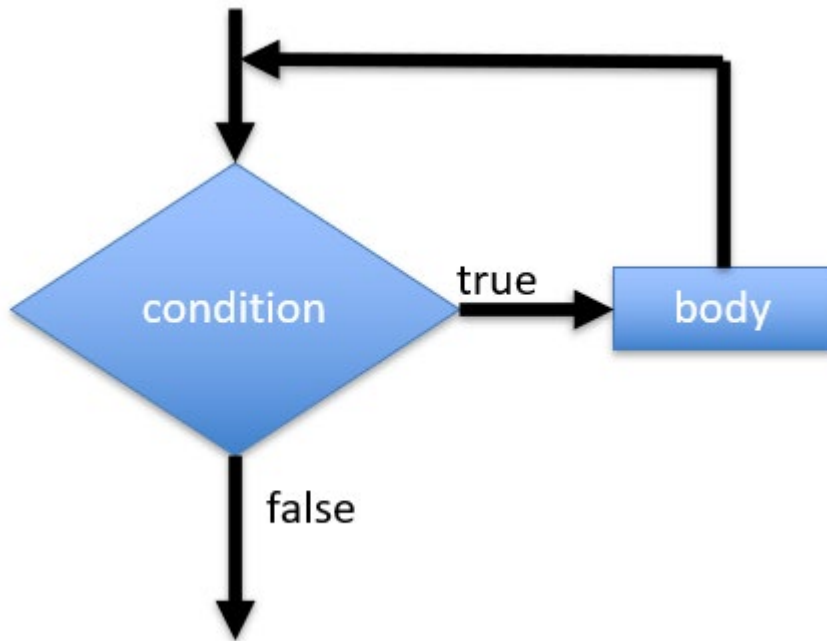


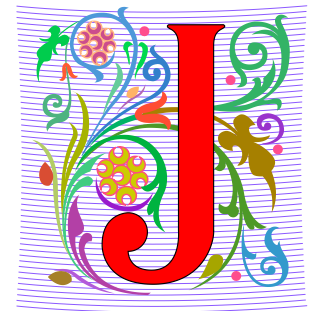
Compsci 101

Files, While loops, Bagels

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 of *de-identification* work. Former CTO of the Federal Trade Comm.

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.

About

Samples

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.

Submit

Your Profile



Results will appear here.

Your Profile

Gender: Female

ZIP Code: [REDACTED] (pop. 46282)

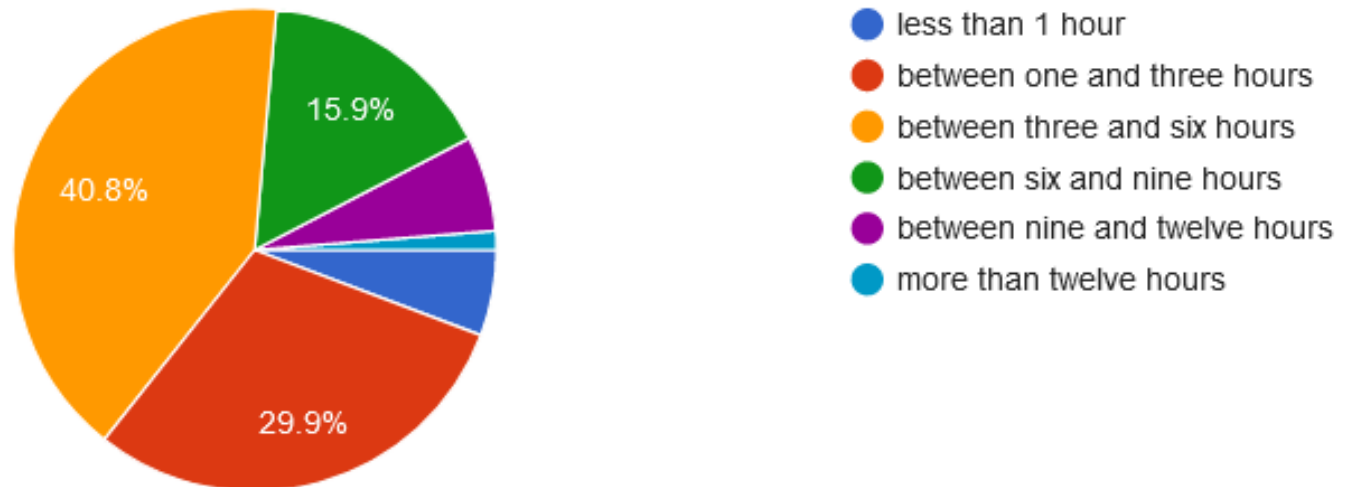
Date of Birth	[REDACTED]	Easily identifiable by birthdate (about 1).
Birth Year	[REDACTED]	Lots with your birth year (about 273).
Range	[REDACTED] to [REDACTED]	Wow! There are lots of people in the same age range as you (about 1365).

