P is for …

- **Python**
  - Whatever you want it to be? Language!!!
- **Parameter**
  - When an argument becomes a variable
- **Power Cycle**
  - Not the last resort. But works
- **P2P**
  - From networking to collaboration

---

The Tech Twins

- Troy and Travis Nunnally
- Between them: 2 master’s and 1 doctorate from Georgia Tech
- Cofounders of Brain Rain Solutions
  - Augmented-reality
  - Internet-of-things
- Applied machine learning

Troy: “My advice would be to stay consistent. Always think persistently and consistently about learning a particular craft.”

Travis: “I think that you have to be passionate and find something that you simply love and enjoy. Not only find that thing — but actually be a lifelong learner around that.”

---

Announcements

- **Assign 4 GuessWord due Thursday! March 17**
  - Sakai Assignment Quiz due WEDNESDAY
    - No late day!
    - We did not do this in a prelab
    - Only 84 of you have done this!!!
- **Assign 5 Clever GuessWord out, due March 29**
  - Sakai Assignment quiz due March 28
  - Will talk about on Thursday
- **APT-5 due Thurs, March 24 (recommend before Exam 3)**
Announcements (2)

• Lab 8 Friday
  • Lots of dictionary prep for Exam 3
  • There is a prelab out today

• Sakai quiz – one due Thursday 10:15am
  • reviews Dictionaries/sorting

• Exam 2 back – regrade requests by Wednesday, 3/23
  • Join SAGE – additional way to practice problem solving

• Fill out Course Survey sent out Monday
  • If 75% fill it out, everyone gets 2pts extra on Exam 2!

Exam 3 – in person – Tues, March 22

• Exam is in class on paper – 10:15am
  • Need pen or pencil

• See materials under 3/22 date
  • Exam 3 Reference sheet - part of exam

• Covers
  • Topics: sets, parallel lists, dictionaries, sorting, tuples, (No images)
  • APTs through APT5
  • Labs through Lab 8
  • Assignments through Assignment 4
  • Sakai Quizzes through 3/17

PFTD

• Dictionaries
• Parallel Lists
• Jotto game

VenmoTracker APT

• If Harry pays Sally $10.23,
  • "Harry:Sally:10.23" then Harry is out $10.23
APT: VenmoTracker

Problem Statement
You've been asked to keep track of how often people spend money using Venmo and whether they receive more money than they pay out. The input to your program is a list of transactions from Venmo. Each transaction has the same form: "from:to:amount" where from is the name of the person paying amount dollars to the person whose name is to. The value of amount will be a valid float with at most two decimal places.

Return a list of strings that has each person who appears in any transaction with the net cash flow through Venmo that person has received. Every cent paid by the person to someone else is a pay-out and every cent received by a person is a pay-in. The difference between pay-out and pay-in is the cash flow received. This will be negative for each person who pays out more than they get via pay-in. See the examples for details.

The list returned should be sorted by name. The name in the list returned are in the format "name:netflow" where the netflow is obtained by using str(val) where val is a float representing the net cash flow for that person.

Store money as list values, multiplying by 100 and dividing by 100 as needed for processing input and output, respectively.

Specification
filename: VenmoTracker.py
def networth(transactions):
    return list of strings based on transactions, which is also a list of strings
    if you write code here return []

Examples
1. transactions: ["owen:susan:10", "owen:robert:10", "owen:drew:10"]
   returns ["drew:10.0", "owen:-30.0", "robert:10.0", "susan:10.0"]
   Owen pays everyone.

WOTO-1 VenmoTracker

Tools We’ve Used Before

• Keep track of every person we see
  • Use a list

• Keep track of net worth: money in, money out
  • Use a parallel list

• Maintain invariant: names[k] <-> money[k]
  • kth name has kth money
Example:

```
[ "Harry:Sally:10.23", "Zeyu:Sally:20.00", "Sally:Barak:10.00"]
```

• How would we solve this?
• Could we use a parallel list?
• What would be the output?

