Compsci 101
DeMorgan's Law, Short circuiting, Images, Tuples
Live Lecture

Announcements

- Assign 2 due today!
- APT-4 due Thursday, March 11
- Lab 5 on Friday
- Exam 1 – Regrade request deadline 5pm TODAY!
- Exam 2 prep
  - Old test 2 links (Calendar-today's date(3/4))

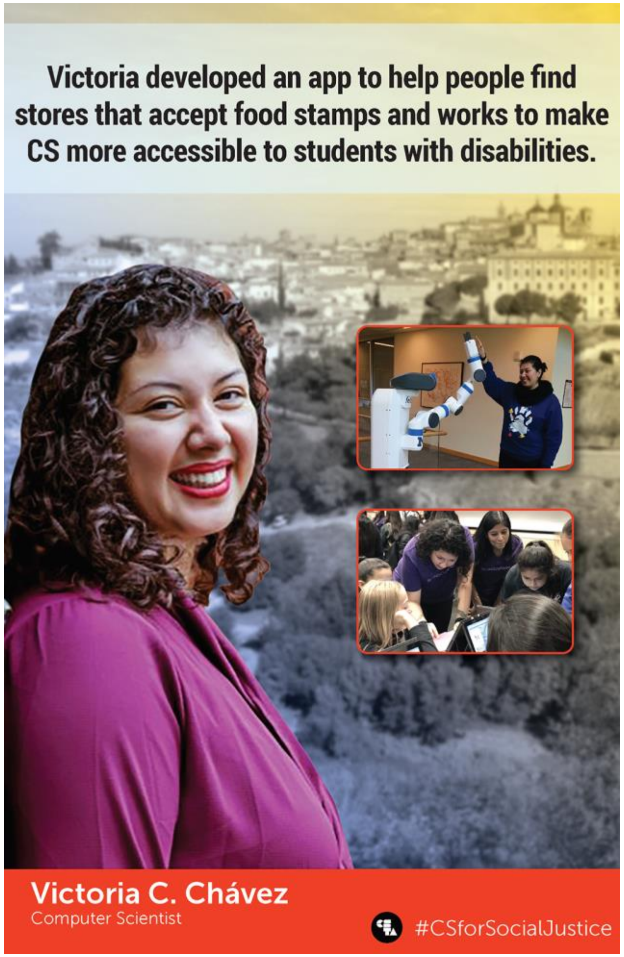
- No class next Tues/Wed-Wellness Days
  - Office/consulting hours affected

APT Quiz 1 tomorrow…

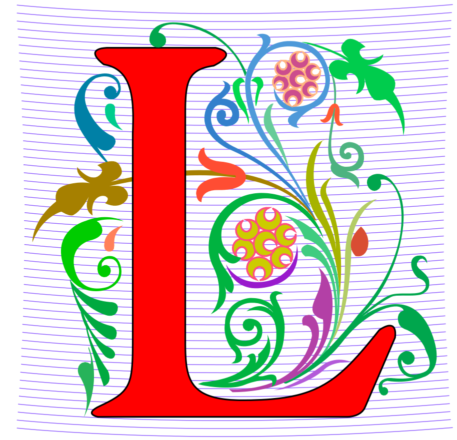
- APT Quiz 1 is 3/5 8AM -3/8 11PM – finish by 11pm
- There are two parts – each part is 1.5 hours
- Pick a start time for each part,
  - Once you start a part, You have 1.5 hours
  - If you get accommodations, you get those
- 4 APTs to solve (2 in each part)
  - Take parts 1 and 2 on same day or different days
- Start APT Quiz on Sakai!
- See old APT Quiz problems so you can practice
  - On APT page – NOT FOR CREDIT

Computer Scientists to Know
Victoria Chávez

- B.S.-CS, Hispanic Studies
- M.S.-CS Education
- Software Engineer
  - Twitter, Microsoft
- K-16 CS educator
  - University of Rhode Island
- SNAPy creator



`L` is for …



- **Loops**
  - While, For, Nested – Iteration!
- **Library**
  - Where we find APIs and Implementations
- **Logic**
  - The Boolean Heart of …
- **Linux**
  - The OS that runs the world?

