Algepocalypse Lab

Instructor Guide

Overview

This lab is designed to be part one in an intro to computer programming (it will be the students' first exposure to writing code in Python.) It teaches some basic concepts but also previews many concepts which we don't expect them to fully master here (for loops, strings, conditionals, import random) to get them excited about what code can do!

Learning Goals

Phase 1
- print() displays text on the terminal

Phase 2
- Variables can hold values, like numbers
- Basic Math Operators: +, -, *, **
- (maybe) / vs // (integer division)

Phase 3
- For loop syntax
- Range parameters: (start, end, step)

Phase 4
- input()
- String as a type

- + and * with strings

Phase 5

- If-Else
- ==
- break (extra/optional)

Personal Growth Goals

- Programming is fun and something I can do.
- Understanding why programming is valuable
- Computational Thinking: students get exposure to the step-by-step, methodical ways to think about writing programs

Skills Required

- A basic understanding of substitutionary algebra...maybe? Not completely necessary.
- What an integer is
- Order of operations

Resources Required

- A projector hooked up to a computer
- Computers for either every student or every pair of students
- Python 3 installed on all the computers
- At a minimum, one mentor per three students

Instructor Preparation

1. Make sure all the computers have Python and IDLE installed.
2. Check to see that students have a way to save/access files.
3. Load all the programming files onto each computer.

In Depth Description

This lab is made up of 5 activities, called “adventures.” This lab is designed to have a central instructor with mentors helping one to three students to make sure they are keeping up with what the instructor is leading them through. When explaining content to the students, the instructors should explain on a whiteboard, as oppose to taking over coding from the student. They should write code on the board when explaining new syntax (which is a lot of this lab).

Note: “...” at all points below means giving students time to think, time to answer your question, or time to code, whatever is appropriate.

Algepocalypse Introduction

[if this is the first class, this is after an intro about Teknowledge, who we are, and a cool video of Teknowledge stuff to get them pumped, otherwise do a more normal intro]
Welcome to the first Teknowledge series of learning phases. Today we'll get a taste of the basics of writing code. But first, a story:

(with illustrations for the story, simple ones)

"A very long time ago, in a distant galaxy, there was a planet, Terra Algebra."

"On Terra Algebra lived numbers, peacefully, for 999 years."

"The very first number was 1, created by a mysterious powerful substance called Mathematica."

Code Phase 1: Hello World - First Print

"When 1 appeared as the first number, he said the famous words 'Hello world!'"

Let's remember this first moment with your first program, as is tradition to do.

[when walking through the code on the projector, DON'T WRITE THE CODE UNTIL YOU TALK ABOUT IT. Let them write it based on what you say, then look up to what you write, then correct what they thought. You learn more guessing and correcting your mistakes than mindlessly copying the front projector.]

1. Now, let's each make sure our code editor is open.
2. Then, let's save this empty file that we'll write our code in, call it '1_hello.py'.
3. Now we will first write 'print' in the code, which is a function that will "print" something on the screen when the code runs. Different functions have different abilities, this is a commonly used one to show you what your code is doing.
4. Next, anytime we use functions, we always put a pair of parenthesis after to "call" that function, which means we will activate its power.
5. Inside the parenthesis goes the information that function needs to work, called "parameters". Here the information is what to print! What do we want to print to the screen?
6. "Hello world". Inside the parenthesis write "Hello world!" but with quotation marks. The quotation marks tell the code that these are to be read as words and letters, and not as code.
7. That's it! To "run" your code to see if it works, you'll _____. (press command-B or go to Terminal, whatever we decide here - I prefer pre-setting it up to run python through Sublime, but they may learn more running it in the command prompt)

Code Phase 2: Variables

“So all the numbers got along fine...until one day. The aliens arrived with their strange shapes and sizes. Instead of 10 basic forms, they had 26! They were the letters, and they caused all sorts of chaos, attaching themselves to numbers, and becoming values of their own.”

Now make a new file called ‘2_variables.py’.

Do you know variables from school. What’s the most common variable?
X, that’s right.
At the top of your code, write “x = 2”.
This is called a variable assignment.
We assign the value 2 to the variable x.

Now we can print that variable, with “print(x)”.
Try running that code.
…
2, nice.

But it’s usually nicer if the code tells you what number it’s printing out. So let’s change the print to something that tells us this is the value of x. Like ‘print(“x =”, x)’. Now what does it print?
…
Yeah, so you can print multiple things together on the same line with a comma. This is actually pretty interesting because the first thing we print is a string, inside the quotes, and it’s a value. The second thing is a variable that holds what type of value? … A whole number, called an integer in python.