Process Transaction

"Harry:Sally:10.23"

```python
names = [ ]
money = [ ]
```

Put Harry in:

"Harry:Sally:10.23"

```python
names = [ "Harry" ]
0
money = [ -10.23 ]
0
```

Put Sally in:

"Harry:Sally:10.23"

```python
names = [ "Harry", "Sally" ]
0 1
money = [ -10.23, 10.23 ]
0 1
```
Process next transaction
"Zeyu:Sally:20.00"

names = [ "Harry", "Sally" ]
0 1

money = [ -10.23, 10.23 ]
0 1

Put Zeyu in:
"Zeyu:Sally:20.00"

names = [ "Harry", "Sally", "Zeyu" ]
0 1 2

money = [ -10.23, 10.23, -20.00 ]
0 1 2

Update Sally in:
"Zeyu:Sally:20.00"

names = [ "Harry", "Sally", "Zeyu" ]
0 1 2

money = [ -10.23, 30.23, -20.00 ]
0 1 2

Process next Transaction
"Sally:Barak:10.00"

names = [ "Harry", "Sally", "Zeyu" ]
0 1 2

money = [ -10.23, 30.23, -20.00 ]
0 1 2
Coding up Venmo

def networth(transactions):
    names = [ ]
    money = [ ]
    for trans in transactions:
        # split up trans
        data = trans.split(":")
        sender = data[0]
        receiver = data[1]
        amount = float(data[2])

Coding up Venmo

def networth(transactions):
    names = [ ]
    money = [ ]
    for trans in transactions:
        # split up trans
        data = trans.split(":")
        sender = data[0]
        receiver = data[1]
        amount = float(data[2])
Coding up Venmo

```
if sender not in names:
    names.append(sender)
    money.append(0)
# similar if receiver not in names
# update money
indexSender = names.index(sender)
indexReceiver = names.index(receiver)
money[indexSender] -= amount
money[indexReceiver] += amount
# create output in correct format
```

Seen parallel lists before

- Solution outlined is reasonable, efficient?
  - How long does it take to find index of name?
  - It depends. Why?

- `list.index(elt) or elt in list`—fast?
  - What does "fast" mean? Relative to what?

Let's try Dictionaries....

- Example:
  ```
  [ "Harry:Sally:10.23", "Zeyu:Sally:20.00",
    "Sally:Barak:10.00"
  ]
  ```

- How would we solve this?
- Could we use a dictionary?
- What would be the output?
Example with Dictionary
1) "Harry:Sally:10.23"
   - Start with empty dictionary, insert Harry

Example with Dictionary
1) "Harry:Sally:10.23"
   - Insert Sally

Example with Dictionary
2) "Zeyu:Sally:20.00"
   - Insert Zeyu

Example with Dictionary
2) "Zeyu:Sally:20.00"
   - Update Sally
Example with Dictionary
3) “Sally:Barak:10.00”

- Update Sally
  Zeyu → -20.00
  Harry → -10.23
  Sally → 20.23

- Insert Barak
  Zeyu → -20.00
  Harry → -10.23
  Sally → 20.23
  Barak → 10.00

Return Value
Zeyu → -20.00
Harry → -10.23
Sally → 20.23
Barak → 10.00

- List of (key, value) pairs
  [ (“Zeyu”, -20.00), (“Harry”, -10.23), (“Sally”, 20.23), (“Barak”, 10.00) ]

Sort by name:
  [ (“Barak”, 10.00), (“Harry”, -10.23), (“Sally”, 20.23), (“Zeyu”, -20.00) ]
Return Value

- Zeyu: -20.00
- Harry: -10.23
- Sally: 20.23
- Barak: 10.00

- [("Barak", 10.00), ("Harry", -10.23), ("Sally", 20.23), ("Zeyu", -20.00)]

Put in final format:
- ["Barak:10.00", "Harry:-10:23", "Sally:20.23", "Zeyu:-20.00"]

How would the code be different if we used a dictionary?

Coding up Venmo with Dictionary

```python
def networth(transactions):
    venmo = {}
    for trans in transactions:
        # split up trans
```

Coding up Venmo with Dictionary

```python
def networth(transactions):
    venmo = {}
    for trans in transactions:
        # split up trans
        data = trans.split(":")
        sender = data[0]
        receiver = data[1]
        amount = float(data[2])
```

Initialize dictionary

This part the same
Coding up Venmo with Dictionary

```python
if sender not in venmo:
    venmo[sender] = 0
# similar if receiver not in names
# update money
venmo[sender] -= amount
venmo[receiver] += amount
# create output in correct format
```

Jotto: Game similar to GuessWord

- [http://jotto.augiehill.com/single.jsp](http://jotto.augiehill.com/single.jsp)
- No letters repeat – have to agree on this
- Shall we play a game?