PFTD

- DeMorgan's Law
- Short Circuiting
- Images & Tuples
  - Start today, finish next class
- Maybe an APT?

Review: Index without error?

```
lst = ["a","b","c","a"]
dex = lst.index("b")
  lst.index("b") is 1
  lst.index("B") ERROR!
  lst.index("B") ??? -1
```

- Use while loop to implement index.
- What is the while loop’s Boolean condition?

```
dex = 0
while BOOL_CONDITION:
   dex += 1
```

Review: DeMorgan's Law

- While loop stopping conditions, stop with either:
  - `lst[dex] == elm`
  - `dex >= len(lst)`
- While loop needs negation: DeMorgan's Laws

  `not (A and B)` equivalent to **`(not A) or (not B)`**

  `not (A or B)` equivalent to **`(not A) and (not B)`**

```
while not (lst[dex] == elm or dex >= len(lst)):

while lst[dex] != elm and dex < len(lst):
```

TPS: DeMorgan's Law

Fill in the blanks

A	B	not (A and B)	(not A) or (not B)
True	True	False	False
True	False	True	True
False	True	True	True
False	False	True	True

A	B	not (A or B)	(not A) and (not B)
True	True	False	False
True	False	False	False
False	True	False	False
False	False	True	True

WOTO-1: Will this work?
http://bit.ly/101s21-0304-1

- If not, what input will not work?

WOTO-1: Will this work?
http://bit.ly/101s21-0304-1

- If not, what input will not work?

```
def index(lst, elm):
    dex = 0
    while lst[dex] != elm and dex < len(lst):
        dex += 1
    if dex < len(lst):
        return dex
    else:
        return -1
```

Short Circuit Evaluation

- Short circuit evaluation, these are not the same!

```
    while lst[dex] != elm and dex < len(lst):
```

```
    while dex < len(lst) and lst[dex] != elm:
```

- As soon as truthiness of expression known
  - Stop evaluating
  - In `(A and B)`, if `A` is false, do not evaluate `B`

Example: To sit in the student section of a game you need to "have a ticket" and "be a student"

Python Logic Summarized

- A and B is True only when A is True and B is True
- A or B is False only when A is False and B is False

- Short-circuit evaluation of A or B ?
  - If A is true, do not evaluate B

A	B	Evaluate B with and?	Evaluate B with or?
True	True	Yes	No
True	False	Yes	No
False	True	No	Yes
False	False	No	Yes

WOTO-2 – Boolean Logic http://bit.ly/101s21-0304-2

- In your groups:
  - Come to a consensus

Example: Images

WOTO-3 – Images
http://bit.ly/101s21-0304-3

- In your groups:
  - Come to a consensus

Review SimpleDisplay.py

- Access to PIL and Image module
  - What type is img?
  - https://pillow.readthedocs.io/en/latest/

```python
from PIL import Image

if __name__ == '__main__':
    img = Image.open("images/bluedevil.png")
    img.show()
    print("width %d, height %d" % (img.width, img.height))
```

Review: Images

- Image is a collection of pixels
  - Organized in rows: # rows is image height
  - Each row has the same length: image width

- Pixels addressed by (x, y) coordinates
  - Upper-left (0,0), Lower-right (width-1,height-1)
  - Typically is a single (x, y) entity: tuple

- Tuple is immutable, indexed sequence (a, b, c)

Review: Tuple: What and Why?

- Similar to a list in indexing starting at 0
  - Can store any type of element
  - Can iterate over
- Immutable - Cannot mutate/change its value(s)
  - Efficient because it can't be altered
- Consider `x = (5,6)` and `y = ([1,2],3.14)`
  - Think: What is `x[0] = 7`? `y[0].append(5)`?

APT 4 - TxMsg

Problem Statement

Strange abbreviations are often used to write text messages on uncomfortable mobile devices. One particular strategy for encoding texts composed of alphabetic characters and spaces is the following:

- Spaces are maintained, and each word is encoded individually. A word is a consecutive string of alphabetic characters.
- If the word is composed only of vowels, it is written exactly as in the original message.
- If the word has at least one consonant, write only the consonants that do not have another consonant immediately before them. Do not write any vowels.
- The letters considered vowels in these rules are 'a', 'e', 'i', 'o' and 'u'. All other letters are considered consonants.

