Compsci 101
Files, While loops, Bagels

condition

true

body

false

Susan Rodger
February 16, 2023
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.

2/16/23
One of her websites you can try:
https://aboutmyinfo.org/identity

How unique am I?
Find out how much different you are among the masses.

Fill out the form below to see how unique you are, and therefore how easy it is to identify you from these values.
*Please note that this service is still under development.*

**Date of Birth**
- Month ▼
- Day ▼
- Year ▼

**Gender**
- Male ○
- Female ○

**ZIP Code**

ZIP code must be 5 digits long.

**Your Profile**

Results will appear here.
Your Profile

Gender: Female
ZIP Code: [redacted] (pop. 46282)

<table>
<thead>
<tr>
<th>Date of Birth</th>
<th>Easily identifiable by birthdate (about 1).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Birth Year</td>
<td>Lots with your birth year (about 273).</td>
</tr>
<tr>
<td>Range</td>
<td>Wow! There are lots of people in the same age range as you (about 1365).</td>
</tr>
</tbody>
</table>

Five year range
Exam 1 Collective Thoughts - Survey

How much time did you spend preparing/studying for the exam outside of reading in the textbook, attending class and labs.

157 responses

- 40.8% spent less than 1 hour
- 29.9% spent between one and three hours
- 15.9% spent between three and six hours
- 15.9% spent between six and nine hours
- Less than 1% spent between nine and twelve hours
- Less than 1% spent more than twelve hours
Exam 1 Collective Thoughts - Survey

As you sat down to start the exam, how did you think you’d do on the exam?

157 responses

1 – poorly

<table>
<thead>
<tr>
<th>Rating</th>
<th>Count</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1.3%</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>8.3%</td>
</tr>
<tr>
<td>3</td>
<td>56</td>
<td>35.7%</td>
</tr>
<tr>
<td>4</td>
<td>72</td>
<td>45.9%</td>
</tr>
<tr>
<td>5</td>
<td>14</td>
<td>8.9%</td>
</tr>
</tbody>
</table>

When you completed the exam, how do you think you did?

157 responses

1 – poorly

<table>
<thead>
<tr>
<th>Rating</th>
<th>Count</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>2.5%</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>5.7%</td>
</tr>
<tr>
<td>3</td>
<td>44</td>
<td>28%</td>
</tr>
<tr>
<td>4</td>
<td>63</td>
<td>40.1%</td>
</tr>
<tr>
<td>5</td>
<td>37</td>
<td>23.6%</td>
</tr>
</tbody>
</table>
Exam 1 Collective Thoughts - Survey

Was the exam fair in terms of the questions asked based on what was covered in class?

157 responses

- 50.3%: it was fair enough
- 33.1%: it was extremely fair
- 14%: neutral
- 5%: unfair
- 1%: extremely unfair
Exam 1 Collective Thoughts - Survey

Did you have enough time to take the exam?

157 responses

Not enough time

Plenty of time

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4 (2.5%)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>6 (3.8%)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>13 (8.3%)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>57 (36.3%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>77 (49%)</td>
<td></td>
</tr>
</tbody>
</table>

2/16/23 Compsci 101, Spring 2023
Announcements

• APT-3 out, due Thurs. Feb 23
• Assignment 2 program due tonight
  • Do have one grace day
• Do prelab before going to Lab on Friday
• APT Quiz 1 coming ... 2/23-2/27
  • 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 today, optional (old problems)
• There will be one more APT Quiz
Sage has added new spots!

- Small groups of students working on additional problems related to CompSci 101
- ADDED MORE SPOTS
- SAGE – Stem Advancement through Group Engagement
- See Ed Discussion Post (pinned at top) on how to sign up
PFTD

• Files and Data
• While loops and Collatz sequence
• Bagel APT
Review - Last Time on Files

• Open and Close file
  
  ```python
  f = open(fname)
  do stuff with file
  f.close()
  ```

• Read line by line
  
  ```python
  for line in f:
      do something with line
  ```

• OR Read file into list of strings – one string for each line
  
  ```python
  listLines = f.readlines()
  ```
Review - Last Time on Files

- Open and Close file
  ```python
  f = open(fname)
  do stuff with file
  f.close()
  ```
- Read line by line
- OR Read file into list of strings – one string for each line
Text File Processing Pattern

• See module FileStuff.py
  • If newline ‘\n’ 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
```python
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
```

- Line is a string, one line from the file
- data is a list of words from line
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:  \texttt{f = open(filename)}

