More Elements of a Python Program

Professor Hugh C. Lauer

CS-1004 — Introduction to Programming for Non-Majors

(Slides include materials from *Python Programming: An Introduction to Computer Science*, 2nd edition, by John Zelle and copyright notes by Prof. George Heineman of Worcester Polytechnic Institute)

Questions on Homework #1?

- What if you and someone else get different answers with the same input data?
- How do you find the problem?
- If you don't have someone or something to compare it with, …
- … how do you know that your program is producing a correct or useful answer?

Module

- **A separate file with .py extension in name**
- **Contains**
  - Python statements
  - Functions definitions
  - Possibly other stuff
- **Packaged for convenient organization**
  - And sanity on part of programmer
  - And programmer's boss or advisor!
- **Lauer's advice to rising programmers:–**
  - Any program that is more than a couple of pages in length is too long to wrap your head around!
  - Divide it into smaller "modules" organized around related groups of functions

What do you do with a module?

- **Import it**
  - E.g., `import math`
- **Import stuff from it**
  - `from math import sin, pi`
- **"Run" it**
  - `Run >> Run Module` menu command
- **Use it to build more complex systems**
  - Especially by larger groups, organizations
- **Share stuff with other people**
  - `import graphics`
- **…**

More about importing

- **Importing a module "runs" the module**
  - Doing all of the things written in the module — e.g.,
  - Defining functions
  - Creating and setting variables
  - Importing other things

- **`from moduleA import name1, name2, …`**
  - Makes the names "**`name1`**", "**`name2`**", etc., available to importing module…
  - … but *NOT* the name "**`moduleA`**"

- **`import moduleB`**
  - Makes the name "**`moduleB`**" available but not the names inside of moduleB

- **Getting access to stuff inside of moduleB with "dot notation"**
  - **`moduleB.v1`**, **`moduleB.v2`**
  - **`moduleB.funcP()`**, **`moduleB.funcQ()`**

More about importing (continued)

- **`import moduleB as mB`**
  - Creates a shorthand name for "**`moduleB`**"
  - "**`mB`**" can be used in place of "**`moduleB`**" anywhere after the import statement

- **Example —**
  - **`import numpy as np`**
  - **`np.arange(10)`**
    `np.eye(4)`
    `np.diag(np.arange(10))`

- **Widely used**
  - Helps maintain sanity
  - Keeps the typing short!
  - Avoids clashes between similarly named modules

Questions?

Reading Assignment

- ***All* of Chapter 2 of textbook**
- **"Steps in Program Design"**
  - Very important
  - No useful examples in course, yet
  - Important habit to get into
  - A little bit like washing hands before a meal!
- **Names and keywords (i.e., reserved words)**
- **More about `print()` function**
  - Optional parameter at end
  - Specified with "keyword parameter"
- **More about input statements**
  - Need to use **`eval()`** for string to numerical data
- **Interesting extension to assignment**
  - A, B = expression$_1$, expression$_2$

