Lecture 16, 15 October 2024

Working with files

Reading files

- Open a file in mode `"r"`

In [1]:
```
fh = open("oz.txt","r")
```

- `read()` returns a string with the entire file
- Note the `\n` characters indicating end of line ("new line")

In [2]:
```
contents = fh.read()
contents
```

Out[2]:
```
'I met a traveller from an antique land\nWho said: Two vast and trunkless legs of stone\nStand in the desaet.
Near them, on the sand,\nHalf sunk, a shattered visage lies, whose frown,\nAnd wrinkled lip, and sneer of cold
command,\nTell that its sculptor well those passions read\nWhich yet survive, stamped on these lifeless thing
s,\nThe hand that mocked them and the heart that fed:\nAnd on the pedestal these words appear:\n"My name is Oz
ymandias, King of Kings:\nLook on my works, ye Mighty, and despair!"\nNo thing beside remains. Round the decay
\nOf that colossal wreck, boundless and bare\nThe lone and level sands stretch far away. \n'
```

- After reading the file, we are at the end
- End of file is indicated by `read()` returning the empty string

In [3]:
```
fh.read()
```

Out[3]:
```
''
```

- To re-read the file, we have to close it and open it again

In [4]:
```
fh.close()
```

- `readline()` reads one line
- Unlike `input()`, the line that is read includes the terminating `\n'

In [5]:
```
fh = open("oz.txt","r")
line1 = fh.readline()
line1
```

Out[5]:
```
'I met a traveller from an antique land\n'
```

- `readlines()` returns the entire file as a list of lines
- Each line is terminated by `\n`

In [6]:
```
fh.close()
fh = open("oz.txt","r")
alllines = fh.readlines()
alllines
```

Out[6]:
```
['I met a traveller from an antique land\n',
 'Who said: Two vast and trunkless legs of stone\n',
 'Stand in the desaet. Near them, on the sand,\n',
 'Half sunk, a shattered visage lies, whose frown,\n',
 'And wrinkled lip, and sneer of cold command,\n',
 'Tell that its sculptor well those passions read\n',
 'Which yet survive, stamped on these lifeless things,\n',
 'The hand that mocked them and the heart that fed:\n',
 'And on the pedestal these words appear:\n',
 '"My name is Ozymandias, King of Kings:\n',
 'Look on my works, ye Mighty, and despair!"\n',
 'No thing beside remains. Round the decay\n',
 'Of that colossal wreck, boundless and bare\n',
 'The lone and level sands stretch far away. \n']
```

- `readlines()` returns the lines from the current position
- In the following example, the list returned is from the second line onwards

In [7]:
```
fh.close()
fh = open("oz.txt","r")
```

```
line1 = fh.readline()      # Read first line
alllines = fh.readlines()  # From second line onwards
alllines
```

Out[7]:
```
['Who said: Two vast and trunkless legs of stone\n',
 'Stand in the desaet. Near them, on the sand,\n',
 'Half sunk, a shattered visage lies, whose frown,\n',
 'And wrinkled lip, and sneer of cold command,\n',
 'Tell that its sculptor well those passions read\n',
 'Which yet survive, stamped on these lifeless things,\n',
 'The hand that mocked them and the heart that fed:\n',
 'And on the pedestal these words appear:\n',
 '"My name is Ozymandias, King of Kings:\n',
 'Look on my works, ye Mighty, and despair!"\n',
 'No thing beside remains. Round the decay\n',
 'Of that colossal wreck, boundless and bare\n',
 'The lone and level sands stretch far away. \n']
```

Stripping white space

- We can strip off leading and trailing "white space" (blank, tab, newline) from a string
- `rstrip()` removes trailing whitespace
- Note that strings are immutable, so the function returns a new string, leaving the original untouched

In [8]:
```
teststr = "   abc    \n"
teststr
```

Out[8]:
```
'   abc    \n'
```

In [9]:
```
teststr.rstrip()
```

Out[9]:
```
'   abc'
```

In [10]:
```
teststr
```

Out[10]:
```
'   abc    \n'
```

- `lstrip()` removes leading whitespace

In [11]:
```
teststr.lstrip()
```

Out[11]:
```
'abc    \n'
```

- `strip()` removes whitespace at both ends

In [12]:
```
teststr.strip()
```

Out[12]:
```
'abc'
```

- Can use these to quickly remove `\n` from lines that we read from a file
- In the first loop below, there is a blank line between actual lines because of the `\n` in the input line followed by the newline inserted by `print()`

In [13]:
```
fh.close()
fh = open("oz.txt","r")
for l in fh.readlines():
    print(l)
```

```
I met a traveller from an antique land

Who said: Two vast and trunkless legs of stone

Stand in the desaet. Near them, on the sand,

Half sunk, a shattered visage lies, whose frown,

And wrinkled lip, and sneer of cold command,

Tell that its sculptor well those passions read

Which yet survive, stamped on these lifeless things,

The hand that mocked them and the heart that fed:

And on the pedestal these words appear:

"My name is Ozymandias, King of Kings:

Look on my works, ye Mighty, and despair!"

No thing beside remains. Round the decay

Of that colossal wreck, boundless and bare

The lone and level sands stretch far away.
```

