Agenda

1. Syllabus and organization - 10 min
2. IDLE and creating files - 1 min
3. Print and functions - 10 min
4. Order of operations, basic math, and expressions - 5 min
5. Comments - 2 min
6. Variables - 5 min
7. Objects and types - 10 min
8. Comments and variable names - 2 min
9. Overview of coding goals - 2 min

Update: Checkoff 3 is now due this **Friday** instead of Thursday. The syllabus has been updated to reflect that.

Due dates this week (all by 5 p.m.):
Monday - Checkoff 1
Wednesday - Checkoff 2
Friday - Interview 1 and Checkoff 3

Syllabus and organization

Where is lecture? On Mondays, Tuesdays, and Thursdays, we will be in 10-250. On Wednesdays and Fridays, we will be in 34-101. **Most of the official MIT listings are incorrect - please rely on the course website!**

Is there a Stellar site? Nope! Our course website, which will be where you find all materials, including lecture code and homework, can be found at http://web.mit.edu/6.149/www/materials.html.

Why is the <assignment/lecture notes/readings> you said you posted missing from the website -OR- why does the website look different for me than it does for my friends? Sometimes the course website will freeze you on an old view of the page. Clearing your cookies should fix this problem.

How will office hours work? Office hours are 10-11 a.m. and noon-5 p.m. in 32-044, which is in the basement of Stata - go down the stairs near the eating area. When you get to office hours, there will be two queues on the board. One is for getting help, and one is for getting a checkoff. Add your name to the end of the appropriate queue, find a seat, and please be patient. Please leave at least 30 minutes for every trip to office hours - we have over 300 students in class, and we want to give everyone time to get their questions addressed thoroughly! Peak hours (especially right after class) may require 60 minutes. **Write down the**

name of the LA who gave you each checkoff to assist in resolving any errors in data entry.

Do I need to bring my computer to lecture? Many students find it helpful to follow along with access to the Python shell so they can try code out for themselves. However, if you'd find your computer distracting, we won't require you to do work on your computer in lecture.

Do I need to bring my computer to office hours? **Yes.** To get checkoffs, and sometimes to complete your interviews, you will need to run code on your computer.

What are the differences between checkoffs and interviews? Interviews are more of a test; we expect you to be prepared for both interviews and checkoffs, but checkoffs are more appropriate for asking (short) questions. All interviews and checkoffs must be done in person - you must notify the instructors if any emergency prevents you from getting to office hours so we can arrange an early/late checkoff.

How do I use Piazza? Piazza is a question-and-answer forum that is used by many MIT courses. Private posts (which can be seen by only the instructors and lab assistants) are appropriate for questions of a personal nature (family/medical emergencies that get in the way of coursework). Otherwise, please make your post public! Everyone in this class is learning how to code, so please ask and answer questions with your name instead of anonymously. It's important that everyone feel welcome to ask questions, even if you're behind or confused about a topic. Avoid posting large blocks of code to ensure everyone does his or her own work.

Other notes:

1. Ask questions in class - if you're wondering, probably at least 20 other people are confused as well. If we're tight on time, we might ask to talk to you after class, and then we'll put a note on Piazza with a clarification, but that doesn't mean your question is bad!
2. Don't get discouraged if you feel like you're not catching onto programming quickly. Your classmates might have had some exposure to programming, or we might not have gotten to the topic that makes it "click" for you.

IDLE and creating files

When you first open IDLE, you will usually see the Shell. You can type input directly into the shell after the ">>>" symbol, hit enter, and get immediate feedback. The Shell is useful for teaching, but since your ability to save work is limited, we won't be writing any substantial programs here.

To open a new file, click File > New Window. You can then save your file (remember to put .py at the end to get the pretty colors!). Hit F5 (or Run > Run Module) to execute your code.

Python examines your code line-by-line, so think of each line as an independent unit unless told otherwise.

Print and functions

Cool! Let's do some math now. The most basic programs we can write involve simple arithmetic computations. Let's write the following program and call it math.py:

```
2+2
```

But when we hit F5 to run the program, what happens? *Nothing.* Or, at least nothing we can see. The computer actually did exactly what we told it to do - it added 2+2. At no point in time did we ask the computer to do anything else. While computers are pretty cool, they can't read minds, so we didn't get any visual confirmation that 2+2 was added.

To fix that, let's use something called a *function*. Think of a function as a black box that carries out some operations in a predictable manner. Some of these black box functions can take inputs, which they'll often manipulate.

A real-life analog for a function is a coin-to-cash machine. You give the machine some input - like a pound of assorted coins - and *something* happens inside it that results in you getting your money back in bill form, minus a small fee.

In this case, we wanted to use a very special function called *print*.[1] Print is a black box that will display whatever inputs you give. When we use a function, we say that we are *calling* it. Let's change our program to the following:

```
print 2+2
```

Now, when we run our code and switch to the Shell, we see a 4 "printed" to the screen. That's because the input to the print function is whatever comes after it on the same line.

Print is possibly the most useful function in coding. It allows you, the programmer, to display information about your code. You may want to show progress messages to a user. You may also be completely stuck with code that doesn't work and want to figure out why. By inserting print statements throughout your file, you can start to figure out where your code is behaving differently than expected.

---

[1] Print works a little differently from the other functions we'll use, so make sure you don't generalize too much from the syntax (i.e., what you type to use the print function).

