CS134: Tuples & Sorting
Announcements & Logistics

- **HW 5** due Monday at 10pm - last HW before midterm
- **Lab 4**
  - Part 1: Feedback returned this afternoon
  - Part 2: Due next Wed/Thur at 10 pm
- **Midterm reminder**: Thur Oct 20: 6-7:30 pm and 8-9:30 pm
- **Midterm review**: Tue Oct 18: 8-9:30 pm
  - Midterm practice problems will be released soon
- **Student help hours** for next week are now posted on webpage
  - Lots of hours on Tue, Wed, and Thur!
  - If the CS common room gets too crowded, we’ll move to TCL 217A/216
Student Help Hours
Next Week

CS 134: Fall 2022

<table>
<thead>
<tr>
<th>Time</th>
<th>Monday</th>
<th>Tuesday</th>
<th>Wednesday</th>
<th>Thursday</th>
<th>Friday</th>
</tr>
</thead>
<tbody>
<tr>
<td>8am</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9am</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10am</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11am</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12pm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1pm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2pm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3pm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4pm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5pm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6pm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7pm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8pm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9pm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10pm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notes:
- Iris Lecture Schow 030 A
- Jeannie Lecture Schow 030 A
- Lida Student Help Hours CS Common Room
- TA Hours TCL 217A
Looking Ahead

• **No HW posted next week**
  • We’ll post practice midterm questions instead

• **Lab on Oct 17/18**
  • Short lab on debugging strategies
  • Start and finish during scheduled lab session!
  • No need to start in advance

• **Things to review in preparation for the midterm**
  • Review lab solutions and HW questions
  • Review Jupyter notebooks and slides
  • Discuss practice midterm questions

• **No class on Fri Oct 21** (regardless of Mountain Day)
Last Time

- Learned about **aliasing** in Python
  - Need to be careful with aliasing when using lists due to mutability
- Discussed ways to create "new" lists (true copies):
  
  ```
  newList = myList[:]  # slicing
  newList = [el for el in myList]  # list comprehension
  ```
- Discussed while loops
  - Needed for ranked-choice voting on Lab 4 Part 2
Recap: Loops

1. Initialize a variable used in the test condition
2. Keyword that indicates the beginning of the loop
3. Test condition that causes the loop to end when False
4. Colon that indicates the end of the loop definition
5. Within the loop body (indented!), update the variable used in the test condition

```python
def printHalves(n):
    while n > 0:
        print(n)
        n = n // 2
```

- Initialize a variable for test condition
- Colon
- Test condition causing loop to end
- Update test condition variable in loop body
- Keyword for beginning of loop
Today’s Plan

• Today we will discuss a new *immutable* sequence: *tuples*
• Revisit sorting and default sorting behavior
• Discuss how we can override the default sorting behavior
Tuples: An Immutable Sequence

- Tuples are an **immutable sequence of values** (almost like immutable lists) separated by commas and enclosed within parentheses ().

```python
# string tuple
>>> names = ("Jeannie", "Iris", "Lida")

# int tuple
>>> primes = (2, 3, 5, 7, 11)

# singleton
>>> num = (5, )  # A tuple of size 1 is called a singleton. Note the (funky) syntax.

# parentheses are optional
>>> values = 5, 6

# empty tuple
>>> emp = ()
```
Tuples as Immutable Sequences

- Tuples, like strings, support any sequence operation that does not involve mutation: e.g,
  - `len()`: function: returns number of elements in tuple
  - `[]`: indexing: access specific element
  - `+`, `*`: tuple concatenation
  - `[:]:`: slicing to return subset of tuple (as a new tuple)
  - `in` and `not in`: check membership
  - `for loop`: iterate over elements in tuple
Multiple Assignment and Unpacking

- Tuples support a simple syntax for assigning multiple values at once, and also for "unpacking" sequence values

```python
>>> a, b = 4, 7
# reverse the order of values in tuple
>>> b, a = a, b
# tuple assignment to “unpack” list elements
>>> cbInfo = ['Charlie Brown', 8, False]
>>> name, age, glasses = cbInfo
```

- Note that the preceding line is just a more concise way of writing:

```python
>>> name = cbInfo[0]
>>> age = cbInfo[1]
>>> glasses = cbInfo[2]
```
Multiple Return from Functions

- Tuples come in handy when returning multiple values from functions.

```python
# multiple return values as a tuple
def arithmetic(num1, num2):
    '''Takes two numbers and returns the sum and product'''
    return num1 + num2, num1 * num2

>>> arithmetic(10, 2)
(12, 20)

>>> type(arithmetic(3, 4))
<class 'tuple'>
```
Conversion between Sequences

