As you arrive:

1. Start up your computer and plug it in.
2. ***Log into Angel*** and go to CSSE 120. Do the ***Attendance Widget*** – the PIN is on the board.
3. Go to the ***Course Schedule*** web page. Open the ***Slides*** for today if you wish.
4. Checkout today's project: `Session14_NestedLoopsAndMutators`

Session 14

Nested Loops and Mutators, Line Following

Nested Loops

Mutators

- Box-and-pointer diagrams

Line following

Checkout today's project:

`Session14_NestedLoopsAndMutators`

Are you in the Pydev perspective? If not:

`Window ~ Open Perspective ~ Other` then `Pydev`

Messed up views? If so:

`Window ~ Reset Perspective`

Troubles getting today's project? If so:

No SVN repositories view (tab)? If it is not there:

`Window ~ Show View ~ Other`
then `SVN ~ SVN Repositories`

1. ***In your SVN repositories view (tab), expand your repository** (the top-level item) if not already expanded.*
   - If no repository, perhaps you are in the wrong Workspace. Get help.

2. ***Right-click on today's project**, then select* `Checkout`*.*
   *Press **OK** as needed.* The project shows up in the
   `Pydev Package Explorer`
   to the left. Expand and browse the modules under `src` as desired.

Wait-for-event Loop Pattern

pre-loop computation
while [the event has NOT occurred]:
        sleep for a bit
post-loop computation

- Used frequently in:
  - Robotics
  - GUI's (responding to events)
  - Operating systems
  - Any application where there are "events"

We saw this pattern in the robot example from last session. Here is another example. You can run this example in ***m1_wait_for_event_example*** in today's project.

```
def waitForEvent(robot, line):
    """
    Busy-waits for the given robot (simulated by a
    Rectangle) to cross the given vertical Line.
    """
    seconds_to_sleep_between_event_checks = 0.01

    while robot.getCenter().getX() < line.getP1().getX():
        time.sleep(seconds_to_sleep_between_event_checks)
```

Note the repeated-dot notation. Make sure you understand it!

Nested Loops

This code and subsequent examples appear in the *m2_nested_loops* module of the project you checked out today.

- A ***nested if*** is an *if* inside the body of another *if*.
- A ***nested loop*** is a *loop* inside the body of another *loop*.
- Trace the code below. What does it print when *main* runs?

```
def classic_example_1(n, m):
    for i in range(n):
        print()
        for j in range(m):
            print(i, j, i * j)

def main():
    classic_example_1(4, 3)
```

n = 4   m = 3

```

0 0 0
0 1 0
0 2 0

1 0 0
1 1 1
1 2 2

2 0 0
2 1 2
2 2 4

3 0 0
3 1 3
3 2 6
```

i = 0 here

i = 1 here

i = 2 here

i = 3 here

Nested Loops, Type 2

```python
def classic_example_1(n, m):
    for i in range(n):
        print()
        for j in range(m):
            print(i, j, i * j)

def classic_example_2(n):
    for i in range(n):
        print()
        for j in range(i):
            print(i, j, i * j)

def main():
    classic_example_1(4, 3)
    classic_example_2(4)
```

Example 1 output
n = 4   m = 3

```
0 0 0
0 1 0
0 2 0

1 0 0
1 1 1
1 2 2

2 0 0
2 1 2
2 2 4

3 0 0
3 1 3
3 2 6
```

i = 0 here

i = 1 here

i = 2 here

i = 3 here

Example 2 output
n = 4

```


1 0 0

2 0 0
2 1 2

3 0 0
3 1 3
3 2 6
```

Nested Loops – Practice

- With your instructor, execute and examine `classic_example2a()` from ***m2_nested_loops***
  - Note how we can make there be no spaces in the output
- We will do a TODO or two from ***m2_nested_loops*** together
  - Each of the problems in this module can be solved in several different ways.
  - One approach that works for all of the pictures is to let the outer loop do the rows (one by one), and the inner loop (or loops) do the characters in a single row.
- You will do the rest of the TODO's for homework

Review: The ***4-step process*** when a function is ***called*** (aka ***invoked***)

1. Calling program pauses at the point of the call.
2. Formal parameters get ***assigned*** the values supplied by the actual arguments.
3. Body of the function is executed.
   - The function may *return* a value.
4. Control returns to the point in calling program just after where the function was called.
   - If the function returned a value, we capture it in a variable or use it directly.

```
import math

def deg_to_rads(deg):
    rad = deg * math.pi / 180
    return rad

degrees = 45
radians = deg_to_rads(degrees)
print(degrees, radians)
```

