CS1110

Lecture 2: Variables; Strings

Announcements

Everything is on the website!

- Piazza instructions
- Class announcements
- Consultant calendar
- Reading schedule
- Lecture slides
- Exam dates

Check it regularly:
`www.cs.cornell.edu/courses/cs1110/2014sp/`

Lab Starts This Week

Attend the lab you are registered for! This is important; there is no extra space.

If you can register for a less full section, please do—you will have more fun and leave space for someone who can't.

If you are having trouble fitting into a lab you can attend, contact the course staff right away.

Assignments

- Major portion (~50%) of your final grade
  - Due roughly every two weeks; longer ones count for more
- First assignment requires **mastery**
  - Submit, get feedback, resubmit, … until correct
  - Everyone eventually scores 10/10 (**only** for first assignment)
- Later assignments are designed to be fun
  - Examples: graphics, image manipulation
  - Final project is a Breakout game project
- Submitted via **Course Management System** (CMS)
  - Visit **cms.csuglab.cornell.edu** to check you are enrolled

CS1110 Grading Scheme

- See website for complete grading scheme
  - In particular, a separate scheme for S/U
- iClickers participation = 1–2%
  - In lecture questions
  - Essentially a form of "stealth attendance"
  - Must answer 75% of questions for credit
  - But actual answers are not graded
  - We won't start counting for a few lectures, to give you time to get a hold of a clicker.

iClickers

- Have you registered your iclicker?
- If not, visit
  - atcsupport.cit.cornell.edu/pollsrvc/
- Instructions on iclickers can be found here:
  - www.it.cornell.edu/services/polling/howto-students.cfm
- Find these links on the course webpage
  - Click "Texts/iClickers"
  - Look under "iClickers"

Helping You Succeed: Other Resources

- **Consultants.** ACCEL Lab Green Room
  - Daily office hours (see website) with consultants
  - Very useful when working on assignments
- **Piazza.** Online forum to ask and answer questions
  - Go here first **before** sending question in e-mail
- **Office Hours.** Talk to the professors and TAs!
  - Professors: in 128 Olin Hall between lectures
  - TAs: see staff page on the website
- **AEW Workshops.** Additional discussion course
  - Runs parallel to this class—completely optional
  - See website; talk to advisors in Olin 167

Warm-Up: Using Python

- How do you plan to use Python?

A. I want to work mainly in the ACCEL lab
B. I want to use my own Windows computer
C. I want to use my own Macintosh computer
D. I want to use my own Linux computer
E. I will use whatever I can get my hands on

Type: Set of values and the operations on them

- Type **int**:
  - **Values**: integers
  - **Ops**: +, –, *, /, %, **, …
- Type **float**:
  - **Values**: real numbers
  - **Ops**: +, –, *, /, **, …
- Type **bool**:
  - **Values**: **True** and **False**
  - **Ops**: not, and, or
- Type **str**:
  - **Values**: string literals
    - Double quotes: `"abc"`
    - Single quotes: `'abc'`
  - **Ops**: + (concatenation)

Will see more types in a few weeks

Operator Precedence

- What is the difference between the following?
  - 2*(1+3) **add, then multiply**
  - 2*1 + 3 **multiply, then add**
- Operations are performed in a set order
  - Parentheses make the order explicit
  - What happens when there are no parentheses?
- **Operator Precedence**: The *fixed* order Python processes operators in *absence* of parentheses

Precedence of Python Operators

- **Exponentiation**: `**`
- **Unary operators**: `+ -`
- **Binary arithmetic**: `* / %`
- **Binary arithmetic**: `+ -`
- **Comparisons**: `< > <= >=`
- **Equality relations**: `== !=`
- **Logical not**
- **Logical and**
- **Logical or**

- Precedence goes downwards
  - Parentheses highest
  - Logical ops lowest
- Same line = same precedence
  - Read "ties" left to right (for all but `**`)
  - Example: `1/2*3` is `(1/2)*3`

- Section 2.7 in your text
- See website for more info
- Major portion of Lab 1

Variables (Section 2.1)

- A **variable**
  - is a **named** memory location (**box**)
  - contains a **value** (in the box)
  - can be used in expressions

The value in the box is then used in evaluating the expression.

