Title: Zoo Algorithm

Author: Shaun Clarke

Goal: This program mimics an animal database, allowing us to add and retrieve animal details.

Steps:

1. Import Dict from typing for type hinting
2. Define a parent class Animal:
    a. This parent class has the basics needed to create an animal.
    b. Define a class variable __zoo_keeper, which is a dict that will hold animals.
    c. The construct or takes the animal's name, species, and animal(type of animal) and initializes the following:
        i. The animals name.
        ii. The animals species
        iii. Calls a method that adds the created animal to the zoo_keeper dict.
    d. Define a method __add_animal:
        i. This method adds the animal to the zoo dict.
    e. Define a method select_animal(self, animal: str) -> object:
        i. This method allows us to select an animal from the zoo dict.
        ii. For loop:
            1. If the animal type in the dict matches what we are looking for:
                a. Return that animal object.
    f. Define a method make_sound(self) -> str:
        i. This method returns the sound the animal makes.
        ii. Return sound
    g. Define a method info(self) -> str:
        i. This method returns the animal attributes formatted for output.
        ii. Return info as a formatted string.
    h. Define a method get_all_animals(self) -> Dict:
        i. This method returns the __zoo_keeper dict with all animal objects.
            1. Return __zoo_keeper
    i. Define a method __str__(self) -> str:
        i. This magic method returns the output of the info method. When the object is printed.
        ii. Return self.info()
3. Define a class Bear(Animal):
    a. This is a subclass of Animal. The info and make_sound methods are overwritten in this subclass.

b. Define a constructor __init__(self, name: str, species: str, animal: str, fur_color: str):
    i. The constructor initializes the following:
        1. super().__init__(name, species, animal)
        2. The fur color attribute
c. Define a method make_sound(self) -> str:
    i. This method is overridden to return the sound this specific animal makes.
    ii. Return sound
d. Define a method info(self) -> str:
    i. This method is overridden to include this specific animal's attributes formatted for output.
    ii. Return info as a formatted string.

4. Define a class Elephant(Animal):
    a. This is a subclass of Animal. The info and make_sound methods are overwritten in this subclass.
    b. Define a constructor __init__(self, name: str, species: str, animal: str, weight: float):
        i. The constructor initializes the following:
            1. super().__init__(name, species, animal)
            2. The weight attribute
    c. Define a method make_sound(self) -> str:
        i. This method is overridden to return the sound this specific animal makes.
        ii. Return sound
    d. Define a method info(self) -> str:
        i. This method is overridden to include this specific animal's attributes formatted for output.
        ii. Return info as a formatted string.
5. Define a class Penguin(Animal):
    a. This is a subclass of Animal. The info and make_sound methods are overwritten in this subclass.
    b. Define a constructor __init__(self, name: str, species: str, animal: str, height: float):
        i. The constructor initializes the following:
            1. super().__init__(name, species, animal)
            2. The height attribute
    c. Define a method make_sound(self) -> str:
        i. This method is overridden to return the sound this specific animal makes.
        ii. Return sound

    d. Define a method info(self) -> str:
        i. This method is overridden to include this specific animal's attributes formatted for output.
        ii. Return info as a formatted string.
6. Define a class HandleInput:
    a. Define a method get_string_input(self, input_message: str) -> str:
        i. This user gets the user input as a string and makes sure it's not empty.
        ii. While loop:
            1. Ask user for input.
                a. Make it lower case and strip whitespace.
            2. If there was no input, raise a ValueError and ask the user to try again.
            3. Otherwise return the user input
    b. Define a method get_number_input(self, input_message: str) -> str:
        i. This method gets the user input and ensures it is a float and not empty.
        ii. While loop
            1. Ask user for input and strip whitespace.
            2. If the user input a float and its equal when converted to an int.
                a. Return user_input
            3. If the input is not a float, raise a ValueError and ask the user to try again.
    c. Define a method handle_add_bear(self, bear_class: Bear) -> object:
        i. This method handles the user inputs needed to create a Bear object.
        ii. Call the get_string_input method to get the bear's name.
        iii. Call the get_string_input method to get the bear's species.
        iv. Call the get_string_input method to get the bear's fur color.
        v. Use the bear_class(name, species, "bear", fur_color) with inputs to create a bear object.
        vi. Return the bear object.
    d. Define a method handle_add_elephant(self, elephant_class: Elephant) -> object:
        i. This method handles the user inputs needed to create an Elephant object.
        ii. Call the get_string_input method to get the elephant's name.
        iii. Call the get_string_input method to get the elephant's species.
        iv. Call the get_number_input method to get the elephant's weight.
        v. Use the elephant_class(name, species, "elephant", weight) with inputs to create an elephant object.
        vi. Return the elephant object.

