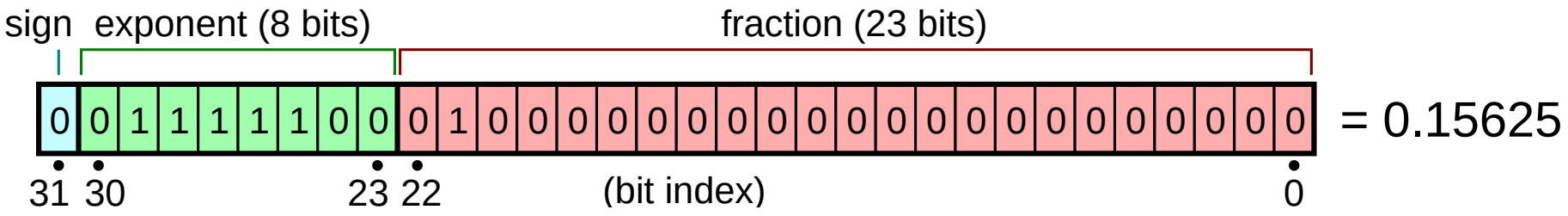
integer




```
import numpy as np

print(np.int8, np.byte)  # same
print(i8  := np.int8(),  i8.nbytes, 2**8)
print(i16 := np.int16(), i16.nbytes, 2**16)
print(i32 := np.int32(), i32.nbytes, 2**32)
print(i64 := np.int64(), i64.nbytes, 2**64)
```

```
<class 'numpy.int8'> <class 'numpy.int8'>
0 1 256
0 2 65536
0 4 4294967296
0 8 18446744073709551616
```

float




```
import numpy as np

print(np.float16)
print(np.float32)
print(np.float64)
print(np.float128, np.longdouble)
```

```
<class 'numpy.float16'>
<class 'numpy.float32'>
<class 'numpy.float64'>
<class 'numpy.longdouble'> <class 'numpy.longdouble'>
```

Not A Number

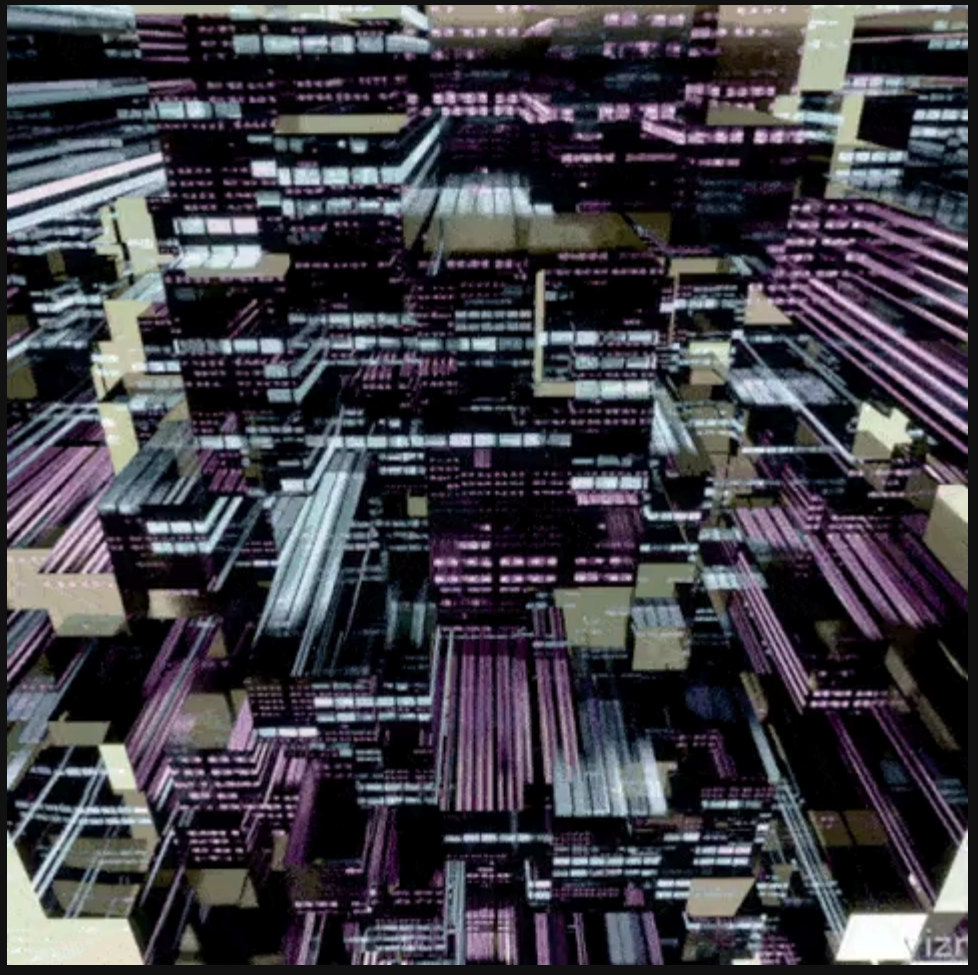
```
import numpy as np

print(np.nan)
print(np.nan + np.nan)
print(np.nan - np.nan)
print(np.nan * np.nan)
print(np.nan / np.nan)
```

```
nan
nan
nan
nan
nan
```

summary

python	numpy	bit
bool	np.bool	8
int	np.int	8, 16, 32, 64
float	np.float	16, 32, 64, 128

DIMENSIONS



ndarray

np.array creates nd array!

```
import numpy as np

print(type(np.array([])))
print(np.ndarray)
# n is number, d is dimension
```

```
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
```

info

```
m = np.array([[0, 1], [2, 3]])

# len is for one dimention
print(f'{m.size=} {len(m)=}')
```

```
m.size=4 len(m)=2
```

```
# other info
print(f'{m.ndim=}')
print(f'{m.shape=}')
print(f'{m.itemsize=}')
print(f'{m.nbytes=}')
```

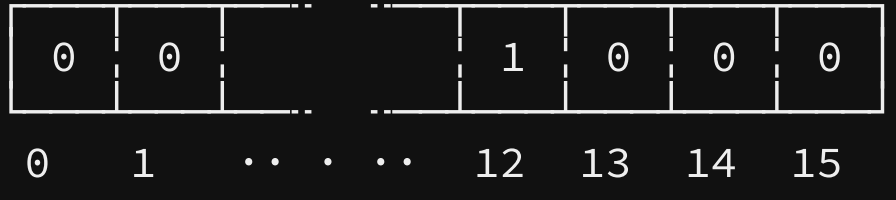
```
m.ndim=2
m.shape=(2, 2)
m.itemsize=8
m.nbytes=32
```

big endian layout (C order)

```
m.ndim  = 2
m.shape = (3, 3)

len(m) = 3
```

0	1	2
3	4	5
6	7	8

```
np.int16.itemsize = 2 bytes
```




```
m.nbytes = 3 * 3 * 2
         = 18
```

memory 1D!

p+00	0x00	0x00	m[0, 0]
p+02	0x00	0x01	m[0, 1]
p+04	0x00	0x02	m[0, 2]
p+06	0x00	0x03	m[1, 0]
p+08	0x00	0x04	m[1, 1]
p+10	0x00	0x05	m[1, 2]
p+12	0x00	0x06	m[2, 0]
p+14	0x00	0x07	m[2, 1]
p+16	0x00	0x08	m[2, 2]

strides of array

```
import numpy as np

m = np.array([
    [0, 1],
    [2, 3],
], dtype=np.int8)  # 8 bit for simplitcity

print((m.shape[1] * m.itemsize, m.itemsize))
print(m.strides)  # ne of bytes to step in each dimension

print(m.tobytes())
print(m[1, 1].tobytes())
```

```
(2, 1)
(2, 1)
b'\x00\x01\x02\x03'
b'\x03'
```

resize

```
import numpy as np

a = np.arange(1, 5)
print(a)
a.resize(2, 2)  # inplace operation
print(a)
```

```
[1 2 3 4]
[[1 2]
 [3 4]]
```

reshape

operation only affect structure not data.

```
import numpy as np

a = np.arange(12)
print(a)
b = a.reshape(2, 6)
b[0, 0] = 100
print(b)
print(np.may_share_memory(a, b))
print(a.reshape(3, 4))
print(a.reshape(12))
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11]
[[100   1   2   3   4   5]
 [  6   7   8   9  10  11]]
True
[[100   1   2   3]
 [  4   5   6   7]
 [  8   9  10  11]]
[100   1   2   3   4   5   6   7   8   9  10  11]
```

flatten

```python
import numpy as np

c = np.arange(8).reshape(2, 2, 2)
print(c)
print(c.flatten())  # creates copy
print(c.flat)  # iterator
print(*c.flat)
print(np.may_share_memory(c, c.flat))
```

```
[[[0 1]
  [2 3]]

 [[4 5]
  [6 7]]]
[0 1 2 3 4 5 6 7]
<numpy.flatiter object at 0x64d7afd38e50>
0 1 2 3 4 5 6 7
[0 1 2 3 4 5 6 7]
True
```

ravel

same as flatten but return view if possible

```
import numpy as np

a = np.arange(8).reshape(2, 2, 2)
b = a.ravel()
print(b)
b[0] = 100  # b is view
print(a)
```

```
[0 1 2 3 4 5 6 7]
[[[100   1]
  [  2   3]]

 [[  4   5]
  [  6   7]]]
```

OPERATION RULES

add operator

```
a = [1, 2, 3]
b = [4, 5, 6]

# concat operation
print(a + b)
```

```
[1, 2, 3, 4, 5, 6]
```

```
import numpy as np

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

print(a + b)
print(np.concatenate((a, b)))
```

```
[5 7 9]
[1 2 3 4 5 6]
```

ordering

```
a = [1, 2, 3]
b = [1, 2, 5]

print(a < b)  # lexicograhpic ordering
```

```
True
```

```
import numpy as np

print(np.array(a) < np.array(b))
```

```
True
[False False  True]
```

slice

```
a = list(range(5))
b = a[1:3]
b[0] = 100
print(a, b)
```

```
[0, 1, 2, 3, 4] [100, 2]
```

```
import numpy as np

a = np.arange(5)
b = a[1:3]
b[0] = 100
print(a, b)

# slice isn't copy
c = a[1:3].copy()
c[0] = 0
print(a, c)
```

```
[  0 100   2   3   4] [100   2]
[  0 100   2   3   4] [0 2]
```

arithmetic

```
import numpy as np

# scalar
print(np.arange(5) + 2)
print(np.arange(5) % 2)
print(np.arange(5) > 2)
```

```
[2 3 4 5 6]
[0 1 0 1 0]
[False False False  True  True]
```

```
# vector
a = np.array([ 2, 4, 6 ])
b = np.array([ 1, 2, 4 ])
print(a + b)
print(a - b)
print(a / b)
print(b ** a)
```

```
[ 3  6 10]
[1 2 2]
[2.  2.  1.5]
[   1   16 4096]
```

simple function

$$y = 2x + 1$$

```
x = 1
y = 2*x + 1
print(y)
```

```
3
```

```
f = lambda x: 2*x + 1
print(f(1))

print(list(map(f, [1, 2, 3])))
print(list(map(f, range(5))))
```

```
3
[3, 5, 7]
[1, 3, 5, 7, 9]
```

```
f = lambda x: 2*x + 1

import numpy as np
print(f(np.arange(5)))
```

```
[1 3 5 7 9]
```

benchmark

```
python -m timeit '[ i**2 for i in range(100) ]'
python -m timeit '[ i**2 for i in range(10000) ]'
python -m timeit '[ i**2 for i in range(1000000) ]'
```

```
100000 loops, best of 5: 3.87 usec per loop
500 loops, best of 5: 438 usec per loop
5 loops, best of 5: 58.3 msec per loop
```

