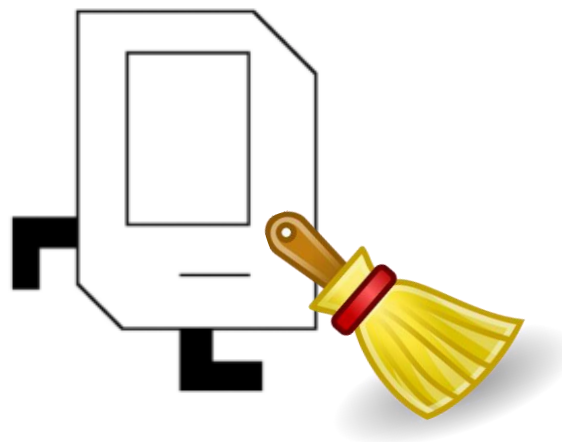




Tuples + Sorting

Chris Piech and Mehran Sahami
CS106A, Stanford University

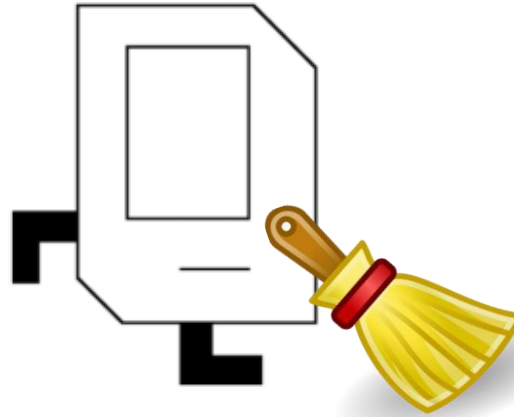
Housekeeping I



- Assignment # 5 due today
- Assignment # 6 out today (due Mon., Nov. 9th)
- 2nd diagnostic assessment on Mon, Nov. 2nd at 1pm PST
 - Exam file available to download shortly before start time
 - Daylight savings time ends 2am on Sun., Nov. 1st ("Fall back")
 - Diagnostic covers through lecture on Oct. 26th
 - Practice diagnostic available on website
- No LaIR on Tues., Nov 3rd. (vote!, work polls, etc.)



Housekeeping II



- Apply for CS51: CS + Social Good Studio: Designing Social Impact Projects
 - 2 unit class in Winter quarter
 - Work with non-profit partner organization to build app
 - More information at: <https://tinyurl.com/cs-studio-info>
 - Application: <http://www.tinyurl.com/cs-studio-apply>
 - Deadline: Nov. 6th at 11:59pm (PST)



Learning Goals

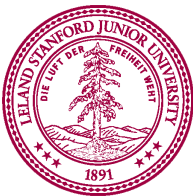
1. Learning about tuples in Python
2. Writing code using tuples
3. Learning about sorting



Tuples

What is a Tuple?

- A **tuple** is way to keep track of an *ordered collection* of items
 - Similar to a list, but immutable (can't be changed in place)
 - Ordered: can refer to elements by their position
 - Collection: list can contain multiple items
- Often used to keep track of data that are *conceptually related*, such as
 - Coordinates for a *point*: (x, y)
 - RGB values for a *color*: (red, green, blue)
 - Elements of an *address*: (street, city, state, zipcode)
- Can be used to return multiple values from a function



Show Me the Tuples!

- Creating tuples
 - Tuples start/end with parentheses. Elements separated by commas.

```
my_tuple = (1, 2, 3)
```

```
point = (4.7, -6.0)
```

```
strs = ('strings', 'in', 'tuple')
```

```
addr = ('102 Ray Ln', 'Stanford', 'CA', 94305)
```

```
empty_tuple = ()
```

- Tuple with one element has a comma (to denote tuple)

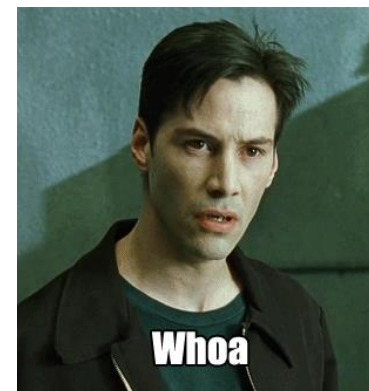
- Could try this out on the console:

```
>>> tuple_one = (1,)
```

```
>>> one = 1
```

```
>>> tuple_one == one
```

```
False
```

