CPE101 Moonlander

Winter 2019
@ Cal Poly SLO
By
Toshi

Decomposition

- Decompose a problem into simpler problems.
  - If a problem seems too difficult or too complex to solve, try to see if you can decompose the problem into easier and simpler problems.
- Write a function to solve one simple problem.
  - Decompose your program into functions.
  - Each function should solve only one problem.
  - If one function seems to be doing a lot, try to decompose that function into smaller functions.
    - If your function is larger than 30 - 40 lines, see if you can break the function into smaller functions.
- Avoid writing the same block of code repeatedly.
  - If you see the same block of code in multiple places, extract it into a separate function.

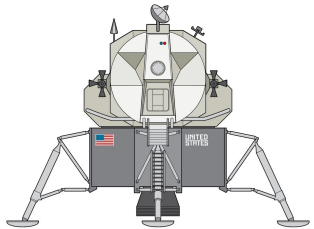
Decomposition Patterns

- Recursion
  - If the problem can be decomposed into smaller version of the same problem, you might be able to solve the problem recursively.
    - E.g. computing factorials, fibonacci series.
- Iteration Patterns
  - Map
    - If the problem can be solved by applying a function to each case repeatedly, try the map pattern.
  - Filter
    - If the problem can be solved by selecting values which meet specific criteria, try the filter pattern.
  - Reduce
    - If the problem can be solved by folding (accumulating) many values, try the reduce pattern.

Decomposition Example - Assignment #4

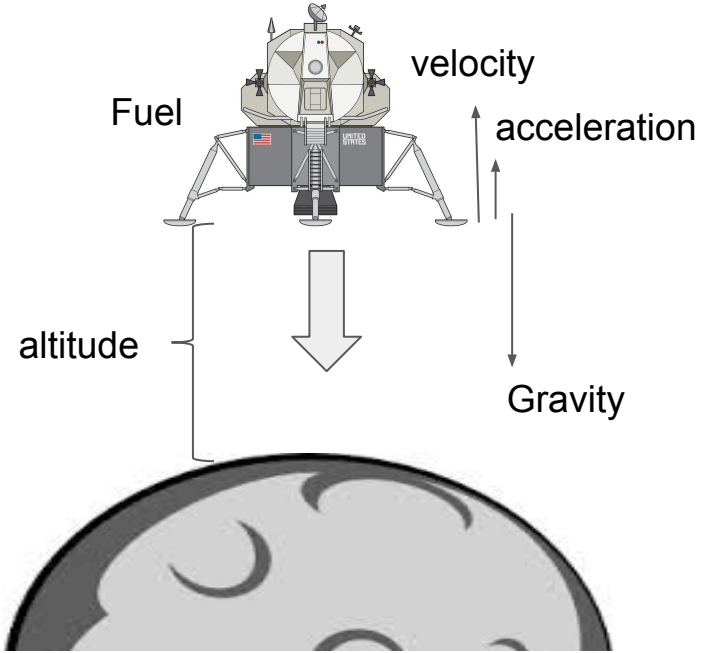
Requirements

- Take user inputs of integer for initial altitude and fuel
- Create Moon lander with
  - Initial Velocity
  - Initial Acceleration
  - Initial Fuel amount
  - Initial Altitude
- Take user inputs of integer for the rate of fuel flow
  - 0-9
- Per cycle, update:
  - the fuel amount based on the inputs
  - the velocity based on the inputs
  - the distance between the lander and the surface
- Output the result to screen.



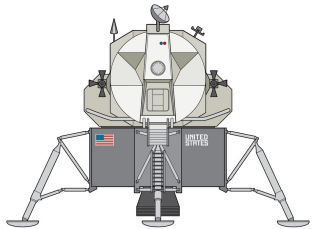
Let’s list data required to handled

- Moon lander object (a composite data type)
  - Fuel amount (int)
  - Velocity (float)
  - Acceleration (float)
  - Altitude (float)
- Gravity (float)
- Fuel flow rate (int)
  - 0 - 9
- Time (int)
  - One series of updates per tick (clock cycle)

