BBM 101
Introduction to Programming I

Lecture #03 – Introduction to Python and Programming, Control Flow
Last time... How to build computers

The Harvey Mudd Miniature Machine (HMMM)

triangle1.hmmm: Calculate the approximate area of a triangle.

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

$python hmmAssembler.py -f triangle1.hmmm -o triangle1.b$

Assembly Successful

<table>
<thead>
<tr>
<th>ASSEMBLY SUCCESSFUL</th>
</tr>
</thead>
</table>

$python hmmSimulator.py -f triangle1.b -n 4 5 10$

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000 0001 0000 0001</td>
</tr>
<tr>
<td>1</td>
<td>0000 0010 0000 0001</td>
</tr>
<tr>
<td>2</td>
<td>1000 0001 0001 0010</td>
</tr>
<tr>
<td>3</td>
<td>0001 0010 0000 0010</td>
</tr>
<tr>
<td>4</td>
<td>1001 0001 0001 0010</td>
</tr>
<tr>
<td>5</td>
<td>0000 0001 0000 0010</td>
</tr>
<tr>
<td>6</td>
<td>0000 0000 0000 0000</td>
</tr>
</tbody>
</table>

Memory

Control

ALU

PC

inst_1

inst_2

inst_3

... inst_N

3.3V

2.8V

0.5V

0.0V

power

input

output

x

y

x \rightarrow y

x \rightarrow y

0.0V

0.5V

2.8V

3.3V

0

1

x

y
Lecture Overview

• Programming languages (PLs)

• Introduction to Python and Programming

Disclaimer: Much of the material and slides for this lecture were borrowed from
— E. Grimson, J. Guttag and C. Terman MIT 6.0001 class
— Ruth Anderson, Michael Ernst and Bill Howe’s CSE 140 class
— Swami Iyer’s Umass Boston CS110 class
Lecture Overview

• Programming languages (PLs)

• Introduction to Python and Programming
Programming Languages

• Syntax and semantics
• Dimensions of a PL
• Programming paradigms
Programming Languages

• An artificial language designed to express computations that can be performed by a machine, particularly a computer.

• Can be used to create programs that control the behavior of a machine, to express algorithms precisely, or as a mode of human communication.

• e.g., C, C++, Java, Python, Prolog, Haskell, Scala, etc..
Creating Computer Programs

• Each programming language provides a set of primitive operations.

• Each programming language provides mechanisms for combining primitives to form more complex, but legal, expressions.

• Each programming language provides mechanisms for deducing meanings or values associated with computations or expressions.
Aspects of Languages

• Primitive constructs
  – Programming language – numbers, strings, simple operators
  – English – words

• Syntax – which strings of characters and symbols are well-formed
  – Programming language – we’ll get to specifics shortly, but for example 3.2 + 3.2 is a valid C expression
  – English – “cat dog boy” is not syntactically valid, as not in form of acceptable sentence
Aspects of Languages

• Static semantics – which syntactically valid strings have a meaning

  – English – “I are big” has form <noun> <intransitive verb> <noun>, so syntactically valid, but is not valid English because “I” is singular, “are” is plural

  – Programming language – for example, <literal> <operator> <literal> is a valid syntactic form, but 2.3/’abc’ is a static semantic error
Aspects of Languages

• Semantics – what is the meaning associated with a syntactically correct string of symbols with no static semantic errors

  – English – can be ambiguous
    • “They saw the man with the telescope.”

  – Programming languages – always has exactly one meaning
    • But meaning (or value) may not be what programmer intended
Where Can Things Go Wrong?

• Syntactic errors
  – Common but easily caught by computer

• Static semantic errors
  – Some languages check carefully before running, others check while interpreting the program
  – If not caught, behavior of program is unpredictable

• Programs don’t have semantic errors, but meaning may not be what was intended
  – Crashes (stops running)
  – Runs forever
  – Produces an answer, but not programmer’s intent
Our Goal

• Learn the syntax and semantics of a programming language

• Learn how to use those elements to translate “recipes” for solving a problem into a form that the computer can use to do the work for us

• Computational modes of thought enable us to use a suite of methods to solve problems
Dimensions of a Programming Language

Low-level vs. High-level

• Distinction according to the level of abstraction

• In low-level programming languages (e.g. Assembly), the set of instructions used in computations are very simple (nearly at machine level)

• A high-level programming language (e.g. Python, C, Java) has a much richer and more complex set of primitives.
Dimensions of a Programming Language
General vs. Targeted

• Distinction according to the range of applications

• In a general programming language, the set of primitives support a broad range of applications.

