

SAMS

Programming A/B

Lecture #1 – Introductions
July 3, 2017

Mark Stehlik

Outline for Today

- Overview of Course
- A Python intro to be continued in lab on Wednesday (group A) and Thursday (group B)

The Course Staff

- Me
 - Mark Stehlik <mjs@cs.cmu.edu>
 - Teaching Professor
 - available 11:30 – noon and 5:00 – 5:30 (GHC 6205)
- "The other guy"
 - Anil Ada <aada@cs.cmu.edu>, teaching group C
- TAs
 - 6 undergrads, available 6:30-9:00 and on weekends

Course Logistics

- Course website:
 - <http://www.cs.cmu.edu/~aada/courses/SAMS17/>
- Lectures
 - Come on time; use of electronic devices is prohibited during lecture (you're here to learn to program, not surf the web or talk to your friends – do that on *your* time)
- Handin
 - via autolab (more on this later)

You

- Students who want to learn about programming and computer science
- No experience necessary (we will instrument that), not probably intending to major in CS (but...)

Course content

An Introduction to Computing (two parts):

- understanding algorithms
- implementing algorithms (writing programs); requires learning about, and practicing with, “the tools”:

functions

expressions

conditionals

loops

strings

lists

graphics

Course elements

- Homeworks due Saturday 5:00pm (this week there will be a short assignment due Thursday to make sure you can use the labs)
- Weekly quizzes on Friday

Collaboration Policy

There are no group assignments in this class

Everyone should read and abide by:

<http://www.cmu.edu/policies/documents/AcademicIntegrity.htm>

Here is some additional information for this course:

- You *are* allowed to talk with/work with other students on homework assignments
 - You can share ideas
 - You can discuss things at a high (algorithmic, non-code) level (pictures)
 - **You should not share (or even look at) code!**
- You *must* turn in your own work
 - Your solution should be different than others
 - The harder the assignment, the more differences we should see
 - **You should NEVER copy another student's file as a basis for your solution. You should not let your files be copied by others!**
- If you need help debugging, who do you ask?

Programming vs. Computer Science



What is Python?

- Python is a programming language
 - What's a programming language?
 - A language that has a set of instructions/statements that, when assembled correctly (syntactically and semantically) can be compiled/interpreted by a computer and run (executed) to perform a task
 - So, it's a language, like English, Spanish, etc. with rules for syntax (creating grammatically correct statements) that have meaning (semantics)
- More on this as we go...

Arithmetic Expressions

- Mathematical Operators

+ Addition

- Subtraction // Integer division

* Multiplication ** Exponentiation

/ Division % Modulo (remainder)

- Python is like a calculator: type an expression and it tells you the value.

```
>>> 2 + 3 * 5
17
```

Order of Evaluation

Precedence	Operator
Highest	** (exponentiation)
	*, /, //, % Multiplication, division, integer division, and remainder
Lowest	+, - Addition and subtraction

- Use parentheses to force alternate precedence
 $7 + 5 * 6 \neq (7 + 5) * 6$
- Operators that have the same precedence are applied left to right except for exponentiation. Exponentiation is applied right to left.

$$5 * 10 \% 4 = (5 * 10) \% 4$$

$$2 + 3 + 4 = (2 + 3) + 4$$

$$2 ** 3 ** 4 = 2 **(3 ** 4)$$

Data Types

- Integers

4 15110 -53 0

- Floating Point Numbers

4.0 -0.8 03333333333333333333
7.34e+014

- Strings

"hello" "A" " " "" "7up!"
'there' "" '15110'

- Booleans

True False

Integer division

In Python3:

- $7 / 2$ equals **3.5**
- $7 // 2$ equals **3**
- $7 // 2.0$ equals **3.0**
- $7.0 // 2$ equals **3.0**
- $-7 // 2$ equals **-4**
 - beware! `//` rounds **down to smaller number, not towards 0!**

Expressions vs. Statements

- Python evaluates an *expression* to get a *value* (number or other value)
- Python executes a *statement* to perform an action that has an *effect* (e.g., binding a value to a variable, printing something)

Variables

- A variable is *not* an “unknown” as in algebra.
- In Python programming, a variable is a *name* you give a value.
- In Python we give a name to a value using an *assignment statement*:

Assignment
statement

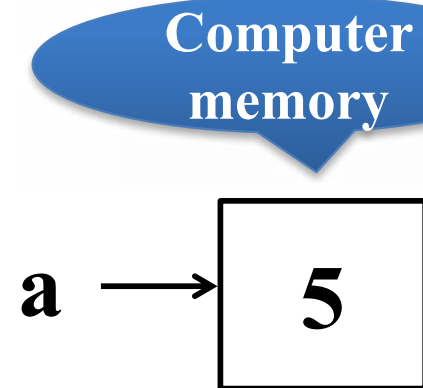
```
>>> a = 5
```

Expression

```
>>> a
```

Python's
response

```
5
```



Variables...

- All variable names must **start with a letter** (lowercase recommended).
- The remainder of the variable name (if any) can consist of any combination of uppercase letters, lowercase letters, digits and underscores (`_`).
- Identifiers in Python are **case sensitive**.
Example: `Value` is different from `value`.

Assignment statements

- In general
 - *variable_name = expression*
- What happens?
 - The expression on the right of the = is evaluated
 - The variable on the left is bound to that value
- Examples
 - $a = 5$ (a is assigned 5)
 - $a = 2 + 5$

Basic output

- Print

```
print("hello")
```

```
print("Mark")
```

- Print multiple items

```
print("hello", "Mark")
```

```
print() # prints a blank line
```

- Print on same line

```
print("hello", end = "")
```

```
print("Mark")
```

Basic input

- Input a string

```
name = input("Enter your name: ")  
print("Your name is:", name)
```

- Input an integer

```
x = input("Enter a number: ")  
print(x, "divided by 2 =", x/2) #Error!
```

- Input an integer correctly with int()

```
x = int(input("Enter a number: "))  
print(x, "divided by 2 =", x/2) #prints as expected
```

Built-in functions

- Math library
 - A predefined module of mathematical values and functions we can use without writing the implementation

- Examples

```
import math
```

```
r = 5 + math.sqrt(2)
```

```
radians = degrees * (math.pi/180)
```

```
print(math.factorial(10))
```

Write your own function

```
def function_name (parameter_list) :  
    □ □ □ □ statements
```

- **def** is a reserved word and cannot be used as a variable name.
- *Indentation is critical.* Use spaces *only*, **not tabs!!!!**

Write your own function (example):

```
def tip(total):  
    return total * 0.18
```

```
>>> tip(100)
```

```
18.0
```

```
>>> tip(135.72)
```

```
24.4296
```

Running Python

- In the shell (at the command line)
- In an IDE (Integrated Development Environment) like IDLE or Pyzo

Program Errors

- Syntax ("compile-time") – Python cannot understand what you have typed
- Runtime – program crashes
- Logical/Semantic – program runs but is incorrect