Homework 0

Due 4:00pm, Wednesday, January 30, 2019

Introduction

Welcome to CS4! This half-credit homework is designed to be an introduction to programming and assumes no prior background other than what has been covered in lecture thus far.

Part I: Picobot focuses on the development and testing of Picobot programs (as described in lecture). You can work on this portion of the assignment from home using the CS4 Picobot Simulator.

Part II: Hello, Python! serves as a brief introduction to Python, which is the programming language you'll be using for the first half of the semester.

Finally, this homework serves as an introduction to working with the CS department machines, which you will need to do in order to hand in the work you complete for this class.

Collaboration Policy

Before you start working on this homework, please familiarize yourself with the CS4 Collaboration Policy and remember to **fill out the online Collaboration Policy Agreement** that is linked to at the bottom of the document. You will not be graded or receive credit for any assignment, including this one, unless you have read the Collaboration Policy and digitally signed and submitted the collaboration policy form.

How to Work on Homeworks[1]

As a CS student, you have access to a number of computer labs in the CS department in which to do your work. In CS4, you will need to use the CS department machines in order to hand in your assignments.

The department machines run an operating system called Linux. Broadly speaking, an operating system is the software that determines how your computer works. Examples of other operating systems are Windows and macOS. If you're used to working with operating systems like Windows or macOS, you'll find Linux a bit different—but by the end of this homework, you'll know everything you need to know about Linux for CS4.

Step 1: Log On to a Department Machine or Use FastX

Head to one of the computing labs in the CIT, such as the Sunlab (CIT 143) or the MS Lab (CIT 167). You can log on to a CS department machine using **your Banner login and password**.

Check out this guide for setting up FastX, a program that you can use on your own computer to work remotely instead of coming into the CIT (although this isn't necessary, it may make your life easier!)

Step 2: File System

Once you've logged into your account, you should see your Desktop. On your Desktop, you should find the **File System** icon and click on it. The File System contains directories, or folders, that all the department machines need. Clicking on the File System icon will display a graphical interface that you can use to navigate the File System.

One of the important directories that lives in the File System is called **`home`**. All CS accounts live in the home directory. CS accounts are actually subdirectories themselves within the **`home`** directory, and the name of the directory associated with your account is your login! [2]

[1] The CS department setup instructions included here are based off of this helpful lab from CS17.

[2] Don't worry, no one can access your login directory except for you.

For example, this is where HTA Griffin's **`Desktop`** folder would be located:

```
/home/gk16/Desktop
```

Step 3: Using the Terminal

In order to download necessary files and hand in your homeworks for CS4, you will need to use **`Terminal`**. A terminal is a window that allows you to type commands for your computer to perform. In this section, you will learn how to navigate the file system using terminal commands.

To get started, click on the **`Terminal`** icon on your desktop or go to the **`Applications`** menu and select `Terminal`. If you don't see `Terminal`, navigate to the `Accessories` submenu, and it should be there. Open this program.

On the top left hand side of the terminal window, it should show you your **working directory**, which is where you are in the File System. You should see something similar to:

```
/gfps/main/home/<your login>
```

You're now going to create a directory for all your coursework this semester using a Terminal command. This will setup your CS account such that you can submit homework for CS4.

Run the `cs4_setup` command from your Terminal window. You can do so by typing in `cs4_setup`, and hitting the `Enter` button. This will add a `~/course/cs0040` folder to your CS department home directory. You only have to do this once at the beginning of the course.

Step 4: Installing CS4 Materials

For each homework assignment, there may be support files that you will need to complete the assignment. These can be copied to your home directory by running the `cs4_install` command from a Terminal on a department machine. For this homework, type the following and press `Enter`:

```
cs4_install hw00
```

There should now be a `hw00` folder within your `homeworks` directory. Using Terminal, you can move to the `hw00` folder with the `cd` command, which stands for change directory:

```
cd ~/course/cs0040/homeworks/hw00
```

You're now ready to begin working on this homework!

Step 5: Working from Home

While we won't go into working on assignments from home in this handout, please refer to the "Working from Home" links on the CS4 website to learn more about working with the CS department machines from outside the department.

