Astro 310 HW4

September 28, 2024

1 ASTR 310 HW 4

1.1 1. Functional Programming in Python

Rewrite each of the following code fragments as directed. Be sure to test your rewritten code to verify that it behaves exactly like the original.

a) Rewrite using dictionaries with the names as keys and the Lx values as values. [5 *pts*]

```
names = [ "RXJ9938.1", "CL2016.33", "A4123", "A1101"]
Lx = [1.5e44, 2.8e44, 1.0e45, 9.7e43]
names = names + ["RXJ1234.8"]
Lx = Lx + [8.4e44]
```

[9]:
```
# Original
names = [ "RXJ9938.1", "CL2016.33", "A4123", "A1101"]
Lx = [1.5e44, 2.8e44, 1.0e45, 9.7e43]
names = names + ["RXJ1234.8"]
Lx = Lx + [8.4e44]
print("ORIGINAL OUTPUT")
print(names, Lx)

# Rewritten
lx_dict = dict(zip(names, Lx))
lx_dict["RXJ1234.8"] = 8.4e44
print("\nREWRITTEN OUTPUT")
print(lx_dict)
```

```
ORIGINAL OUTPUT
['RXJ9938.1', 'CL2016.33', 'A4123', 'A1101', 'RXJ1234.8'] [1.5e+44, 2.8e+44,
1e+45, 9.7e+43, 8.4e+44]

REWRITTEN OUTPUT
{'RXJ9938.1': 1.5e+44, 'CL2016.33': 2.8e+44, 'A4123': 1e+45, 'A1101': 9.7e+43,
'RXJ1234.8': 8.4e+44}
```

b) Rewrite using list comprehensions. [5 *pts*]

```
x = [0.] * 100
y = [0.] * 100
for i in range(100):
```

```
    x[i] = 6. + i*0.3
    y[i] = 4.*x[i]**2 - x[i]**0.5
```

[15]:
```
# Original
x = [0.] * 100
y = [0.] * 100
for i in range(100):
    x[i] = 6. + i*0.3
    y[i] = 4.*x[i]**2 - x[i]**0.5
print("ORIGINAL OUTPUT")
print(x, y)


# Rewritten
x2 = [6. + i*0.3 for i in range(100)]
y2 = [4. * x[i]**2 - x[i]**0.5 for i in range(100)]
print("\nREWRITTEN OUTPUT")
print(x2, y2)

# Check
print(f"\nThe lists are equal: {(x == x2) & (y == y2)}")
```

```
ORIGINAL OUTPUT
[6.0, 6.3, 6.6, 6.9, 7.2, 7.5, 7.8, 8.1, 8.4, 8.7, 9.0, 9.3, 9.6, 9.9, 10.2,
10.5, 10.8, 11.1, 11.399999999999999, 11.7, 12.0, 12.3, 12.6,
12.899999999999999, 13.2, 13.5, 13.8, 14.1, 14.4, 14.7, 15.0,
15.299999999999999, 15.6, 15.9, 16.2, 16.5, 16.799999999999997, 17.1, 17.4,
17.7, 18.0, 18.299999999999997, 18.6, 18.9, 19.2, 19.5, 19.799999999999997,
20.1, 20.4, 20.7, 21.0, 21.299999999999997, 21.6, 21.9, 22.2, 22.5, 22.8,
23.099999999999998, 23.4, 23.7, 24.0, 24.3, 24.599999999999998, 24.9, 25.2,
25.5, 25.8, 26.099999999999998, 26.4, 26.7, 27.0, 27.3, 27.599999999999998,
27.9, 28.2, 28.5, 28.8, 29.099999999999998, 29.4, 29.7, 30.0, 30.3,
30.599999999999998, 30.9, 31.2, 31.5, 31.8, 32.099999999999994, 32.4, 32.7,
33.0, 33.3, 33.599999999999994, 33.9, 34.2, 34.5, 34.8, 35.099999999999994,
35.4, 35.7] [141.5505102572168, 156.25001992039776, 171.67095348426696,
187.81321489268728, 204.67671842700025, 222.26138721247418, 240.5671519912462,
259.5939501058485, 279.3417246507621, 299.81042375924943, 321.0,
342.9104098636046, 365.541613323034, 388.89357345548956, 412.9662561154657,
437.75962965079606, 463.2736646549691, 489.50833375020846, 516.4636113967731,
544.1394737247025, 572.5358983848622, 601.65286441665, 631.4903521301402,
662.0483430007863, 693.3268195750829, 725.3257653858252, 758.0451648757987,
791.4850033288963, 825.645266807798, 860.5259420974637, 896.1270166537926,
932.4484785568783, 969.4903164683736, 1007.2525195925247, 1045.7350776405003,
1084.937980797682, 1124.8612196936158, 1165.5047853743731, 1206.8686692770768,
1248.9528632064073, 1291.7573593128807, 1335.282150072758, 1379.5272282694307,
1424.4925869761428, 1470.1782195399587, 1516.584119566836, 1563.7102809077421,
1611.5566976457083, 1660.1233640837454, 1709.410274733569, 1759.417424305044,
1810.1448076963138, 1861.5924199845513, 1913.7602564172807, 1966.648312404244,
```

