Moonlander

CPE101

Functional 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.

Functional 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.

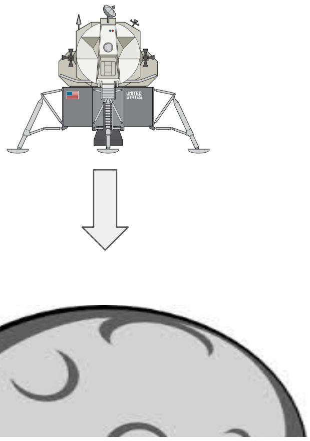
Functional Decomposition

- 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.

Functional Decomposition Example - Project 1

Requirements

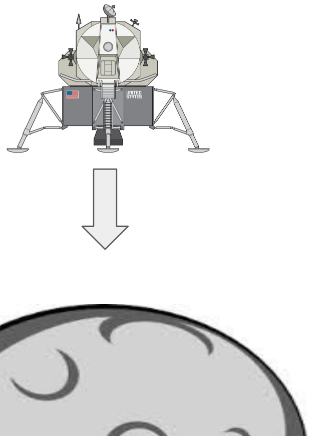
- Take user inputs for initial altitude and fuel



Functional Decomposition Example - Project 1

Requirements

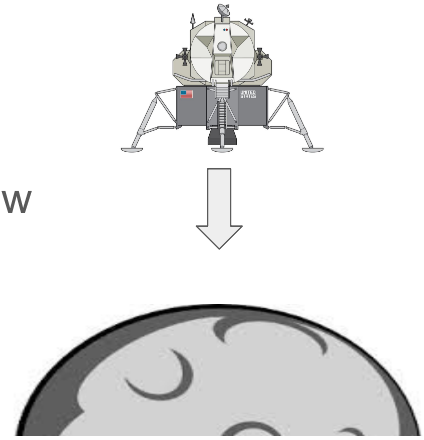
- Take user inputs of integer for initial altitude and fuel
- Create a Moon lander with
  - Initial Velocity
  - Initial Acceleration
  - Initial Fuel amount
  - Initial Altitude



Functional Decomposition Example - Project 1

Requirements

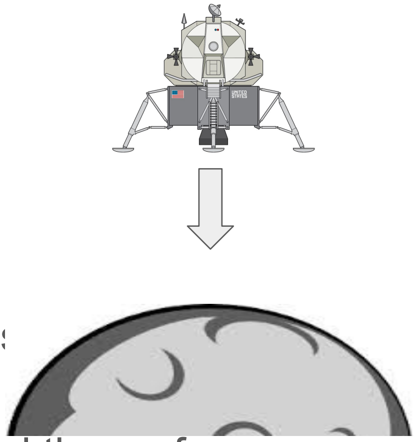
- Take user inputs for initial altitude and fuel
- Create a Moon lander with
  - Initial Velocity
  - Initial Acceleration
  - Initial Fuel amount
  - Initial Altitude
- Take user inputs for the rate of fuel flow
  - 0-9



Functional Decomposition Example - Project 1

Requirements

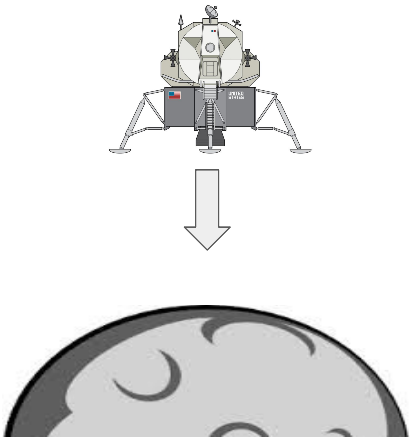
- Take user inputs for initial altitude and fuel
- Create a Moon lander with
  - Initial Velocity
  - Initial Acceleration
  - Initial Fuel amount
  - Initial Altitude
- Take user inputs for the rate of fuel flow
  - 0-9
- Per cycle, update:
  - the fuel amount based on the inputs
  - the velocity based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



Functional Decomposition Example - Project 1

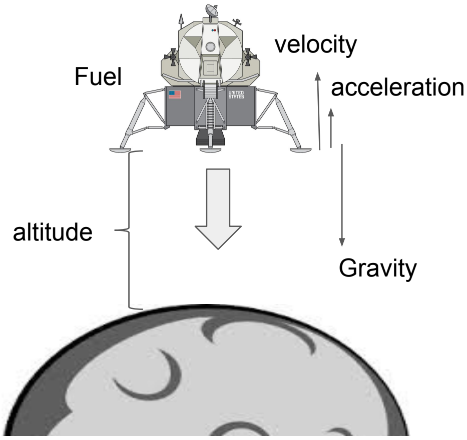
Requirements

- Take user inputs for initial altitude and fuel
- Create a Moon lander with
  - Initial Velocity
  - Initial Acceleration
  - Initial Fuel amount
  - Initial Altitude
- Take user inputs 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 on screen.