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.

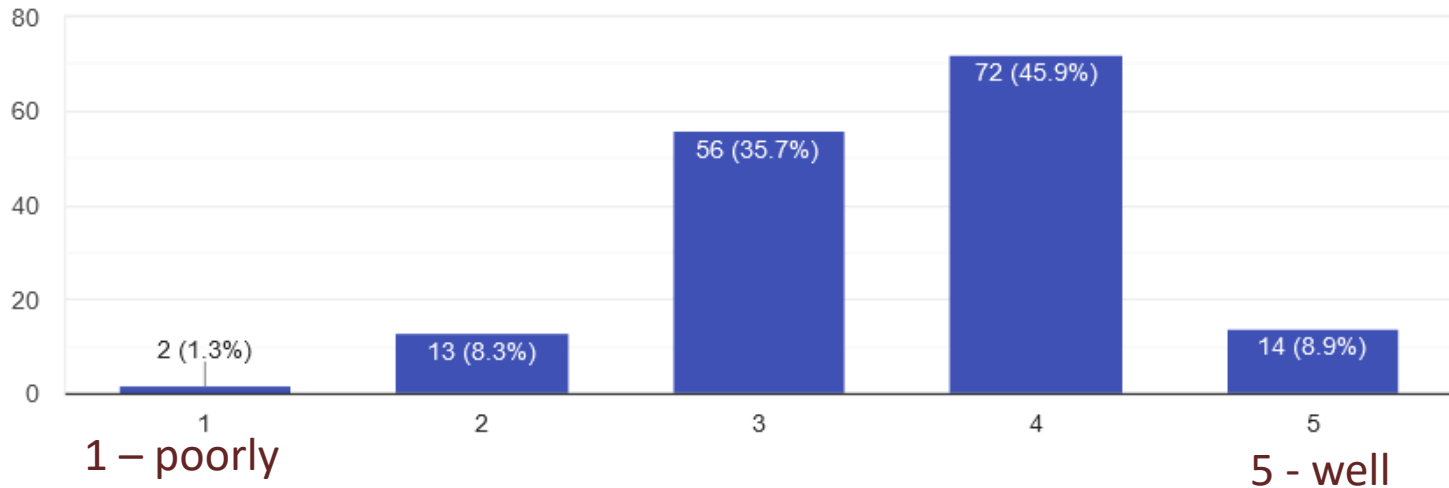
157 responses



Exam 1 Collective Thoughts - Survey

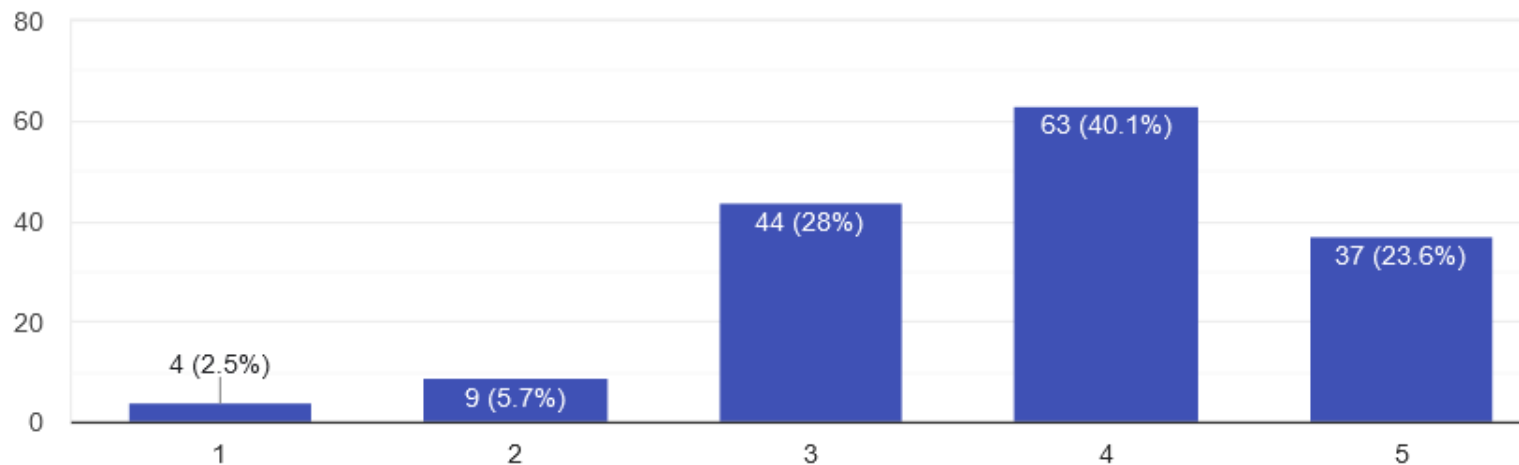
As you sat down to start the exam, how did you think you'd do on the exam?