2020.2565835097475, 2074.585065445475, 2129.6337540637205, 2185.4026453510205,
2241.8917354221444, 2299.1010205144335, 2357.0304969824533, 2415.6801612929444,
2475.0500100200497, 2535.1400398407955, 2595.950247530819, 2657.4806299603197,
2719.7311840902216, 2782.7019069685334, 2846.3927957268943, 2910.8038475772933,
2975.935059808955, 3041.786429785374, 3108.357954941502, 3175.649632781059,
3243.661460873984, 3312.393436854001, 3381.845558416295, 3452.017823315309,
3522.9102293626242, 3594.5227744249482, 3666.855456422191, 3739.9082733256237,
3813.681223156125, 3888.1743039824923, 3963.387513919839, 4039.3208511280513,
4115.974313810312, 4193.3479002116965, 4271.441608617803, 4350.255437353462,
4429.7893847814985, 4510.043449301523, 4591.017629348796, 4672.711923393115,
4755.126329937764, 4838.260847518498, 4922.1154747025585, 5006.690210087742,
5091.985052301485]

REWRITTEN OUTPUT
[6.0, 6.3, 6.6, 6.9, 7.2, 7.5, 7.8, 8.1, 8.4, 8.7, 9.0, 9.3, 9.6, 9.9, 10.2,
10.5, 10.8, 11.1, 11.399999999999999, 11.7, 12.0, 12.3, 12.6,
12.899999999999999, 13.2, 13.5, 13.8, 14.1, 14.4, 14.7, 15.0,
15.299999999999999, 15.6, 15.9, 16.2, 16.5, 16.799999999999997, 17.1, 17.4,
17.7, 18.0, 18.299999999999997, 18.6, 18.9, 19.2, 19.5, 19.799999999999997,
20.1, 20.4, 20.7, 21.0, 21.299999999999997, 21.6, 21.9, 22.2, 22.5, 22.8,
23.099999999999998, 23.4, 23.7, 24.0, 24.3, 24.599999999999998, 24.9, 25.2,
25.5, 25.8, 26.099999999999998, 26.4, 26.7, 27.0, 27.3, 27.599999999999998,
27.9, 28.2, 28.5, 28.8, 29.099999999999998, 29.4, 29.7, 30.0, 30.3,
30.599999999999998, 30.9, 31.2, 31.5, 31.8, 32.099999999999994, 32.4, 32.7,
33.0, 33.3, 33.599999999999994, 33.9, 34.2, 34.5, 34.8, 35.099999999999994,
35.4, 35.7] [141.5505102572168, 156.25001992039776, 171.67095348426696,
187.81321489268728, 204.67671842700025, 222.26138721247418, 240.5671519912462,
259.5939501058485, 279.3417246507621, 299.81042375924943, 321.0,
342.9104098636046, 365.541613323034, 388.89357345548956, 412.9662561154657,
437.75962965079606, 463.2736646549691, 489.50833375020846, 516.4636113967731,
544.1394737247025, 572.5358983848622, 601.65286441665, 631.4903521301402,
662.0483430007863, 693.3268195750829, 725.3257653858252, 758.0451648757987,
791.4850033288963, 825.645266807798, 860.5259420974637, 896.1270166537926,
932.4484785568783, 969.4903164683736, 1007.2525195925247, 1045.7350776405003,
1084.937980797682, 1124.8612196936158, 1165.5047853743731, 1206.8686692770768,
1248.9528632064073, 1291.7573593128807, 1335.282150072758, 1379.5272282694307,
1424.4925869761428, 1470.1782195399587, 1516.584119566836, 1563.7102809077421,
1611.5566976457083, 1660.1233640837454, 1709.410274733569, 1759.417424305044,
1810.1448076963138, 1861.5924199845513, 1913.7602564172807, 1966.648312404244,
2020.2565835097475, 2074.585065445475, 2129.6337540637205, 2185.4026453510205,
2241.8917354221444, 2299.1010205144335, 2357.0304969824533, 2415.6801612929444,
2475.0500100200497, 2535.1400398407955, 2595.950247530819, 2657.4806299603197,
2719.7311840902216, 2782.7019069685334, 2846.3927957268943, 2910.8038475772933,
2975.935059808955, 3041.786429785374, 3108.357954941502, 3175.649632781059,
3243.661460873984, 3312.393436854001, 3381.845558416295, 3452.017823315309,
3522.9102293626242, 3594.5227744249482, 3666.855456422191, 3739.9082733256237,
3813.681223156125, 3888.1743039824923, 3963.387513919839, 4039.3208511280513,
4115.974313810312, 4193.3479002116965, 4271.441608617803, 4350.255437353462,

```
4429.7893847814985, 4510.043449301523, 4591.017629348796, 4672.711923393115,
4755.126329937764, 4838.260847518498, 4922.1154747025585, 5006.690210087742,
5091.985052301485]

The lists are equal: True
```