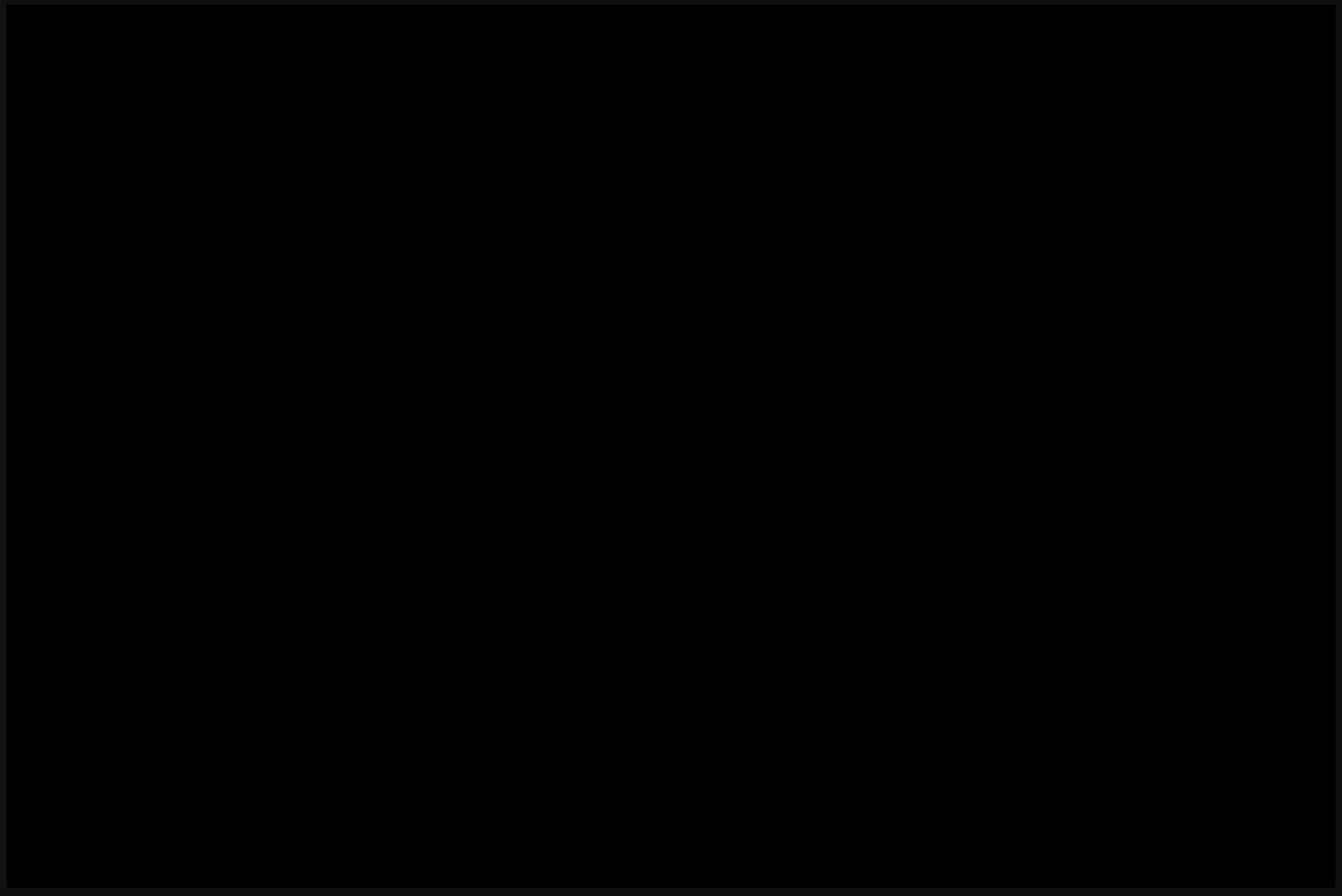
```
python -m timeit -s 'import numpy as np' \
'[ i**2 for i in np.arange(1000000) ]'
python -m timeit -s 'import numpy as np' 'np.arange(1000000)**2'
```

```
5 loops, best of 5: 79.7 msec per loop
100 loops, best of 5: 2.2 msec per loop
```

summary

1. only permittied for same shapes
2. mathmatical operations applys element wise
3. reduce operation (mean, std, skew, sum) apply to whole array.
4. missing values propogate unless explicitly ignored (nanmean, nansum)

PLOT



package

```
pip install matplotlib
```

alternative installation using conda, active state or system packages

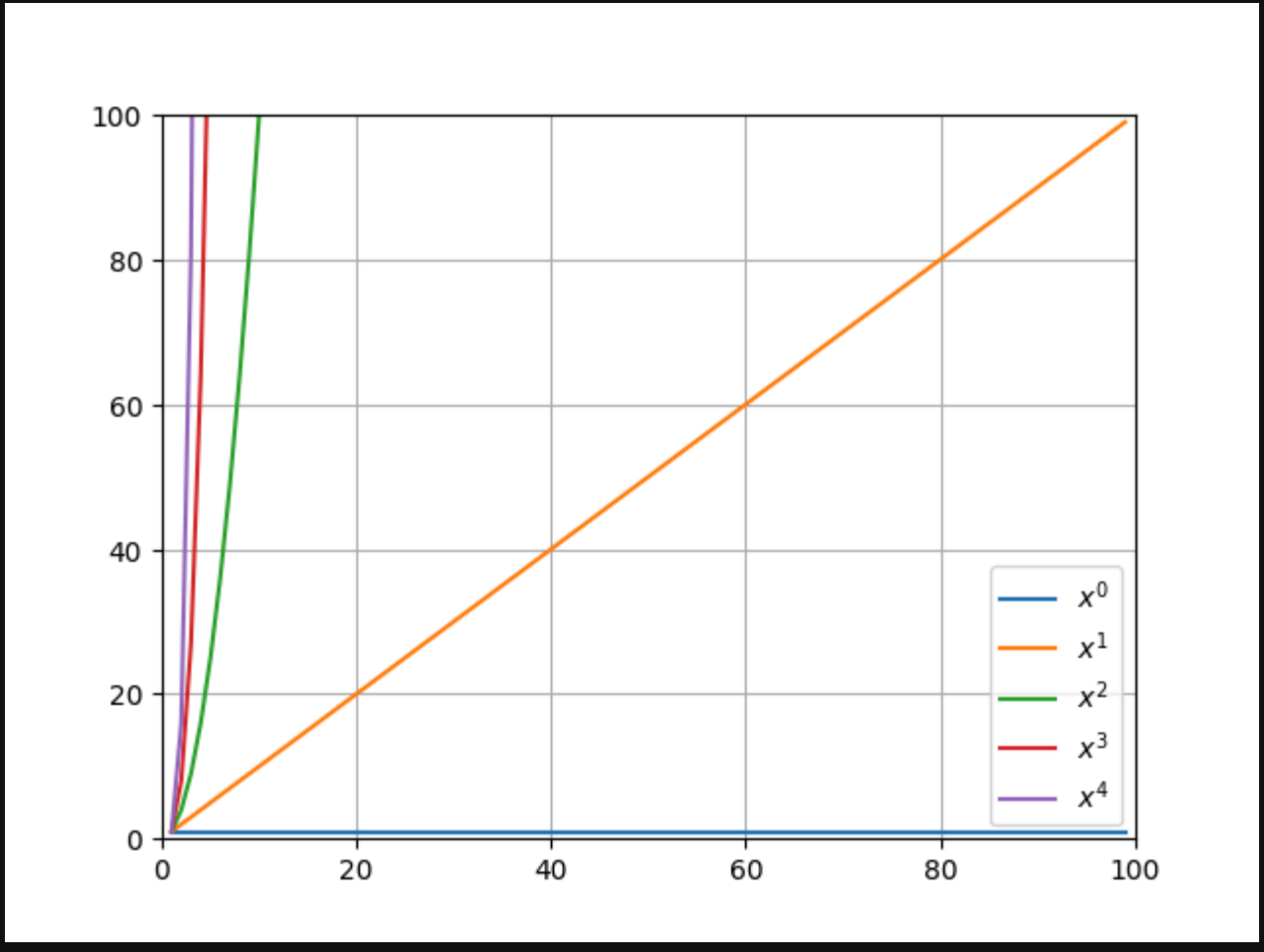
TIPS use virtual enviroment

```
python -m venv venv
source venv/bin/activate
```

