Astronomy and Light Pollution

By: Dylan Winer and Jake Kasitz

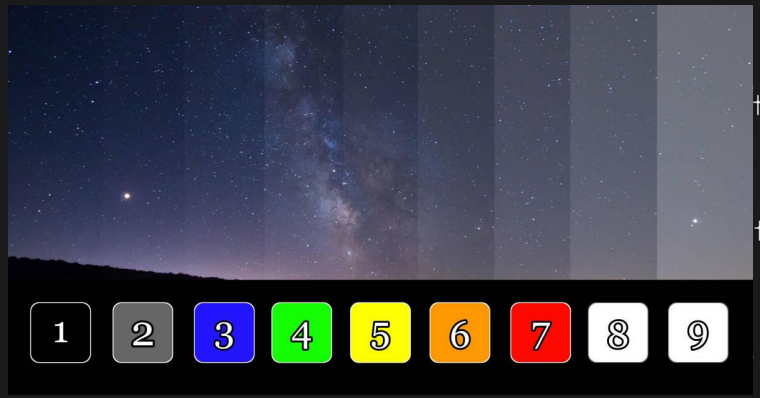
YSP Python A - Dr. Kyle Shaw

Introduction

What our topic is and why we chose it

Our Topic Choice

- The scientific problem we are solving is taking in images of the night sky and then quantifying the quality of the image
  - Determine the number of stars, constellations, and light pollution to assign a rating
  - Bortle Scale: method to quantify light pollution from 1-9
    - 1 is amazing (no light pollution)
    - 9 is atrocious (extreme light pollution)



Bortle Scale: https://nightskypix.com/bortle-scale/

Our Topic Choice Cont.

https://earthobservatory.nasa.gov/features/NightLights

- Additionally, we wanted to output the user's location against a map of light pollution visible from space
  - Shows user whether they should expect high or low visibility of the night sky
- Likewise, we wanted to show the user possible constellations so they can identify them themselves
  - The program will then output a second scale for quality and compare to the bortle scale

Our Topic Motivation

- We chose this topic because we have always been interested in astronomy and studying the night sky
  - Recent images from the James Webb Telescope have also inspired us to observe the cosmos
  - We want to analyze light pollution and illustrate its negative impact on stargazing
- The night sky is amazing to view, which light pollution unfortunately detracts from

https://www.npr.org/2022/07/17/1111714756/james-webb-telescope-big-bang-galaxy-image-interview-project-manager-bill-ochs

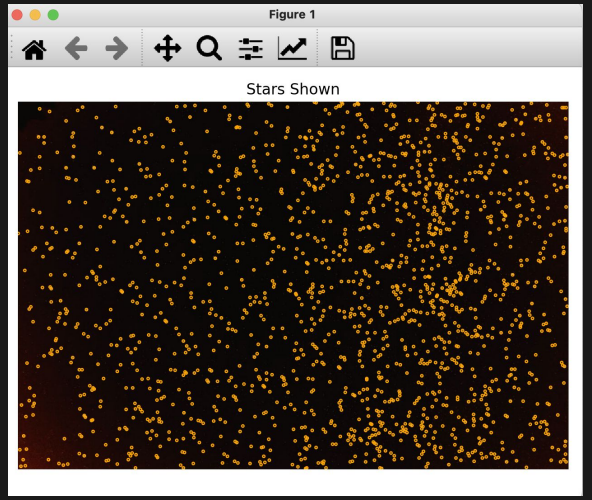
Counting Stars (not the song)

```
def countStars(name):
    img = cv2.imread(name)
    image_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    blobs_log = blob_log(image_gray, max_sigma=20, num_sigma=10, threshold=.05)

    figure = plt.figure()
    ax = figure.add_subplot(1, 1, 1)

    ax.set_title('Stars Shown')
    ax.imshow(img)

    stars_count = 0
    for pixel in blobs_log:
        y, x, r = pixel
        if r > 2:
            continue
        ax.add_patch(plt.Circle((x, y), r, color="orange", linewidth=2, fill=False))
        stars_count += 1

    findAverageandDominate()
    text = str(stars_count)
    self.stars.setText(text)

    ax.set_axis_off()
    plt.tight_layout()
    plt.show()
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

- We used OpenCV to process images of stars
  - Image is converted to grayscale
  - Grayscale gets plotted
  - Iterate through pixels if radius smaller than 2, add pixel patch to total stars
- From there we used a blob detection function from OpenCV to detect the bright stars from from the darker sky
  - Output second plot with circles highlighted

Counting Stars Output

Original Image



Number of Stars: 1698

Output

Detecting Light Pollution

- We take the stars detected and remove them from the image
- Then we take an average value of the color of the sky
- We then find this score
- From that average color, we compute the score of light population based on brightness and hue
  - If the color is closer to a yellow hue, the score increases
  - Same if the color is brighter

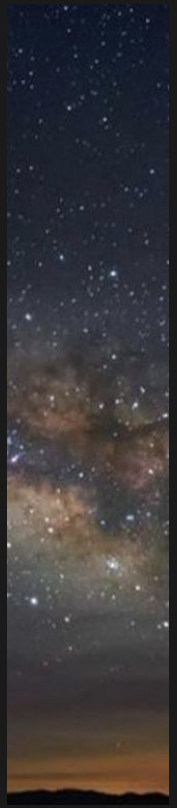
```
def findAverageandDominate():
    from colorthief import ColorThief
    img = io.imread('starrating.jpeg')[:, :, :-1]
    import colorsys