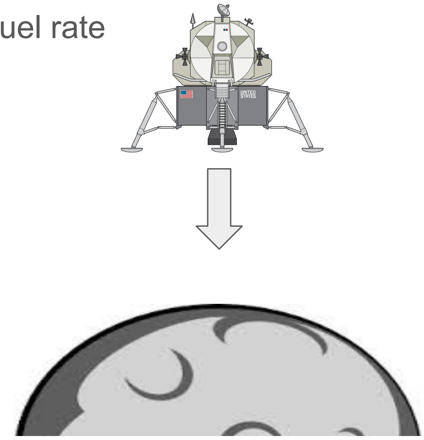
Let’s list data required to handled

- A Moon lander object (a compound data)
  - 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)

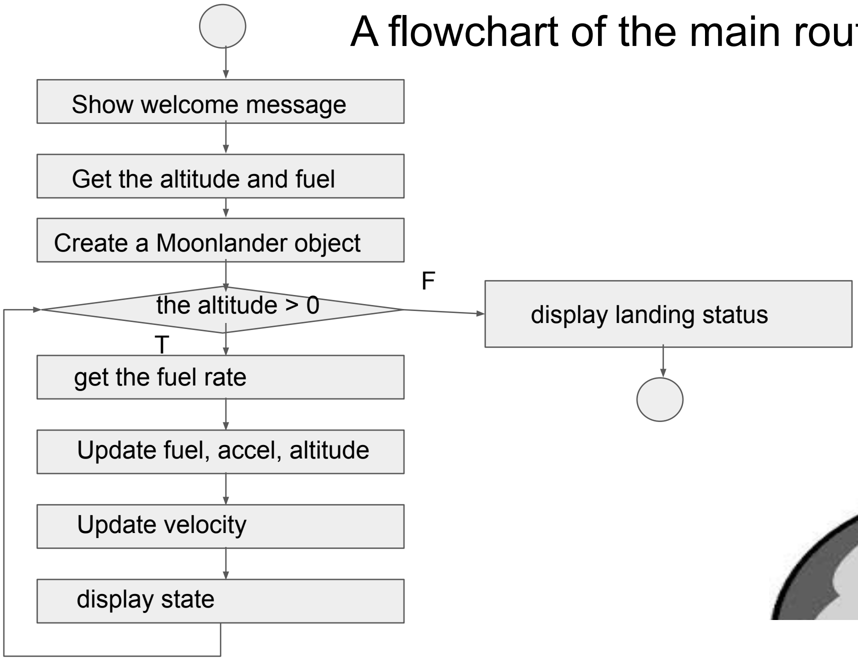


Let’s list operations needed to be performed

- Initialize a Moon lander
  - Create a moon lander with initial altitude, fuel, and velocity
- Get inputs from the user:get altitude, get fuel, get fuel rate
- 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



