ME 5194: MIDTERM 1

Adrian Bakhtar

Problem 1 (10 Points):

Suppose you have a list of key–score tuples like the following:

[(‘sarah’,10), ('tom',8), (‘bob’,20), ('sarah',17), (‘tom’,12), ('bob',5),...]

Write a function that takes such a list as a parameter and prints out a table of average scores for each person.

In [ ]:

```
import numpy as np
import statistics as st
from tabulate import tabulate


def avg_score(lst):
    """
    Computes the average score for each person in a list of key-score tuple

    Parameters:
    -----------
    lst : list
        List of key-score tuples, each containing the name of the person an

    Returns:
    --------
    None

    """

    #create a table headers
    table = [["Name", "Average Score"]]

    #Creates empty dictionaries
    name_scores, name_avg = [{} for i in range(2)]

    for n,s in lst:

        #if n (name) appears in name_scores already
        if n in name_scores:

            #Append the score to the name
            name_scores[n].append(s)

        #Otherwise add the name to the dictionary along with the score
        else:
            name_scores[n] = [s]

    for key in name_scores:

        #Take the average score for each person and create a row
        row = [key, st.mean(name_scores[key])]
        table.append(row)

    # use tabulate to print the table
    print(tabulate(table, tablefmt="fancy_grid"))


name_score = [('sarah',10), ('tom',8), ('bob',20), ('sarah',17), ('tom',12)

avg_score(name_score)
```

Problem 2 (15 Points):

Have you ever tried to speak “Pig Latin”? Write a function named toPigLatin that takes a normal English sentence in the form of a string as a parameter and translates it into Pig Latin. The function should return a string containing the Pig Latin version of the original sentence. Here’s a reference for Pig Latin:

https://www.wikihow.com/Speak-Pig-Latin (https://www.wikihow.com/Speak-Pig-Latin)

In [ ]:

```
import enchant as en


def is_vowel(l):
    '''
    Check if the given letter is a vowel.

    Parameters
    ----------
    l : str
        The letter to be checked

    Returns
    -------
    bool
        True if the letter is a vowel, False otherwise

    '''
    vowels = ['a', 'e', 'i', 'o', 'u']

    for v in vowels:
        if l.lower() == v: return True
    return False


def is_two_letter_sound(word):
    '''
    Check if the first two letters of a given word form one of the two-lett

    Parameters
    ----------
    word : str
        The word to be checked

    Returns
    -------
    bool
         True if the first two letters form a two-letter sound, False other

    '''
    two_sound = ['ch', 'sh', 'th', 'ph', 'wh']
    first_two_l = word[:2].lower()
    for lets in two_sound:
        if first_two_l == lets: return True
    return False


def is_compound(word):
    '''
    Check if a given word is a compound word using the check() function of

    Parameters
    ----------
    word : str
        The word to be checked

    Returns
    -------
    list
        A list containing the two words if the given word is a compound wor

    '''

    #Creating an English Dicionrtary (not the data type) that has access to
    eng_dict = en.Dict("en_US")
    word_len = len(word)

    for i in range(1, word_len):

        #Splits the potential compound word along different indices
        first_word = word[:i]
        second_word = word[i:]

        #If the check returned True for the first word and its length is gr
        if eng_dict.check(first_word) and len(first_word) > 1:

            #If the check returned True for the second word and its length
            if eng_dict.check(second_word) and len(second_word) > 1:

                #If the verify_compound function returned True
                if verify_compound(first_word, second_word):
                    return [first_word, second_word]

    return None



def verify_compound(first_word, second_word):
    '''
    Prompts the user to verify if the compound word was correctly identifie

    Parameters
    ----------
    first_word : str
        The first part of the compound word
    second_word : str
        The second part of the compound word

    Returns
    -------
    bool
        True if the user verifies the compound word, False if not

    '''
    while True:
        user_in = input(f'First Word:\t{first_word}\t\tSecond Word:\t{secon

        if user_in.lower() == 'y':
            return True
        elif user_in.lower() == 'n':
            return False
        else:
            print("Please make sure you have entered any one of the followi
```

```python
def pig_latin(sentence):
    '''
    Takes a sentence in English and converts it to Pig Latin and prints it

    Parameters
    ----------
    sentence : str
        The sentence to be converted to Pig Latin

    Returns
    -------
    None.

    '''

    #Splits the sentence string into a list of the words that comprise it
    word_list = sentence.split()
    new_word_list = []

    for word in word_list:

        #Checks to see if there are any compound words
        compound_words = is_compound(word)

        #If there are, add the two words to the new word list (extend allow
        if compound_words is not None:
            new_word_list.extend(compound_words)

        #If not, just append the single word from word_list
        else:
            new_word_list.append(word)

    for i, word in enumerate(new_word_list):

        #Checks if the word begins with a vowel
        if is_vowel(word[0]):
            new_word_list[i] = word + "-yay"

        #Checks if the word has the two-letter sound
        elif is_two_letter_sound(word):
            new_word_list[i] = word[2:] + word[:2] + '-ay'

        #Checks if the word begins with a consonant
        elif not is_vowel(word[0]):
            new_word_list[i] = word[1:] + '-' + word[0] + 'ay'
        else:
            print('Please make sure your sentence is comprised of letters o

    #Combines the words from the new word list that contains the pig latin
    new_sen = ' '.join(new_word_list) + '.'
    print(new_sen)



sen = "I would like to buy a hamburger"

pig_latin(sen)
```

This problem was a bit more involved that I initially thought. I installed the PyEnchant library to use as my English dictionary. For the compound word check, I would take the compound word (for example 'toothbrush') and would split it at different indices and use PyEnchant's check() function to see if two valid English words were produced. If it returned True for both words, then the word must be a compound word. However, it was returning True for single letters, so I had to make the minimum length be greater than 2 to use this. Additionally, the is_compound() function is not always correct. For example, 'atone' can be split up into 'at' and 'one', which are two valid words, but 'atone' itself is not a compound word. I was unsure of what to put in place to prevent this from happening, so I added that verify_compound() function to let the user decide before continuing.

