1 Administrative Topics

- Stephanie takes her quiz.
- Did everyone make it to lab this week?
- Plan for today: go over syllabus, talk about algorithms, go through example code for project

2 Syllabus and Technical Aspects of Course

- You can talk to each other about code and even write things down on paper for each other. But never electronically copy someone else’s code. It is really tough to have to fail someone because they copied from someone else. But it isn’t fair to others who do their own work, and it isn’t fair to you because you aren’t learning. So we cannot give credit to you for work that you didn’t do, and if you copy a project, you are putting yourself in a position to fail the entire course.

- To turn in code, you will need to mount the Courses/CS151 directory. We will talk about this on Monday, when it is time for you to turn in code.
3 Algorithms and Programs

We want to use computers. Computer scientists make them do many, varied things. Recall our list from Wednesday. What is common among these items? One unifying theme is “precise thinking”. But precise thinking about what? About how to do things - how to network two computers, how to simulate biological systems. They all study process: how we do things, how we specify what we do, how we specify what the stuff is that we are processing.

This brings us to the term “algorithm”. An algorithm is a sequence of instructions to perform a specific task or to solve a specific problem. An algorithm should be unambiguous to a human, but it need not be executable by a computer.

Program – an implementation of an algorithm in a computer programming language, making it unambiguous to a computer

In your first project, you will be developing algorithms to draw specific shapes. You will then implement of your shape-drawing algorithms by writing Python code. Your algorithm is a precise set of instructions a human can understand, and your code will be a precise set of instructions that Python can understand.

4 Python

For the rest of the class, we will spend some time talking about how to write Python code for the projects.

There are two ways to run Python code:

1. Python can be run in interpreter mode, by simply typing python in the Terminal window.

2. You have already seen that. It can also be used to run specific Python programs, but typing python and then the name of the program file after it.

The interpreter mode is useful for testing out your understand of new functions or little bits of code. I will often use it in lecture. But to run longer
programs, it is best to write the code in an file (as you did with smart.py in
lab).

Our first few projects will use the turtle to draw pictures. To draw with
the turtle, you call, or execute, the turtle functions. A function is a chunk
(sorry, chunk is not a technical term) of code with a particular name and a
particular purpose. In lab, you executed several turtle functions, including
left, forward, and reset.

Python has a lot of packages (of which the turtle is one). In order to gain
access to the functions in a particular package, or module, you need to import
that module. So you typed from turtle import * in the interpreter.

When you are writing programs, it is best to write them in files, rather than
type them the interpreter. This way, you can write your once, then edit it
as necessary, if you don’t like your first attempt (which you NEVER will :)
). This is where Text Wrangler enters the picture. You need to write your
code using Text Wrangler, and execute it using Python. You do this python
execution in the Terminal. To execute a python program, you need to run
python from the directory (folder) that contains the program file. Open your
Terminal and navigate to the appropriate directory (using cd). Then create
and edit your file, saving it in the same directory.

Now we are ready to write a program that uses the turtle. (Note that the
following examples are not the exact examples used in lecture, but they
illustrate the same points. I have posted the code from today’s lecture sep-
ARately.)

```python
# Stephanie Taylor
# first_turtle.py
# This is a simple program that demos the turtle.
from turtle import *

# Draw a wall
forward( 200 )
left( 200 )
forward( 200 )

# Wait for input
raw_input( "Press enter " )
```
We need to import the turtle to had access to the function. Then we execute the drawing functions. Then we tell Python to wait for keyboard input, so that the program doesn’t end and take down our drawing. The words after the pound sign are comments, which help human readers understand the purpose of the code.

For the next step in your project, you will be writing your own functions to draw shapes. Let’s make an L shape.

You can define your own function like this:

```python
# Draw an L with a 200–pixel long edge and # 100–pixel short edge.
# Note: Draws the long edge first.
def drawL():
    forward( 200 )
    left( 90 )
    forward( 100 )
```

But there is something very limiting about my drawL function. Any time I call it, it will draw an L of the same size!!! I would like to draw an L that is resizable. To do that, I need to allow my function to take input. I will input the the distance of the short edge. I can define a parameter when I define my function, by adding including its name in the parentheses. I then refer to the parameter by name in the function’s code.

```python
# Draw an L with the given short edge length.
# The long edge will be twice as long.
# Note: Draws the long edge first.
def drawL( short_edge_length ):
    forward( short_edge_length )
    forward( short_edge_length )
    left( 90 )
    forward( short_edge_length )
```

Here is an example of a file that contains both function definitions and the code to execute those functions:

```python
# Stephanie Taylor
# l_drawing_example.py
# Draw a simple L using the turtle
```
from turtle import *

# Draw an L with the given short edge length.
# The long edge will be twice as long.
# Note: Draws the long edge first.
def drawL(short_edge_length):
    forward(short_edge_length)
    forward(short_edge_length)
    left(90)
    forward(short_edge_length)

# main code
# Draw an L
right(90)
drawL(100)

# Wait for keyboard input.
raw_input('howdy!')