c) Rewrite using list comprehensions and `filter`. Hint: you can nest list comprehensions (use one inside another). Yes, this will be much easier with numpy arrays, but we are not doing that yet. The intent here is to practice with list comprehensions so solutions using numpy will be graded as incorrect. What's going on here? We're building a 3D "cube" (a nested list) and rho is the distance to the center of the cube. [7 *pts*]

```
x = list(range(101))
y = list(range(101))
z = list(range(101))
for i in range(101):
    x[i] = x[i] * 0.01
    y[i] = y[i] * 0.01
    z[i] = z[i] * 0.01
rho = []
s = 0.
for k in range(101):
    rho = rho + [[]]
    for j in range(101):
        rho[k] = rho[k] + [[]]
        for i in range(101):
            rho[k][j] = rho[k][j] + [((x[i]-0.5)**2 + \
                                      (y[j]-0.5)**2 + \
                                     (z[k]-0.5)**2)**0.5]
            if rho[k][j][i] > 0.5:
                s = s + rho[k][j][i]
```

[9]:
```
# Original
x = list(range(101))
y = list(range(101))
z = list(range(101))
for i in range(101):
    x[i] = x[i] * 0.01
    y[i] = y[i] * 0.01
    z[i] = z[i] * 0.01
rho = []
s = 0.
for k in range(101):
    rho = rho + [[]]
    for j in range(101):
        rho[k] = rho[k] + [[]]
        for i in range(101):
            rho[k][j] = rho[k][j] + [((x[i]-0.5)**2 + \
```

```
                                        (y[j]-0.5)**2 + \
                                       (z[k]-0.5)**2)**0.5]
            if rho[k][j][i] > 0.5:
                s = s + rho[k][j][i]

# Rewritten
x2 = [i * 0.01 for i in range(101)]
y2 = [i * 0.01 for i in range(101)]
z2 = [i * 0.01 for i in range(101)]
rho2 = [[[((x[i]-0.5)**2 + (y[j]-0.5)**2 + (z[k]-0.5)**2)**0.5 for i in␣
 ↪range(101)] for j in range(101)] for k in range(101)]
s2 = sum(filter(lambda r: r > 0.5, [rho2[k][j][i] for i in range(101) for j in␣
 ↪range(101) for k in range(101)]))

print(f"original s: {s}\nrewritten s: {s2}")
```

```
original s: 303575.1160929
rewritten s: 303575.1160929159
```

d) Rewrite using list comprehensions and `zip`. What's going on here? We've constructed a spherical model of a star, so we have the star's density as a function of radius, and we're integrating the total mass of the star. Notice the use of $dM = 4\pi r^2 \rho \, dr$. [8 *pts*]

```
r = [1.0, 2.0, 4.0, 8.0, 16.0, 32.0]
rho = [0.99, 0.87, 0.79, 0.61, 0.50, 0.45]
i = 0
m = lastr = 0.
vfact = 1./3.    # innermost step is a filled sphere
while i < 6:
    m = m + 4.*3.14159 * r[i]**2 * rho[i] * (r[i] - lastr)
    print("%5.2f %5.2f %10.2f" % (r[i], rho[i], m))
    lastr = r[i]
    vfact = 1.   # after the first step,  use spherical shells
    i = i + 1
```