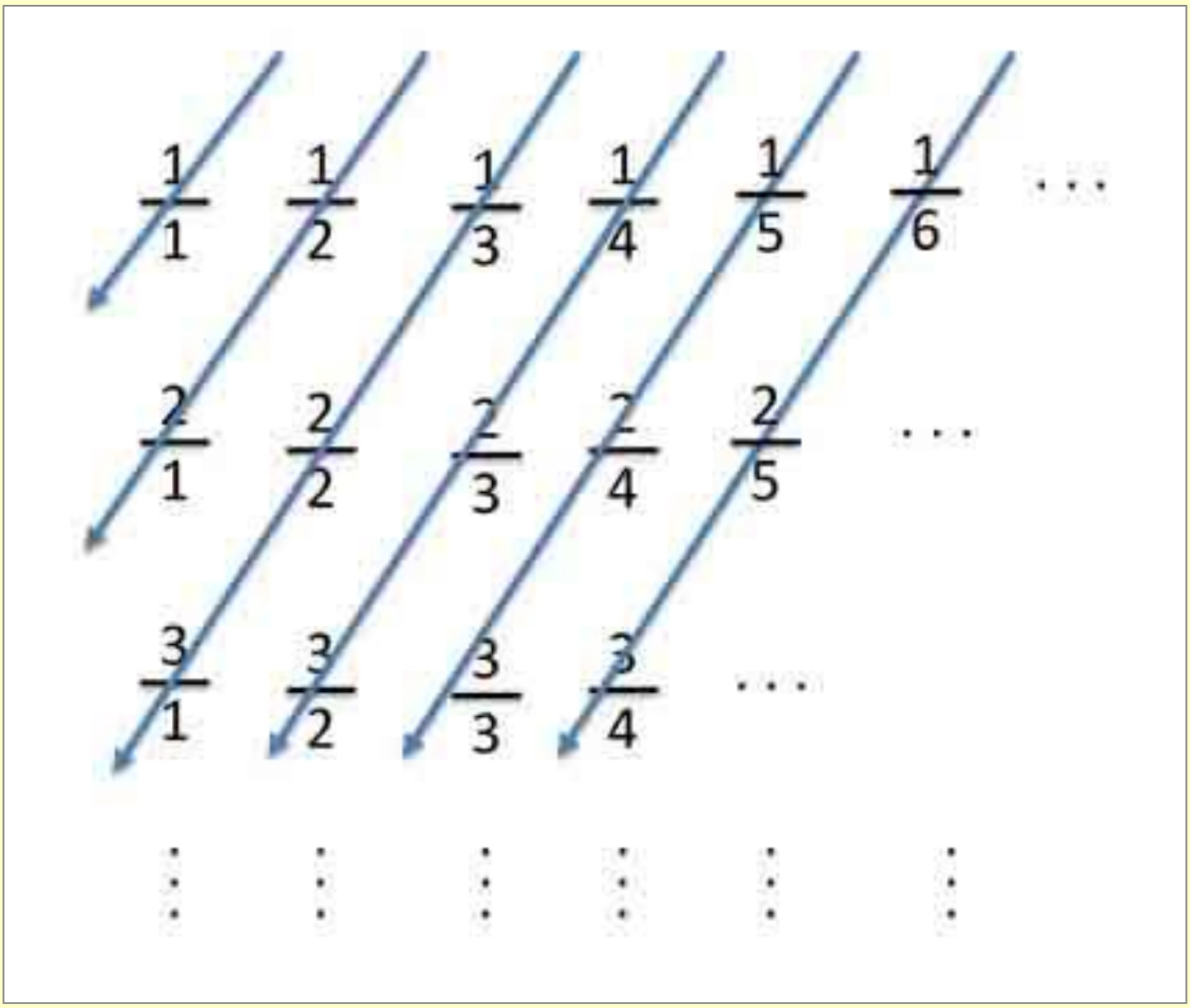
What's in a number?

- **Three types of numbers in real life:–**
  - Integers
  - Rational numbers
  - Irrational numbers ← **Incorrectly called "real numbers"!**

- **Integers**
  - What are they used for?
  - How many?
  - Can you count them?

- **Rational numbers**
  - How defined?
  - What are they used for?
  - How many?
  - Can you count them?

Disgression — Counting the Rationals



What's in a number? (continued)

- **Rational numbers (continued)**
  - …
  - What about decimal numbers?
  - What about scientific notation?
- **Irrational numbers?**
  - How to describe them?
  - Can you give examples?
  - How many are there?
  - Can you count them?