Now let’s add a second variable. What’s the second most common variable in algebra? Yeah, y. Let’s assign a value to y using x. Write “y = x + 40”.
What value will y have?
Let’s print it out - ‘print(“y =”, y)’.
…

Cool, now let’s add a third variable and use both x and y. You pick what you want to do with them - play with it! Most of the mathematical operations you know and love work in python - adding, subtracting, multiplying, dividing, exponents - try something out that uses both x and y to assign to z.
As an example I’ll write “z = x * y + x”.
Then ‘print(“z =”, z)’.
…
(discuss what people’s findings are with various mathematical operations,
perhaps show exponent is **,

perhaps explain integer division,
perhaps order of operations)

Code is best learned when you experiment with it, play around with it, see what it can do. It's one of the areas you are actually rewarded for trying crazy things - the worst that can happen is you get a code error.

The next part we'll learn about how code can do repetitive tasks for us really easily.

(maybe take a break here if people are fatigued, or it's been 45 minutes,
otherwise do it after phase 3)

Code Phase 3: All The Numbers - First Loop

"After this, the Numbers with the help of variables continued to add to their population, multiply, and cover the planet, each inhabiting their own space and making their mark on the world."

What if we wanted to print all the numbers' names, from 1 to 100? How could we do this with the code you know so far? It is silly, but we could type in a new file "print(1)" then "print(2)" then "print(3)". Does someone want to do this? No? Then my friend _ (one of the helpers), can do it, just for show. [have a helper write this "inefficient" program while the main teacher guides them through writing a simple loop on the board]

But there must be a better way, right? This is a key idea in writing code - if you feel like you're doing something silly because it's so repetitive...there is always a better way, a shorter way. Let's look at that way right now.

Make a new file called "2_loop.py".

So, we want to print all the numbers from 1 to 100, as _ is doing right now. Let's write what we want to do out: "for 1 - 100, print(_)"

That's a start, but it isn't code. How to do this in code? Well first, there's that print, which we want to repeat 100 times.

Now, because we are repeating some code, we'll use what's called a "loop". That loop would change a variable to each number we want it to be in a range, from 1 to 100. How we write this is: "for i in range(1,100):"

To keep track of this number in the range that changes, or to *remember* a value, we use this variable, i, which stands for index. You can pick the name for a variable, it could just as well be the letter x or the word number - there's flexibility here.

For the “variable” i in our “range” from 1 to 100, we will do “print(i)”. But you need to indent it using tab because that code is *inside* the loop. So make sure any repeated code is inside the loop using an indent. Once you’re done you can write code unindented. Now write out these two lines yourself.

Pretty short, huh? How is _ doing over there? Let’s see if we can beat them.
...
So, now you see how programming is cool - you can do something repetitive like this really fast. __, you can stop now, we won!

But wait! Did we get up to 100? No, only up to 99. This is because the range starts at 1, but doesn’t include 100. This is for good reason, trust me, you’ll see later, why not including the last number makes sense when writing more code.

Now rewrite it to include 100.
…
Cool, you have to make the ending 101, right.

Now, let me show you something. Let’s add a 2 to the end of this range function call, adding a third “parameter” that is optional. Sorry that was a lot of coding terms, basically we just add one more thing to the range. [write it on the board: range(1, 100, 2)] Try it yourself, what did it do?
…
[wait until people run the code again] Yeah, it only did the odd numbers. What do you think the 2 does?
…
Yeah, the 2 skips by 2s instead of 1. Try putting 1 back in that third parameter.
…
See, now it is like before.

How would we get only numbers that are divisible by 5 in the range from 1 to 100, including 100?
… [range(5,101,5)]
Nice work! You have to first start at 5, then make sure to skip by 5, and lastly end the range after 100, like 101.

One more example: what do you think “range(5)” does? With only one number?
… [run after guesses]
The 5 is the ending number. The range starts at 0 by default. Up to 4, not including 5. Starting at 0 is another thing computers do that will make a lot more sense the more you code.

In summary, we can write these range “parameters” like [write on board] “range(start, end, step)”. With two parameters, it becomes just “range(start, end)” and goes by 1 step. With one

parameter it is “range(end)” starting at 0 and counting by 1. In all of these, the range does not include the last number.

