SASTA OLIPHANT SCIENCE AWARDS

SOUTH AUSTRALIAN SCIENCE TEACHERS ASSOCIATION

Prize Winner

Programming, Apps & Robotics

Year 7-8

Riley Lorenz

Pedare Christian College

Intro & Background
For my Oliphant Science Awards entry this year, I decided to choose the programming category. This was because I had entered programs the last few years, and have adequate skills in multiple programming languages. I decided to choose Python because of its flexibility and relative simplicity. I decided to create a ‘space calculator’ because they can be useful learning aids and fun to use. I tried to keep it simpler than my other entries but was still faced with numerous difficulties. I was unable to complete all categories, but it can always be expanded in the future. I had to accept that high-standard programs take a long time and that I would have to complete as much as possible before the deadline.

I find the vastness of space interesting, and I hope that through my project others will experience the enjoyment that comes with learning science well.

I have been coding for about 4 years, but only used an advanced language in the last 2-2.5 years. I attempted to use the Unity game engine for my last project, but the difficulty was much higher than I expected. I thought about using Unity this year but decided to choose python, as it has a large amount of functionality, but is also relatively simple.

“Neptune is 26,470 times farther away than New York.” That is just one example of a fact you could learn from my Super Space Calculator. Currently, you can calculate how long it will take to get to different planets (and the Moon/Sun) at different speeds. Because of the immense distances in space, I had to deal in Km/S instead of Km/h. You can also calculate the relative distance of locations (Celestial bodies).

My program was written in python, on an Asus Chromebook with a 10th gen Intel Core i5 processor. While the computer is quite high-end, the code itself should run on any computer. I used a web compiler called Programiz, which allowed me to rapidly test my code without having to download IDLE.

Limitations
While I was unable to include all celestial bodies, I included all the planets, The Sun and The Moon. I was unable to create a GUI, but I have designed some examples: The Super Space Calculator GUI.pdf . Because of hardware limitations, I was unable to add everything I wanted. Because I am using an online editor, the internet is prone to fluctuations and sometimes the program will not run. This is not because of the code, so please try running the program again. While I noticed it, I didn’t have enough time to fix the error where when you compare distances if the first is shorter than the second, you get a zero error.

Aim
The aim of my project this year was as a learning aid for primary school classrooms. The target age demographic was 10 to 12-year-olds. It allows teachers to relax for a while as students discover facts about the Solar System. It allows people to learn in a fun and innovative way. Using comparisons is a great way to visualise data, and I hope this project is no exception. While this is just a preliminary copy, I believe with a bit of time, my program could evolve into a fun learning tool, used in classrooms everywhere. With some work,

science could become the favourite subject of many more students. This could be used in many places as well, kids could use it to learn facts to impress their friends, or to sound good in class.

Running the Program

First, copy-paste the python code into the text input in either IDLE or www.programiz.com/python-programming/online-compiler/, hit run, then press either 1, 2, or 3 to select a calculator mode. Hit enter to lock your selection. Any type of computer should be sufficient to run the program. Then follow the prompts given, before hitting enter again. If at any time the program stops or doesn't do anything, you can restart it by clicking run again.

Glossary

GUI - Graphical User Interface, or the buttons and other representations that you interact with on the screen.

Input - Whatever you type into the computer.

Print - In programming terms, to print something is to show it on the screen.

Variable - A set of information that the computer remembers so that it can be used elsewhere e.g. A password or username, a web address, an IP address, etc

Note: ## in programming tells the computer to ignore it. Just copy and paste everything under the heading into the input box