Part I: Picobot

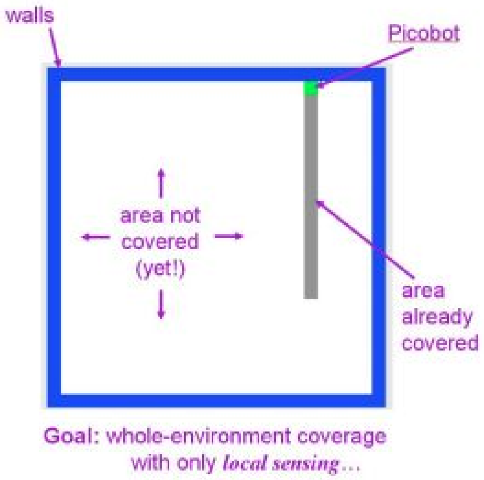
As a baby, you love to explore! You've just been put in a new nursery, and you are eager to explore the whole room. The problem is that you've only just learned how to crawl, so you are slow and get tired very quickly. Fortunately, you have Picobot, a robot that can detect its surroundings and can be programmed to your heart's content. Once you figure out how to program Picobot properly, it can explore the room for you while you sit back and relax (and maybe even take a nap).

Place your answers to this part of the homework in the `hw00pr01.txt` file located in your `hw00` directory. You can open up this file in your text editor of choice. To learn how to open a file in Atom, see the instructions in the section *Problem 0.2) Write your First Python Program*.

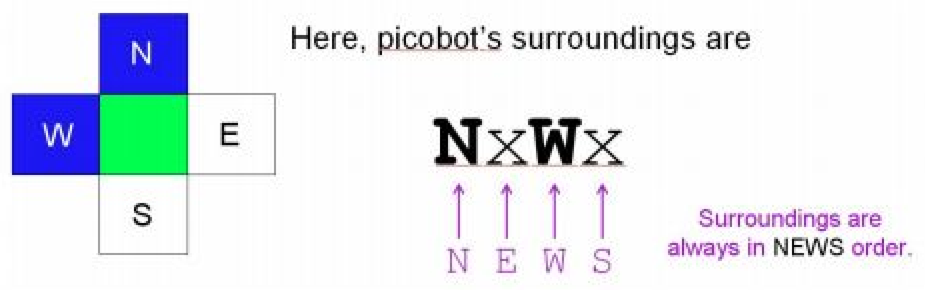
Overview

The Picobot simulator (found at http://cs.brown.edu/courses/cs004/picobot/) allows you to create and test Picobot programs.

Recall from lecture that Picobot can only sense its immediate surroundings. It starts at a random location in a room and you must write rules to tell it which direction it should go in certain circumstances.


walls
Picobot
area not covered (yet!)
area already covered
Goal: whole-environment coverage with only *local sensing*...

Picobot senses its surroundings and represents them like this:


N
W
E
S
Here, picobot's surroundings are
NxWx
N E W S
Surroundings are always in NEWS order.

An "N", "E", "W", or "S" means that Picobot is blocked in that direction, whereas an "x" means that Picobot could move in that direction.

Picobot also has an attribute called a state. It is a single value from 0 to 99 available to it as memory. In general, "state" refers to the relevant context in which computation takes place. Here, you might think of each "state" as one piece or behavior that the robot uses to achieve its overall goal. Picobot always begins in state 0. The state and the surroundings are all the information that Picobot has available to make its decisions!

So, what does a rule look like?

```
CurrentState Surroundings -> MoveDirection NewState
```

`CurrentState` = a number between 0 and 99, inclusive
`Surroundings` = a combination of N, E, W, S, x, and *, as described above
`MoveDirection` = N, E, W, S, or X (stand still)
`NewState` = a number between 0 and 99, inclusive

For example,

```
0 xxxS -> N 0
```

is a rule that says "if Picobot is in state 0 and sees the surroundings xxxS, it should move North and stay in state 0."

The asterisk (*) can be used in describing surroundings to mean "I don't care whether there is a wall or not in that position." For example, `xE**` means "there is no wall North, there is a wall to the East, and there may or may not be walls to the West or South." As an example, the rule

```
0 x*** -> N 0
```

is a rule that says "if Picobot is in state 0 and sees any surroundings without a wall to the North, it should move North and stay in state 0."

