Functional Decomposition

Programs are getting larger…

- Our programs are starting to get large enough that we can't hold all the details of the implementation in our heads
- We need a toolset to help us break down large programming problems into smaller, more manageable sub-problems

Breaking up our programs

- When should break up our programs into small problems?
    - …before we try to write them!
- Why?
    - We can create **re-usable, debuggable** pieces that save us time
- This means that we should integrate this notion into our early design process

Functional Decompostion

- ***Functional decomposition*** * works to break up a large programming assignment into smaller sub-problems
  - Working from the ***abstract*** to the ***concrete***
- This is also known as ***top-down design***

* Special thanks to Dr. McCormick for the use of his materials from the book: Dale, Weems, and McCormick. Programming and Problem Solving with ADA 95.

Design

- ***Abstract step*** – a list of major steps in our solution
- ***Concrete step*** – algorithmic steps that can be translated directly into Python code
  - …or, the code of any programming language!

Design

- We need to break down a solution from a series of very high-level abstract steps into concrete algorithmic steps that can be implemented in code
- Each of the major steps becomes an independent sub-problem that we can work on independently

Design

- Why would we want to do this?
  - It’s much easier to focus on one problem at a time.
  - Can get lost in large specifications without a plan

Design

- We can create a ***hierarchical solution tree*** that goes from the most abstract steps to the concrete steps
- Each level of the tree is a complete solution to the problem that is less abstract than the level above it
  - This is known as ***functional equivalence***