(maybe take a break here if people are fatigued, or it’s been 45 minutes)

Code Phase 4: Input and Strings

“Then the letters got jealous and started asking questions. They stored all sorts of responses and said that their ‘facts’ were more important than any numerical value.”

Remember the quotation marks in “hello world”? The quotation marks tell the code that these are to be read as words and letters, and not as code. This is called a “string” in code. A type of value we can have, like int for numbers.

So let’s first get acquainted with strings through a program that can ask you for your name and remember it. Better than most humans, right? I wish I could do that for everyone I met. Make a new file called “4_convo.py”.

Let’s get it to ask for your name. To do this, the program needs an input. So you write “input(“Enter your name: “)”.

Then try running your program.
…
So you type your name and that’s it.

But...your code needs to *remember* what you type in. To remember something, you use a … Variable, right! We do what’s called “assigning” the value to the variable using an equals sign. But again, make sure your variable has a good name. Here we can just call it “name”. So, “name = input(“Enter your name: ”)”.

Then we need to do something with that name. Let’s print it, but with a message.
Here we add together strings to make a bigger string.
Try: “print("Hi " + name + "!")”
Try running that.
…
Pretty fun! Let me show you what you’ll make next and then you try making it yourself.
[run rest of program of “4_name.py” with interesting fact:

```
name = input("Enter your name: ")

print("Hi " + name + "!")

fact = input("Tell me an interesting fact: ")
```

```
print(fact + "...that's interesting.")
print("Bye " + name + "!")
]
```

Now, try writing it yourself! Use your mentor as a resource. The text I'll leave up here.
… [if needed, as a hint:] Notice it just repeats the fact that the person enters in and adds "dot dot dot that's interesting".

Cool, this is the code as I wrote it - you can compare and see.

Phase 5: Guess a Number - Conditionals

"In the midst of all the craziness, a new game was created that managed to entertain both numbers and letters alike and kept them at peace for many years. It was called…'Guess A Number'."

Let's code this game that they played. Basically one person picked a number, then the other person tried to guess the number. If they got it right they won, otherwise they lost.

First, we need the player to input a guess.
You know how to do this. Prompt them to "Enter a guess: " and make sure you remember their guess.
…

Then we need something new, a way to tell *if* they got it right, if the guess is equal to some secret number, for now let's say...5. [write on board] So we write in code, "if (guess == 5):". This is called an "if statement". The double-equals is called a "comparison operator", kind of like + but it compares two things and will tell you True or False.

So we need something to happen if this is true. Indent below the if statement and write code to print "You win!".
...

Then it'd be nice to have some feedback if they don't get it right. So then you can unindent below and write "else:" and write what should happen in all the other cases.
…

Now we have a game - try it out! Try winning and try losing to make sure it works.
…

It doesn't work when you win, right! Hmm, this is called a programming bug, a tricky problem where you are unsure of the solution. Any guesses? ... Well, for interest of time, I'll tell you what's wrong - this one actually still gets me tripped up sometimes.

The guess that is made from the input() function is of type STRING not type INT. So the comparison is never equal!

So we have to fix it by turning the string that the input returns into an integer. We do this by the int() function, which you don't know yet, but will solve our problem here! Write the function name int before the function input, then surround the whole input part in parenthesis.
… Now try winning.
… It should work!

But...it's not very fun. The number's always the same! Here's where we can add a cool function that will get a random number from a range.

At the top of your code, write "from random import randint" - this is basically getting a special function randint from another file.

Now to use randint, instead of 5 write "randint(1, 3)". This will pick a number from 1 to 3, including 3, and will pick randomly every time your code runs. Try it out.

...

Alright, now we've got a good game going!

[If time, do the below]

Things that could make the game better:
- Let player 1 pick the number and player 2 have 5 guesses. (use another input)
- Tell the player "too high" or "too low" after their guess (will need to know < and >)
- Make it a number between 1 and 10 but give the player 5 guesses. (use a loop with break, as in the "guessANumber_extra.py" file)

Conclusion

So hopefully you see code can do some pretty cool stuff! And just with this little bit of knowledge so far. We'll be doing much more in the coming weeks, and it will only get more cool.

And as for the Algepocalypse storyline? To be continued in a future lesson…

Lesson Plan

*(:10) means that this part should be done by the tenth minute of the lesson)

1. Setup (:15)
    a. Divide into groups
    b. Get ready to code (open IDLE all the “adventure” Python files.)
2. Hello World (:25)
3. First Loop (:40)
4. Input and Strings (:60)
5. Guess a Number (:end)