157 responses



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

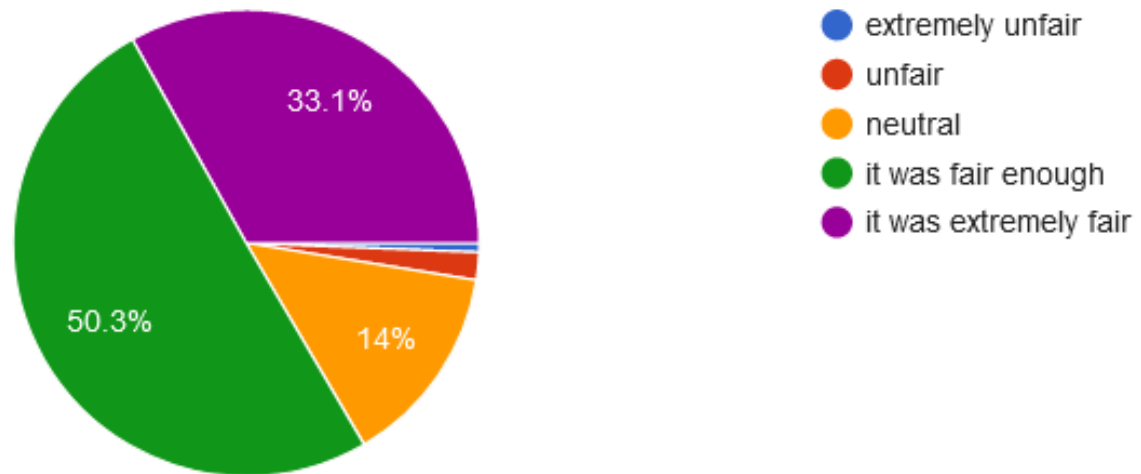
157 responses



Exam 1 Collective Thoughts - Survey

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

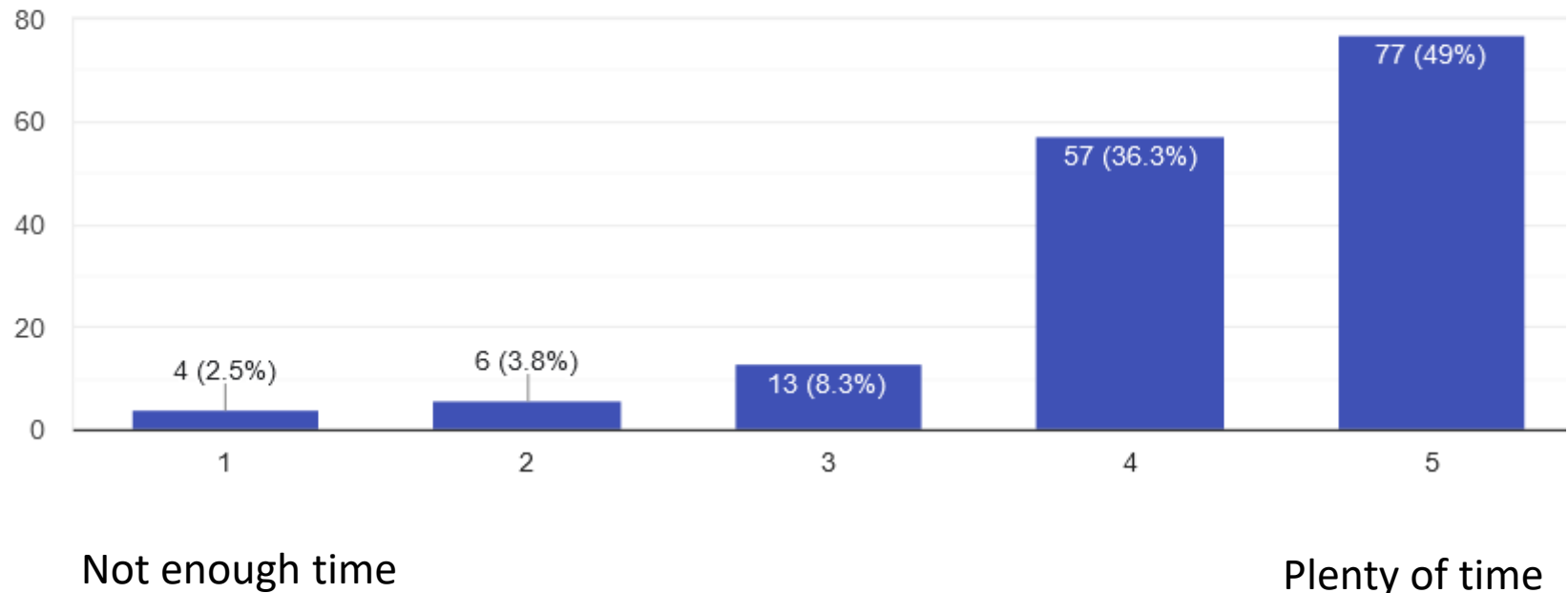
157 responses



Exam 1 Collective Thoughts - Survey

Did you have enough time to take the exam?

157 responses



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 – **S**tem **A**dvancement through **G**roup **E**ngagement**
- **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**

```
f = open(fname)
```

do stuff with file

```
f.close()
```

- **Read line by line**

```
for line in f:
```

do something with line

- **OR Read file into list of strings – one string for each line**

```
listLines = f.readlines()
```

Review - Last Time on Files

- **Open and Close file**

```
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()`

FileStuff.py: avgWord

```
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
```


FileStuff.py: avgWord

```
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

```
20 ▶ if __name__ == '__main__':  
21     files = ["poe.txt", "confucius.txt", "kjb10.txt", "oz.txt", "species.txt"]  
22     for f in files:  
23         avg = avgWord("data/"+f)  
24         print(f, avg)
```

