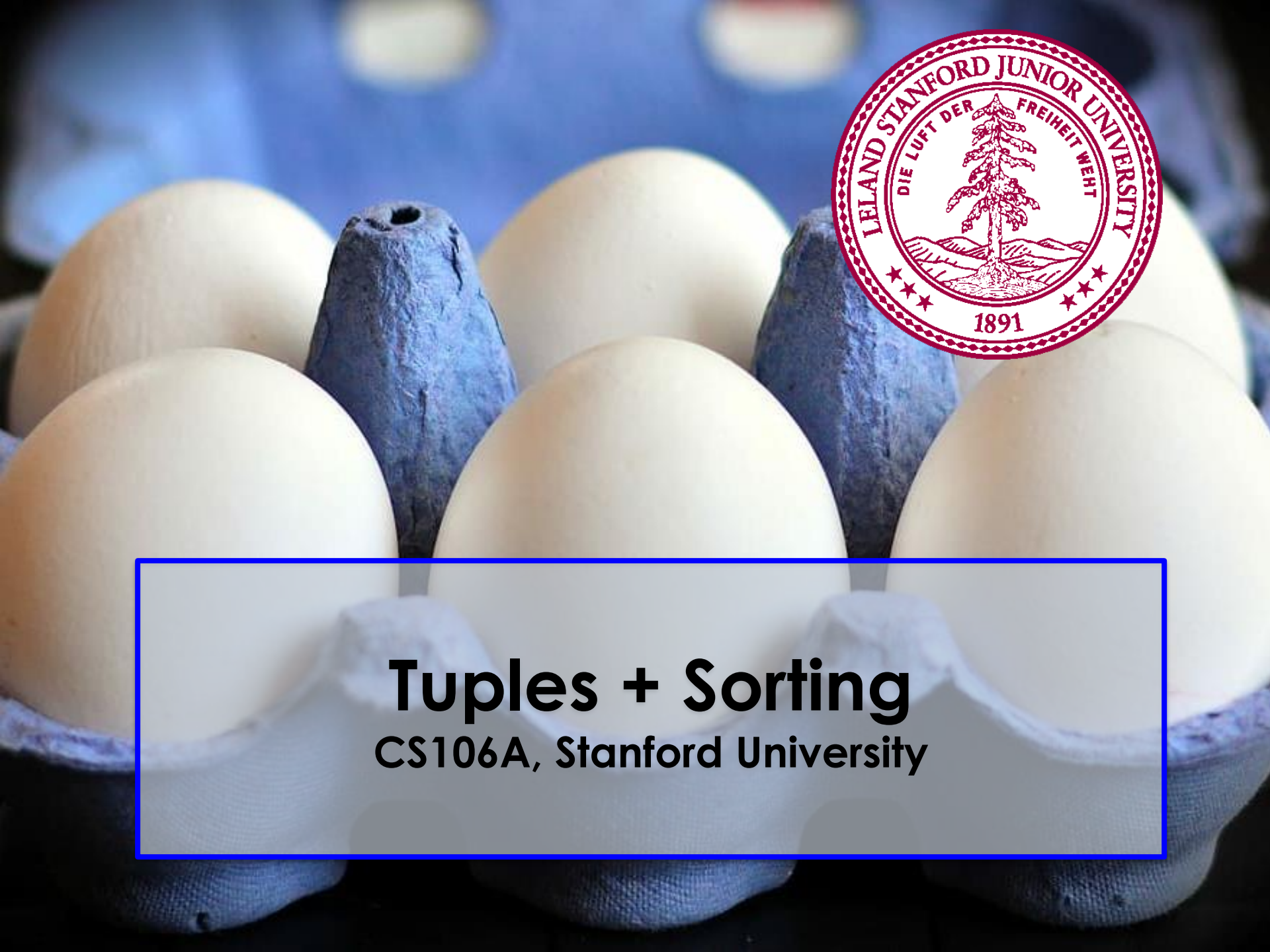
