Practice 2: Script writing

Camila Cárdenas Uribe*
Universidad de Antofagasta, Antofagasta, Chile

(Dated: June 7, 2025)

This report presents the solutions implemented for Practice 2 of the course, which focuses on data processing and script programming based on what was learned during the lecture sessions. It includes three tasks: converting CSV light curve files using Bash, classifying stellar spectra based on temperature, and calculating the Julian day from a given date using Python. Each task demonstrates the application of basic programming and algorithms, as well as mastery of file and parameter inputs.

I. INTRODUCTION

The objective of this lab is to apply tools to modify files such as CSVs and parameter inputs provided by a user from the environment. We used Bash scripts for file manipulation and Python for sorting input parameters such as temperature and algebraic calculations such as the Julian date. Python was used for its simplicity and widespread adoption, especially for tasks involving numerical calculations, user interaction, and algorithmic logic [2].

Here, as in Lab 1, it is important to consult light curve data obtained by the Transiting Exoplanet Survey Satellite (TESS). A space telescope developed by NASA and launched in April 2018. Its primary mission is to photometrically map approximately 85% of the sky to detect exoplanets using the transit method [3], which identifies planets as they pass in front of their stars, causing small decreases in their brightness.

II. METHODOLOGY AND RESULTS

This section describes the solutions for tasks 1 to 3 proposed for this practice with a detailed description of the codes used and an overview of the responses provided by the program.

A. Light Curve .csv to .lc

To automate the processing of CSV files from Practice 1, we wrote a Bash script that:

1. Changes the delimiter from commas to spaces.
2. Extracts only the relevant columns for plotting light curves: `TIME` and `PDCSAP_FLUX`.
3. Renames the file extension from `.csv` to `.lc`.

```
#!/bin/bash

# Loop through all files ending in .csv
for archive in *.csv; do
  # Create a new file name
  # with a .lc extension
  new_archive="${archive%.csv}.lc"

  # Replace commas with spaces
  awk -F',' '

  # Extract TIME and PDCSAP_FLUX
  BEGIN {
   OFS = " "
  }
  NR==1 {
     for (i = 1; i <= NF; i++) {
       if ($i=="TIME") time_col=i
       if ($i=="PDCSAP_FLUX") flux_col=i
     }
  }
  {
     print $time_col, $flux_col
  }
  ' "$archive" > "$new_archive"
  echo "File: $archive -> $new_archive"
done
```

* Institutional email: camila.cardenas.uribe@uamail.cl

This Bash script automates the processing of .csv files containing light curve data worked on in Practice 1. First, it loops through all files in the current directory ending in .csv, which in this case are located in the CSV_File folder of the repository [1]. For each file, it creates a new name by changing the extension to .lc. Then, it uses awk to modify the file content: it sets the comma (',') as the field delimiter ('-F',''), and defines the space ('" "') as the new output separator ('OFS'). By default, awk separates output fields with spaces, but defining it explicitly with the Begin function ensures that even if the system has a different configuration, the extracted values (in this case, the TIME and PDCSAP_FLUX columns) will be written separated by a single space instead of commas or another delimiter. In awk, the special variable NR stands for "Number of Records", that is, the current line number being processed while NF stands for "Number of Fields" and represents the number of columns or "fields" in the current line, determined by the delimiter that has been defined [1]. In the first row of the file (containing the headers), identify the positions of the columns named 'TIME' and 'PDCSAP_FLUX'. From there, print only the values corresponding to these two columns in each row, removing the others, and save the result in the new .lc file. Finally, print a message to the terminal indicating that the file was processed successfully.

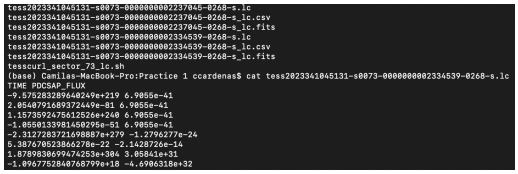

FIG. 1. Output of .lc files restricted to the columns associated with the light curve plot: TIME and PDCSAP_FLUX.

B. Spectral classification

Stars can be classified spectrally according to their surface temperature. Spectral classes are represented by the letters O, B, A, F, G, K, and M, ordered from highest to lowest temperature in Kelvin, such that:

- O: 30000 - 60000
- G: 5000 - 6000
- B: 10000 - 30000
- K: 3500 - 5000
- A: 7500 - 10000
- M: 2000 - 3500
- F: 6000 - 7500

In this activity, you are asked to implement a Python program that, given a temperature value in Kelvin, returns the corresponding spectral class or an error message if the temperature does not fall within a known range.

```
def get_spectral_class(temp):
    if 30000 <= temp <= 60000:
        return "O"
    elif 10000 <= temp < 30000:
        return "B"
    elif 7500 <= temp < 10000:
        return "A"
    elif 6000 <= temp < 7500:
        return "F"
    elif 5000 <= temp < 6000:
        return "G"
    elif 3500 <= temp < 5000:
        return "K"
    elif 2000 <= temp < 3500:
        return "M"
    else:
        return None

try:
    temp=int(input("Star's T in K:"))
    spectral_class=get_spectral_class(temp)

    if spectral_class:
        print("Spectral class:")
        print(spectral_class)
    else:
        print("T out of range")
except ValueError:
    print("Invalid input")
```

This function takes a temperature `temp` in Kelvin as an argument and evaluates its range to return the letter corresponding to the spectral class. The user is prompted to enter a temperature using the `input` function. The value is converted to an integer using `int()`. If the temperature does not fall within either of these ranges, `None` is returned and a message is printed indicating that the temperature is out of range. If