> Common mistake: Students frequently forget to include print statements and wonder why their code isn't "working." Make sure you've asked your computer to do everything you want!

Order of operations, basic math, and expressions

When we do math in our code, the order of operations is the same as in any other class you've taken. If our program was:

```
print 2+2*8
```

the output would be 18, not 24 because multiplication happens before addition. Python knows the order of operations. You can use parentheses as well:

```
print (2+2)*8
```

would yield 32.

You can add (+), subtract (-), multiply (*), divide(/), raise to an exponent(**), or take the modulus (%). The modulus is the remainder when the first number is divided by the second. 4%2 = 0, 3%2 = 1, and 9%5 = 4.

> Common mistake: Raising a number to an exponent is **not** done with a caret (^) symbol. The caret actually has another valid meaning in Python, so you won't get a *syntax* or *runtime* error (something that prevents your code from running to completion). Instead, you'll experience what we call a semantic error, which means that your code will run, but not do what you expect.

Note that when we asked our program to print 2+2, it didn't print the expression "2+2" - it calculated the value of 2+2 and printed 4. In the Python order of operations, we also need to consider *expressions*. Loosely speaking, expressions are anything that have values. (The value of 2+2 is 4.) Before sending 2+2 as the input to our black-box function print, Python is smart enough to evaluate - or calculate the value of - that expression.

```
print 2+2
```

gets simplified down to

```
print 4
```

in the “brain” of Python. The input that print actually gets sent is 4 - not 2+2! Always remember to think about how Python evaluates expressions in your code.

Variables

Sometimes we want to store values for later use. We can do so by using *variables.* A variable is a name that the computer associates with a specific location in its memory; this location in memory will hold the information we want to save. To make a new variable in Python, write a (descriptive) variable name, followed by an equal sign, followed by what you want the variable to refer to. This process is called assigning the variable.

You can assign variables to expressions, and you can send variables as input to the print function.

```
my_first_sum = 2+2
print my_first_sum
```

If you try to use a variable - for instance, as input to the print function - before assigning it to a value, Python will throw an error - usually a mass of red text that halts your program. Python tries to *evaluate the variable* by looking up the memory address the name is associated with. But since the variable was never given a value (and therefore never associated with a location in memory), the computer doesn’t know what to send to the print function.

```
my_second_sum = 3+3
print my_third_sum
```

Note that in our first example of printing variables, the input that is finally passed to the print function is 4; in the second example, no input is successfully evaluated to sent as input, so the function is officially called.

You can assign variables to each other, and you can re-assign a variable that has already been associated with a value. In fact, most things that you can do in algebra with a variable can be done in Python.

```
my_first_sum = 2+2
my_second_sum = my_first_sum
print my_first_sum
print my_second_sum
my_second_sum = 5
print my_first_sum
print my_second_sum
```

Note that in the previous code block, changing the value of my_second_sum *doesn't* affect the value of my_first_sum. That won't always be the case, but when we're working with numbers, it's a safe assumption.

Objects and types

Everything in Python is an *object*. Every object has a *type* (we also say it's a member of a certain *class*). Objects and types are analogous to their real-world counterparts. Think of robots - robots are objects of type robot. Though Python doesn't show up on your computer with any concept of a robot built-in, we can write code that will fix that for us - and we will later in the course!

When we talk about numbers, we're actually being imprecise. The two types of what we call numbers in Python are integers and floats. This whole class, we've been working with integers; but if we wanted to represent decimal numbers, we'd use the float type by simply typing in numbers with decimal points.

To check the type of a value or variable, use the type function.

```
type(3.0)
```

will print out float.

In some cases, we can change the type of an object. Floats and integers are great examples of this property. Try the following block of code:

```
a = 3
print type(a)
b = float(a)
print b
print type(b)
```

When coding, we frequently need to represent text. In Python, text can be represented by strings. Strings are simply words enclosed in single or double quotation marks. The quotation marks differentiate between a string and a variable name.

```
name = "Mary"
print type(name)
```

Just like with numbers, we can add strings together (we call this *concatenating* two strings).

```
first_name = "Mary"
```

```
last_name = “Jones”
full_name = first_name + “ “ + last_name
```

Classes and types are important because they allow us to create ways to reuse pieces of code. Functions can also be defined to only work with certain types. For instance, a common operation on a string is making all the letters it contains lowercase. Having the function the performs that operation also work with integers doesn’t make much sense - what exactly is a lowercase integer?

Comments and variable names

If you want to write text in your code that isn’t run or examined by the interpreter, you can use a *comment*. To write a comment in Python, put a number/pound sign (#) at the beginning of the line you don’t want to run. Comments are helpful for making notes in your code about what it does so that other people (and you a year from now!) can easily read your code.

```
print “Now you see me!”
# print “Now you don’t!”
```

It’s also very important to use good variable names in your code. Examples of **good** variable names are:

force, mass, and accel
student_1, student_2, and student_3
x_val and y_val
first_name, last_name, date_of_birth,

Examples of **bad** variable names are:

a, b, and c
s1, s2, and s3
lol, rofl, roflcopter

Overview of coding goals

By the end of Week 1, we’ll learn about most of the classes that Python ships with. We’ll also learn how to execute code if certain conditions are met, and write code that performs an action (or multiple similar actions) multiple times.

By the end of Week 2, we’ll write our own functions. By the end of Week 3, we’ll write our own classes.