# examTemplate2

November 14, 2023

## 1 ASTR 1040 Jupyter Notebook Exam Template

This is a template file you can use to complete your exam. The first cell (below after FAQ section) contains lots of helpful constants you may need. Make sure to run it before you start working!

Below I've also included examples of how to do common `astropy` / math things:

### 1.1 Astropy FAQ

This is a template file you can use to do your homeworks in. I recommend copying this each time you start a homework. The first cell (below) contains lots of helpful constants you may need to use throughout the semester. Make sure to run it each time before you start working!

For reference, all of the `astropy` constants as well as examples can be found here. Likewise all of the units and their names / how to access them can be found here.

To create a variable with units you need to multiply by the corresponding unit class:

```
d = 1 * units.au
```

You can then convert unitful quantities to other units by calling the `to` method and passing the desired end unit class:

```
d_m = d.to(units.m)
```

If you have a ratio of quanities where all the units should cancel out, you can obtain the dimensionles number using the `dimesionless_unscaled` method. For example writing:

```
d_m/d
```

yields $1.4959787 \times 10^{11} \frac{\mathrm{m}}{\mathrm{AU}}$, but doing:

```
(d_m/d).to(units.dimensionless_unscaled)
```

returns `1.0` as expected.

It may also be useful to convert quantities to SI units, and you can do this on any quantity that has units by adding `.si` to it. For example:

```
d.si
```

returns $1.4959787 \times 10^{11}$ m as this is an AU in SI units (SI unit of distance is the m).

If you ever need to obtain just the *number* of a quantity (without any units) you can do:

```
d.value
```

which returns 1 (no units, just the number 1 since we said `d = 1 * units.au`).

## 1.2 Math FAQ

Let's say we have two numbers assigned to variables `x` and `y`, i.e.:

```
x = 1.
y = 2.
```

`Addition/subtraction/multiplication/division work largely how you would expect:`

To add:

```
x_plus_y = x+y
print(x_plus_y)
>>> 3.
```

To subtract:

```
x_minus_y = x-y
print(x_minus_y)
>>> -1.
```

To multiply:

```
x_times_y = x*y
print(x_times_y)
>>> 2.
```

To divide:

```
x_over_y = x/y
print(x_over_y)
>>> 0.5
```

Unfortunately exponents are probably *not* what you would expect, but to raise to a power you do:

```
y_tothe_x = y**x #note the ** for exponents
```

Oftentimes in astrophysics we have very large numbers, i.e. the mass of the supermassive black hole at the center of our galaxy (Sag A*) is $\sim 4 \times 10^6 M_\odot$. To write this number in python we can use the convenient "e" syntax:

```
from astropy.constants import M_sun
SagAMass = 4e6*M_sun
print(SagAMass)
>>> 7.953639482792203e+36 kg
```

To take the nth root of something raise it to the power of 1/n, i.e. to take the cubic root of `x`:

```
cubeRoot = x**(1/3)
```

### 1.2.1 numpy

Oftentimes we will also use numpy to do math, as it provides convenient functionality to interface with all of the trig things we will need. This is also where you will get $\pi$ from! A few examples:

“‘python import numpy as np pi = np.pi sinx = np.sin(x) sinx_plus_2pi = np.sin(x+2*pi) #should be the same as sinx angle = np.atan2(y/x) #arctangent, use atan2 if you care about which quadrant! h = np.sqrt(x**2+y**2) #let x and y define sides of triangle y_trig = h*np.cos(angle) #should be the same as y

[ ]:
```
#SETUP CELL (modify at your own peril)
from astropy import units #access units by doing units.<unit> (i.e. units.au)
from astropy import constants
import numpy as np #common math functions (i.e. np.sin(x)) and better arrays (i.e. np.array([1,2,3])
import matplotlib.pyplot as plt #plotting functions (i.e. plt.plot(x,y))
G = constants.G # gravitational constant
M_sun = constants.M_sun # mass of the sun
R_sun = constants.R_sun # radius of the sun
L_sun = constants.L_sun # luminosity of the sun
M_earth = constants.M_earth # mass of the earth
R_earth = constants.R_earth # radius of the earth
M_jup = constants.M_jup # mass of jupiter
R_jup = constants.R_jup # radius of jupiter
sigma_sb = constants.sigma_sb # Stefan-Boltzmann constant
b_wien = constants.b_wien # Wien's displacement constant
c = constants.c # speed of light
h = constants.h # Planck constant
k_B = constants.k_B # Boltzmann constant
m_e = constants.m_e # mass of electron
m_p = constants.m_p # mass of proton
m_n = constants.m_n # mass of neutron (basically just the mass of a proton but whatever)
g0 = constants.g0 # standard gravity, 9.8 m/s^2
e = constants.e # absolute value of electron/proton charge
```