- If we strip each line before printing, the blank lines are eliminated

In [14]:
```
fh.close()
fh = open("oz.txt","r")
for l in fh.readlines():
    print(l.rstrip())
```

```
I met a traveller from an antique land
Who said: Two vast and trunkless legs of stone
Stand in the desaet. Near them, on the sand,
Half sunk, a shattered visage lies, whose frown,
And wrinkled lip, and sneer of cold command,
Tell that its sculptor well those passions read
Which yet survive, stamped on these lifeless things,
The hand that mocked them and the heart that fed:
And on the pedestal these words appear:
"My name is Ozymandias, King of Kings:
Look on my works, ye Mighty, and despair!"
No thing beside remains. Round the decay
Of that colossal wreck, boundless and bare
The lone and level sands stretch far away.
```

Writing files

- Open a file in mode `"w"`
- Opening a non-existent file for reading generates an error
- Opening a non-existent file for writing creates a new file
- If the file already exists, write will overwrite the contents

In [15]:
```
infile = open("newfile.txt","r")
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[15], line 1
----> 1 infile = open("newfile.txt","r")

File ~/python-venv/venv/lib/python3.11/site-packages/IPython/core/interactiveshell.py:324, in _modified_open(file, *args, **kwargs)
    317 if file in {0, 1, 2}:
    318     raise ValueError(
    319         f"IPython won't let you open fd={file} by default "
    320         "as it is likely to crash IPython. If you know what you are doing, "
    321         "you can use builtins' open."
    322     )
--> 324 return io_open(file, *args, **kwargs)

FileNotFoundError: [Errno 2] No such file or directory: 'newfile.txt'
```

In [16]:
```
outfile = open("newfile.txt","w")
```

In [17]:
```
outfile.close()
```

- `fh.write(s)` writes the string `s` to the file associated with file handle `fh`
- Need to add `\n` ourselves to ensure the end of line
- `fh.writelines(sl)` writes a list of strings `ls`
- Again, we need to ensure `\n` is present at the end of each string in the list
  - The name of the function is misleading
  - It is more accurate to call it `writestrings()` since we have to insert `\n` manually

- The following loop copies the input to the output
  - Each line that is read includes the `\n`
  - Hence each line that is written also has the corresponding `\n`
- However, after this loop, typically the output file will be empty
- Need to close the file handle for the buffer to be "flushed" to the disk

In [18]:
```
infile = open("oz.txt","r")
outfile = open("newfile.txt","w")
for l in infile.readlines():
    outfile.write(l)
```

- Close all file handles when you are done with them

In [19]:
```
infile = open("oz.txt","r")
outfile = open("newfile.txt","w")
for l in infile.readlines():
    outfile.write(l)
infile.close()
outfile.close()
```

- We can also read all the lines into a list using `readlines()` and write them out using `writelines()`

In [20]:
```
infile = open("oz.txt","r")
outfile = open("newfile.txt","w")
contents = infile.readlines()
outfile.writelines(contents)
infile.close()
outfile.close()
```

Appending output

- We can append our writes at the end of a file
- Open the file with mode `"a"` instead of `"w"`
- After the following, `newfile.txt` should have two copies of `oz.txt`

In [21]:
```
infile = open("oz.txt","r")
outfile = open("newfile.txt","a")
contents = infile.readlines()
outfile.writelines(contents)
infile.close()
outfile.close()
```

- There are other functions we have not discussed
- `fh.seek(n)` moves to position `n` in the file
  - After reading a file, use `fh.seek(0)` to return to the start
- `fh.read(n)` reads `n` characters from the current position

In [22]:
```
infile = open("oz.txt","r")
c1 = infile.read()
infile.seek(0)
c2 = infile.read()
```

In [23]:
```
c1
```

Out[23]:
```
'I met a traveller from an antique land\nWho said: Two vast and trunkless legs of stone\nStand in the desaet.
Near them, on the sand,\nHalf sunk, a shattered visage lies, whose frown,\nAnd wrinkled lip, and sneer of cold
command,\nTell that its sculptor well those passions read\nWhich yet survive, stamped on these lifeless thing
s,\nThe hand that mocked them and the heart that fed:\nAnd on the pedestal these words appear:\n"My name is Oz
ymandias, King of Kings:\nLook on my works, ye Mighty, and despair!"\nNo thing beside remains. Round the decay
\nOf that colossal wreck, boundless and bare\nThe lone and level sands stretch far away. \n'
```

In [24]:
```
c2
```

Out[24]:
```
'I met a traveller from an antique land\nWho said: Two vast and trunkless legs of stone\nStand in the desaet.
Near them, on the sand,\nHalf sunk, a shattered visage lies, whose frown,\nAnd wrinkled lip, and sneer of cold
command,\nTell that its sculptor well those passions read\nWhich yet survive, stamped on these lifeless thing
s,\nThe hand that mocked them and the heart that fed:\nAnd on the pedestal these words appear:\n"My name is Oz
ymandias, King of Kings:\nLook on my works, ye Mighty, and despair!"\nNo thing beside remains. Round the decay
\nOf that colossal wreck, boundless and bare\nThe lone and level sands stretch far away. \n'
```

Using numpy

- Arrays and lists
- Arrays are "homogenous" with regular structure
- Lists are flexible

Load numpy

In [25]:
```
import numpy as np
```

Constructing arrays

`np.array()` constructs an array from an input sequence

- Sequence can be a list, tuple, output of a `range()` command ...
- Size of the array is fixed by the sequence
- Underlying type is also fixed

In [26]:
```
b = np.array(range(10))
b
```

Out[26]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Use nested sequences to produce multi-dimensional arrays

- A 2d array is an array of 1d arrays
- Note: can mix and match notation for sequences, but dimensions much match

