Decomposition
Chris Piech and Mehran Sahami
CS106A, Stanford University
1. Be able to approach a problem “top down” by using decomposition (aka stepwise refinement)
First, a cool program
Quick review
Karel the Robot
def main():
    go_to_moon()

def go_to_moon():
    build_spaceship()
    # a few more steps

def build_spaceship():
    # todo
    put_beeper()
For Loops

def main():
    # repeats the body 99 times
    for i in range(99):
        # the "body"
        put_beeper()
def main():
    # while condition holds runs body
    # checks condition after body completes
    while front_is_clear():
        move()
def main():

    # If the condition holds, runs body
    if front_is_clear():
        move()
def main():
    # If the condition holds,
    if beepers_present():
        # do this
        pick_beeper()
    else:
        # otherwise, do this
        put_beeper()
The Full Karel

<table>
<thead>
<tr>
<th>Base Karel commands:</th>
<th>Conditions:</th>
</tr>
</thead>
<tbody>
<tr>
<td>move()</td>
<td>if condition:</td>
</tr>
<tr>
<td>turn_left()</td>
<td>code run if condition passes</td>
</tr>
<tr>
<td>put_beeper()</td>
<td>if condition:</td>
</tr>
<tr>
<td>pick_beeper()</td>
<td>code block for &quot;yes&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Karel program structures:</th>
<th>else:</th>
</tr>
</thead>
</table>
| # Comments can be included in any part | code block for "no"
| # of a program. They start with a # |
| # and include the rest of the line. |
| def main():               | |
| code to execute          | |
| declarations of other functions |

<table>
<thead>
<tr>
<th>Names of the conditions:</th>
<th>Loops:</th>
</tr>
</thead>
<tbody>
<tr>
<td>front_is_clear()</td>
<td>for i in range(count):</td>
</tr>
<tr>
<td>beepers_present()</td>
<td>code to execute</td>
</tr>
<tr>
<td>beepers_in_bag()</td>
<td>code to repeat</td>
</tr>
<tr>
<td>left_is_clear()</td>
<td>while condition:</td>
</tr>
<tr>
<td>right_is_clear()</td>
<td>code to repeat</td>
</tr>
<tr>
<td>facing_north()</td>
<td></td>
</tr>
<tr>
<td>facing_south()</td>
<td></td>
</tr>
<tr>
<td>facing_east()</td>
<td></td>
</tr>
<tr>
<td>facing_west()</td>
<td></td>
</tr>
</tbody>
</table>

| Function Declaration:   | |
|-------------------------| |
| def name():             | |
| code in the body of the function. |

| Extra Karel Commands:   | |
|-------------------------| |
| paint_corner(COLOR_NAME) |
| corner_color_is(COLOR_NAME) |
End review
def friday():
    # heres our plan
    decomposition()
    mountain_karel()
    rhoomba_karel()
    if extra_time():
        word_search_karel()
def friday():
    # here's our plan
    decomposition()
    mountain_karel()
    rhoomba_karel()
    if extra_time():
        word_search_karel()
def friday():
    # heres our plan
    decomposition()
    mountain_karel()
    rhoomoba_karel()
    if extra_time():
        word_search_karel()
def friday():
    # heres our plan
    decomposition()
    mountain_karel()
    rhoomba_karel()
    if extra_time():
        word_search_karel()
Mountain Karel

Muhammed ibn Musa Al Kwarizmi
Mountain Karel
Do your thing.
Pro Tips

- A good function should do "one conceptual thing"

- Know what it does by looking at its name

- Less than 10 lines, 3 levels of indentation

- Reusable and easy to modify

- Well commented

There are two types of programs. One is so complex, there is nothing obvious wrong with it. One is so clear, that this obviously nothing wrong with it.
Aside: Common Errors

Lather, Rinse, Repeat
def turn_to_wall():
    while left_is_clear():
        turn_left()
def turn_to_wall():
    while left_is_clear():
        turn_left()
def turn_to_wall():
    while left_is_clear():
        turn_left()
def turn_to_wall():
    while left_is_clear():
        turn_left()
Pre/Post that Don’t Match

```python
def jump_hurdles:
    for i in range(8):
        if front_is_clear():
            move()
        else:
            jump_hurdle()
```

What do you assume here?

Does the “post condition” match?
def friday():
    # heres our plan
    decomposition()
    mountain_karel()
    rhoomba_karel()
    if extra_time():
        word_search_karel()
def friday():
    # heres our plan
    decomposition()
    mountain_karel()
    rhoomba_karel()
    if extra_time():
        word_search_karel()
Rhoomomba Karel

- Write a **Roomba** Karel that sweeps the entire world of all beepers.
  - Karel starts at (1,1) facing East.
  - The world is rectangular, and some squares contain beepers.
  - There are no interior walls.
  - When the program is done, the world should contain 0 beepers.
  - Karel's ending location does not matter.

- How should we approach this tricky problem?
Possible Algorithm 1
Possible Algorithm 4

Piech and Sahami, CS106A, Stanford University
Rhoomba Karel

Welcome to Karel!

Start Program
Reset Program
Load World
New World
Edit World

Slow Fast
def friday():
    # heres our plan
    decomposition()
    mountain_karel()
    rhoombaba_karel()
    if extra_time():
        word_search_karel()
def friday():
    # here's our plan
    decomposition()
    mountain_karel()
    rhoomba_karel()
    if extra_time():
        word_search_karel()
def friday():
    # here's our plan
    decomposition()
    mountain_karel()
    rhoomba_karel()
    if extra_time():
        word_search_karel()
Happy Friday