Connected to Python 3.11.7

In [ ]:

```
"""
-------------------------
Final Project: Pet Matcher Driver
-------------------------
STUDENT: Pamela Alvarez
SEMESTER: Fall 2023
"""
import constants

def validate_input(prompt: str) -> int:
    """
    Ensures that the user provides a valid answer: 1, 2 or 3. If an invalid answer
    is entered, a custom message prints to the user and prompts them to provide an

    Args:
        prompt (str): a string containing the message to be printed to the user.
        The user will respond to this message with a 1, 2 or 3. If anything else is
        the user is prompted to choose a valid answer.

    Returns:
        int: the integer value 1, 2 or 3 based on the user's answer
    """
    user_answer = input(prompt)
    if user_answer.isnumeric():
        number = int(user_answer)
        if number > 0 and number < 4:
            return number
        else:
            print("Your answer must be 1, 2, or 3.")
            return validate_input(prompt)  # provide the same prompt to user until
    else:
        print("Your answer cannot contain letters. It must be the whole number '1',
        return validate_input(prompt)  # provide the same prompt to user until a va



def get_prompt_answers() -> list:
    """
    Obtains the user's response to each prompt. Asks the prompts in a rotating
    order, starting with a prompt for Pet A, then a prompt for Pet B, and so on
    until all prompts have been asked. Stores the ordered answers in a list. Return

    Args:
        None

    Returns:
        list [int]: a list of integers. Each integer is the user's response to a pr
            The order of the elements in this list is [answer_A, answer_B, answer_C
            in this pattern until an answer for all prompts is obtained.
    """
    list = []
    for i in range(0, constants.NUMBER_OF_PROMPTS):  # the loop iterations must lin
```

```
        answer_A = validate_input(constants.PET_A_PROMPTS[i])  # rotate through the
        answer_C = validate_input(constants.PET_C_PROMPTS[i])  # a pet category are
        answer_B = validate_input(constants.PET_B_PROMPTS[i])
        answer_D = validate_input(constants.PET_D_PROMPTS[i])
        list.extend([answer_A, answer_B, answer_C, answer_D])  # keep the list in a
    return list                                                #  for each pet cate

def sum_results(list: [int], index: int) -> int:
    """
    Takes a list of integers and sums the every 4th element, starting with the
    first element at the index value provided.

    Example:
    >>> sum_results([1, 2, 2, 3, 1, 2, 2, 1, 3, 1, 3, 3], 2)
    7

    Args:
       list [int]: a list consisting of integers.
       index (int): the index of the first element in the list to be added to the t

    Returns:
        int: the sum of every 4th element in the list, starting at the index provid
    """
    final_score = list[index]
    while index < (len(list) - 4):  #subtract 4 because there are 4 total pet categ
        index += 4                  # add 4 because there are 4 pet categories. Eve
        final_score += list[index]  # the starting index will correspond to the sam
    return final_score

def log_scores(list: [int]) -> dict:
    """
    Takes a list of integers containing the user answers and returns a
    dictionary where each key is a pet type and each value is the corresponding
    sum of the user's answers.

    Example:
    constants.PET_A = Cat
    constants.PET_B = Reptile
    constants.PET_C = Hamster
    constants.PET_D = Rabbit
    >>> log_scores([2, 3, 1, 1, 2, 2, 1, 3])
    {'Cat': 4, 'Reptile': 5, 'Hamster': 2, 'Rabbit': 4}

    Args:
       list [int]: a list consisting of integers. These are the user's answers to o

    Returns:
        dict {str: int}: a dictionary where each key is a pet type, and each value
            of the user's responses to this pet type's prompts.
    """
    results = {}
    results[constants.PET_A] = sum_results(list, 0)  # according to the list order
    results[constants.PET_B] = sum_results(list, 1)  # we know the exact starting i
    results[constants.PET_C] = sum_results(list, 2)  # has its first value stored i
    results[constants.PET_D] = sum_results(list, 3)
    return results
```

```
def find_pet_match(dict: {str: int}) -> list:
    """
    Takes a dictionary and finds the highest value in the dictionary. Returns a lis
    where each element is a key corresponding to the highest value in the dictionar
    If the highest score is 10, and only one pet type has that value, only that pet
    is returned in the list. If there is a tie, then all the pet types with the tie
    score are returned in the list.

    Example:
    sample = {'Cat': 4, 'Reptile': 5, 'Hamster': 2, 'Rabbit': 4}
    >>> find_pet_match(sample)
    ['Reptile']

    Args:
         dict {str: int}: a dictionary where each key is a pet type, and each value
            of the user's responses to this pet type's prompts.

    Returns:
        list [str]: a list containing the key(s) in the dictionary with the highest
            is a tie for the highest value, all the keys with that value are return
    """
    max_score = max(dict.values())
    match = []  # build a list to store the keys in because there may be ties for h
    for key in dict:
        if dict[key] == max_score:
            match.append(key)
    return match

def print_pet_match(list) -> None:
    """
    Takes a list and prints the contents to the user. If only one element is in the
    message for one pet match is printed. If more than one element is in the list,
    for multiple pet matches is printed.

    Example:
    sample = [Cat]
    >>> print_pet_match(sample)
    'Hooray! Your pet match is:
    Cat'


    Args:
         list [str]: a list of one or more strings, representing the user's pet mat

    Returns:
        None. Prints a message to let the user know their pet matches.
    """
    if len(list) > 1:        # if the list built in find_pet_match() has more than
        separator = " and "  # more than one pet matches, so our printout message m
        matches = separator.join(list)
        print("Wow! You matched with the following pet types:")
        print(matches)
    else:
        print("Hooray! Your pet match is:")
        print(list[0])
```

```
def print_pet_info(list: [str]) -> None:
    """
    Takes a list containing the user's pet matches and prints corresponding
    information to the terminal based on the pet match.

    Args:
        list [str]: a list of one or more strings, representing the user's pet matc

    Returns:
        None. If a pet type is in the list, prints information about that pet type
    """
    print("Here is some information about your pet match:")
    if constants.PET_A in list:      # go through each if statement sequentially be
        print(constants.PET_A_INFO)  # any combination of pet matches, and we want
    if constants.PET_B in list:      # for each pet category that was matched
        print(constants.PET_B_INFO)
    if constants.PET_C in list:
        print(constants.PET_C_INFO)
    if constants.PET_D in list:
        print(constants.PET_D_INFO)


def main() -> None:
    """
    Runs pet matching program. A welcome message introduces the program
    to the user. A series of prompts will then print to terminal. After each prompt
    user must enter a number: 1, 2 or 3 in order to receive the next prompt.

    If an invalid value is entered, message to user regarding answer requirements p
    and user is provided the same prompt until a valid answer is provided.

    Each prompt corresponds to one of four possible pet categories. The answers to
    logged. Once all prompts have been answered the total sums for each pet categor

    The pet category with the highest final score is printed to the terminal. It is
    than one pet category if there is a tie for the highest score. Information abou
    is printed to the terminal.

    Args:
        None.

    Returns:
        None. Prints a series of messages to the terminal for the user.
    """

    print(constants.WELCOME_MESSAGE)  # introduce program and provide instructions
    user_answers = get_prompt_answers()  # collect user answers to each prompt, one
    results = log_scores(user_answers)  # store the results in a dictionary
    pet_match = find_pet_match(results)  # compare the total scores for each pet ca
    print_pet_match(pet_match)  # share results with the user
    print_pet_info(pet_match)  # share additional information about the user's pet
    print(constants.GOODBYE_MESSAGE)  # goodbye message for user

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