    average = img.mean(axis=0).mean(axis=0)
    color_thief = ColorThief('starrating.jpeg')
    # get the dominant color
    dominant_color = color_thief.get_color(quality=1)
    palette = color_thief.get_palette(color_count=2)
    print(dominant_color, palette)
    hslcolor = colorsys.rgb_to_hls(0,0,0)
    hslcolor = (hslcolor[0]*360,hslcolor[1]/255,abs(hslcolor[2]))
    getYellowness(hslcolor)
def getYellowness(hsl):
    score = (1 / (1 + abs(50 - hsl[0]) * .13)) * .7 + (hsl[1]*2) * .7 + hsl[2] * .07
    print(score,hsl,1 / (1 + abs(50 - hsl[0]) * .13) * .7,(1 / (abs(hsl[1] - .5) * 2 + 1)) * .7,hsl[2] * .07)
```

Rating Light Pollution

```
def get_rating(pollution):
    global bortle
    if pollution < 0.1:  # If ratio is essentially 0
        bortle = 1
        category = "Excellent dark-sky site"
    elif pollution < 0.2:  # less than 5%
        bortle = 2
        category = "Typical truly dark site"
    elif pollution < 0.3:  # less than 15%
        bortle = 3
        category = "Rural sky"
    elif pollution < 0.4:  # less than 35%
        bortle = 4
        category = "Rural/suburban transition"
    elif pollution < 0.5:  # less than 65%
        bortle = 5
        category = "Suburban sky"
    elif pollution < 0.6:  # greater than 65%
        bortle = 6
        category = "Bright suburban sky"
    elif pollution < 0.7:  # greater than 65%
        bortle = 7
        category = "Suburban/urban transition"
    elif pollution < 0.8:  # greater than 65%
        bortle = 8
        category = "City sky"
    else:
        bortle = 9
        category = "Inner-city sky"
```

- We then take this score and calculate a bortle rating
  - Bortle rating of 1 = excellent
  - Bortle rating of 9 = urban-city light pollution
- Ratio of pixels with light pollution compared to total pixels determines bortle rating
- Also output a category like excellent site or rural sky depending on rating

Rating Light Pollution Output



https://suburbanastro.com/astrophotography/bortle-scale-light-pollution/

Bortle Scale (1-9): 3

https://www.theguardian.com/science/2019/apr/17/not-so-starry-night-light-pollution-spoils-the-view-for-stargazers

Bortle Scale (1-9): 5

Constellations

- We ask the user for their longitude, latitude, and season
  - Positive coordinates mean northern hemisphere
  - Negative coordinates mean southern hemisphere
  - Constellations visible change depending on season and hemisphere
- Based on these attributes, we tell the user what constellations they might be able to see.

