Module 12: Programming as a productivity tool

Topics:

- Design choices in Python programs
- A case study in Python

Why are there so many different ways to store data?

We have seen lists, classes, and dictionaries

- Each storage tool answers a different question
- One may be favoured over the others in certain situations
  - Speed of operations
  - Ease of programming
  - Memory requirements

When to use lists?

- Lists are a good choice when order matters –
  - Sorted order (numerical, alphabetical, etc.), or
  - Length of time in the collection (first added at beginning or end of list)

When to use dictionaries?

- Dictionaries are very powerful when the primary operations is searching via a key value
  - Easier to maintain than lists
  - Incredibly fast to search (essentially `O(1)`)
- Don’t do a whole lot more
  - There is no order (and if you end up sorting the dictionary entries a lot – consider a list instead)
  - Reverse look-up is brute force

Example: Architectural History Website

- Suppose we have information about a collections of buildings, including the year that the construction began
- Task: find all buildings built in a specified year and afterwards
- Note: year is not a unique identifier for a building – multiple buildings could have been built in a single year

How should we organize the data?

- List?
  - Sorted? Unsorted?
  - How to retrieve information?
- Dictionary?
  - What would be the key?
  - What would be the associated value?
- Compare the options

Another Example: DNA Sequences

Suppose you run a genetics lab, and want to study patterns in the Y chromosome. You have a collection of Y chromosome sequences. As part of your study, you want to retrieve the symbols stored at specific locations in the sequences (e.g. at position 12,025,774) as efficiently as possible.

How should we organize the data?

- List?
  - Sorted? Unsorted?
  - How to retrieve information?
- Dictionary?
  - What would be the key?
  - What would be the associated value?
- Compare the options

When to use classes?

- Use a new class when you have several pieces of related information and want to treat them as a single item
- If your class has only two fields, and one is a unique identifier, then consider a dictionary instead of a list of objects

Design Choices

- It isn't always an easy answer
- How we use the data may change
- Consider:
  - Algorithm
  - Ease of programming
  - Efficiency of algorithm
  - Memory requirements

→ Need to be flexible and adjust as needed

Putting it all together

How can programming a computer improve your productivity and your life?

- Programming can automate tasks that are mindless but important
- A computer can do complicated calculations with more accuracy
- Your programs can solve problems much more quickly than you could by hand

Case Study: Thanking a list of charitable donors

A charity accepts on-line donations. At the end of the year, the charity would like to:

- Send one thank-you note to everyone who donated at least once
- Send one receipt to each donor, for the total amount given

Where to start?

- What does the data look like?
  - Charity maintains a data file
  - Each donation is on a separate line, containing (in order)
    - Email of donor (e.g. **`generous@person.com`**)
    - Date of donation (in the format month/day/year, e.g. **`11/24/15`**)
    - Amount given in dollars (e.g. **`50`** or **`67.21`**)
  - There may be any number of spaces between the email, data and amount (at least one)

Sample input file

```
pinesap@moergrobben.cz    1/1/14   50
youthfulness@lamusic.it  2/12/14  5.25
angel@tm-druck.at        2/18/14  50
viii@bldsci.com          2/18/14  100
youthfulness@lamusic.it  5/2/14   10
```

What should be written?

- Another program will return the actual thank you notes and receipts
- Your program needs to print information about each donor on a line, containing (in order)
  - Total amount given (to 2 decimal places)
  - Email address
- Write exactly one space between the email and the amount, and place a newline after the amount
- The donors do not need to be listed in any particular order.

Sample output file

```
50.00 pinesap@moergrobben.cz
15.25 youthfulness@lamusic.it
50.00 angel@tm-druck.at
100.00 viii@bldsci.com
```

More formally

Write a function **`process_donations`** that consumes two file names: **`donors_in`** and **`donors_out`**. The function reads the donations from **`donors_in`**, and writes the distinct donors (and amount given) to **`donors_out`**, in the formats previously illustrated.

