PYTHON I/O AND EXCEPTIONS

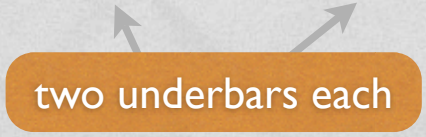
Curt Clifton
Rose-Hulman Institute of Technology

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__)



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(__builtins__.str)*

two underbars each

STRING FORMATTING

- The % operator with a string as the first argument generates formatted strings
- Examples:
  - *"%4d %4d" % (42*2, 42**2)* → *' 84 1764'*
  - *"%5.2f %s %s" % (sqrt(42), 'sheep', 'plummet')* → *' 6.48 sheep plummet'*
- Most format specifiers work like C

NAMED FORMAT ARGUMENTS

- Can use names within format specifiers if right-hand argument is a dictionary!
- Example:
  - *x, y = 3.2, 5.4*
    print "%5.3f, %5.3f" % (x, y)
    print "%(x)5.3f, %(y)5.3f" % vars()

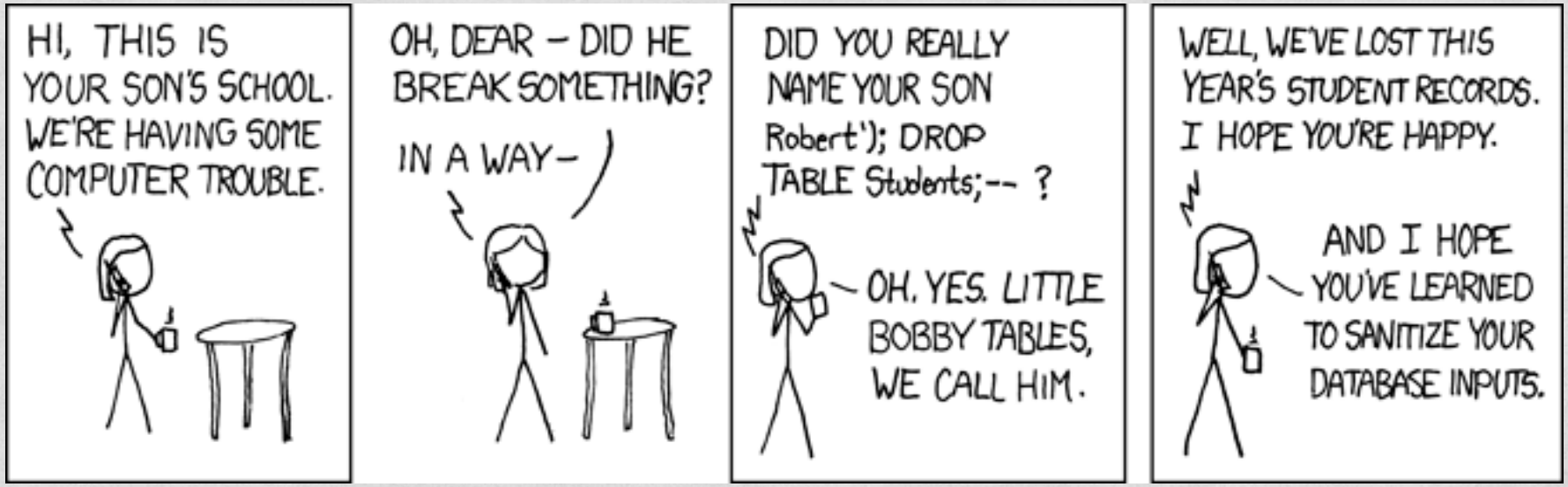
Equivalent

- What does *vars()* return?

INPUT AND *RAW_INPUT*

- *raw_input(prompt)*
  - Displays prompt, accepts console input, returns it as a string
- *input(prompt)*
  - Equivalent to *eval(raw_input(prompt))*
  - Essentially gives back what Python would if you typed at the interpreter prompt

SPEAKING OF INPUTS



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:
  - *for line in f:*
    # do something with line

FILE I/O WITH *WITH*

- Files (and other objects) in Python 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
- To use in Python 2.5:

```
from __future__ import with_statement
```

GETTING PICKLED

- The *pickle* module is used to convert objects to streams and back
  - *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 [, optArg]:
    # Handles ExceptionType
except OtherExceptionType [, optArg]:
    # Handles OtherExceptionType
except:
    # Handles any other exceptions
else:
    # Runs if no exceptions
finally:
    # Runs no matter what happened above
```

Generally frowned upon

MILESTONE 1

- Take a few minutes to review milestone description
  - See handout
  - Also linked from schedule
- Questions?