Python Code

```
##This section tells the program what software libraries it needs to access.
import numpy as np

##This section sets the values of variables.
#Kms -
moonDistance = 384400
sunDistance = 149597870
mercDistance = 77000000
venusDistance = 40000000
marsDistance = 350820000
jupiterDistance = 778000000
saturnDistance = 1400000000
uranusDistance = 2900000000
neptuneDistance = 4500000000

lightSpeed = 300000
# carSpeed = 0.016668

NYDistance = 17000
londonDistance = 16200
```

```
#Names
sunNames = ["sun", "the sun", "The Sun", "the Sun", "Sun"]
moonNames = ["moon", "the moon", "The Moon", "the Moon", "Moon"]
mercuryNames = ["Mercury", "mercury"]
venusNames = ["Venus", "venus"]
marsNames = ["Mars", "mars"]
jupiterNames = ["Jupiter", "jupiter", "Jupit", "jupit"]
saturnNames = ["Saturn", "saturn", "Sat", "sat"]
uranusNames = ["Uranus", "uranus",  "Uran", "uran"]
neptNames = ["Neptune", "neptune",  "Nept", "nept"]

NYNames = ["NY", "ny", "New York", "new york", "New york", "new York"]
londonNames = ["London", "london"]

lghtSpdNames = ["light speed", "Light speed", "Light Speed", "Lightspeed", "lightspeed"]
# carSpdNames = ["car speed", "Car speed", "Car Speed"]

##This section collects and processes input from the user
def info():
    global place
    global speed
    global mode
    global size

    print("What mode would you like to use?")
    print("1. Speed Calculator")
    print("2. Relative Distance Calculator")
    print("3. Relative Diameter Calculator - Unavailable")
    mode=input(">")

    print("")
    print("Celestial Bodies Supported: ")
    print("The Sun, Mercury, Venus, The Moon, Mars, Jupiter, Saturn, Uranus, and Neptune.")
    print()
    print("Locations Supported: ")
    print("All celestial bodies, New York, London")


    ##This section collects extra information from the user, depending on the mode.
    if mode == "1":
        place=input("What celestial body would you like to go to? ")
        speed=input("What speed (km/s)? ")
        if speed in lghtSpdNames:
            speed = lightSpeed
            setPlace(place)
        else:
            setPlace(place)
    if mode == "2":
        placeDistance=input("What is one celestial body you want to compare? ")
        location2=input("What 'location' do you want to compare it to? ")
        setLocation(location2, placeDistance)

##This section calculates the results for section 1 using a simple division sum.
def setPlace(location):
        if location in sunNames:
            time = round(sunDistance / (int(speed)), 2)
            distance = sunDistance
        elif location in moonNames:
            time = round(moonDistance / (int(speed)), 2)
            distance = moonDistance
        elif location in mercuryNames:
            time = round(mercDistance / (int(speed)), 2)
            distance = mercDistance
        elif location in venusNames:
            time = round(venusDistance / (int(speed)), 2)
            distance = venusDistance
        elif location in marsNames:
            time = round(marsDistance / (int(speed)), 2)
            distance = marsDistance
        elif location in jupiterNames:
            time = round(jupiterDistance / (int(speed)), 2)
            distance = jupiterDistance
        elif location in saturnNames:
            time = round(saturnDistance / (int(speed)), 2)
            distance = saturnDistance
        elif location in uranusNames:
            time = round(uranusDistance / (int(speed)), 2)
            distance = uranusDistance
        elif location in neptNames:
            time = round(neptuneDistance / (int(speed)), 2)
            distance = neptuneDistance
        else:
            print("Location not found. Please restart the program.")
            raise SystemExit

        ##This section prints the information from section 1 to the screen.
        if mode == "1":
            timeMins = round(time / 60, 2)
            timeHours = round(timeMins / 60, 2)
            timeDays = round(timeHours / 24, 2)
            timeYears = round(timeDays / 365, 2)

            #print(time)
            #speedHours = speed * 60 * 60, 2
```

```
	print("At %s km/s it will take %s seconds or %s minutes or %s hours or %s days or
%s years." % (speed, time, timeMins, timeHours, timeDays, timeYears))



##This section changes variables based on user input.
def setLocation(location, placeDistance):
	if location in sunNames:
	    locationDistance = sunDistance
	    placeDistance2 = sunDistance
	elif location in moonNames:
	    locationDistance = moonDistance
	    placeDistance2 = moonDistance
	elif location in mercuryNames:
	    locationDistance = mercDistance
	    placeDistance2 = mercDistance
	elif location in venusNames:
	    locationDistance = venusDistance
	    placeDistance2 = venusDistance
	elif location in marsNames:
	    locationDistance = marsDistance
	    placeDistance2 = marsDistance
	elif location in jupiterNames:
	    locationDistance = jupiterDistance
	    placeDistance2 = jupiterDistance
	elif location in saturnNames:
	    locationDistance = saturnDistance
	    placeDistance2 = saturnDistance
	elif location in uranusNames:
	    locationDistance = uranusDistance
	    placeDistance2 = uranusDistance
	elif location in neptNames:
	    locationDistance = neptuneDistance
	    placeDistance2 = neptuneDistance
	elif location in NYNames:
	    locationDistance = NYDistance
	    placeDistance2 = NYDistance
	elif location in londonNames:
	    locationDistance = londonDistance
	    placeDistance2 = londonDistance
	else:
	    print("Location not found. Please restart the program.")
	    raise SystemExit

	if placeDistance in sunNames:
	    distance = sunDistance
	elif placeDistance in moonNames:
	    distance = moonDistance
```

```
        elif placeDistance in mercuryNames:
            distance = mercDistance
        elif placeDistance in venusNames:
            distance = venusDistance
        elif placeDistance in marsNames:
            distance = marsDistance
        elif placeDistance in jupiterNames:
            distance = jupiterDistance
        elif placeDistance in saturnNames:
            distance = saturnDistance
        elif placeDistance in uranusNames:
            distance = uranusDistance
        elif placeDistance in neptNames:
            distance = neptuneDistance
        else:
            print("Location not found. Please restart the program.")
            raise SystemExit


        ##This section prints the information from section 2 to the screen.
        timesFurther = round(distance / locationDistance)

        print("'%s' is %s times further away than '%s'." % (placeDistance, timesFurther,
location))


##This section runs all the code in the correct order.
def gui():
    pass

def main():
    gui()
    info()

main()

##End
```

BIBLIOGRAPHY

I had minimal external support. My parents helped me brainstorm ideas, and my peer Thomas helped me to debug my first error - I eventually realised it was due to logic operations. I also went to the Adelaide Planetarium, which helped inspire me.

2021, *Solar System Information Chart*, Chart, Adelaide Planetarium, viewed 5 June 2021

Wikipedia n.d., *Voyager 2*, viewed 15 July 2021, <https://en.wikipedia.org/wiki/Voyager_2>.