```python
"""Exercise Calculator

Author: Daniel Kareh
Description: Calculates how many calories were burned when biking, running,
             or swimming, given information about the exercise, such as
             distance and duration.
Sources:
    - https://www.omnicalculator.com/sports/calories-burned-biking
    - https://www.omnicalculator.com/sports/running-calorie
    - https://www.omnicalculator.com/sports/swimming-calorie
"""

__author__ = "Daniel Kareh"

KILOMETERS_PER_MILE = 1.609344
KILOGRAMS_PER_POUND = 0.45359237
ACRES_PER_SQUARE_MILE = 640
CALORIES_PER_SQUARE = 100
CALORIES_BAR_LABEL_SIZE = 15
MINUTES_PER_HOUR = 60
LABEL_COLUMN_SIZE = 30
# Most terminals support these "escape sequences". If yours doesn't, change
# the "True" to "False".
# Source: https://en.wikipedia.org/wiki/ANSI_escape_code
TERMINAL_SUPPORTS_ESCAPE_SEQUENCES = True
if TERMINAL_SUPPORTS_ESCAPE_SEQUENCES:
    CURSOR_UP = "\x1b[A"
    CLEAR_REST_OF_SCREEN = "\x1b[J"
    CLEAR_REST_OF_LINE = "\x1b[K"
else:
    CURSOR_UP = CLEAR_REST_OF_SCREEN = CLEAR_REST_OF_LINE = ""
SWIMMING_STYLES = [
    "intense backstroke",
    "backstroke",
    "intense breaststroke",
    "breaststroke",
    "butterfly",
    "intense crawl",
    "crawl",
    "sidestroke",
    "high effort treading water",
    "treading water",
]
# MET means "Metabolic Equivalent of Task".
SWIMMING_STYLE_METS = [9.5, 4.8, 10.3, 5.3, 13.8, 10.0, 8.3, 7.0, 9.8, 3.5]
TITLE = "Daniel's Exercise Calculator"
COMMANDS = ["biking", "running", "swimming", "exit"]


def constrain(lower_limit, number, upper_limit):
    """Return the number constrained to lie between the limits.

    Examples:
    constrain(1, 5, 10) = 5
    constrain(1, 100, 10) = 10
    constrain(1, -100, 10) = 1
    """
    return min(max(lower_limit, number), upper_limit)


def get_positive_number(prompt):
    """Read and return a positive number entered by the user.

    Arguments:
    prompt -- the message that prompts the user to enter input
    """
    # Add enough spaces to make the prompt LABEL_COLUMN_SIZE characters long.
    prompt = prompt.ljust(LABEL_COLUMN_SIZE) + " | " + CLEAR_REST_OF_LINE
    got_a_number = False
    number = None
    while not got_a_number:
        user_input = input(prompt).strip().lower()
        # Clear any previous error message.
        print(CLEAR_REST_OF_SCREEN, end="")
        try:
            number = float(user_input)
            if number > 0:
                got_a_number = True
            else:
                print(
                    user_input + " is not greater than zero. Please try again."
                )
                # Move the cursor up to line containing the prompt.
                print(CURSOR_UP * 2, end="")
        except ValueError:
            print("'" + user_input + "' is not a number. Please try again.")
            print("(You must use digits, not words, as in '10', not 'ten'.)")
            # Move the cursor up to line containing the prompt.
            print(CURSOR_UP * 3, end="")
    return number


def get_index_of_one_of(prompt, options):
    """Read one of the provided options and return its index.

    Arguments:
    prompt -- the message that prompts the user to enter input
    options -- the list of options
    """
    prompt = prompt.ljust(LABEL_COLUMN_SIZE) + " | " + CLEAR_REST_OF_LINE
    got_an_option = False
    option_index = None
    while not got_an_option:
        user_input = input(prompt).strip().lower()
        print(CLEAR_REST_OF_SCREEN, end="")
        if user_input == "options":
            print("\nOptions:")
            for option in options:
                print(option)
            print("")  # Print a blank line.