Problem 0.1a) Empty Room

Develop a set of rules that allow Picobot to completely cover an empty square room, ***regardless* of where it starts within the room**. (The empty square room is the one that comes up when you first open the Picobot simulator page.) You can use the Reset and Teleport buttons to try different starting positions. Be sure to test special starting positions (e.g., some place along each edge, and in each corner).

You may find it helpful to review the material on Picobot in Chapter 1 of the CS For All textbook and in the lecture notes.

Open the Picobot simulator, and modify/replace the existing rules with your own rules until you can get it to work. Whenever you modify the rules, be sure to click the Enter rules for Picobot button before you click Go.

As discussed in lecture, you should *avoid repeat rules* (two or more rules that would both be triggered by a given combination of state and surroundings).

You should include comments that describe your rules. For Picobot programs, these comments should be preceded by a hashtag # symbol. For example,

```
# state 0 with nothing N: go one step N
0 x*** -> N 0

# state 0 with something to the N: go W and into state 1
# ** This will crash if picobot has a wall to the W! **
0 N*** -> W 1
```

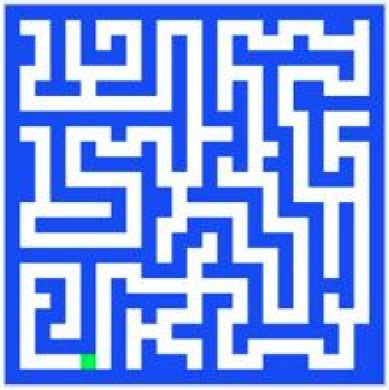
Again, make sure to copy and paste your solution into the `hw00pr01.txt` file.

Extra Credit: At the heart of computer science are questions of complexity and efficiency: questions about how simple, fast, etc. the solution to a problem can be. For an optional and difficult challenge, try creating a solution for problem 1a that uses only 6 rules!

Problem 0.1b) Maze

You've figured out how to get Picobot to explore the nursery, but now you're curious about what the world is like outside of this tiny room. Maybe you can even find where Dandy the puppy has been hiding. Let's see if we can program Picobot to help you explore the whole house!

Develop a set of rules that enable Picobot to completely navigate a connected maze with corridors one square wide, *regardless* of its starting point. You can obtain an example of such a maze by opening the Picobot simulator and clicking on the right-arrow (–>) button below the lower right-hand corner of the room, which will show you this maze:



Modify/replace the existing rules with your own rules until you can get it to work. Whenever you modify the rules, be sure to click the **Enter rules for Picobot** button before you click **Go**.

Like for the last problem, **make sure to include comments that describe your rules** (commented lines are denoted by the # character)**.**

As discussed in lecture (you can access the slides on the course website), you should:

- *Avoid repeat rules* – two or more rules that would both be triggered by a given combination of state and surroundings
- *Use a right-hand rule having four states* – one state for each direction in which the robot could be facing. Start with at least three rules for each state: one for when the wall is on your right and you are able to move forward, one for when you “lose the wall,” and one or two rules for hitting a dead end – i.e., for when you can no longer make progress in the current direction because there’s a wall in front of you.

For this part as well, you should be copying and pasting your solution into `hw00pr01.txt`.

Extra Credit: Create a solution for problem 1b that uses only 8 rules. This is *extremely* challenging, and we don’t recommend starting with only 8 rules! Rather, begin by having at least 3 rules for each state as discussed in lecture, and then see if you combine rules in some way.

Part II: Hello, Python!

Open a terminal and start Python by running the **`python3`** command. This will take you to the

Python shell and the >>> prompt indicates that the shell is waiting for you to type your commands. To exit the shell, type `exit()` or press `Ctrl-D`.

Arithmetic in Python

You can use Python as a calculator! Run the following commands and check out the results of each.

```
>>> 5 + 6
>>> 4.2 - 5.1
>>> 10 * 20
>>> 2 ** 4
>>> 5 / 3
>>> 5 // 3
>>> 20 % 6
```

Note the difference between `/` and `//` division.

Variables

Variables allow us to store values for later use. Type the following commands into your Python shell:

```
>>> x = 4
>>> y = 5
>>> z = x * y
```

`x`, `y` and `z` are all variables that store certain values. The value of `x` is 4 and the value of `y` is 5. Note the value of `z`. To print the value of a variable in a Python shell, you can just type the variable name and hit Enter.