Problem 3 (25 Points):

Create a Solar System class that is composed of planet objects (which are each instances of the planet class) and sun objects (instances of the sun class). The objects should have the following parameters:

Planets: name, radius, mass, radial distance from the sun
Suns: name, radius, mass, average surface temperature
Solar system: name, planet(s), sun(s)

Write methods for the solar system class that allow it to add and delete suns and planets, to calculate the overall mass of the solar system, to print a list of the planets in either of the following orders: relative distance from one of the sun(s), or relative mass (low to high). Discuss any of the concepts and approaches that were covered in the book that you've used in your solar system application and how they were useful/helpful as well as any challenges that you faced and how you overcame them.

Creating the classes for the planet, sun, and solar system

In [132]:

```
import ipywidgets as widgets
from IPython.display import display

class Celestial_Base_Class:
    """
    The base (parent) class for celestial objects.

    Attributes:
    -----------
    name : str
        The name of the celestial object.
    radius : float
        The radius of the celestial object in km.
    mass : float
        The mass of the celestial object in kg.
    """

    def __init__(self, name, radius, mass):
        self.name = name
        self.radius = radius
        self.mass = float(mass)

    def __str__(self):
        """
        Return the name of the celestial object when the object is printed.
        """
        return f"{self.name}"

class Planet(Celestial_Base_Class):
    """
    A subclass (child) of Celestial_Base_Class for planet objects.

    Attributes:
    -----------
    radial_distance : float
        The radial distance of the planet from its sun in km.
    """

    def __init__(self, name, radius, mass, radial_distance):
        """
        Initialize a Planet object.

        Parameters:
        -----------
        name : str
            The name of the planet.
        radius : float
            The radius of the planet in km.
        mass : float
            The mass of the planet in kg.
        radial_distance : float
            The radial distance of the planet from its star in km.
        """
        super().__init__(name, radius, mass)
        self.radial_distance = radial_distance

class Sun(Celestial_Base_Class):
    """
    A subclass (child) of Celestial_Base_Class for sun objects.

    Attributes:
    -----------
    avg_t_sur : float
        The average surface temperature of the sun in Kelvin.
    """

    def __init__(self, name, radius, mass, avg_t_sur):
        """
        Initialize a Sun object.

        Parameters:
        -----------
        name : str
            The name of the sun.
        radius : float
            The radius of the sun in km.
        mass : float
            The mass of the sun in kg.
        avg_t_sur : float
            The average surface temperature of the sun in Kelvin.
        """
        super().__init__(name, radius, mass)
        self.avg_t_sur = avg_t_sur

class Solar_System:
    """
    A class representing a solar system.

    Attributes
    ----------
    name : str
        The name of the solar system.
    planets : list
        A list containing planet objects.
    suns : list
        A list containing sun objects.
     """
    def __init__(self, name):
        """
        Initializes a new instance of the Solar_System class.

        Parameters
        ----------
        name : str
            The name of the solar system.

        Returns
        -------
        None.

        """
        self.name = name
        self.planets = []
        self.suns = []
    def __str__(self):
        """
        Returns the name of the solar system.

        Returns
        -------
        str
            The name of the solar system.

        """
        return f"{self.name}"

    def add_celestial(self, celestial_objs):
        """
        Adds celestial (Planet or Sun) objects to the solar system.

        Parameters
        ----------
        celestial_objs : object or list
            The celestial object/list to be added to the solar system.

        Returns
        -------
        None.

        """
        #Checks if input is a list
        if isinstance(celestial_objs, list):

            #If so, loop through, check if it is a Planet or Sun instance,
            for obj in celestial_objs:
                if isinstance(obj, Planet):
                    self.planets.append(obj)
                    print(f"Added planet {obj.name} to {self.name}")
                elif isinstance(obj, Sun):
                    self.suns.append(obj)
                    print(f"Added sun {obj.name} to {self.name}")
                else:
                    print(f"{obj} is not a valid celestial object")

        #Checks if input was a planet instance and adds to planet list if s
        elif isinstance(celestial_objs, Planet):
            self.planets.append(celestial_objs)
            print(f"Added planet {celestial_objs.name} to {self.name}")

        #Checks if input was a sun instance and adds to sun list of so
        elif isinstance(celestial_objs, Sun):
            self.suns.append(celestial_objs)
            print(f"Added sun {celestial_objs.name} to {self.name}")
        else:
            print(f"{celestial_objs} is not a valid celestial object")


    def delete_celestial(self, celestial_type, name):
        """
        Deletes a celestial object from the solar system.
        Parameters
        ----------
        celestial_type : str
            The type of celestial object to be deleted ("Planet" or "Sun").
        name : str
            The name of the celestial object to be deleted.

        Returns
        -------
        None.

        """
        #If the type was a Planet instance
        if celestial_type == "Planet":

            #Search the planets list to see if the name of the planet is th
            for planet in self.planets:
                #print(planet)
                if planet.name == name:
                    self.planets.remove(planet)
                    print(f"Removed planet {name} from {self.name}")
                    return
            print(f"No planet named {name} found in {self.name}")

        #If the type was a Sun instance
        elif celestial_type == "Sun":

            #Search the sun list to see if the name of the planet is there
            for sun in self.suns:
                #print(sun)
                if sun.name == name:
                    self.suns.remove(sun)
                    print(f"Removed sun {name} from {self.name}")
                    return
            print(f"No sun named {name} found in {self.name}")

        else:
            print(f"\n\n{celestial_type} is not a valid celestial type\nChe

    def get_total_mass(self):
        """
        Calculates the total mass of all the celestial objects (planets and

        Returns
        -------
        total_mass : float
            The sum of the masses of all the celestial objects in the solar
        """
        total_mass = 0

        #Loops through all of the planets in the list and adds their masses
        for planet in self.planets:
            total_mass += planet.mass

        #Loops through all of the suns in the list and adds their masses
        for sun in self.suns:
            total_mass += sun.mass
        print(f"Total mass of {milky_way.name}: {total_mass}")

        return total_mass

    def order_planets(self, option, direct):
        """
        Sorts the planets in the solar system based on a given option (by m
        and direction (high to low or low to high).

        Parameters
        ----------
        option : str
            The option to sort the planets by: 'Mass' and 'Relative distanc
        direct : str
            The direction to sort the planets by: 'High to low' and 'Low to

        Returns
        -------
        list :
            The list of planets sorted according to the given option and di
        """

        #If the user picked to sort by Mass
        if option == 'Mass':

            #Uses sorted() function to order the planets by mass. If direct
            return sorted(self.planets, key=lambda x: x.mass, reverse=(dire

        #If the user picked sort by Relative distance
        elif option == 'Relative distance':

            #Uses sorted() function to order the planets by radial distance
            return sorted(self.planets, key=lambda x: x.radial_distance, re
        else:
            print("Invalid option")
            return None
```