Integers in *Python*

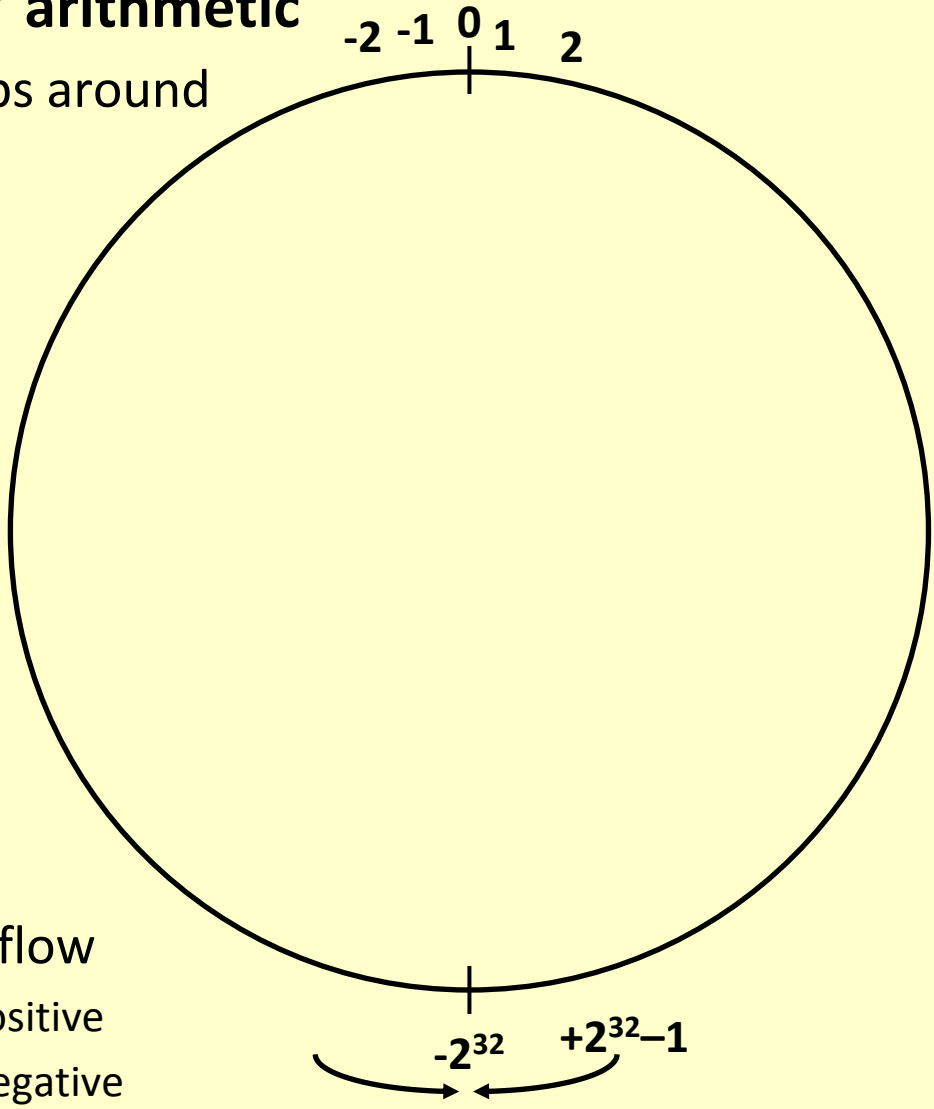
- ***Python 3* gets integer arithmetic right!**

- **Examples:–**
  - 30,000 × 30,000 = 900,000,000
  - 40,000 × 40,000 = 1,600,000,000
  - 50,000 × 50,000 = 2,500,000,000
  - …

- **Very big numbers:–**
  - factorial(100)
  - Factorial(1000)

Integers in Other Computers and Languages

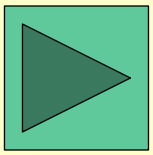
- ***All* hardware and most other languages use finite precision (including *Python* 2.7)**
  - Usually 32- or 64 bits (these days)
- **"Clock" arithmetic**
  - Wraps around



  - Overflow
    - Positive
    - Negative

Numbers in Computers — Integers (continued)

- **Most other languages use finite precision**
  - Usually 32- or 64 bits (these days)
- **"Clock" arithmetic**
  - Wraps around
- **Nasty examples:–**
  - 30,000 × 30,000 = 900,000,000
  - 40,000 × 40,000 = 1,600,000,000
  - 50,000 × 50,000 = -352,516,352 (!!)



- **One week of CS-2011 devoted to this topic!**
- **Nearly every other computer language uses internal representation of integers**
  - Subject to this problem

How does *Python* do it?

- **Uses internal arithmetic by default**
- **Checks for overflow on *every* operation**
  - If overflow, fall back to "long-hand" arithmetic

What about other languages?

- **Need to manually import a *BigNum* package**
- **Call arithmetic functions for *every* operation**

Don’t need to worry about integer precision for *this course*

But you may need to worry about it in other systems and languages!

Numbers in Computers — Floating Point

- **Used to *approximate* rationals and irrationals**
- **A little bit like scientific notation, but in binary**
  - Fraction part f:– 1.0 <= f < 2.0
  - Exponent part: 2^e, where *e* can be positive or negative
  - Different levels of precision
    - 32-bit, 64-bit, 80-bit, 128-bit, etc.
- **A lot of mathematical theory behind**
  - Mathematical calculations
  - Rounding off results
- **IEEE standard**
  - So that we get same answers on different computers!
- ***Python* uses IEEE standard and internal floating point arithmetic hardware and representation**

Numbers in Computers — Floating Point (continued)

- **Bad news:–**
  - Minor rounding errors in most trivial places
- **Good news:–**
  - Same answers to same expression on all computers!