```
def find_constellations(coords, date):
    print("Coords are:", coords)
    print("Season is:", date)
    date = date.lower()
    hemisphere = ""
    constellations = []

    if coords[0] >= 0:                    0:
        hemisphere = "northern"
    elif coords[0] < 0:                   0:
        hemisphere = "southern"
    else:
        print("Invalid coordinates")
    print("Hemisphere is", hemisphere)

    if hemisphere == "northern" and date == "winter":
        constellations = ["Orion", "Gemini", "Taurus", "Ursa Minor"]

    global constellations_lower
    constellations_lower = []
    for name in constellations:
        name_new = name.lower()
        constellations_lower.append(name_new)

    print("Names of constellations:", constellations)
    ' '.join(constellations)
    text3 = str(constellations)
    self.consts.setText(text3)

    for const in constellations_lower:
        img = cv2.imread(const+".jpeg")
        cv2.imshow(const, img)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
```

Constellations Continued

- Possible combinations of northern or southern hemispheres combined with seasons provide below lists:

```
if hemisphere == "northern" and date == "winter":
    constellations = ["Orion", "Gemini", "Taurus", "Ursa Minor"]
if hemisphere == "northern" and date == "spring":
    constellations = ["Leo", "Virgo", "Ursa Minor"]
if hemisphere == "northern" and date == "summer":
    constellations = ["Scorpius", "Sagittarius", "Cygnus", "Ursa Minor"]
if hemisphere == "northern" and date == "fall":
    constellations = ["Pegasus", "Pisces", "Ursa Minor"]

if hemisphere == "southern" and date == "winter":
    constellations = ["Aquila", "Cygnus", "Hercules", "Lyra", "Ophiuchus", "Sagittarius", "Scorpius"]
if hemisphere == "southern" and date == "spring":
    constellations = ["Andromeda", "Aquarius", "Capricornus", "Pegasus", "Pisces"]
if hemisphere == "southern" and date == "summer":
    constellations = ["Canis Major", "Cetus", "Eridanus", "Gemini", "Orion", "Perseus", "Taurus"]
if hemisphere == "southern" and date == "fall":
    constellations = ["Bootes", "Cancer", "Crater", "Hydra", "Leo", "Virgo"]
```

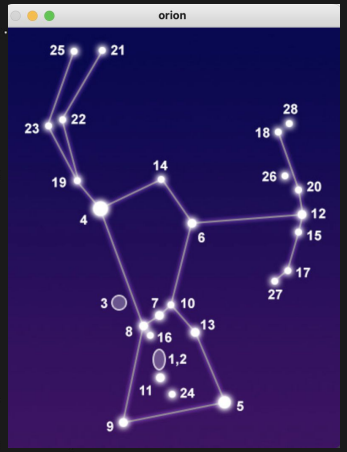
Constellations Analyzed

- We also output the constellations to the user, and they can input which of the constellations they see

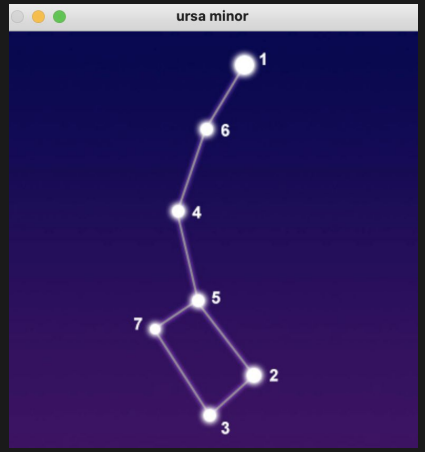
```
global constellations_lower
constellations_lower = []
for name in constellations:
    name_new = name.lower()
    constellations_lower.append(name_new)

print("Names of constellations:", constellations)
' '.join(constellations)
text3 = str(constellations)
self.consts.setText(text3)

for const in constellations_lower:
    img = cv2.imread(const+".jpeg")
    cv2.imshow(const, img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Constellations Analyzed Cont.

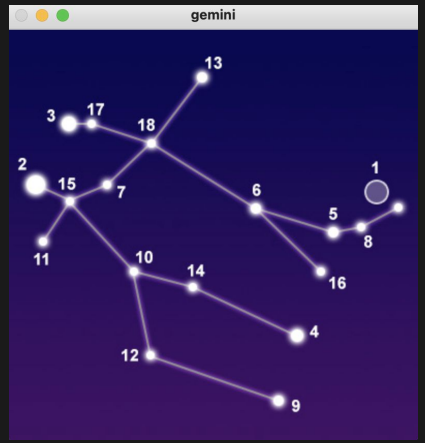
- Example for northern hemisphere and winter:



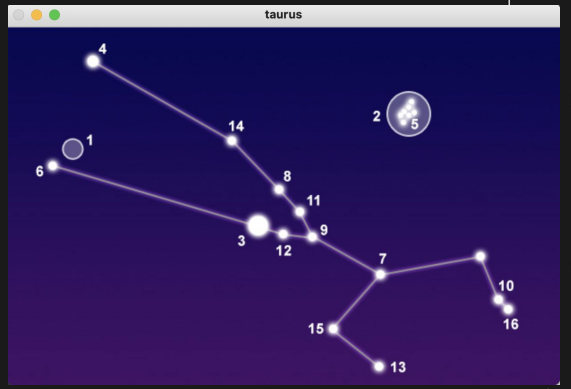
Orion



Ursa Minor



Gemini



Taurus

Constellations Analyzed Cont.