Creating an empty list to store all planets and suns that are created

In [133]:

```
planets_list = []
suns_list = []
solar_system_list = []
```

Creating required planet functions to use widgets

In [134]:

```
def on_planet_button_clicked(button):
    '''
    Creates a button that when pressed displays the parameters required for

    Parameters
    ----------
    button : widget button object
        The button object that triggers this function to be called

    Returns
    -------
    None
    '''
    #Made these global variables for ease
    global p_name_text, p_radius_text, p_mass_text, p_radial_distance_text

    #Creates a input box for the planet name
    p_name_text = widgets.Text(description='Name:')

    #Creates an input box for the planet radius
    p_radius_text = widgets.Text(description='Radius:')

    #Creates an input box for the planet mass
    p_mass_text = widgets.Text(description='Mass:')

    #Creates an input box for the planet's radial distance to the sun (form
    p_radial_distance_text = widgets.Text(description='Radial distance from

    #Creates the Planet submit button (for after the user has entered the r
    p_submit_button = widgets.Button(description='Create Planet')

    #When the planet submit button is clicked, call the on_planet_submit_bu
    p_submit_button.on_click(on_planet_submit_button_clicked)

    #Sets the planet inputs to be vertically aligned
    planet_box = widgets.VBox([p_name_text, p_radius_text, p_mass_text, p_r

    #Displays the planet inputs
    display(planet_box)

def on_planet_submit_button_clicked(button):
    '''
    Creates a planet object and adds it to the list of planets in the solar

    Parameters
    ----------
    button : widget button object
        The button object that triggers this function to be called

    Returns
    -------
    None
    '''

    planet = Planet(name=p_name_text.value, radius=p_radius_text.value, mas
                       radial_distance=p_radial_distance_text.value)
    planets_list.append(planet)
    print(f"New planet created: {planet}")


def on_sun_button_clicked(button):
    '''
    Creates a button that when pressed displays the parameters required for

    Parameters
    ----------
    button : widget button object
        The button object that triggers this function to be called

    Returns
    -------
    None
    '''

    #Made these global variables for ease
    global s_name_text, s_radius_text, s_mass_text, s_avg_t_sur

    #Creates a input box for the sun name
    s_name_text = widgets.Text(description='Name:')

    #Creates a input box for the sun radius
    s_radius_text = widgets.Text(description='Radius:')

    #Creates a input box for the sun mass
    s_mass_text = widgets.Text(description='Mass:')

    #Creates a input box for the sun average surface temperature (formatted
    s_avg_t_sur = widgets.Text(description='Average Surface Temperature:',

    #Creates the sun submit button (for after the user has entered the requ
    s_submit_button = widgets.Button(description='Create sun')

    #When the sun button is clicked, call the on_sun_submit_button_clicked(
    s_submit_button.on_click(on_sun_submit_button_clicked)

    #Sets the sun inputs to be vertically aligned
    sun_box = widgets.VBox([s_name_text, s_radius_text, s_mass_text, s_avg_

    #Displays the sun inputs
    display(sun_box)

def on_sun_submit_button_clicked(button):
    '''
    Creates a sun object and adds it to the list of suns in the solar syste

    Parameters
    ----------
    button : widget button object
        The button object that triggers this function to be called

    Returns
    -------
    None
    '''
    sun = Sun(name=s_name_text.value, radius=s_radius_text.value, mass=s_ma
                           avg_t_sur= s_avg_t_sur.value)

    suns_list.append(sun)
    print(f"New sun created: {sun}")


def on_order_button_clicked(button):
    """
    Sorts and displays the planets of a selected solar system object based
    ascending or descending order.

    Parameters
    ----------
    button : widget button object
        The button object that triggers this function to be called

    Returns
    -------
    None.

    """
    # Get the selected solar system object from dropdown menu
    selected_solar_system = solar_system_dropdown.value

    # Get the selected ordering option from dropdown menu
    order_option = order_option_dropdown.value

    # Get the selected ordering direction from dropdown menu
    order_direction = order_direction_dropdown.value

    #Calls the order_planets method of the solar system class and outputs b
    ordered_planets = selected_solar_system.order_planets(order_option, ord

    # Display the ordered planets
    print(f"\n\nThe ordered planets in {selected_solar_system.name} are:")
    for planet in ordered_planets:
        print(planet.name)
```

In [135]:

```
#Makes initial Create Planet button
planet_button = widgets.Button(description='Create Planet')

#When this initial create planet button is pressed, call the on_planet_butt
planet_button.on_click(on_planet_button_clicked)


#Makes initial Create Sun button
sun_button = widgets.Button(description='Create Sun')

#When this initial create sun button is pressed, call the on_sun_button_cli
sun_button.on_click(on_sun_button_clicked)


#Displays the planet and sun buttons
display(planet_button)
display(sun_button)
```