Write program where Computer Guesses Your Word

- Brute force, no thinking or eliminating letters
  - Pick a word at random, guess it
  - If x letters in common? Only keep words with x letters in common
  - Repeat until guessed

WOTO-2 Approaching Implementation

- What is needed?
- What order should the code do things?
Start with Blank Screen

1. Computer gets a list of words
2. Computer chooses a word at random
3. User/player enters # letters in common
4. Only keep words with that # in common

Initialization
Loop

Iterative Programming!

- Start with a task
- Implement only that task
- Write some code to check that the code works
  - Run and debug until it works
- Repeat

“Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.”
- Brian Kernighan (original Unix contributor)

SimpleJotto.py

- We have a file of five letter words: kwords5.txt
  - Would you like to play a game?
- Let's start! Simple version that sort of works 😊

Jotto Step 1

- Read the file kwords5.txt
  - Read the file kwords5.txt
  - Don't continue until we know this works
- Let's go code up!
Jotto Step 2

• Pick a word at random
  • Pick a word at random, show the user
  • Don't continue until we know this works

Jotto Step 3

• Get number of letters in common
  • Get # letters in common, do something?
  • Don't continue until we know this works

WOTO-3 More on Jotto

• What is needed?
• What order should the code do things?

Next slides show the code we did in class
Code up getWordList and chooseAWord

```python
def getWordList(filename):
    ret = []
    f = open(filename)
    for line in f:
        ret.append(line.strip())
    f.close()
    return ret

def chooseAWord(words):
    return random.choice(words)
```

Code up updateWordList and commonCount

```python
def handleUserInput():
    wordToGuess = input("Guess a word with 5 letters: ")
    return wordToGuess

def updateWordList(words, numInCommon, userword):
    return [w for w in words if numInCommon == commonCount(w, userword)]

def commonCount(word1, word2):
    # assumes words don't have duplicate letters
    set1 = set(list(word1))
    set2 = set(list(word2))
    #print(set1, set2)
    return len(set1 & set2)
```

playGame First part

```python
def playGame():
    won = False
    wordToGuess = handleUserInput()
    numTries = 0

    # Initialize
    # Read the file kwords5.txt
    words = getWordList('kwords5.txt')
```

playGame loop

```python
while (not won and numTries < 30):
    # Loop
    ## Pick a word at random
    nextWord = chooseAWord(words)
    ## Get number of letters in common
    numInCommon = commonCount(wordToGuess, nextWord)
    ## Only keep words with that number in common
    words = updateWordList(words, numInCommon, nextWord)
    print("next word is: ", nextWord)
    print("num letters in common is: ", numInCommon)
    print("words remaining: ", len(words))

    #check stopping condition
    if nextWord == wordToGuess:
        won = True
        numTries += 1
return won
```
Rest of these slides we did not do, but they outline what we did with the code.

**commonCount**
- Given two words, return \# letters in common
- Similar to SandwichBar?
  - Sandwich ingredients \(\rightarrow\) letters in a word

- Let’s go code up commonCount and chooseAWord

---

**Jotto Step 4**
- Only keep words with that number in common

- Example: User enters 0, keep only words with 0 letters in common (replace 0 with 1, 2, \(\ldots\), N)
  - What are the steps here?
  - What’s similar?
  - What do we do to solve this?
- Helper function: commonCount
- Let’s go code up!

---

**Finishing SimpleJotto**
- When is the game over? How to signal that
  - Interaction is via \# letters in common

- Add functionality: number of guesses?
  - Remind the user where they are

- Let’s go code up!
Writing and Testing functions

- We'd like to test the function isolated from game
  - Ensure we don't have to play to test it
  - Unit testing similar to APT tests

- Can do this in your main!
  - Remove for final submission for hand grading

- Also use the APT testing framework and Gradescope

Summary: Jotto

- Break down entire game into steps
- Recognize what steps belong where
  - Initialization: Before loop
  - Inside loop
  - Anything after loop? End of game stuff
- Code one step at a time (or one function)
  - Test if that step worked (or submit to Gradescope)