1. Questions?

2. Miscellany:
   (a) Comparison using is vs ==. A discussion of None.
   (b) From Monday: str, repr, and eval.

3. List object. A lot can be learned from pydoc3 lists
   (or visit https://docs.python.org/3/library and search for list)
   (a) Lists keep their objects in order. They're objects can be accessed by index. They can be modified; they're mutable.
   (b) Like strings, they can be indexed and sliced. Unlike strings, their objects can be different types; they're heterogeneous.
   (c) They can be constructed from other iterable objects:

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(set(sorted('hello, world')))
['o', 'h', 'w', 'd', ' ', 'l', 'e', ',', 'r']
```
   (d) They, themselves, are iterable: you can encounter their elements, in order:

```
>>> l = ['bon','do','go','iwa','pi','to']  # halfword words
>>> for x in l
...    print(x+x)
bonbon
dodo
gogo
iwaiwa
pipi
toto
```
   (e) You can determine their length (use len), and you can concatenate them (use +).
   (f) Because they're mutable, you can change them:
      i. You can assign pre-existing elements (l[i] = 'tar').
      ii. You can add new elements at the end of a list with l.append(x). This is common when you're building up lists of results.
      iii. You append all the elements from another iterable with l.extend(container).
You can remove elements. Remove and return the last with `l.pop()`, or the element at position `i` with `l.pop(i)`:

```python
>>> d = [ 'Bashful', 'Doc', 'Dopey', 'Grumpy', 'Happy', 'Sleepy', 'Sneezy' ]
>>> l = list(d)
>>> while l:
...     i = randint(0,len(l)-1)
...     print(l.pop(i))
Sneezy
Sleepy
Grumpy
Happy
Doc
Dopey
Bashful
>>> print(l)
[]
>>> print(d)
['Bashful', 'Doc', 'Dopey', 'Grumpy', 'Happy', 'Sleepy', 'Sneezy']
```

Python has a similar operator, `del`, that does not return the value.

**Other methods of lists that are useful:**

- `l.clear()`: Destructively remove all elements of the list.
- `l.copy()`: Return a new, shallow copy: a new list containing shared references to contained objects. The following is true of non-empty lists, `l`:
  
  ```python
  l.copy()[0] is l[0]
  ```
- `l.count(v)`: Returns the number of times `v` appears in `l`.
- `l.index(v)`: Returns the index of the first occurrence of `v`, or raises an error. (Sadly, unlike strings, there is no `find(v)` method, so check first, if necessary.)
- `l.remove(v)`: Removes (without returning) a value from `l`.
- `l.insert(i,o)`: Inserts `o` in `l`, so that it will have index `i`.
- `l.reverse()`: Destructively reverses `l`.
- `l.sort(key=None, reverse=False)`: Sorts a list, in-place, destructively. The `key` argument allows you to specify a function that, given a value constructs a key to be used for sorting; if `reverse=True`, the sort is opposite. *All sorting in python is stable.*

**List comprehensions.** You can construct new lists using iteration with a for expression:

```python
# Typical, simple:
>>> words = [ line.strip() for line in open('/usr/share/dict/words') ]
# Guarded. The \id{if} always follows.
>>> wds = [ word for word in words if len(word) == 3 ]
# Nested.
>>> split6 = [ x+y for x in wds for y in wds if x+y in words ]
```