• A targeted programming language aims at a very specific set of applications.
  – e.g., MATLAB (matrix laboratory) is a programming language specifically designed for numerical computing (matrix and vector operations)
Dimensions of a Programming Language
Interpreted vs. Compiled

• Distinction according to how the source code is executed

• In interpreted languages (e.g. LISP), the source code is executed directly at runtime (by the interpreter).
  – Interpreter control the flow of the program by going through each one of the instructions.

• In compiled languages (e.g. C), the source code first needs to be translated into an object code (by the compiler) before the execution.
Programming Language Paradigms

• Functional
  • Treats computation as the evaluation of mathematical functions (e.g. Lisp, Scheme, Haskell, etc.)

• Imperative
  • Describes computation in terms of statements that change a program state (e.g. FORTRAN, BASIC, Pascal, C, etc.)

• Logical (declarative)
  • Expresses the logic of a computation without describing its control flow (e.g. Prolog)

• Object oriented
  • Uses "objects" – data structures consisting of data fields and methods together with their interactions – to design applications and computer programs (e.g. C++, Java, C#, Python, etc.)
Programming Language Paradigms

• **Functional**
  - Treats computation as the evaluation of mathematical functions (e.g. Lisp, Scheme, Haskell, etc.)

• **Imperative**
  - Describes computation in terms of statements that change a program state (e.g. FORTRAN, BASIC, Pascal, C, etc.)

• **Logical (declarative)**
  - Expresses the logic of a computation without describing its control flow (e.g. Prolog)

• **Object oriented**
  - Uses "objects" – data structures consisting of data fields and methods together with their interactions – to design applications and computer programs (e.g. C++, Java, C#, Python, etc.)
Lecture Overview

• Programming languages (PLs)

• Introduction to Python and Programming
Programming in Python

• Our programming environment
  – Python programming language
  – PyCharm, an integrated development environment (IDE)
  – Terminal
Programming in Python

• To program in Python
  – Compose a program by typing it into a file named, say, `helloworld.py`
  – Run (or execute) the program by typing `python helloworld.py` in the terminal window
Input and Output

• Bird’s-eye view of a Python program

  input → my_program.py → output

  – **Input types:** command-line arguments, standard input, file input
  – **Output types:** standard output, file output, graphical output, audio output
Input and Output

• Command-line arguments are the inputs we list after a program name when we run the program

    $ python my_program.py arg_1 arg_2 ... arg_n

• The command-line arguments can be accessed within a program, such as my_program.py above, via the array (aka list) sys.argv\(^1\) as sys.argv[1], sys.argv[2], . . . . , sys.argv[n]

• The name of the program (my_program.py) is stored in sys.argv[0]

\(^1\)The sys module provides access to variables and functions that interact with the Python interpreter
import sys

print('Hi, ', end='')
print(sys.argv[1], end='')
print(' How are you?')

$ python useargument.py Alice
Hi, Alice. How are you?
$ python useargument.py Bob
Hi, Bob. How are you?
$ python useargument.py Carol
Hi, Carol. How are you?
1. Python is like a calculator

2. A variable is a container

3. Different types cannot be compared

4. A program is a recipe

**Colvin Run Mill Corn Bread**

1 cup cornmeal  
1 cup flour  
1/2 teaspoon salt  
4 teaspoons baking powder  
3 tablespoons sugar  
1 egg  
1 cup milk  
1/4 cup shortening (soft) or vegetable oil

Mix together the dry ingredients. Beat together the egg, milk and shortening/oil. Add the liquids to the dry ingredients. Mix quickly by hand. Pour into greased 8x8 or 9x9 baking pan. Bake at 425 degrees for 20-25 minutes.
1. Python is Like a Calculator
You Type Expressions. Python Computes Their Values.

- 5
- 3+4
- 44/2
- 2**3
- 3*4+5*6
- (72 – 32) / 9 * 5

Python has a natural and well-defined set of precedence rules that fully specify the order in which the operators are applied in an expression:

- For arithmetic operations, multiplication and division are performed before addition and subtraction.
- When arithmetic operations have the same precedence, they are left associative, with the exception of the exponentiation operator **, which is right associative.
- We can use parentheses to override precedence rules.
An Expression is Evaluated From the Inside Out

• How many expressions are in this Python code?

\[
(72 - 32) / 9.0 * 5
\]

\[
(72 - 32) / 9.0 * 5
\]

\[
(40) / 9.0 * 5
\]

\[
40 / 9.0 * 5
\]

\[
4.44 * 5
\]

\[
22.2
\]
Another Evaluation Example

\[
\frac{(72 - 32)}{(9.0 \times 5)}
\]

\[
\frac{40}{(9.0 \times 5)}
\]

\[
40 \div (9.0 \times 5)
\]

\[
40 \div (45.0)
\]

\[
40 \div 45.0
\]

\[.888\]
2. A Variable is a Container

A variable is a name associated with a data-type value.
Variables Hold Values

• Recall variables from algebra:
  – Let x = 2 ...
  – Let y = x ...