In [27]:
```
c = np.array([(0,1,0),[2,3,2]])
c
```

Out[27]:
```
array([[0, 1, 0],
       [2, 3, 2]])
```

In [28]:
```
d = np.array([(0,1,0),range(3)])
d
```

Out[28]:
```
array([[0, 1, 0],
       [0, 1, 2]])
```

- A 3d array is an array of 2d arrays

In [29]:
```
d = np.array([[(0,1,0),[2,3,2]],[[4,5,4],[6,7,6]]])
d
```

Out[29]:
```
array([[[0, 1, 0],
        [2, 3, 2]],

       [[4, 5, 4],
        [6, 7, 6]]])
```

Broadcasting

- In simplest form, scalar operations applied pointwise

In [30]:
```
a = np.arange(10)  # arange(n) is same as array(range(n))
a
```

Out[30]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [31]:
```
a**3  # Replace each element by its cube
```

Out[31]:
```
array([  0,   1,   8,  27,  64, 125, 216, 343, 512, 729])
```

In [32]:
```
a+3   # Add 3 to each element
```

Out[32]:
```
array([ 3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

In [33]:
```
3*a   # Multiply each element by 3
```

Out[33]:
```
array([ 0,  3,  6,  9, 12, 15, 18, 21, 24, 27])
```

In [34]:
```
3+a   # Same as a+3
```

Out[34]:
```
array([ 3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

In [35]:
```
a-3, 3-a  # Order is important because subtraction is not commutative
```

Out[35]:
```
(array([-3, -2, -1,  0,  1,  2,  3,  4,  5,  6]),
 array([ 3,  2,  1,  0, -1, -2, -3, -4, -5, -6]))
```

In [36]:
```
3**a  # Powers of 3
```

Out[36]:
```
array([    1,     3,     9,    27,    81,   243,   729,  2187,  6561,
       19683])
```

Stacking arrays

- `np.random.random(m,n)` creates an mxn array with random numbers drawn uniformly from $[0, 1)$
- `10*np.random.random()` scales the entries by 10
- `np.floor()` removes the fractional part

In [37]:
```
a = np.floor(10*np.random.random((2,2)))
b = np.floor(10*np.random.random((2,2)))
print(a)
print(b)
```

```
[[2. 5.]
 [1. 1.]]
[[7. 4.]
 [6. 0.]]
```

- `vstack()` stacks a sequence of arrays vertically -- should have same number of columns

In [38]:
```
np.vstack((a,b))
```

Out[38]:
```
array([[2., 5.],
       [1., 1.],
       [7., 4.],
       [6., 0.]])
```

- Cannot `vstack()` if number of columns don't match

In [39]:
```
c = np.floor(10*np.random.random((2,3)))
c
```

Out[39]:
```
array([[0., 5., 5.],
       [2., 5., 1.]])
```

In [40]:
```
np.vstack((a,c))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[40], line 1
----> 1 np.vstack((a,c))

File ~/python-venv/venv/lib/python3.11/site-packages/numpy/_core/shape_base.py:287, in vstack(tup, dtype, casting)
    285 if not isinstance(arrs, tuple):
    286     arrs = (arrs,)
--> 287 return _nx.concatenate(arrs, 0, dtype=dtype, casting=casting)

ValueError: all the input array dimensions except for the concatenation axis must match exactly, but along dimension 1, the array at index 0 has size 2 and the array at index 1 has size 3
```

- Number of rows does not matter if columns match

In [41]:
```
d = np.floor(10*np.random.random((3,2)))
d
```

Out[41]:
```
array([[0., 6.],
       [2., 8.],
       [0., 2.]])
```

In [42]:
```
np.vstack((a,d))
```

Out[42]:
```
array([[2., 5.],
       [1., 1.],
       [0., 6.],
       [2., 8.],
       [0., 2.]])
```

- Can stack any length sequence of arrays, not just two arrays

In [43]:
```
np.vstack([a,b,d])
```

Out[43]:
```
array([[2., 5.],
       [1., 1.],
       [7., 4.],
       [6., 0.],
       [0., 6.],
       [2., 8.],
       [0., 2.]])
```

- Likewise, `hstack()` stacks horizontally, number of rows must match

In [44]:
```
np.hstack((a,b))
```

Out[44]:
```
array([[2., 5., 7., 4.],
       [1., 1., 6., 0.]])
```

In [45]:
```
np.hstack((b,d))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[45], line 1
----> 1 np.hstack((b,d))

File ~/python-venv/venv/lib/python3.11/site-packages/numpy/_core/shape_base.py:358, in hstack(tup, dtype, casting)
    356     return _nx.concatenate(arrs, 0, dtype=dtype, casting=casting)
    357 else:
--> 358     return _nx.concatenate(arrs, 1, dtype=dtype, casting=casting)

ValueError: all the input array dimensions except for the concatenation axis must match exactly, but along dimension 0, the array at index 0 has size 2 and the array at index 1 has size 3
```

- For `hstack()`, number of columns does not matter
- Like `vstack()`, can combine a sequence of arrays

In [46]:
```
e = np.floor(10*np.random.random((2,3)))
e
```

Out[46]:
```
array([[5., 8., 2.],
       [2., 6., 3.]])
```

In [47]:
```
np.hstack((a,b,e))
```

Out[47]:
```
array([[2., 5., 7., 4., 5., 8., 2.],
       [1., 1., 6., 0., 2., 6., 3.]])
```

