SAMS
Programming A/B

Lecture #1 – Introductions & Basics
July 2, 2018

Mark Stehlik

Outline for Today

- Overview of Course

- An intro to programming (in Python3) to be continued in lab on Tuesday (group B) and Friday (group A)

The Course Staff

- Me
  - Mark Stehlik <mjs@cs.cmu.edu>
    - available 11:45 – 1:00 and 5:00 – 5:30 (GHC 6205)
- "The other professor"
  - Kelly Rivers <krivers@cs.cmu.edu>, group C
    - Similarly available (GHC 4109)
- TAs
  - 4 undergrads, available in Citadel Commons (Gates 5th floor) from 6:30 – 9 Mon-Thu, 5 – 7 Sat, 1–5 Sun

Course Logistics

- Course website: krivers.net/SAMS-m18/

- 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)
  - Lecture slides will be posted after lecture

- Assignments
  - Posted to course website
  - Handin via autolab (more on this in lab)

You

- Students who want to learn about programming and computer science

- No experience necessary (we will instrument that), not necessarily intending to major in CS (but who knows…)

Course content

An Introduction to Computing (two parts):

- understanding and creating algorithms
- implementing algorithms (writing programs); requires learning about, and practicing with, “the tools”:
  - functions
  - expressions
  - conditionals
  - loops
  - strings
  - lists
  - graphics

Course elements

- Homeworks are due Sunday 5:00pm (you can start working on HW1 as early as today or tomorrow as the first few problems only require basic functions)
- Two "exams"

Collaboration Policy

There are no group assignments in this class

Everyone should read and abide by:

https://www.cmu.edu/policies/student-and-student-life/academic-integrity.html

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…

Data Types

- Integers (int)
  4 0 702 -53

- Floating Point Numbers (float)
  4.0 -0.8 0.3333333333333333
  7.34e+9

- Strings (string)
  "hello" "A" " " "" "7/02"
  'there' 'SAMS'

- Booleans (bool)
  True False

Arithmetic Expressions

- An expression is (an) operand(s) joined by operators
- Mathematical Operators

+	Addition	/	Division (returns a float)
-	Subtraction	//	Integer division
*	Multiplication	%	Modulo (remainder)
**	Exponentiation		

- 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 ≠ (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)

Integer (floor) division

In Python3:

- 7 / 2 equals **3.5**
- 7 // 2 equals **3**
- 7 // 2.0 equals **3.0**
- 7 // 2.5 equals **2.0**
- 7.0 // 2 equals **3.0**
- -7 // 2 equals **-4**
  - beware! // rounds **down to smaller number, not towards 0!**

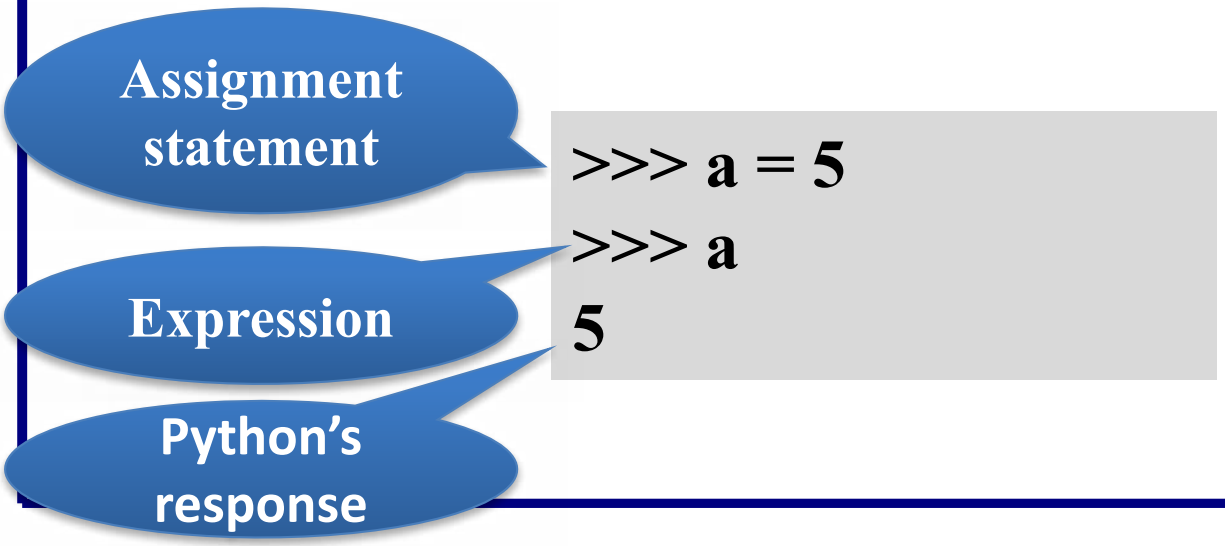
Expressions vs. Statements

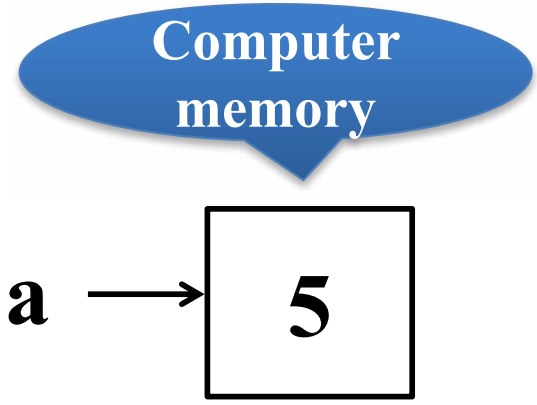
- Python evaluates an *expression* to get a *value* (number or other value)

- Python executes a *statement* to perform an action that has an *effect* (e.g., assigning 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 use to store a value.
- In Python we give a name to a value using an *assignment statement (=):*





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 = 42 is not the same as value = 42

Assignment statements

- In general
  - *variableName* = *expression*
- What happens?
  - The expression on the right of the = is evaluated
  - The variable on the left is assigned that value
- Examples
  - a = 5 (a is assigned 5)
  - a = 2+5 (a is assigned 7, the result of evaluating 2+5)

Basic output

- Printing

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

- Printing multiple items

```
print("hello", "Mark")
print() # prints a blank line
```

- Printing 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
```

Functions

- The building blocks of all programs
- Python provides some for you (built-in functions), for example:
  - abs(*parameter*)
  - float(*parameter*)
  - input(*parameter*)
  - int(*parameter*)
  - print(*parameter[s]*)
  - type(*parameter*)

More "built-in" functions using libraries

- 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 functionName(parameter[s]):
    statements
```

- **def** is a reserved word and cannot be used as a variable name.
- *functionName* follows the rules for variable names
- Indentation is critical. Your editor should automatically indent the next line when you hit <return>
- *functionName(argument[s])* is how it is *called*

Write your own function (example):

```
def tip(total): #defining function
    return total * 0.18


tip(100)  #calling the function
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