[1] `https://github.com/CamilaCU1613/Astroinformatics_Practice.git`

the input is not a number, a `ValueError` exception is thrown and a message is displayed indicating that the input is invalid or misspelled.

```
def get_spectral_class(temp):
    if 30000 <= temp <= 60000:
        return "O"
    elif 10000 <= temp < 30000:
        return "B"
    elif 7500 <= temp < 10000:
        return "A"
    elif 6000 <= temp < 7500:
        return "F"
    elif 5000 <= temp < 6000:
        return "G"
    elif 3500 <= temp < 5000:
        return "K"
    elif 2000 <= temp < 3500:
        return "M"
    else:
        return None

try:
    temp = int(input("Enter the star's temperature in Kelvin: "))
    spectral_class = get_spectral_class(temp)

    if spectral_class:
        print(f"The spectral class is {spectral_class}.")
    else:
        print("Temperature out of known spectral class range.")
except ValueError:
    print("Invalid input. Please enter a numeric temperature.")
```

```
Enter the star's temperature in Kelvin:  for history. Search history with c-↑/c-↓
```

FIG. 2. Visualization in jupyter notebook.

It is also possible to view it as a script using "python + file.py" to execute it:

```
(base) Camilas-MacBook-Pro:Practice 1 ccardenas$ python spectral_class.py
Enter the star's temperature in Kelvin: 2567
The spectral class is M.
(base) Camilas-MacBook-Pro:Practice 1 ccardenas$ python spectral_class.py
Enter the star's temperature in Kelvin: 567000000000
Temperature out of known spectral class range.
(base) Camilas-MacBook-Pro:Practice 1 ccardenas$ python spectral_class.py
Enter the star's temperature in Kelvin: abecg
Invalid input. Please enter a numeric temperature.
```

FIG. 3. Results for the classification in the three cases.

C. Julian date

The Julian day is a continuous way of counting days from a reference date. To calculate it from a calendar date (year, month, and day), you can use an approximate formula given by the statement. Implementing this in Python:

```
def julian_day(year:int,month:int, day:int):
    # Adjustment for January and February
    if month == 1 or month == 2:
        month += 12
# January: 13, February: 14
        year -= 1

    julian = (36525*year)//100+(306001*
        (month +1))//10000+day+1720981
    return julian


try:
    # Ask the user for year/month/day
    year = int(input("Enter year:"))
    month = int(input("Enter month"))
    day = int(input("Enter day:"))

    j_day = julian_day(year, month, day)
    print(f"The Julian day is: {j_day}")

except ValueError:
    print("Invalid input.")
```

As in the previous case, we can view it in Jupyter notebook and from the execution of a script. As in the previous case, we can view it in Jupyter Notebook and from the execution of a script. Where the solution for 07/06/2008 is: Julian day equal to 2454628

```
def calculate_julian_day(year:int, month:int, day:int):
    # Adjustment for January and February
    if month == 1 or month == 2:
        month += 12  # January → 13, February → 14
        year -= 1

    julian = (36525 * year) // 100 + (306001 * (month + 1)) // 10000 + day + 1720981
    return julian

try:
    # Ask the user for year, month, and day as integers
    year = int(input("Enter year (e.g., 2008): "))
    month = int(input("Enter month (1=Jan-12=Dec): "))
    day = int(input("Enter day of the month: "))

    julian_day = calculate_julian_day(year, month, day)
    print(f"The Julian day is: {julian_day}")

except ValueError:
    print("Invalid input. Please enter integer values for year, month, and day.")
```

```
Enter year (e.g., 2008):  2008
Enter month (1=Jan-12=Dec):  6
Enter day of the month:  7
The Julian day is: 2454624
```

FIG. 4. Visualizing the results in Jupyter Notebook after input.

```
(base) Camilas-MacBook-Pro:Practice 1 ccardenas$ python julian_day.py
Enter year (e.g., 2008): 2008
Enter month (1=Jan-12=Dec): 6
Enter day of the month: 7
The Julian day is: 2454624
```

FIG. 5. Results for the date established within the task.

III. CONCLUSION

This practice integrated fundamental scripting tools and Python code. In Task 1, shell scripting commands were applied to transform CSV files derived from Transiting Exoplanet Survey Satellite (TESS) observations into .lc formats for light curve analysis. This code is located in the repository folder called LC_File_LightCurve along with the corresponding .lc files. Task 2 reinforced stellar spectral classification by creating a .py program that associates temperature ranges with spectral classes, incorporating user input validation. This code is located in the Spectral_Class

folder. Finally, in Task 3, inside the Julian_day folder, an algorithm was implemented to calculate the Julian day from calendar dates, obtaining the Julian date 2454624 for June 7, 2008. Everything is documented in the repository associated with the task [2].

[1] GeeksforGeeks (2025). AWK command in Unix/Linux with examples.

[2] Python.org (2025). Welcome to python. *https://www.python.org/about/*.

[3] Ricker, G. R., Latham, D. W., Vanderspek, R. K., Ennico, K. A., Bakos, G., Brown, T. M., Burgasser, A. J., Charbonneau, D., Clampin, M., Deming, L. D., Doty, J. P., Dunham, E. W., Elliot, J. L., Holman, M. J., Ida, S., Jenkins, J. M., Jernigan, J. G., Kawai, N., Laughlin, G. P., Lissauer, J. J., Martel, F., Sasselov, D. D., Schingler, R. H., Seager, S., Torres, G., Udry, S., Villasenor, J. N., Winn, J. N., and Worden, S. P. (2018). Transiting Exoplanet Survey Satellite (TESS). 215:450.06.

[2] `https://github.com/CamilaCU1613/Astroinformatics_Practice.git`