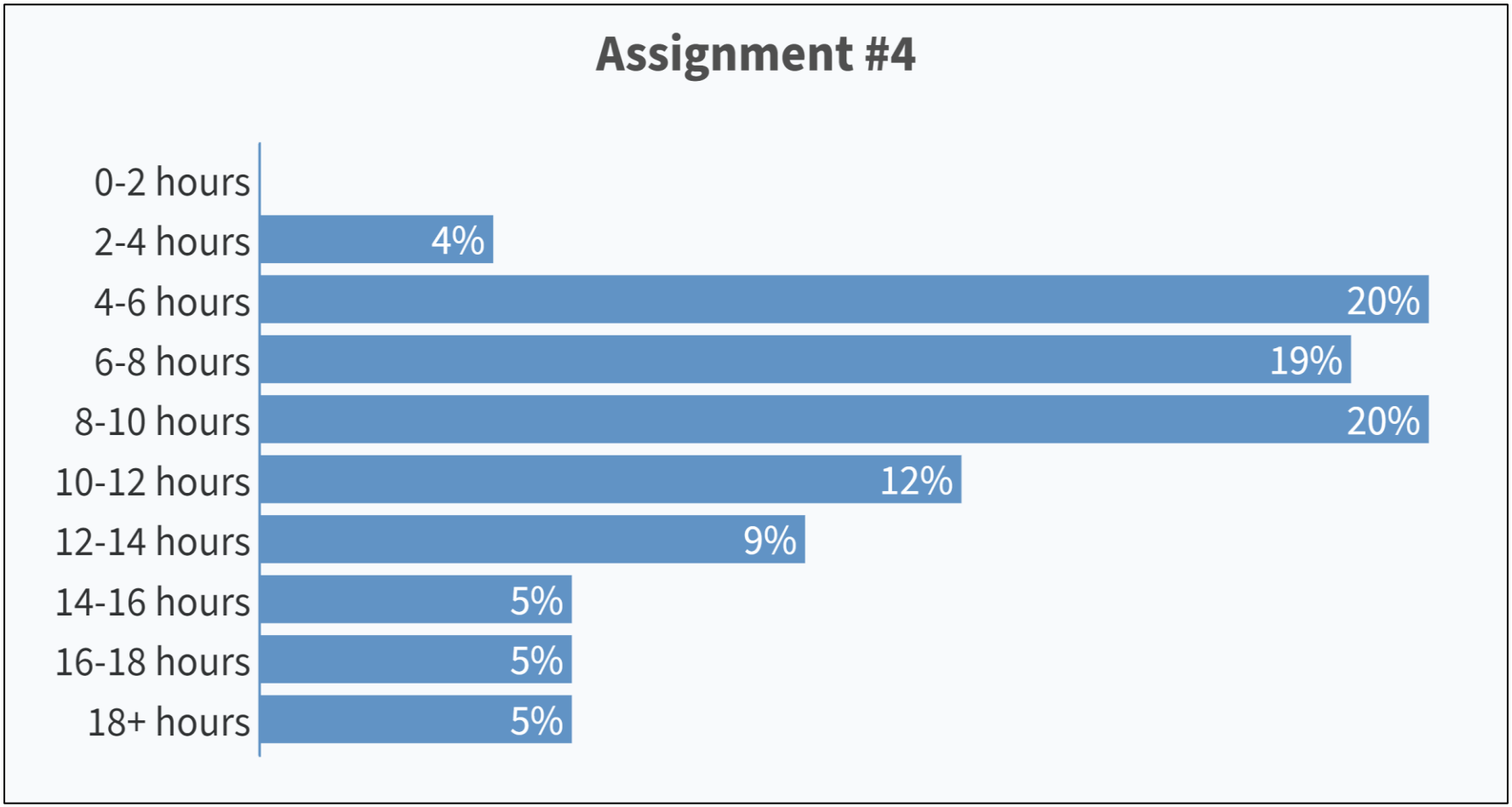


