Functions and Decomposition

CS106AP Lecture 3
Day 1!

Programming Basics

The Console

Images

Data structures

Midterm

Graphics

Object-Oriented Programming

Everyday Python

Life after CS106AP!
Roadmap

Programing Basics
- Variables
- Control Flow
- Decomposition + Functions
- Karel the Robot

The Console

Images

Graphics

Object-Oriented Programming

Data structures

Midterm

Everyday Python

Life after CS106AP!
Today’s questions

Why do we use decomposition?

How might we approach breaking down (decomposing) difficult problems?
Today’s topics

1. Review
   Functions
   Control flow
   make_beeper_row_better()  

2. Top-down decomposition

3. Examples!
   MakeGardenKarel
   if *enough time*:
     HurdlesKarel

4. What’s next?
Review
Functions
**Definition**

**function**
A smaller part of your program that solves a specific sub-problem
The structure of a function

```python
def my_function_name():
    
    """
    Write a good description for your function here.
    """
    # Put other Karel commands here

    ...

def caller_function():
    my_function_name() # Function call
```
Control Flow
Definition

control flow structures
Code features that affect the order, or flow, in which the lines of code in a program happen
Control Flow: if statements
The structure of an if statement

```python
if boolean expression:
    # Do something if condition is True
```
Control Flow: while loops
The structure of a **while loop**

```python
while boolean expression:
    # Do something while condition is True
```
The structure of a **while loop**

```python
while boolean expression:
    # Do something while condition is True
```

Unlike `if` statements, if the condition is still true after all of the lines inside the **while** loop finish, the code repeats!
Boolean expressions in Karel
### Karel functions that evaluate to boolean expressions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>front_is_clear()</code></td>
<td>Is there no wall in front of Karel?</td>
</tr>
<tr>
<td><code>left_is_clear()</code></td>
<td>Is there no wall to Karel’s left?</td>
</tr>
<tr>
<td><code>right_is_clear()</code></td>
<td>Is there no wall to Karel’s right?</td>
</tr>
<tr>
<td><code>on_beeper()</code></td>
<td>Is there a beeper on the corner where Karel is standing?</td>
</tr>
<tr>
<td><code>facing_north()</code></td>
<td>Is Karel facing north?</td>
</tr>
<tr>
<td><code>facing_south()</code></td>
<td>Is Karel facing south?</td>
</tr>
<tr>
<td><code>facing_east()</code></td>
<td>Is Karel facing east?</td>
</tr>
<tr>
<td><code>facing_west()</code></td>
<td>Is Karel facing west?</td>
</tr>
</tbody>
</table>
A note about `not`

- We can use `not` in front of a boolean expression to negate it!

```python
if not boolean expression:
    # Do something if expression is not True
```
A note about not

- We can use `not` in front of a boolean expression to negate it!

```python
if not front_is_clear():
    turn_around()
```
A better square
Think/Pair/Share:
How can we use a while loop inside `draw_square()` to reduce repeated code?
Let’s put everything together!
If statements *inside* while loops

Sometimes we may not want all of the lines inside the while loop to happen...
If statements *inside* while loops

Sometimes we may not want all of the lines inside the while loop to happen...

```python
while ________________:
    ...

if ________________:
    # Do something
    ...
```
If statements *inside* while loops

Sometimes we may not want all of the lines inside the while loop to happen...

```
while _______________
    ...
    if _______________
        # Do something
        ...
    ...
```
make_beeper_row()
def make_beeper_row():

    while front_is_clear():
        put_beeper()
        move()
        put_beeper()
What if some corners already have beepers?
What if some corners already have beepers?

- Combine what we know about `if` and `while`!

- Potentially useful:
  - `on_beeper()`
  - `not`
What if some corners already have beepers?

- Combine what we know about `if` and `while`!

