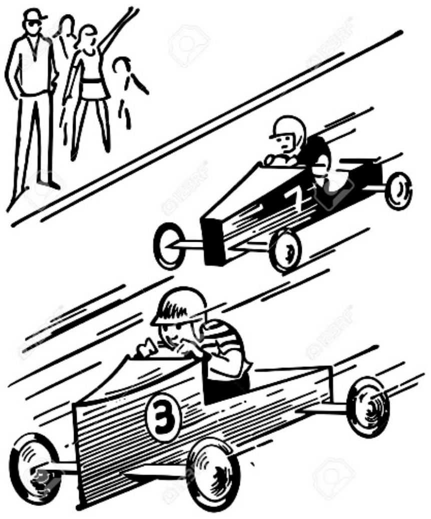
TPRG 1131 Project 1: Soapbox Derby



Release 1: Mar 18, 2019

A soapbox derby is a race for children driving gravity powered cars down a inclined ramp[1]. The soap boxes start at the top of the ramp and run out on a flat track toward a finish line. The physics problem of a block on a ramp, the acceleration due to gravity, friction and the equations of motion can be used to model the motion of a soapbox.

Develop a program that reads a data file where the first line gives information about the track, and the remaining lines give the details of the soapboxes, then determines the top speed and finish time of each soapbox.

Deliverables

This assignment is to be submitted to the DC Connect assignment folder as a single Zip file that includes two separate files:

1. The analysis phase as a MS Word or PDF document; alternatively you may hand in the analysis on paper on or before the project due date. Grading will be based on correct math and physics formulas, and neatly presented.
2. The Python source code file `soapbox.py` . Grading will be based on correct operation, and generally well organized layout.

Requirements

Program requirements

- The file name shall be: `soapbox.py` *Note: the Zip file name is not important.*
- Prompt the user for a data file name (optionally, the program should default to `soapboxes.csv` if the user enters nothing). The user is only prompted once for the file name (not once per record!).
- For each record in the data file, the program prints out several values:
    1. Whether or not the soapbox will actually roll; in other words, determine if the static friction f_s is less than the downward force f_d pulling the racer downward; if the static friction prevents the soapbox from starting, a message is printed, and processing of the record stops and the loop continues immediately to the next record. Hint: critical angle θ_c.
    2. Top speed, equivalent to the velocity at the bottom of the ramp, before the track.
        a. Initial velocity $v_0 = 0$ at the start
        b. Use the coefficient of kinetic friction f_k; rolling friction (wheels, bearings) is assumed lumped in with the coefficient of kinetic friction.
    3. Determine if the soapbox has enough speed to roll to the finish line. The force of friction must be recalculated because the angle is now 0. If the rolling distance is

[1] Illustration from (User "Retro Clipart" n.d.)

less than the track length, a message is printed, and processing of the record stops and the loop continues immediately to the next record.
4. The total time from start to finish.
5. Which soapbox won the race? Print the number of the soapbox with the shortest travel time. Soapboxes are numbered 1 to N based on the line number in the data file (first soapbox line is car number 1)

- Continuous loop operation: the program loops after every record until the end of file.
- The program must exit gracefully at end-of-file (no ugly error messages).
- The program must define and use one or more functions as required to compute the force, acceleration and velocity values. The functions are left to the discretion of the student, however modularize your program and avoid duplicated code.
- Constants like *g* must be defined near the top of the module instead of using the literal values throughout; Any constants available in the math module (e.g. π) must be defined using Python standard library modules (for example, you cannot write pi = 3.14).
- The program file shall conform to the coding style guide (see reference materials), including the proper heading at the top of the file.

Analysis section requirements:

Generally this section must be properly presented, as if you were writing a section of a textbook.

- Each equation or formula must be derived from the learning materials in the STEM1131 slides, the College Physics textbook or other sources (cite your sources!).
- All sources must be cited, including STEM1131 materials.
- The file must be prepared in MS Word or other document preparation software[2].
- If hand writing the equations and formulas, they must be written neatly and clearly in pen. Print out the document with blank areas for the formulas and carefully write them in.

Grading scheme (15 points)

4	Functionality	Does it fulfill its functional requirements?
2	Style	Conformance to rules 1.4 to 1.9 of the course style guide.
2	Technique to be used	Does the program use the specified technique? The file must be organized as functions to avoid code duplication, with the main program for input and output. The data is read from a file.
1	File name	As specified, packed in a Zip file (this is a 0 or 1 mark)
2	heading & comments	Conform to style guide 1.1 to 1.3
4	Analysis	"Show your work" (as in Electricity or STEM) on paper to develop the formulas and transform the track and soapbox measurements into the variables required by the formulas to compute the required answers (see the Deliverables section, item 1). The analysis section must be properly presented.

If the program crashes on start-up, the maximum program mark is 4.4/11 (analysis is separate).

[2] MS Word includes an equation editor. You can also use an online $\LaTeX$ math typesetting system such as https://www.overleaf.com or http://rogercortesi.com/eqn . Overleaf is better for beginners.