Create Planet

Create Sun

Name: Pythonica

Radius: 38232000

Mass: 3.683e22

Radial distance from sun: 1.2e13

Create Planet

```
New planet created: Pythonica
```

Name: Helios

Radius: 795700000

Mass: 1.2324e32

Average Surface Temperature: 6228

Create sun

```
New sun created: Helios
```

Name: Poptropica

Radius: 48232000

Mass: 2.683e22

Radial distance from sun: 1.1e13

Create Planet

```
New planet created: Poptropica
```

In [147]:

```python
#Demonstrates how to create a planet and sun object without the widgets
saturn = Planet('Saturn', 58232000, 5.683*(10^26), 1.4*(10^12))
sun = Sun('Big Ol Sun', 695700000, 1.9891*(10^30), 5778)

#Creating solay system object
milky_way = Solar_System('Milky Way')

#Adding solar system object to solar system list
solar_system_list.append(milky_way)

#Adding the planet and sun object list to the milky way solar system
milky_way.add_celestial(planets_list)
milky_way.add_celestial(suns_list)
milky_way.add_celestial(saturn)

#Calculates the total mass of the milky way solar system
total_mass = milky_way.get_total_mass()
```

```
Added planet Pythonica to Milky Way
Added planet Poptropica to Milky Way
Added sun Helios to Milky Way
Added planet Saturn to Milky Way
Total mass of Milky Way: 1.2324000006366001e+32
```

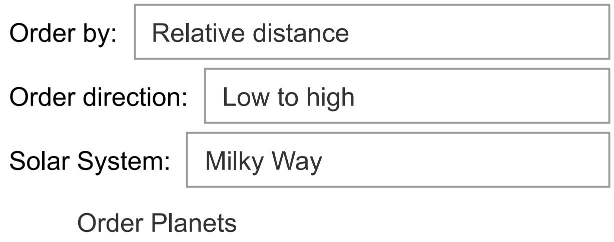
In [148]:

```python
#Deletes a planet/sun and shows the total mass decrease
milky_way.delete_celestial('Sun', 'Helios')
total_mass = milky_way.get_total_mass()
```

```
Removed sun Helios from Milky Way
Total mass of Milky Way: 6.366e+22
```

In [151]:

```
# Create the dropdown for selecting the order option
order_option_dropdown = widgets.Dropdown(options=['Mass', 'Relative distanc

# Create the dropdown for selecting the order direction
order_direction_dropdown = widgets.Dropdown(options=['High to low', 'Low to

# Create the dropdown for selecting the solar system
solar_system_dropdown = widgets.Dropdown(options= solar_system_list, descri

# Create the order button
order_button = widgets.Button(description='Order Planets')
order_button.on_click(on_order_button_clicked)

# Create the widget container for the ordering options
order_box = widgets.VBox([order_option_dropdown, order_direction_dropdown,

display(order_box)
```



Order by: Relative distance

Order direction: Low to high

Solar System: Milky Way

```
The ordered planets in Milky Way are:
Pythonica
Poptropica
Saturn


The ordered planets in Milky Way are:
Saturn
Poptropica
Pythonica
```

Explanation

I tried to make this problem have some type of user interface by using widgets. With this tool, I was able to create buttons, text inputs, and dropdown menus to make for a more intuitive way to create planets/suns and to order the planets. I tried to use widgets for the entire problem but was having some difficulties and ran into a time crunch. The code right now is a bit all over the place, but I tried to add headings to make the workflow easier to understand. If I had a little bit more time, I would have continued working on making this entire problem have the ability to be completed with widgets.

References

Problem 4 (50 Points):

Lists, dictionaries, and tuples can all be used to store and manipulate information. Discuss the following:

a. (5 pts) Describe the differences and similarities between these three types.

The table below offers a clearer visual on how lists, tuples, and dictionaries compare to each other. To summarize, lists and tuples are an ordered collection of items that can be sliceed and indexed. They also allow for duplicate items. For dictionaries, the order is not of importance. The values can be accessed using unique keys, meaning that there cannot be duplicate key values. Lists and dictionaries are mutable, meaning their contents can be manipulated in any way, while tuples are immutable. The way that the elements of each type are enclosed are different: [] for lists, () for tuples, and {} for dictionaries.

```
                          Table 1: Comparison of Lists, Tuples, and Dictionaries
```

Lists	Tuples	Dictionaries
Ordered collection of items	Ordered collection of items	Unordered collection of key-value pairs
Can be changed (mutable)	Cannot be changed (immutable)	Can be changed (mutable)
Elements are enclosed in square brackets (`[]`)	Elements are enclosed in parentheses (`()`)	Elements are enclosed in curly braces (`{}`)
Can be sliced and indexed	Can be sliced and indexed	Can be accessed using keys
Allows duplicate elements	Allows duplicate elements	Must have unique keys

Reference for formatting: Table section of [https://www.datacamp.com/tutorial/markdown-in-jupyter-notebook#tables (https://www.datacamp.com/tutorial/markdown-in-jupyter-notebook#tables)](https://www.datacamp.com/tutorial/markdown-in-jupyter-notebook#tables)

b. (10 pts) Give examples of the type of information that is well suited to each of these types, and also examples of what types of information would be ill-suited to each data type.

```
                          Table 2: Ideal use cases for lists, tuples, and dictionaries
```

Lists	Tuples	Dictionaries
Data that is frequently modified (keeping track of inventory)	Data that should not be modified (personal info)	Storing/retrieving data with clear relationship (Name: Phone Number

Lists	Tuples	Dictionaries
Ordered series of data (time signal, sensor data)	Collection of different data types (name-string, age-int, height-float)	Categorical data (Fruits: apples, bananas...)
Data that has duplicates (purchase orders)	Storing small, related data (personal info example again)	Collection of data results (Number of steps: #, move and settle time: #, average step time: #, standard deviation: #)

```
                  Table 3: Ill-suited use cases for lists,
tuples, and dictionaries
```

Lists	Tuples	Dictionaries
Data that needs to be secure (banking information)	Anything that requires data modification (inventory)	Data that doesn't have key:value relationship (position data)
Large amounts of data (might consider database or some other type)	Large, unrelated data	Data that needs to be ordered (time data)

c. (35 pts) Design an application that illustrates the use of each data type, using good interface and implementation methods (see page 203 of text for more information). Include the use of functions, classes and methods in your application. Also include the use of operator overloading, polymorphism, and inheritance. Include a brief description and discussion of how you designed your application and why you selected or utilized the different data types, objects, classes, methods, overloading, polymorphism, and inheritance approaches. Use a real-world application if possible or develop a new product or application – and include a good description of any new product or application you’re creating. NOTE: You have the option to create one or more applications to illustrate the use of these data types, however a deduction of two points will be made for every additional application.