- The functions `tuple()`, `list()` , and `str()` convert between sequences

```python
>>> word = "Williamstown"
>>> charList = list(word)  # string to list
>>> charList
['W', 'i', 'l', 'l', 'i', 'a', 'm', 's', 't', 'o', 'w', 'n']
>>> charTuple = tuple(charList)  # list to tuple
>>> charTuple
('W', 'i', 'l', 'l', 'i', 'a', 'm', 's', 't', 'o', 'w', 'n')
>>> list((1, 2, 3, 4, 5))  # tuple to list
[1, 2, 3, 4, 5]
```
Conversion between Sequences

• The functions `tuple()`, `list()`, and `str()` convert between sequences

```python
>>> str(('hello', 'world'))  # tuple to string
"('hello', 'world')"

>>> numRange = range(12)

>>> list(numRange)  # range to list
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

>>> str(list(numRange))  # range to list to string
'[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]'
```

• See Jupyter for more examples
Sorting Tuples & More
**sorted()**

- **sorted()** is a built-in Python function (not a method!) that takes a sequence (string, list, tuple) and returns a *new sorted sequence as a list*

- By default, **sorted()** sorts the sequence in **ascending order** (for numbers) and alphabetical (dictionary) order for strings

- **sorted()** does not alter the sequence it is called on and always returns the type list

```
>>> nums = (42, -20, 13, 10, 0, 11, 18)  # tuple of ints
>>> sorted(nums)  # this returns a list!
[-20, 0, 10, 11, 13, 18, 42]
```

```
>>> letters = ('a', 'c', 'z', 'b', 'Z', 'A')
>>> sorted(letters)
['A', 'Z', 'a', 'b', 'c', 'z']
```
• **sorted**(string) returns a sorted **list** of **strings** (or more specifically, characters). It does not return a string!

```python
>>> sorted("Iris")
['I', 'i', 'r', 's']
>>> sorted("Jeannie")
['J', 'a', 'e', 'e', 'i', 'n', 'n']
>>> sorted("*hello!*")
['!', '*', '*', 'e', 'h', 'l', 'l', 'o']
```
Sorting Strings