- Potentially useful:
  - `on_beeper()`
  - `not`

Write your code in the `MakeBeeperRowKarel.py` file in the `pycharm_intro` folder you downloaded.
Your turn to try!

make_beeper_row_better()
def make_beeper_row():

    while front_is_clear():
        put_beeper()
        move()
        put_beeper()
Common bugs

issues
Common bugs

- Forgetting not
Common bugs

- Forgetting `not`
- Forgetting an if statement around the first `put_beeper()`
Common bugs

- Forgetting `not`
- Forgetting an if statement around the first `put_beeper()`
- Forgetting to call your function!
Common bugs

- Forgetting `not`
- Forgetting an if statement around the first `put_beeper()`
- Forgetting to call your function!
- Getting stuck in an infinite loop
Common bugs

- Forgetting not
- Forgetting an if statement around the first `put_beeper()`
- Forgetting to call your function!
- Getting stuck in an infinite loop

**Definition**

**infinite loop**

When your code gets stuck inside a loop because the condition never evaluates to false
Common bugs

- Forgetting **not**
- Forgetting an if statement around the first `put_beeper()`
- Forgetting to call your function!
- Getting stuck in an infinite loop
Why do we use decomposition?
Definition

decomposition
The process of breaking down a large problem (program) into smaller parts (functions)
Having multiple small functions makes your program easier to read and understand
And (as we’ll see in a bit) also easier to write!

**Style note**

decomposition
Having multiple small functions makes your program easier to read and understand
The structure of a program

```python
import ...

def main():
    ...

if __name__ == '__main__':
    ...
```
The structure of a program

import ...

def main():
    ...

if __name__ == '__main__':
    ...

Definition

computer program

A set of instructions for the computer – an “app.” These instructions are also called lines of “source code.”
The structure of a program

```
import ...

def main():
    ...

if __name__ == '__main__':
    ...
```

import modules (more on that later in the quarter)
The structure of a program

import ...

def main():
    ...

if __name__ == '__main__':
    ...

define functions
The structure of a program

import ...

def main():
    ...

if __name__ == '__main__':
    ...

call your main function (you won’t need to worry about this)
The structure of a program

import ...

def main():
    ...

if __name__ == '__main__':
    ...

This is where we’ve been writing our code!
The structure of a program

def main():
    ...

This is where we’ve been writing our code!
The structure of a program

```python
def function_one():
    ...

def function_two():
    ...

def main():
    function_one()
    function_two()
    ...
```

Programs are made up of multiple functions.
NOTE: Main is a function!

def main():
    """
    Write a good description for your function here.
    """
    put_beeper()
    move()
    move()
    move()
    turn_left()
    turn_left()
    ...

The structure of a program

import ...

def main():
    ...

if __name__ == '__main__':
    ...

call your main function (you won’t need to worry about this)
How might we decompose difficult problems?

Top-down decomposition!
Top-down decomposition

● “Divide-and-conquer”
  ○ Break the problem down into smaller parts
  ○ Ask: What are the steps that make up the larger problem? What tasks are repeated and might make good functions?
Top-down decomposition

- “Divide-and-conquer”

- Plan out your milestones (functions) first before writing them:
  - What are the pre-conditions and post-conditions for each function?
  - Which functions will call which?
Top-down decomposition

- “Divide-and-conquer”

- Plan out your milestones (functions) first before writing them

- Use the “blackbox model” for functions you’re not working on
  - When working on a particular function, assume that the others exist and already do what you want
Top-down decomposition

- “Divide-and-conquer”

- Plan out your milestones (functions) first before writing them

- Use the “blackbox model” for functions you’re not working on
Steps

1. Plan your milestones (functions).

2. Write all of the function definitions and their pre-conditions and post-conditions in the function comments.

3. Work on one function at a time: write its code, test it, and debug as necessary before moving on to the next milestone (function).

4. Make sure your program works on all possible scenarios (worlds)!
Plant Garden Karel
PlantGardenKarel

begin

end
begin
PlantGardenKarel
PlantGardenKarel
PlantGardenKarel
PlantGardenKarel

Repeat!
PlantGardenKarel
Steps

1. Plan your milestones (functions).

2. Write all of the function definitions and their pre-conditions and post-conditions in the function comments.

3. Work on one function at a time: write its code, test it, and debug as necessary before moving on to the next milestone (function).

