Module 1:

```
import math

species, current_pop = '', ''

#Reasonable, Range, Existence, Data type
while species == '':
    species = input('Enter the species. (b = baw baw frog, m = mountain pygmy possum, t = tiger quoll): ')
    while species != 'b' and species != 'm' and species != 't':
        species = input('Enter the species. (b = baw baw frog, m = mountain pygmy possum, t = tiger quoll): ')
print('\n')

# Had to use chat gpt for the while true to prevent syntax error
# when typing letters for population while still repeating the input

while True:
    current_pop = input('Enter the current population of the species: ')
    try:
        if float(current_pop) < 0 or float(current_pop) % 1 != 0:
            current_pop = input('Enter the current population of the species: ')
        else:
            break
    except ValueError:
        print("Invalid input. Please enter a number.")
print('\n')

#Constants
B_MULT = 25
M_MULT = 78
T_MULT = 165

MAX_POPB = 50000
MAX_POPM = 6000
MAX_POPT = 1500

GROWTH_B = 1.75
GROWTH_M = 1.30
GROWTH_T = 1.50


def var_set():
    global max_pop, growth, mult, species
    if species == 'b':
        max_pop = MAX_POPB
        growth = GROWTH_B
        mult = B_MULT
    elif species == 'm':
        max_pop = MAX_POPM
        growth = GROWTH_M
        mult = M_MULT
    elif species == 't':
        max_pop = MAX_POPT
        growth = GROWTH_T
        mult = T_MULT

var_set()

current_pop = int(current_pop)

while current_pop > max_pop:
    print('Error, input population is above maximum population')
    while True:
        current_pop = input('Enter the current population of the species: ')
        var_set()
        current_pop = float(current_pop)
        try:
            if float(current_pop) < 0 or float(current_pop) % 1 != 0:
                current_pop = input('Enter the current population of the species: ')
            else:
                break
        except ValueError:
            print("Invalid input. Please enter a number.")


for i in range (0, 20):
    print('\n')
    annual_cost = (current_pop**0.8) * mult
    if current_pop >= max_pop:
        print(f'Population limit of {max_pop} reached in Year {i}!')
        break
    print(f'Year: {i} ' + f' Population: {math.floor(current_pop)}' + f' Annual cost: ${round(annual_cost)}')
    current_pop = current_pop * growth
```

Module 2:

```
#Todo:
#newline when csv saving, add scrollbars to table

import tkinter as tk
from tkinter import ttk, messagebox, filedialog
import csv, os

root = tk.Tk()
root.geometry('800x500')
root.title('Animal Data Converter')

file_path = None
new_records = []

#used chat gpt for code to convert csv to table.
def load_csv():
    global file_path
    file_path = filedialog.askopenfilename(filetypes=[("CSV Files", "*.csv")])
    if not file_path:
        return

    for row in table.get_children():
        table.delete(row)

    with open(file_path, newline='', encoding='utf-8') as file:
        reader = csv.reader(file)
        headers = next(reader)
        table["columns"] = headers
        table["show"] = "headings"

        #had to use AI to find out i needed stretch=tk.NO for the table to follow width
        for col in headers:
            table.heading(col, text=col)
            table.column(col, anchor="center", width=100, stretch=tk.NO)

        for row in reader:
            table.insert("", "end", values=row)




info_frame = tk.Frame(root)
info_frame.pack(expand='true')

csv_button = tk.Button(info_frame, text="Load CSV", bg='#156082', fg='#ffffff', width=10, command=load_csv)
csv_button.grid(pady=5, padx=5, column=0, row=0)



table = ttk.Treeview(info_frame)
table.grid(pady=5, padx=5, column=0, row=1)

def add_record():
    global file_type
    global form

    global name_input
    global species_input
    global age_input
    global sex_input
    global source_name_input
    global source_address_input
    global note1_input
    global note1_date_input
    global note2_date_input
    global note2_input
    if file_path == None:
         messagebox.showerror("Error", "Please choose a file to begin")
         return
    form = tk.Tk()
    form.title('Add New Record')
    form.geometry('500x500')

    info_frame = tk.Frame(form)
    info_frame.pack(expand='true')

    name_label = tk.Label(info_frame, text='Name')
    name_label.grid(row=0, column=0)
    name_input = tk.Entry(info_frame)
    name_input.grid(row=0, column=2, pady=10)

    species_label = tk.Label(info_frame, text='Species')
    species_label.grid(row=1, column=0)
```

```
    species_input = tk.Entry(info_frame)
    species_input.grid(row=1, column=2, pady=10)

    age_label = tk.Label(info_frame, text='Age')
    age_label.grid(row=2, column=0)
