# CompSci 101
# Fall 2021

Lecture 14

# Reminders

- Assignments
  - APT 4 due
- Exam 2
  - 10/26
  - Review session (10/21)
- Small-group tutoring

# Key instructions

- Input ✔
- Output ✔
- Assignments* ✔
- Math/Logic ✔
- Conditionals ✔
- Repetition ✔

**not listed in book*

# Python Data Types

- int, float, bool ✔
- Collections
  - Strings ✔
  - Lists ✔
  - Tuples ✔
  - Sets ✔
  - Dictionaries ⟵

# PFTD

- Dictionaries

# KISS Principle

- Think of the non-computing context for any word/terms
- KISS model
  - Work smarter, not harder!!
- "Good programmers are simply good designers."
  - *-Dr. Washington*
- Design first and always!
- Importance of reusability
- *USE PyCharm/PythonTutor IF YOU HAVE QUESTIONS!*

# People to Know: Dr. Latanya Sweeney

- PhD-MIT
  - First Black woman to earn PhD in CS from MIT
- BS-Harvard Extension School
- Professor of the Practice, Government and Technology (Harvard)
- “87% of US population can be uniquely identified with just zip code, gender, and DOB.”
- k-anonymity
  - Privacy model-protect person’s privacy in data sharing situations

- PBS segment (w/Dr. Safiya Noble)
  - https://www.pbs.org/wgbh/nova/video/search-engine-breakdown/

# ~~How the~~ Dictionary ~~is made~~

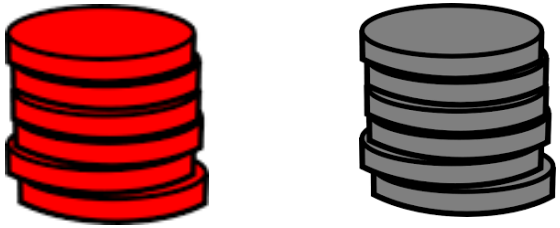
- Using a dictionary is reasonably straight-forward
  - We will be clients, not implementers
  - Efficiency not a large concern in 101
  - Our goal is to just get stuff done ☺




# Motivation for Dictionary

http://bit.ly/venmotracker

- If Harry pays Sally $10.23,
  - "Harry:Sally:10.23" then Harry is out $10.23
- How do we extract sender, receiver, amount?
  - How to process
  - In Python



# 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]`**
  - $k^{th}$ name has $k^{th}$ money

TPS: Do 7-step's Step 3: What are the generalized steps for this APT?

# Activity 1: https://bit.ly/101f21-10-19-1

# Some APT details

- Given a person's name, if we haven't seen it…
  - Append to **`names`**, append 0 to **`money`**
  - Why must we do both? Invariant, update!
- Find index of person's name in **`names`**
  - Update corresponding entry in **`money`**
- Use $$ * 100 to avoid floating point issues
- Sorted names: **`sorted(…)`**
  - Returns a sorted list of the passed in sequence

# Sorting functions

- sorted(*x*, key=*key*, reverse=*reverse*)
  - x→iterable object
  - *key* (optional)→determine order
    - Default is None
  - *reverse* (optional)→ascending/descending
    - Default is False

- *list*.sort(reverse=*reverse*, key=*func*)
  - Only for lists
  - key must be a function

- *Recommend: Stick to sorted for now (understand sort for future)

# 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?

- Such a common idiom most languages support fast alternative: dictionary aka map aka hash …

# What is a Dictionary?

- A collection of (key, value) pairs (abstract view)
  - Look up key, find the value
- Very, very fast: essentially index by key
  - For list **`a[3]`** takes same time as **`a[3000]`**
- For Dictionary: **`d["cake"]`**
  - Finding the value associated with "cake"

# How to use a Dictionary

- Create: d = **{}**
  - `d = {'a': 10, 'b': 100}`
  - `d = dict([('a', 10), ('b', 100)])`
- Insert: **`d[KEY] = VALUE`**
- Update/Reassign: **`d[KEY] = VALUE`**
- Get a value (like list indexing): **`d[KEY]`**
- *Key* membership (not values): **`KEY in d`**
  - No membership check for values

# How to use a Dictionary

- Like lists, but with keys
- KEY – immutable type, unique within dictionary
- VALUE – any type, not unique within dictionary
- Unordered collection of (KEY, VALUE) pairs

# Activity 2: https://bit.ly/101f21-10-19-2

# Short Code and Long Time

- See module WordFrequencies.py
  - Find # times each word in a list of words occurs
  - We have tuple/pair: word and word-frequency

```
37  def slowcount(words):
38      pairs = [(w,words.count(w)) for w in set(words)]
39      return sorted(pairs)
```

- TPS: How many times is **`words.count(w)`** called?
  - Why is **`set(words)`** used in list comprehension?

# WordFrequencies with Dictionary

- If start with a million words, then…
- We look at a million words to count # "cats"
  - Then a million words to count # "dogs"
  - Could update with parallel lists, but still slow!
  - Look at each word once: dictionary!

- Key idea: use word as the "key" to find occurrences, update as needed
  - Syntax similar to **`counter[k] += 1`**

# Using fastcount

- Update count if we've seen word before
  - Otherwise it's the first time, occurs once

```
28  def fastcount(words):
29      d = {}
30      for w in words:
31          if w in d:
32              d[w] += 1
33          else:
34              d[w] = 1
35      return sorted(d.items())
```

  - d.items()-returns key-value pairs as list of tuples

# Reminders

- Work smarter, not harder
- Design first
- Try to identify where you are stuck
  - Identify resources to help solve problem
- Leverage your design and PythonTutor to understand program flow of control
  - http://pythontutor.com