- Strings are sorted based on the **ASCII values** of their characters
- ASCII stands for “American Standard Code for Information Interchange”
- Common character encoding scheme for electronic communication (that is, anything sent on the Internet!)
- Special characters come first, followed by capital letters, then lowercase letters
- Characters encoded using integers from 0–127
- Can use Python functions `ord()` and `chr()` to work with these:
  - `ord(str)`: takes a character and returns its ASCII value as `int`
  - `chr(int)`: takes an ASCII value as `int` and returns its corresponding character (`str`)
<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hex</th>
<th>Char</th>
<th>Decimal</th>
<th>Hex</th>
<th>Char</th>
<th>Decimal</th>
<th>Hex</th>
<th>Char</th>
<th>Decimal</th>
<th>Hex</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>[NULL]</td>
<td>32</td>
<td>20</td>
<td>[SPACE]</td>
<td>64</td>
<td>40</td>
<td>@</td>
<td>96</td>
<td>60</td>
<td>`</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>[START OF HEADING]</td>
<td>33</td>
<td>21</td>
<td>!</td>
<td>65</td>
<td>41</td>
<td>A</td>
<td>97</td>
<td>61</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>[START OF TEXT]</td>
<td>34</td>
<td>22</td>
<td>&quot;</td>
<td>66</td>
<td>42</td>
<td>B</td>
<td>98</td>
<td>62</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>[END OF TEXT]</td>
<td>35</td>
<td>23</td>
<td>#</td>
<td>67</td>
<td>43</td>
<td>C</td>
<td>99</td>
<td>63</td>
<td>c</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>[END OF TRANSMISSION]</td>
<td>36</td>
<td>24</td>
<td>$</td>
<td>68</td>
<td>44</td>
<td>D</td>
<td>100</td>
<td>64</td>
<td>d</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>[ENQUIRY]</td>
<td>37</td>
<td>25</td>
<td>%</td>
<td>69</td>
<td>45</td>
<td>E</td>
<td>101</td>
<td>65</td>
<td>e</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>[ACKNOWLEDGE]</td>
<td>38</td>
<td>26</td>
<td>&amp;</td>
<td>70</td>
<td>46</td>
<td>F</td>
<td>102</td>
<td>66</td>
<td>f</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>[BELL]</td>
<td>39</td>
<td>27</td>
<td>'</td>
<td>71</td>
<td>47</td>
<td>G</td>
<td>103</td>
<td>67</td>
<td>g</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>[BACKSPACE]</td>
<td>40</td>
<td>28</td>
<td>(</td>
<td>72</td>
<td>48</td>
<td>H</td>
<td>104</td>
<td>68</td>
<td>h</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>[HORIZONTAL TAB]</td>
<td>41</td>
<td>29</td>
<td>)</td>
<td>73</td>
<td>49</td>
<td>I</td>
<td>105</td>
<td>69</td>
<td>i</td>
</tr>
<tr>
<td>11</td>
<td>B</td>
<td>[VERTICAL TAB]</td>
<td>43</td>
<td>2B</td>
<td>+</td>
<td>75</td>
<td>4B</td>
<td>K</td>
<td>107</td>
<td>6B</td>
<td>k</td>
</tr>
<tr>
<td>12</td>
<td>C</td>
<td>[FORM FEED]</td>
<td>44</td>
<td>2C</td>
<td>,</td>
<td>76</td>
<td>4C</td>
<td>L</td>
<td>108</td>
<td>6C</td>
<td>l</td>
</tr>
<tr>
<td>13</td>
<td>D</td>
<td>[CARRIAGE RETURN]</td>
<td>45</td>
<td>2D</td>
<td>.</td>
<td>77</td>
<td>4D</td>
<td>M</td>
<td>109</td>
<td>6D</td>
<td>m</td>
</tr>
<tr>
<td>14</td>
<td>E</td>
<td>[SHIFT OUT]</td>
<td>46</td>
<td>2E</td>
<td>,</td>
<td>78</td>
<td>4E</td>
<td>N</td>
<td>110</td>
<td>6E</td>
<td>n</td>
</tr>
<tr>
<td>15</td>
<td>F</td>
<td>[SHIFT IN]</td>
<td>47</td>
<td>2F</td>
<td>/</td>
<td>79</td>
<td>4F</td>
<td>O</td>
<td>111</td>
<td>6F</td>
<td>o</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>[DATA LINK ESCAPE]</td>
<td>48</td>
<td>30</td>
<td>0</td>
<td>80</td>
<td>50</td>
<td>P</td>
<td>112</td>
<td>70</td>
<td>p</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>[DEVICE CONTROL 1]</td>
<td>49</td>
<td>31</td>
<td>1</td>
<td>81</td>
<td>51</td>
<td>Q</td>
<td>113</td>
<td>71</td>
<td>q</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>[DEVICE CONTROL 2]</td>
<td>50</td>
<td>32</td>
<td>2</td>
<td>82</td>
<td>52</td>
<td>R</td>
<td>114</td>
<td>72</td>
<td>r</td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>[DEVICE CONTROL 3]</td>
<td>51</td>
<td>33</td>
<td>3</td>
<td>83</td>
<td>53</td>
<td>S</td>
<td>115</td>
<td>73</td>
<td>s</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>[DEVICE CONTROL 4]</td>
<td>52</td>
<td>34</td>
<td>4</td>
<td>84</td>
<td>54</td>
<td>T</td>
<td>116</td>
<td>74</td>
<td>t</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>[NEGATIVE ACKNOWLEDGE]</td>
<td>53</td>
<td>35</td>
<td>5</td>
<td>85</td>
<td>55</td>
<td>U</td>
<td>117</td>
<td>75</td>
<td>u</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>[SYNCHRONOUS IDLE]</td>
<td>54</td>
<td>36</td>
<td>6</td>
<td>86</td>
<td>56</td>
<td>V</td>
<td>118</td>
<td>76</td>
<td>v</td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>[ENG OF TRANS. BLOCK]</td>
<td>55</td>
<td>37</td>
<td>7</td>
<td>87</td>
<td>57</td>
<td>W</td>
<td>119</td>
<td>77</td>
<td>w</td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>[CANCEL]</td>
<td>56</td>
<td>38</td>
<td>8</td>
<td>88</td>
<td>58</td>
<td>X</td>
<td>120</td>
<td>78</td>
<td>x</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>[END OF MEDIUM]</td>
<td>57</td>
<td>39</td>
<td>9</td>
<td>89</td>
<td>59</td>
<td>Y</td>
<td>121</td>
<td>79</td>
<td>y</td>
</tr>
<tr>
<td>26</td>
<td>1A</td>
<td>[SUBSTITUTE]</td>
<td>58</td>
<td>3A</td>
<td>:</td>
<td>90</td>
<td>5A</td>
<td>Z</td>
<td>122</td>
<td>7A</td>
<td>z</td>
</tr>
<tr>
<td>27</td>
<td>1B</td>
<td>[ESCAPE]</td>
<td>59</td>
<td>3B</td>
<td>;</td>
<td>91</td>
<td>5B</td>
<td>[</td>
<td>123</td>
<td>7B</td>
<td>{</td>
</tr>
<tr>
<td>28</td>
<td>1C</td>
<td>[FILE SEPARATOR]</td>
<td>60</td>
<td>3C</td>
<td>&lt;</td>
<td>92</td>
<td>5C</td>
<td>\</td>
<td>124</td>
<td>7C</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>1D</td>
<td>[GROUP SEPARATOR]</td>
<td>61</td>
<td>3D</td>
<td>=</td>
<td>93</td>
<td>5D</td>
<td>]</td>
<td>125</td>
<td>7D</td>
<td>}</td>
</tr>
<tr>
<td>30</td>
<td>1E</td>
<td>[RECORD SEPARATOR]</td>
<td>62</td>
<td>3E</td>
<td>&gt;</td>
<td>94</td>
<td>5E</td>
<td>^</td>
<td>126</td>
<td>7E</td>
<td>~</td>
</tr>
<tr>
<td>31</td>
<td>1F</td>
<td>[UNIT SEPARATOR]</td>
<td>63</td>
<td>3F</td>
<td>?</td>
<td>95</td>
<td>5F</td>
<td>_</td>
<td>127</td>
<td>7F</td>
<td>[DEL]</td>
</tr>
</tbody>
</table>
An aside: sort() vs sorted()