    age_input = tk.Entry(info_frame)
    age_input.grid(row=2, column=2, pady=10)

    sex_label = tk.Label(info_frame, text='Sex')
    sex_label.grid(row=3, column=0)
    sex_input = tk.Entry(info_frame)
    sex_input.grid(row=3, column=2, pady=10)

    source_name_label = tk.Label(info_frame, text='Source Name')
    source_name_label.grid(row=4, column=0)
    source_name_input = tk.Entry(info_frame)
    source_name_input.grid(row=4, column=2, pady=10)

    source_address_label = tk.Label(info_frame, text='Source Address')
    source_address_label.grid(row=5, column=0)
    source_address_input = tk.Entry(info_frame)
    source_address_input.grid(row=5, column=2, pady=10)

    note1_label = tk.Label(info_frame, text='Note 1')
    note1_label.grid(row=6, column=0)
    note1_input = tk.Entry(info_frame)
    note1_input.grid(row=6, column=2, pady=10)

    note1_date_label = tk.Label(info_frame, text='Note 1 Date')
    note1_date_label.grid(row=7, column=0)
    note1_date_input = tk.Entry(info_frame)
    note1_date_input.grid(row=7, column=2, pady=10)

    note2_label = tk.Label(info_frame, text='Note 2')
    note2_label.grid(row=8, column=0)
    note2_input = tk.Entry(info_frame)
    note2_input.grid(row=8, column=2, pady=10)

    note2_date_label = tk.Label(info_frame, text='Note 2 Date')
    note2_date_label.grid(row=9, column=0)
    note2_date_input = tk.Entry(info_frame)
    note2_date_input.grid(row=9, column=2, pady=10)

    button = tk.Button(info_frame, text="Save", bg='#156082', fg='#ffffff', width=10, command=save_record)
    button.grid(pady=5, padx=5, row=10, column=1)


    form.mainloop()

#used chatgpt for code to identify selected row and delete the row accordingly
def delete_record():
    global file_path
    if file_path == None:
        messagebox.showerror("Error", "Please choose a file to begin")
        return

    selected_item = table.selection()
    if not selected_item:
        messagebox.showwarning("Warning", "Please select a record to delete.")
        return
    confirmation = messagebox.askyesno("Confirm Delete", "Are you sure you want to delete the selected record?")
    if confirmation:
        for item in selected_item:
            table.delete(item)

def save_csv():
    global new_records

    global file_path
    if file_path == None:
        messagebox.showerror("Error", "Please choose a file to begin")
        return

    # new line bug fix, if pre existing code
    file_exists = os.path.isfile(file_path) and os.path.getsize(file_path) > 0

    #put this on notes
    with open(file_path, mode='a', newline='') as file:
            writer = csv.writer(file)

            if file_exists:
                file.write('')

            writer.writerows(new_records)
```

```
    messagebox.showinfo("Success", "New data has been appended")


def save_record():
        global form

        global data
        global new_records

        global name_input
        global species_input
        global age_input
        global sex_input
        global source_name_input
        global source_address_input
        global note1_input
        global note1_date_input
        global note2_date_input
        global note2_input

        name = name_input.get()
        species = species_input.get()
        age = age_input.get()
        sex = sex_input.get()
        source_name = source_name_input.get()
        source_address = source_address_input.get()
        note1 = note1_input.get()
        note1_date = note1_date_input.get()
        note2 = note2_input.get()
        note2_date = note2_date_input.get()



        data = [name, species, age, sex, source_name, source_address, note1, note1_date, note2, note2_date]
        table.insert("", "end", values=data)

        new_records.append(data)

        messagebox.showinfo("Success", "New data has been appended")
        form.destroy()


record_frame = tk.Frame(info_frame)
record_frame.grid(row=2, column=0)

button = tk.Button(record_frame, text="Add Record", bg='#156082', fg='#ffffff', width=10, command=add_record)
button.grid(pady=5, padx=5, column=0, row=2)

button = tk.Button(record_frame, text="Delete Record", bg='#156082', fg='#ffffff', width=10, command=delete_record)
button.grid(pady=5, padx=5, column=1, row=2)

button = tk.Button(record_frame, text="Save CSV", bg='#156082', fg='#ffffff', width=10, command=save_csv)
button.grid(pady=5, padx=5, column=2, row=2)



root.mainloop()
```

Module 3:

```
#todo search, optional add a update page instead of everything in page up

import tkinter as tk

page = 1
sort_last = False
sort_expertise = False
search_criteria = False
searched_rangers = []

root = tk.Tk()
root.geometry('500x500')
root.title('Ranger Viewer')

#misses code