In [12]:

```
import ipywidgets as widgets
from tabulate import tabulate

# Define the base class
class BudgetTracker:
    def __init__(self, description, cost):
        self.description = description
        self.cost = cost

    def add_expense(self, amount):
        self.total += amount

    def get_total(self):
        return self.total

# Define the inherited classes
class RentUtilities(BudgetTracker):
    def __init__(self, description, cost):
        super().__init__(description, cost)
        self.category = "Rent/Utilities"

class Groceries(BudgetTracker):
    def __init__(self, description, cost):
        super().__init__(description, cost)
        self.category = "Groceries"

class Insurance(BudgetTracker):
    def __init__(self, description, cost):
        super().__init__(description, cost)
        self.category = "Insurance"
class Miscellaneous(BudgetTracker):
    def __init__(self, description, cost):
        super().__init__(description, cost)
        self.category = "Miscellaneous"
class Chilling(BudgetTracker):
    def __init__(self, description, cost):
        super().__init__(description, cost)
        self.category = "Chilling"
class Hobbies(BudgetTracker):
    def __init__(self, description, cost):
        super().__init__(description, cost)
        self.category = "Hobbies"

class TotalExpenses:
    def __init__(self, name, account_number, phone_number):
        self.name = name
        self.account_info = (self.name, account_number, phone_number)

        self.expenses = {
            "Rent/Utilities": [],
            "Groceries": [],
            "Insurance": [],
            "Miscellaneous": [],
            "Chilling": [],
            "Hobbies": []
        }
    def add_expense(self, category, expense):
        self.expenses[category].append(expense)

    def get_total_expenses(self):
        total = 0
        for category in self.expenses:
            for expense in self.expenses[category]:
                total += expense.cost
        return total

    def __add__(self, other_obj):
        new_expenses = TotalExpenses(f"{self.name} and {other_obj.name}", (
                                     (self.account_info[2], other_obj.accou
        for category, expenses in self.expenses.items():
            new_expenses.expenses[category] += expenses

        for category, expenses in other_obj.expenses.items():
            new_expenses.expenses[category] += expenses
        return new_expenses


    def __str__(self):
        rows = []
        for category, expenses in self.expenses.items():
            if expenses:
                for expense in expenses:
                    rows.append([category, expense.description, f"${expense
            else:
                rows.append([category, "", ""])
        return tabulate(rows, headers=["Category", "Expense Description", "
```

Here is the code version to show that the functions are working as intended.

In [129]:

```
# create two TotalExpenses objects
expenses1 = TotalExpenses("Adrian", 554123326, 3122236879)
expenses2 = TotalExpenses("Sara", 324983124, 4330981234)

# create a new Chilling expense
chilling_expense = Chilling("Ant Man Tickets 1", 35.00)
chilling_expense2 = Chilling("Ant Man Tickets 2", 25.00)

rent_expense = RentUtilities("Monthly Rent", 999.99)

misc_expense2 = Miscellaneous('New Basketball Shoes', 79.99)
misc_expense2_2 = Miscellaneous('New Basketball', 29.99)

expenses1.add_expense(chilling_expense.category, chilling_expense)
expenses1.add_expense(rent_expense.category, rent_expense)

print(expenses1.account_info)
print(expenses1.get_total_expenses())
print(expenses1)

expenses2.add_expense(chilling_expense.category, chilling_expense2)
expenses2.add_expense(misc_expense2.category, misc_expense2)
expenses2.add_expense(misc_expense2_2.category, misc_expense2_2)
print(expenses2.account_info)
print(expenses2.get_total_expenses())
print(expenses2)
```

```
('Adrian', 554123326, 3122236879)
1034.99
╒════════════════╤══════════════════════╤══════════════╕
│ Category       │ Expense Description  │ Expense Cost │
╞════════════════╪══════════════════════╪══════════════╡
│ Rent/Utilities │ Monthly Rent         │ $999.99      │
├────────────────┼──────────────────────┼──────────────┤
│ Groceries      │                      │              │
├────────────────┼──────────────────────┼──────────────┤
│ Insurance      │                      │              │
├────────────────┼──────────────────────┼──────────────┤
│ Miscellaneous  │                      │              │
├────────────────┼──────────────────────┼──────────────┤
│ Chilling       │ Ant Man Tickets 1    │ $35.00       │
├────────────────┼──────────────────────┼──────────────┤
│ Hobbies        │                      │              │
╘════════════════╧══════════════════════╧══════════════╛
('Sara', 324983124, 4330981234)
134.98
╒════════════════╤══════════════════════╤══════════════╕
│ Category       │ Expense Description  │ Expense Cost │
╞════════════════╪══════════════════════╪══════════════╡
│ Rent/Utilities │                      │              │
├────────────────┼──────────────────────┼──────────────┤
│ Groceries      │                      │              │
├────────────────┼──────────────────────┼──────────────┤
│ Insurance      │                      │              │
├────────────────┼──────────────────────┼──────────────┤
│ Miscellaneous  │ New Basketball Shoes │ $79.99       │
├────────────────┼──────────────────────┼──────────────┤
│ Miscellaneous  │ New Basketball       │ $29.99       │
├────────────────┼──────────────────────┼──────────────┤
│ Chilling       │ Ant Man Tickets 2    │ $25.00       │
├────────────────┼──────────────────────┼──────────────┤
│ Hobbies        │                      │              │
╘════════════════╧══════════════════════╧══════════════╛
```

```
In [130]: # add the objects together
          total_expenses = expenses1 + expenses2

          # print the result
          print(total_expenses)
          print(total_expenses.name)
          print(total_expenses.get_total_expenses())
          print(total_expenses.account_info)
```

```
╒════════════════╤══════════════════════╤══════════════╕
│ Category       │ Expense Description  │ Expense Cost │
╞════════════════╪══════════════════════╪══════════════╡
│ Rent/Utilities │ Monthly Rent         │ $999.99      │
├────────────────┼──────────────────────┼──────────────┤
│ Groceries      │                      │              │
├────────────────┼──────────────────────┼──────────────┤
│ Insurance      │                      │              │
├────────────────┼──────────────────────┼──────────────┤
│ Miscellaneous  │ New Basketball Shoes │ $79.99       │
├────────────────┼──────────────────────┼──────────────┤
│ Miscellaneous  │ New Basketball       │ $29.99       │
├────────────────┼──────────────────────┼──────────────┤
│ Chilling       │ Ant Man Tickets 1    │ $35.00       │
├────────────────┼──────────────────────┼──────────────┤
│ Chilling       │ Ant Man Tickets 2    │ $25.00       │
├────────────────┼──────────────────────┼──────────────┤
│ Hobbies        │                      │              │
╘════════════════╧══════════════════════╧══════════════╛
Adrian and Sara
1169.97
('Adrian and Sara', (554123326, 324983124), (3122236879, 4330981234))
```

