CSE231 - Lab 01

Syllabus Day, Types, Variables, Assignment, Mathematical Operators, Comments, Error Messages and Introduction to the Python SL

oh god

Welcome to CSE231
Introduction to Programming I!

Preface

This is going to be longer than usual. We have to go through the syllabus and a lot of introductory Python. My presentations will not normally be this long.

Before we begin...

github.com/braedynl/SS20-CSE231-Lettinga

Who am I?

My name is Braedyn Lettinga, I just go by “Braedyn” -- I’m not a professor.

I’m a Computer Science major, my focuses are software engineering and data science. You can feel free to ask me questions about the major and future classes you’ll be taking.

You might have seen me around campus, you might even have or have had a class with me. Come say hi and we can grab a coffee!

Contact details coming later.

What is this class?

This is (probably) your first introduction to programming. In this class we'll be teaching you guys Python, a very intuitive and powerful programming language.

Through Python, we'll show you guys all of the fundamental concepts that go into almost all other programming languages, so you can begin to become, not a Python programmer, but a ***general*** programmer.

Once you go through this class, you'll (hopefully) find that learning more programming languages is extremely easy.

And of course, you'll also be able to apply your knowledge gained here to your future programming internships and jobs.

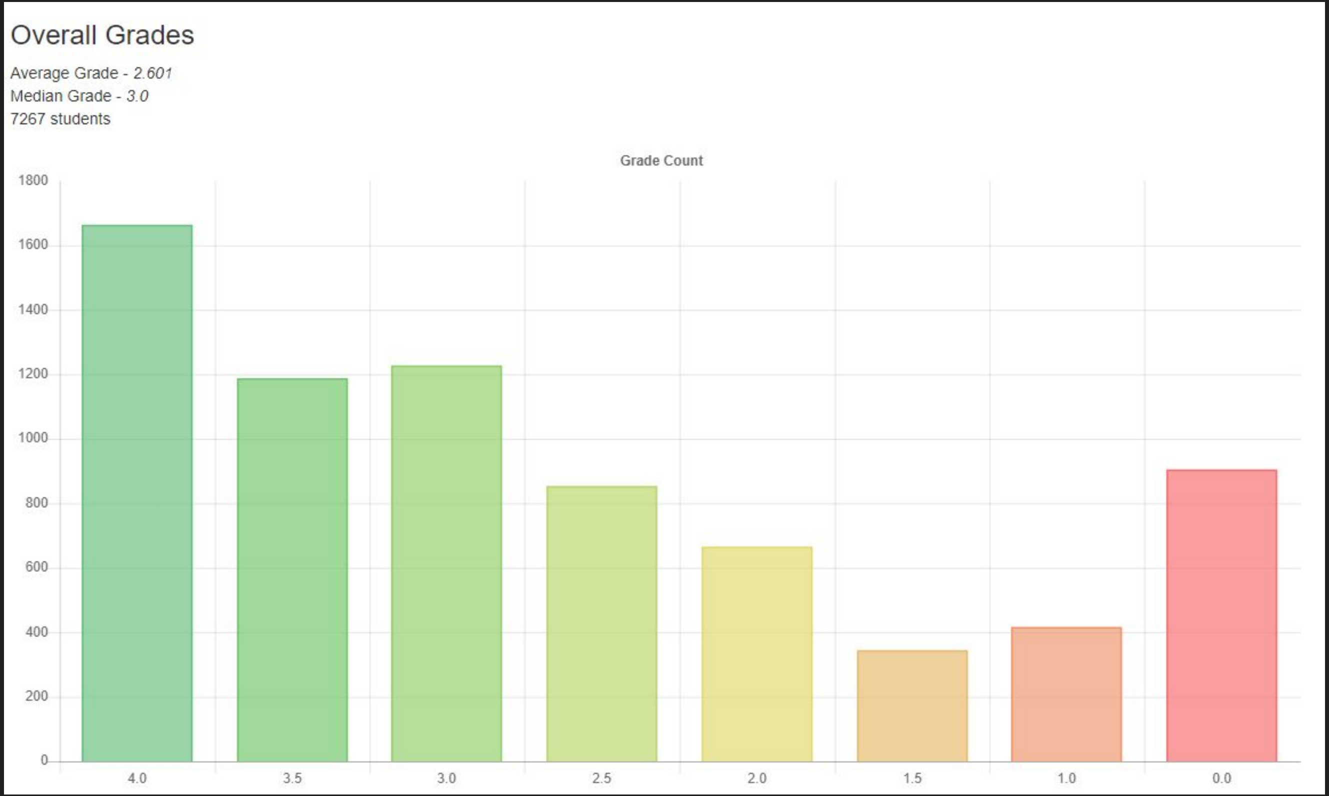
Why is this class so dreaded?

