# Thinking Like a Programmer in Python

GirlDevelopIt / PhillyPUG
April 24, 2016

instructor: matt wartell
organizers: sarah gray, maneesha sane
assistants: kimberly fekete, sharon warner




This presentation, supporting materials, and license can be found at
https://github.com/mwartell/gdi-ct

# Introduction

- Small data processing and analysis
- How to approach a task
  - Exploration
  - Problem decomposition
  - Result synthesis
- How to make mistakes
- How to write better code

# The Ice Cream Preference Data

| Respondent | Gender | Preference |
|---|---|---|
| 1011 | f | p |
| 1085 | m | s |
| 1099 | m | s |
| 1149 | f | s |
| 1221 | m | p |
| 1279 | f | s |
| 1378 | m | p |
| 1396 | f | s |
| 1420 | m | s |
| 1435 | f | s |
| … | … | … |

https://github.com/mwartell/gdi-ct

# The Data File

```
respondent,gender,preference
1011,f,p
1085,m,s
1099,m,s
1149,f,s
1221,m,p
1279,f,s
1378,m,p
1396,f,s
1420,m,s
…
```

# A simple reader

```
"""The simplest data reader."""

for line in open('ice-cream.csv'):
    row = line.split(',')
    print(row)
```

```
['respondent', 'gender', 'preference\n']
['1011', 'f', 'p\n']
['1085', 'm', 's\n']
['1099', 'm', 's\n']
['1149', 'f', 's\n']
…
```

# Anti-bugging

- It works, what's buggy?
  - Header line
  - Unwanted newlines
  - Separation of concerns

```
['respondent', 'gender', 'preference\n']
['1011', 'f', 'p\n']
['1085', 'm', 's\n']
['1099', 'm', 's\n']
['1149', 'f', 's\n']
…
```

# A Less Simple Reader

```
def read_data(path):
    """Reads comma separated data from path."""
    infile = open(path)
    infile.readline()   # discard headers
    for line in infile:
        line = line.strip()
        row = line.split(',')
        print(row)

read_data('ice-cream.csv')
```

```
['1011', 'f', 'p']
['1085', 'm', 's']
['1099', 'm', 's']
['1149', 'f', 's']
…
```

# Data Validation

- The world is messy
- Types matter
- How to fail?
  - Can respondent be divided by 10?
  - Are gender and preference validly coded?
  - Unexpected input format
- How to respond?
  - Fault
  - Assert
  - Ignore

# Validation Failures

- Fault

```
>>> '1011' / 10
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for /: 'str' and 'int'
```

- Assert

```
>>> row = ['1011', 'f', 't']
>>> assert row[1] in 'mf' and row[2] in 'ps'

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError
```

# Validation Failures 2

- Ignore

```
for line in infile:
    line = line.strip()
    row = line.split(',')
    if row[1] not in 'mf' or row[2] not in 'ps':
        continue
    …
```

- Invalid source format

```
>>> line = '"damon, matt", m, p'
>>> line.split(',')
['"damon', ' matt"', ' m', ' p']
```

# Anti-bugging 2

- Wrong answers are a Bad Thing
  - Design decision: Fail
    - Halted program does not give answers at all
    - Requires data cleaning prior to run
  - Design decision: Continue
    - Answers may be incorrect
    - Data might not allow cleaning
- Losing track of data
  - What does `row[2]` contain?
- Wrong tool for the job?

# CSV and csv.DictReader

- Standard library for parsing standard data
- Handles common header row
- Handles format complexities
- Returns data in a more self-documenting form
- Does not perform validation
  - But doesn't stop you from validating
- Search for “python csv”

# A Proper Reader

```
import csv

def read_data(path):
    """Reads comma separated data from path."""
    with open(path) as csvfile:
        reader = csv.DictReader(csvfile)
        for row in reader:
            assert row['gender'] in 'mf'
            assert row['preference'] in 'ps'
            print(row['gender'], row['preference'])

read_data('ice-cream.csv')
```

```
f p
m s
m s
f s
m p
…
```

# Internal Representation

- Input: a table of observations
  - Respondent: ignored
  - Gender: categorical (for tutorial purposes)
  - Preference: categorical (but open-ended)
- Data reduction and refining
  - Without reduction, we've just got the input
  - How can we reduce? Counting.
- How can counts be represented usefully?

# Sidebar: Python Dictionaries

- They're everywhere in inside Python (modules, namespaces, attributes, …)
- They're everywhere in real life (contacts, files, gradebooks, …)
- Dictionaries of integers? Of course.
- Dictionaries of dictionaries? Why not?
- Methods
  - Add key, remove key, list keys, access values, enumerate pairs, …

# Counting

| | pistachio | strawberry |
|---|---|---|
| male | 12 | 8 |
| female | 4 | 16 |

```
>>> counts = {
 'm':
    {'s': 8,
     'p': 12},
 'f':
    {'s': 16,
     'p': 4}
}
>>> counts['f']
{'s': 16, 'p': 4}
>>> counts['f']['p']
4
```

# Counting Reader

```
import csv
from collections import defaultdict

def ice_cream_counts(path):
    """Returns gender/preference counts from csv in path."""
    counts = {'m': defaultdict(int), 'f': defaultdict(int)}

    with open(path) as csvfile:
        reader = csv.DictReader(csvfile)
        for row in reader:
            counts[row['gender']][row['preference']] += 1

    return counts

counts = ice_cream_counts('ice-cream.csv')
print(counts)
```

```
{'f': {'p': 3, 's': 6},
 'm': {'p': 4, 's': 7}}
```

# Discussion

- How many genders are handled? How many flavor preferences?
- What is the return type of `ice_cream_count()`?
- What did the use of `defaultdict` allow?
  - What's the alternative?

# HTML Output

```
def write_html(counts):
    flavors = {'p': 'pistachio', 's': 'strawberry'}

    heading = """<!DOCTYPE html>
    <html><head><title>Ice Cream Preferences</title></head><body>
    <table border="1">
    <tr> <th>preference</th> <th>female</th> <th>male</th> </tr>"""
    table_row = '<tr> <td>{}</td> <td>{}</td> <td>{}</td> </tr>'

    print(heading)
    for f, flavor in flavors.items():
        print(table_row.format(flavor, counts['f'][f], counts['m'][f]))
    print('</table></body></html>')

counts = ice_cream_counts('ice-cream.csv')
write_html(counts)
```

| preference | female | male |
|---|---|---|
| pistachio | 3 | 4 |
| strawberry | 6 | 7 |

# Further Activities

- Reading The Fine Manuals
  - Batteries included
  - Can't use something if you don't know it exists
- Reading source code
- Reading the Literature
  - “The Practice of Programming” Kernighan & Pike (1999)
- Pandas Data Analysis Library