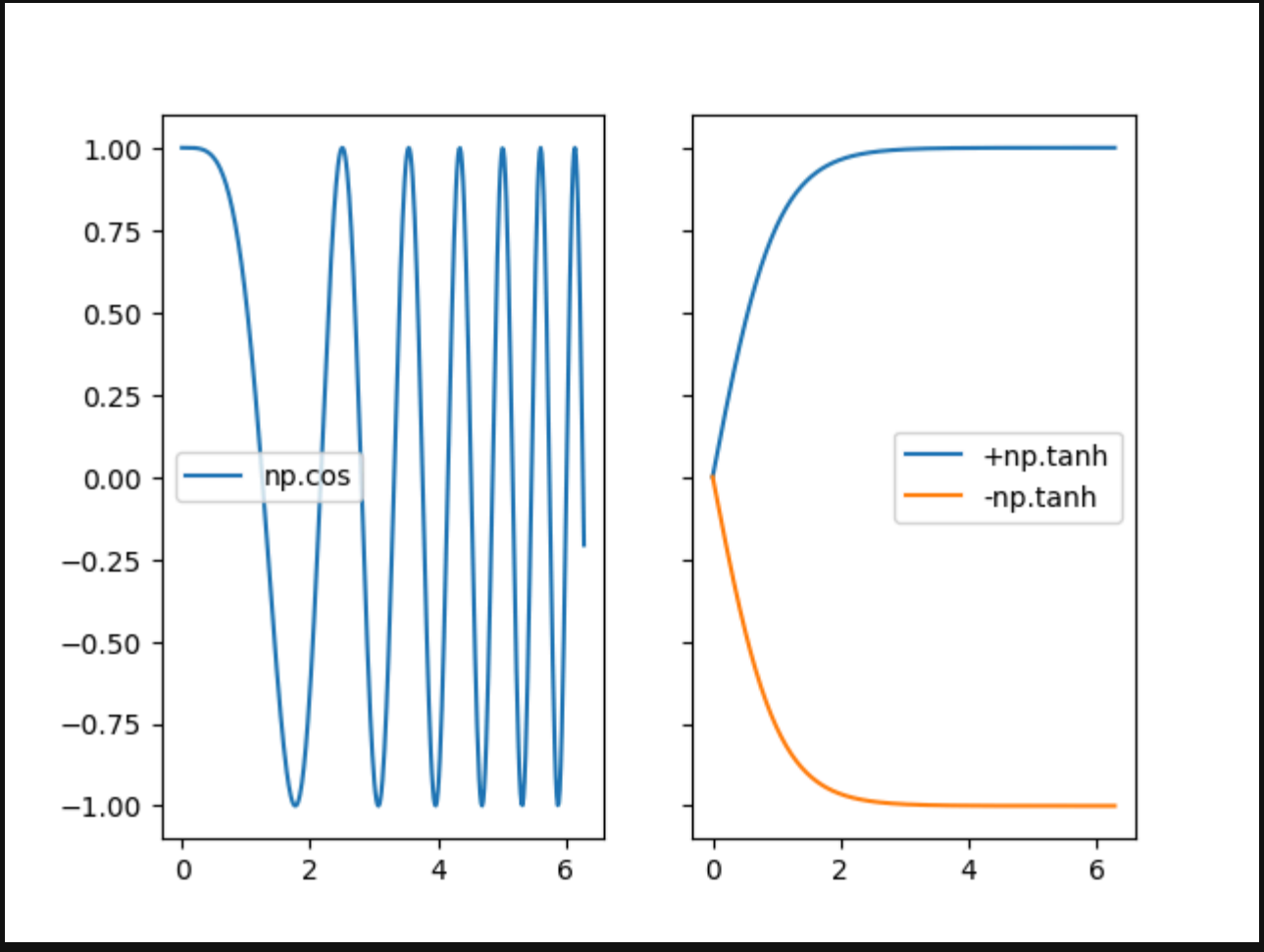
syntax

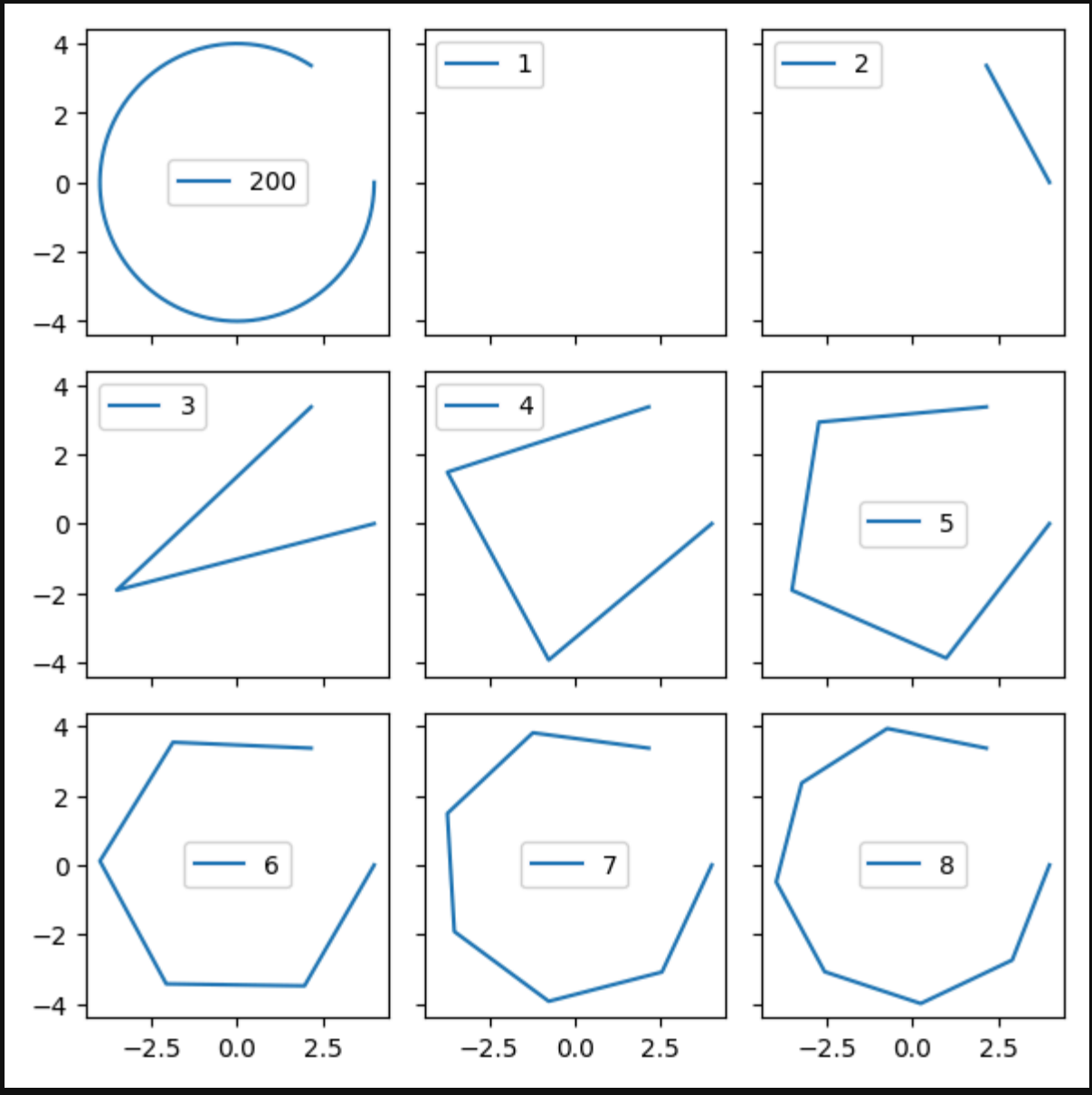
```
import numpy as np
import matplotlib.pyplot as plt

x = [1, 3, 4, 5]
y = [3, 6, 9, 12]

fig, ax = plt.subplots()
ax.plot(x, y)
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_title('My plot')
plt.show()
```

plt.plot



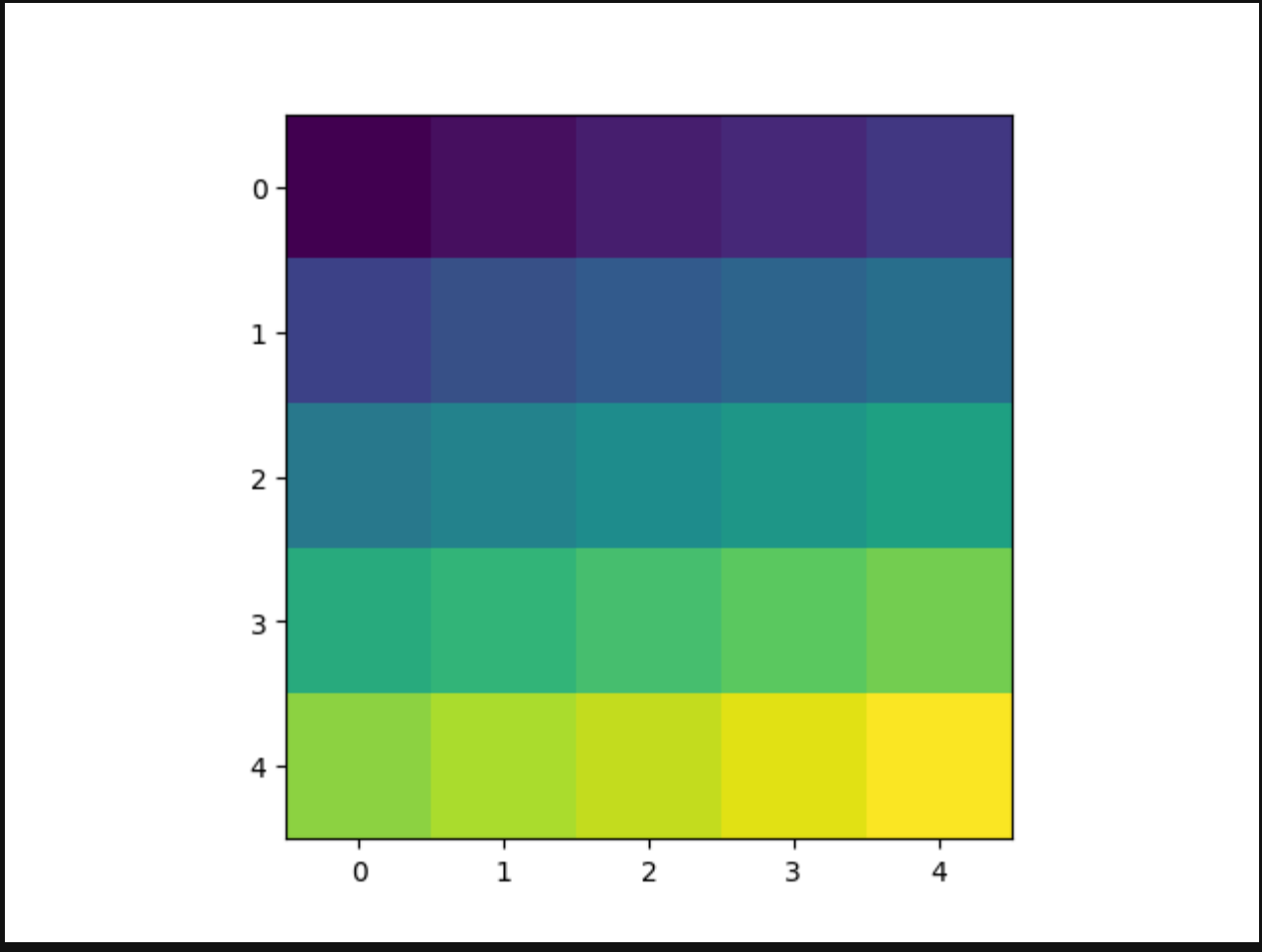
plt.subplots()




200
1
2
3
4
5
6
7
8
4
2
0
−2
−4
−2.5
0.0
2.5

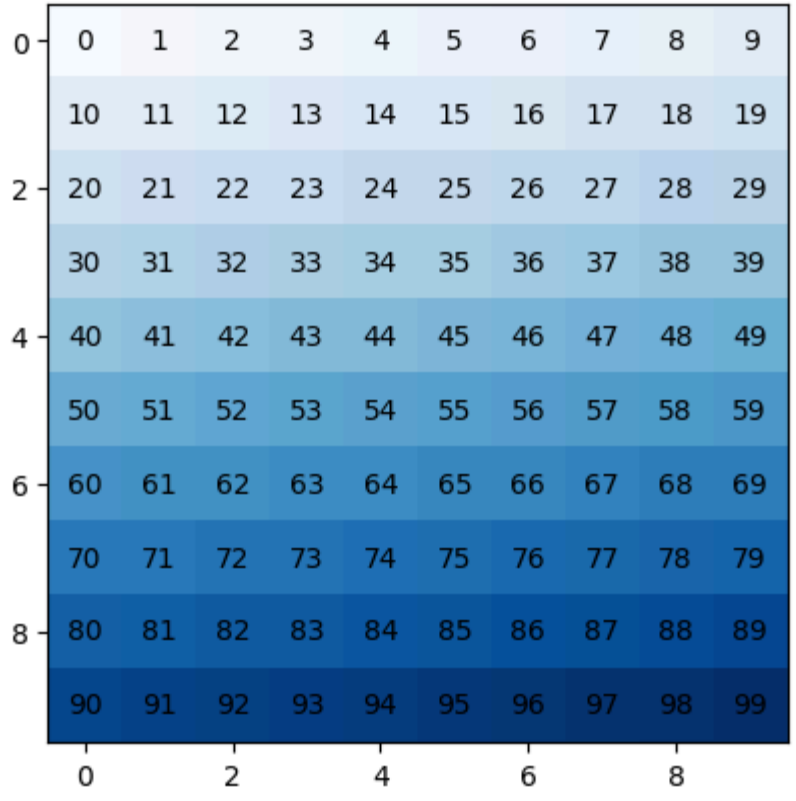

IMSHOW

plt.imshow()



```
import numpy as np
import matplotlib.pyplot as plt

m = np.arange(25).reshape(5, 5)
fig, ax = plt.subplots()
ax.imshow(m)
plt.show()
```

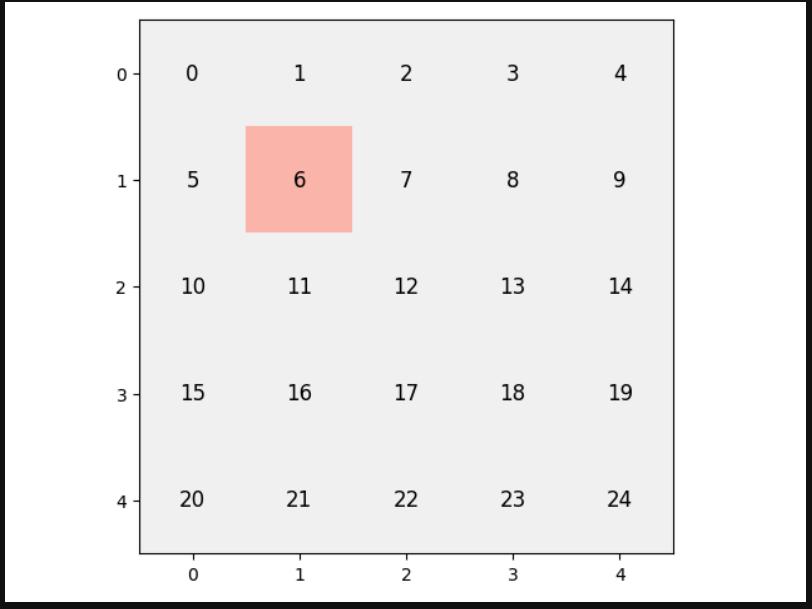
plt.text()


0 1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59
60 61 62 63 64 65 66 67 68 69
70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89
90 91 92 93 94 95 96 97 98 99
0
2
4
6
8

```
import numpy as np
import matplotlib.pyplot as plt

m = np.arange(100).reshape(10, 10)

fig, ax = plt.subplots()
ax.imshow(m, cmap='Blues')
for i in range(100):
    y, x = divmod(i, 10)
    ax.text(x, y, i, ha='center', va='center')

plt.show()
```

problem

```
def matrix_plot(m):
    NotImplementedError

m = np.zeros(25, dtype=np.int8).reshape(5, 5)
m[1, 1] = 1

fig = matrix_plot(m)
plt.show()
```