- We then sorted constellations into bright, middle, and low groups based on online listings

```
bright = ["orion", "taurus", "canis major", "centaurus", "scorpius", "pegasus", "lyra",
          "ophiuchus", "eridanus", "bootes", "hydra"]
middle = ["leo", "virgo", "cygnus", "aquila", "hercules", "andromeda", "cetus", "perseus"]
low = ["ursa minor", "gemini", "pisces", "sagittarius", "aquarius", "capricornus", "cancer", "crater"]
```

- Iterate through to find dimmest constellation visible, which shows a higher quality, as dim constellations are harder to see
- Maximum difficulty to see is quality value kept

```
qualities = []
for constellation in visibles_list:
    if constellation in bright:
        qualities.append(1)
    elif constellation in middle:
        qualities.append(2)
    elif constellation in low:
        qualities.append(3)
```

Constellations Analyzed Cont.

- Determine quality based on constellation
- Compare that quality to bortle quality, and see if they match
- Output whether they match back to user

```
max_quality = max(qualities)
if max_quality == 3:
    qual = "very high"
    print("Quality is very high")
elif max_quality == 2:
    qual = "high"
    print("Quality is high")
elif max_quality == 1:
    qual = "medium"
    print("Quality is medium")
elif max_quality == 0:
    qual = "low"
    print("Quality is bad")
```

```
if 7 <= bortle <= 9:
    bortle_qual = "low"
elif 5 <= bortle < 7:
    bortle_qual = "medium"
elif 3 <= bortle < 5:
    bortle_qual = "high"
elif 1 <= bortle < 3:
    bortle_qual = "very high"
```

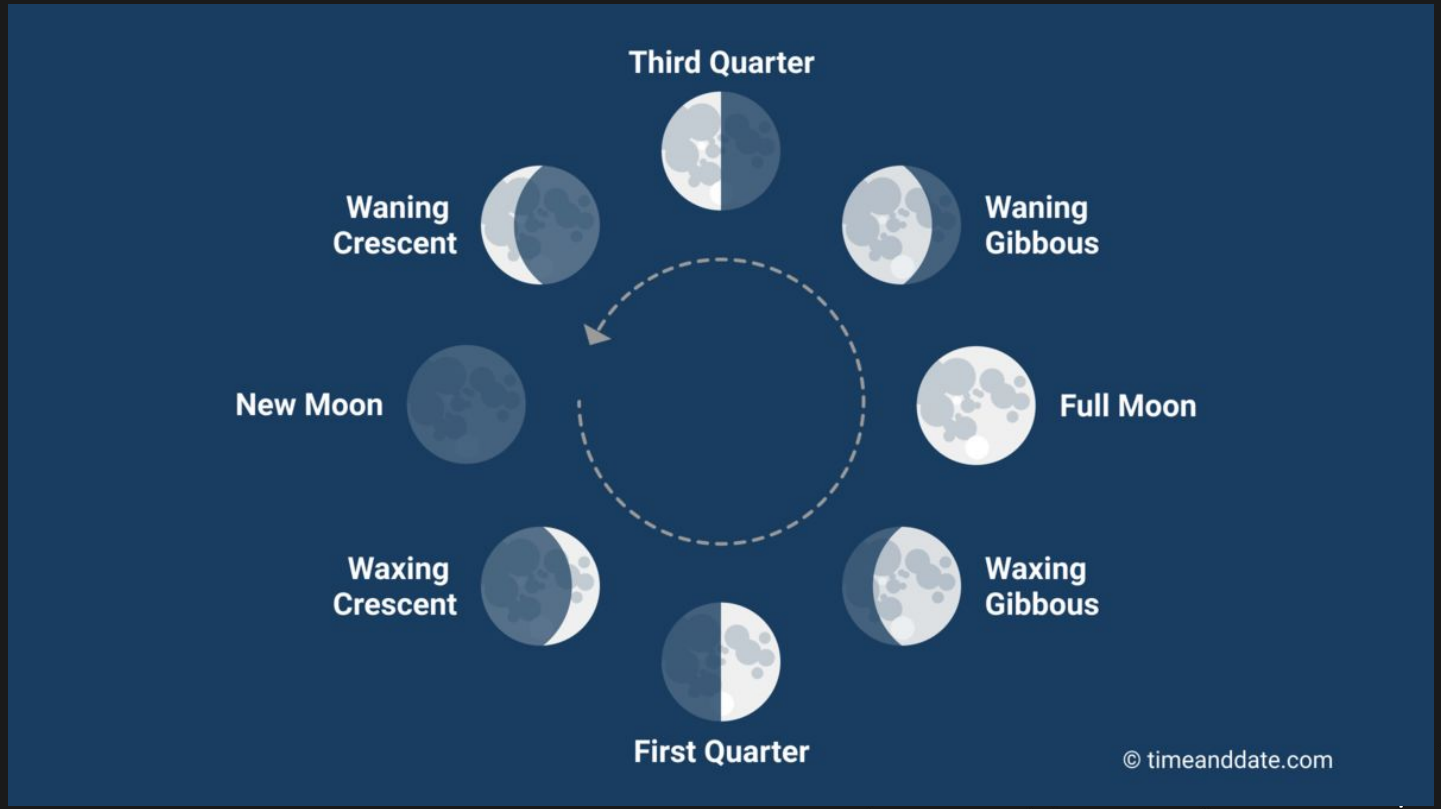
```
if max_quality == 0 and 7 <= bortle <= 9:
    self.match.setText("Quality matches")
elif max_quality == 1 and 5 <= bortle < 7:
    self.match.setText("Quality matches")
elif max_quality == 2 and 3 <= bortle < 5:
    self.match.setText("Quality matches")
elif max_quality == 3 and 1 <= bortle < 3:
    self.match.setText("Quality matches")
else:
    match_text = "Quality of", qual, "does not match bortle rating qual of", bortle_qual
    self.match.setText(str(match_text))
```

