CS1110

Lecture 2: Variables; Strings

Announcements

Problem emails

(as of Sunday)

disabled/discontinued/not found:

jason.luu719@yahoo.com

jamiechowsl@gmail.com

mailbox full and can't accept messages:

xh89@cornell.edu

ars279@cornell.edu

Added late & missed lab?

Download the lab handout from the course website and complete it on your own this week.

Then, bring it to next week's lab and ask a TA to check it in.

Catch up on lectures using VideoNote: see course website.

Assignments

- Major portion (40%) of your final grade
  - Larger projects due every two weeks
- First assignment requires **mastery**
  - Submit, get feedback, resubmit, … until correct
  - Everyone eventually scores 10/10
- 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

Participation: 2% of Final Grade

- **iClickers.** In lecture questions
    - Essentially a form of “stealth attendance”
    - Must answer 75% of questions for credit
    - But actual answers are not graded
- **Surveys.** What do you think of the class?
    - This is the first year teaching Python
    - Want data on who you are/why taking course?
    - What do you like/dislike about assignments?
    - Must answer 75% of surveys for full credit

Things to Do Before Next Class

1. Register your iClicker
   - Does not count for grade if not registered
2. Enroll in Piazza
3. Sign into CMS
   - ~~Quiz: About the Course~~
   - ~~Complete Survey 0~~
4. Read the textbook
   - Chapter 1 (browse)
   - Chapter 2 (in detail)

- Everything is on website!
  - Piazza instructions
  - Class announcements
  - Consultant calendar
  - Reading schedule
  - Lecture slides
  - Exam dates
- Check it regularly:
  - www.cs.cornell.edu/courses/cs1110/2013sp/

Helping You Succeed: Other Resources

- **Consultants.** ACCEL Lab Green Room
  - Daily office hours (see website) with consultants
  - Very useful when working on assignments
- **AEW Workshops**. Additional discussion course
  - Runs parallel to this class – completely optional
  - See website; talk to advisors in Olin 167.
- **Piazza.** Online forum to ask and answer questions
  - Go here first **before** sending question in e-mail
- **Office Hours.** Talk to the professors!
  - Available in Thurston 102 between lectures

iClickers

- Have you registered your iclicker?
- If not, visit
  - atcsupport.cit.cornell.edu/pollsrvc/
- Instructions on iclickers can be found here:
  - atc.cit.cornell.edu/course/polling/clickers.cfm
- Find these links on the course webpage
  - Click “Texts”
  - Scroll down on the page that opens.

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`

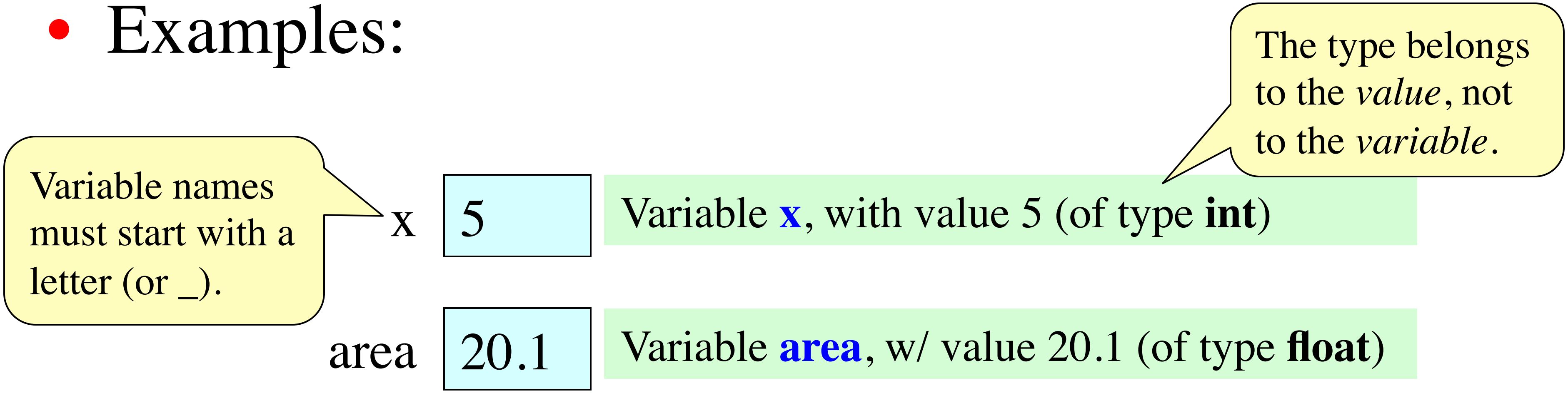
- Section 2.7 in your text
- See website for more info
- Major portion of Lab 1

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:



Variable **x**, with value 5 (of type **int**)

Variable **area**, w/ value 20.1 (of type **float**)

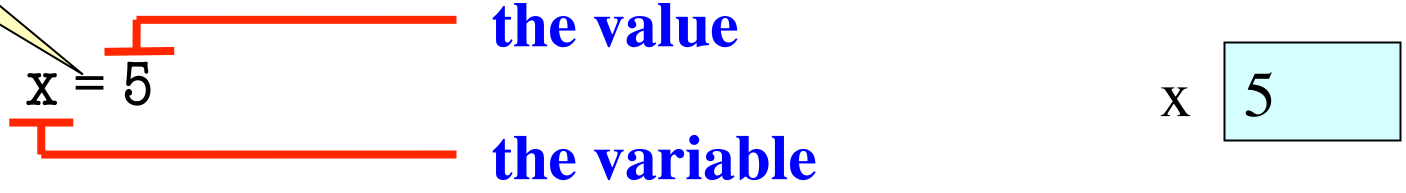
Variable names must start with a letter (or _).

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

Variables and Assignment Statements

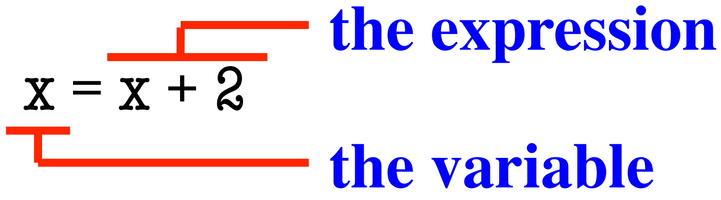
- Variables are created by **assignment statements**
  - Create a new variable name and give it a value

“gets”



```
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



```
x = x + 2
```

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

x ~~5~~ 7

- 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 ~~3~~ ~~7~~ 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 ~~5~~ ~~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 ~~5~~ ~~7~~ 22.   interestRate ~~4~~ 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 ~~5~~ ~~7~~ 22. interestRate ~~4~~ 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

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.

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

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

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)