CSE231 - Lab 01

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

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.

Who am I?

I’m Braedyn… you can call me Braedyn.

I’m a computer science major, I’m currently in my third year here at MSU. My focuses are in software engineering and machine learning.

I’ve been TA’ing CSE231 since I was a wee sophomore, I’m also a research assistant for the media and information department.

I’ve been doing the COVID-19 teaching thing for a little bit now. I think it’s unfortunate that I won’t be able to see you guys in-person, but safety is obviously more important here.

What is this class?

This is (possibly) 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 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 be able to apply your knowledge gained here to your future programming internships and jobs (if you’re a compsci major).

How the class is graded

- Exams - 45%
  - Two midterms, one final
    - Midterm 1 - 10%
    - Midterm 2 - 15%
    - Midterm 3 (Final) - 20%
  - You are allowed a single sheet of paper notes, front and back. No electronic devices.
- Projects - 45%
  - Usually due on Mondays, submissions 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 are reduced by 50% for being one day late, no credit after one full day.
    - Condition can be changed under an emergency situation.
  - Infinite amount of submissions before deadline -- *best* submission is graded.
- Exercises - 10%
  - Usually due on Saturdays, submissions on Mimir.
  - Problem credit is entirely based on Mimir’s test-cases. Grading is automated, I do not touch them.
  - 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
    - :(

Contact me before lab (preferably early) if you are *guaranteed* to miss one and you don’t want it to count against you, your miss must be for a good reason.

Course Links

I don’t like Enbody’s website, (and many others don’t like it either), so I have my own that you can use if you so choose:

https://github.com/braedynl/CSE231-GITHUB

It’s basically a layer on top of Enbody’s but a whole lot more condensed.

I make my own presentations and examples inspired by the ones Enbody gives the TAs, and so my stuff will be on this website if you ever need it.

I have a bot that’ll send out my lab materials every week if you don’t like my site.

Academic Dishonesty Reports (ADR)

You ***cannot***, under any circumstance, share code for the projects. Even seeing another person's implementation of a solution may lead you to do something similar. Please don't risk trying to copy another person's work, it's not worth it. We keep databases of Chegg code, don't you dare use Chegg lol.

If we find that your code has been plagiarized, you will receive an ADR. Course punishment may vary (I believe you typically get a 0), but a report is always filed with the University.

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

https://ombud.msu.edu/

Fundamental Types

In the lectures, we discussed the fundamental types:

- `int`, an integer number value
- `float`, a decimal number value
- `str`, 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.

L1-1

Type-Casting

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

You can use the `int()`, `float()`, `str()`, and `bool()` functions to cast your objects into your desired type. Not every type can explicitly be converted into another, however. It depends on the value being held.

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, (the equality symbol, ‘=’).

You initialize variables when you plan on *reusing* that value somewhere else in your code. Variables can hold *anything*.

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. This is also a requirement for the projects. (We’ll talk about that in a second)

Variable 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 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 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`

Typically raises a `SyntaxError` or `ValueError`.

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.

Operator	Name
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
%	Modulus
//	Floor Division

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 operators we just talked about.

Examples:

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

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. You have to *import* these modules.

Modules / import

Example:

```
import math
```

The ‘`import`’ keyword brings in utilities from a library/file for you to use in your program.

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 its square root function, for example, we would say:

```
math.sqrt()
```

`print()`

The print function takes an object or many objects and displays it in the console for the user to see. Like in mathematics, variables/values input to a function are placed inside the parentheses.

You can input multiple objects to a 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 how functions work and how to create your own 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, type: int/float
- `ndigits` : the nth decimal place you want to round to (optional), type:int/float

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 the user to input text to the console. It takes one optional parameter:

```
input([prompt])
```

- `prompt` : a type:str that displays an in-line message to the console for the user to read.

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

Comments

One of the most important things you can do as a programmer is *document* your code. Documenting your code **is a requirement** for the projects because it’s 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/double quotes. Any text continuing afterwards in the same and following lines is considered a part of the comment until another set of three quotes.

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

Projects

This class has a project due on Mimir nearly every Monday (check the course schedule frequently). I’ve waited this long to talk more about projects because you have to be aware of comments, which is part of something called the “Coding Standard”.

The TAs are required to partially grade each project. Part of the points we reward are given to you if you follow the parts of the coding standard that are necessary for the project. The relevant coding standard procedures you need to follow will (should) be listed in the project description.

I’ll be pretty lenient about following the standard for the first couple of projects, but I will be enforcing and docking more points as time goes on. It’s good to get into the habit of following the standards now.

At this stage in the course, the most important thing you’ll want in your project code are simple comments and headers. Use comments where you think an explanation is necessary -- you’ll have to make the judgment call.

Projects (cont.)

Projects will always have a procedure to follow. The procedure can range anywhere from extremely specific to heartbreakingly vague. It’s good to **start projects early** to get questions you have answered before the deadline.

Projects will always use the concepts we learned from proceeding weeks, since programming concepts build off of each other naturally.

Start early. Don’t copy code.

The remaining TA points

So part of the points I reward are for following the coding standard, but the remaining are for clean-looking, understandable code.

What this means is that you should have descriptive variable names (like the ones we've discussed), and non-repetitive code (we'll talk about what this means at a later date).

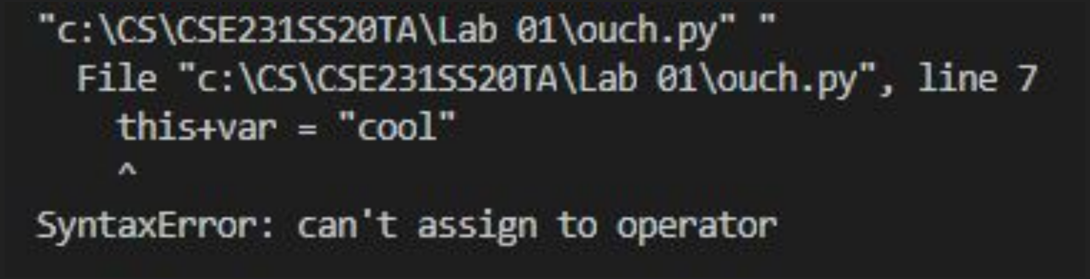
I'm probably the most lenient grader out of all the TAs 😅

But still, you should try to pass all the test-cases, make your code presentable, etc..

If you believe you should have gotten more points on a past project, get in contact. Please note, however, that I may be restricted in giving some points out due to instructions given to me by the professors.

Reading Error Messages

A lot of 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 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.