You’ve probably heard legends of this class, either online or through people that have taken it already.

This class has ***a lot*** of homework. It sucks, yes. But there’s a good reason for why there’s so much, although you may disagree.

If we’re going to teach you how to program from scratch, then you’re going to need the practice. The hardest part about learning a programming language by yourself is getting that practice, people tend to fall in the rut that is: “Well I know how to program now, but what do I do?”. This class requires you to get so much practice that, by the end, you’ll hopefully feel extremely comfortable with Python.

A lot of people 4.0 this class, you can be one of them.


Overall Grades

Average Grade - *2.601*
Median Grade - *3.0*
7267 students

How the class is graded

- Exams - 45%
  - Two midterms, one final
    - Midterm 1 - 10%
    - Midterm 2 - 15%
    - Final - 20%
  - You are allowed a single sheet of paper notes, front and back. No electronic devices.
- Projects - 45%
  - Due on Mondays, on Mimir, there is a small intro video for every one
  - Most points come from passing the automated test grading, with some points given by me.
  - Submissions reduced by 50% for being one day late, no credit after one day
    - Condition can be changed under an emergency situation
  - Infinite amount of submissions before deadline -- *best* submission is graded
- Exercises - 10%
  - Due on Saturdays, on Mimir
  - Problems are credit/no credit, no points are given by me.
  - Infinite amount of submissions before deadline -- *latest* submission is graded

You probably noticed...

The labs DO NOT contribute to your overall grade. But…

If you miss more than two labs, your ***overall*** course grade GPA is reduced by 0.5 for each lab missed beyond two.

If you had a 4.0 prior, and missed 4 labs, you will drop to a 3.0.

- Two free misses
- Two real misses
    - 4.0 - 2(0.5) = 3.0
    - :(

Why the labs, then? What are the pre-labs for?

The labs are simply just another opportunity for you to get more practice with the concepts you learned in the lecture videos, without having to worry about being graded for perfection.

You must complete the pre-lab before coming here, or else I cannot give you credit on the in-class lab. The pre-labs are also good review for the exams.

I keep track of whether or not you attended and gave the lab a shot (even if the shot was shitty). I’m also here to help with the class in general. If you need help understanding the lectures, or you’ve encountered a problem in your exercises and/or project, feel free to ask.

Some general tips for this class

Start projects ***early***. I cannot stress this enough.

Sometimes, you might be faced with a problem that you just have no idea how to implement an algorithm for. Coming back to a problem the next day, or simply taking a break and doing something else for a bit can really help.

Having problems with your computer or with your IDE sucks, and could happen -- leave yourself time to get it resolved before the project deadline comes.

Always, always, always test your code as you're writing it. Run your code frequently, and make sure it's doing what you're expecting it to do.

Some general tips for this class (cont.)

Piazza and the help room are here for you as well. I will be at help room at variable times of the week. If you want help from me specifically, shoot me an email and ask when I’ll be going for the week, or we can meet somewhere else. Otherwise, there will always be other TAs at help room.

Piazza is a great tool as well, although it may sometimes be difficult to get the answer you’re looking for. You can ask questions about your issues anonymously in the comfort of your own dorm.

Your IDE has a debugger, which is an incredible tool to help diagnose code issues. See if your debugger can help you before asking instructors or going on Piazza. (I’m not sure when Enbody has this planned, I might just show it to you guys early if I think he’s taking too long)

Some general tips for this class (cont.)

When Googling coding issues, you'll likely find StackOverflow.com is always the first website to appear. This is a great resource to use if you're trying to write an algorithm in a specific way, or learn the syntax of a certain line/block.

Please attempt to get acquainted with others in the class and see if they can help you as well. You're allowed to collaborate on everything except the projects. You can discuss the problems in the projects with other people, but you ***cannot*** share your code or take the code of others that are ***in this course.***

The reason "***in this course.***" is bolded and italicized is because you ***are*** allowed to take code you find online, ***as long as it was not specifically made to solve the issues in the projects for this course.*** It is recommended that you cite where you obtained your code from, a simple comment with a link is fine. This leads nicely into the next point...