The Design Recipe still applies

- Data Analysis
- Purpose and Effects
- Contract
- Examples
- Function body
- Testing

Data Analysis

```
class Donation:
    'fields: email (Str), date (Str), amount (Float)'
    def __init__(self, m, d, a):
        self.email = m
        self.date = d
        self.amount = a
    def __repr__(self):
        s = "Donor {0.email} gave {0.amount} on {0.date}"
        return s.format(self)
    def __eq__(self, other):
       return isinstance(other, Donation) and \
          self.email == other.email and \
          self.amount==other.amount \
          and self.date==other.date
```

Contract, Purpose and Effects

```
# process_donations(donors_in, donors_out)
#   reads donation information from donors_in,
#   and writes a summary of the information
#   to donors_out
# Effects: reads donors_in,
#   and writes to donors_out.
# process_donations: Str Str -> None
# Example: See sample input and output
#   files.
def process_donations(donors_in,donors_out):
```

Identify main steps

Input

- Open the input file
- Process the input file
  - Read each line as a string
  - Convert to a **`donation`** object
- Close the input file

Output

- Open the output file
- Combine **`donation`** objects into unique donors
- Close the output file

```
# Helper function
def str_to_donation (s):
  fields = s.split()
  d = Donation(fields[0],
       fields[1], float(fields[2]))
  return d

# Processing the input file
donationfile = open(donors_in,'r')
donations = map (str_to_donation,
  donationfile.readlines())
donationfile.close()
```

Creating a unique collection of donors

- We have a list of **`donation`** objects
- Create a collection of unique donors:
  - Examine each donation
    - If donor already in unique collection, update total given
    - If not, add to unique collection
- We could use a list or a dictionary for the unique donor collection.
  - Which is better? Why?

Building the unique donor dictionary

```
# build the dictionary of donors
donor_dict = {}
for donation in donations:
  if donation.email in donor_dict:
     donor_dict[donation.email] =    \
         donor_dict[donation.email]  \
         + donation.amount
  else:
     donor_dict[donation.email] =    \
         donation.amount
```

Clean-up: Writing the file

```
donor_file = open(donors_out,'w')
for donor in donor_dict:
  s = '{0:.2f} {1}\n'
  donor_file.write(
     s.format(donor_dict[donor],
              donor))
donor_file.close()
```

Testing **`process_donations`**

- Create sample data files, including
  - Empty text file (no donations)
  - Single donation
  - Several donations, no repeated donors
  - Several donations, including repeated donors
  - Larger file, be sure to include repeated donors
- For each file,
  - Create a text file for the output you would expect to see
  - Use **`check.set_file`**
  - Use **`check.expect`** or **`check.within`**
  - *Be sure to use different file names for input and output so files don't "disappear" before you check them!*

Example of a test

```
# Test 2: input file: contains
# info for one donor, output
# file: info for that donor
check.set_file_exact(
 "actual2.txt", "out2.txt")
check.expect("t2",
  process_donations("test2.txt",
  "actual2.txt"), None)
```

Changing requirements

- Suppose the charity now wants the output file to list the donors in decreasing order of amount given
- Dictionaries cannot be sorted

  → Convert dictionary of unique donors to a list of unique donors

  → Sort it

Convert a dictionary to a list

Take the list of donor emails (the keys for **`donor_dict`**) and create a list of entries of the form **`[amount, email]`**

```
donor_list = list(map(lambda x:
  [donor_dict[x], x],
  donor_dict))
```

Sort into decreasing order

Reorder **`donor_list`** so that donor with highest total appears first in the list

- Modify any sorting algorithm studied previously
- Or, use the build in list method **`sort`**
- **`L.sort()`** will sort **L** into increasing order
- **`L.sort(reverse=True)`**
  - sorts **L** into decreasing order
  - If **L** is a list of lists, sorts **L** into decreasing order by first entry in each list (i.e. by total given, in this case)

Other approaches

- The **`date`** field is never used in this application – another solution involves just skipping the **`Donation`** class entirely.

The end of CS116

Learning to program computers is an extremely challenging task, and is harder for some people than others

- Computers do not tolerate errors
- There are also lots of places for errors
- Small changes can significantly affect run-time

Knowing how to program can be profoundly powerful!

Will you use Racket or Python ever again?

- Python is an extremely powerful tool for processing files efficiently – it might prove very useful in other contexts
- Racket programs can be quite handy for solving mathematical problems quickly
- You have developed useful resources. The knowledge is now yours to use!

After CS116 …

In subsequent courses (234, 230, 330, etc.), you can learn more about how to use computers effectively in other ways:

- Building databases
- Developing more complex mathematical ways to structure your data
- Managing large information systems projects
- Developing mid-scale software despite not being a computer scientist
- Learn about the mathematics behind algorithms

Interested in majoring in CS?

- Talk to a CS advisor about the requirements and your options.
- You will need to take CS136.
- In CS136, you will study many of the concepts from CS116 in more depth, and will be exposed to new topics as well!
- *Experience shows that students who take CS115/116 do as well as CS majors as those who start in CS135/136.*

Goals of Module 12

- Understand that multiple factors influence the best way to structure data for a specific task
  - Efficiency
  - Memory requirements
  - Simplicity
- Understand how you can dramatically improve productivity using your programming skills