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

• Close file:   \texttt{f.close()}

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

TRACE:

```
lst is [4, 1, 8]
sum 0
i 0
0< 3  TRUE!
```
Example: while loop – sum list

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

TRACE:

```
lst  is [4, 1, 8]
sum   4
i 0
```

Execute body of while
Example: while loop – sum list

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

TRACE:
lst is [4, 1, 8]
sum = 4
i = 1

Execute body of while
Example: while loop – sum list

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

TRACE:

```
lst is [4, 1, 8]
sum 4
i 1
```

1 < 3  TRUE!
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)
```

**TRACE:**

- lst is [4, 1, 8]
- sum = 5
- i = 1

Execute body of while
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)
```

TRACE:
- lst is [4, 1, 8]
- sum = 5
- i = 2

Execute body of while
Example: while loop – sum list

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

TRACE:

```
lst is [4, 1, 8]
sum 5
i 2
```

2 < 3  TRUE!
Example: while loop – sum list

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

**TRACE:**
```
lst is [4, 1, 8]
sum 13
i 2
```

Execute body of while
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)
```

TRACE:

```plaintext
lst is [4, 1, 8]
sum 13
i 3
```

Execute body of while
Example: while loop – sum list

lst = [4,1,8]
sum = 0
i = 0

while i < len(lst):
    sum += lst[i]
    i += 1

print(sum)

3 < 3    FALSE!    Exit while loop

TRACE:

lst  is [4, 1, 8]
sum   13
i 3

3 < 3    FALSE!    Exit while loop
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)
```

Output: 13
Summary: while loop

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

- Initialize loop variable
- Check condition, True or false?
- Update loop variable, should make condition closer to being false
Summary: while loop

\[ \text{lst} = [4, 1, 8] \]
\[ \text{sum} = 0 \]
\[ i = 0 \]
\[ \text{while } i < \text{len(lst)}: \]
\[ \quad \text{sum } += \text{lst}[i] \]
\[ \quad i += 1 \]
\[ \text{print(sum)} \]
Summary: while loop

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

1) Check condition, True or false?
2) False! Loop is over, go to statement following loop
History: From while to for loops

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
Alternative while -while True

initialize

while True:
    if something:
        break
    if something2:
        update
        update
Continue or return

while true, looks like infinite loop

Use “if” with “break”
- Break exits the loop

Still need to update to get closer to exiting loop
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)
```
Compare: while - while True

```
lst = [4,1,8]
sum = 0
i = 0

while i < len(lst):
    sum += lst[i]
    i += 1
print(sum)
```

```
lst = [4,1,8]
sum = 0
i = 0

while True:
    if i >= len(lst):
        break
    sum += lst[i]
    i += 1
print(sum)
```

Conditions are opposites!
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
```
Collatz Code

```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
```

What is new in this code? What does that new stuff do?

What is this code doing? What gets updated? Is the loop guaranteed to stop?
```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
```
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
Collatz: Guaranteed to stop?

```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
```

- Current influences the stopping condition.
- Since current is always changed, this should eventually stop.
Sample run

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

Output:
```
```
Collatz Data – Average no. of steps

• How do we gather data for numbers <= 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
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))
```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))
```
counts list when limit is 8?

- Counts is of size 8+1, we ignore slot 0
- hailstone(1), get 0
- hailstone(2), get 1 step, just divide by 2

Store answer for hailstone(1) in index 1

Store answer for hailstone(2) in index 2
counts list when limit is 8?

- **hailstone(3), get 7** (10, 5, 16, 8, 4, 2, 1)
  
  Store answer for hailstone(3) in index 3

- **hailstone(4), get 2**
  
  Store answer for hailstone(4) in index 4
counts list when limit is 8?

- hailstone(5), get 5 (16, 8, 4, 2, 1)
- And so on…..
- Hailstone(6) is 8, hailstone(7) is 16, hailstone(8) is 3

Store answer for hailstone(5) in index 5
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.

```
class Bagels:
    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. \( \text{orders} = [1, 3, 5, 7] \)

   Returns: 16

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

2. \( \text{orders} = [11, 22, 33, 44, 55] \)

   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]
Step 1 and 2

• **Step 1: Solve an instance (think)**
  - orders = [11, 3, 24, 17]
  - $11 + 3 + (24+2) + (17+1) = 58$
  - Total: 58

• **Step 2: What did we do?**
  - Write down in words
WOTO-3  Step 3: Generalize