- Examples:

The type belongs to the *value*, not to the *variable*.

Variable names must start with a letter (or _).

x	5	Variable **x**, with value 5 (of type **int**)
area	20.1	Variable **area**, w/ value 20.1 (of type **float**)

Variables and Assignment Statements

- Variables are created by **assignment statements**

  Create a new variable name and give it a value

  "gets"

  the value

  `x = 5`

  the variable

  x | 5

- This is a **statement**, not an **expression**
  - Tells the computer to DO something (not give a value)
  - Typing it into >>> gets no response (but it is working)
- Assignment statements can have expressions in them
  - These expressions can even have variables in them

  the expression

  `x = x + 2`

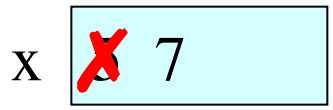
  the variable

Two steps to execute an assignment:
1. evaluate the expression on the right
2. store the result in the variable on the left

Execute the statement: `x = x + 2`

A: I did it correctly!
B: I drew another box named x
C: I did something else
D: I did nothing—just watched

- Draw variable x on piece of pa



- Step 1: evaluate the expression `x + 2`
  - For x, use the value in variable x
  - Write the expression somewhere on your paper
- Step 2: Store the value of the expression in `x`
  - Cross off the old value in the box
  - Write the new value in the box for x
- Check to see whether you did the same thing as your neighbor, discuss it if you did something different.

Execute the statement: `x = 3. * x + 1.`

A: I did it correctly!
B: I drew another box named x
C: I did something else
D: I did nothing –just watched

- You have this:

  x ✗ ✗ 22.

- Execute this command:
  - Step 1: **Evaluate** the expression `3. * x + 1.`
  - Step 2: **Store** its value in x
- Check to see whether you did the same thing as your neighbor, discuss it if you did something different.

Execute the statement: `x = 3. * x + 1.`

- You now have this:

  x ~~3.~~ ~~7.~~ 22.

- The command:
  - Step 1: **Evaluate** the expression `3. * x + 1.`
  - Step 2: **Store** its value in x

- This is how you execute an assignment statement
  - Performing it is called **executing the command**
  - Command requires both **evaluate** AND **store** to be correct
  - Important *mental model* for understanding Python

Exercise: Understanding Assignment

- Add another variable, interestRate, to get this:

  x ~~x~~ ~~x~~ 22. interestRate ~~x~~ 5.5

- Execute this assignment:

```
interestRate = x / interestRate
```

- Check to see whether you did the same thing as your neighbor, discuss it if you did something different.

A: I did it correctly!
B: I drew another box called "interestRate"
C: I stored the value in the box for x
D: I thought it would use **int** division
E: I did something else (or nothing)

Exercise: Understanding Assignment

- You now have this:

x	✗ ✗ 22.	interestRate	✗ 5.5	intrestRate	27.5

- Execute this assignment:

```
intrestRate = x + interestRate
```

- Check to see whether you did the same thing as your neighbor, discuss it if you did something different.

Spelling mistakes in Python are bad!!

A: I did it correctly!
B: I stored the value in "interestRate"
C: I stored the value in x
D: I did something else (or nothing)

Dynamic Typing

- Python is a **dynamically typed language**
  - Variables can hold values of any type
  - Variables can hold different types at different times
  - Use `type(x)` to find out the type of the value in x
  - Use names of types for conversion, comparison

```
type(x) == int
x = float(x)
type(x) == float
```

- The following is acceptable in Python:

```
>>> x = 1
```
← x contains an **int** value

```
>>> x = x / 2.0
```
← x now contains a **float** value

- Alternative is a **statically typed language** (e.g. Java)
  - Each variable restricted to values of just one type