Program file structure

1. File heading
2. Import statement(s)
3. Function definitions
4. Main program

Test cases

The files `soapboxes.csv` and `soapboxes1.csv` each give the information for a specific event, with multiple racers on the same track (Figure 1). Each line has a predetermined set of floating point values to input as strings separated by commas. The line must be split into substrings, and each substring converted to a value of type `float`.

Line 1: Ramp length (m), ramp angle (degrees), track length (m) along the ground.

Lines 2 to end: Mass of car and driver (kg), coefficient of static friction *μs*, coefficient of kinetic friction *μk*, (coefficients of friction are dimensionless values, $0 < \mu \leq 1$).

Test cases and expected results are posted in a separate document: `testcases.pdf` .

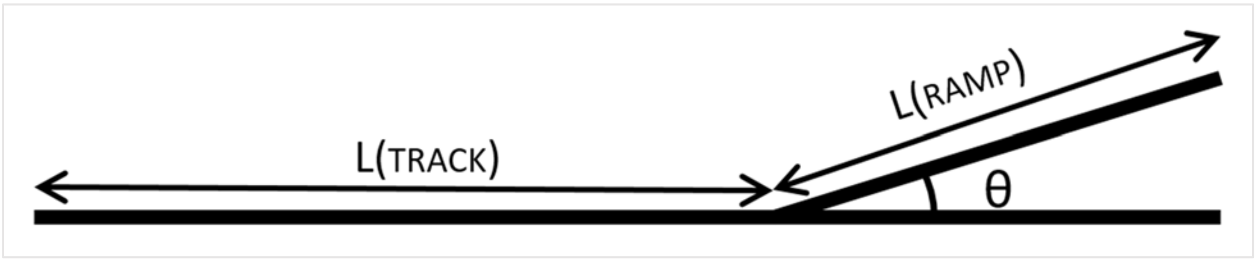


Figure 1: Ramp and track geometry

Program development

Programs are developed in sequential phases: analysis, design, coding, testing.

Analysis

Determine the formulas needed to convert the given data into the required results. Please refer to your notes from STEM1131 – Math and Physics (Schuett 2018a) as well as the OpenStax College Physics textbook ("5.1 Friction" 2017).

Linear motion can be calculated with the formulas used in STEM1131 (Schuett 2018b):

- v_f = final velocity (m/s)
- v_i = initial velocity (m/s)
- x = distance (m)
- a = acceleration (m/s^2)
- v_f = final velocity (m/s)

$$v_f = v_i + at$$

$$x = \frac{1}{2}(v_f + v_i)t$$

$$x = v_i t + \frac{1}{2}at^2$$

$$v_f^2 = v_i^2 + 2ax$$

Design

Determine the functions that will be required to transform the data in the file into the required answers. If you need to repeatedly compute some intermediate values, use functions to avoid code duplication. The use of functions reduces the size and complexity of the main program.

Determine the "flow" of the main program, expanding on the basic:

1. Prompt the user for the data file name
2. Open the data file
3. Read the first line to store the ramp and track parameters
4. In a loop, iterate over every record in the remainder of the file and print the answers
5. Close the file and exit gracefully

Note: You may use the Python standard library module `csv` to read the file.

Program development: edit – test – repeat

Proceed in small steps, coding each portion and testing as you go. Do not code everything at once then start debugging – don't launch your program from a tall bridge!

The test values given in the assignment handout are the same as those in the data file.

Function stubs: "stubbing out" a function is a development technique to make a required function temporarily return a known test value (e.g. 10.0). Later, when the rest of the program is complete, go back to the stub and complete the function development.

Thonny debug mode: Thonny lets you step through your program one statement at a time (F6 = Step Over) or observe the detailed execution of a statement (F7 = Step Into). The menu item View > Variables opens a separate pane where you can view the variables in the current scope and their values.

Debugging print statements: at various times, you may wonder why the output is not what you expect and would like to see the intermediate values. Place print function calls wherever you want to see an intermediate value, but flag those print functions with a comment such that you can comment them out in the final version.

Comment out, don't delete: Simply comment out code that you are not sure of, or that is not ready to test. Place a comment at the start of the block to remind yourself why. It's much easier to uncomment a few lines than to have to retype deleted code from memory.

References

"5.1 Friction." 2017. In *College Physics*, by Paul Peter Urone and Roger Hinrichs, 173–79. OpenStax.

Schuett, Dave. 2018a. "Friction on a Ramp: STEM1131 Week 6 Lecture 2 of 2." presented at the STEM1131 Fall 2018, Oshawa, October.

———. 2018b. "Gravity and Basic Motion: STEM1131 Week 4 Lecture 1 of 2." presented at the STEM1131 Fall 2018, Oshawa, October.

User "Retro Clipart." n.d. *Soap Box Derby*. Dreamstime.com. Accessed March 15, 2019. https://www.dreamstime.com/stock-illustration-soap-box-derby-image42095456.