2: ***deg*** is another name for the value ***45***

3

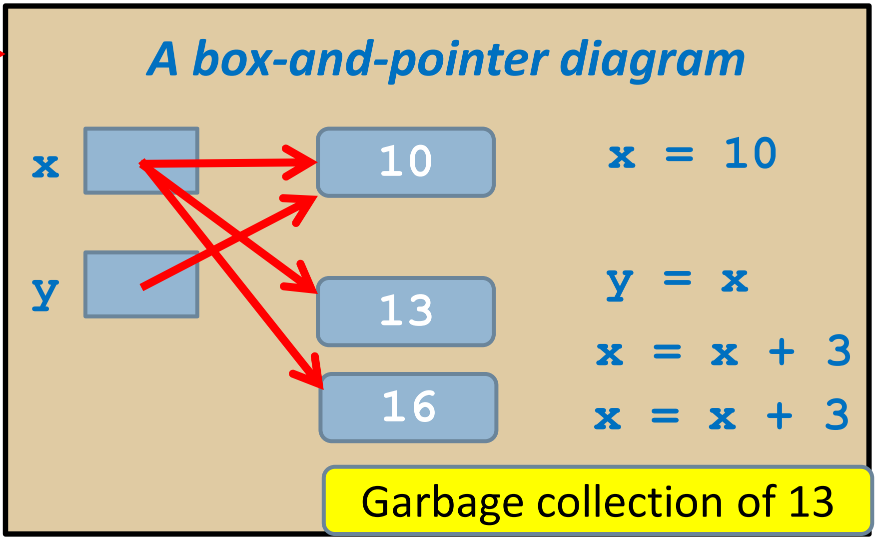
0: ***degrees*** is a name for the value ***45***

1: Pause here

4: Continue from here

Variables and parameter passing in Python

- In Python "everything is an object" and hence all variable names are ***references*** to objects
  - They act like sticky notes
  - When we pass a variable to a function, we are passing a reference to an object.
    - This is efficient (fast) – we copy only the reference, not all the data that is referenced. For example, when we pass a list, we pass a reference to the list, not all the data in the list.
    - If the object is mutable, we can mutate it in the function – this is convenient and efficient. If the object is not mutable, we are assured that it is unchanged when we return from the function – this makes it easier to write correct code. So both mutable and immutable objects have their place.



Mutators and Makers

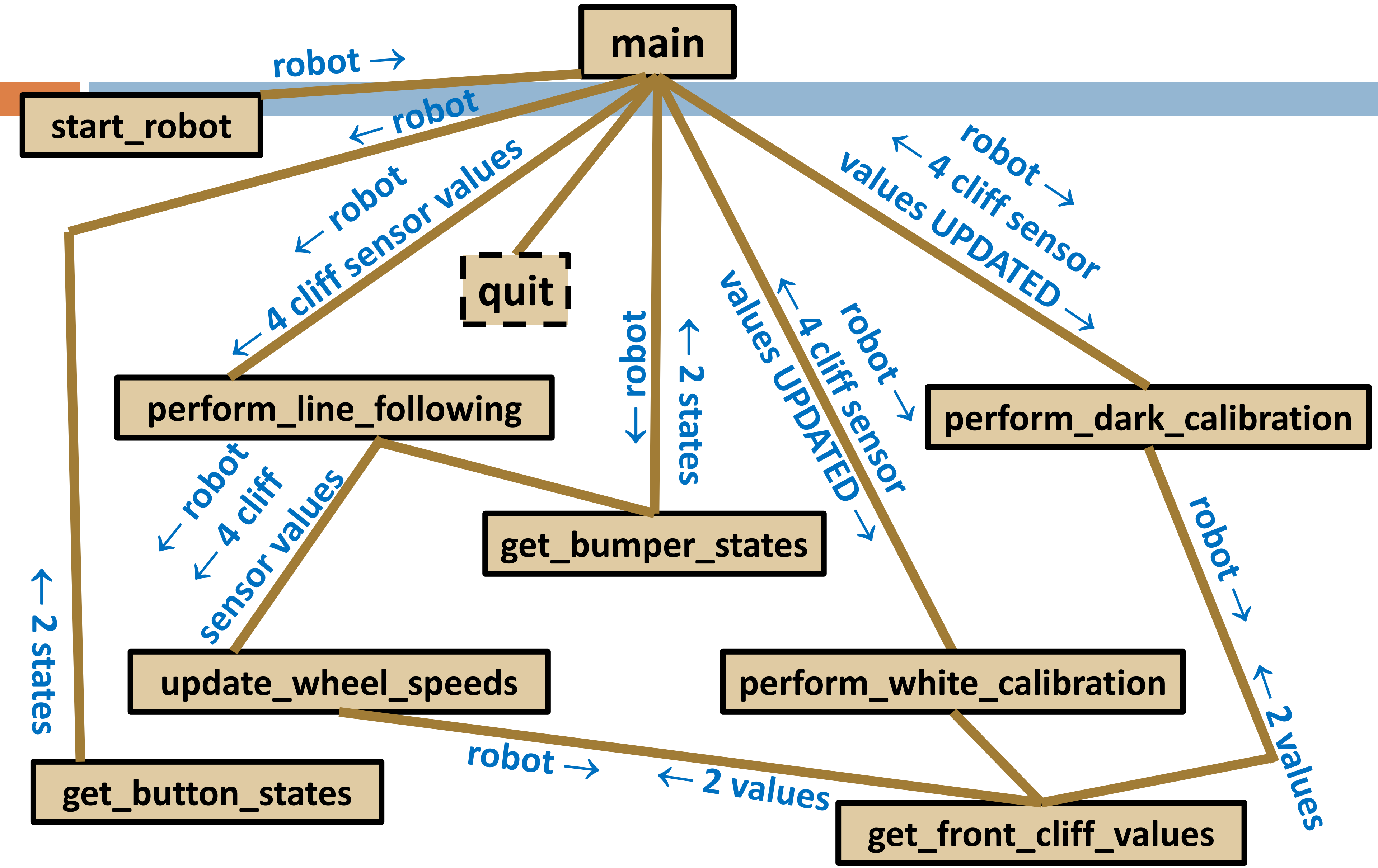
- With your instructor, execute and study ***m3_mutators***.
  - Uncomment each of the four *chunks* in ***main***, one at a time, and execute and study the relevant code.
  - Be sure you understand:
    - How lists and some other objects are ***mutated*** and what effect that has.
    - How you can ***copy*** a list or object and what effect that has.
    - The ***box-and-pointer diagram*** for `demo_a_tricky_example()` and how that makes it easy to understand why one list changes in the example and the other does not.