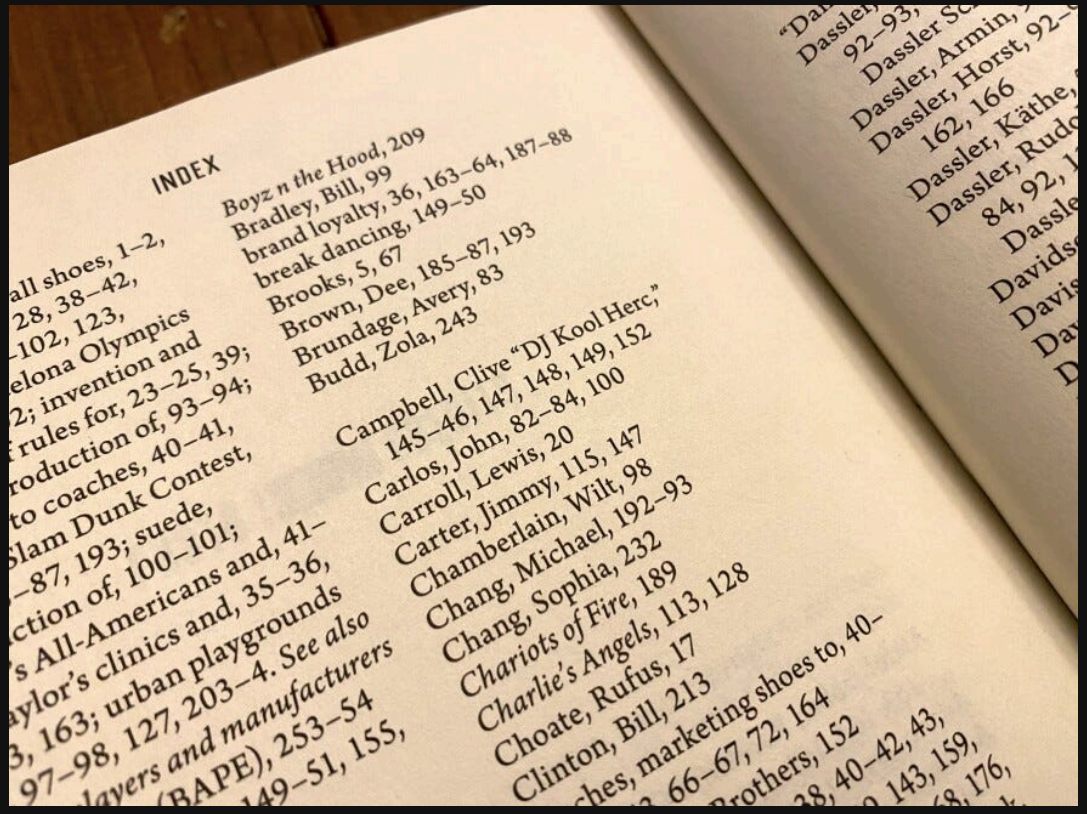
solution

```
def matrix_plot(m):
    fig, ax = plt.subplots()
    ax.imshow(m, cmap='Pastel1_r')

    shape = m.shape
    for i in range(m.size):
        y, x = divmod(i, shape[0])
        ax.text(x, y, i, size=12, ha='center', va='center')

    return fig
```

summary

- designed like MATLAB
- access to rendering back-ends

INDEXING



INDEX

all shoes, 1–2,
28, 38–42,
–102, 123,
elona Olympics
2; invention and
f rules for, 23–25, 39;
roduction of, 93–94;
to coaches, 40–41,
Slam Dunk Contest,
–87, 193; suede,
ction of, 100–101;
s All-Americans and, 41–
aylor's clinics and, 35–36,
3, 163; urban playgrounds
97–98, 127, 203–4. *See also*
layers and manufacturers
(BAPE), 253–54
149–51, 155,

Boyz n the Hood, 209
Bradley, Bill, 99
brand loyalty, 36, 163–64, 187–88
break dancing, 149–50
Brooks, 5, 67
Brown, Dee, 185–87, 193
Brundage, Avery, 83
Budd, Zola, 243

Campbell, Clive "DJ Kool Herc,"
145–46, 147, 148, 149, 152
Carlos, John, 82–84, 100
Carroll, Lewis, 20
Carter, Jimmy, 115, 147
Chamberlain, Wilt, 98
Chang, Michael, 192–93
Chang, Sophia, 232
Chariots of Fire, 189
Charlie's Angels, 113, 128
Choate, Rufus, 17
Clinton, Bill, 213
ches, marketing shoes to, 40–
66–67, 72, 164
Brothers, 152
28; 40–42, 43,
9, 143, 159,
8, 176,

"Da
Dassler
92–93,
Dassler S
Dassler, Armin,
Dassler, Horst, 92–
162, 166
Dassler, Käthe,
Dassler, Rudo
84, 92, 1
Dassle
Davids
Davis
Da
D

index

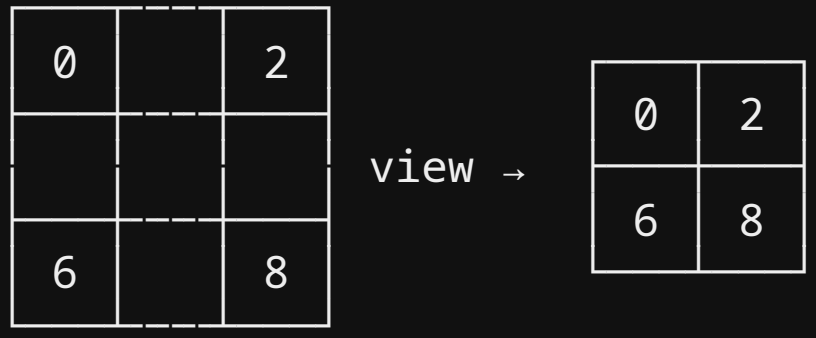
```
import numpy as np

c = np.arange(27).reshape(3, 3, 3)

# python way
print(c[0][0][0])
print(c[0][1][2])

# numpy way
print(c[0, 0, 0])
ix = 0, 1, 2
print(c[ix])
```

```
0
5
0
5
```

slice [start : stop : stride]

```
import numpy as np

m = np.arange(9).reshape(3, 3)
print(s := m[::2, ::2])
print(np.may_share_memory(m, s))
```

```
[[0 2]
 [6 8]]
True
```

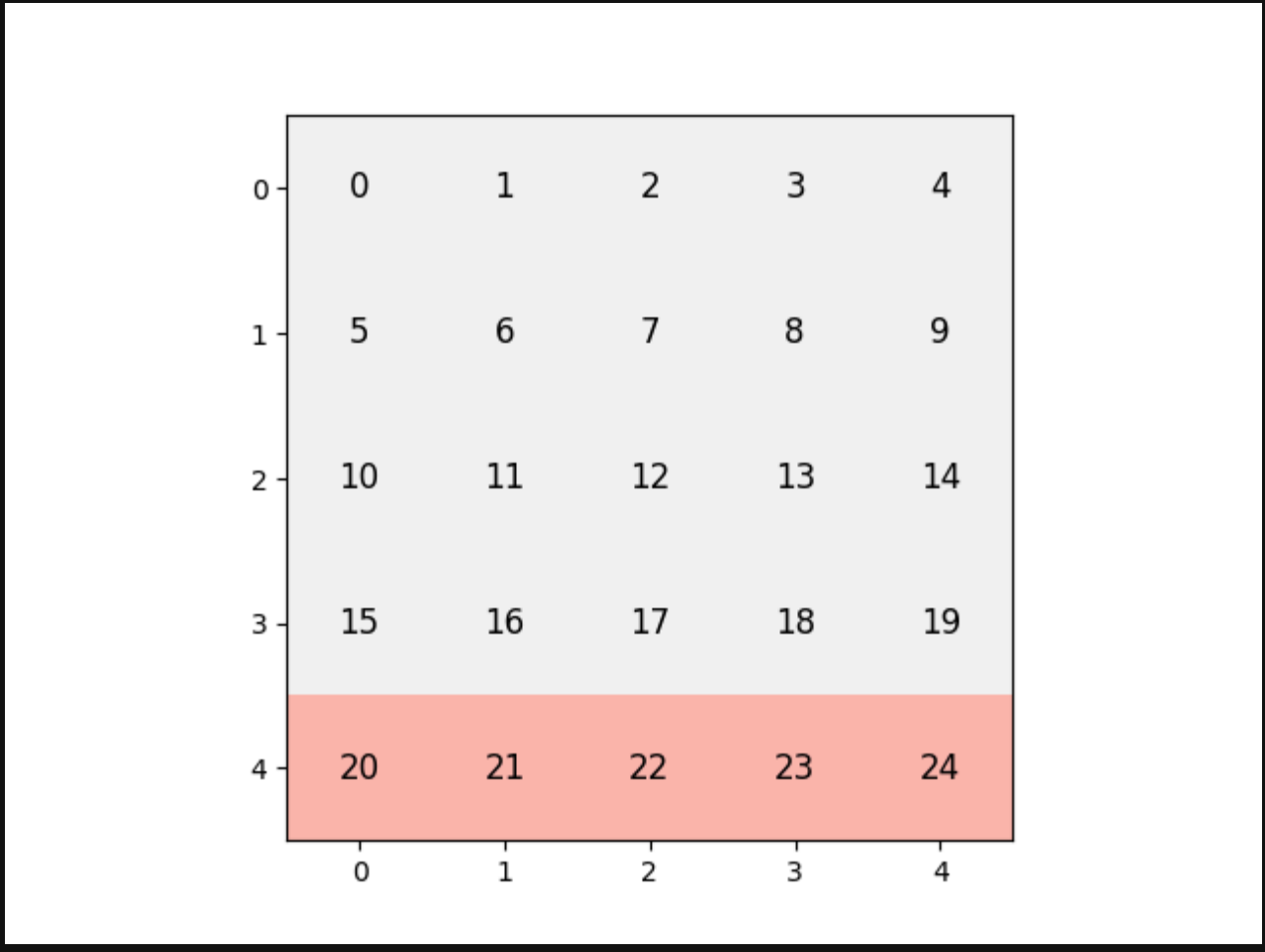


ellipsis...