A flowchart of the main routine



Show welcome message

Get the altitude and fuel

Create a Moonlander object

the altitude > 0

T

get the fuel rate

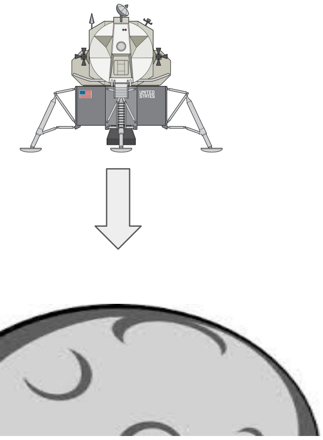
Update fuel, accel, altitude

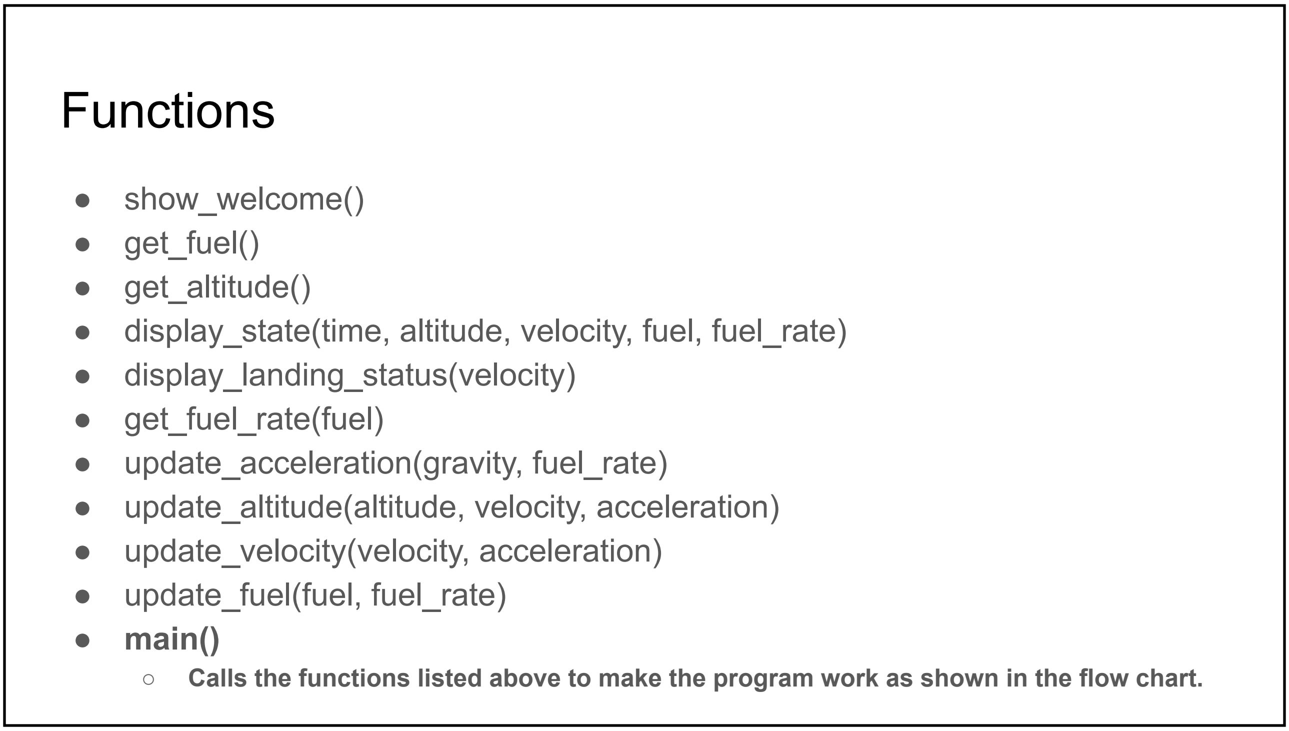
Update velocity

display state

F

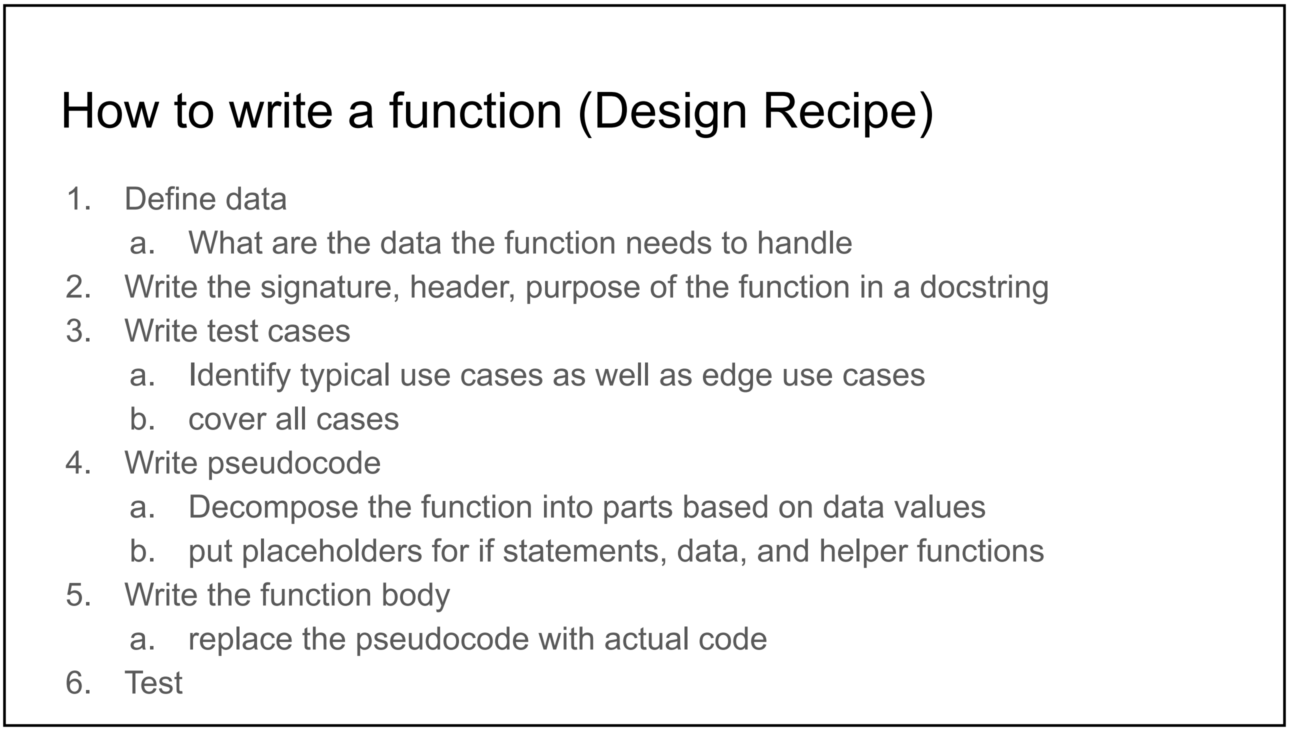
display landing status





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()**
  - **Calls the functions listed above to make the program work as shown in the flow chart.**



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 in a docstring
3. Write test cases
   a. Identify typical use cases as well as edge use cases
   b. cover all cases
4. Write pseudocode
   a. Decompose the function into parts based on data values
   b. put placeholders for if statements, data, and helper functions
5. Write the function body
   a. replace the pseudocode with actual code
6. Test