CS2110 Lecture 2 Jan. 27, 2021

- First homework available, due Feb. 4
- I will post first Discussion Section Assignment, DS1, tomorrow, due Tuesday afternoon. Posting it early because it's a great first step for HW1. Don't work on HW1 until you do DS1 Assignment. You *can* wait until DS on Tuesday to do it but then you won't have a lot of time left for HW1.
- Office hours for me and TA are on the main course website
- Given the on-line nature of the course, I want to be as available for help as possible
  - Don't hesitate to send questions and comments to me and TA by **email**. BUT, when you send email to TA, always also include me.
  - DO NOT sent multiple messages (one to TA, one to me). This often wastes people's time and makes a mess!
  - For homework-related email: *ALWAYS* **attach** file of your current code (UI changed attachment rules, disallowing .py attachments. For now put .py file in a folder, zip it, and attach .zip file to email) Don't just copy/paste code into message body!
  - You may send me simple questions by **text** (e.g. "Do you have time for a quick Zoom meeting today at ??"). Identify yourself when doing so ("I am … in your CS2110 class")
  - Also, USE Piazza. We'll answer questions quickly there as well (so far only some of you have "joined" the Piazza "course")

IMPORTANT THINGS TO DO THIS WEEK

- Read the textbook, Ch 1 and Ch2
- Install Python on your computer
- Practice basic expressions in the Python interpreter
- Make sure you can access Piazza (discussion forum) through the ICON

Today

- Course overview
- Chapter 1 of text, and a bit of Ch 2
  - Programs
  - Python
  - Evaluating simple expressions
  - Types
  - Natural vs formal languages
  - Debugging
- Example GUI/web program demo you'll soon be able to create

Course overview

- The first seven weeks: basic python programming
  - Expressions, types, variables, conditions, functions, iteration
  - Use of sequence and dictionary data types

  Quizzes 1 and 2 will focus on these basics
- Additional topics (approx. one week each)
  - Classes and object oriented programming
  - Comprehensions and other more powerful Python language tools
  - Program design and debugging
  - Algorithmic efficiency
  - Searching and sorting
  - GUIs
  - Graphing, plotting
  - Accessing web data
  - Machine learning or other advanced topics

Ch 1.2 and 1.5: Algorithms and programs

It is useful to distinguish between the words “algorithm” and “program.”

dictionary.com: An algorithm is a “finite set of unambiguous instructions performed in a prescribed sequence to achieve a goal.”

(non-computing examples?)

Programming is the process of expressing an algorithm in a programming language (so you can execute it on a computer)

Or, I like to say:

program = algorithm + programming language

Ch 1: Programs

Key components of programming, independent of particular programming language.

- Expressions
- Variables and assignment
- if-then-else (decision making/branching)
- Iteration
- Functions

Essentially all programming languages are built around these components. Once you understand how to describe computations using them, you can program. **Learn these basics well!** Changing programming languages is usually just a matter of looking up details of how to "say" a particular standard thing in the different language.

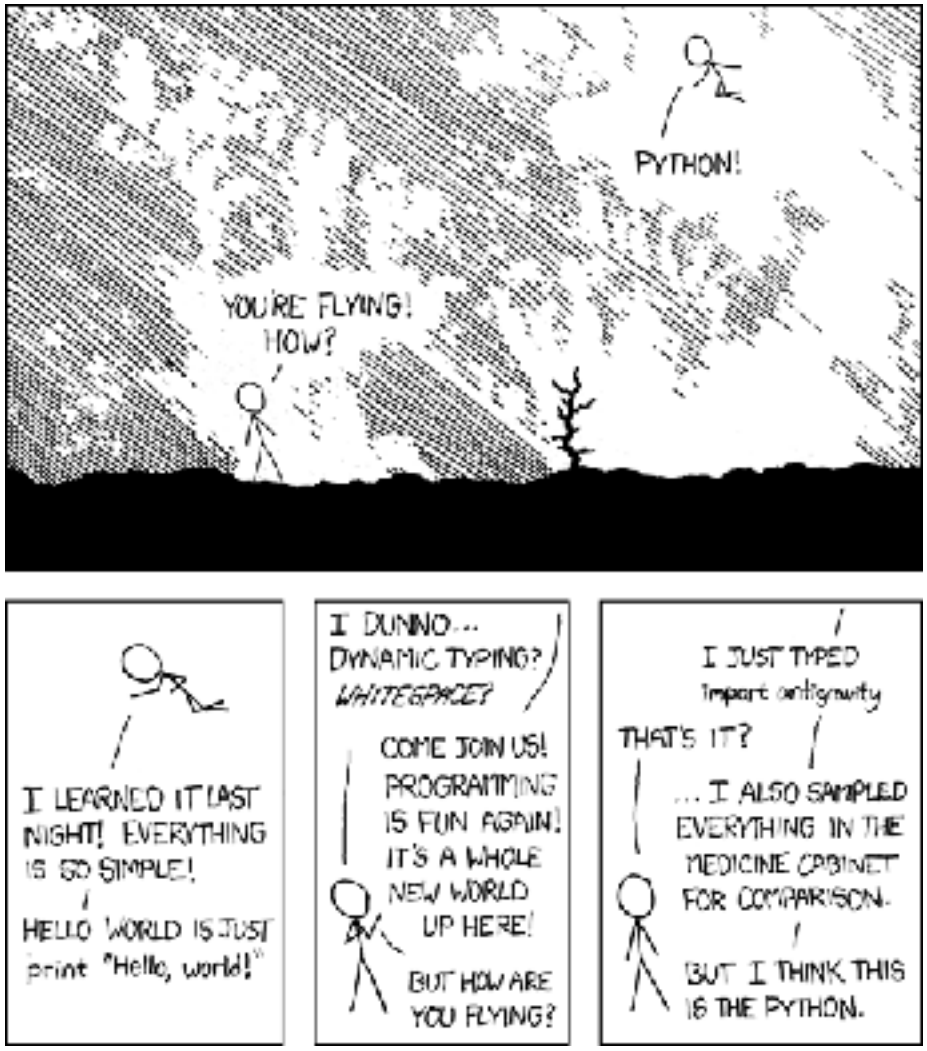
(In 1.5, the textbook also includes input and output in this list and lumps expressions, variables, and assignment under 'math and logic')