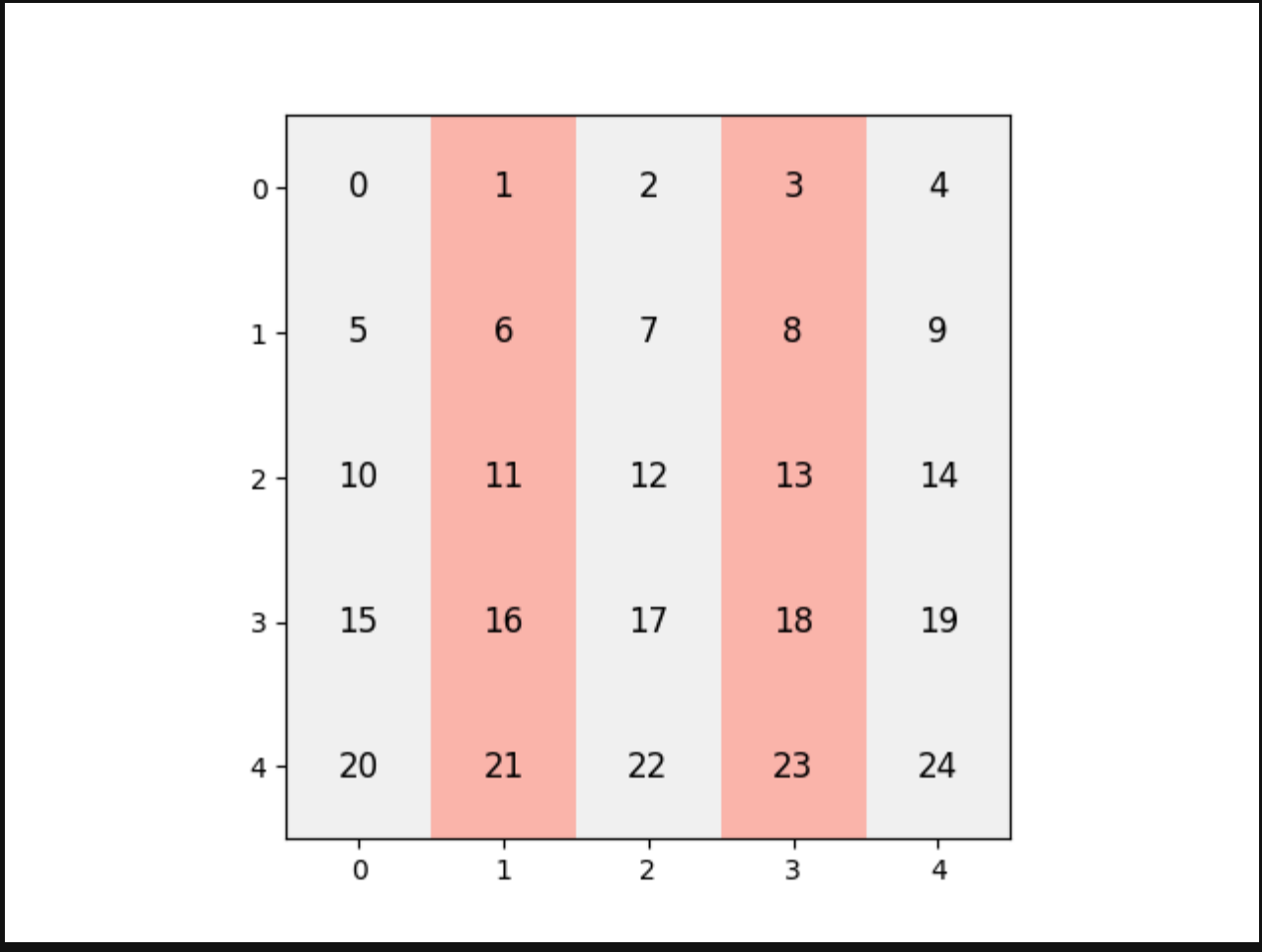
```
import numpy as np

c = np.arange(8).reshape(2, 2, 2)
print(c[1, :, :])
print(c[1, ...])  # same with ellipsis
print(c[..., 0])  # can be done at front
```

```
[[4 5]
 [6 7]]
[[4 5]
 [6 7]]
[[0 2]
 [4 6]]
```

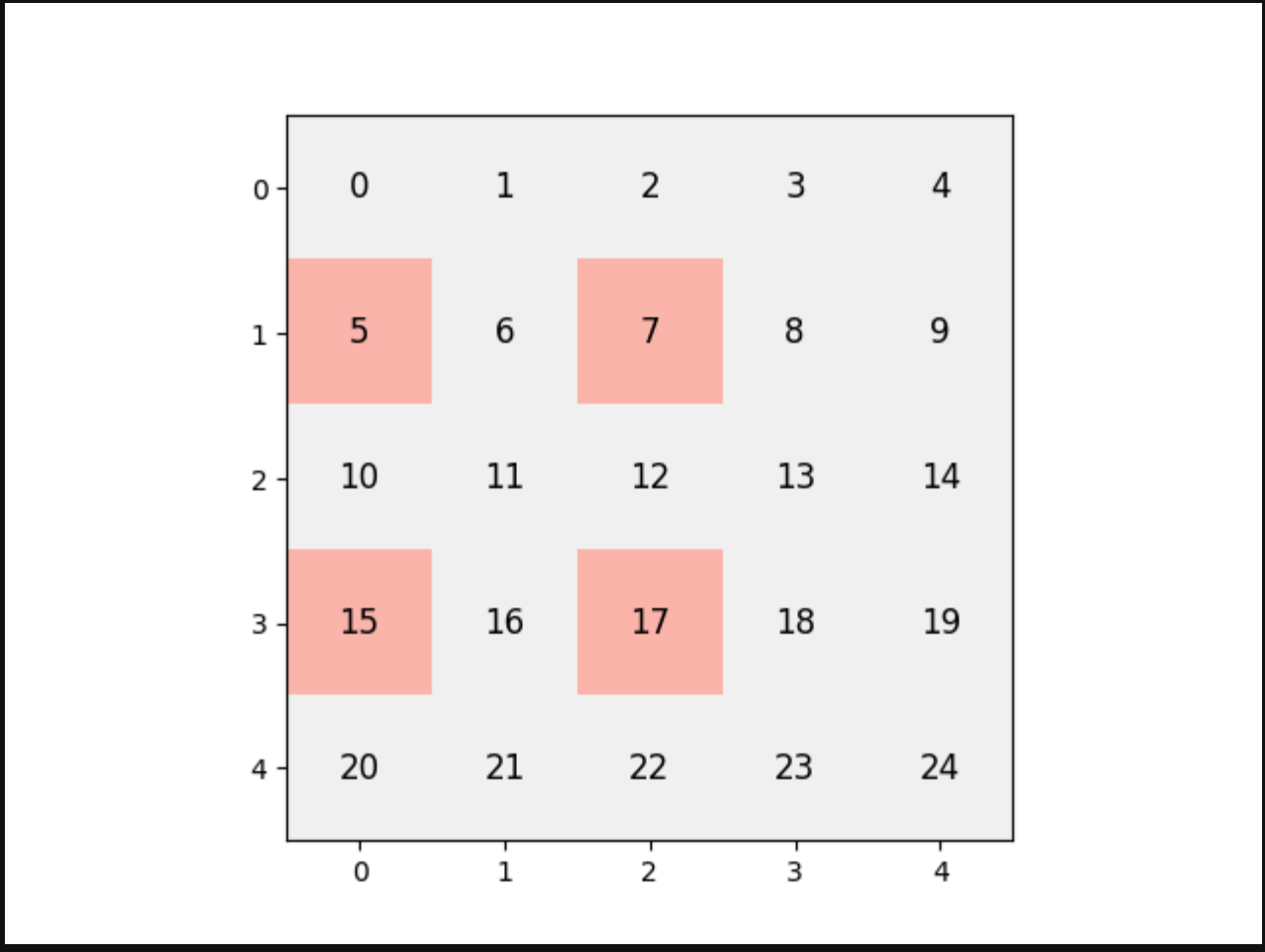
problem 1


0
1
2
3
4
0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
15 16 17 18 19
20 21 22 23 24
0
1
2
3
4

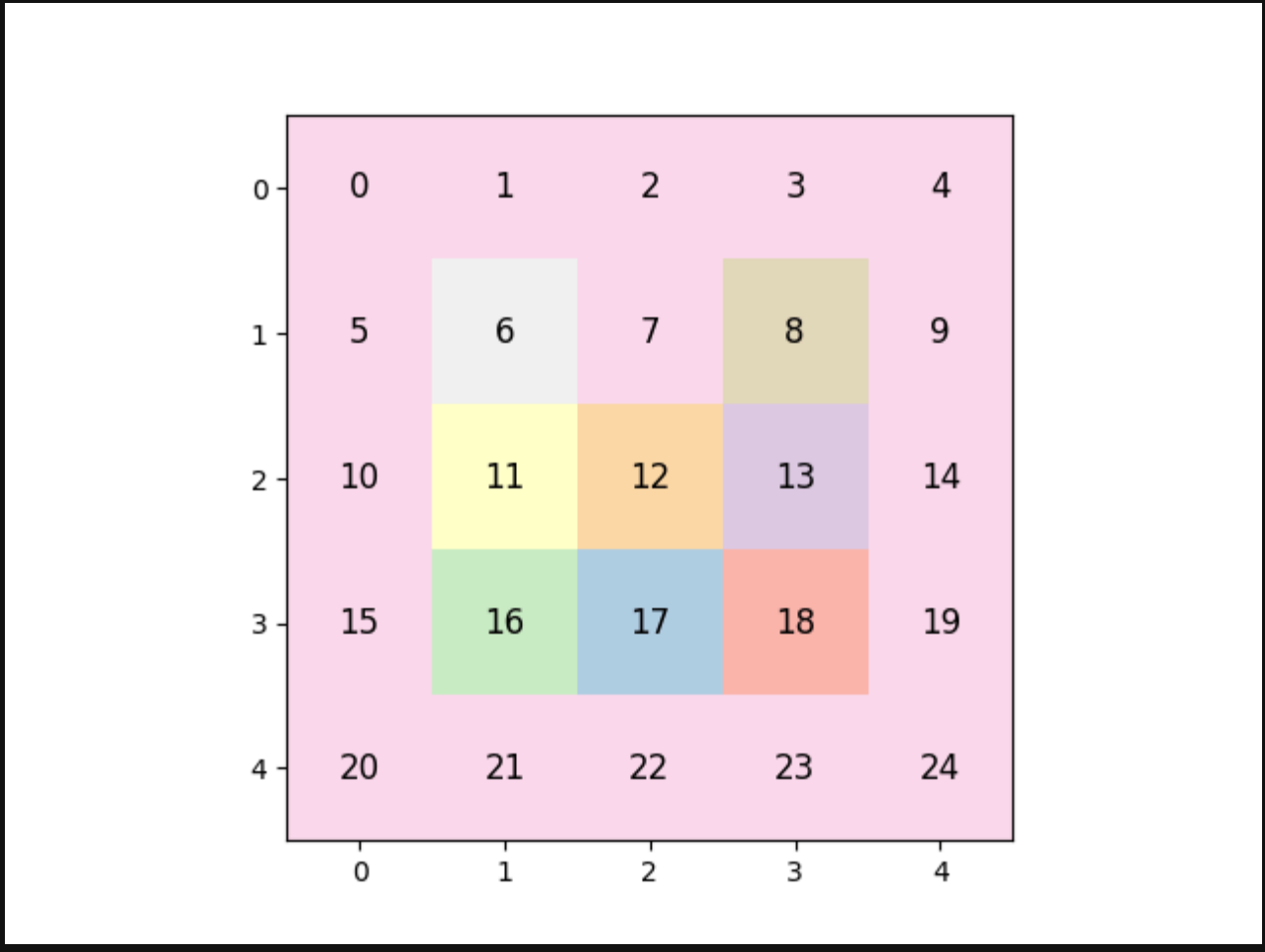
problem 2



problem 3



problem 4



IMAGE

plt.imread()

```
import matplotlib.pyplot as plt

img = plt.imread('./np-img/parrots.png')
fig, ax = plt.subplots(layout='tight')
ax.imshow(img)
plt.show()
```