Moon

- Allow the user to crop the image into a rectangle with the edges of the moon bordering the rectangle
- Convert image to black and white, and use ratio of black and white pixels to assign a phase
- Also take input of whether left or right side is visible to assign a phase

```
if ratio <= 0.03:
    phase = "New Moon"
elif 0.03 < ratio <= 0.37 and moon_side == 'r':
    phase = "Waxing Crescent"
elif 0.37 < ratio <= 0.41 and moon_side == 'r':
    phase = "First Quarter"
elif 0.41 < ratio <= 0.74 and moon_side == 'r':
    phase = "Waxing Gibbons"
elif ratio > 0.74:
    phase = "Full Moon - Beware of werewolves"
elif 0.41 < ratio <= 0.74 and moon_side == 'l':
    phase = "Waning Gibbons"
elif 0.37 < ratio <= 0.41 and moon_side == 'l':
    phase = "Last Quarter"
elif 0.03 < ratio <= 0.37 and moon_side == 'l':
    phase = "Waning Crescent"
```

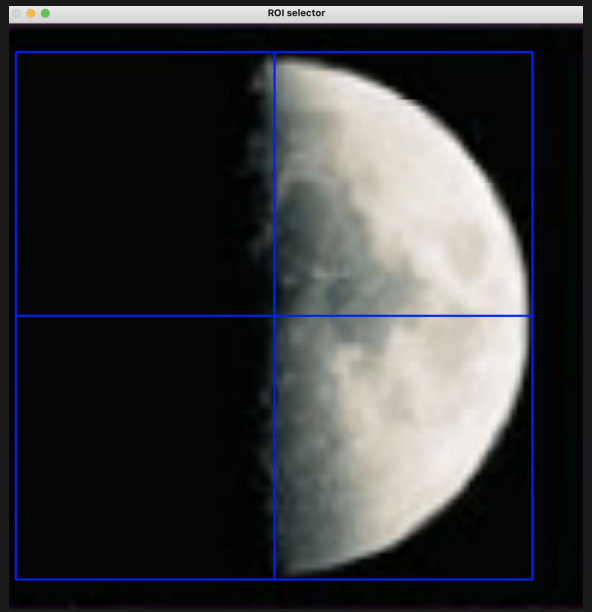
Moon Phases



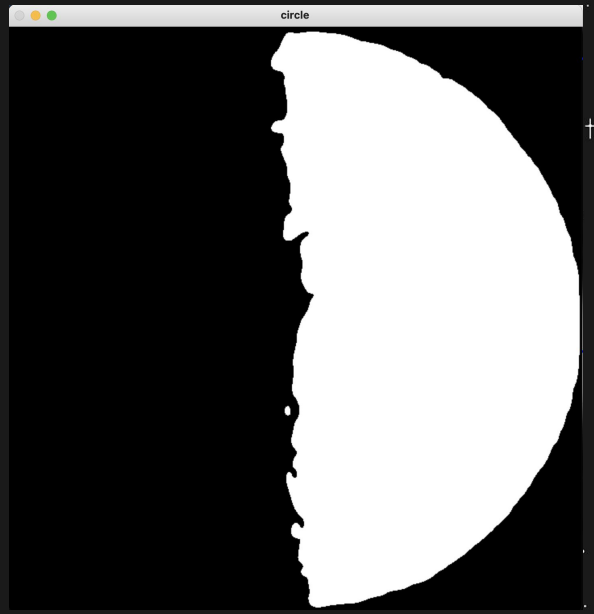
https://www.timeanddate.com/astronomy/moon/phases.html