e. Define a method handle_add_penguin(self, penguin_class: Penguin) -> object:
    i. This method handles the user inputs needed to create a Penguin object.
    ii. Call the get_string_input method to get the penguin's name.
    iii. Call the get_string_input method to get the penguin's species.
    iv. Call the get_number_input method to get the penguin's height.
    v. Use the penguin_class(name, species, "elephant", weight) with inputs to create a penguin object.
    vi. Return the penguin object.
f. Define a method handle_select_animal(self, an_animal: Animal, animal_type: str) -> object:
    i. This method handles the user input when selecting an animal to print.
    ii. Using the Animal object that was passed in, call the .slect_animal method with the animal type.
    iii. If the animal does not exist.
        1. Raise a keyerror animal was not found.
        2. Otherwise return the animal object that was found.

7. Define a class Main:
    a. Define a constructor def __init__(self, bear_class: Bear, elephant_class: Elephant, penguin_class: Penguin, input_object: HandleInput) -> None:
        i. This constructor takes the other classes as constructor inputs and initializes the following.
            1. The Bear class
            2. The Elephant class
            3. The Penguin class
            4. The HandleInput object
            5. A variable called an animal that holds an animal object when added.
            6. Main menu list
            7. Add animal sub menu
            8. Print specific animal sub menu
    b. Define a method get_menu_number_input(self,input_message: str, menu_options: List) -> str:
        i. This method prompts the user to enter a menu selection.
        ii. Uses the specified menu options list parameter to enforce input requirements.
        iii. While loop:
            1. Prompt user for input and strip whitespace.
            2. If user input is empty:

            a. Raise a keyerror input cannot be empty and have the user try again.
        3. If the user's input number meets the menu number requirements, return the user's input.
        4. Otherwise, have the user try again.
c. Define a method display_menu(self, menu: List) -> int:
    i. This method displays the specified menu and return the selection.
    ii. Define a variable that holds the footer border line.
    iii. Print the menu title
    iv. Print the menu selections
    v. Print the border footer variable
    vi. Call the get_menu_number_input("Choose a menu item to continue", menu) function.
    vii. Return the menu number input.
d. Define a method main(self):
    i. This method runs the program when called.
    ii. While loop:
        1. Call the display_menu function to display the main menu.
        2. If the user selects 1 from the main menu:
            a. Call the display_menu function to display the add amimal sub menu.
            b. If the user selects 1 from add animal sub menu:
                i. Call the handle_add_bear from the HandleInput object.
                ii. If the bear was added, update the an_animal variable with the bear object.
            c. If the user selects 2 from add animal sub menu:
                i. Call the handle_add_elephant from the HandleInput object.
                ii. If the elephant was added, update the an_animal variable with the elephant object.
            d. If the user selects 3 from add animal sub menu:
                i. Call the handle_add_penguin from the HandleInput object.
                ii. If the penguin was added, update the an_animal variable with the penguin object.
        3. If the user selects 2 from the main menu:
            a. If the an_animal variable is not None:

        i. Tell the user they must add an animal before using this option.
    b. Use the an_animal variable that holds an animal object, to access the get_all_animals method.
        i. Loop through all animals, printing them.
4. If the user selects 3 from the main menu:
    a. Call the display_menu function to display the print specific menu.
    b. If the user selects 1 from add print specific sub menu:
        i. If the an_animal variable is not None:
            1. Tell the user they must add an animal before using this option.
        ii. Get the first option from the print specific menu.
        iii. Split the item into a list and slice[-1] the last item(animal's name)
        iv. Use the an_animal variable that holds an animal object, to access the select_animal method with the animal name.
        v. If the animal exists print it.
        vi. If not tell the user it was not found and go back to main menu.
    c. If the user selects 2 from add print specific sub menu:
        i. If the an_animal variable is not None:
            1. Tell the user they must add an animal before using this option.
        ii. Get the first option from the print specific menu.
        iii. Split the item into a list and slice[-1] the last item(animal's name)
        iv. Use the an_animal variable that holds an animal object, to access the select_animal method with the animal name.
        v. If the animal exists print it.
        vi. If not tell the user it was not found and go back to main menu.
    d. If the user selects 3 from add print specific sub menu:
        i. If the an_animal variable is not None:
            1. Tell the user they must add an animal before using this option.

        ii. Get the first option from the print specific menu.
        iii. Split the item into a list and slice[-1] the last item(animal's name)
        iv. Use the an_animal variable that holds an animal object, to access the select_animal method with the animal name.
        v. If the animal exists print it.
        vi. If not tell the user it was not found and go back to main menu.
    e. If the user selects 4 from the main menu:
        i. Exit the program.
8. Create the main object passing in the following classes:
    a. Bear, Elephant, Penguin, and HandleInput.
9. Run the program calling the .main method.