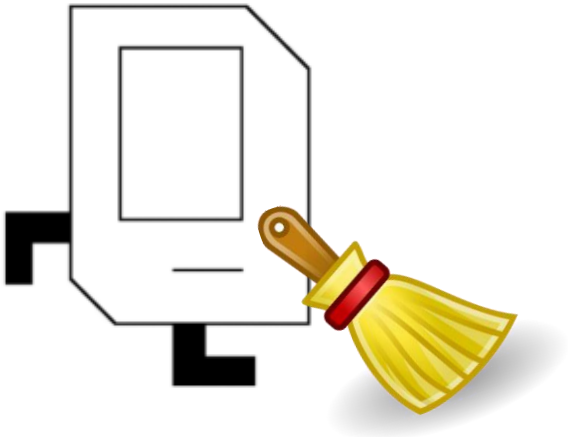
Tuples + Sorting

CS106A, Stanford University

Assignment #4 Poll



Housekeeping



- Midterm regrade requests
  - Open until 11:59pm (Pacific Time) on Monday, May 16th
  - Request regrades through Gradescope
  - Please add note explaining what you believe was misgraded
  - We reserve the right to regrade the entire exam
- Graphics/image contest (OPTIONAL) released today
  - Due Friday, May 27th
  - Just for fun and fabulous prizes!

Review: parsing strings

Time for
musicdatabase.py

Reading File

music.txt

```
[Song] [Artist] [Album] [Genre]
[Billie Jean] [Michael Jackson] [Thriller] [Pop]
[Up] [Cardi B] [] [Hip Hop]
[De Do Do Do, De Da Da Da] [The Police] [Zenyatta Mondatta] [Rock]
[Shivers] [Ed Sheeran] [=] [Pop]
[WusYaName] [Tyler, the Creator] [Call Me If You Get Lost] [Hip Hop]
```

```
def load_music_data(filename):
    music_data = {}
    with open(filename) as file:
        next(file)                      # Skip header line
        for line in file:
            line = line.strip()
            parts = parse_line(line)
            key = parts[0]              # Key is song name
            value = {                   # Value is dictionary of song data
                'Artist': parts[1],
                'Album': parts[2],
                'Genre': parts[3]
            }
            music_data[key] = value
        return music_data
```

Reading File

music.txt

```
[Song] [Artist] [Album] [Genre]
[Billie Jean] [Michael Jackson] [Thriller] [Pop]
[Up] [Cardi B] [] [Hip Hop]
[De Do Do Do, De Da Da Da] [The Police] [Zenyatta Mondatta] [Rock]
[Shivers] [Ed Sheeran] [=] [Pop]
[WusYaName] [Tyler, the Creator] [Call Me If You Get Lost] [Hip Hop]
```

```
def load_music_data(filename):
    music_data = {}
    with open(filename) as file:
        next(file)                      # Skip header line
        for line in file:
            line = line.strip()
            parts = parse_line(line)
            key = parts[0]              # Key is song name
            value = {                   # Value is dictionary of song data
                'Artist': parts[1],
                'Album': parts[2],
                'Genre': parts[3]
            }
            music_data[key] = value
        return music_data
```

Parsing with find()

line

```
[Billie Jean] [Michael Jackson] [Thriller] [Pop]
```

```
NUM_COMPONENTS = 4

def parse_line(line):
    list = []
    end = 0
    for i in range(NUM_COMPONENTS):
        start = line.find('[', end) + 1
        end = line.find(']', start)
        list.append(line[start:end])
    return list
```

Parsing with find()

line

```
[Billie Jean] [Michael Jackson] [Thriller] [Pop]
```

list `[]`

```
NUM_COMPONENTS = 4

def parse_line(line):
    list = []
    end = 0
    for i in range(NUM_COMPONENTS):
        start = line.find('[', end) + 1
        end = line.find(']', start)
        list.append(line[start:end])
    return list
```