```
Welcome to the Pet Matcher!
You will receive a series of prompts that will determine the best pet for you!
Answer each prompt with a 1, 2, or 3, using the guide below.
1 -- I don't agree with this at all
2 -- I'm not sure how I feel about this
3 -- I totally agree with this


Hooray! Your pet match is:
TestD
Here is some information about your pet match:
test d1
test d2

Thank you for using the Pet Matcher!
```

Start of a different test run. The above test run was correct: the expected result was TestD.

Below, different answers to get_prompt_answers() were provided. The expected results based on the new answers below is correct: TestB and TestD

```
print(get_prompt_answers())
```

```
[1, 3, 2, 3, 1, 3, 2, 3, 1, 3, 2, 3]
```

```
print(log_scores([1, 3, 2, 3, 1, 3, 2, 3, 1, 3, 2, 3]))
```

```
{'TestA': 3, 'TestB': 9, 'TestC': 6, 'TestD': 9}
```

```
print(find_pet_match({'TestA': 3, 'TestB': 9, 'TestC': 6, 'TestD': 9}))
```

```
['TestB', 'TestD']
```

In [ ]:
```
print(print_pet_match(['TestB', 'TestD']))
```

```
Wow! You matched with the following pet types:
TestB and TestD
None
```

In [ ]:
```
print(print_pet_info(['TestB', 'TestD']))
```

```
Here is some information about your pet match:
test b1
test b2
test b3
test b4

test d1
test d2

None
```