Splitting arrays

In [48]:
```
a = np.floor(10*np.random.random((2,12)))
a
```

Out[48]:
```
array([[8., 9., 2., 7., 5., 9., 3., 1., 9., 7., 2., 4.],
       [6., 9., 5., 4., 0., 7., 1., 3., 2., 7., 5., 4.]])
```

- `hsplit(A,n)` splits array `A` into `n` equal parts horizontally
- `n` must be a divisor of the number of columns in `A`

In [49]:
```
np.hsplit(a,3)
```

Out[49]:
```
[array([[8., 9., 2., 7.],
        [6., 9., 5., 4.]]),
 array([[5., 9., 3., 1.],
        [0., 7., 1., 3.]]),
 array([[9., 7., 2., 4.],
        [2., 7., 5., 4.]])]
```

In [50]:
```
np.hsplit(a,6)
```

Out[50]:
```
[array([[8., 9.],
        [6., 9.]]),
 array([[2., 7.],
        [5., 4.]]),
 array([[5., 9.],
        [0., 7.]]),
 array([[3., 1.],
        [1., 3.]]),
 array([[9., 7.],
        [2., 7.]]),
 array([[2., 4.],
        [5., 4.]])]
```

In [51]:
```
np.hsplit(a,5)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[51], line 1
----> 1 np.hsplit(a,5)

File ~/python-venv/venv/lib/python3.11/site-packages/numpy/lib/_shape_base_impl.py:948, in hsplit(ary, indices_o
r_sections)
    946     raise ValueError('hsplit only works on arrays of 1 or more dimensions')
    947 if ary.ndim > 1:
--> 948     return split(ary, indices_or_sections, 1)
    949 else:
    950     return split(ary, indices_or_sections, 0)

File ~/python-venv/venv/lib/python3.11/site-packages/numpy/lib/_shape_base_impl.py:874, in split(ary, indices_or
_sections, axis)
    872     N = ary.shape[axis]
    873     if N % sections:
--> 874         raise ValueError(
    875             'array split does not result in an equal division') from None
    876 return array_split(ary, indices_or_sections, axis)

ValueError: array split does not result in an equal division
```

- Can also specify where to split as a list of columns
- `hsplit(A,[c1,c2,..,ck])` will split as `A[:c1]`, `A[c1:c2]`,..., `A[ck:]`

In [52]:
```
np.hsplit(a,(2,5,7)) # a[:2], a[2:5], a[5:7], a[7:]
```

Out[52]:
```
[array([[8., 9.],
        [6., 9.]]),
 array([[2., 7., 5.],
        [5., 4., 0.]]),
 array([[9., 3.],
        [7., 1.]]),
 array([[1., 9., 7., 2., 4.],
        [3., 2., 7., 5., 4.]])]
```

- Similarly, `vsplit` for vertical split

In [53]:
```
np.vsplit(a,2) # Split a vertically
```

Out[53]:
```
[array([[8., 9., 2., 7., 5., 9., 3., 1., 9., 7., 2., 4.]]),
 array([[6., 9., 5., 4., 0., 7., 1., 3., 2., 7., 5., 4.]])]
```

In [54]:
```
np.split(a,2) # behaves like vsplit
```

Out[54]:
```
[array([[8., 9., 2., 7., 5., 9., 3., 1., 9., 7., 2., 4.]]),
 array([[6., 9., 5., 4., 0., 7., 1., 3., 2., 7., 5., 4.]])]
```

Reshaping arrays

- Can change the *shape* of an array if the dimensions match

In [55]:
```
a.shape
```

Out[55]:
```
(2, 12)
```

In [56]:
```
a.shape = 4,6
a
```

Out[56]:
```
array([[8., 9., 2., 7., 5., 9.],
       [3., 1., 9., 7., 2., 4.],
       [6., 9., 5., 4., 0., 7.],
       [1., 3., 2., 7., 5., 4.]])
```

Copy and view

In [57]:
```
a.shape = 2,12
```

In [58]:
```
c = a.copy()  # Creates a disjoint copy of the array
d = a.view()  # Creates another link to the same array
e = a         # Aliases e to point to same array as a
```

In [59]:
```
a, c, d, e
```

Out[59]:
```
(array([[8., 9., 2., 7., 5., 9., 3., 1., 9., 7., 2., 4.],
        [6., 9., 5., 4., 0., 7., 1., 3., 2., 7., 5., 4.]]),
 array([[8., 9., 2., 7., 5., 9., 3., 1., 9., 7., 2., 4.],
        [6., 9., 5., 4., 0., 7., 1., 3., 2., 7., 5., 4.]]),
 array([[8., 9., 2., 7., 5., 9., 3., 1., 9., 7., 2., 4.],
        [6., 9., 5., 4., 0., 7., 1., 3., 2., 7., 5., 4.]]),
 array([[8., 9., 2., 7., 5., 9., 3., 1., 9., 7., 2., 4.],
        [6., 9., 5., 4., 0., 7., 1., 3., 2., 7., 5., 4.]]))
```

- Updating `c` has no effect on the others since it is a disjoint copy

In [60]:
```
c[0,4] = 88
a, c, d, e
```

Out[60]:
```
(array([[8., 9., 2., 7., 5., 9., 3., 1., 9., 7., 2., 4.],
        [6., 9., 5., 4., 0., 7., 1., 3., 2., 7., 5., 4.]]),
 array([[ 8.,  9.,  2.,  7., 88.,  9.,  3.,  1.,  9.,  7.,  2.,  4.],
        [ 6.,  9.,  5.,  4.,  0.,  7.,  1.,  3.,  2.,  7.,  5.,  4.]]),
 array([[8., 9., 2., 7., 5., 9., 3., 1., 9., 7., 2., 4.],
        [6., 9., 5., 4., 0., 7., 1., 3., 2., 7., 5., 4.]]),
 array([[8., 9., 2., 7., 5., 9., 3., 1., 9., 7., 2., 4.],
        [6., 9., 5., 4., 0., 7., 1., 3., 2., 7., 5., 4.]]))
```