Dynamic Typing

- Often want to track the type in a variable
  - What is the result of evaluating x / y?
  - Depends on whether x, y are **int** or **float** values
- Use expression `type(<expression>)` to get type
  - `type(2)` evaluates to `<type 'int'>`
  - `type(x)` evaluates to type of contents of x
- Can use in a boolean expression to test type
  - `type('abc') == str` evaluates to **True**

String: Text as a Value

- String are quoted characters
  - 'abc d' (Python prefers)
  - "abc d" (most languages)
- How to write quotes in quotes?
  - Delineate with "other quote"
  - **Example**: " ' " or ' " '
  - What if need both " and ' ?
- **Solution**: escape characters
  - Format: \ + letter
  - Special or invisible chars

Type: `str`

Char	Meaning
\'	single quote
\"	double quote
\n	new line
\t	tab
\\	backslash

String are Indexed

- s = 'abc d'

0	1	2	3	4
a	b	c		d

- Access characters with []
  - s[0] is 'a'
  - s[4] is 'd'
  - s[5] causes an error
  - s[0:2] is 'ab' (excludes c)
  - s[2:] is 'c d'
- Called "string slicing"

- s = 'Hello all'

0	1	2	3	4	5	6	7	8
H	e	l	l	o		a	l	l

- What is s[3:6]?

A: 'lo a'
B: 'lo'
C: 'lo '
D: 'o '
E: I do not know

String are Indexed

- s = 'abc d'

0	1	2	3	4
a	b	c		d

- Access characters with []
  - s[0] is 'a'
  - s[4] is 'd'
  - s[5] causes an error
  - s[0:2] is 'ab' (excludes c)
  - s[2:] is 'c d'
- Called "string slicing"

- s = 'Hello all'

0	1	2	3	4	5	6	7	8
H	e	l	l	o		a	l	l

- What is s[3:6]?

A: 'lo a'
B: 'lo'
C: 'lo ' **CORRECT**
D: 'o '
E: I do not know

String are Indexed

- s = 'abc d'

0	1	2	3	4
a	b	c		d

- Access characters with []
  - s[0] is 'a'
  - s[4] is 'd'
  - s[5] causes an error
  - s[0:2] is 'ab' (excludes c)
  - s[2:] is 'c d'
- Called "string slicing"

- s = 'Hello all'

0	1	2	3	4	5	6	7	8
H	e	l	l	o		a	l	l

- What is s[:4]?

A: 'o all'
B: 'Hello'
C: 'Hell'
D: Error!
E: I do not know

String are Indexed

- s = 'abc d'

0	1	2	3	4
a	b	c		d

- Access characters with []
  - s[0] is 'a'
  - s[4] is 'd'
  - s[5] causes an error
  - s[0:2] is 'ab' (excludes c)
  - s[2:] is 'c d'
- Called "string slicing"

- s = 'Hello all'

0	1	2	3	4	5	6	7	8
H	e	l	l	o		a	l	l

- What is s[:4]?

A: 'o all'
B: 'Hello'
C: 'Hell' **CORRECT**
D: Error!
E: I do not know

Strings have many other powers

```
s = 'abracadabra'
'a' in s == True
'cad' in s == True
'foo' in s == False
s.index('a') == 0
s.index('rac') == 2
s.count('a') == 5
len(s) == 11
s.strip('a') == 'bracadabr'
' cs1110 '.strip() == 'cs1110'
```

s_1 in s_2 asks whether s_1 is a substring of s_2. Result is type **bool**.

s_1.index(s_2) returns the index of the first occurrence of s_2 in s_1.

s_1.count(s_2) returns the number of occurrences of s_2 in s_1.

len(s) returns the number of characters in s.

s_1.strip(s_2) returns a copy of s_1 with characters in s_2 removed from the ends.

Just s_1.strip() defaults to removing white space from the ends.

More (too much!) information in Python documentation on **www.python.org** (see Library Reference, built-in types)

(all these boolean expressions evaluate to True)