Moon Visualized

Input Image



Crop Moon

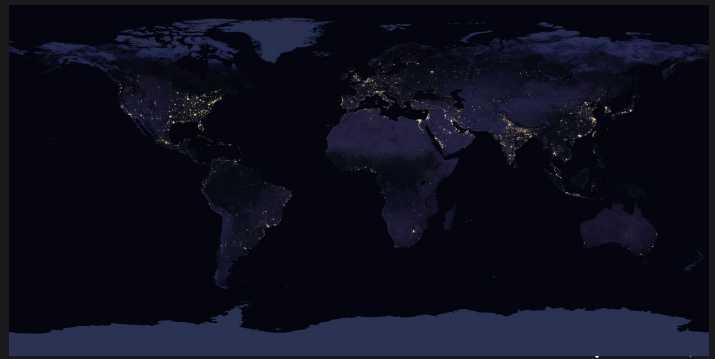


Phase of Moon:
First Quarter

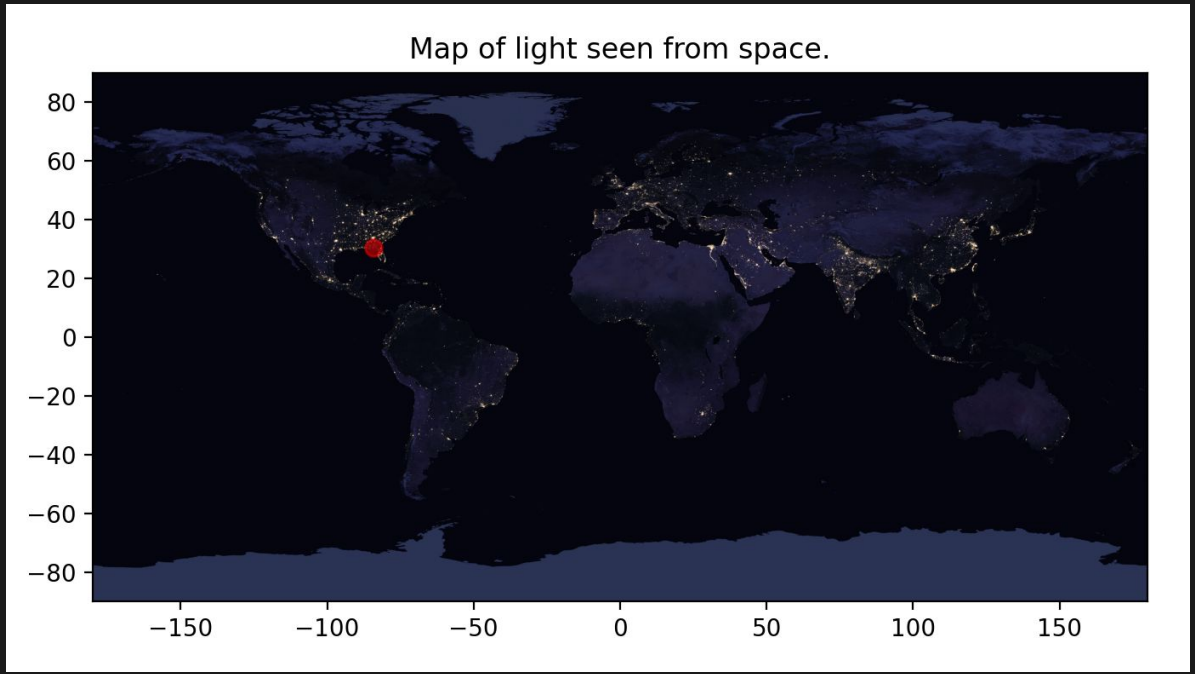
Map

- Using the longitude and latitude the user gave earlier, we can plot that point on a map.
- We use NASA's Black Marble to show how city lights are seen from space.
- Bright spots of the map will have more light pollution.

```
def plotmap():
    map = plt.imread('BlackMarble.jpeg')
    BBox = [-180, 180, -90, 90]
    lat = 30.2
    lon = -80.2
    fig, ax = plt.subplots(figsize=(8, 7))
    ax.scatter(lon, lat, zorder=1, c='r', s=50, alpha=.5)
    ax.set_title('Map of light seen from space.')
    ax.set_xlim(-180, 180)
    ax.set_ylim(-90, 90)
    ax.imshow(map, zorder=0, extent=BBox, aspect='equal')
    plt.show()
```



Map Visualized



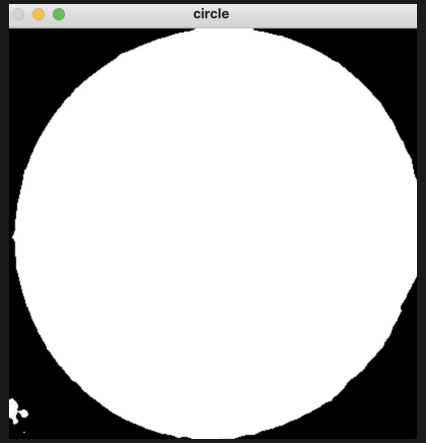
FSU's Coordinates 30.4419° N, 84.2985° W (-)

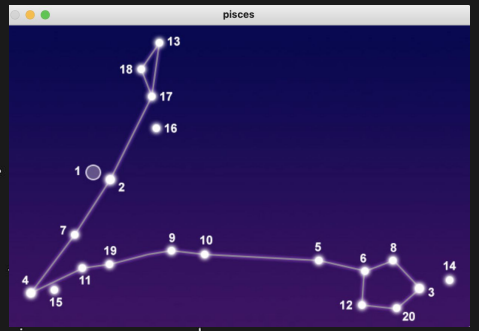
GUI Code

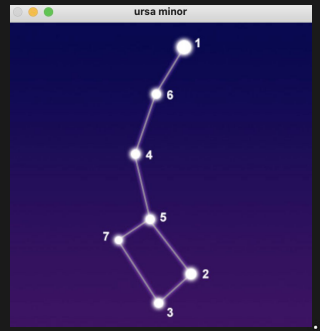
```
def retranslateUi(self, MainWindow):
    _translate = QtCore.QCoreApplication.translate
    MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
    self.label1.setText(_translate("MainWindow", "Python QT Astronomical Observations"))
    self.label2.setText(_translate("MainWindow", "Created By: Dylan Winer and Jake Kasitz"))