Parsing with find()

line

```
[Billie Jean] [Michael Jackson] [Thriller] [Pop]
```

list `[]`

end `0`

```
NUM_COMPONENTS = 4

def parse_line(line):
    list = []
    end = 0
    for i in range(NUM_COMPONENTS):
        start = line.find('[', end) + 1
        end = line.find(']', start)
        list.append(line[start:end])
    return list
```

Parsing with find()

line

```
[Billie Jean] [Michael Jackson] [Thriller] [Pop]
```

list `[]`

i `0`

end `0`

```
NUM_COMPONENTS = 4

def parse_line(line):
    list = []
    end = 0
    for i in range(NUM_COMPONENTS):
        start = line.find('[', end) + 1
        end = line.find(']', start)
        list.append(line[start:end])
    return list
```

Parsing with find()

line

```
[Billie Jean] [Michael Jackson] [Thriller] [Pop]
```

list `[]`

start `1`

end `0`

i `0`

```
NUM_COMPONENTS = 4

def parse_line(line):
    list = []
    end = 0
    for i in range(NUM_COMPONENTS):
        start = line.find('[', end) + 1
        end = line.find(']', start)
        list.append(line[start:end])
    return list
```

Parsing with find()

line

```
[Billie Jean] [Michael Jackson] [Thriller] [Pop]
```

list `[]`

start `1`

end `12`

i `0`

```
NUM_COMPONENTS = 4

def parse_line(line):
    list = []
    end = 0
    for i in range(NUM_COMPONENTS):
        start = line.find('[', end) + 1
        end = line.find(']', start)
        list.append(line[start:end])
    return list
```

Parsing with find()

line

```
[Billie Jean] [Michael Jackson] [Thriller] [Pop]
```

list `['Billie Jean']`

start `1`

end `12`

i `0`

```
NUM_COMPONENTS = 4

def parse_line(line):
    list = []
    end = 0
    for i in range(NUM_COMPONENTS):
        start = line.find('[', end) + 1
        end = line.find(']', start)
        list.append(line[start:end])
    return list
```

Parsing with find()

line

```
[Billie Jean] [Michael Jackson] [Thriller] [Pop]
```

list `['Billie Jean']`

start `1`

end `12`

i `1`

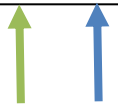
```
NUM_COMPONENTS = 4

def parse_line(line):
    list = []
    end = 0
    for i in range(NUM_COMPONENTS):
        start = line.find('[', end) + 1
        end = line.find(']', start)
        list.append(line[start:end])
    return list
```

Parsing with find()

line

```
[Billie Jean] [Michael Jackson] [Thriller] [Pop]
```



list `['Billie Jean']`

start `15`

end `12`

i `1`

```
NUM_COMPONENTS = 4

def parse_line(line):
    list = []
    end = 0
    for i in range(NUM_COMPONENTS):
        start = line.find('[', end) + 1
        end = line.find(']', start)
        list.append(line[start:end])
    return list
```

Parsing with find()

line

```
[Billie Jean] [Michael Jackson] [Thriller] [Pop]
```

list `['Billie Jean']`

start `15`

end `30`

i `1`

```
NUM_COMPONENTS = 4

def parse_line(line):
    list = []
    end = 0
    for i in range(NUM_COMPONENTS):
        start = line.find('[', end) + 1
        end = line.find(']', start)
        list.append(line[start:end])
    return list
```

Parsing with find()

line

```
[Billie Jean] [Michael Jackson] [Thriller] [Pop]
```

list `['Billie Jean', 'Michael Jackson']`

start `15`

end `30`

i `1`

```
NUM_COMPONENTS = 4

def parse_line(line):
    list = []
    end = 0
    for i in range(NUM_COMPONENTS):
        start = line.find('[', end) + 1
        end = line.find(']', start)
        list.append(line[start:end])
    return list
```

