Compsci 101
Files, While loops, Bagels

Susan Rodger
October 4, 2022
J is for ...

- **JSON**
  - Format for data transmitted across the web

- **JPEG**
  - Image format based on lossy compression

- **Jacquard Loom**
  - 1804 "automated" loom
Latanya Sweeney
PhD. Computer Science, MIT – first black woman
Over 100 publications, Fellow ACMI

“I am a computer scientist with a long history of weaving technology and policy together to remove stakeholder barriers to technology adoption. My focus is on "computational policy" and I term myself a "computer (cross) policy" scientist. I have enjoyed success at creating technology that weaves with policy to resolve real-world technology-privacy clashes.

http://latanyasweeney.org/
Identify 87% of US population using (dob,zip,gender). Prof. Government and Technology @ Harvard, instrumental in HIPAA because if de-identification work. Former CTO of the Federal Trade Comm.
One of her websites you can try: https://aboutmyinfo.org/identity
Announcements

• APT-3 out, due Thurs. Oct 13
• Assignment 2 Quiz due tonight!
• Assignment 2 program due Thursday
• No lab this Friday 10/7!
• APT Quiz 1 coming ... 10/13-17
  • APTs you take **by yourself** during this period
  • Take online, timed, there are two parts
    • Each part has two problems
  • APT practice quiz is up, optional
• There will be one more APT Quiz
PFTD

• Files and Data
• While loops and Collatz sequence
• Bagel APT
Text File Processing Pattern

• See module `FileStuff.py`
  • If newline `'
'` is read, call `.strip()`
  • If want to break line into “words”, call `.split()`

• Process the list that is returned by `.split()`
  • May need to convert strings to int or float or ...

• The `for line in f: pattern` is efficient
  • Contrast list returned by `f.readlines()`
def avgWord(fname):
    f = open(fname, encoding="utf-8")
    totalWords = 0
    totalLen = 0
    for line in f:
        line = line.strip() # remove newline
        data = line.split()
        for word in data:
            totalWords = totalWords + 1
            totalLen = totalLen + len(word)
    f.close()
    return totalLen/totalWords
Run FileStuff

```python
if __name__ == '__main__':
    files = ['poe.txt', 'confucius.txt', 'kjv10.txt', 'oz.txt', 'species.txt']
    for f in files:
        avg = avgWord('data/' + f)
        print(f, avg)
```

Output:

```
poe.txt  4.601549053356282
confucius.txt  4.398126192817072
kjv10.txt  4.245566037162798
oz.txt  4.496446700507614
species.txt  5.036
```
Files - Summary

• Open file:   `f = open(filename)`

• “Process” file (2 different ways):
  • `for line in f:`    # get one line at a time with “\n”
  • `x = f.readlines()`  # x is a list of lines with “\n”

• Close file:   `f.close()`

• To think about when processing lines
  • Line is a string with “\n” – `.strip()` it
  • Maybe `.split()` line into list of strings (words)?
  • Convert string to int or float - `int(“376”)`
When is a game of chess over?

• If you were to write a program to play chess
  • how many rounds in a game?
Another type of loop: While loop

- Repetition when you stop a loop based on a condition
- while CONDITION:
  BODY

- As long as condition is true, keep executing loop body.
- Must have an update in the body to get closer to condition being false
Example: while

- Playing chess

while (game not over)
make a move in the game
(game must get closer to ending)
Example: while loop – sum list

```python
lst = [4,1,8]
sum = 0
i = 0
while i < len(lst):
    sum += lst[i]
    i += 1
print(sum)
```
while loop (sum list)

lst = [4,1,8]
sum = 0
i = 0
while i < len(lst):
    sum += lst[i]
    i += 1
print(sum)

for loop (sum list)

lst = [4,1,8]
sum = 0
for n in lst:
    sum += n
print(sum)
Alternative while - while True

initialize

while True:
    if something:
        break
    if something2:
        update
        update

Continue or return
while condition vs while True

while condition:
  body
  continue

while True:
  body
  if condition:
    break
  continue

While condition is true - must update
- must get closer to making condition false
- use break to exit
Compare: while - while True

```python
lst = [4, 1, 8]
sum = 0
i = 0
while i < len(lst):
    sum += lst[i]
    i += 1
print(sum)
```

```python
lst = [4, 1, 8]
sum = 0
i = 0
while True:
    if i >= len(lst):
        break
    sum += lst[i]
    i += 1
print(sum)
```
WOTO-1 While loops

Now let’s see a problem that needs a while loop
Collatz Conjecture (Hailstone)

If number is even divide by 2
If number is odd multiply by 3 and add 1

Always end up at 1!
Why Solve This? In Python?