Hierarchical Solution Tree

- Concrete steps are shaded

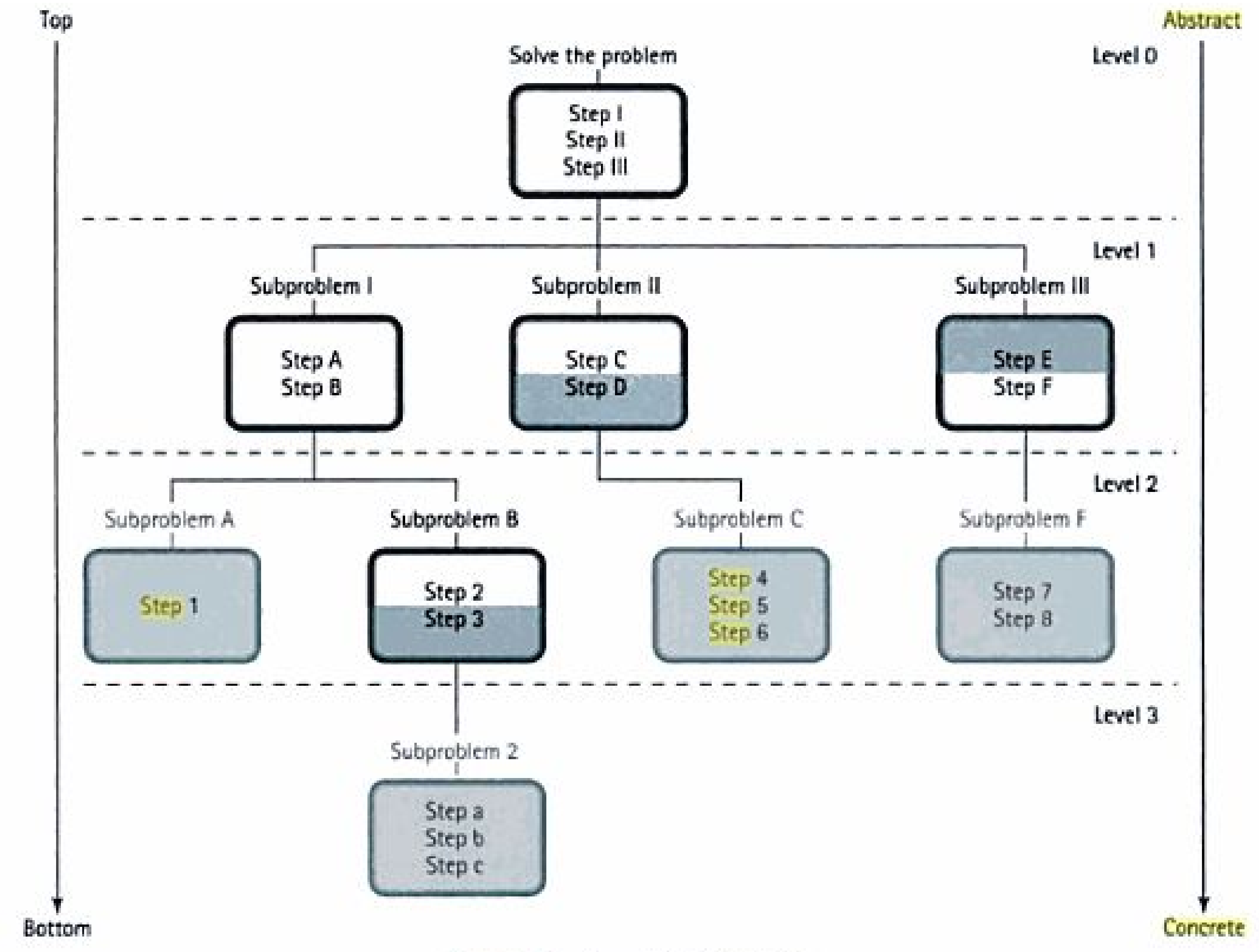


Figure 4-3 *Hierarchical Solution Tree*

Modules

- Each box represents a ***module***
- Modules are a self-contained collection of steps that solves a problem or subproblem
    - They can contain both concrete and abstract steps
    - Concrete steps are often written in pseudocode

Design Warm Up – mileage.py

- Dr. Mobile asks you to write a program that asks for the starting and ending mileage as well as the total gasoline consumed.
- It then calculates the MPG and prints a nice message regarding the MPG of the car.
    - Recall, mpg = (ending mileage – start mileage) / gas consumed
- It must also print out if the car can be legally driven as-is in California by 2016 (>= 35 MPG)

Mileage Solution Tree

- Level 0 Solve the Problem

1. Get Data
2. Calculate MPG
3. Print Data

- No steps are shaded, so these are all abstract steps (must be broken down more to solve in code

Mileage Solution Tree

- Level 1 Get Data

1. Ask for starting mileage
2. Ask for ending mileage
3. Ask for gas consumed

- All steps are shaded, so these are all concrete steps that we can translate directly into Python code

Get Data

```
startMileage = int(input(“Please enter the starting mileage: “))
endMileage = int(input(“Please enter the ending mileage: “))
gasConsumed = int(input(“Please enter the gas consumed: “))
```

Mileage Solution Tree

- Level 1 Calculate MPG

1. Mpg = (end-start)/gas consumed

- This is a concrete step that we can translate directly into Python code

Calculate MPG

mpg = (endMileage – startMileage) / gasConsumed

Mileage Solution Tree

- Level 1 Print Data

1. Print mpg
2. Print if car can be driven in California

- We still need to break down step 2 into something more concrete

Mileage Solution Tree

- Level 2   Drive in California

```
1.   if mpg < 35:
         print cannot drive
     else
         print can drive
```