Parsing with find()

line

```
[Billie Jean] [Michael Jackson] [Thriller] [Pop]
```

list `['Billie Jean', 'Michael Jackson', 'Thriller']`

start `33`

end `41`

i `2`

```
NUM_COMPONENTS = 4

def parse_line(line):
    list = []
    end = 0
    for i in range(NUM_COMPONENTS):
        start = line.find('[', end) + 1
        end = line.find(']', start)
        list.append(line[start:end])
    return list
```

Parsing with find()

line

```
[Billie Jean] [Michael Jackson] [Thriller] [Pop]
```

list `['Billie Jean', 'Michael Jackson', 'Thriller', 'Pop']`

start `44`

end `47`

i `3`

```
NUM_COMPONENTS = 4

def parse_line(line):
    list = []
    end = 0
    for i in range(NUM_COMPONENTS):
        start = line.find('[', end) + 1
        end = line.find(']', start)
        list.append(line[start:end])
    return list
```

Parsing with find()

line

```
[Billie Jean] [Michael Jackson] [Thriller] [Pop]
```

list `['Billie Jean', 'Michael Jackson', 'Thriller', 'Pop']`

start `44`

end `47`

i `3`

```
NUM_COMPONENTS = 4

def parse_line(line):
    list = []
    end = 0
    for i in range(NUM_COMPONENTS):
        start = line.find('[', end) + 1
        end = line.find(']', start)
        list.append(line[start:end])
    return list
```

Reading File

parts `['Billie Jean', 'Michael Jackson', 'Thriller', 'Pop']`

```
def load_music_data(filename):
    music_data = {}
    with open(filename) as file:
        next(file)                      # Skip header line
        for line in file:
            line = line.strip()
            parts = parse_line(line)
            key = parts[0]              # Key is song name
            value = {                   # Value is dictionary of song data
                'Artist': parts[1],
                'Album': parts[2],
                'Genre': parts[3]
            }
            music_data[key] = value
        return music_data
```

Reading File

parts `['Billie Jean', 'Michael Jackson', 'Thriller', 'Pop']`

key `'Billie Jean'`

```
def load_music_data(filename):
    music_data = {}
    with open(filename) as file:
        next(file)                      # Skip header line
        for line in file:
            line = line.strip()
            parts = parse_line(line)
            key = parts[0]              # Key is song name
            value = {                   # Value is dictionary of song data
                'Artist': parts[1],
                'Album': parts[2],
                'Genre': parts[3]
            }
            music_data[key] = value
        return music_data
```

Reading File

parts `['Billie Jean', 'Michael Jackson', 'Thriller', 'Pop']`

key `'Billie Jean'`

value `{'Artist':'Michael Jackson', 'Album':'Thriller', 'Genre':'Pop'}`

```
def load_music_data(filename):
    music_data = {}
    with open(filename) as file:
        next(file)                      # Skip header line
        for line in file:
            line = line.strip()
            parts = parse_line(line)
            key = parts[0]              # Key is song name
            value = {                   # Value is dictionary of song data
                'Artist': parts[1],
                'Album': parts[2],
                'Genre': parts[3]
            }
            music_data[key] = value
        return music_data
```

Reading File

parts `['Billie Jean', 'Michael Jackson', 'Thriller', 'Pop']`

key `'Billie Jean'`

value `{'Artist':'Michael Jackson', 'Album':'Thriller', 'Genre':'Pop'}`

music_data

```
{
'Billie Jean':
  {'Artist':'Michael Jackson',
   'Album':'Thriller',
   'Genre':'Pop'}
}
```

```
def load_music_data(filename):
    music_data = {}
    with open(filename) as file:
        next(file)
        for line in file:
            line = line.strip()
            parts = parse_line(line)
            key = parts[0]              # Key is song name
            value = {                   # Value is dictionary of song data
                'Artist': parts[1],
                'Album': parts[2],
                'Genre': parts[3]
            }
            music_data[key] = value
        return music_data
```

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

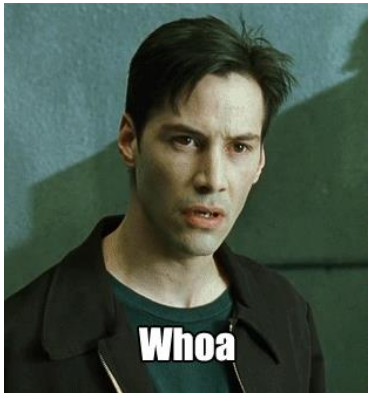
Show Me the Tuples!