Academic Dishonesty Reports (ADR)

Like I said, you ***cannot***, under any circumstance, share code for the projects. Even seeing another person's implementation of a solution may lead you to doing something similar. Please don't risk trying to copy another person's work, it's not worth it.

If we find that your code has been plagiarized, you will receive an ADR. Course punishment may vary (I believe you typically get an instant course failure), but a report is always filed with the University.

You can appeal if you believe you have wrongly been reported.

https://ombud.msu.edu/academic-integrity/academic_dishonesty_report.html

Course Links & Contact Details

There are ***a lot*** of links to keep track of for this course. I’m going to try and make this as easy as possible for you guys.

This is hopefully the only link you’ll need. You can navigate everywhere else from here.

github.com/braedynl/SS20-CSE231-Lettinga

We’ll do a quick site tour, go over the content (finally), and then jump into the lab. Bookmark this page if you prefer navigating from here over juggling 20 different links.

My specially created slides and contact details will all be hosted here.

Now that all of that’s out of the way...

Let’s talk about Python.

In the videos we talked about types, assignment and the mathematical operators.

We’ll quickly cover these again, but I’m going to cover some extra stuff that I think you guys should know that’ll be useful for exams and later content.

Fundamental Types

In the lectures, we discussed the fundamental types:

- int, an integer number value
- float, a decimal number value
- string, a series of characters
- bool, true or false

These are all part of the base language, and are typically what make up future types that we'll be discussing at a later date.

Demo time, L1-1

Type-Casting/Conversion

A lot of the time, you’re going to have to deal with the conversion of variable types. Certain types will only work with certain functions or have some sort of functionality that another doesn’t. These are cases where you would want to convert between types.

You can use the int(), float(), str(), and bool() function to cast your objects into your desired type.

Demo time, L1-2

Variables and the Assignment Operator

You guys have seen variables and assignment in action a lot already, now.

Variable names hold the value and type they are assigned through the assignment operator, ‘=’.

You initialize variables when you plan on reusing that value somewhere else in your code. Variables in Python can hold **a lot** of different things, as we’ll see later on in the course.

Please for all that is holy, make your variable names clear.

This is a good habit to get into for readability, not only for yourself, but for other people as well. (Like me and your future managers -- don’t disappoint your managers)

Good/Bad Variable Names, Naming Conventions

Good:

- avg_interest_rate = 11
- current_credit_score = 660

Bad:

- a = 24
- cool_str = “This is my cool string.”

How good your variable name is, however, dependent on the context. You’ll have to make the call on whether or not your variable’s name implies it’s usage.

Also note how variables are lowercase and use underscores. This is not a requirement, but a common practice.

Illegal Variable Names

You’ll likely come across this writing code during your time in this class, but I’ll quickly cover it here.

- 123_var = 123   # Leading number values
  - var_123 = 123   # Non-leading = legal
- +var = “var”   # Operators/syntactic elements
- print = 3.14   # Previously declared namespace of a function, type or class

Typically raises a SyntaxError

(Mathematical) Operators

There are a lot more operators that we’ll get to at a later date, but for right now we’re going to look at the ones that we can do math with.

- +, addition
- -, subtraction
- *, multiplication
- /, division
- %, modulus
- //, floor/integer division
- **, power

Demo time, L1-3

Compound Assignment Operators

Alongside all of the mathematical operations, there are variations of each that also double as an assignment operation. You do this by appending an ‘=’ symbol to any of the operations we just talked about.

Examples:

- +=, adds value *and then* assigns to variable
- **=, raises value to power *and then* assigns to variable

Demo time, L1-3

The Python Standard Library (SL)

Python, by default, comes with **a lot** of different functions and utility. We’ll be teaching a lot of them to get you started throughout this course, but we won’t have time to cover everything in it.

You’ve already seen print(), which is integrated with the language and doesn’t need to be imported from the rest of the standard library.

You might have seen something like math.sqrt() being used before. This comes from the ‘math’ module in the Python SL. The Python SL is split into separate modules to allow more room for user-created namespaces among other things.

The hell is a module?

You’ve likely seen the line:

```
import math
```

Or something to that effect in Python code at the top of a file. The ‘import’ command brings in utilities from a library or file for you to use in your program. We’ll be teaching you guys a lot more about this later.