- **sort() method** is **only for lists** and sorts by mutating the list in place; invoked using **dot notation**
- **sorted() function** can be used to **sort any sequence** (strings, lists, tuples). It always **returns a new sorted list**, and does NOT modify the original sequence

**Example:**

list1 = [6, 3, 4];  list2 = [6, 3, 4]

list1.sort()  # sort list1 by mutating values
sorted(list2)  # returns a *new* sorted list

<table>
<thead>
<tr>
<th>list1 Before</th>
<th>list1 After</th>
<th>list2 Before</th>
<th>list2 After</th>
</tr>
</thead>
<tbody>
<tr>
<td>[6, 3, 4]</td>
<td>[3, 4, 6]</td>
<td>[6, 3, 4]</td>
<td>[6, 3, 4]</td>
</tr>
</tbody>
</table>

Does not change!
Sorting Tuples and Lists

• Sorting a list of (or a tuple of) tuples with `sorted()` sorts elements in ascending order by their first item.

• If there is a tie, Python breaks the tie by comparing the second items.

• If the second items are also tied, it compares the third items, and so on.

```python
>>> fruits = [(12, 'apples'), (4, 'bananas'), (27, 'grapes')]
>>> sorted(fruits)
[(4, 'bananas'), (12, 'apples'), (27, 'grapes')]

>>> pairs = [(4, 5), (0, 2), (12, 1), (11, 3)]
>>> sorted(pairs)
[(0, 2), (4, 5), (11, 3), (12, 1)]
```

• Note: The same is true for lists and lists of lists.

• This sorting behavior is referred to as lexicographical sorting.
Sorting Tuples and Lists

• Sorting a list of (or a tuple of) tuples with `sorted()` sorts elements in ascending order by their first item.

• If there is a tie, Python breaks the tie by comparing the second items.

• If the second items are also tied, it compares the third items, and so on.

```python
>>> triples = [(1, 2, 3), (2, 2, 1), (1, 2, 1)]
>>> sorted(triples)
[(1, 2, 1), (1, 2, 3), (2, 2, 1)]
```

```python
>>> chars = [(8, 'a', '$'), (8, 'a', '!'), (7, 'c', '@')]
>>> sorted(chars)
[(7, 'c', '@'), (8, 'a', '!'), (8, 'a', '$')]
```

**Question:** How do we sort based on the second/third item in tuples? Or sort in reverse order?
Changing the Default Sorting Behavior

- To better understand the `sorted()` function, look at documentation

```
help(sorted)
```

Help on built-in function sorted in module builtins:

```
sorted(iterable, /, *, key=None, reverse=False)
    Return a new list containing all items from the iterable in ascending order.

    A custom key function can be supplied to customize the sort order, and the
    reverse flag can be set to request the result in descending order.
```

- An **iterable** is any object over which we can iterate (list, string, tuple, range)
- The optional parameter **key** specifies a function that determines how each
  element should be compared to other elements
- The optional boolean parameter **reverse** (which by default is set to `False`) allows us to sort in reverse order
- Note: the `.sort()` list method also supports these options
Reverse Sorting