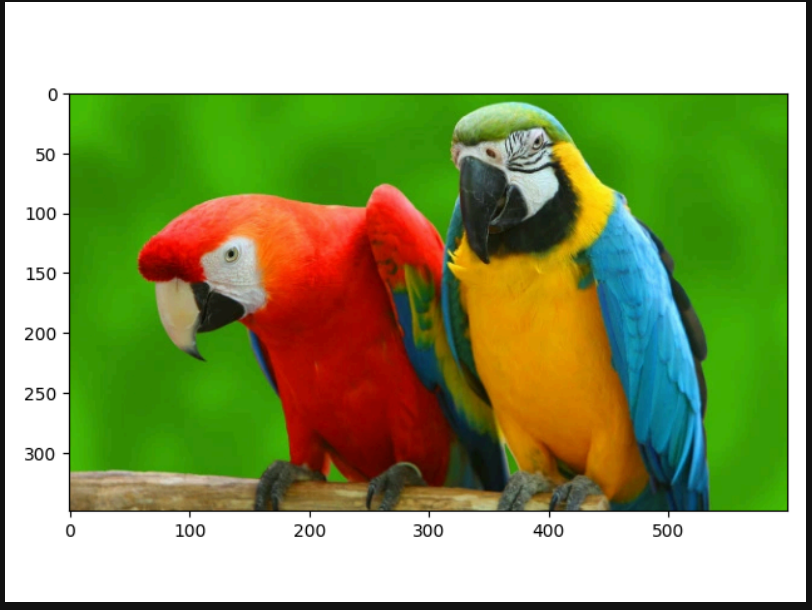


image-data

```python
import matplotlib.pyplot as plt

img = plt.imread('./np-img/parrots.png')
print(type(img))
print(img.shape)  # dimension (height, width, channels)
print(img.dtype)  # float32
print(img.itemsize)  # in bytes
print(img)
```

```
<class 'numpy.ndarray'>
(349, 600, 3)
float32
4
[[[0.27058825 0.7372549  0.02745098]
  [0.27058825 0.7372549  0.02745098]
  [0.27058825 0.7372549  0.02745098]
  ...
  [0.23529412 0.6156863  0.02352941]
  [0.23529412 0.6156863  0.02352941]
  [0.23529412 0.6156863  0.02352941]]

 [[0.27058825 0.7372549  0.02745098]
  [0.27058825 0.7372549  0.02745098]
```

```
 [0.27058825 0.7372549  0.02745098]
 ...
 [0.23921569 0.61960787 0.02745098]
 [0.23921569 0.61960787 0.02745098]
 [0.23921569 0.61960787 0.02745098]]

[[0.27058825 0.7372549  0.02745098]
 [0.27058825 0.7372549  0.02745098]
 [0.27058825 0.7372549  0.02745098]
 ...
 [0.23529412 0.6156863  0.02352941]
 [0.23529412 0.6156863  0.02352941]
 [0.23529412 0.6156863  0.02352941]]

...

[[0.4862745  0.4        0.22352941]
 [0.4862745  0.4        0.23137255]
 [0.52156866 0.42745098 0.27058825]
 ...
 [0.2784314  0.73333335 0.02745098]
 [0.2784314  0.73333335 0.02745098]
 [0.2784314  0.73333335 0.02745098]]

[[0.46666667 0.3764706  0.2       ]
 [0.45490196 0.36862746 0.2       ]
 [0.4862745  0.39607844 0.23921569]
```

```
  ...
  [0.2784314  0.73333335 0.02745098]
  [0.2784314  0.73333335 0.02745098]
  [0.2784314  0.73333335 0.02745098]]

 [[0.4745098  0.3882353  0.21176471]
  [0.44705883 0.35686275 0.19215687]
  [0.45882353 0.36862746 0.21176471]
  ...
  [0.2784314  0.73333335 0.02745098]
  [0.2784314  0.73333335 0.02745098]
  [0.2784314  0.73333335 0.02745098]]]
```

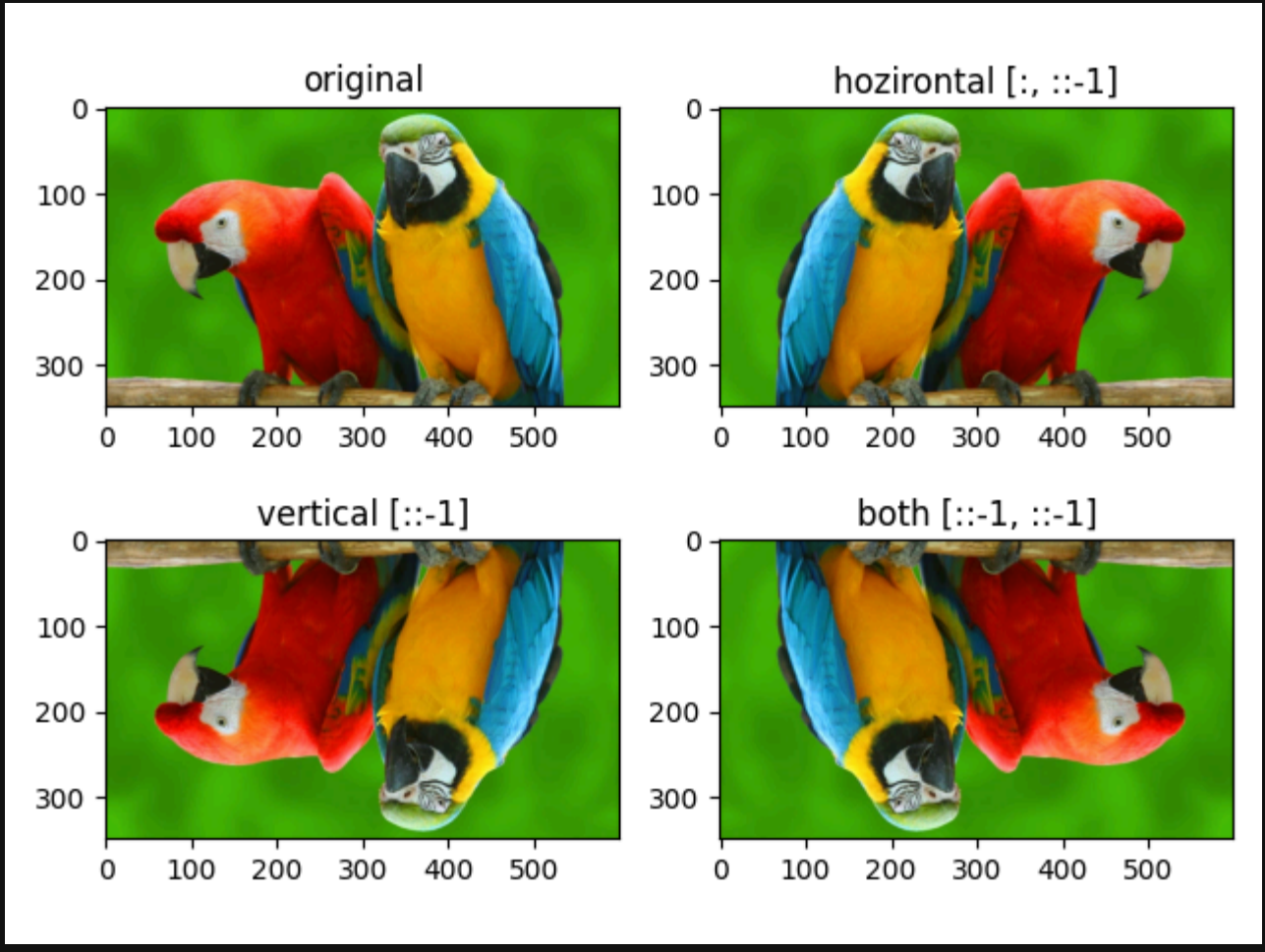
channels

colors

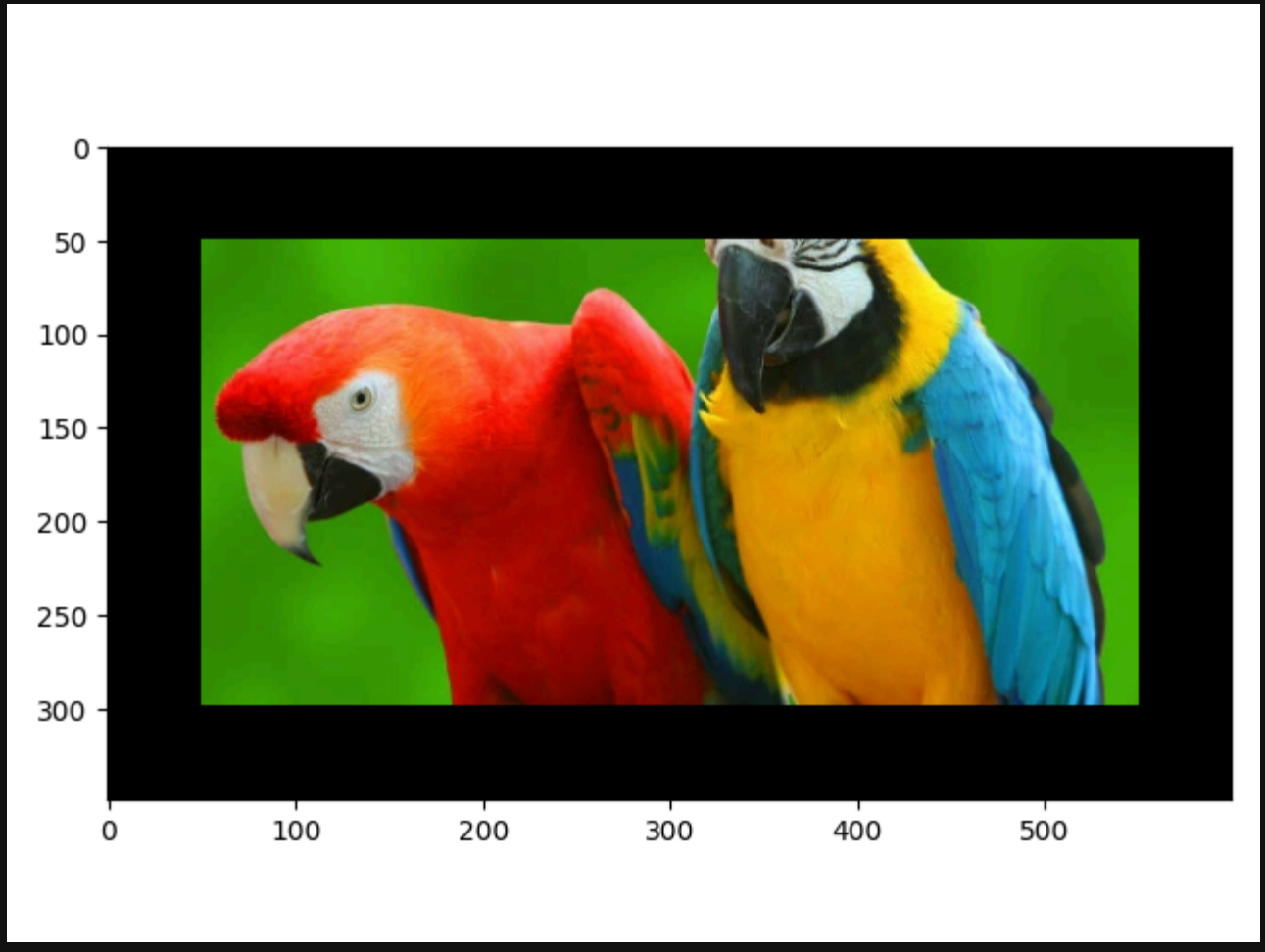
```
import matplotlib.pyplot as plt

img = plt.imread('./np-img/parrots.webp')
fig, ax = plt.subplots(2, 2, layout='tight')

ax[0, 0].imshow(img)
ax[0, 1].imshow(img[..., 0], cmap=plt.cm.Reds)
ax[1, 0].imshow(img[..., 1], cmap=plt.cm.Greens)
ax[1, 1].imshow(img[..., 2], cmap=plt.cm.Blues)
plt.show()
```

crop

flip


original
hozirontal [:, ::-1]
vertical [::-1]
both [::-1, ::-1]
0
100
200
300
400
500

write



BROADCASTING

operation of different dimension array

type I

```
import numpy as np

a = np.arange(6).reshape(2, 3)
b = np.ones((2, 3), dtype=np.int8)
print(a)
print(b)
print(a + b)
```

```
[[0 1 2]
 [3 4 5]]
[[1 1 1]
 [1 1 1]]
[[1 2 3]
 [4 5 6]]
```

type II

```
import numpy as np

a = np.ones((3, 3), dtype=np.int8)
b = np.arange(3)
print(a)
print(b)
print(a + b)
```

```
[[1 1 1]
 [1 1 1]
 [1 1 1]]
[0 1 2]
[[1 2 3]
 [1 2 3]
 [1 2 3]]
```

type III

```python
import numpy as np

a = np.ones((3, 1), dtype=np.int8)
b = np.arange(3)
print(a)
print(b)
print(a + b)
```

```
[[1]
 [1]
 [1]]
[0 1 2]
[[1 2 3]
 [1 2 3]
 [1 2 3]]
```

summary

1. Prepends ones to smaller array
2. Dimentions of size 1 are repeted without copying

MASK

bits

```
a = int('1010', 2)
mask = int('0011', 2)
print(bin(a & mask))
```

```
0b10
```

boolean

```python
import numpy as np

a = np.arange(5)
print(a)

mask = a > 2
print(mask)
print(a[a > 2])
```

```
[0 1 2 3 4]
[False False False  True  True]
[3 4]
```

chain

```
import numpy as np

a = np.arange(10)
print(a)
print(a[a < 5])

try:
    2 < a < 8
    # operator only work with single value
except ValueError as e:
    print(e)

# use bitwise operator &, |, ~, ^
print(a[(2 < a) & (a < 8)])
```