def read_file():
    global rangers
    rangers = []

    with open('rangers.txt', 'r', encoding='utf-8') as file:
        for line in file:
            values = line.strip().split('#')
            rangers.append({
                'first_name': values[0],
                'last_name': values[1],
```

```
                'dob': values[2],
                'expertise': values[3],
                'employed': values[4]
            })

read_file()

def update_display(data_list, index):
    global record_frame

    record_frame.destroy()
    record_frame = tk.Frame(root, bd=1, relief="solid")
    record_frame.grid(column=0, row=0, pady=20, padx=20)

    if len(data_list) > 0:
        record_name = tk.Label(record_frame, text=f"First name: {data_list[index]['first_name']}")
        record_name.pack(pady=10, padx=10)
        record_last_name = tk.Label(record_frame, text=f"Last name: {data_list[index]['last_name']}")
        record_last_name.pack(pady=10, padx=10)
        record_dob = tk.Label(record_frame, text=f"Date of birth: {data_list[index]['dob']}")
        record_dob.pack(pady=10, padx=10)
        record_area = tk.Label(record_frame, text=f"Area of expertise: {data_list[index]['expertise']}")
        record_area.pack(pady=10, padx=10)
        record_current = tk.Label(record_frame, text=f"Current employee: {data_list[index]['employed']}")
        record_current.pack(pady=10, padx=10)
    else:
        record_name = tk.Label(record_frame, text="No records found")
        record_name.pack(pady=10, padx=10)

def update_page_number():
    global page_label
    page_label.config(text=page)

def page_next():
    global rangers
    global page
    global sort_last
    global sort_expertise
    global search_criteria
    global searched_rangers

    data_to_use = searched_rangers if search_criteria else rangers

    if len(data_to_use) == 0:
        return

    if sort_last or sort_expertise:
        page = 1
        sort_last = False
        sort_expertise = False
    elif page >= len(data_to_use):
        page = 1
    else:
        page += 1

    update_page_number()
    update_display(data_to_use, page - 1)

def page_previous():
    global rangers
    global page
    global sort_last
    global sort_expertise
    global search_criteria
    global searched_rangers

    data_to_use = searched_rangers if search_criteria else rangers

    if len(data_to_use) == 0:
        return

    if page <= 1:
        page = len(data_to_use)
    else:
        page -= 1

    update_page_number()
    update_display(data_to_use, page - 1)

#Put on notes (searching function)
#Do search
def search():
    global search_entry
    global rangers
```

```
    global searched_rangers
    global page
    global search_criteria

    # Reset search results
    searched_rangers = []
    search_term = search_entry.get().lower()  # Case insensitive search

    # If search term is empty, show all rangers
    if not search_term:
        search_criteria = False
        page = 1
        update_page_number()
        update_display(rangers, page - 1)
        return

    #DYNAMIC SEARCH, SEARCHED WHOLE DICT FOR THE SEARCH TERM
    # Find matches
      # Find matches in any field
    for ranger in rangers:
        # Check each value in the ranger dictionary
        for value in ranger.values():
            if search_term in value.lower():
                searched_rangers.append(ranger)
                break  # Once we find a match in any field, no need to check other fields

    # Update display with search results
    search_criteria = True
    page = 1

    # Update the global searched_rangers list

    update_page_number()
    update_display(searched_rangers, 0 if searched_rangers else -1)

def sortlast():
    global rangers
    global page
    global sort_last
    rangers.sort(key=lambda x: x['last_name'])
    sort_last = True
    page = 1
    update_page_number()
    update_display(rangers, 0)

#add sorting to cheat sheet notes
def sortexpertise():
    global rangers
    global page
    global sort_expertise
    rangers.sort(key=lambda x: x['expertise'])
    sort_expertise = True
    page = 1
    update_page_number()
    update_display(rangers, 0)

record_frame = tk.Frame(root, bd=1, relief="solid")
record_frame.grid(column=0, row=0, pady=20, padx=20)

record_name = tk.Label(record_frame, text=f"First name: {rangers[0]['first_name']}")
record_name.pack(pady=10, padx=10)
record_last_name = tk.Label(record_frame, text=f"Last name: {rangers[0]['last_name']}")
record_last_name.pack(pady=10, padx=10)
record_dob = tk.Label(record_frame, text=f"Date of birth: {rangers[0]['dob']}")
record_dob.pack(pady=10, padx=10)
record_area = tk.Label(record_frame, text=f"Area of expertise: {rangers[0]['expertise']}")