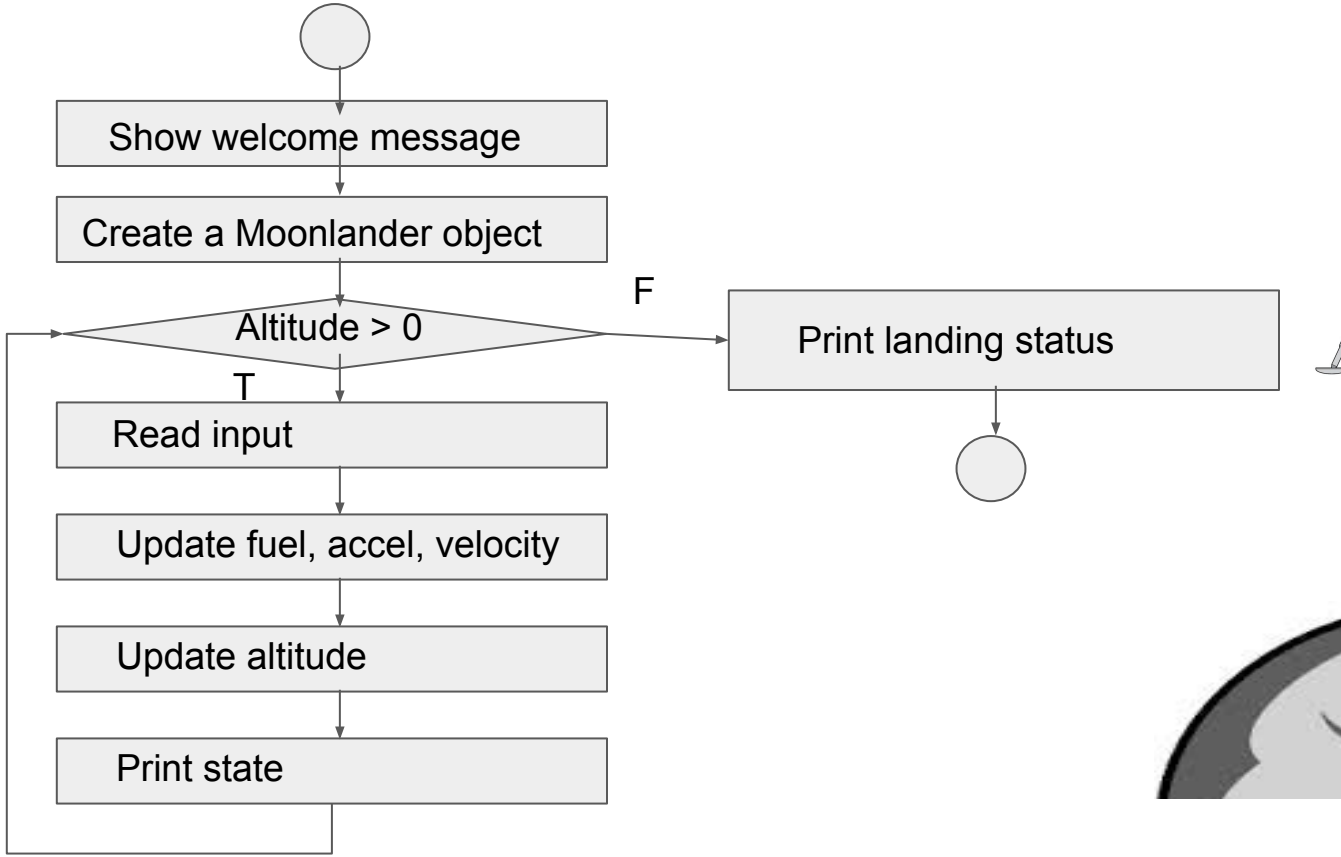


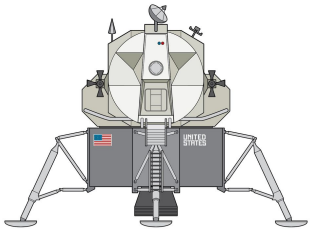
Let’s list functions needed to be performed

- Initialize the Moon lander
  - Create the moon lander with initial altitude, fuel, and velocity
- Read input
- Update fuel
- Update Acceleration
- Update velocity
- Update altitude
- Manage a session
  - Initialize a session
  - Check the end session criteria
  - Advance one clock cycle
  - Close the session
- print output



Flow chart


Show welcome message
Create a Moonlander object
Altitude > 0
F
T
Print landing status
Read input
Update fuel, accel, velocity
Update altitude
Print state



Functions

- show_welcome()
- get_fuel()
- get_altitude()
- display_state(time, altitude, velocity, fuel, fuel_rate)
- display_landing_status(velocity)
- get_fuel_rate(fuel)
- update_acceleration(gravity, fuel_rate)
- update_altitude(altitude, velocity, acceleration)
- update_velocity(velocity, acceleration)
- update_fuel(fuel, fuel_rate)
- main()

How to write a function (Design Recipe)

1. Define data
    a. What are the data the function needs to handle
2. Write the signature, header, purpose of the function
3. Write test cases
    a. Identify typical cases as well as edge cases
4. Decompose the function into parts based on data values
    a. Trivial case and non-trivial case
    b. Typical case and non-typical case
5. Write the function body
6. Test