[16]:
```
# Original
print("Original")
r = [1.0, 2.0, 4.0, 8.0, 16.0, 32.0]
rho = [0.99, 0.87, 0.79, 0.61, 0.50, 0.45]
i = 0
m = lastr = 0.
vfact = 1./3.    # innermost step is a filled sphere
while i < 6:
    m = m + 4.*3.14159 * r[i]**2 * rho[i] * (r[i] - lastr)
    print("%5.2f %5.2f %10.2f" % (r[i], rho[i], m))
    lastr = r[i]
    vfact = 1.   # after the first step,  use spherical shells
    i = i + 1
```

```
# Rewritten
print("\nRewritten")
lastr2 = 0.0
m2 = 0.0

for r, rho in zip(r, rho):
    m2 += 4.0 * 3.14159 * r**2 * rho * (r - lastr2)
    print(f"{r:5.2f} {rho:5.2f} {m2:10.2f}")
    lastr2 = r
```

```
Original
 1.00  0.99      12.44
 2.00  0.87      56.17
 4.00  0.79     373.85
 8.00  0.61    2336.21
16.00  0.50   15204.16
32.00  0.45  107853.42

Rewritten
 1.00  0.99      12.44
 2.00  0.87      56.17
 4.00  0.79     373.85
 8.00  0.61    2336.21
16.00  0.50   15204.16
32.00  0.45  107853.42
```

1.2 2. Working with tabular data from a file

Obtain the file "stars.txt" from the course website. This is a text file containing a table of observational quantities for the 26 brightest stars as seen from Earth. Each line of the file contains the name of a star, its Bayer designation (e.g. Alpha Canis Majoris), its distance in light-years (ly), its apparent visual magnitude, its absolute visual magnitude, and its spectral type. The spectral types include the luminosity class (e.g. V for dwarf stars) and some additional designations (e.g. v for variable, or the spectral type of a binary companion, such as "G2V + K1V").

a) Construct a dictionary that stores the mapping between the abbreviations for constellation names and the full names of the constellations. For example, the key "CMa" should be associated with the value "Canis Major". The 88 official IAU constellations are described at https://www.iau.org/public/themes/constellations/. You don't need all 88, just the ones represented in the "stars.txt" file. [4 *pts*]

[3]:
```
constellation_dict = {
    "CMa" : "Canis Major",
    "Car" : "Carina",
    "Cen" : "Centaurus",
    "Boo" : "Bootes",
    "Lyr" : "Lyra",
```

```
    "Aur" : "Auriga",
    "Ori" : "Orion",
    "CMi" : "Canis Minor",
    "Eri" : "Eridanis",
    "Cru" : "Crux",
    "Aql" : "Aquila",
    "Tau" : "Taurus",
    "Sco" : "Scorpius",
    "Vir" : "Virgo",
    "Gem" : "Gemini",
    "PsA" : "Piscis Austrinus",
    "Cyg" : "Cygnus",
    "Leo" : "Leo",
}
```

b) Write a function that computes the absolute magnitude M given the apparent magnitude m and the distance d in parsecs using $m - M = 5(\log d - 1)$. [4 *pts*]

[8]:
```
from math import *
def absMag(appMag, distance):
    return appMag - 5 * (log10(distance) - 1)
```

c) Read the "stars.txt" file and use it to construct a dictionary in which the keys are the names of stars and the values are dictionaries containing * the constellation name using the dictionary from part a, * the distance in parsecs (3.26 ly = 1 pc), * the apparent magnitude, * the absolute magnitude from the file, * the absolute magnitude calculated from the apparent magnitude and the distance, using your function, and * the major spectral class (of the first component if the star is a binary).