            # Instead of moving the cursor up, let's leave the list of options
            # on the screen so that the user can continue to reference it.
            continue
        try:
            option_index = options.index(user_input)
            got_an_option = True
        except ValueError:
            print("'" + user_input + "' is not an option. Please try again.")
            print("(To see the possible options, enter 'options'.)")
            print(CURSOR_UP * 3, end="")
    return option_index

```

```python

def get_yes_or_no(prompt, default=None):
    """Read a yes or no and return true or false, respectively.

    Arguments:
    prompt -- the message that prompts the user to enter input
    default -- the default value, if any (default None)
    """
    default_response = "yes" if default is True else "no"

    if default is not None:
        prompt += " (default: " + default_response + ")"
    prompt = prompt.ljust(LABEL_COLUMN_SIZE) + " | " + CLEAR_REST_OF_LINE

    got_a_response = False
    response = None
    while not got_a_response:
        user_input = input(prompt).strip().lower()
        print(CLEAR_REST_OF_SCREEN, end="")
        if user_input in ["yes", "y"]:
            got_a_response = True
            response = True
        elif user_input in ["no", "n"]:
            got_a_response = True
            response = False
        elif user_input == "" and default is not None:
            got_a_response = True
            response = default
            # Output the prompt again, but with the default response added, as
            # if the user entered the default response.
            print(CURSOR_UP, end="")
            print(prompt + default_response)
        else:
            print("'" + user_input + "' is not a yes or no. Please try again.")
            print(CURSOR_UP * 2, end="")
    return response


def print_stat(label, number, units=""):
    """Print the justified label, the number, and its units."""
    print(label.ljust(LABEL_COLUMN_SIZE), "|", round(number, 2), units)


def calc_biking(weight_kg, distance_mi, duration_hours):
    """Calculate the number of calories burned while biking."""
    speed_mph = distance_mi / duration_hours
    # Approximate the "Metabolic Equivalent of Task", a way of measuring the
    # amount of energy expended by doing some physical activity.
    # 1 MET is approximately equal to 1 calorie per kilogram per hour.
    met = constrain(4, speed_mph - 5, 16)
    print_stat("Metabolic Equivalent of Task", met)
    # Evenly divide the distance covered between the four sides of a square.
    side_length_mi = distance_mi / 4
    # The area of a square is always its side length squared.
    area_square_mi = side_length_mi**2
    # Example: If there are 640 acres per square mile, and we have two square
    # miles of land, then we have 2 * 640, or 1,280 acres.
    area_acres = area_square_mi * ACRES_PER_SQUARE_MILE
    print_stat("Traced square area", area_square_mi, "square miles")
    print_stat("Traced square area", area_acres, "acres")
    calories_burned = duration_hours * met * weight_kg
    return calories_burned


def calc_running(weight_kg, distance_km):
    """Calculate the number of calories burned while running."""
    age_years = get_positive_number("Age (in years)?")
    resting_heart_bpm = get_positive_number("Resting heart rate (in BPM)?")
    on_a_treadmill = get_yes_or_no("On a treadmill?", False)
    # Your maximum heart rate decreases as you age, so we *subtract* a number
    # proportional to your age to determine your maximum heart rate.
    max_heart_bpm = 208 - 0.7 * age_years
    # Calculate the maximal oxygen consumption.
    # Source: https://en.wikipedia.org/wiki/VO2_max
    vo2_max = 15.3 * max_heart_bpm / resting_heart_bpm
    # Calculate a "cardio-respiratory fitness factor".
    car_resp_fitness_factor = constrain(1, 1.285 - 0.005 * vo2_max, 1.07)
    # The *additional* 0.84 when the user is not on a treadmill is due to air
    # resistance. You have to expend more energy to push against the air!
    if not on_a_treadmill:
        air_resistance_factor = 0.84
    else:
        air_resistance_factor = 0
    calories_burned = (0.95 * weight_kg + air_resistance_factor) * distance_km
    calories_burned *= car_resp_fitness_factor
    return calories_burned


def calc_swimming(weight_kg, duration_hours):
    """Calculate the number of calories burned while swimming."""
    style_index = get_index_of_one_of("Swimming style?", SWIMMING_STYLES)
    style_met = SWIMMING_STYLE_METS[style_index]
    return duration_hours * style_met * weight_kg


def print_calories_bar(label, calories):
    """Print a bar showing how many calories were burned."""