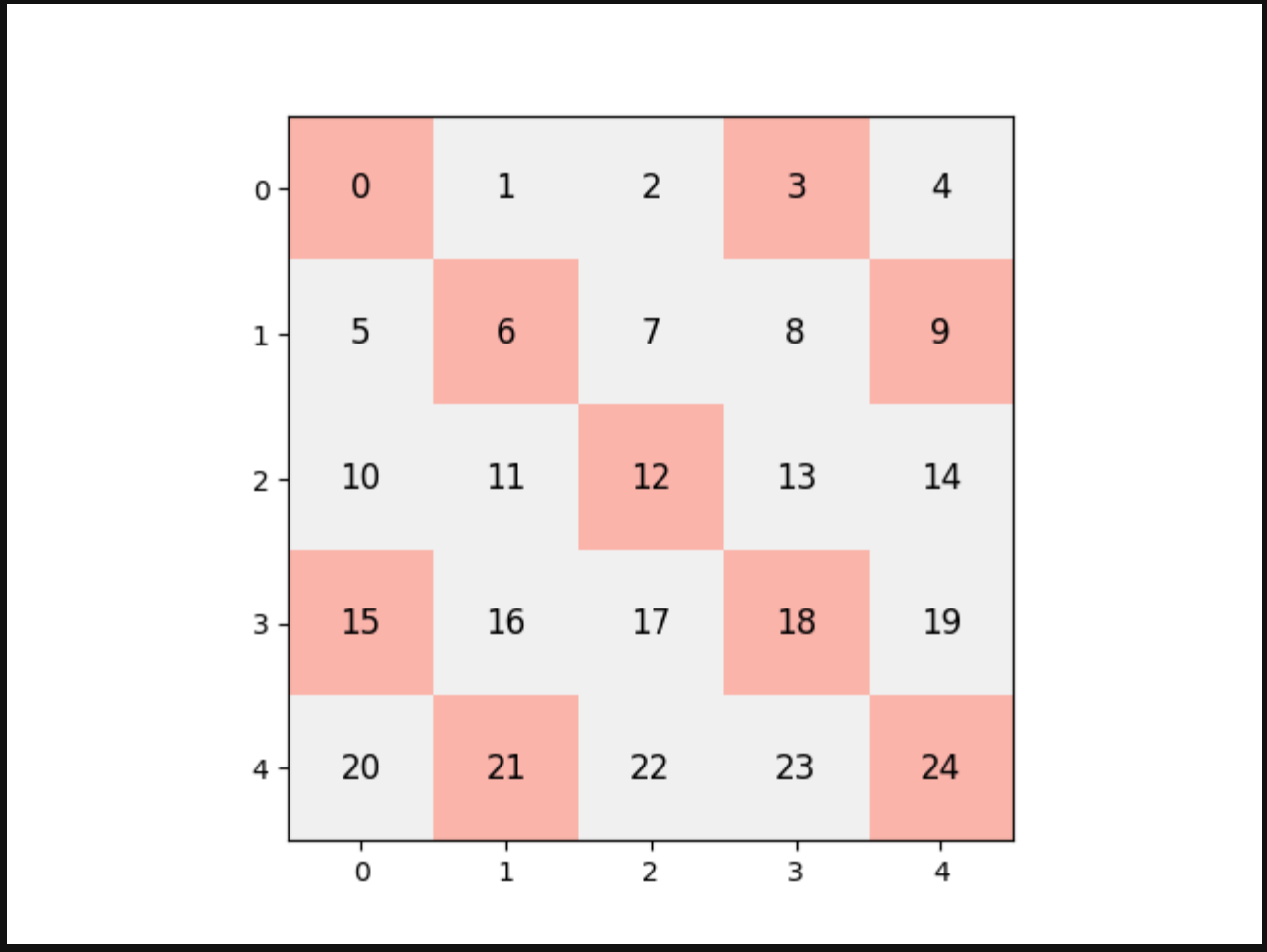
```
[0 1 2 3 4 5 6 7 8 9]
[0 1 2 3 4]
The truth value of an array with more than one element is ambiguous. Use a
[3 4 5 6 7]
```

find and replace

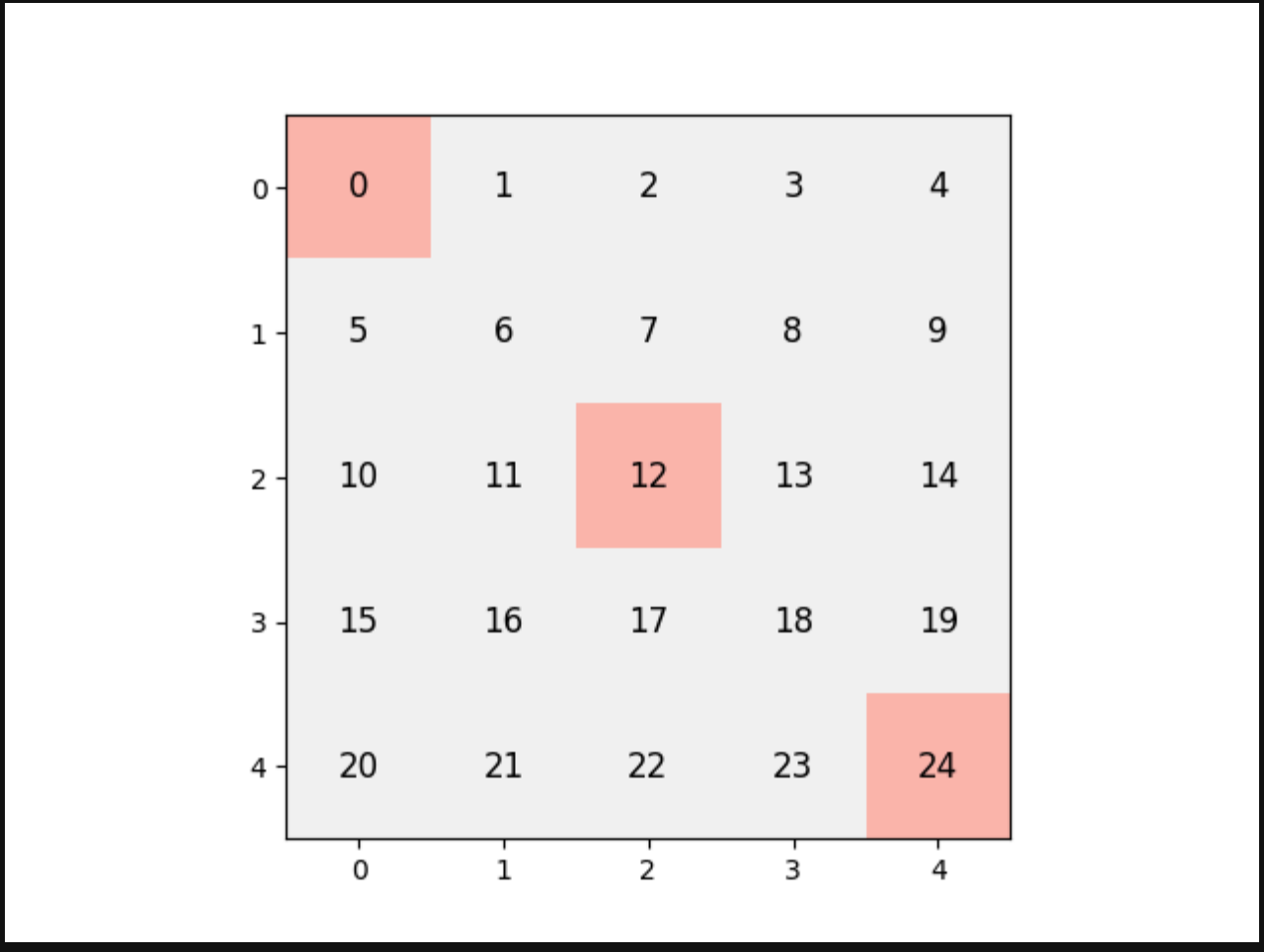
```python
import numpy as np

a = np.arange(-5, 5)
print(a)

print(np.where(a % 2 == 0))  # return index
print(a[np.where(a % 2 == 0)])  # return elements
print(np.where(a % 2 == 0, True, a))  # where with replace
```

```
[-5 -4 -3 -2 -1  0  1  2  3  4]
(array([1, 3, 5, 7, 9]),)
[-4 -2  0  2  4]
[-5  1 -3  1 -1  1  1  1  3  1]
```

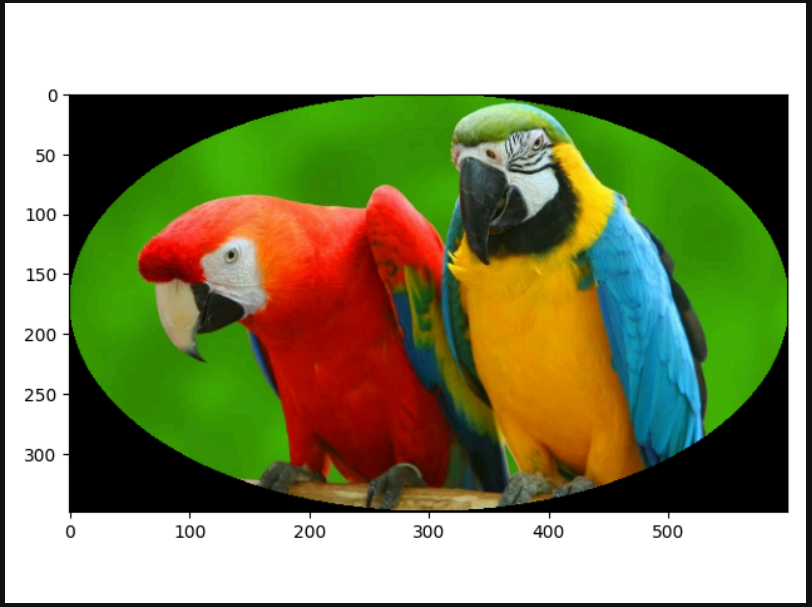
problem: divisible by 3



problem: divisible by 3 & 4



mask



$$\frac{(x-x_0)^2}{a^2}+\frac{(y-y_0)^2}{b^2}=1$$

FANCY INDEXING

indexing normal and fancy

returns a copy of the data rather than view

multi-index

```
import numpy as np

a = np.arange(5)
print(a)
print(a[0])

# multi index
print(a[[0]])
print(a[[1, 2]])
print(a[[1, 2, 3]])

z = a[[0, 1, 2]]  # slice copy operation
print(np.may_share_memory(a, z))  # check
```

```
[0 1 2 3 4]
0
[0]
[1 2]
[1 2 3]
False
```

repeat

[i, i, i]