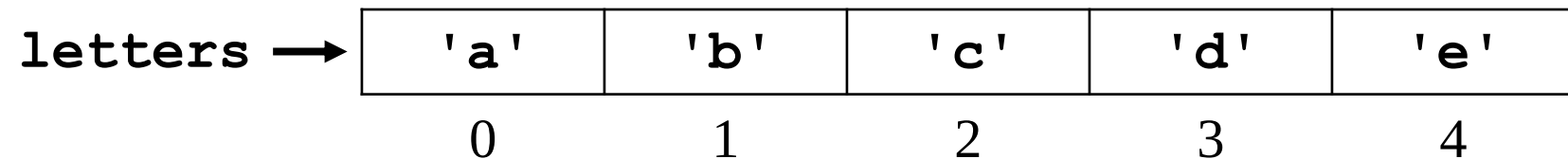


Accessing Elements of Tuple

- Consider the following tuple:

```
letters = ('a', 'b', 'c', 'd', 'e')
```

- Access elements of tuple just like a list:
 - Indexes start from 0



- Access individual elements:

```
letters[0] is 'a'
```

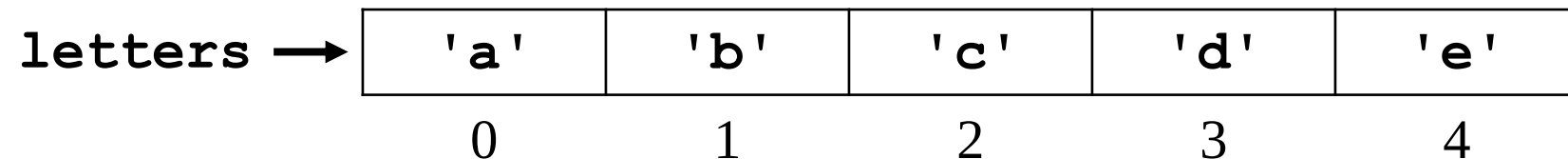
```
letters[4] is 'e'
```


Accessing Elements of Tuple

- Consider the following tuple:

```
letters = ('a', 'b', 'c', 'd', 'e')
```

- Access elements of tuple just like a list:
 - Indexes start from 0



- Cannot** assign to individual elements:

- Tuples are immutable

```
letters[0] = 'x'
```

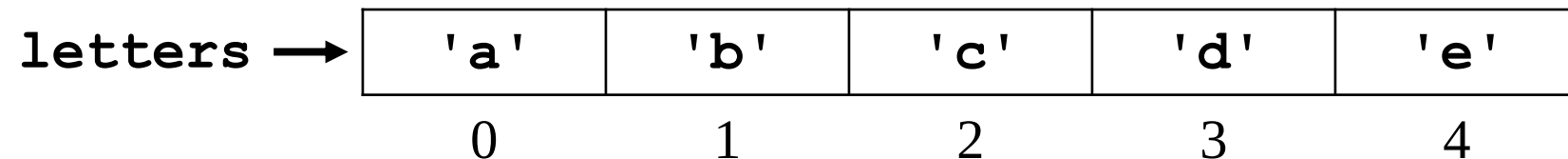
TypeError: 'tuple' object does not support item assignment

Accessing Elements of Tuple

- Consider the following tuple:

```
letters = ('a', 'b', 'c', 'd', 'e')
```

- Access elements of tuple just like a list:
 - Indexes start from 0



- Cannot** assign to individual elements:
 - Tuples are immutable
 - Also, there are no **append/pop** functions for tuples
 - Tuples cannot be changed in place
 - To change, need to create new tuple and overwrite variable

Getting Length of a Tuple

- Consider the following tuple:

```
letters = ('a', 'b', 'c', 'd', 'e')
```

- Can get length of tuple with `len` function:

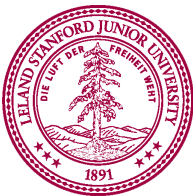
```
len(letters) is 5
```

– Elements of tuple are indexed from 0 to length – 1

- Using length to loop through a tuple:

```
for i in range(len(letters)):  
    print(str(i) + " -> " + letters[i])
```

```
0 -> a  
1 -> b  
2 -> c  
3 -> d  
4 -> e
```



Indexes and Slices

- Consider the following tuple:

```
letters = ('a', 'b', 'c', 'd', 'e')
```

- Negative indexes in tuple work just the same as lists
 - Work back from end of tuple
 - Example:

```
letters[-1] is 'e'
```

- Slices work on tuples same way as lists (but get tuple)

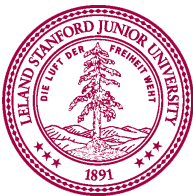
```
>>> aslice = letters[2:4]
```

```
>>> aslice
```

```
('c', 'd')
```

aslice →

'c'	'd'
0	1



Good Times with Tuples

- More tuple examples:

```
chartreuse_rgb = (127, 255, 0)
```

```
stanford = ('450 Serra Mall', 'Stanford', 'CA', 94305)
```