Output:

```
poe.txt 4.601549053356282  
confucius.txt 4.398126192817072  
kjb10.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”)`

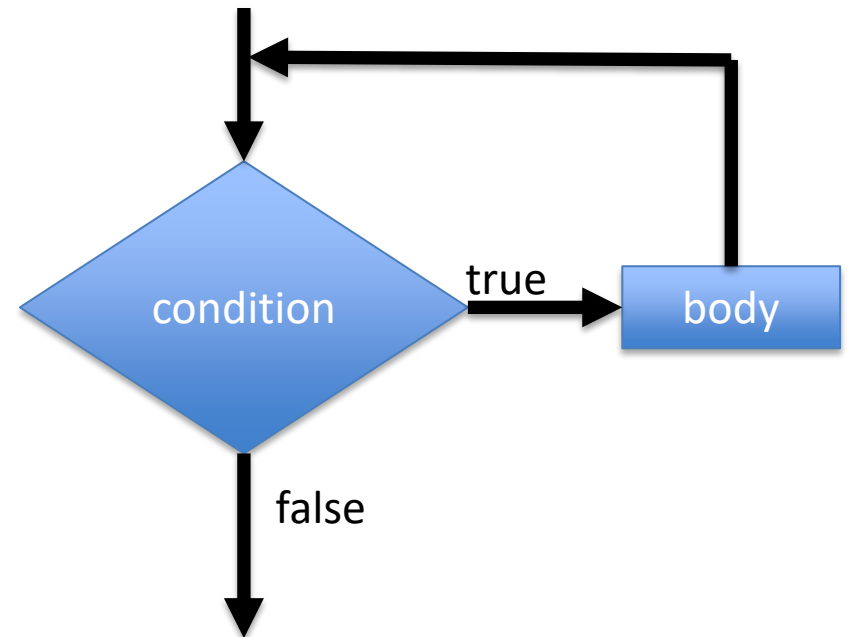
When is a game of chess over?

- **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

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

Example: while loop – sum list

0 < 3 TRUE!

```
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

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

1 < 3 TRUE!

```
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

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 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

5

i

2

Execute body of while

Example: while loop – sum list

2 < 3 TRUE!

```
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

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

```
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 3

Execute body of while

Example: while loop – sum list

3 < 3 FALSE! Exit while loop

```
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 3

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 3

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

```
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) True! Execute
body

3) Then check
condition again.
True or false?

Summary: while loop

```
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

```
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)
```

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)
```

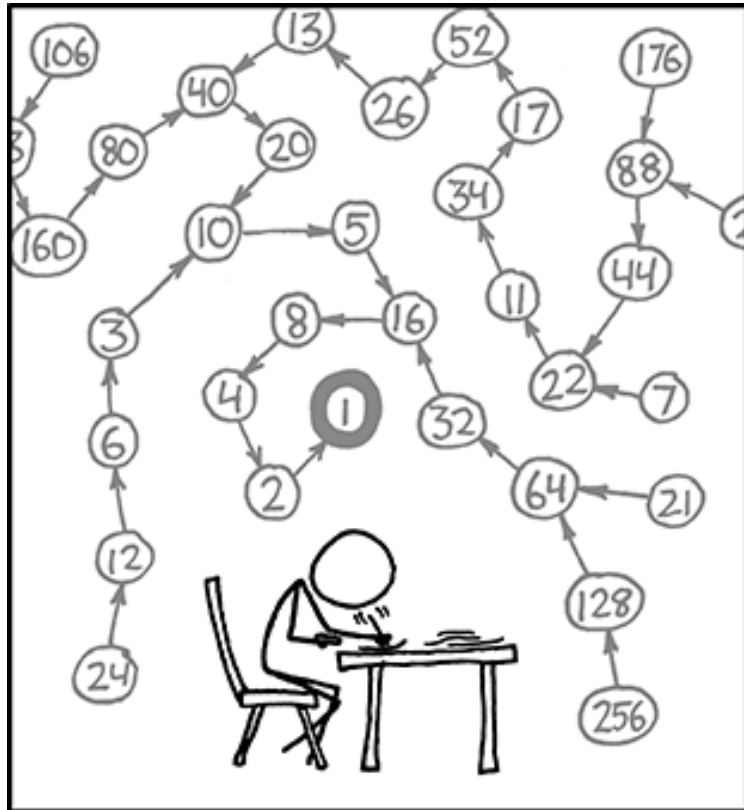
Conditions are
opposites!

WOTO-1 While loops

<http://bit.ly/101s23-0216-1>

Now let's see a problem that needs
a while loop

<https://xkcd.com/710/>



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!

THE COLLATZ CONJECTURE STATES THAT IF YOU PICK A NUMBER, AND IF IT'S EVEN DIVIDE IT BY TWO AND IF IT'S ODD MULTIPLY IT BY THREE AND ADD ONE, AND YOU REPEAT THIS PROCEDURE LONG ENOUGH, EVENTUALLY YOUR FRIENDS WILL STOP CALLING TO SEE IF YOU WANT TO HANG OUT.

Why Solve This? In Python?

- https://en.wikipedia.org/wiki/Collatz_conjecture
- **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 afterward
- **Do know: condition that should be true after loop**
 - Its negation is the expression for `BOOL_CONDITION` (loop guard)