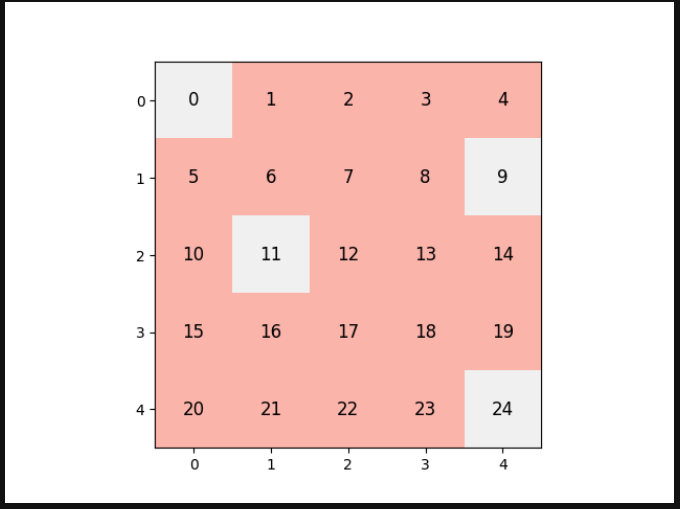
```
import numpy as np

a = np.array([1])
print(a)
print(a[0])
print(a[[0, 0, 0]])
# print(a[[0, 1, 2]])
```

```
[1]
1
[1 1 1]
```

vector

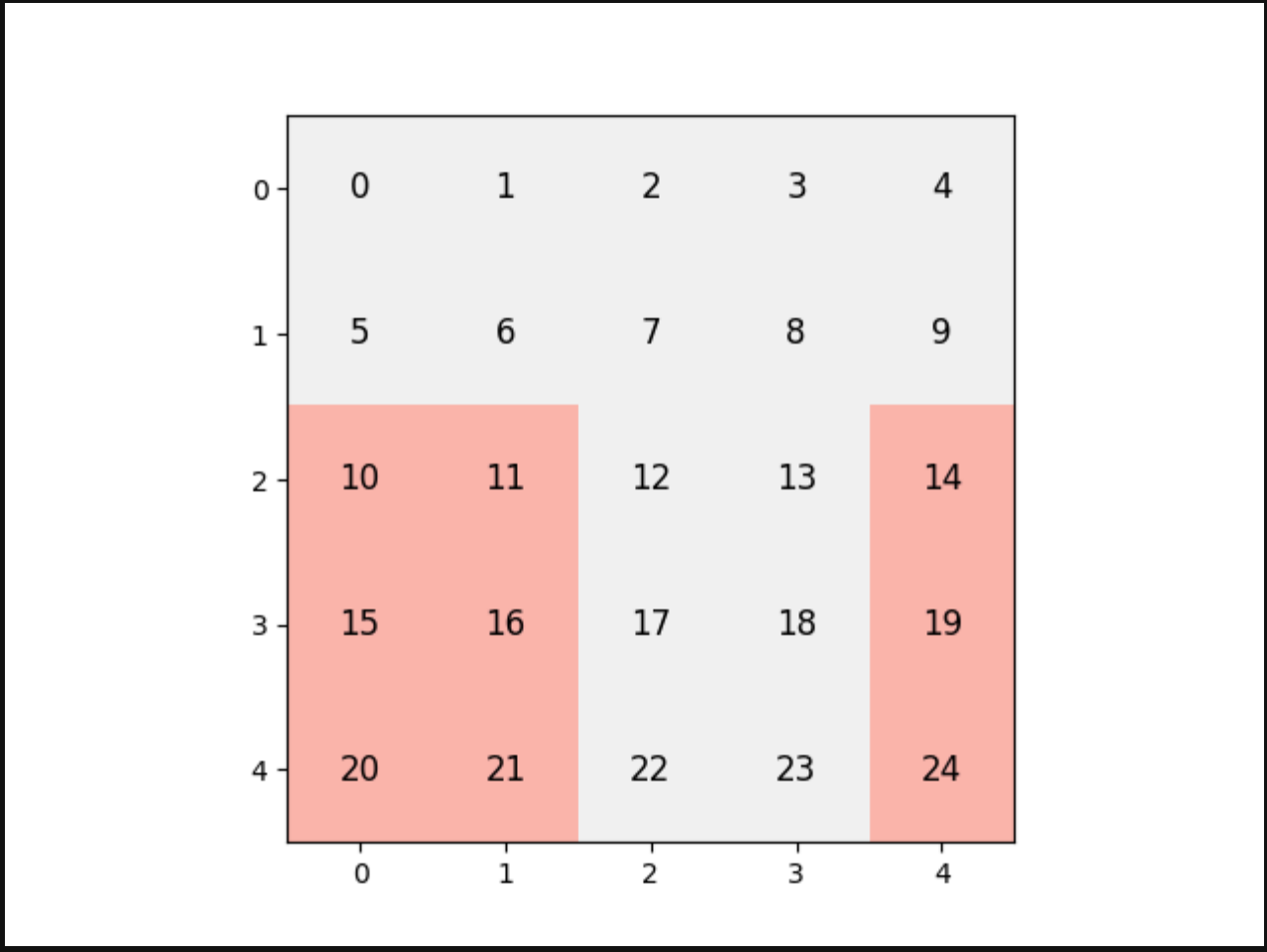
[[y-vector], [x-vertor]]

```
import numpy as np
import matplotlib.pyplot as plt

m = np.ones(25).reshape(5, 5)
m[[0, 2, 1, 4], [0, 1, 4, -1]] = 0
fig = matrix_plot(m)
plt.show()
```



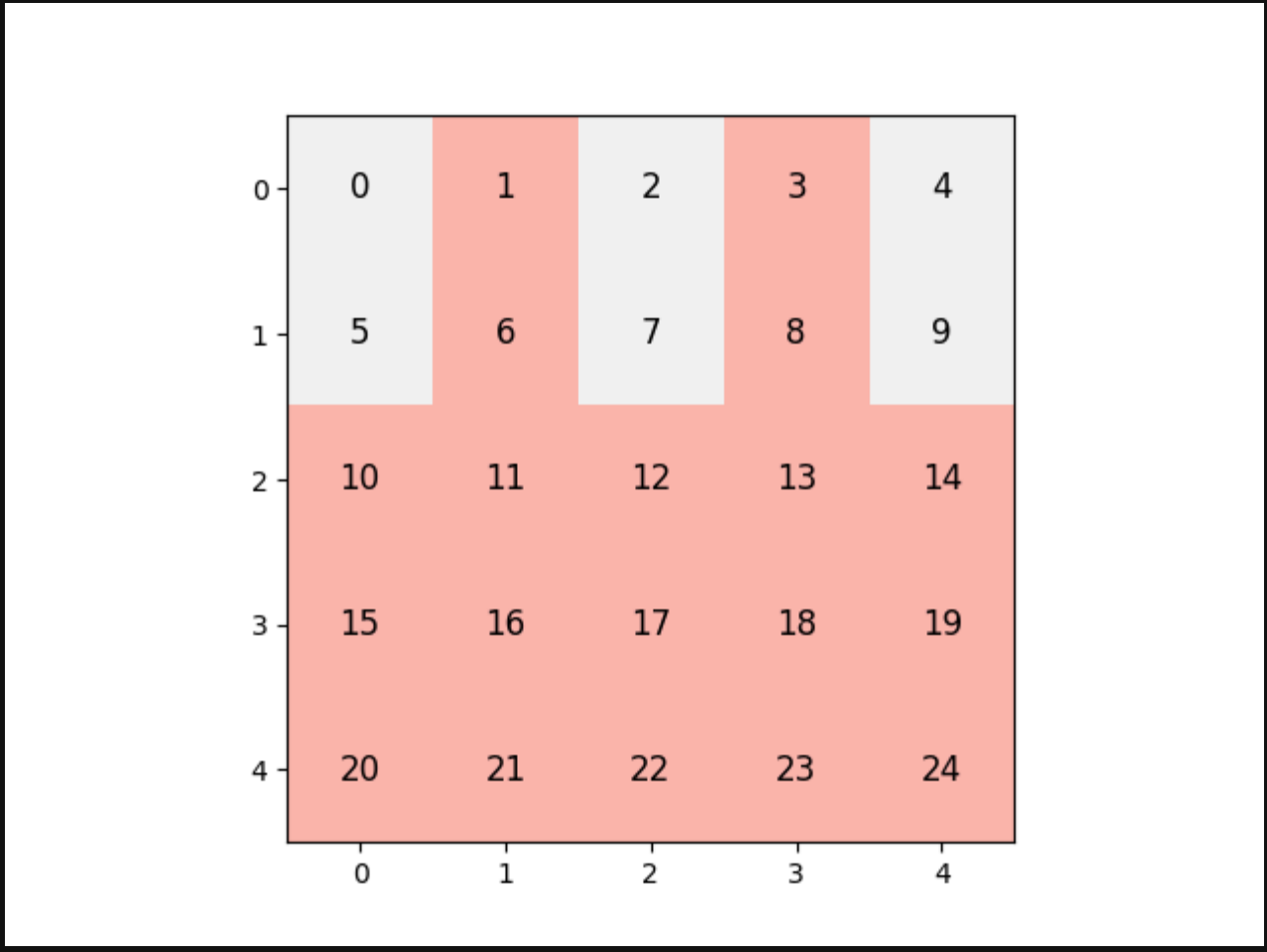
problem 1

[slice, [multi]]



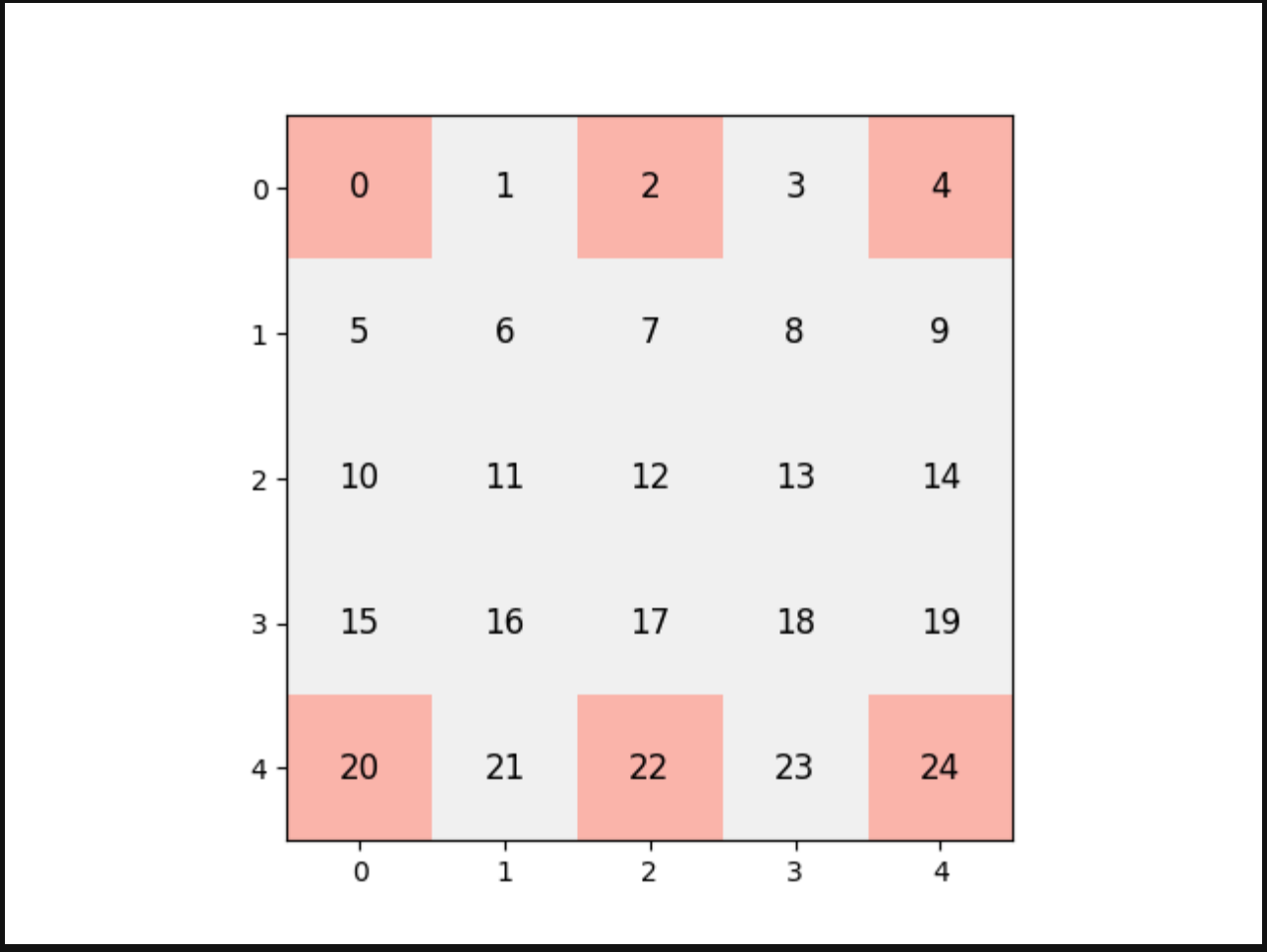
problem 2

[mask, slice]



problems 3

get first and last element of alternative column



AXIS

new axis

```
import numpy as np

m = np.ones(1, int)
print(m)
print(m[:, np.newaxis], m.reshape(1, -1))
# reshape only works for 2d
print(m[:, np.newaxis, np.newaxis])
```

```
[1]
[[1]] [[1]]
[[[1]]]
```

sum

```
import numpy as np

print(m := np.arange(6).reshape(2, 3))
print(sum(m))
print('axis 0:', m.sum(axis=0))
print('axis 1:', m.sum(axis=1))
```

```
[[0 1 2]
 [3 4 5]]
[3 5 7]
axis 0: [3 5 7]
axis 1: [ 3 12]
```