4. Make sure your program works on all possible scenarios (worlds)!
Definition

pseudocode
A non-code outline of your code!
**Definition**

**pseudocode**
A non-code outline of your code!

*Use this for milestone planning!*
Steps

1. Plan your milestones (functions).

2. Write all of the function definitions and their pre-conditions and post-conditions in the function comments.

3. Work on one function at a time: write its code, test it, and debug as necessary before moving on to the next milestone (function).

4. Make sure your program works on all possible scenarios (worlds)!
Steps

1. Plan your milestones (functions). [whiteboard]

2. Write all of the function definitions and their pre-conditions and post-conditions in the function comments.

3. Work on one function at a time: write its code, test it, and debug as necessary before moving on to the next milestone (function).

4. Make sure your program works on all possible scenarios (worlds)!
Steps

1. Plan your milestones (functions).

2. Write **all of the function definitions** and their pre-conditions and post-conditions in the function comments.

3. Work on **one function at a time**: write its code, test it, and debug as necessary before moving on to the next milestone (function).

4. Make sure your program works on all possible scenarios (worlds)!
Steps

1. Plan your milestones (functions).

2. Write **all of the function definitions** and their pre-conditions and post-conditions in the function comments.

3. Work on one function at a time: write its code, test it, and debug as necessary before moving on to the next milestone (function).

4. Make sure your program works on all possible scenarios (worlds)! Make sure your pre- and post-conditions match!
Steps

1. Plan your milestones (functions).

2. Write all of the function definitions and their pre-conditions and post-conditions in the function comments.

3. Work on one function at a time: write its code, test it, and debug as necessary before moving on to the next milestone (function).

4. Make sure your program works on all possible scenarios (worlds)!
Steps

1. Plan your milestones (functions).

2. Write all of the function definitions and their pre-conditions and post-conditions in the function comments.

3. Work on one function at a time: write its code, test it, and debug as necessary before moving on to the next milestone (function).

4. Make sure your program works on all possible scenarios (worlds)!
HurdlesKarel
Hurdles Karel

begin

end
Karel will always start with a hurdle directly to its right.
Hurdles

Four moves, two types
Follow wall on right
Hurdles

Karel

Follow wall until blocked
Steps

1. Plan your milestones (functions).

2. Write **all of the function definitions** and their pre-conditions and post-conditions in the function comments.

3. Work on **one function at a time**: write its code, test it, and debug as necessary before moving on to the next milestone (function).

4. Make sure your program works on all possible scenarios (worlds)!
Steps

1. Plan your milestones (functions).

2. Write all of the function definitions and their pre-conditions and post-conditions in the function comments.

3. Work on one function at a time: write its code, test it, and debug as necessary before moving on to the next milestone (function).

4. Make sure your program works on all possible scenarios (worlds)!
Steps

1. Plan your milestones (functions). [whiteboard]

2. Write all of the function definitions and their pre-conditions and post-conditions in the function comments.

3. Work on one function at a time: write its code, test it, and debug as necessary before moving on to the next milestone (function).

4. Make sure your program works on all possible scenarios (worlds)!
Steps

1. Plan your milestones (functions).

2. Write all of the function definitions and their pre-conditions and post-conditions in the function comments.

3. Work on one function at a time: write its code, test it, and debug as necessary before moving on to the next milestone (function).

4. Make sure your program works on all possible scenarios (worlds)!
Steps

1. Plan your milestones (functions).

2. Write all of the function definitions and their pre-conditions and post-conditions in the function comments.

3. Work on one function at a time: write its code, test it, and debug as necessary before moving on to the next milestone (function).

4. Make sure your program works on all possible scenarios (worlds)!
Steps

1. Plan your milestones (functions).

2. Write all of the function definitions and their pre-conditions and post-conditions in the function comments.

3. Work on one function at a time: write its code, test it, and debug as necessary before moving on to the next milestone (function).

4. Make sure your program works on all possible scenarios (worlds)!
Steps

1. Plan your milestones (functions).

2. Write all of the function definitions and their pre-conditions and post-conditions in the function comments.

3. Work on one function at a time: write its code, test it, and debug as necessary before moving on to the next milestone (function).

4. Make sure your program works on all possible scenarios (worlds)!
What’s next?
Wrapping up Karel tomorrow!