```
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?

```
while value != 1:  
    loop body  
    # modify value somehow
```

Collatz Code

```
6 def hailstone(start, printing=False):
7     """ ... """
14    steps = 0
15    current = start
16    while current != 1:
17        if printing:
18            print("{:3d}\t{:6d}".format(steps, current))
19        if current % 2 == 0:
20            current //= 2
21        else:
22            current = current * 3 + 1
23        steps += 1
24
25    if printing:
26        print("{:3d}\t{:6d}".format(steps, current))
27    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?

Collatz Code

```
6 def hailstone(start, printing=False):
7     """..."""
14    steps = 0
15    current = start
16    while current != 1:
17        if printing:
18            print("{:3d}\t{:6d}".format(steps, current))
19        if current % 2 == 0:
20            current //= 2
21        else:
22            current = current * 3 + 1
23        steps += 1
24
25    if printing:
26        print("{:3d}\t{:6d}".format(steps, current))
27    return steps
```

Collatz: New stuff

```
6 def hailstone(start, printing=False):
7     """..."""
14    steps = 0
15    current = start
16    while current != 1:
17        if printing:
18            print("{:3d}\t{:6d}".format(steps, current))
19        if current % 2 == 0:
20            current //= 2
21        else:
22            current = current * 3 + 1
23        steps += 1
24
25    if printing:
26        print("{:3d}\t{:6d}".format(steps, current))
27    return steps
```

Default value, if no argument

Syntax for nicer formatting

Collatz: Guaranteed to stop?

```
6 def hailstone(start, printing=False):
7     """ ... """
14    steps = 0
15    current = start
16    while current != 1:
17        if printing:
18            print("{:3d}\t{:6d}".format(steps, current))
19        if current % 2 == 0:
20            current //= 2
21        else:
22            current = current * 3 + 1
23        steps += 1
24
25    if printing:
26        print("{:3d}\t{:6d}".format(steps, current))
27    return steps
```

current influences the stopping condition

Since current is always changed, this should eventually stop

Sample run

```
44 ▶ if __name__ == '__main__':  
45     num = 6  
46     s = hailstone(num, True)  
47     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

Think: Analysis in Collatz.py

```
29 def analyze(limit):
30     counts = []
31     # max index into count is limit, but start at 1
32     for _ in range(limit+1):
33         counts.append(0)
34
35     for n in range(1, limit+1):
36         counts[n] = hailstone(n)
37
38     avg = sum(counts)/len(counts)-1 # ignore index 0
39     mx = max(counts)
40     dex = counts.index(mx)
41     print("average", avg)
42     print("max is %d at %d" % (mx, dex))
```

Why do both range calls have +1?

Why no printing when this is called?

Analysis in Collatz.py

```
29 def analyze(limit):
30     counts = []
31     # max index into count is limit, but start at 1
32     for _ in range(limit+1):
33         counts.append(0)
34
35     for n in range(1, limit+1):
36         counts[n] = hailstone(n)
37
38     avg = sum(counts)/len(counts)-1 # ignore index 0
39     mx = max(counts)
40     dex = counts.index(mx)
41     print("average", avg)
42     print("max is %d at %d" % (mx, dex))
```

counts list when limit is 8?

- **Counts is of size 8+1, we ignore slot 0**

analyze	
limit	8
counts	•
—	8

analyze(limit)

list	0	1	2	3	4	5	6	7	8
	0	0	0	0	0	0	0	0	0

Store answer for hailstone(1) in index 1

- **hailstone(1), get 0**
- **hailstone(2), get 1 step, just divide by 2**

analyze	
limit	8
counts	•
—	8
n	2

analyze(limit)

list	0	1	2	3	4	5	6	7	8
	0	0	1	0	0	0	0	0	0

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)**

analyze	
limit	8
counts	•
—	8
n	3

analyze(limit)

list									
0	1	2	3	4	5	6	7	8	
0	0	1	7	0	0	0	0	0	0

Store answer for hailstone(3) in index 3

- **hailstone(4), get 2**

analyze	
limit	8
counts	•
—	8
n	4

analyze(limit)

list									
0	1	2	3	4	5	6	7	8	
0	0	1	7	2	0	0	0	0	0

Store answer for hailstone(4) in index 4

counts list when limit is 8?