The ‘math’ module has a ton of different functions that you can then use in your code. To invoke a function from the module, we have to call through ‘math’, and use dot notation to denote what function we want to use from it. To access it’s square root function for example, we would say:

```
math.sqrt()
```

print()

The print function takes anything and displays it in the console for the programmer to see. Like in mathematics, the variables input to the function are put inside the parentheses.

You can input multiple things to the function by separating each object with a comma. These inputs are called **parameters**, and the print() function can take an infinite amount of them.

This is getting a bit too in-depth for now though, we’ll be covering functions at a later date.

round()

If you want to round your floating-point numbers, you can do that using the round() function. It takes two parameters:

round(number[, ndigits])

- number, the value you want to round
- ndigits, the nth decimal place you want to round to (optional)

The formatting for the parameters of the function shown above is commonplace in online documentation, and so it’s important to know how to read this. The square brackets denote that the following parameters are *optional*. You can invoke the function solely with a number value, (which, with how the function is programmed, will round to the ones place), or with a number value **and** an ndigits value.

input()

This is a function you'll be getting quite familiar with in this class. input() pauses your program and waits for a user to input text to the console. It takes one optional parameter:

input([prompt])

- prompt, a string that displays an inline message to the console for the user to read

Anything read from input() is returned as a string. You'll typically have to type-cast if you want to perform int/float operations.

Demo time, L1-4

Comments

One of the most important things you can do as a developer is document your code. Documenting your code **is a requirement** for the projects because it makes it easier for you to come back to your code, and because your manager at a dev job is going to want you to anyways. (It's just good practice)

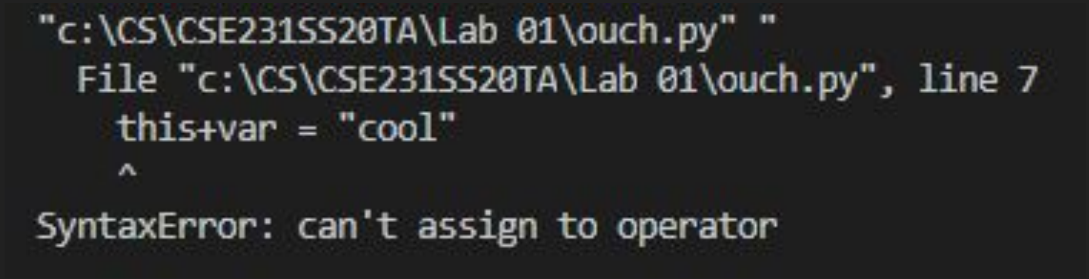
There are two types of comments, single-line and multi-line.

- Single-line is denoted by a '#' character. You can then type any message you want in the same line after the '#'.
- Multi-line is denoted by three single quotes. Any text continuing afterwards in the same line and following lines is considered a comment until another set of three single quotes.

You **do not** have to comment on every single line of code. Please don't. Instead, give overview comments on blocks of code, or single-line comments on complex-looking lines.

Reading Error Messages

A lot of beginner students panic when they get an error message. This is probably the worst thing you could do. Calmly read what it says, and go to the area where it says there's a problem.



```
"c:\CS\CSE231SS20TA\Lab 01\ouch.py" "
  File "c:\CS\CSE231SS20TA\Lab 01\ouch.py", line 7
    this+var = "cool"
    ^
SyntaxError: can't assign to operator
```

Googling error messages usually won't find *your* problem specifically, since error messages cover such a broad range of problems that could occur. It's best if you walk through your code thinking about what's being input to the line that could be resulting in the issue.

In this message, we simply have incorrect syntax on line 7.

Aaaand that’s all I got. It’s finally lab time.

Today’s lab should be pretty simple. Go to either my repository or the course website:

github.com/braedynl/SS20-CSE231-Lettinga / web.cse.msu.edu/~cse231/

and navigate to the lab for this week. For the labs, you’ll follow and complete the instructions given on lab0X.pdf. Sometimes you’ll download starter-code with it, lab0X.py, that will be part of those instructions.

For the first couple of weeks, you are to work with another person in the room on one computer.

On Mimir, you can find a submission domain to test your program. Show me that your finished program can complete all of the test cases and you’re free to go whenever.

Call me over if you have questions or are having trouble!