- 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
```

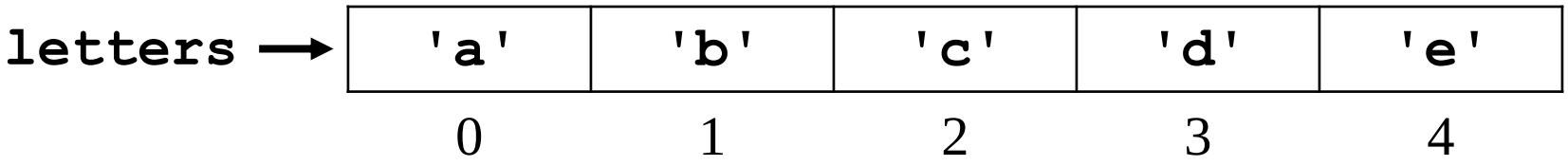


Accessing Elements of Tuple

- 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'`

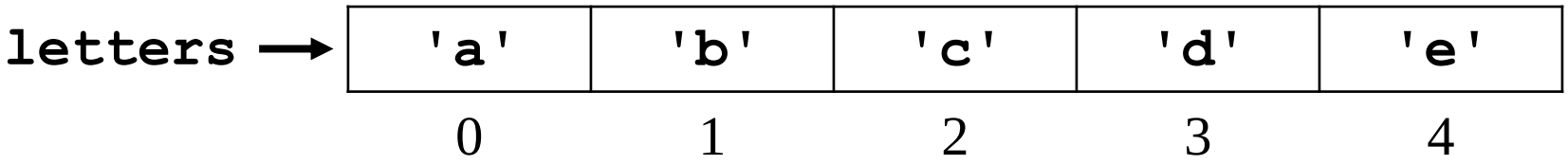
`letters[4]` is `'e'`

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'
```

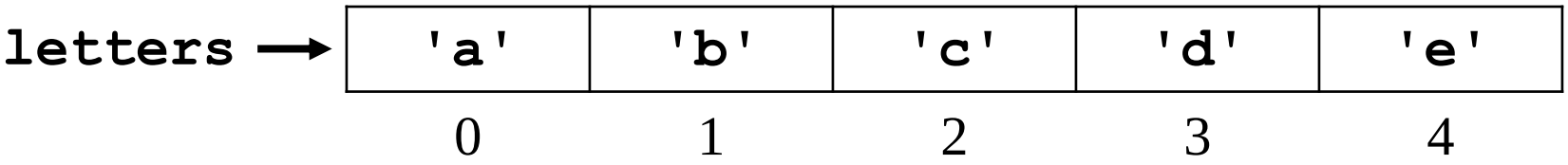
```
TypeError: 'tuple' object does not support
item assignment
```

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:

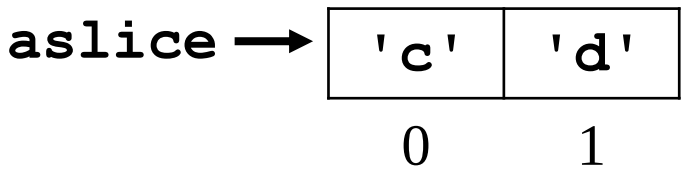
```
letters = ('a', 'b', 'c', 'd', 'e')
```

- 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')
```



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`**

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:

```
In remove_red: (0, 255, 0)
In main: (127, 255, 0)
```

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', 2022]
>>> a_tuple = tuple(a_list)
>>> a_tuple
('need', 'to', 'vote', 'in', 2022)
```

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 though 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'`**