Lecture 9 – Comparisons, Control Statements, Randomness
Announcements

- HW03 - Functions, Histograms, and Groups
  - Due Monday (02/21)

- No lab this week

- Checkpoint/Project 1:
  - Paired assignment that covers the previous section of the course material
  - Released this morning (02/17) and in two weeks Thursday 03/03
pivot groups together rows that share a combination of values. It differs from group because it organizes the resulting values in a grid.
Group vs Pivot

Pivot
- One combo of grouping variables per entry

Group
- One combo of grouping variables per row

```python
more_cones.pivot('Flavor', 'Color', values='Price', collect=sum)
```

<table>
<thead>
<tr>
<th>Color</th>
<th>bubblegum</th>
<th>chocolate</th>
<th>strawberry</th>
</tr>
</thead>
<tbody>
<tr>
<td>dark brown</td>
<td>0</td>
<td>10.5</td>
<td>0</td>
</tr>
<tr>
<td>light brown</td>
<td>0</td>
<td>4.75</td>
<td>0</td>
</tr>
<tr>
<td>pink</td>
<td>4.75</td>
<td>0</td>
<td>8.8</td>
</tr>
</tbody>
</table>

```python
more_cones.group([['Flavor', 'Color'], sum])
```

<table>
<thead>
<tr>
<th>Flavor</th>
<th>Color</th>
<th>Price sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>bubblegum</td>
<td>pink</td>
<td>4.75</td>
</tr>
<tr>
<td>chocolate</td>
<td>dark brown</td>
<td>10.5</td>
</tr>
<tr>
<td>chocolate</td>
<td>light brown</td>
<td>4.75</td>
</tr>
<tr>
<td>strawberry</td>
<td>pink</td>
<td>8.8</td>
</tr>
</tbody>
</table>
Group vs Pivot

Pivot
- One combo of grouping variables **per entry**
- **Two** grouping variables: columns and rows
- Aggregate values of values column
- Missing combos = 0 (or empty string)

Group
- One combo of grouping variables **per row**
- **Any number** of grouping variables
- Aggregate values of **all other columns** in table
- Missing combos **absent**

Copyright © 2016 Barnard College
t.select(column, ...) or t.drop(column, ...)
t.take([row, ...]) or t.exclude([row, ...])
t.sort(column, descending=False)
t.where(column, are.condition(...))
t.apply(function, column, ...)
t.group(column) or t.group(column, function)
t.group([column, ...]) or t.group([column, ...], function)
t.pivot(cols, rows) or t.pivot(cols, rows, vals, function)
t.join(column, other_table, other_table_column)

https://coms1016.barnard.edu/python-reference.html
Comparison Operators

The result of a comparison expression is a **bool** value: True, False

<table>
<thead>
<tr>
<th>Operator</th>
<th>Table predicate</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>are.equal_to</td>
</tr>
<tr>
<td>!=</td>
<td>are.not_equal_to</td>
</tr>
<tr>
<td>&gt;</td>
<td>are.above</td>
</tr>
<tr>
<td>&gt;=</td>
<td>are.above_or_equal_to</td>
</tr>
<tr>
<td>&lt;</td>
<td>are.below</td>
</tr>
<tr>
<td>&lt;=</td>
<td>are.below_or_equal_to</td>
</tr>
</tbody>
</table>
The result of a comparison expression is a **bool** value

\[
x = 2 \quad y = 3
\]
The result of a comparison expression is a \textit{bool} value

\begin{align*}
x &= 2 \\
y &= 3
\end{align*}
Comparison Operators

The result of a comparison expression is a \texttt{bool} value

\begin{align*}
x &= 2 \\
y &= 3 \\
x &> 1 \\
x &> y \\
x &= y \\
x &!= 2 \\
y &\geq 3 \\
2 &< x < 5
\end{align*}
Comparison Operators

The result of a comparison expression is a `bool` value

```
x = 2
y = 3

x > 1    x > y    y >= 3
x == y    x != 2    2 < x < 5
```
The result of a comparison expression is a `bool` value

\[ a = \text{True} \quad b = \text{False} \]

\[ \text{not b} \quad a \text{ or b} \quad a \text{ and not b} \]

\[ a \text{ and b} \quad \text{not (a or b)} \quad b \text{ and b} \]
The result of a comparison expression is a **bool** value

- \( a = \text{True} \)
- \( b = \text{False} \)

- \( \text{not } b \) - Evaluate to True
- \( a \text{ or } b \)
- \( a \text{ and not } b \)
- \( a \text{ and } b \)
- \( \text{not } (a \text{ or } b) \) - Evaluate to False
- \( b \text{ and } b \)
Aggregating Comparisons

Summing an array or list of bool values count the number of True values

\[1 + 0 + 1 = 2\]
\[True + False + True = 2\]
\[\text{sum([1, 0, 1])} = 2\]
\[\text{sum([True, False, True])} = 2\]
Control Statements
These statements control the sequence of computations that are performed

- The keywords if and for begin control statements
- The purpose of if is to define functions that choose different behavior based on their arguments
Random Selection
Random Selection

np.random.choice
- Selects at random
- With replacement
- From an array
- A specific number of times

np.random.choice(some_array, sample_size)
A Longer Array

- `np.append(array_1, value):`
  - new array with value appended to `array_1`
  - value has to be of the same type as elements of `array_1`

- `np.append(array_1, array_2):`
  - new array with `array_2` appended to `array_1`
  - Elements of `array_2` have to be of the same type as elements of `array_1`
for statements

- `for` is a keyword that begins a control statement
- The purpose of `for` is to perform a computation for every element in a list or array