- Updating `d` will indirectly update `a` and `e`, but not `c`

In [61]:
```
d[0,5] = 66
a, c, d, e
```

Out[61]:
```
(array([[ 8.,  9.,  2.,  7.,  5., 66.,  3.,  1.,  9.,  7.,  2.,  4.],
        [ 6.,  9.,  5.,  4.,  0.,  7.,  1.,  3.,  2.,  7.,  5.,  4.]]),
 array([[ 8.,  9.,  2.,  7., 88.,  9.,  3.,  1.,  9.,  7.,  2.,  4.],
        [ 6.,  9.,  5.,  4.,  0.,  7.,  1.,  3.,  2.,  7.,  5.,  4.]]),
 array([[ 8.,  9.,  2.,  7.,  5., 66.,  3.,  1.,  9.,  7.,  2.,  4.],
        [ 6.,  9.,  5.,  4.,  0.,  7.,  1.,  3.,  2.,  7.,  5.,  4.]]),
 array([[ 8.,  9.,  2.,  7.,  5., 66.,  3.,  1.,  9.,  7.,  2.,  4.],
        [ 6.,  9.,  5.,  4.,  0.,  7.,  1.,  3.,  2.,  7.,  5.,  4.]]))
```

- Likewise, updating `e` updates `a` and `d`

In [62]:
```
e[0,6] = 77
a, c, d, e
```

Out[62]:
```
(array([[ 8.,  9.,  2.,  7.,  5., 66., 77.,  1.,  9.,  7.,  2.,  4.],
        [ 6.,  9.,  5.,  4.,  0.,  7.,  1.,  3.,  2.,  7.,  5.,  4.]]),
 array([[ 8.,  9.,  2.,  7., 88.,  9.,  3.,  1.,  9.,  7.,  2.,  4.],
        [ 6.,  9.,  5.,  4.,  0.,  7.,  1.,  3.,  2.,  7.,  5.,  4.]]),
 array([[ 8.,  9.,  2.,  7.,  5., 66., 77.,  1.,  9.,  7.,  2.,  4.],
        [ 6.,  9.,  5.,  4.,  0.,  7.,  1.,  3.,  2.,  7.,  5.,  4.]]),
 array([[ 8.,  9.,  2.,  7.,  5., 66., 77.,  1.,  9.,  7.,  2.,  4.],
        [ 6.,  9.,  5.,  4.,  0.,  7.,  1.,  3.,  2.,  7.,  5.,  4.]]))
```

- We can flatten an array into a sequence

In [63]:
```
for i in a.flat:
    print(i,end=" ")
```

```
8.0 9.0 2.0 7.0 5.0 66.0 77.0 1.0 9.0 7.0 2.0 4.0 6.0 9.0 5.0 4.0 0.0 7.0 1.0 3.0 2.0 7.0 5.0 4.0
```

- `base` tells us if an array shares its storage with another array
- For the original array, `base` is `None`
- For a view, the `base` points to the "parent" array

In [64]:
```
a.base, c.base, d.base, e.base
```

Out[64]:
```
(None,
 None,
 array([[ 8.,  9.,  2.,  7.,  5., 66., 77.,  1.,  9.,  7.,  2.,  4.],
        [ 6.,  9.,  5.,  4.,  0.,  7.,  1.,  3.,  2.,  7.,  5.,  4.]]),
 None)
```

In [65]:
```
d.base is a
```

Out[65]:
```
True
```

- Changing the shape of the base does array not affect the shape of a view

In [66]:
```
a.shape = 4,6
a,c,d,e
```

Out[66]:
```
(array([[ 8.,  9.,  2.,  7.,  5., 66.],
        [77.,  1.,  9.,  7.,  2.,  4.],
        [ 6.,  9.,  5.,  4.,  0.,  7.],
        [ 1.,  3.,  2.,  7.,  5.,  4.]]),
 array([[ 8.,  9.,  2.,  7., 88.,  9.,  3.,  1.,  9.,  7.,  2.,  4.],
        [ 6.,  9.,  5.,  4.,  0.,  7.,  1.,  3.,  2.,  7.,  5.,  4.]]),
 array([[ 8.,  9.,  2.,  7.,  5., 66., 77.,  1.,  9.,  7.,  2.,  4.],
        [ 6.,  9.,  5.,  4.,  0.,  7.,  1.,  3.,  2.,  7.,  5.,  4.]]),
 array([[ 8.,  9.,  2.,  7.,  5., 66.],
        [77.,  1.,  9.,  7.,  2.,  4.],
        [ 6.,  9.,  5.,  4.,  0.,  7.],
        [ 1.,  3.,  2.,  7.,  5.,  4.]]))
```

