Plan for FWON

- Review current assignments and APTs
  - Review Dictionaries and how to use them
  - Code and APT walk-through
  - Algorithms, searching, sorting?

- Toward understanding sorting
  - What are the algorithms for sorting?
  - What are the libraries for sorting?

ACM MidAtlantic Programming Contest

- Saturday, Nov 7
- 185 teams!
- Each team: 3 students, one computer
- 5 hours to solve 6-8 problems

- Need volunteers to help!
  - T-shirt, meals, snacks! Fun!
  - Deliver printouts to teams
- Signup here:

Answer Questions


Clever, Snarky, Evil, Frustrating Hangman

- Computer changes secret word every time player guesses to make it "hard" to guess
  - Must be consistent with all previous guesses
  - Idea: the more words there are, harder it is
    - Not always true!

- Example of greedy algorithm
  - Locally optimal decision leads to best solution
  - More words to choose from means more likely to be hung
Canonical Greedy Algorithm

- How do you give change with fewest number of coins?
  - Pay $1.00 for something that costs $0.43
  - Pick the largest coin you need, repeat

Greedy not always optimal

- What if you have no nickels?
  - Give $0.31 in change
  - Algorithms exist for this problem too, not greedy!

Clever Hangman

- When you guess a letter, you're really guessing a category (secret word "salty")
  - _ _ _ _ _ _ and user guesses 'a'
    - "gates", "cakes", "false" are all the same
    - "flats", "aorta", "straw", "spoon" are all different

- How can we help ensure player always has many words to distinguish between?
**Debugging Output and Game Play**

- Sometimes we want to see debugging output, and sometimes we don't
  - While using Microsoft Word, don't want to see the programmer's debugging statements
  - Release code and development code

- You'll approximate release/development using a global variable DEBUG
  - Initialize to False, set to True when debugging
  - Ship with DEBUG = False

**Look at howto and function categorize**

- Play a game with a list of possible words
  - Initially this is all words
  - List of possible words changes after each guess

- Given template "_ _ _ _", list of all words, and a letter, choose a secret word
  - Choose all equivalent secret words, not just one
  - Greedy algorithm, choose largest category

```python
words = categorize(words, guess, letter)
```

**Completing function categorize**

- Loop over every string in words, each of which is consistent with guess (template)
  - This is important, also letter cannot be in guess
  - Put letter in template according to word
  - _ _ _ a _ t might become _ _ _ a n t

- How to re-use guess (template) make copy?
- How to create key in dictionary
  - Why can't key be a list?

**Voterigging APT**

- For example: [5, 10, 7, 3, 8] answer is 4, why?
  - If you steal a vote, who do you steal from?
  - Why?
  - Why is this like coin problem? Clever Hang?

- How do you find who to steal from?
  - At least two approaches, functions or loop
    - Use max and index, or write a loop to find max
  - When are you done stealing?
    - This governs writing the APT
Python shortcuts you can ignore

- The zip function, tuples from two lists
- Does something right if lists have different sizes. Look it up

```python
words = ['dog', 'cat', 'fish', 'guava']
counts = [3, 2, 1, 5]
cc = zip(words, counts)
```

```
[('dog', 3), ('cat', 2), ('fish', 1), ('guava', 5)]
```

Python shortcuts you can ignore

- `enumerate` - the iterable
  - Sometimes you need an index, sometimes elt
  - `for elt in lst:` or
  - `for dex in range(len(lst))`:

```python
for dex, elt in enumerate(['a', 'b', 'c']):
    print dex, elt
```

```
0 'a'
1 'b'
2 'c'
```

Python shortcuts you can ignore

- Default dictionaries
  - Typically we see if key in D before modifying
  - If initialization always same for new keys ...

```python
import collections
dd = collections.defaultdict(int)
dd['apple']
0
ee = {}
e[e['apple']]
Key error
```
Python functions you CANNOT ignore

- We know how to sort, we call sorted
  - Example from lab and class, sorting tuples
  - Function sorted returns a new list, original not changed

```
xx = [('dog', 3), ('cat', 2), ('fish', 1), ('guava', 2)]
yy = sorted(xx)
[('cat', 2), ('dog', 3), ('fish', 1), ('guava', 2)]
```

First use what you know

- You can re-organize data to sort it as you'd like, list comprehensions are your friend

```
xx = [('dog', 3), ('cat', 2), ('fish', 1), ('guava', 2)]
... nlist = [{t[1],t[0]} for t in xx]
[(3, 'dog'), (2, 'cat'), (1, 'fish'), (2, 'guava')]
yy = sorted(nlist)
[(1, 'fish'), (2, 'cat'), (2, 'guava'), (3, 'dog')]
```

Many algorithms for sorting

- In some classes knowing these algorithms matters
  - Not in Compsci 101
  - Quick, Merge, ...
  - We'll use built-in, library sorts, all languages have some

- We will concentrate on calling or using these
  - How does API work?
  - What are characteristics of the sort?
  - How to use in a Pythonic way?

How do we sort? Ask Tim Peters!

- Sorting API
  - Sort lists (or arrays, ...)
  - Backwards, forwards, ...
  - Change comparison
    - First, Last, combo, ...

- Sorting Algorithms
  - We'll use what's standard!

Best quote: `import this`

I've even been known to get Marmite *near* my mouth -- but never actually in it yet. Vegamite is right out