A *line-following* program

- Your boss wants a line-following program that works like this:
  - It starts the robot, putting it in FULL mode.
  - Then it enters a loop in which the user can press any of the following:
    - Play Button – the robot begins following the line (and stops when it bumps into anything).
    - Advance Button – the program shuts down the robot and exits.
    - Left Bumper – the program reads the two front cliff sensor values and saves them. The program expects that the user will have placed the robot on a WHITE surface just before pressing this bumper.
    - Right Bumper – the program again reads the two front cliff sensor values and saves them. But now the program expects that the user will have placed the robot on a BLACK surface just before pressing this bumper.

  When the robot does its line following, it uses the 2 pairs of cliff sensor readings for calibration.

A *structure chart* for a *line-following* program



Line-following algorithms

- There are many algorithms for following lines, depending on how many and where your sensors are, along with other factors. Let's figure out a simple 2-sensor approach.
- First, what is the effect of different wheel speeds?
  - Left faster → veer right
    Right faster → veer left
- Now look at the situations to the right, starting at the bottom. What should the robot do in each situation?

Left light sensor sees *white* (light)
Right light sensor sees *black* (dark)
Action:
- Speed up the *left* wheel
- Slow down the *right* wheel
- So the robot veers right

Both light sensors see *white*
(the robot is straddling the line)
Action:
- Set wheel speeds equal
- So the robot goes straight ahead

Left light sensor sees *black* (dark)
Right light sensor sees *white* (light)
Action:
- Speed up the *right* wheel
- Slow down the *left* wheel
- So the robot veers left

Line-following algorithms

- If you speed up to a fixed, large amount, and slow down to a fixed, small amount, and ignore the middle case, that is called ***bang-bang control***.
- You could speed up the wheels *proportional* to how far from dark the sensor readings are:
  - So completely white by a sensor would speed up its wheel to 100% and completely black would slow it to 0% of its normal speed
  - Let W, D = completely white and dark. Let L be the current reading for the left sensor. What should the left motor speed be?

Left light sensor sees *white* (light)
Right light sensor sees *black* (dark)
Action:
- Speed up the *left* wheel
- Slow down the *right* wheel
- So the robot veers right

Both light sensors see *white*
(the robot is straddling the line)
Action:
- Set wheel speeds equal
- So the robot goes straight ahead

Left light sensor sees *black* (dark)
Right light sensor sees *white* (light)
Action:
- Speed up the *right* wheel
- Slow down the *left* wheel
- So the robot veers left

Line-following algorithms

- ***Proportional control:***
- Let W, D = completely white and dark. Let L be the current reading for the left sensor. What should the left motor speed be?
- Answer: White numbers are large and black are small (near 0).
  p = (L – D) / (W – D)
  speed = p * some_constant

  But add to speed to give it a minimum speed, and clip it at a maximum speed.
- Similarly for the right wheel

Left light sensor sees *white* (light)
Right light sensor sees *black* (dark)
Action:
- Speed up the *left* wheel
- Slow down the *right* wheel
- So the robot veers right

Both light sensors see *white*
(the robot is straddling the line)
Action:
- Set wheel speeds equal
- So the robot goes straight ahead

Left light sensor sees *black* (dark)
Right light sensor sees *white* (light)
Action:
- Speed up the *right* wheel
- Slow down the *left* wheel
- So the robot veers left

Rest of Session

- ***Continue working on your m9_line_follower from Session13***
  - With help from your instructor and the assistants as needed
  - Ask questions as needed!
- **If you finish, begin HW 14**
- **Sources of help after class:**
  - **Assistants in the CSSE lab**
    - And other times as well (see link on the course home page)
  - **Email**
    - You get faster response from the above than from just your instructor

```
CSSE lab: Moench F-217
7 to 9 p.m.
Sundays thru Thursdays
```

```
csse120-staff@rose-hulman.edu
```