• Let's consider the optional reverse parameter to sorted()

• Sort sequences in reverse order by setting this parameter to be True

```python
>>> fruits = [(12, 'apples'), (4, 'bananas'), (27, 'grapes')]
>>> sorted(fruits, reverse=True)
[(27, 'grapes'), (12, 'apples'), (4, 'bananas')]

>>> letters = ('a', 'c', 'z', 'b', 'Z', 'A')
>>> sorted(letters, reverse=True)
['z', 'c', 'b', 'a', 'Z', 'A']

>>> nums = (42, -20, 13, 10, 0, 11, 18)
>>> sorted(nums, reverse=True)
[42, 18, 13, 11, 10, 0, -20]
```
Sorting with a key Function
Sorting with a **key** function

- Now suppose we have a list of tuples that we want to sort by something other than the first item.
- Example: A list of course tuples, where the first item is the course name, second item is the enrollment cap, and third item is the term (Fall/Spring).

```python
courses = [('CS134', 90, 'Spring'), ('CS136', 60, 'Spring'), ('AFR206', 30, 'Spring'), ('ECON233', 30, 'Fall'), ('MUS112', 10, 'Fall'), ('STAT200', 50, 'Spring'), ('PSYC201', 50, 'Fall'), ('MATH110', 90, 'Spring')]
```

- Suppose we want to sort these courses by their **capacity** (second element).
- We can accomplish this by supplying the `sorted()` function with a `key` function that tells it how to compare the tuples to each other.
Sorting with a **key** function

- **Defining a key function explicitly:**
  - We can define an explicit **key** function that, when given a tuple, returns the parameter we want to sort the tuples with respect to.

```python
def capacity(courseTuple):
    '''Takes a sequence and returns item at index 1'''
    return courseTuple[1]

# we can tell sorted() to sort by capacity instead
sorted(courses, key=capacity)
```
Sorting with a **key** function

• **Defining a key function explicitly:**

  • We can define an explicit **key** function that, when given a tuple, returns the parameter we want to sort the tuples with respect to.

    ```python
    def capacity(courseTuple):
        '''Takes a sequence and returns item at index 1'''
        return courseTuple[1]
    # we can tell sorted() to sort by capacity instead
    sorted(courses, key=capacity)
    ```

    ```python
    courses = [('CS134', 90, 'Spring'), ('CS136', 60, 'Spring'), ('AFR206', 30, 'Spring'), ('ECON233', 30, 'Fall'), ('MUS112', 10, 'Fall'), ('STAT200', 50, 'Spring'), ('PSYC201', 50, 'Fall'), ('MATH110', 90, 'Spring')]
    # we can tell sorted() to sort by capacity instead
    sorted(courses, key=capacity)
    ```

    ```python
    [('MUS112', 10, 'Fall'), ('AFR206', 30, 'Spring'), ('ECON233', 30, 'Fall'), ('STAT200', 50, 'Spring'), ('PSYC201', 50, 'Fall'), ('CS136', 60, 'Spring'), ('CS134', 90, 'Spring'), ('MATH110', 90, 'Spring')]
    ```
Python Sorting is Stable

- Python's sorting functions are **stable**, which means that items that are equal according to the sorting **key** have the same relative order as in the original sequence.

```python
courses = [('CS134', 90, 'Spring'), ('CS136', 60, 'Spring'), ('AFR206', 30, 'Spring'), ('ECON233', 30, 'Fall'), ('MUS112', 10, 'Fall'), ('STAT200', 50, 'Spring'), ('PSYC201', 50, 'Fall'), ('MATH110', 90, 'Spring')]

def term(courseTuple):
    ''' Takes a sequence and returns item at index 2 '''
    return courseTuple[2]

sorted(courses, key=term)

[['ECON233', 30, 'Fall'],
 ('MUS112', 10, 'Fall'),
 ('PSYC201', 50, 'Fall'),
 ('CS134', 90, 'Spring'),
 ('CS136', 60, 'Spring'),
 ('AFR206', 30, 'Spring'),
 ('STAT200', 50, 'Spring'),
 ('MATH110', 90, 'Spring')]
```

Notice the ordering of courses with Fall term and those with Spring term.
Takeaways

• Tuples are a new immutable sequence that
  • supports all sequence operations such as indexing and slicing
  • are useful for argument unpacking, multiple assignments
  • are useful for handling list-like data without aliasing issues

• `sorted()` function and `.sort()` list method sorts sequences in ascending and lexicographic order by default

• We can override the default sorting behavior by supplying optional parameters `key` (function), and `reverse` (Boolean)
The end!