Questions?

Numbers and numerical issues are discussed in Chapter 3

Data types in *Python* Programs

- Integers
- Floats
- Strings
- Lists
- <other stuff deferred to later>

String

- **Any sequence of printable characters enclosed in matching quotes**
  - In any language!
- **Quotes may be single or double**
  - But must match
- **Quote mark in string may *not* match the quotes delineating string**
- **More later in course**

Variables

- **Every variable in *Python* "knows" its own type**
- **May change as a result of subsequent assignments**
  - Examples
- **Unlike other programming languages**
  - Type of a variable is fixed at time program is written!

Questions?

Lists

- **A collection of values enclosed in square brackets, separated by commas**
- **[1, 2, 3, 5, 7, 11, 13, 17, 19, 23]**
- **May be all the same type**
  - Examples
- **May be used as control of for-loop**
  - for i in [1, 2, 3, 5, 7, 11, 13, 17, 19, 23]:
- **May be assigned to variables**
- **May be passed as arguments to functions**
  - And results!
- **Needed for HW #2**

Adding something to a list

- `listObject.append(value)`
- **Creates a new list element at end of list**
- **Assigns *value* to that list element**
  - Just like a regular assignment to a variable
- **In effect, a *list* is really a sequence of variables**
  - Of no fixed length!
  - Treated as one object
- **More about lists later in the course**
  - Much more!

Questions?

For-Loop

- What does a for-loop look like? (Lab #1)
- How would you explain it to a friend not yet in this course?

```
for var in <something>:
   body statement1
   body statement2
   …
   body statementk
```

This is a new *variable* name!

y not be
rently in use

Each continuation line is indented one "unit"— i.e., tab

End of for-loop denoted by reve
to previous indentation level

For use only in loop body

For-Loop

- **What does a for-loop look like? (Lab #1)**
- **How would you explain it to a friend not yet in this course?**

```
for var in <something>:
   body statement1
   body statement2
   …
   body statementk
```

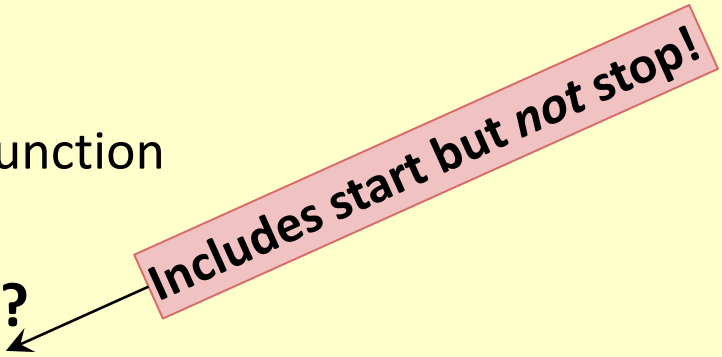
- **Meaning:–**
  - Go thru (i.e., enumerate) **`<something>`**
  - For each item in enumeration …
  - … assign **`var = that_item`**
  - … execute the body statements using **`var`**
  - Repeat with next item of enumeration, etc.
  - Loop stops when enumeration is exhausted

What can we enumerate?

- **More or less anything!**
- **For now, we will enumerate integers:–**
  - E.g., range(10)
- **Meaning:–**
  - Each time around loop, call **`range()`** to emit the next value
  - Stop when **`range()`** has emitted all that it is going to emit!
- **`range()` is a special kind of *Python* function …**
  - … called a *generator!*
  - Remembers what it did last
  - Each time, it returns the next item
  - …
  - … until the end, when it emits a special code to tell loop to stop

For-Loop (continued)

- **Explain `range(10)`**
  - i.e., what numbers are generated?
- **Can we see a "range"?**
  - Yes, use the `list()` function
- **Another form of range?**
  - `range(start, stop)`
    `range(start, stop, step)`



Reading: Chapter 3!

Homework #2

Numerical calculations using for-loops and lists

- **Generate several lists representing points on *x*-axis**
- **Calculate *cosine* function from first principles …**
- **… i.e., by summing the first *n* terms of an infinite series!**

$$myCos(x) \cong 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \ldots$$

- Plot using *pyplot* (from *matplotlib*)

Questions?