Attempt to add widgets--succesful until trying to combine expense objects. Also, there is definitely a way to shorten all of these repetetive functions, I just could not figure it out in time.

In [121]:

```
def on_rent_utilities_button_clicked(button):
    '''
    Creates a button that when pressed displays the parameters required for

    Parameters
    ----------
    button : widget button object
        The button object that triggers this function to be called

    Returns
    -------
    None
    '''
    global description_text_ru, cost_text_ru, add_button_ru

    # Check if a current account has been selected

    description_text_ru = widgets.Text(description='Description:')
    cost_text_ru = widgets.FloatText(description='Cost:')

    add_button_ru = widgets.Button(description='Add Expense')
    add_button_ru.on_click(on_rent_utilities_add_clicked)

    #Sets the inputs to be vertically aligned
    rent_utilities_box = widgets.VBox([description_text_ru, cost_text_ru, a

    #Displays the inputs
    display(rent_utilities_box)

def on_rent_utilities_add_clicked(button):
    '''
    Creates a rent/utilities expense object and adds it to the list of expe

    Parameters
    ----------
    button : widget button object
        The button object that triggers this function to be called

    Returns
    -------
    None
    '''
    global description_text_ru, cost_text_ru

    rent_util = RentUtilities(description_text_ru.value, cost_text_ru.value
    current_account.add_expense(rent_util.category, rent_util)
    print(f"Rent/Utility Added: {description_text_ru.value}\tCost: ${cost_t

def on_groceries_button_clicked(button):
    '''
    Creates a button that when pressed displays the parameters required for

    Parameters
    ----------
    button : widget button object
        The button object that triggers this function to be called
    Returns
    -------
    None
    '''
    global description_text_gr, cost_text_gr, add_button_gr

    description_text_gr = widgets.Text(description='Description:')
    cost_text_gr = widgets.FloatText(description='Cost:')

    add_button_gr = widgets.Button(description='Add Expense')
    add_button_gr.on_click(on_groceries_add_clicked)

    #Sets the inputs to be vertically aligned
    groceries_box = widgets.VBox([description_text_gr, cost_text_gr, add_bu

    #Displays the inputs
    display(groceries_box)

def on_groceries_add_clicked(button):
    '''
    Creates a rent/utilities expense object and adds it to the list of expe

    Parameters
    ----------
    button : widget button object
        The button object that triggers this function to be called

    Returns
    -------
    None
    '''
    global description_text_gr, cost_text_gr

    gro = RentUtilities(description_text_gr.value, cost_text_gr.value)
    current_account.add_expense(gro.category, gro)
    print(f"Groceries Added: {description_text_gr.value}\tCost: ${cost_text

def on_insurance_button_clicked(button):
    '''
    Creates a button that when pressed displays the parameters required for

    Parameters
    ----------
    button : widget button object
        The button object that triggers this function to be called

    Returns
    -------
    None
    '''
    global description_text_ins, cost_text_ins, add_button_ins


    description_text_ins = widgets.Text(description='Description:')
    cost_text_ins = widgets.FloatText(description='Cost:')

    add_button_ins = widgets.Button(description='Add Expense')
    add_button_ins.on_click(on_insurance_add_clicked)

    #Sets the inputs to be vertically aligned
    insurance_box = widgets.VBox([description_text_ins, cost_text_ins, add_

    #Displays the inputs
    display(insurance_box)

def on_insurance_add_clicked(button):
    '''
    Creates a Insurance expense object and adds it to the list of expenses

    Parameters
    ----------
    button : widget button object
        The button object that triggers this function to be called

    Returns
    -------
    None
    '''
    global description_text_ins, cost_text_ins

    ins = RentUtilities(description_text_ins.value, cost_text_ins.value)
    current_account.add_expense(ins.category, ins)

    print(f"Insurance Added: {description_text_ins.value}\tCost: ${cost_tex

def on_miscellaneous_button_clicked(button):
    '''
    Creates a button that when pressed displays the parameters required for

    Parameters
    ----------
    button : widget button object
        The button object that triggers this function to be called

    Returns
    -------
    None
    '''
    global description_text_misc, cost_text_misc, add_button_misc


    description_text_misc = widgets.Text(description='Description:')
    cost_text_misc = widgets.FloatText(description='Cost:')

    add_button_misc = widgets.Button(description='Add Expense')
    add_button_misc.on_click(on_miscellaneous_add_clicked)

    #Sets the inputs to be vertically aligned
    miscellaneous_box = widgets.VBox([description_text_misc, cost_text_misc

    #Displays the inputs
    display(miscellaneous_box)

def on_miscellaneous_add_clicked(button):
    '''
    Creates a miscellaneous expense object and adds it to the list of expen

    Parameters
    ----------
    button : widget button object
        The button object that triggers this function to be called

    Returns
    -------
    None
    '''
    global description_text_misc, cost_text_misc

    misc = RentUtilities(description_text_misc.value, cost_text_misc.value)
    current_account.add_expense(misc.category, misc)

    print(f"Miscellaneous Added: {description_text_misc.value}\tCost: ${cos


def on_chilling_button_clicked(button):
    '''
    Creates a button that when pressed displays the parameters required for

    Parameters
    ----------
    button : widget button object
        The button object that triggers this function to be called

    Returns
    -------
    None
    '''
    global description_text_chil, cost_text_chil, add_button_chil


    description_text_chil = widgets.Text(description='Description:')
    cost_text_chil = widgets.FloatText(description='Cost:')

    add_button_chil = widgets.Button(description='Add Expense')