record_area.pack(pady=10, padx=10)
record_current = tk.Label(record_frame, text=f"Current employee: {rangers[0]['employed']}")
record_current.pack(pady=10, padx=10)

page_frame = tk.Frame(root)
page_frame.grid(column=0, row=1, pady=20, padx=20)

page_button = tk.Button(page_frame, text='Previous', bg='#156082', fg='#ffffff', command=page_previous)
page_button.grid(column=0, row=0, pady=10, padx=10)
page_label = tk.Label(page_frame, text=page)
page_label.grid(column=1, row=0, pady=10, padx=10)
page_button = tk.Button(page_frame, text='Next', bg='#156082', fg='#ffffff', command=page_next)
page_button.grid(column=2, row=0, pady=10, padx=10)

search_frame = tk.Frame(root, bd=1, relief='solid')
search_frame.grid(column=1, row=0, pady=20, padx=20)

search_entry = tk.Entry(search_frame)
```

```
search_entry.pack(pady=20, padx=20)
search_button = tk.Button(search_frame, text='Search', bg='#156082', fg='#ffffff', command=search)
search_button.pack(pady=20, padx=20)

sort_frame = tk.Frame(root, bd=1, relief='solid')
sort_frame.grid(column=1, row=1, pady=20, padx=20)

sort_button = tk.Button(sort_frame, text='Sort by last name', bg='#156082', fg='#ffffff', command=sortlast)
sort_button.pack(pady=20, padx=20)
sort_button = tk.Button(sort_frame, text='Sort by expertise', bg='#156082', fg='#ffffff', command=sortexpertise)
sort_button.pack(pady=20, padx=20)

root.mainloop()
```

Module 43

```
from abc import ABC, abstractmethod

# Define species and operation variables
species, operation = '', ''
total = None

# Easier way of typing data validation
while species not in ['g', 'b', 'e', 'u']:
    species = input('Enter the species. (g = giraffe, b = bison, e = elk, ENTER = unknown): ')
print('\n')

# Define the Patient class
class Patient(ABC):
    def __init__(self, species, weight, operation):
        self.species = species
        self.weight = weight
        self.operation = operation

    @abstractmethod
    def calculate_cost(self):
        pass

# Define the Giraffe class
class Giraffe(Patient):
    def calculate_cost(self):
        global total
        if self.operation == "minor":
            total = 300
            if self.weight > 500:
                total += 450
            elif self.weight > 150:
                total += 250
        elif self.operation == "major":
            total = 600
            if self.weight > 500:
                total += 650
            elif self.weight > 150:
                total += 350
        return round(total)

# Define the Bison class
class Bison(Patient):
    def calculate_cost(self):
        global total
        if self.operation == "minor":
            total = (self.weight + 100) * 1.2
        elif self.operation == "major":
            total = (self.weight * 2 + 150) * 1.6
        return round(total)

# Define the Elk class
class Elk(Patient):
    def calculate_cost(self):
        global total
        if self.operation == "minor":
            total = (self.weight + 100) + 300
        elif self.operation == "major":
            total = (self.weight * 2 + 150) + 700
        return round(total)

# Define the UnknownSpecies class
class UnknownSpecies(Patient):
    def calculate_cost(self):
        global total
        if self.operation == "minor":
            total = self.weight + 100
        elif self.operation == "major":
            total = self.weight * 2 + 150
        return round(total)
```

```
# Put in notes for validation, allows for both float and integer (replace float() with int() to only allow int only)
weight = input('Enter the weight in kg: ')

weight_validate = False
while not weight_validate:
    try:
        weight = float(weight)
        if weight < 0:
            weight = input('Enter the weight in kg: ')
        else:
            weight_validate = True
    except ValueError:
        print('Error, invalid weight', '\n')
        weight = input('Enter the weight in kg: ')

# Operation selection
while operation != '1' and operation != '2':
    operation = input('Which operation? (1 = minor, 2 = major): ')

# Create the correct species object based on the user's input
if species == 'g':
    species_object = Giraffe(species, weight, 'minor' if operation == '1' else 'major')
elif species == 'b':
    species_object = Bison(species, weight, 'minor' if operation == '1' else 'major')
elif species == 'e':
    species_object = Elk(species, weight, 'minor' if operation == '1' else 'major')
else:
    species_object = UnknownSpecies(species, weight, 'minor' if operation == '1' else 'major')

# Main function to calculate and display the cost
def main():
    cost = species_object.calculate_cost()
    print(f'The operation will cost ${cost}')

if __name__ == "__main__":
    main()
```