- Printing tuples:

```
>>> print(chartreuse_rgb)
```

```
(127, 255, 0)
```

```
>>> print(stanford)
```

```
('450 Serra Mall', 'Stanford', 'CA', 94305)
```

- Check if tuple is empty (empty tuple is like "False")

```
if stanford:
```

```
    print('stanford is not empty')
```

```
else:
```

```
    print('stanford is empty')
```



More Good Times with Tuples

- More tuple examples:

```
chartreuse_rgb = (127, 255, 0)
```

```
stanford = ('450 Serra Mall', 'Stanford', 'CA', 94305)
```

- Check to see if a tuple contains an element:

```
state = 'CA'
```

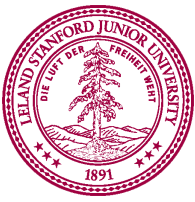
```
if state in stanford:
```

```
    # do something
```

- General form of test (evaluates to a Boolean):

element in tuple

- Returns **True** if *element* is a value in *tuple*, **False** otherwise
- Can also test if element is not in tuple using **not in**



A Few Tuple Functions

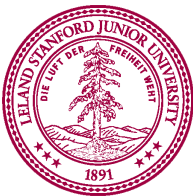
```
chartreuse_rgb = (127, 255, 0)
```

- Function: `max(chartreuse_rgb)`
 - Returns maximal value in the tuple

```
>>> max(chartreuse_rgb)
255
```
- Function: `min(chartreuse_rgb)`
 - Returns minimal value in the tuple

```
>>> min(chartreuse_rgb)
0
```
- Function: `sum(chartreuse_rgb)`
 - Returns sum of the values in the tuple

```
>>> sum(chartreuse_rgb)
382
```



Looping Through Tuple Elements

```
stanford = ('450 Serra Mall', 'Stanford', 'CA', 94305)
```

- For loop using `range`:

```
for i in range(len(stanford)) :  
    elem = stanford[i]  
    print(elem)
```

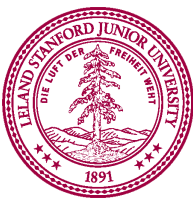
Output:

```
450 Serra Mall  
Stanford  
CA  
94305
```

- For-each loop:

```
for elem in stanford:  
    print(elem)
```

- These loops both iterate over all elements of the tuple
 - Variable `elem` is set to each value in tuple (in order)
 - Works just the same as iterating through a list



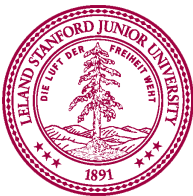
Tuples as Parameters

- When you pass a tuple as a parameter, think of it like passing an integer or a string
 - Tuple is immutable, so changes in function do not persist!

```
def remove_red(rgb_tuple):  
    rgb_tuple = (0, rgb_tuple[1], rgb_tuple[2])  
    print("In remove_red: " + str(rgb_tuple))  
  
def main():  
    chartreuse_rgb = (127, 255, 0)  
    remove_red(chartreuse_rgb)  
    print("In main: " + str(chartreuse_rgb))
```

Output:

<pre>In remove_red: (0, 255, 0) In main: (127, 255, 0)</pre>
--



Assignment with Tuples

- Can use tuples to assign multiple variables at once:
 - Number of variables on left-hand side of assignment needs to be the same as the size of the tuple on the right-hand side

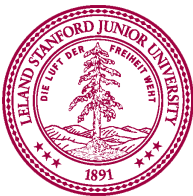
```
>>> (x, y) = (3, 4)
```

```
>>> x
```

```
3
```

```
>>> y
```

```
4
```



Returning Tuples from Functions

- Can use tuples to return multiple values from function
 - Stylistic point: values returned should make sense as something that is grouped together (e.g., (x, y) coordinate)

```
def get_date():  
    day = int(input("Day (DD): "))  
    month = int(input("Month (MM): "))  
    year = int(input("Year (YYYY): "))  
    return day, month, year  
  
def main():  
    (dd, mm, yyyy) = get_date()  
    print(str(mm) + "/" + str(dd) + "/" + str(yyyy))
```

Terminal:

```
Day (DD): 10  
Month (MM): 05  
Year (YYYY): 1970  
5/10/1970
```

Returning Tuples from Functions

- Can use tuples to return multiple values from function
 - Stylistic point: values returned should make sense as something that is grouped together (e.g., (x, y) coordinate)

```
def get_date():  
    day = int(input("Day (DD): "))  
    month = int(input("Month (MM): "))  
    year = int(input("Year (YYYY): "))  
    return day, month, year  
  
def main():  
    (dd, mm, yyyy) = get_date()  
    print(str(mm) + "/" + str(dd) + "/" + str(yyyy))
```

- Note: all paths through a function should return a tuple of the same length, otherwise program might crash
- For functions that return tuples, comment should specify the number of return values (and their types)