    self.label3.setText(_translate("MainWindow", "Press Button Below for Image"))
    self.button1.setText(_translate("MainWindow", "Browse"))
    self.label5.setText(_translate("MainWindow", "Output"))
    self.label6.setText(_translate("MainWindow", "Number of Stars:"))
    self.label7.setText(_translate("MainWindow", "Phase of Moon:"))
    self.label9.setText(_translate("MainWindow", "Constellations Possible:"))
    self.label8.setText(_translate("MainWindow", "Constellation Quality vs Bortle:"))
    self.label10.setText(_translate("MainWindow", "Bortle Scale (1-9):"))
    self.label12.setText(_translate("MainWindow", "Type of Sky:"))
    self.label.setText(_translate("MainWindow", "Planet Idea Credit: Katerina Patounakis"))
    self.label13.setText(_translate("MainWindow", "Latitude:"))
    self.label14.setText(_translate("MainWindow", "Longitude:"))
    self.label15.setText(_translate("MainWindow", "Season:"))
    self.label16.setText(_translate("MainWindow", "Constellations Visible?:"))
    self.label17.setText(_translate("MainWindow", "Side of moon? (L/R/None):"))

    self.button1.clicked.connect(self.browse_1)
    self.button5.clicked.connect(self.find_date)
    self.button6.clicked.connect(self.analyze_const)
```