• To assign a variable, use “varname = expression”
  \[
  \text{pi} = 3.14 \\
  \text{pi} \\
  \text{var} = 6 \times 10^{23} \\
  22 = x \quad \# \text{Error!}
  \]

• Not all variable names are permitted!
  • Variable names must only be one word (as in no spaces)
  • Variable names must be made up of only letters, numbers, and underscore (_)
  • Variable names cannot begin with a number
Changing Existing Variables ("re-binding" or "re-assigning")

\[ x = 2 \]
\[ x \]
\[ y = 2 \]
\[ y \]
\[ x = 5 \]
\[ x \]
\[ y \]

- "=" in an assignment is **not** a promise of eternal equality
  - This is **different** than the mathematical meaning of "="

- Evaluating an expression gives a new (copy of a) number, rather than changing an existing one
How an Assignment is Executed

1. Evaluate the right-hand side to a value
2. Store that value in the variable

```python
x = 2
print(x)
y = x
print(y)
z = x + 1
print(z)
```

State of the computer:
```
x: 2
y: 2
z: 3
```

Printed output:
```
2
2
3
```

To visualize a program’s execution: [http://pythontutor.com](http://pythontutor.com)
More Expressions: Conditionals
(value is True or False)

22 > 4  # condition, or conditional
22 < 4  # condition, or conditional
22 == 4  ...
x = 100  # Assignment, not conditional!
22 = 4  # Error!
x >= 5
x >= 100
x >= 200
not True
not (x >= 200)
3<4 and 5<6
4<3 or 5<6
temp = 72
water_is_liquid = (temp > 32 and temp < 212)

Numeric operators: +, *, **
Boolean operators: not, and, or
Mixed operators: <, >=, ==
More Expressions: strings

• A string represents **text**
  – 'Python'
  – `myString = "BBM 101-Introduction to Programming"
  – ''

• Empty string is not the same as an unbound variable
  – '' and ‘‘ are the same

• We can specify tab, newline, backslash, and single quote characters using escape sequences ‘\t’, ‘\n’, ‘\’’, and ‘’’, respectively

**Operations:**

• **Length:**
  – `len(myString)`

• **Concatenation:**
  – "Hacettepe" + " " + ' University'

• **Containment/searching:**
  – 'a' in `myString`
  – "a" in `myString`
Strings

ruler1 = '1'
ruler2 = ruler1 + ' 2 ' + ruler1
ruler3 = ruler2 + ' 3 ' + ruler2
ruler4 = ruler3 + ' 4 ' + ruler3
print(ruler1)
print(ruler2)
print(ruler3)
print(ruler4)
3. Different Types cannot be Compared

```python
anInt = 2
aString = "Hacettepe"
anInt == aString  # Error
```
Types of Values

• Integers (**int**): \(-22, 0, 44\)
  – Arithmetic is **exact**
  – Some funny representations: \(12345678901L\)

• Real numbers (**float**, for “floating point”): \(2.718, 3.1415\)
  – Arithmetic is **approximate**, e.g., \(6.022*10**23\)

• Strings (**str**): "I love Python", " "

• Truth values (**bool**, for “Boolean”): True, False

George Boole
Operations Behave differently on Different Types

3.0 + 4.0
3 + 4
3 + 4.0
"3" + "4"  # Concatenation
3 + "4"   # Error
3 + True  # Error

Moral: Python only *sometimes* tells you when you do something that does not make sense.
## Operations on Different Types

<table>
<thead>
<tr>
<th></th>
<th>Python 3.5</th>
<th>Python 2.x</th>
</tr>
</thead>
<tbody>
<tr>
<td>15.0 / 4.0</td>
<td>3.75</td>
<td>3.75</td>
</tr>
<tr>
<td>15 / 4</td>
<td>3.75</td>
<td>3</td>
</tr>
<tr>
<td>15.0 / 4</td>
<td>3.75</td>
<td>3.75</td>
</tr>
<tr>
<td>15 / 4.0</td>
<td>3.75</td>
<td>3.75</td>
</tr>
<tr>
<td>15.0 // 4.0</td>
<td>3.0</td>
<td></td>
</tr>
<tr>
<td>15 // 4</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>15.0 // 4</td>
<td>3.0</td>
<td></td>
</tr>
<tr>
<td>15 // 4.0</td>
<td>3.0</td>
<td></td>
</tr>
</tbody>
</table>

Before Python version 3.5, operand used to determine the type of division.