Assignment Statements

Assignment statements store a value in a variable. In the statement `x = 4` above, 4 is stored in the variable `x`. The sign "=" is called the assignment operator.

Assignment statements follow these general steps:

1. evaluate the expression on the right hand side of the =
2. assign the resulting value to the variable on the left hand side of the =

Let’s walk through the following example.

```
>>> a = 12
>>> b = 30
>>> a = a + b
>>> b = b * a
```

What are the values of `a` and `b` after each assignment?

Note that a variable can appear on both sides of the assignment operator.

Expressions

Expressions are combinations of values, variables, and operators. The following lines are both examples of expressions.

```
>>> temp = 4
>>> res = (temp - 2) * (5 / 2)
```

Check out the value of `res`.

Strings

In addition to numbers, Python also has strings. A string is a sequence of characters and/or symbols. We identify strings by **‘ ’** or **" "** around the sequence (in this class, we will use **" "**). Here are some examples of strings:

```
>>> c = "Hello!"
>>> c 'Hello'
>>> c = "This is a string!"
>>> c 'This is a string!'
>>> c = "3"
```

> Note: the string `”3”` is not the same as the integer `3`

Print Statements

Print statements display one or more values to the Terminal. The basic syntax is:

```
print(expr)
print(expr_1, expr_2, expr_3)
```

where each `expr` is an expression.

Let’s look at these examples:

```
>>> print("The results are:", 15 + 5, 15 - 5)
The results are: 20 10
>>> cents = 89
>>> print("You have", cents, "cents")
You have 89 cents
```

Note that `print` does not print the quotes around the strings.

Program Flow in Python

A program in Python is simply a series of commands executed sequentially from top to bottom. This means that the first line of code will evaluate before the second, and so on. An example program might look like:

```
total = 0
num1 = 5
num2 = 10
total = num1 + num2
```

Problem 0.2) Write your First Python Program

There are many environments to write code in. Some popular text editors (i.e. plain-text file editors) are Atom, Gedit, and Sublime Text. In this course we recommend the use of Atom, but feel free to use whatever plain-text editor you are most comfortable with. Atom is already installed on the CS department machines, but if you want to set up Atom (and Python) on your personal machine, follow the instructions in the CS4 Installation Guide.

To open a file with Atom on a department machine, navigate to the directory containing the file using the `cd` command you used earlier and use Atom to open the file by typing:

```
atom <filename> &.
```

Note: The `&` allows Atom to run in the background, so we are still free to use our Terminal.

For this homework, open Terminal on a department machine and navigate to the `hw00` directory and use Atom to open `hw00pr02.py`. The commands to do this are shown below:

```
cd ~/course/cs0040/homeworks/hw00
atom hw00pr02.py &
```

Add the following code to your file. Make sure to replace `<your_name>` with your name.

```
name = "<your_name>"
print("Hello", name)
```

Now we will save this file with `Ctrl+S`.

Finally, let’s go back to the terminal and run our first Python program! Run the following line:

```
python3 hw00pr02.py
```

Yay! You just wrote your first Python program!

Handing In

At this point, your CS account has been set up, and your solutions should be complete. From a Terminal on a CS department machine, navigate to your `hw00` directory and check its contents by running:

```
cd ~/course/cs0040/homeworks/hw00
ls
```

Are both of your homework solution files there? You should see `hw00pr01.txt` and `hw00pr02.py` listed. Make sure you have included your answers to Part I and Part II in those files, respectively.

Project hand-in. Be sure to turn in all requested files. When you’re ready to submit the files for this homework, run:

```
cs4_handin hw00
```

from your `~/course/cs004/homeworks/hw00` directory. The entire contents of `~/course/cs004/homeworks/hw00` will be handed in.

Check for a confirmation email to ensure that your assignment was correctly submitted. You can resubmit this assignment at any time using the `cs4_handin` command, but be careful, as only your most recent submission will be graded. If you resubmit, make sure that *all* requested files are resubmitted even if you only make changes to one file. You are scored based on the date of your last handin, so be careful about resubmitting after an assignment deadline has passed.

Reminder: Do not put any identifying information (name, login, Banner ID) in your homework submissions. All handins are graded anonymously by the course staff.

Please let us know if you find any mistakes, inconsistencies, or confusing language in this document or have any concerns about this and any other CS4 document by posting on Piazza or filling out our anonymous feedback form.