### 1.3 Cheat sheet area

If you have opted to make a cheat sheet as a Jupyter notebook cell, copy and paste your markdown content into the empty markdown cell below and any code content into the empty code cell below:

[ ]:
```
#your cheat sheet code content here
```

### 1.4 Exam instructions:

For the multiple choice and short answer questions you have two options: 1. Answer these on the actual test (i.e. circle the multiple choice answer you select and write out by hand the answers to short answer questions) 2. Clearly label and type them into markdown cells (i.e. for MC type 1: a if you picked a for q1)

For the quantitative questions, if you opt to use the notebook you have two options: 1. Just use the notebook to do calculations, writing your final answers down by hand on the physical test. If you do this make sure you note this on the test so I know to check your work here! 2. Clearly label and fully answer the questions here, with a combination of markdown and code cells as you see fit.

After you have finished the test, create a PDF from your work (just like you have done with your homework) and upload it to the corresponding Midterm 1 Canvas assignment.

[ ]:

### 1.4.1 MULTIPLE CHOICE

1. a
2. b
3. a
4. b
5. a
6. d
7. a
8. b
9. e
10. d
11. a
12. c
13. b
14. a
15. b
16. c
17. d
18. c
19. c
20. b

### 1.4.2 SHORT ANSWER QUESTIONS

1. Based on the plot, astronomers of the time would think the universe was more than we think it is now. The age can be thought about as taking $\frac{1}{H_0}$, where $H_0$ is the 'slope' of the line on the plot. Because the slope of the graph for 1929 is ~50 km/s/Mpc, and our current estimate for $H_0$ is 70 km/s/Mpc, this implies that they would think the universe is older than we think it is today.

[ ]:
```
H_1929=((500*units.km/units.s) / (10e6*units.parsec)).to(units.km/units.s/units.
↪Mpc)
age_1929 = (1 / H_1929).to(units.year)
H0 = 70 * units.km / units.s / units.Mpc
age_now = (1/H0).to(units.year)
print(f"age 1929: {age_1929}, age now: {age_now}")
H_1929
```

```
age 1929: 19555844433.615784 yr, age now: 13968460309.72556 yr
```

[ ]: $50 \frac{\text{km}}{\text{Mpc s}}$

2. Black Hole Freediving

a. Assuming Sterling's mind is set, I would encourage him to jump into a supermassive black hole as opposed to a stellar mass, because a supermassive black hole would technically be less violent in the spaghettification process. This is because a black hole's density is inversely proportional to the square of its mass, meaning that the larger the black hole, the less dense it will be and thus this would make its effects less harsh on Sterling's body.

b. As he jumps in, I will perceive him as being blueshifted as he crosses the event horizon, which would imply he ages incredibly fast. He would also go through "spagghetification", as his feet would be closer to the horizon compared to his head, essentially pulling him apart.

c. As Sterling falls in, he will see the entire history of the Universe in an instant, and then he will be gone. He will not notice himself aging at a different rate as outside observers would.

### 1.4.3 QUANTITATIVE

[ ]:
```
# 1.

 _rest = 0.12 * units.micron
 _emit = 1.5 * units.micron
z = ( _rest /  _emit)
v = ((1+z)**2 - 1) / ((1+z)**2 + 1) * c
d = v / H0
t = d/c
t.to(units.year)
```

[ ]: $1.0729098 \times 10^{9}$ yr

[ ]:
```
# 2.

r = 30 * units.kpc
  = (12.5e9 * M_sun) / ((4/3) * np.pi * r**3)

M = lambda r,  : (  * (4/3) * np.pi * r**3).to(units.M_sun)
v = lambda r,  : (2*r * np.sqrt((G* *np.pi)/(3))).to(units.km/units.s)

M1 = M(r, )
v1 = v(r, )
Mv = ((v1**2 * r) / G).to(units.M_sun)
ratio = Mv/M1
ratio
```

[ ]: 1