- **hailstone(5), get 5 (16, 8, 4, 2, 1)**

```
analyze
limit | 8
counts |
- | 8
n | 5
```

analyze(limit)

list

0	1	2	3	4	5	6	7	8
0	0	1	7	2	5	0	0	0

Store answer for hailstone(5) in index 5

- **And so on.....**
- **Hailstone(6) is 8, hailstone(7) is 16, hailstone(8) is 3**

```
analyze
limit | 8
counts |
- | 8
n | 8
```

analyze(limit)

list

0	1	2	3	4	5	6	7	8
0	0	1	7	2	5	8	16	3

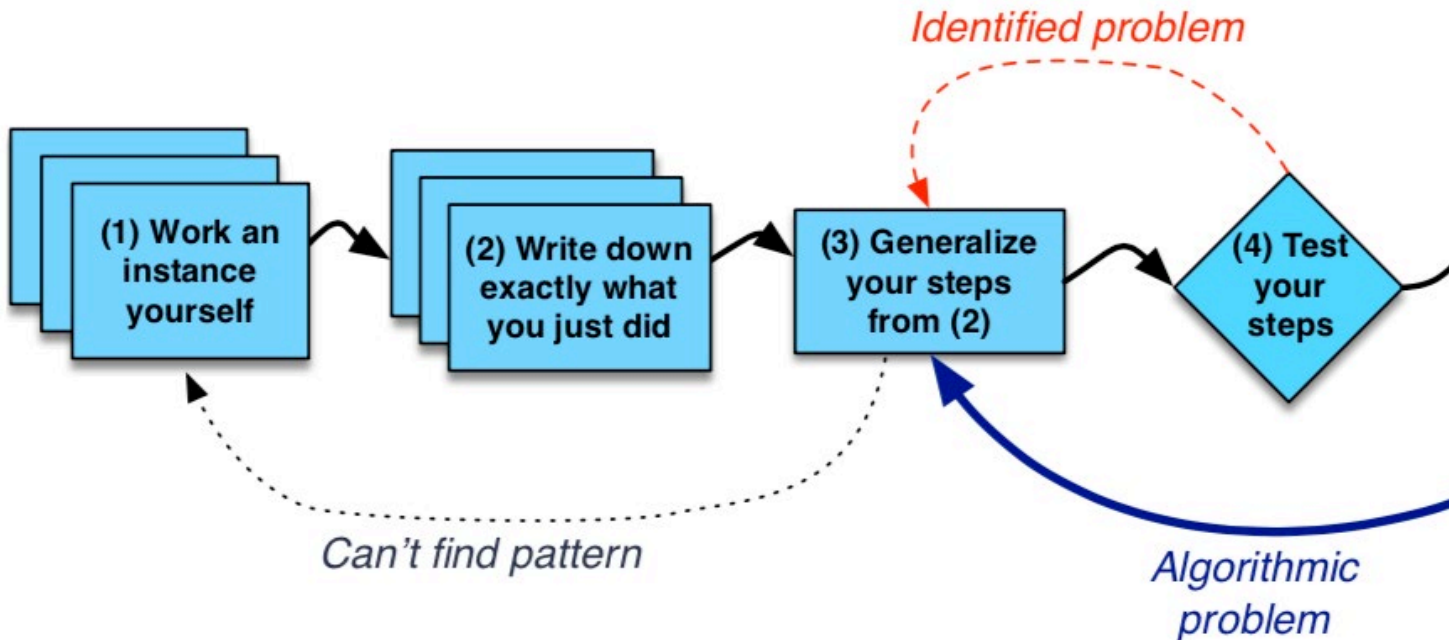
WOTO-2 Collatz and While
<http://bit.ly/101s23-0216-2>



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

```
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

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]

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
<http://bit.ly/101s23-0216-3>