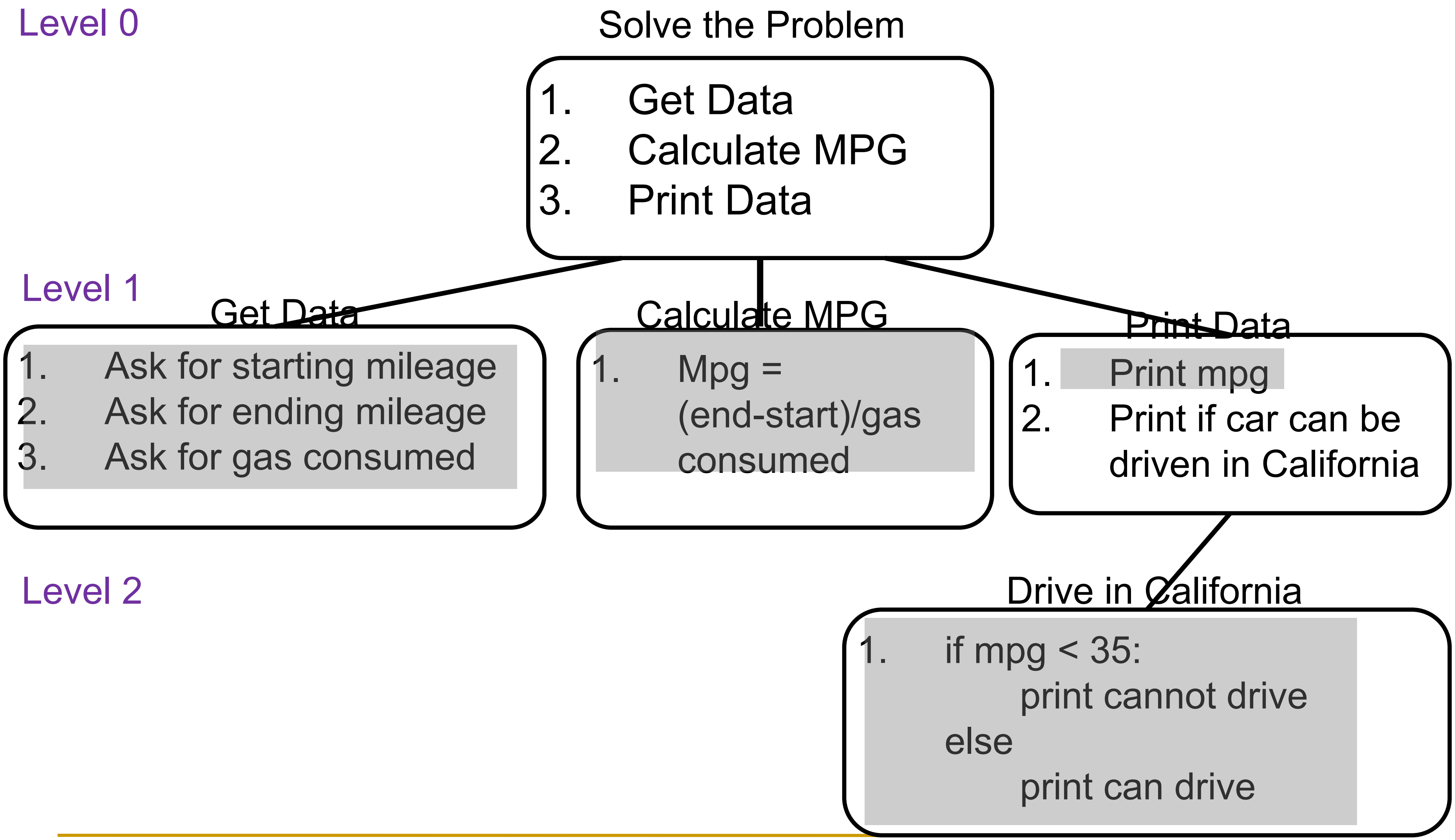
- These are concrete

Print Data

```
# Print mpg
print(“The total mpg is: “, mpg)

#Drive in California
If mpg < 35:
    print(“Cannot drive in California.”)
else
    print(“Can drive in California!”)
```

Total Table



Level 0

Solve the Problem

1. Get Data
2. Calculate MPG
3. Print Data

Level 1

Get Data

1. Ask for starting mileage
2. Ask for ending mileage
3. Ask for gas consumed

Calculate MPG

1. Mpg = (end-start)/gas consumed

Print Data

1. Print mpg
2. Print if car can be driven in California

Level 2

Drive in California

```
1.   if mpg < 35:
          print cannot drive
     else
          print can drive
```

Next step

- Each module in our solution tree could be broken into a function
- How do we decide?
  - For now, let's not divide up the input and print portions into their own functions
    - Relatively easy to leave them as they are
  - That leaves the calculation and California portions…