For instance, "ps i love u" would be abbreviated as "p i lv u" while "please please me" would be abbreviated as "ps ps m". You will be given the original message in the string parameter `original`. Return a string with the message abbreviated using the described strategy.

Specification

```
filename: TxMsg.py

def getMessage(original):
    """
    return String that is 'textized' version
    of String parameter original
    """

    # you write code here
```

Example

Examples

1. ```
   "text message"

   Returns "tx msg"
   ```
   ```

WOTO-4 – TxMsg
http://bit.ly/101s21-0304-4

- In your groups:
  - Come to a consensus

Debugging APTs: Going green

- TxMsg APT: from ideas to code to green
  - What are the main parts of solving this problem?
  - Transform words in original string
    - Abstract that away at first
  - Finding words in original string
    - How do we do this?

```
def getMessage(original):
    ret = [ ]

        ret.append(transform(word))
    return ret
```

Debugging APTs: Going green

- TxMsg APT: from ideas to code to green
  - What are the main parts of solving this problem?
  - Transform words in original string
    - Abstract that away at first
  - Finding words in original string
    - How do we do this?

```
def getMessage(original):
    ret = [ ]
    for word in  original.split() :
        ret.append(transform(word))
    return ret
```

Debugging APTs: Going green

- TxMsg APT: from ideas to code to green
  - What are the main parts of solving this problem?
  - Transform words in original string
    - Abstract that away at first
  - Finding words in original string
    - How do we do this?

```
def getMessage(original):
    ret = [ ]
    for word in  original.split() :
        ret.append(transform(word))
    return ret   # join?
```

Write helper function *transform*

- How?
- Use seven steps
- Work an example by hand

Transform word - Step 1: work small example by hand

- Word is “please”
- Letter is ‘p’, YES
- answer is “p”
- Letter is ‘l’, NO
- Letter is ‘e’, NO
- Letter is ‘a’, NO
- Letter is ‘s’, YES
- answer is “ps”
- Letter is ‘e’, NO

Step 2: Describe what you did

- Word is “please”, create an empty answer
- Letter is ‘p’, consonant, no letter before, YES
- Add ‘p’ to answer
- Letter is ‘l’, consonant, letter before “p”, NO
- Letter is ‘e’, vowel, letter before ‘l’, NO
- Letter is ‘a’, vowel, letter before ‘e’, NO
- Letter is ‘s’, consonant, letter before ‘a’, YES
- Add ‘s’ to answer
- Letter is ‘e’, vowel, letter before ‘s’, NO
- Answer is “ps”

Step 3: Find Pattern and generalize

Need letter before, pick “a”

answer is empty

for each letter in word

If it is a consonant, and the letter before is a vowel, then add the letter to the answer

This letter is now the letter before

return answer

Step 4 – Work another example

- Word is message
- Letter is ‘m’, before is ‘a’, add ‘m’ to answer
- Letter is ‘e’, before is ‘m’, NO
- Letter is ‘s’, before is ‘e’, add ‘s’ to answer
- Letter is ‘s’, before is ‘s’, NO
- Letter is ‘a’, before is ‘s’, NO
- Letter is ‘g’, before is ‘a’, add ‘g’ to answer
- Letter is ‘e’, before is ‘g’, NO
- Answer is “msg” WORKS!!

Step 5: Translate to Code

```
# Letter before is “a”

# answer is empty

# for each letter in word
```

```
# start with a vowel
```

Step 5: Translate to Code

```
# Letter before is "a"        # start with a vowel
before = 'a'
# answer is empty
answer = [ ]          # or this could be an empty string
# for each letter in word
for ch in word:
```

Step 5: Translate to Code (code)

```
#If it is a consonant, and the letter before is a
#vowel, then add the letter to the answer


#This letter is now the letter before

# return answer
```

STOP HERE…

- You finish
- May need to debug

Why use helper function 'transform'?

- **Structure of code is easier to reason about**
  - Harder to develop this way at the beginning
  - Similar to accumulate loop, build on what we know

- **We can debug pieces independently**
  - What if transform returns "" for every string?
  - Can we test transform independently of `getMessage`?