//: Integer Division

/: Division
Type Conversion

`float(15)` 15.0
`int(15.0)` 15
`int(15.5)` 15
`int("15")` 15
`str(15.5)` 15.5
`float(15) / 4` 3.75
A Program is a Recipe
Design the Algorithm Before Coding

• We should think (design the algorithm) before coding

• Algorithmic thinking is the logic. Also, called problem solving

• Coding is the syntax

• Make this a habit

• Some students do not follow this practice and they get challenged in all their courses and careers!
What is a Program?

• A program is a sequence of instructions

• The computer executes one after the other, as if they had been typed to the interpreter

• Saving your work as a program is better than re-typing from scratch

```
x = 1
y = 2
x + y
print(x + y)
print("The sum of", x, "and", y, "is", x+y)
```
The `print()` Statement

- The `print` statement always prints one line
  - The next print statement prints below that one

- Write 0 or more expressions after `print`, separated by commas
  - In the output, the values are separated by spaces

- Examples:
  ```
  x = 1
  y = 2
  print(3.1415)
  print(2.718, 1.618)
  print()
  print(20 + 2, 7 * 3, 4 * 5)
  print("The sum of", x, end="")
  print(" and", y, "is", x+y)
  ```

  3.1415
  2.718 1.618
  22 21 20
  The sum of 1 and 2 is 3
  To avoid newline
Exercise: Convert Temperatures

• Make a temperature conversion chart as the following

• Fahrenheit to Centigrade, for Fahrenheit values of: -40, 0, 32, 68, 98.6, 212

• \( C = (F - 32) \times \frac{5}{9} \)

• Output:

<table>
<thead>
<tr>
<th>Fahrenheit</th>
<th>Centigrade</th>
</tr>
</thead>
<tbody>
<tr>
<td>-40</td>
<td>-40.0</td>
</tr>
<tr>
<td>0</td>
<td>-17.7778</td>
</tr>
<tr>
<td>32</td>
<td>0.0</td>
</tr>
<tr>
<td>68</td>
<td>20.0</td>
</tr>
<tr>
<td>98.6</td>
<td>37.0</td>
</tr>
<tr>
<td>212</td>
<td>100.0</td>
</tr>
</tbody>
</table>

• You have created a Python program!

• (It doesn’t have to be this tedious, and it won’t be.)
Expressions, Statements, and Programs

- An **expression** evaluates to a value
  
  \[ 3 + 4 \]
  
  \[ \pi \times r^{2} \]

- A **statement** causes an effect
  
  \[ \pi = 3.14159 \]
  
  \[ \text{print}(\pi) \]

- Expressions appear within other expressions and within statements
  
  \[ (\text{fahr} - 32) \times (5.0 / 9) \]
  
  \[ \text{print}(\pi \times r^{2}) \]

- A statement may *not* appear within an expression
  
  \[ 3 + \text{print}(\pi) \quad \# \text{Error!} \]

- A **program** is made up of statements
  
  - A program should do something or communicate information
1. Python is like a calculator

2. A variable is a container

3. Different types cannot be compared

4. A program is a recipe

**Colvin Run Mill Corn Bread**

1 cup cornmeal
1 cup flour
1/2 teaspoon salt
4 teaspoons baking powder
3 tablespoons sugar
1 egg
1 cup milk
1/4 cup shortening (soft) or vegetable oil

Mix together the dry ingredients. Beat together the egg, milk and shortening/oil. Add the liquids to the dry ingredients. Mix quickly by hand. Pour into greased 8x8 or 9x9 baking pan. Bake at 425 degrees for 20-25 minutes.
Programming Languages

• A programming language is a “language” to write programs in, such as Python, C, C++, Java

• The concept of programming languages are quite similar

• Python:  \texttt{print(“Hello, World!”)}

• Java:  \texttt{public static void main(String[] args) { 
  System.out.println(“Hello, World!”); 
}}

• Python is simpler! That’s why we are learning it first 😊
Evolution of Programming Languages
The 2017 Top Programming Languages

<table>
<thead>
<tr>
<th>Language Rank</th>
<th>Types</th>
<th>Spectrum Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Python</td>
<td>![globe] ![desktop] ![mobile]</td>
<td>100.0</td>
</tr>
<tr>
<td>5. C#</td>
<td>![globe] ![mobile] ![desktop]</td>
<td>89.4</td>
</tr>
<tr>
<td>6. PHP</td>
<td>![globe]</td>
<td>84.9</td>
</tr>
<tr>
<td>7. R</td>
<td>![desktop]</td>
<td>82.9</td>
</tr>
<tr>
<td>8. JavaScript</td>
<td>![globe] ![mobile]</td>
<td>82.6</td>
</tr>
<tr>
<td>9. Go</td>
<td>![globe] ![desktop]</td>
<td>76.4</td>
</tr>
<tr>
<td>10. Assembly</td>
<td>![desktop]</td>
<td>74.1</td>
</tr>
</tbody>
</table>
