PYTHON I/O AND EXCEPTIONS

Curt Clifton
Rose-Hulman Institute of Technology

TODAY’S PLAN

- *dir()*
- Strings in Python
- *input* and *raw_input*
- File I/O, *pickle*
- Exception Handling
- Milestone 1 overview

WHAT'S *DIR* FOR?

- Gives a sorted list of the names defined in a module
- Examples to try:

  -
    ```
    >>> import sys
    >>> dir(sys)
    >>> dir()
    >>> dir(__builtins__)
    ```

two underbars each

SOME STRING FUNCTIONS

- *s = ‘Hello’*
- *s.capitalize()*
- *s.center(30, ‘X’)*
- *s.index(‘lo’)*
- *s.ljust(20)*, also *rjust*
- *s.lower()*
- *s.replace(‘ello’, ‘i’)*
- *‘a,b,c’.split(‘,’)*
- *s.startswith(‘H’)*
- *s.strip()*, also *lstrip*, *rstrip*
- Try: *help(str)*

Q2

STRING FORMATTING

- % operator on strings → **deprecated**
- Use *format* method on *strings*:
  - "{0:4d} {1:4d}".format(42*2, 42**2) → ' 84 1764'
  - "{**1**:5d} {**0**:5d}".format(42*2, 42**2) → ' 1764   84'
  - "{0:**5.2f**} {1} {2}".format(sqrt(42), 'sheep', 'plummet') → ' 6.48 sheep plummet'

NAMED FORMAT ARGUMENTS

- *"X Coord.: {**x**:.2f},Y Coord.: {**y**:.2f}".format(**x=3.145, y=2.71**)*
  → *'X Coord.: 3.15,Y Coord.: 2.71'*

- ***d={"phone":8793, 'fax':6060}***
  *"Phone {0**[phone]**} or fax {0**[fax]**}".format(d)*
  → *'Phone 8793 or fax 6060'*

- *x, y = 3.2, 5.4*
  *"{**x**:5.2f}{**y**:5.2f}".format(****vars()**)*
  → *' 3.20 5.40'*

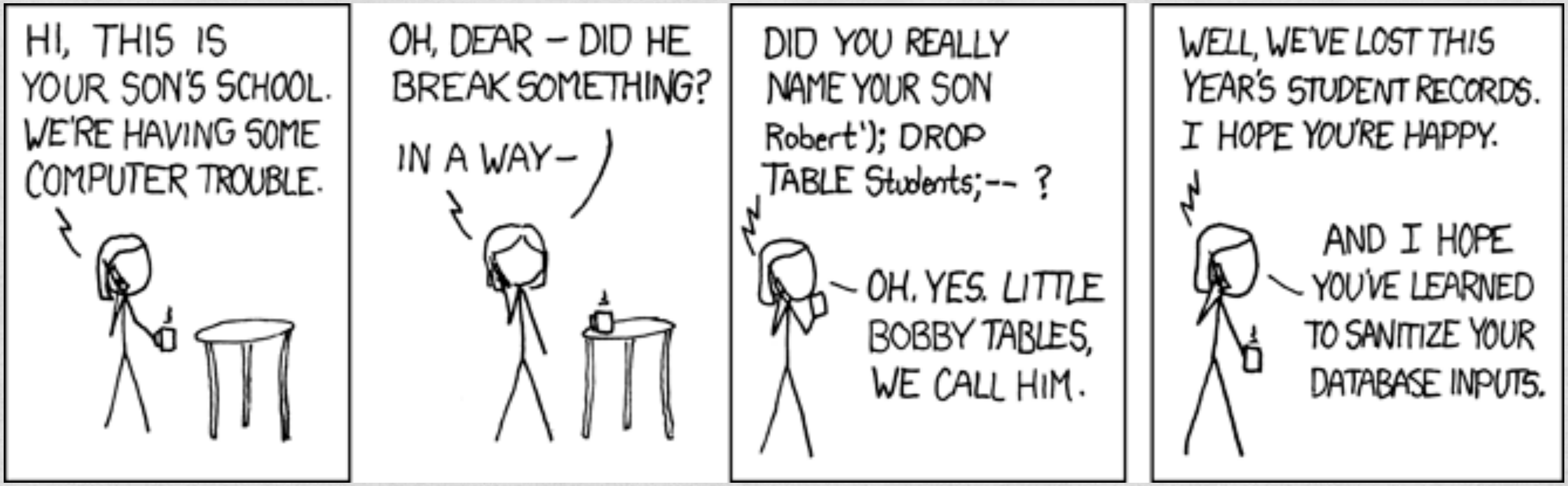
- What does *vars()* return?

Q4

INPUT AND *RAW_INPUT*

- *input(prompt)*
  - Displays prompt, accepts console input, returns it as a string
- *raw_input(prompt)* is **gone** in Python 3

SPEAKING OF INPUTS


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

http://xkcd.com/327/

Her daughter is named
Help I'm trapped in a driver's license factory.

FILE I/O

- Opening: *f* = *open(file_path, mode)*
  - *file_path* is the path to the file (duh!)
  - *mode* is the access mode: *'r', 'w', 'a', 'r+', 'rb', 'wb'*
- Writing: *f.write('String to write')*
- Closing: *f.close()*

reading and writing

READING FROM AN OPEN FILE

- *f.read()*, returns entire contents of file
- *f.readline()*, returns next line of file
- *f.readlines()*, returns entire contents as a list of strings
- Often better to iterate over file:
    - *for line in f:*
      # do something with line

FILE I/O WITH *WITH*

- Files (and others) can clean up after themselves
- Example:

```
with open("myfile.txt", 'r') as f:
    for line in f:
        # do something with line
```

- *with* statement automatically closes file

GETTING PICKLED

- The *pickle* module converts objects to/from streams
  - *pickle.dump(obj, file)*
  - *obj = pickle.load(file)*
- What can be pickled? (partial list)
  - *None*, *True*, *False*, numbers, and strings
  - tuples, lists, sets, dictionaries of picklable things

Note: File must be opened in binary mode

EXCEPTION HANDLING

```
try:
    # Code that might raise an exception
except ExceptionType [as var]:
    # Handles ExceptionType
except OtherExceptionType [as var]:
    # Handles OtherExceptionType
except:
    # Handles any other exceptions
else:
    # Runs if no exceptions
finally:
    # Runs no matter what happened above
```

Generally frowned upon

EXCEPTION ARGUMENTS

```
try:
    raise Exception('spam','eggs')
except Exception as inst:
    print(type(inst))
    print(inst.args)
    print(inst)
    x, y = inst.args
    print('x = {}, y = {}'.format(x,y))
```

<class 'Exception'>

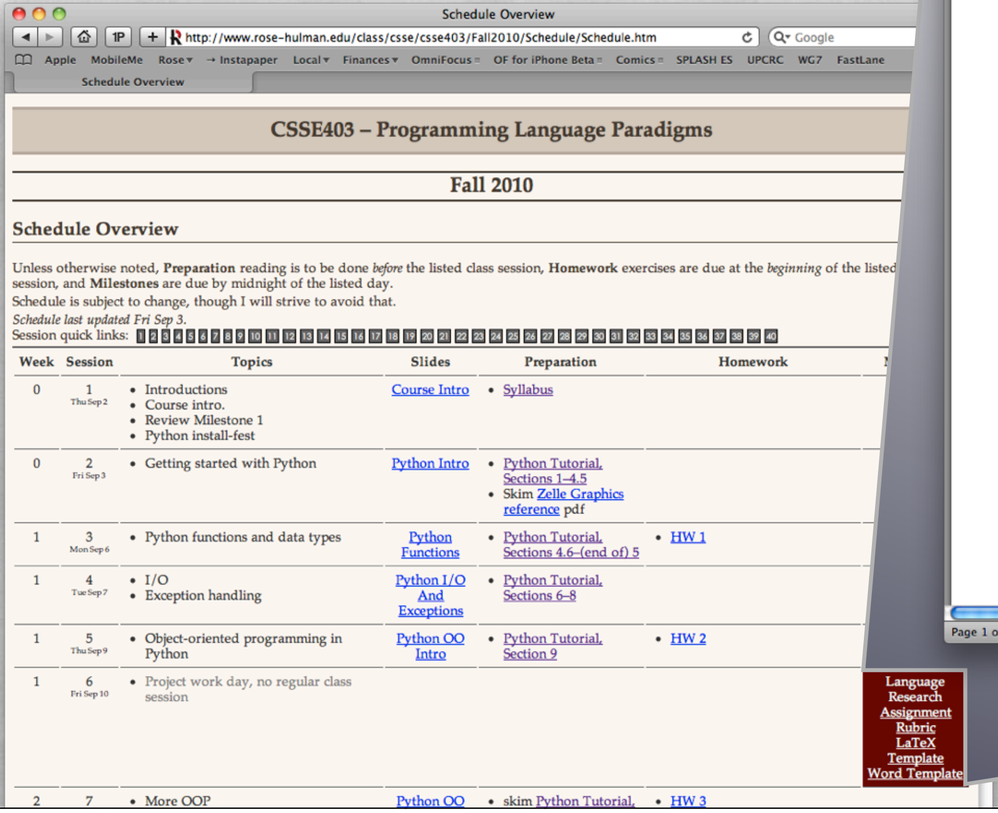
('spam', 'eggs')

('spam', 'eggs')

x = spam, y = eggs

See docs for defining your own exception types

MILESTONE 1


Schedule Overview
http://www.rose-hulman.edu/class/csse/csse403/Fall2010/Schedule/Schedule.htm
Google
Apple MobileMe Rose Instapaper Local Finances OmniFocus OF for iPhone Beta Comics SPLASH ES UPCRC WG7 FastLane
Schedule Overview

CSSE403 – Programming Language Paradigms

Fall 2010

Schedule Overview

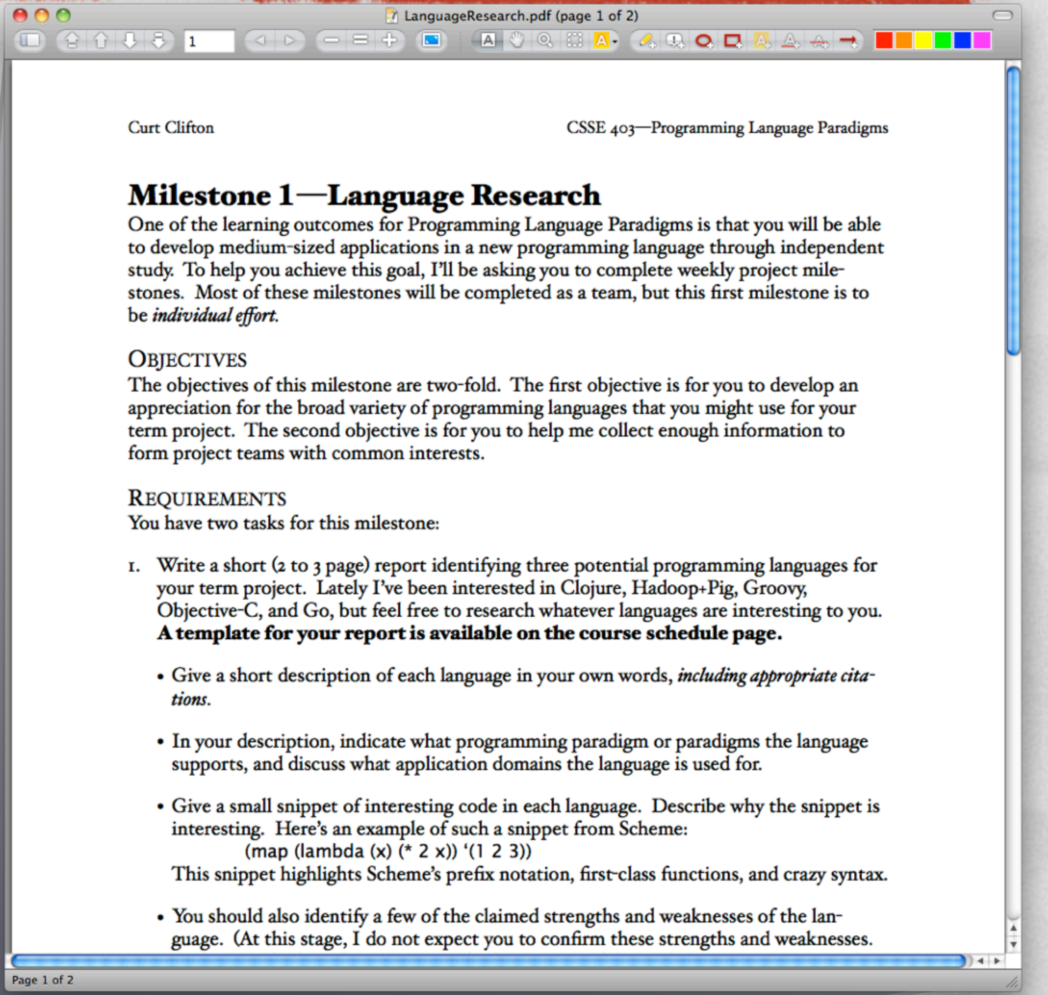