For example the key-value pair for Rigil Kentaurus would be something like the following. [6 *pts*]

```
"Rigil Kentaurus" : {"constellation" : "Centaurus", "distance" : 1.3, "m" : -0.27, "M" : 4.4,
```

[14]:
```
f = open("stars.txt")
lines = f.readlines()
f.close()

my_dict = dict()

for item in lines[::-1]:
    name = item[0:16].strip()
    const = item[17:32].strip().split(" ")[1]
    distance = float(item[32:41].strip())
    appMag = float(item[41:51].strip())
    absMagnum = float(item[51:62].strip())
    sclass = item[62::].strip()


    my_dict[name] = {
```

```
        "constellation": constellation_dict[const],
        "distance": distance,
        "m": appMag,
        "M": absMagnum,
        "newM": absMag(appMag, distance),
        "class": sclass
        }

my_dict
```

[14]:
```
{'Shaula': {'constellation': 'Scorpius',
  'distance': 330.0,
  'm': 1.63,
  'M': -3.5,
  'newM': -5.962569699389438,
  'class': 'B1.5IV'},
 'Gacrux': {'constellation': 'Crux',
  'distance': 120.0,
  'm': 1.63,
  'M': -1.2,
  'newM': -3.765906230238124,
  'class': 'M3.5III'},
 'Castor': {'constellation': 'Gemini',
  'distance': 49.0,
  'm': 1.57,
  'M': 0.5,
  'newM': -1.8809804001425683,
  'class': 'A1V + A2V'},
 'Adhara': {'constellation': 'Canis Major',
  'distance': 570.0,
  'm': 1.5,
  'M': -4.8,
  'newM': -7.279374278362457,
  'class': 'B2II'},
 'Regulus': {'constellation': 'Leo',
  'distance': 69.0,
  'm': 1.35,
  'M': -0.3,
  'newM': -2.8442454536862765,
  'class': 'B7Vn'},
 'Deneb': {'constellation': 'Cygnus',
  'distance': 1500.0,
  'm': 1.25,
  'M': -7.2,
  'newM': -9.630456295278407,
```

```
 'class': 'A2Ia'},
'Becrux': {'constellation': 'Crux',
 'distance': 460.0,
 'm': 1.25,
 'M': -4.7,
 'newM': -7.06378915840787,
 'class': 'B0.5III'},
'Fomalhaut': {'constellation': 'Piscis Austrinus',
 'distance': 22.0,
 'm': 1.16,
 'M': 2.0,
 'newM': -0.5521134041110309,
 'class': 'A3Va'},
'Pollux': {'constellation': 'Gemini',
 'distance': 40.0,
 'm': 1.14,
 'M': 0.7,
 'newM': -1.8702999566398117,
 'class': 'K0IIIb'},
'Spica': {'constellation': 'Virgo',
 'distance': 220.0,
 'm': 0.98,
 'M': -3.2,
 'newM': -5.732113404111031,
 'class': 'B1V'},
'Antares': {'constellation': 'Scorpius',
 'distance': 520.0,
 'm': 0.96,
 'M': -5.2,
 'newM': -7.6200167181739955,
 'class': 'M1.5Iab'},
'Aldebaran': {'constellation': 'Taurus',
 'distance': 60.0,
 'm': 0.85,
 'M': -0.3,
 'newM': -3.0407562519182183,
 'class': 'K5III'},
'Altair': {'constellation': 'Aquila',
 'distance': 16.0,
 'm': 0.77,
 'M': 2.3,
 'newM': -0.25059991327962394,
 'class': 'A7Vn'},
'Acrux': {'constellation': 'Crux',
 'distance': 510.0,
 'm': 0.76,
 'M': -4.6,
```

```
 'newM': -7.777850880489682,
 'class': 'B0.5Iv + B1Vn'},
'Hadar': {'constellation': 'Centaurus',
 'distance': 320.0,
 'm': 0.61,
 'M': -4.4,
 'newM': -6.915749891599529,
 'class': 'B1III'},
'Betelgeuse': {'constellation': 'Orion',
 'distance': 1400.0,
 'm': 0.5,
 'M': -7.2,
 'newM': -10.23064017839119,
 'class': 'M2Iab'},
'Achernar': {'constellation': 'Eridanis',
 'distance': 69.0,
 'm': 0.46,
 'M': -1.3,
 'newM': -3.7342454536862766,
 'class': 'B3Vnp'},
'Procyon': {'constellation': 'Canis Minor',
 'distance': 11.4,
 'm': 0.38,
 'M': 2.6,
 'newM': 0.09547574331763642,
 'class': 'F5IV-V'},
'Rigel': {'constellation': 'Orion',
 'distance': 1400.0,
 'm': 0.12,
 'M': -8.1,
 'newM': -10.610640178391192,
 'class': 'B81ae'},
'Capella': {'constellation': 'Auriga',
 'distance': 41.0,
 'm': 0.08,
 'M': 0.4,
 'newM': -2.983919283598677,
 'class': 'G6III + G2III'},
'Vega': {'constellation': 'Lyra',
 'distance': 25.0,
 'm': 0.03,
 'M': 0.6,
 'newM': -1.9597000433601885,
 'class': 'A0Va'},
'Arcturus': {'constellation': 'Bootes',
 'distance': 34.0,
 'm': -0.04,
```

```
 'M': 0.2,
 'newM': -2.697394585211276,
 'class': 'K1.5IIIp'},
'Rigil Kentaurus': {'constellation': 'Centaurus',
 'distance': 4.3,
 'm': -0.27,
 'M': 4.4,
 'newM': 1.5626577221020672,
 'class': 'G2V + K1V'},
'Canopus': {'constellation': 'Carina',
 'distance': 74.0,
 'm': -0.72,
 'M': -2.5,
 'newM': -5.066158598654881,
 'class': 'A9II'},
'Sirius': {'constellation': 'Canis Major',
 'distance': 8.6,
 'm': -1.46,
 'M': 1.4,
 'newM': -1.1324922562178383,
 'class': 'A1Vm'}}
```