Ch 1: Python

There are many many programming languages!

Why Python?

- Clean syntax, powerful constructs enable beginners to more quickly get computer to do interesting things
- Interactive mode encourages experimentation
- Extensive standard and third party libraries for web programming, scientific computation, data analysis, etc.



https://xkcd.com/353/

Ch 1: Using Python

- When working with Python, we use a Python **interpreter**, which is a program that reads Python code and executes it.
- Demo: Python interpreter from Mac terminal
- Demo (the most common way I will do things and recommended approach for students): Python interpreter plus IDLE IDE.
- Demo: Anaconda's Spyder IDE
- Demo: you can use Python directly within the on-line textbook (sec 1.4)

Expressions, arithmetic, types (from Ch 2 – we'll cover more on Friday)

- You can use the Python interpreter like a calculator, typing in many different mathematical (and other) expressions and seeing immediate results
  - print("hello")
  - 3 + 2
  - 2**128
  - 5/2+1
- Expressions yield **values.** Every value has a **type.**
  - 3's type in **int** (for integer)
  - 3.5's type is **float** (for floating point number)
    - (What about 3.0? 3.0's type is float. It is not the exact same thing as 3 in Python but (fortunately) it *is* "equal" to 3. More on this later.)
  - "hello"'s type is **string**
  - You can ask Python for a value's type

    >>> type(3.5)

(Note: don't use commas in big numbers: 1,000,000 is not million in Python!)

Again … basic programming components, independent of particular programming language.

- Expressions
- Variables and assignment
- if-then-else (decision making/branching)
- Iteration
- Functions

Expression: `3 + (2 * b)`

Assignment:
```
counter = 1
counter = counter + 2
```

Decision making:
```
if counter > 0:
    print("counter is positive")
else:
    print("counter is not positive")
```

Iteration:
```
while (counter > 0):
    print(counter * counter)
    counter = counter – 1
```

Function:
```
def plus5(number):
    return number + 5
```

1.6-1.9 Debugging and programming errors

- ”Debugging” is the process of finding and correcting errors in programs. It is a critically important skill. It DOES not require special tools (debuggers). Many people do most of their debugging using a combination of carefully-placed “print” statements and careful critical reading of their code.
- Error types:
  - Syntax error: improperly formed Python
    - E.g. 3 = x Python won’t even let you try to execute this
  - Runtime error:
    - E.g. 3/x when x has value 0 Python generates an error while executing the program
  - Semantic error:
    - Errors where the output does not meet your specification. The program has no syntax errors, and executes without encountering a runtime error, but does not compute what was required. Python doesn’t detect this kind of error! You need to carefully assess whether your code is computing the correct thing in all required cases!

Ch 1.11: natural vs formal languages

One major difference: formal programming languages are unambiguous.

Natural languages are ambiguous, which can be annoying and confusing but also enables playfulness and creativity (jokes, poetry, etc.)

Natural languages have lots of redundancy – many ways to express the same thing. Programming languages also redundant but less so.

Programming languages are dense but they can be directly and unambiguously translated into basic machine-level operations

Computer systems read them quickly but humans need to read them too. STRONG RECOMMENDATION: get in the habit of reading programs slowly, carefully, and critically. Make sure you understand what they really say rather than what you believe or hope they say)

- IC Arrest Blotter
- Demo: blotter.py

Later in the semester you should be able to make this program.

Next time - very important basics

- CH 2: values, expressions, types, variables, and assignment
- For next time, read and do practice work in Ch2