- Changing the shape of the view does not affect the shape of the base array

In [67]:
```
d.shape = 3,8
a,c,d,e
```

Out[67]:
```
(array([[ 8.,  9.,  2.,  7.,  5., 66.],
        [77.,  1.,  9.,  7.,  2.,  4.],
        [ 6.,  9.,  5.,  4.,  0.,  7.],
        [ 1.,  3.,  2.,  7.,  5.,  4.]]),
 array([[ 8.,  9.,  2.,  7., 88.,  9.,  3.,  1.,  9.,  7.,  2.,  4.],
        [ 6.,  9.,  5.,  4.,  0.,  7.,  1.,  3.,  2.,  7.,  5.,  4.]]),
 array([[ 8.,  9.,  2.,  7.,  5., 66., 77.,  1.],
        [ 9.,  7.,  2.,  4.,  6.,  9.,  5.,  4.],
        [ 0.,  7.,  1.,  3.,  2.,  7.,  5.,  4.]]),
 array([[ 8.,  9.,  2.,  7.,  5., 66.],
        [77.,  1.,  9.,  7.,  2.,  4.],
        [ 6.,  9.,  5.,  4.,  0.,  7.],
        [ 1.,  3.,  2.,  7.,  5.,  4.]]))
```

- Since they still have the same base, updating `a` changes the corresponding element in `d`

In [68]:
```
a[3,0] = 99
a,c,d,e
```

Out[68]:
```
(array([[ 8.,  9.,  2.,  7.,  5., 66.],
        [77.,  1.,  9.,  7.,  2.,  4.],
        [ 6.,  9.,  5.,  4.,  0.,  7.],
        [99.,  3.,  2.,  7.,  5.,  4.]]),
 array([[ 8.,  9.,  2.,  7., 88.,  9.,  3.,  1.,  9.,  7.,  2.,  4.],
        [ 6.,  9.,  5.,  4.,  0.,  7.,  1.,  3.,  2.,  7.,  5.,  4.]]),
 array([[ 8.,  9.,  2.,  7.,  5., 66., 77.,  1.],
        [ 9.,  7.,  2.,  4.,  6.,  9.,  5.,  4.],
        [ 0.,  7., 99.,  3.,  2.,  7.,  5.,  4.]]),
 array([[ 8.,  9.,  2.,  7.,  5., 66.],
        [77.,  1.,  9.,  7.,  2.,  4.],
        [ 6.,  9.,  5.,  4.,  0.,  7.],
        [99.,  3.,  2.,  7.,  5.,  4.]]))
```

- Likewise, updating the shape of `e`, which is an alias for `a`, does not affect the shape of the view `d`

In [69]:
```
e.shape = 8,3
a,c,d,e
```

Out[69]:
```
(array([[ 8.,  9.,  2.],
        [ 7.,  5., 66.],
        [77.,  1.,  9.],
        [ 7.,  2.,  4.],
        [ 6.,  9.,  5.],
        [ 4.,  0.,  7.],
        [99.,  3.,  2.],
        [ 7.,  5.,  4.]]),
 array([[ 8.,  9.,  2.,  7., 88.,  9.,  3.,  1.,  9.,  7.,  2.,  4.],
        [ 6.,  9.,  5.,  4.,  0.,  7.,  1.,  3.,  2.,  7.,  5.,  4.]]),
 array([[ 8.,  9.,  2.,  7.,  5., 66., 77.,  1.],
        [ 9.,  7.,  2.,  4.,  6.,  9.,  5.,  4.],
        [ 0.,  7., 99.,  3.,  2.,  7.,  5.,  4.]]),
 array([[ 8.,  9.,  2.],
        [ 7.,  5., 66.],
        [77.,  1.,  9.],
        [ 7.,  2.,  4.],
        [ 6.,  9.,  5.],
        [ 4.,  0.,  7.],
        [99.,  3.,  2.],
        [ 7.,  5.,  4.]]))
```

Matrix operations

In [70]:
```
a = np.array([[1,2],[3,4]])
b = np.array([[5,6],[7,8]])
```

In [71]:
```
a,b
```

Out[71]:
```
(array([[1, 2],
        [3, 4]]),
 array([[5, 6],
        [7, 8]]))
```

- Pointwise addition and multiplication

In [72]:
```
a+b, a*b
```

Out[72]:
```
(array([[ 6,  8],
        [10, 12]]),
 array([[ 5, 12],
        [21, 32]]))
```

- Matrix multiplication

In [73]:
```
np.matmul(a,b)
```

Out[73]:
```
array([[19, 22],
       [43, 50]])
```

- Transpose and inverse

In [74]:
```
a.T
```

Out[74]:
```
array([[1, 3],
       [2, 4]])
```

In [75]:
```
np.linalg.inv(a)
```

Out[75]:
```
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
```

- AA^{-1} should give the identity matrix
- Note the small imprecision due to round off error

In [76]:
```
np.matmul(a,np.linalg.inv(a))
```

Out[76]:
```
array([[1.0000000e+00, 0.0000000e+00],
       [8.8817842e-16, 1.0000000e+00]])
```

- Fit a function f to a set of data points $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$
- Compute *mean square error (MSE)*

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(f(x_i) - y_i)^2$$

In [77]:
```
predictions = np.array([1.2,2.3,3.1]) # (f(x1), f(x2), f(x3))
```

```
values = np.array([1,2.5,3]) # (y1, y2, y3)
```

- `predictions - values` creates the array $(f(x_1) - y_1, f(x_2) - y_2, f(x_3) - y_3)$
- Applying `np.square` to this squares each element
- Applying `np.sum` to the result sums up the squares of the errors

In [78]:
```
n = len(predictions)
mse = 1/n * np.sum(np.square(predictions-values))
mse
```

Out[78]:
```
np.float64(0.03000000000000002)
```

Axes

In [79]:
```
a = np.random.random((4,3))*10
a
```

Out[79]:
```
array([[9.22019521, 0.13761742, 5.54624285],
       [0.91993814, 2.52040868, 5.58711698],
       [7.34663475, 7.9160672 , 7.52764646],
       [2.19223373, 0.31380292, 3.79908788]])
```