    add_button_chil.on_click(on_chilling_add_clicked)

    #Sets the inputs to be vertically aligned
    chilling_box = widgets.VBox([description_text_chil, cost_text_chil, add

    #Displays the inputs
    display(chilling_box)

def on_chilling_add_clicked(button):
    '''
    Creates a chilling expense object and adds it to the list of expenses i

    Parameters
    ----------
    button : widget button object
        The button object that triggers this function to be called
    Returns
    -------
    None
    '''
    global description_text_chil, cost_text_chil

    chill = RentUtilities(description_text_chil.value, cost_text_chil.value
    current_account.add_expense(chill.category, chill)

    print(f"chilling Added: {description_text_chil.value}\tCost: ${cost_tex

def on_hobbies_button_clicked(button):
    '''
    Creates a button that when pressed displays the parameters required for

    Parameters
    ----------
    button : widget button object
        The button object that triggers this function to be called

    Returns
    -------
    None
    '''
    global description_text_hob, cost_text_hob, add_button_hob


    description_text_hob = widgets.Text(description='Description:')
    cost_text_hob = widgets.FloatText(description='Cost:')

    add_button_hob = widgets.Button(description='Add Expense')
    add_button_hob.on_click(on_hobbies_add_clicked)

    #Sets the inputs to be vertically aligned
    hobbies_box = widgets.VBox([description_text_hob, cost_text_hob, add_bu

    #Displays the inputs
    display(hobbies_box)

def on_hobbies_add_clicked(button):
    '''
    Creates a hobbies expense object and adds it to the list of expenses in

    Parameters
    ----------
    button : widget button object
        The button object that triggers this function to be called

    Returns
    -------
    None
    '''
    global description_text_hob, cost_text_hob

    hobl = RentUtilities(description_text_hob.value, cost_text_hob.value)
    current_account.add_expense(hobl.category, hobl)
    print(f"hobbies Added: {description_text_hob.value}\tCost: ${cost_text_

def on_total_expenses_add_clicked(button):
    global total_expenses_name, account_number_text, phone_number_text

    total_expenses_name = widgets.Text(description='Name on Account:', styl
    account_number_text = widgets.Text(description='Account Number:', style
    phone_number_text = widgets.Text(description='Phone Number:', style={'d

    # Creates the Planet submit button (for after the user has entered the
    total_expenses_button = widgets.Button(description='Finalize Account')

    # Sets the inputs to be vertically aligned
    total_expenses_box = widgets.VBox([total_expenses_name, account_number_



    # When the planet submit button is clicked, call the on_planet_submit_b
    total_expenses_button.on_click(total_expenses_button_clicked)

    # Displays the inputs
    display(total_expenses_box)

    # Makes initial Rent/Utilities button
    ru_button = widgets.Button(description='Rent/Utilities')
    # When this initial Rent/Utilities button is pressed, call the on_rent_
    ru_button.on_click(on_rent_utilities_button_clicked)

    # Makes initial Groceries button
    gr_button = widgets.Button(description='Groceries')
    # When this initial Groceries button is pressed, call the on_groceries_
    gr_button.on_click(on_groceries_button_clicked)

    # Makes initial Insurance button
    ins_button = widgets.Button(description='Insurance')
    # When this initial Insurance button is pressed, call the on_insurance_
    ins_button.on_click(on_insurance_button_clicked)

    # Makes initial miscellaneous button
    misc_button = widgets.Button(description='Miscellaneous')
    # When this initial miscellaneous button is pressed, call the on_miscel
    misc_button.on_click(on_miscellaneous_button_clicked)

    # Makes initial chilling button
    chil_button = widgets.Button(description='Chilling')
    # When this initial chilling button is pressed, call the on_chilling_bu
    chil_button.on_click(on_chilling_button_clicked)

    # Makes initial hobbies button
    hob_button = widgets.Button(description='Hobbies')
    # When this initial hobbies button is pressed, call the on_hobbies_butt
    hob_button.on_click(on_hobbies_button_clicked)

    button_row = widgets.HBox([ru_button, gr_button, ins_button, misc_butto
    display(button_row)
```

```
def total_expenses_button_clicked(button):
    global total_expenses_name, account_number_text, phone_number_text, cur

    total_expenses = TotalExpenses(total_expenses_name.value, account_numbe
    current_account = total_expenses
    total_expense_list.append(current_account)
    print(f"Account Name: {total_expenses_name.value}\tAccount Number: {acc


# Define the function to run when the "Combine Expenses" button is clicked
def combine_expenses_button_click(button):
    # Create the dropdown menus with the available TotalExpense objects
    dropdown1 = widgets.Dropdown(options=total_expense_list, description="E
    dropdown2 = widgets.Dropdown(options=total_expense_list, description="E

    # Display the dropdown menus and button
    display(dropdown1, dropdown2, combine_button)

    # Get the selected TotalExpenses objects from the dropdown menus
    expense1 = dropdown1.value
    expense2 = dropdown2.value

    # Combine the expenses using the __add__ method
    combined_expenses = expense1 + expense2

    # Add the combined expenses to the list
    total_expense_list.append(combined_expenses)

    # Print a success message
    print(f"Successfully combined {expense1.name} and {expense2.name} into




def on_print_button_clicked(button):
    """
    Prints the string representation of the current account's TotalExpenses

    Parameters
    ----------
    button : widget button object
        The button object that triggers this function to be called

    Returns
    -------
    None
    """
    global current_account, total_expense_list

    if not current_account:
        print("Please select a current account first.")
        return

    print(current_account)
```

```
current_account = None
total_expense_list = []
```