Unless otherwise noted, **Preparation** reading is to be done *before* the listed class session, **Homework** exercises are due at the *beginning* of the listed session, and **Milestones** are due by midnight of the listed day.

Schedule is subject to change, though I will strive to avoid that.

Schedule last updated Fri Sep 3.

Session quick links: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

Week	Session	Topics	Slides	Preparation	Homework	
0	1 Thu Sep 2	• Introductions • Course intro. • Review Milestone 1 • Python install-fest	Course Intro	• Syllabus		
0	2 Fri Sep 3	• Getting started with Python	Python Intro	• Python Tutorial, Sections 1–4.5 • Skim Zelle Graphics reference pdf		
1	3 Mon Sep 6	• Python functions and data types	Python Functions	• Python Tutorial, Sections 4.6–(end of) 5	• HW 1	
1	4 Tue Sep 7	• I/O • Exception handling	Python I/O And Exceptions	• Python Tutorial, Sections 6–8		
1	5 Thu Sep 9	• Object-oriented programming in Python	Python OO Intro	• Python Tutorial, Section 9	• HW 2	
1	6 Fri Sep 10	• Project work day, no regular class session				Language Research Assignment Rubric LaTeX Template Word Template
2	7	• More OOP	Python OO	• skim Python Tutorial,	• HW 3	



Curt Clifton CSSE 403—Programming Language Paradigms

Milestone 1—Language Research

One of the learning outcomes for Programming Language Paradigms is that you will be able to develop medium-sized applications in a new programming language through independent study. To help you achieve this goal, I'll be asking you to complete weekly project milestones. Most of these milestones will be completed as a team, but this first milestone is to be *individual effort*.

OBJECTIVES

The objectives of this milestone are two-fold. The first objective is for you to develop an appreciation for the broad variety of programming languages that you might use for your term project. The second objective is for you to help me collect enough information to form project teams with common interests.

REQUIREMENTS

You have two tasks for this milestone:

1. Write a short (2 to 3 page) report identifying three potential programming languages for your term project. Lately I've been interested in Clojure, Hadoop+Pig, Groovy, Objective-C, and Go, but feel free to research whatever languages are interesting to you. **A template for your report is available on the course schedule page.**
   - Give a short description of each language in your own words, *including appropriate citations.*
   - In your description, indicate what programming paradigm or paradigms the language supports, and discuss what application domains the language is used for.
   - Give a small snippet of interesting code in each language. Describe why the snippet is interesting. Here's an example of such a snippet from Scheme:

     ```
     (map (lambda (x) (* 2 x)) '(1 2 3))
     ```

     This snippet highlights Scheme's prefix notation, first-class functions, and crazy syntax.
   - You should also identify a few of the claimed strengths and weaknesses of the language. (At this stage, I do not expect you to confirm these strengths and weaknesses.