Administration

- Just to be clear, it's okay to ask questions about the assignment at office hours, even if it's in the last 24 hours.

- Assignment 2 will come out over the weekend, and the deadline will be moved to the 27th.
  - Office hours will be held Monday instead of Tuesday that week.

- The midterm will be held June 30th at the regular lecture time and regular lecture room.
Administration

- There is a request for a volunteer note-taker.
- There is a student in this class who requires a volunteer notetaker as an accommodation for a disability. By signing up and posting your notes, you can make a significant difference for this individual's capacity to fully participate in this course. Go to: http://www.studentlife.utoronto.ca/accessibility/pcourseslist.aspx or come in person to Accessibility Services 215 Huron St. Suite 939.
- Many students notice the quality of their notetaking improves through volunteering.
- You will also receive a certificate of recognition.
Immutable objects.

- So far all we've seen are immutable objects.

- That is objects don't change.

- Instead of making an old int into a new one, we make a new int, and throw the old one away.
Immutable objects.

- What if we want to change an immutable object?

- It's a lot of work, we need to make a new object that is identical to the old one except for our changes.

- This is fine for small things like ints and strings, but takes a lot of time for large things like images.
Mutable Objects.

- If we want to change a really large object without keeping the original, then making a big copy, modifying it and tossing the rest is wasteful.
- Instead, we can use a mutable object, that we're allowed to change.
- This also allows us to define functions that change objects, rather than return new ones.
Aliasing

- Consider:
  
  \[ \begin{align*} 
  x &= 10 \\
  y &= x \\
  x &= 5 \\
  \text{print } x, y 
  \end{align*} \]

- We know this will print 5 10 to the screen, because ints are immutable.
Aliasing

Let pic be an already initialised picture and consider:

```python
x = pic
y = x
# sets the green to 0.
for pixel in x:
    media.set_green(pixel, 0)
media.show(y)
```

Pics are mutable, so this will show a picture with no green.
Aliasing and functions.

- When one calls a function, one is effectively beginning with a bunch of assignment statements.
  - That is, the parameters are assigned to the local variables.
- But with mutable objects, these assignment statements mean that the local variable refers to a mutable object that it can change.
- This is why functions can change mutable objects, but not immutable ones.
Break, the first.
Lists

- Recall from the assignment that you had to refer to each co-ordinate by a variable.
  - This is annoying, and can easily be really slow in high-dimensional spaces.
- Python has a way of grouping similar items called a list.
- Denoted by:
  
  ```python
  list_name = [list_elt0, list_elt1, ..., list_eltn]
  ```
Lists

• To get to the i-th element of a list we use:
  \[ \text{list\_name}[i-1] \]

• We use i-1 because lists are indexed from 0.

• This means to refer to the elements of a 4 element list named list\_name we use
  \[ \text{list\_name}[0], \text{list\_name}[1], \text{list\_name}[2], \text{list\_name}[3] \]

• Lists are mutable.
Lists

- You can also have an empty list: [].
- You can index into lists from the back.
- list_name[-i] returns the ith element from the back.
- Lists are heterogeneous:
  - That is, the elements in a list need not be the same type, can have ints and strings.
  - Can even have lists themselves.
Lists: Functions

• Lists come with lots of useful functions and methods.
• `len(list_name)`, as with strings, returns the length of the list.
• `min(list_name)` and `max(list_name)` return the min and max so long as the list is well defined.
• `sum(list_name)` returns the sum of elements so long as they're numbered.
  • *Not* defined for lists of strings.
Lists: Methods

- `append(value)` – adds the value to the end of the list.
- `sort()` - sorts the list so long as this is well defined. (need consistent notions of > and ==)
- `insert(index, value)` – inserts the element value at the index specified.
- `remove(value)` – removes the first instance of value.
- `count(value)` – counts the number of instances of value in the list.
Looping over Lists.

- Often we want to do a similar operation to every element of the list.
- Python allows us to do this using for loops.

```python
for item in list:
    block
```

- This is equivalent to:

```python
item = list[0]
block
item = list[1]
block
...
```
Loopying over Lists.

- Loops can be tricky with immutable objects
  
  ```python
  for item in list:
      block
  ```
- Here, item is immutable, so we can't alter the list elements.
- If we want to alter the list elements, we need to refer to the indices of the list.
To do that, we use the `range()` function.

- `range(i)` returns an ordered list of ints ranging from 0 to i-1.
- `range(i, j)` returns an ordered list of ints ranging from i to j-1 inclusive.
- `range(i, j, k)` returns a list of ints ranging from i to j-1 with a step of at least k between ints.

So `range(i, k) == range(i, k, 1)`

To modify a list element by element we use:

```python
for i in range(len(list)):
    block
```
List slicing.

- Sometimes we want to perform operations on a sublist.
- To refer to a sublist we use list slicing.
- \( y = x[i:j] \) gives us a list \( y \) with the elements from \( i \) to \( j-1 \) inclusive.
  - \( x[:] \) makes a list that contains all the elements of the original.
  - \( x[i:] \) makes a list that contains the elements from \( i \) to the end.
  - \( x[:j] \) makes a list that contains the elements from the beginning to \( j-1 \).
- \( y \) is a new list, so that it is not aliased with \( x \).
Break, the second.
Tuples.

- Sometimes we want our lists to be immutable.
- Can help if we're worried about aliasing carelessness.
- To do that we can make a tuple.
  
  \[ \text{tuple\_name}=(\text{item0, item1, item2, ...}) \]

  - Items are referenced by `tuple\_name[i]` not `tuple\_name(i)`
  - Single element tuples must be defined with a comma to avoid ambiguity
    - \((8+3)\) vs. \((8+3,)\)
Strings revisited.

- Strings can be considered tuples of individual characters. (since they are immutable).
- In particular, this means that we can use the list knowledge that we gained, and apply it to strings.
  - Can reference individual characters by string[+/-i].
  - Strings are not heterogenous, they can only contain characters.
  - `min()` and `max()` defined on strings, but `sum()` is not.
  - You can slice strings just as you can lists.
String methods revisited.

- Now that we know that we can index into strings, we can look at some more string methods.
  - `find(substring)`: give the index of the first character in a matching the substring from the left or -1 if no such character exists.
  - `rfind(substring)`: same as above, but from the right.
  - `find(substring,i,j)`: same as `find()`, but looks only in `string[i:j]`. 
Nested Lists

- Because lists are heterogeneous, we can have lists of lists.
- This is useful if we want matrices, or to represent a grid or higher dimensional space.
- We then reference elements by `list_name[i][j]` if we want the jth element of the ith list.
- So then naturally, if we wish to loop over all the elements we need nested loops:

```python
for item in list_name:
    for item2 in item:
        block
```