e) Construct a list of the names of the stars in order sorted by absolute magnitude. Using this list, print a table containing the stars' names, constellations, absolute magnitudes, and spectral classes. The table should have an appropriate header, and each column should be formatted to be the width of the longest entry in that column. Absolute magnitudes should have two digits after the decimal point and have the decimal points line up. Below is an example of decent formatting, though these aren't sorted correctly. [5 *pts*]

```
Name          Constellation   AbsMag     Class
Achernar      Eridani         -1.17        B3
Altair        Aquila           2.32        A7
```

[63]:
```
sortedbyAbsMag = sorted(my_dict.keys(), key=lambda x: my_dict[x]['M'])
print("{n:16}{c:17}{M:6} {cl:15}\n".
  ↪format(n="Name",c="Constellation",M="AbsMag",cl="Class"))
for star in sortedbyAbsMag:
    print("{n:16}{c:17}{M:<+6.2f} {cl:15}".
  ↪format(n=star,c=my_dict[star]['constellation'],M=my_dict[star]['M'],cl=my_dict[star]['class
```

```
Name            Constellation    AbsMag Class

Rigel           Orion            -8.10  B81ae
Deneb           Cygnus           -7.20  A2Ia
Betelgeuse      Orion            -7.20  M2Iab
Antares         Scorpius         -5.20  M1.5Iab
Adhara          Canis Major      -4.80  B2II
Becrux          Crux             -4.70  B0.5III
Acrux           Crux             -4.60  B0.5Iv + B1Vn
```

Hadar	Centaurus	-4.40	B1III
Shaula	Scorpius	-3.50	B1.5IV
Spica	Virgo	-3.20	B1V
Canopus	Carina	-2.50	A9II
Achernar	Eridanis	-1.30	B3Vnp
Gacrux	Crux	-1.20	M3.5III
Regulus	Leo	-0.30	B7Vn
Aldebaran	Taurus	-0.30	K5III
Arcturus	Bootes	+0.20	K1.5IIIp
Capella	Auriga	+0.40	G6III + G2III
Castor	Gemini	+0.50	A1V + A2V
Vega	Lyra	+0.60	A0Va
Pollux	Gemini	+0.70	K0IIIb
Sirius	Canis Major	+1.40	A1Vm
Fomalhaut	Piscis Austrinus	+2.00	A3Va
Altair	Aquila	+2.30	A7Vn
Procyon	Canis Minor	+2.60	F5IV-V
Rigil Kentaurus	Centaurus	+4.40	G2V + K1V