    # Calculate the number of squares with division, but always round down.
    # We can't print half of a square!
    number_of_squares = int(calories // CALORIES_PER_SQUARE)
    squares = "█" * number_of_squares
    # Calculate the number of calories that the squares don't represent i.e.
    # the remainder.
    remainder = calories % CALORIES_PER_SQUARE
    # Decide which remainder character to use. For instance, if the remainder
    # is 50 out of 100, print a special character that fills half the area of
    # a normal square.
    remainder_character = "░▌█"[round(remainder / (CALORIES_PER_SQUARE / 2))]
    # Put the label, squares, and remainder together via concatenation.
    print(
        label.ljust(CALORIES_BAR_LABEL_SIZE) + squares + remainder_character,
        round(calories),
    )


def run_command(command):
    """Run the command and return the number of calories burned."""
    weight_pounds = get_positive_number("Weight (in pounds)?")
    weight_kg = weight_pounds * KILOGRAMS_PER_POUND
    print_stat("Weight", weight_kg, "kilograms")

    distance_mi = None
    distance_km = None
    duration_hours = None

    if command == "biking" or command == "running":
        distance_mi = get_positive_number("Distance (in miles)?")
        distance_km = distance_mi * KILOMETERS_PER_MILE
        print_stat("Distance", distance_km, "kilometers")

    # The unnecessarily verbose condition is just to show that I can use
    # "and", "not", and "!=". De Morgan's laws tell us that the condition is
    # equivalent to: command == "biking" or command == "swimming"
    if not (command != "biking" and command != "swimming"):
        duration_minutes = get_positive_number("Duration (in minutes)?")
        duration_hours = duration_minutes / MINUTES_PER_HOUR

    if command == "biking":
        calories_burned = calc_biking(weight_kg, distance_mi, duration_hours)
    elif command == "running":
        calories_burned = calc_running(weight_kg, distance_km)
    else:
        # If the user isn't biking nor running, they must be swimming.
        calories_burned = calc_swimming(weight_kg, duration_hours)

    print_stat("Calories burned", calories_burned, "calories")
    return calories_burned


def main():
    """Ask the user for exercise information and print stats."""
    # Print the title and enough horizontal lines ("-") to underline the title.
    print(TITLE, "-" * len(TITLE), sep="\n")
    print(
        "When prompted by a label and question mark, such as 'Command?', "
        + "type the relevant information and press enter."
    )
    # Print the commands separated by commas.
    print("The commands are:", ", ".join(COMMANDS))

    exercises = []
    should_exit = False
    while not should_exit:
        print("")  # Print a blank line.
        command_index = get_index_of_one_of("Command?", COMMANDS)
        command = COMMANDS[command_index]
        if command == "exit":
            should_exit = True
        else:
            calories_burned = run_command(command)
            # Use a nested list to keep track of the exercises *and* the
            # calories burned.
            exercises.append([command, calories_burned])

    if len(exercises) > 0:
        total_calories_burned = 0
        print("\nCalories burned per exercise:")
        for index in range(len(exercises)):
            label = str(index + 1) + ". " + exercises[index][0]
            print_calories_bar(label, exercises[index][1])
            # Accumulate the calories burned from each exercise to get the
            # total calories burned.
            total_calories_burned += exercises[index][1]
        print("\nTotal:", round(total_calories_burned), "calories")
    elif len(exercises) == 0:
        print("\nNo exercises recorded.")


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