- `np.sum()` adds up all the entries in the array

In [80]:
```
np.sum(a)
```

Out[80]:
```
np.float64(53.02699221695091)
```

- Selecting `axis = 0` applies the operation column by column

In [81]:
```
np.sum(a,axis=0)
```

Out[81]:
```
array([19.67900183, 10.88789622, 22.46009416])
```

- Selecting `axis = 1` applies the opertion row by row

In [82]:
```
np.sum(a,axis=1)
```

Out[82]:
```
array([14.90405548,  9.02746379, 22.79034841,  6.30512453])
```

- `concatenate()` generalizes `vstack()` and `hstack()`
- By default it is `vstack()`, equivalent to setting `axis = 0`
- If we set `axis = 1`, we get `hstack()`

In [83]:
```
b = np.random.random((4,3))*10
b
```

Out[83]:
```
array([[2.5080732 , 8.24675148, 2.82022833],
       [0.93329686, 2.05862046, 2.98499718],
       [3.28746196, 4.57913158, 7.51318614],
       [5.23831641, 2.22004755, 0.32695537]])
```

In [84]:
```
np.vstack((a,b)), np.hstack((a,b))
```

Out[84]:
```
(array([[9.22019521, 0.13761742, 5.54624285],
        [0.91993814, 2.52040868, 5.58711698],
        [7.34663475, 7.9160672 , 7.52764646],
        [2.19223373, 0.31380292, 3.79908788],
        [2.5080732 , 8.24675148, 2.82022833],
        [0.93329686, 2.05862046, 2.98499718],
        [3.28746196, 4.57913158, 7.51318614],
        [5.23831641, 2.22004755, 0.32695537]]),
 array([[9.22019521, 0.13761742, 5.54624285, 2.5080732 , 8.24675148,
         2.82022833],
        [0.91993814, 2.52040868, 5.58711698, 0.93329686, 2.05862046,
         2.98499718],
        [7.34663475, 7.9160672 , 7.52764646, 3.28746196, 4.57913158,
         7.51318614],
        [2.19223373, 0.31380292, 3.79908788, 5.23831641, 2.22004755,
         0.32695537]]))
```

In [85]:
```
np.concatenate((a,b))  # Implicitly, axis = 0, so vstack()
```

Out[85]:
```
array([[9.22019521, 0.13761742, 5.54624285],
       [0.91993814, 2.52040868, 5.58711698],
       [7.34663475, 7.9160672 , 7.52764646],
       [2.19223373, 0.31380292, 3.79908788],
       [2.5080732 , 8.24675148, 2.82022833],
       [0.93329686, 2.05862046, 2.98499718],
       [3.28746196, 4.57913158, 7.51318614],
       [5.23831641, 2.22004755, 0.32695537]])
```

In [86]:
```
np.concatenate((a,b),axis=1) # Concatenate row-wise, so same as hstack()
```

Out[86]:
```
array([[9.22019521, 0.13761742, 5.54624285, 2.5080732 , 8.24675148,
        2.82022833],
       [0.91993814, 2.52040868, 5.58711698, 0.93329686, 2.05862046,
        2.98499718],
       [7.34663475, 7.9160672 , 7.52764646, 3.28746196, 4.57913158,
        7.51318614],
       [2.19223373, 0.31380292, 3.79908788, 5.23831641, 2.22004755,
        0.32695537]])
```

In [87]:
```
c = np.random.random((1,7))
d = np.random.random((1,7))
c,d
```

Out[87]:
```
(array([[0.07267494, 0.3783258 , 0.65656752, 0.68281043, 0.05638744,
         0.37733566, 0.43123276]]),
 array([[0.06481104, 0.08371931, 0.43723602, 0.17467156, 0.29884436,
         0.65714388, 0.61828877]]))
```

In [88]:
```
np.concatenate((c,d),axis=1)
```

Out[88]:
```
array([[0.07267494, 0.3783258 , 0.65656752, 0.68281043, 0.05638744,
        0.37733566, 0.43123276, 0.06481104, 0.08371931, 0.43723602,
        0.17467156, 0.29884436, 0.65714388, 0.61828877]])
```

Broadcasting

- Array with scalar --- map operation to each array element

In [89]:
```
a = np.array([1.0, 2.0, 3.0])
b = 2.0
a * b
```

Out[89]:
```
array([2., 4., 6.])
```

- Array with array/sequence of same length --- pointwise application of operation

In [90]:
```
a = np.array([1.0, 2.0, 3.0])
b = np.array([2.0, 2.0, 2.0])
a * b
```

Out[90]:
```
array([2., 4., 6.])
```

- More generally, can broadcast to an array of same dimension as rightmost index

Example

- Store an $m \times n$ image as 3 layers, Red, Blue and Green
- Array dimensions are $(m, n, 3)$
- Want to scale RGB values by different amounts
- Multiply image by $(rscale, bscale, gscale)$