- We want to illustrate an indefinite loop
  - One of many mathematical sequences, but ...

- There's an XKCD comic about it!
  - Not everyone enjoys XKCD, but ...

- Mathematics is foundational in computer science, but
  - Not everyone enjoys logic/math puzzles, but ...
Developing and Reasoning about While Loops

- Don't know: *how many times* loop executes
  - *a priori* knowledge, we'll know afterword

- Do know: condition that should be true after loop
  - Its negation is the expression for `BOOL_CONDITION` (loop guard)

```python
while BOOL_CONDITION:
    LOOP_BODY
    # modify variables, affect expression
```
Concrete Example: Collatz/Hailstone

• Don't know: *how many times* loop executes
  • some numbers: long sequences, others short

• Do know: condition that should be true after loop
  • It's negation is the expression for loop guard!
  • What is true after loop below finishes?

```python
while value != 1:
    loop body
    # modify value somehow
```
```python
def hailstone(start, printing=False):
    steps = 0
    current = start
    while current != 1:
        if printing:
            print("{:3d}\t{:6d}".format(steps, current))
        if current % 2 == 0:
            current //= 2
        else:
            current = current * 3 + 1
        steps += 1

    if printing:
        print("{:3d}\t{:6d}".format(steps, current))

    return steps
```
Sample run

```python
if __name__ == '__main__':
    num = 6
    s = hailstone(num, True)
    print('num =', num, 'steps =', s)
```

Output:

```
0   6
1   3
2   10
3   5
4   16
5   8
6   4
7   2
8   1
num = 6 steps = 8
```
Collatz Data – Average no. of steps

• How do we gather data for numbers \( \leq 10,000 \)?
  • In general for numbers in range(low, high)?
  • Call function, store result, store 10,000 results?

• We'd like counts\([k]\) to be length of sequence for \( k \)
  • How do we allocate 10,000 list elements?
  • Like there is "hello" * 3
  • There is \([0] * 10000\)
```python
def analyze(limit):
    counts = []

    # max index into count is limit, but start at 1
    for _ in range(limit+1):
        counts.append(0)

    for n in range(1, limit+1):
        counts[n] = hailstone(n)

    avg = sum(counts)/len(counts)-1  # ignore index 0
    mx = max(counts)
    dex = counts.index(mx)

    print("average", avg)
    print("max is %d at %d" % (mx, dex))
```
WOTO-2 Collatz and While
Bagels
(Accumulation)
APT Bagels

• How figure out how many bagels needed?
  • 7-steps!
# APT: Bagel Counting

## Problem Statement

You are in charge of web-based orders for your neighborhood bagel store, *The Bagel Byte*. Each evening you must total the orders to be picked up the next day. Some orders are simply for $N$ bagels, but each order of a dozen or more bagels is topped off with an extra bagel, the so-called "baker's dozen". This means, for example, that an order for 25 bagels actually requires 27 bagels to fulfill since there are two extra bagels needed for each dozen in the order. An order for 11 bagels doesn't require any extra since it's for less than a dozen.

Given a list of integers representing bagel orders determine the number of bagels needed to fulfill all the orders.

```python
filename: Bagels.py
def bagelCount(orders):
    '''
    return number of bagels needed to fulfill
    the orders in integer list parameter orders
    '''

    # you write code here
```
Examples

1. orders = [1, 3, 5, 7]

   Returns: 16

   No order is for more than a dozen, return the total of all orders.

2. orders = [11, 22, 33, 44, 55]

   Returns: 175 since 11 + (22+1) + (33+2) + (44+3) + (55+4) = 175
Step 1 and 2

- **Step 1: Solve an instance (think)**
  - orders = [11, 3, 24, 17]
WOTO-3  Step 3: Generalize