In [131]:

```
total_expenses_button = widgets.Button(description='Add account', style={'d
total_expenses_button.on_click(on_total_expenses_add_clicked)
display(total_expenses_button)

print_button = widgets.Button(description='Expense Table', style={'descript
print_button.on_click(on_print_button_clicked)
display(print_button)



# Create the button widget
combine_button = widgets.Button(description="Combine Expenses")

# Attach the combine_expenses_button_click function to the button click eve
combine_button.on_click(combine_expenses_button_click)


# Display the "Combine Expenses" button
display(combine_button)
```

Rent/Util… Groceries Insurance Miscella… Chilling Hobbies

```
Account Name: Adrian    Account Number: 234512123       Phone Numbe
r: 3122345567
```

Description: Rent

Cost: 999.99

Add Expense

```
Rent/Utility Added: Rent        Cost: $999.99
```

Description: Aldi trips

Cost: 264.65

I am able to create multiple accounts, add items from the widget set, and print out the expense table. Issues come when I try to combine two expense objects. It just combines the first one to itself. Additionally, the drop downs do not have the correct contents (expense objects). I felt very close though, but maybe I was not.

Explanation of Program

The program I created is meant for someone to keep track of their expenses (length of time is not a factor here). I split up the expenses into 6 categories: rent/utilities, groceries, insurance, miscellaneous, chilling, and hobbies. I then created a base BudgetTracker class that takes in the description and cost, and keeps track of the total cost. All 6 categories inherit this parent class, since nothing much is expected to be different. I then created a TotalExpense class that takes in a name, account number, and phone number. I figured this should be unique to each person. This class combines all of these expense classes, similar to that solar system class. The TotalExpense class contains a dictionary, where the keys are these 6 expense categories. The values of this dictionary are the list of every instance of the expense categories. I figured duplicates would occur, since maybe you go to the store twice in one week. I created a tuple that stores these values, since they should probably not be altered. For operator overloading, I modified **add** to combine two TotalExpense objects, say to find the total expense of a couple or family. For the polymorphism, the add_expense() method is able to read in any of the expense category objects and add to the respective spot of the dictionary.

References

1. Widgets: https://towardsdatascience.com/bring-your-jupyter-notebook-to-life-with-interactive-widgets-bc12e03f0916 (https://towardsdatascience.com/bring-your-jupyter-notebook-to-life-with-interactive-widgets-bc12e03f0916)

Original Idea

Create a music editor. Can create mp3's of any YouTube video, then you would have the option to change the duration, (goal of) pitch, and speed of the song, as well as combine songs. Was able to successfuly download mp3 from YouTube link but ran into major brick wall when I tried to shorten the song.

In [ ]:

```
import os
from pytube import YouTube
from pydub import AudioSegment
os.environ['FFMPEG_PATH'] = "C:/Users/Adrian Bakhtar/anaconda3/Lib/site-pac

import os

class Song:
    def __init__(self, url, title, artist):
        self.url = url
        self.title = title
        self.artist = artist

    def url_to_mp3(self, folder, slicing=False, start_time=None, end_time=N
        # Create the folder if it doesn't exist
        folder_path = folder + '//' + self.title

        # Download the audio from YouTube
        yt = YouTube(self.url)
        audio_stream = yt.streams.filter(only_audio=True, file_extension='m
        out_file = os.path.join(folder_path, self.title + '.mp4')
        audio_stream.download(output_path=folder_path, filename=self.title)

        # Get the base filename without extension
        base_filename = os.path.splitext(self.title)[0]

        # Convert the audio to mp3 format
        new_file = os.path.join(folder_path, base_filename + '.mp3')
        if slicing:
            audio = AudioSegment.from_file(out_file)
            sliced_audio = self.slice_audio(audio, start_time, end_time)
            sliced_audio.export(new_file, format='mp3')
        else:
            sound = AudioSegment.from_file(out_file)
            sound.export(new_file, format='mp3')

        # Delete the original mp4 file
        os.remove(out_file)



    def time_to_ms(time_string):
        mm, ss = time_string.split(':')

        ms = (int(mm)*60 + int(ss))*1000
        return ms


#ewtrtw = Song("https://www.youtube.com/watch?v=SFU1GeGFpzY", "Everybody Wa
#ewtrtw_mp3 = ewtrtw.url_to_mp3('C:/Users/Adrian Bakhtar/Documents/Smart Pr
```

In [ ]:

```
my_hero = Song("https://www.youtube.com/watch?v=IKOv68AOmV4", "My Hero (Aco
my_hero_mp3 = my_hero.url_to_mp3('C:/Users/Adrian Bakhtar/Documents/Smart P
```

In [ ]:

```
my_hero_cut = Song("https://www.youtube.com/watch?v=IKOv68AOmV4", "My Hero 

my_hero_mp3_cut = my_hero_cut.url_to_mp3('C:/Users/Adrian Bakhtar/Documents
                                    start_time = '1:30')
```

References:

1. Pytube functionality: https://stackoverflow.com/questions/27473526/download-only-audio-from-youtube-video-using-youtube-dl-in-python-script (https://stackoverflow.com/questions/27473526/download-only-audio-from-youtube-video-using-youtube-dl-in-python-script)
2. Audio segmentation: https://github.com/jiaaro/pydub (https://github.com/jiaaro/pydub)

In [ ]:

```

```