Exercise: Car Class

Table of Contents

Background

For this exercise you will develop a module containing a `Car` class. Instances of the class will be able to turn and drive forward. They will have three attributes: `x`, `y` and `heading`.

`x` and `y` will be the coordinates of the car object. For this assignment, `x` coordinates will increase as a car moves east; `y` coordinates will increase as a car moves south.

Heading indicates the direction in which the car will drive, in degrees. For this assignment, a heading of 0 indicates due north; a heading of 90 is due east; etc.

Trigonometric functions

For this exercise, you will need to use three trigonometry-related functions from the `math` module. `sin()` and `cos()` compute the sine and cosine, respectively, of a specified angle in radians. `radians()` converts a number of degrees to radians. So, for example, if you wanted to find the cosine of 30 degrees, here's how you could calculate it with these functions:

```
cos(radians(30))
```

Problem statement

Write a Python script containing a Car class. The class should have attributes `x`, `y` and `heading` and methods `turn()` and `drive()`.

Write a `sanity_check()` function that instantiates the class, invokes the methods, and prints the attributes, as described below.

Instructions

Create a Python script from scratch. Save the script as `car.py`. Follow the instructions below to populate this script.

`import` statement

Either import the entire `math` module, or just the following functions from that module: `cos`, `radians`, `sin`.

`Car` class

Create a class called `Car`. Define the following methods:

`__init__()` method

Define a method called `__init__()` (note the double underscores). Your method should have one required parameter (`self`) and three optional parameters as follows (please use these exact names):

- `x`: the starting x coordinate of the car, as a float.[1] Default: 0.
- `y`: the starting y coordinate of the car, as a float. Default: 0.
- `heading`: the starting heading, as a float. Default: 0.

Your `__init__()` method should set three attributes (`x`, `y`, and `heading`) to the values of their corresponding parameters.

`turn()` method

Define a method called `turn()` that has two required parameters, `self` and a number of degrees expressed as a float.[2] A positive number of degrees indicates a clockwise turn; a negative number of degrees indicates a counterclockwise turn. Use the following steps to assign a new value to the `heading` attribute (these can be combined into a single expression):

1. Add the specified number of degrees to the previous value of `heading`.
2. Reduce the result of step 1 modulo 360 (this ensures that `heading` is between 0 and 360).

For example, if `heading` is 270 and the number of degrees is 100, the `turn()` method should set `heading` to (270 + 100) mod 360, which is 10.

`drive()` method

Define a method called `drive()` that has two required parameters, `self` and a distance expressed as a float.

In the formulas below, `d` is the distance; `h` is the heading in radians (you will need to convert the heading from degrees to radians).

Update the `x` attribute by adding `d sin(h)` to the attribute's current value. (*Hint: the* `+=` *operator is your friend*).

Update the `y` attribute by subtracting `d cos(h)` from the attribute's current value. (*Hint: the* `-=` *operator is your friend*).

`sanity_check()` function

Define a `sanity_check()` function that takes no arguments. This function is not supposed to be part of the `Car` class—please de-indent the function header accordingly.

Inside this function, create an instance of the `Car` class. Have your instance follow these steps:

- Turn 90 degrees.
- Drive 10 units.
- Turn 30 degrees.
- Drive 20 units.

Print the location of your instance on one line and the heading on the next line, in the following format:

```
Location: 27.320508075688775, 9.999999999999996
Heading: 120
```

At the end of your function, return the instance you created.

`if __name__ == "__main__":` statement

At the end of your code, write an `if __name__ == "__main__":` statement that invokes your `sanity_check()` function.

Other instructions

- Please write docstrings for your class, each of your methods, and your function.
    - Your class docstring should start with a brief statement of the purpose of the class. It should then document the attributes of the class in an `Attributes:` section.
    - Your method/function docstrings should start with a brief statement of the function's purpose.

- If your method or function takes arguments, provide an `Args:` section to document them, including their names, data types, and a brief description of the purpose of each. You do not need to document `self` in the `Args:` section.
- If your method or function returns a value, write a `Returns:` section to describe the return value, including its data type and a brief description of its purpose.
- If your method causes side effects, include a `Side effects:` section to describe the side effects. Examples of side effects include setting or changing an attribute's value and printing information to the console.

Submitting your code

Upload your `car.py` class to Gradescope. An autograder script will give you (near-)instant feedback. If you did not pass all the test cases, you can revise your code and resubmit as many times as you want until the deadline.

Testing your code

To run your program within the VS Code built-in terminal, first make sure you have opened (in VS Code) the directory where your program is saved. If necessary, you can go to the VS Code File menu and select "Open..." on macOS or "Open Folder..." on Windows, and navigate to the directory where your program is.

Then, open the VS Code built-in terminal. Type `python3` (on macOS) or `python` (on Windows) followed by a space and the name of your program. Below is an example:

```
python3 car.py
```

`driving_range.py`

The program `driving_range.py` is designed to import your class and create a graphical representation of two instances of your `Car` class. Your class serves as the back-end for this program. The program depends upon your code following the naming conventions specified in these instructions.

`driving_range.py` requires the Tkinter module. If you installed Python 3.9 from Python.org, Tkinter should normally have been included. If you run into issues related to Tkinter, please contact the instructor.

To use `driving_range.py`, ensure that it is in the same directory as `car.py`. Then, open the VS Code built-in terminal and type `python3` (on macOS) or `python` (on Windows) followed by a space and the name of the program. Below is an example:

```
python3 driving_range.py
```

Two car objects appear as circles with arrows in the middle. By default, the cars are "self-driving". You can control them in both self-driving mode and regular mode. Here are the keys you can use to control the cars:

Action	**Red car key**	**Blue car key**
Toggle self-driving mode	Down	s
Drive forward	Up	w
Turn clockwise	Right	d
Turn counterclockwise	Left	a

[1] In other words, you can assume that the value of this parameter will be a float.

[2] In other words, you can assume that the value of this parameter will be a float.