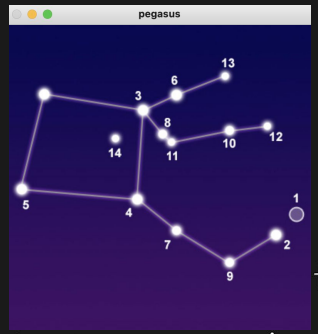
```
def browse_1(self, astronomer):
    self.open_dialog_box()

def browse_2(self, astronomer):
    self.open_dialog_box()

def open_dialog_box(self):
    filename = QFileDialog.getOpenFileName()
    print(filename)
    path = filename[0]
    print(path)
    global newname
    newname = os.path.basename(path)
    print("Newname", newname)

def find_date(self):
    global coords
    global lati
    global longi
    global moon_side
    lati = str(self.latitude.text())
    if lati[0] == '-':
        lati = lati * -1
        lati = float(lati)
    else:
        lati = float(lati)
    longi = float(self.longitude.text())
    coords = (lati, longi)
    date = str(self.date.text())
    moon_side = str(self.side.text())
    self.astronomer(newname, coords, date, moon_side)
```

Everything Combined

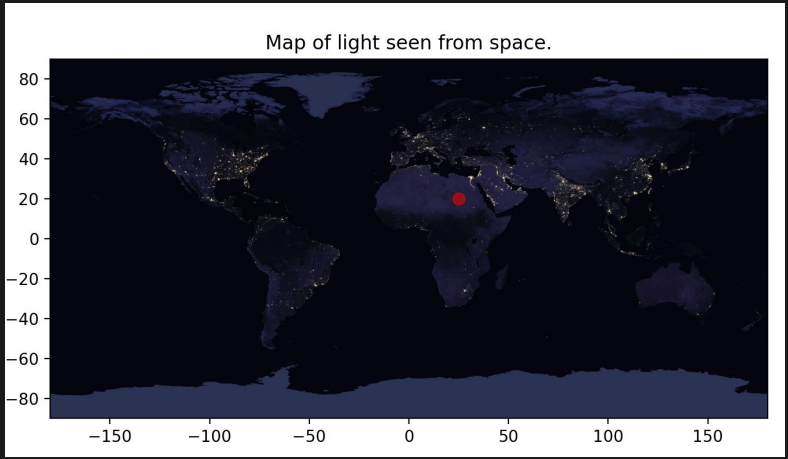


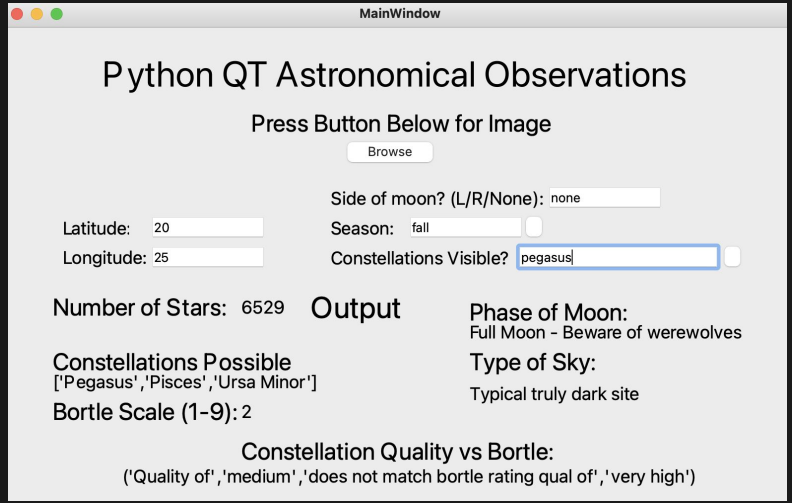






Everything Combined





References

- https://www.sciencea-z.com/science/resource/SL_Gr_5_Earth,_Sun,_and_Stars_L5_all_printable_resources.pdf
- https://www.windows2universe.org/the_universe/Constellations/south_constellations.html
- https://www.astronomytrek.com/star-constellations-brightest-stars/
- https://grove.ccsd59.org/wp-content/uploads/sites/10/2015/10/2.1a-How-The-Moon-Phases-Workpdf.pdf
- https://viirsland.gsfc.nasa.gov/Products/NASA/BlackMarble.html
- https://www.almanac.com/astronomy/moon/calendar