In [91]:
```
pic = np.random.random((4,4,3))*10
```

In [92]:
```
pic
```

Out[92]:
```
array([[[7.60061179, 1.10183023, 3.71766621],
        [1.73451238, 8.99714649, 8.96094895],
        [9.96641339, 2.09685052, 4.79691403],
        [6.4925484 , 6.80629118, 1.79486827]],

       [[5.34744685, 5.30129566, 2.75368974],
        [7.99303197, 1.22315687, 9.02146656],
        [2.55361894, 3.1242574 , 9.95910227],
        [3.43776671, 4.29265437, 9.01634868]],

       [[4.53021805, 3.40430857, 4.23341826],
        [5.97222599, 4.29933185, 5.61726507],
        [6.40558387, 5.53437176, 5.27726982],
        [9.56150731, 4.92602833, 2.58522762]],

       [[3.84844276, 1.37952909, 0.51063819],
        [1.46683956, 8.88963941, 3.6951251 ],
        [0.40364782, 1.83155109, 3.00337368],
        [2.94853305, 3.22853114, 2.57837861]]])
```

In [93]:
```
pic*[1,100,1000]
```

Out[93]:
```
array([[[7.60061179e+00, 1.10183023e+02, 3.71766621e+03],
        [1.73451238e+00, 8.99714649e+02, 8.96094895e+03],
        [9.96641339e+00, 2.09685052e+02, 4.79691403e+03],
        [6.49254840e+00, 6.80629118e+02, 1.79486827e+03]],

       [[5.34744685e+00, 5.30129566e+02, 2.75368974e+03],
        [7.99303197e+00, 1.22315687e+02, 9.02146656e+03],
        [2.55361894e+00, 3.12425740e+02, 9.95910227e+03],
        [3.43776671e+00, 4.29265437e+02, 9.01634868e+03]],

       [[4.53021805e+00, 3.40430857e+02, 4.23341826e+03],
        [5.97222599e+00, 4.29933185e+02, 5.61726507e+03],
        [6.40558387e+00, 5.53437176e+02, 5.27726982e+03],
        [9.56150731e+00, 4.92602833e+02, 2.58522762e+03]],

       [[3.84844276e+00, 1.37952909e+02, 5.10638192e+02],
        [1.46683956e+00, 8.88963941e+02, 3.69512510e+03],
        [4.03647824e-01, 1.83155109e+02, 3.00337368e+03],
        [2.94853305e+00, 3.22853114e+02, 2.57837861e+03]]])
```

Note

- In the example above, a shape of `(3,4,4)` would display more intuitively as 3 layers of colour for the base image of shape `(4,4)`
- However, for broadcasting, we need the *rightmost* index to match, so we wrote it as `(4,4,3)` , which is laid out less intuitively by `numpy` as 4 layers with shape `(4,3)`

Broadcasting example

- Find the nearest point to a given point in a collection
- Given (x, y) and $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, report j such that (x_j, y_j) is the closest point to (x, y)
- Distance of each point is $\sqrt{(x_i - x)^2 + (y_i - y)^2}$
- $\arg\min$ reports index where $\min$ is achieved

- Here is the whole computation in one shot

In [94]:
```
observation = np.array([111.0, 188.0])  # (x,y)
codes = np.array([[132.0, 193.0],  # [(x1,y1), (x2,y2), (x3,y3), (x4,y4)]
                  [102.0, 203.0],
                  [45.0, 155.0],
                  [57.0, 173.0]])

diff = codes - observation
dist = np.sqrt(np.sum(diff**2,axis=1))
dist, np.argmin(dist)
```

Out[94]:
```
(array([21.58703314, 17.49285568, 73.79024326, 56.04462508]), np.int64(1))
```

- Let us break this up into steps

1. The setup

In [95]:
```
observation = np.array([111.0, 188.0])  # (x,y)
codes = np.array([[132.0, 193.0],  # [(x1,y1), (x2,y2), (x3,y3), (x4,y4)]
                  [102.0, 203.0],
                  [45.0, 155.0],
                  [57.0, 173.0]])
```

2. Use broadcast to subtact (x, y) from each (x_i, y_i) to get the array $[(x_1 - x, y_1 - y), (x_2 - x, y_2 - y), (x_3 - x, y_3 - y), (x_4 - x, y_4 - y)]$

In [96]:
```
diff = codes - observation
```

In [97]:
```
diff
```

Out[97]:
```
array([[ 21.,   5.],
       [ -9.,  15.],
       [-66., -33.],
       [-54., -15.]])
```

3. Use scalar broadcast to map each pair $(x_i - x, y_i - y)$ to $((x_i - x)^2, (y_i - y)^2)$

In [98]:
```
diff**2
```

Out[98]:
```
array([[ 441.,   25.],
       [  81.,  225.],
       [4356., 1089.],
       [2916.,  225.]])
```

4. Add up each row as $(x_i - x)^2 + (y_i - y)^2$ by computing `np.sum()` along `axis = 1`

In [99]:
```
np.sum(diff**2,axis=1)
```

Out[99]:
```
array([ 466.,  306., 5445., 3141.])
```

5. Apply `np.sqrt()` pointwise to this array of squared differences

In [100…
```
np.sqrt(np.sum(diff**2,axis=1))
```

Out[100…
```
array([21.58703314, 17.49285568, 73.79024326, 56.04462508])
```

6. Find the index where this value is minimum

In [101…
```
dist = np.sqrt(np.sum(diff**2,axis=1))
np.argmin(dist)
```

Out[101…
```
np.int64(1)
```

- Or, equivalently, without the intermediate variable `dist`

In [102…
```
np.argmin(np.sqrt(np.sum(diff**2,axis=1)))
```

Out[102…
```
np.int64(1)
```

- In fact, the entire computation can be written in one line

In [103…
```
np.argmin(np.sqrt(np.sum((codes-observation)**2,axis=1)))
```

Out[103…
```
np.int64(1)
```