Tuples and Lists

- Can create lists from tuples using `list` function:

```
>>> my_tuple = (10, 20, 30, 40, 50)
>>> my_list = list(my_tuple)
>>> my_list
[10, 20, 30, 40, 50]
```

- Can create tuples from lists using `tuple` function:

```
>>> a_list = ['need', 'to', 'vote', 'in', 2020]
>>> a_tuple = tuple(a_list)
>>> a_tuple
('need', 'to', 'vote', 'in', 2020)
```

Tuples and Dictionaries

- Can get key/value pairs from dictionaries as tuples using the `items` functions:

```
>>> dict = {'a':1, 'b':2, 'c':3, 'd':4}
>>> list(dict.items())
[('a', 1), ('b', 2), ('c', 3), ('d', 4)]
```

- Can loop through key/value pairs as tuples:

```
for key, value in dict.items():
    print(str(key) + " -> " + str(value))
```

Output:

a	->	1
b	->	2
c	->	3
d	->	4

Tuples in Dictionaries

- Can use tuples as keys in dictionaries:

```
>>> dict = {('a', 1): 10, ('b', 1): 20, ('a', 2): 30}
>>> list(dict.keys())
[('a', 1), ('b', 1), ('a', 2)]
>>> list(dict.values())
[10, 20, 30]
```

- Can use tuples as values in dictionaries:

```
>>> colors = { 'orange': (255, 165, 0),
                'yellow': (255, 255, 0),
                'aqua':    (0, 128, 128)  }
>>> list(colors.values())
[(255, 165, 0), (255, 255, 0), (0, 128, 128)]
>>> list(colors.keys())
['orange', 'yellow', 'aqua']
```

Putting it all together:
colors.py

Sorting

Basic Sorting

- The `sorted` function orders elements in a collection in increasing (non-decreasing) order
 - Can sort any type that support `<` and `==` operations
 - For example: int, float, string
 - `sorted` returns new collection (original collection unchanged)

```
>>> nums = [8, 42, 4, 8, 15, 16]
```

```
>>> sorted_list = sorted(nums)
```

```
>>> sorted_list
```

```
[4, 8, 8, 15, 16, 42]
```

```
>>> nums
```

```
[8, 42, 4, 8, 15, 16]      # original list not changed
```

```
>>> strs = ['banana', 'zebra', 'apple', 'donut']
```

```
>>> sorted(strs)
```

```
['apple', 'banana', 'donut', 'zebra']
```



Intermediate Sorting

- Can sort elements in decreasing (non-increasing) order
 - Use the optional parameter `reverse=True`

```
>>> nums = [8, 42, 4, 8, 15, 16]
```

```
>>> sorted(nums, reverse=True)
```

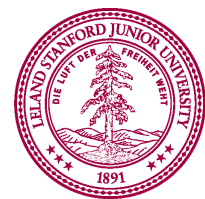
```
[42, 16, 15, 8, 8, 4]
```

```
>>> strs = ['banana', 'CHERRY', 'apple', 'donut']
```

```
>>> sorted(strs, reverse=True)
```

```
['donut', 'banana', 'apple', 'CHERRY']
```

- Note case sensitivity of sorting strings!
 - Any uppercase letter is less than any lowercase letter
 - For example: `'z' < 'a'`



Advanced Sorting

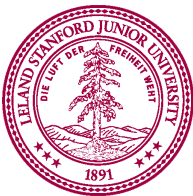
- Sorting using a custom function
 - Use the optional parameter `key=<function name>`

```
def get_len(s):  
    return len(s)
```

```
def main():  
    strs = ['a', 'bbbb', 'cc', 'zzz']  
    sorted_strs = sorted(strs, key=get_len)  
    print(sorted_strs)
```

Output:

```
['a', 'cc', 'zzz', 'bbbb']
```



Super Deluxe Advanced Sorting

- Sorting a list of tuples with a custom function
 - Use the optional parameter **key=<function name>**

```
def get_count(food):  
    return food[1]  
  
def main():  
    foods = [('apple', 5), ('banana', 2), ('chocolate', 137)]  
    sort_names = sorted(foods)  
    print(sort_names)  
    sort_count = sorted(foods, key=get_count)  
    print(sort_count)  
    rev_sort_count = sorted(foods, key=get_count, reverse=True)  
    print(rev_sort_count)
```

Output:

```
[('apple', 5), ('banana', 2), ('chocolate', 137)]  
[('banana', 2), ('apple', 5), ('chocolate', 137)]  
[('chocolate', 137), ('apple', 5), ('banana', 2)]
```

Learning Goals

1. Learning about tuples in Python
2. Writing code using tuples
3. Learning about sorting





Yes, that's in sorted order!