Lecture 6: Introduction to Python

Fall 2022

Jason Tang

Topics

- How computers run code
- Anatomy of Python
- CMSC 104 coding standards

Machine Code

- In the beginning, people “wrote” programs in machine code (i.e., binary)
  - Error prone
  - Not exactly fun
    - Nobody does this anymore

```
01111100111000110001100111010110
01111101000010000100000111010110
01111100011001110100010000010100
01001110100000000000000000100000
```

- Machine code (obviously) only ran on specific machines

Assembly

- As a shortcut, programmers created pneumonic “words” for common binary patterns
  - Computer assembled these words to binary code
  - Still not fun, but better than nothing
  - Still specific to particular machines

```
MULLW   R7,R3,R3
MULLW   R8,R4,R4
ADD     R3,R7,R8
BLR
```

- Still in use today -- including by your friendly lecturer!

First Computer Languages

- Fortran - scientific computing
    - Oldest programming language still in use (1957)
- Lisp - based on lambda calculus
    - Second oldest language still used today (1958)
- Modern languages are no longer tied to specific machines

```
S1 = 3.0
S2 = 4.0
H2 = (S1 * S1) + (S2 * S2)
```

```
(defun sumsquare (s1 s2)
  (+ (* s1 s1) (* s2 s2)))
```

C Programming Language

- Derived from language “B”, 1972
  - Fairly efficient, close to the machine
  - Sometimes called “portable assembly”
- Language used to write Linux itself
- One of the most widely used computer language today

```
int sumsquare(int s1, int s2) {
    return ((s1*s1) + (s2*s2));
}
```

Python

- Introduced in 1991 as a modern language
  - Named after *Monty Python's Flying Circus*
- Emphasizes code readability and reducing number of lines of code necessary to express the same concepts as C
  - Useful side-effect is that Python is a good first language for beginners
- Latest version is Python 3.10
  - In this class, anything from Python 3.6 to newer is acceptable

https://www.geeksforgeeks.org/history-of-python/;
https://www.python.org/community/logos/

Why Different Computer Languages?

- Just like human languages, different computer languages were designed to meet different goals:
    - Solve different kinds of problems
    - Speed of execution
    - Size (in bytes of memory)
    - Ease of learning
    - Ease of writing

Compiled Languages

- In the end, computers are still machines that operate in binary

- Programmers use an editor, like **`emacs`** or **`nano`**, to create source files containing computer code

- For some computer languages like C, programmers then use a compiler

  - Compiler reads source files

  - Compiler converts source code into usable machine code

- Find out more about machine code in CMSC 411

Interpreted Languages

- Alternative is for programmers to write their code with a text editor
- Then the computer runs a special program called an interpreter that takes the source code and dynamically turns it into execution for *just that instance*

Compiled Versus Interpreted Languages

- Example of Compiled: Write a book in English, republish the text into Braille, and then give the new text to a visually impaired reader

- Example of Interpreted: Write a book in English, and then read the book out load to a visually impaired person

Language Type	Pro	Con	Examples
Compiled	Takes full advantage of hardware	Harder to learn and master	C, C++, Java, Fortran
Interpreted	Quicker to write and prototype	Slower execution	Python, PHP, Perl, Lisp

Sample Python Program

```
print("Hello, world!")
```

- This is a valid, syntactically, and semantically correct Python program

- These lecture slides will have syntax highlighting to automatically colorize parts of the code; the colors themselves are ignored by the compiler

- Simple text editors, like **`nano`**, do not perform syntax highlighting

- More advanced text editors, like **`emacs`**, add syntax highlighting and other advanced functionality

Actual File Contents

```
print("Hello, world!")
```

- While the above program is human readable text, internally the computer transforms every character into a binary value via the ASCII encoding

- If the above text were saved to the file **`foo.py`**, the computer would store exactly these bytes:

```
0x70 0x72 0x69 0x6e 0x74 0x28 0x22 0x48
0x65 0x6c 0x6c 0x6f 0x2c 0x20 0x77 0x6f
0x72 0x6c 0x64 0x21 0x22 0x29 0x0a
```

Running Python Code

- Most common way to run Python code is via its interpreter
- On GL, that interpreter is at **`/usr/bin/python`**
- Execute the code like so:

```
$ /usr/bin/python foo.py
```

- In Linux, **`/usr/bin`** is usually within your search path, so you can also run the code like so:

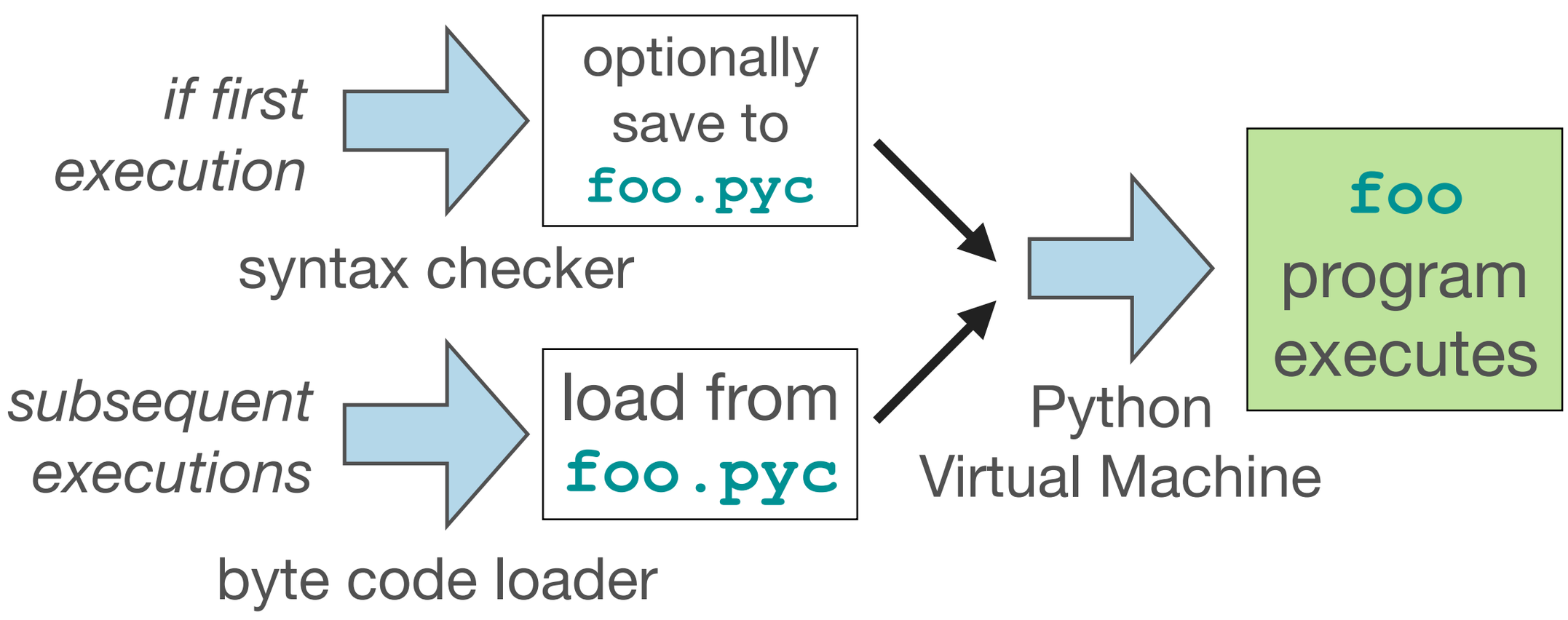
```
$ python foo.py
```

- If you don't have a Linux computer handy, you can also use the website **`https://www.online-python.com`** to prototype some code

How Interpreters Really Work

- All interpreters read the source code file and then executes it
- Simple interpreters read one line at a time, analyzes the syntax, and then runs just that line
- Newer interpreters read a line, then transforms it into an intermediate form called byte code
  - Complex lines are broken down into multiple, simpler byte code instructions
  - Each byte code instruction roughly corresponds to an actual computer hardware operation
  - Byte code can be saved to disk, to speed up later program invocations

Python Execution Summary



Anatomy of a Python Program

```
1 print("Hello, world!")
```

- Save the above to a file named **`foo.py`** and then run the code

- These slides and many printed Python programs prefix code with line numbers
  - These line numbers merely identify lines of code; programmers do not actually enter those lines
  - Line 1 causes the computer to output ("print") a line for the human
- All parts of this code are necessary: opening and closing parentheses, both double quotation marks

Python Syntax Error

```
1 print("Hello, world!"")
```

- Modify the code by adding a third double-quotation mark (indicated above in pink)

- Now try to run the code

```
$ python foo.py
  File "/afs/umbc.edu/users/j/t/jtang/home/foo.py", line 1
    print("Hello, world!"")
                          ^
SyntaxError: EOL while scanning string literal
```

- Mnemonic: for every opening double-quotation mark, there needs to be a closing mark before the end of line (EOL)

Another Python Program

```
1 #
2 # This is my second program
3 #
4 print("Will\nit\nblend?")
```

- Lines 1 through 3 is a program comment
    - Comments are for the benefit of human reader; the interpreter ignores them
- Line 4 uses the **`print`** routine to display something to the screen

Comments

- Descriptive text to aid the *reader* to understand program contents
- Ignored by preprocessor (and therefore compiler and linker)
- Begins with **#** (“pound character”, “hashtag”, “octothorpe”) and continues to end of line
- All of your homework and project submissions **must** have file header comments
  - This comment **must** have your name and email address; when submitting a project then also include your partner’s name and address
  - Then write a short description of the program’s purpose

`print("Will\nit\nblend?")`

- A single Python statement

- Calls pre-made Python algorithm (i.e., function) named **`print`** with a single argument (i.e., input)

- That argument is the string **`Will\nit\nblend?`**

`"Will\nit\nblend?"`

- Python strings are sequences of characters enclosed by either single-quotes or double-quotes
    - `'Will\nit\nblend?'` and `"Will\nit\nblend?"` are equivalent
        - **Caution**: `'` is not the same as ‘; likewise `"` is not the same as “
- Backslash introduces an escape sequence
    - `\n` represents a newline (as if hitting Enter key), `\'` is the literal single quote
- So what does `"Will\nit\nblend?"` represent?

Coding Style

- See course web site for Python programming standards and indentation guidelines

- *All* homework submissions must conform to these standards, such as including file header comments

  - **All assignments have a style grade**

  - Bad style will negatively impact your grade

- Note: comments and proper spacing are not required during exams