Tuples

CS106AP Lecture 23
Life after CS106AP!

Day 1!

Programming Basics

The Console

Images

Graphics

Object-Oriented Programming

Midterm

Data structures

Everyday Python

Life after CS106AP!
Day 1!

1. Programming Basics
2. The Console
3. Images
4. Graphics
5. Midterm
6. Data structures
7. Everyday Python
   - Tuples
   - List Comp.
   - Jupyter
   - Internet
   - Life after Final Exam
   - Life after CS106AP!
Today’s questions

What can we learn from the midterm?

How can we bundle together small amounts of information?
Today’s topics

1. Exam Takeaways
2. Review
3. Tuples
4. What’s next?
Review
Creating an Interactive Graphical Game

- ZoneGraphics class handles all of the drawing and provides methods for client
- Client is responsible for main event loop and using the ZoneGraphics methods for gameplay logic
- Abstracting ZoneGraphics away allows us to create similar games
  - e.g. while graphics.ball_in_zone(): # do something
- Loose ends:
  - Multiple lives
  - Making sure ball isn’t reset in the zone
Creating an Interactive Graphical Game

- ZoneGraphics class handles all of the drawing and provides methods for client
- Client is responsible for main event loop and using the ZoneGraphics methods for gameplay logic
- Abstracting ZoneGraphics away allows us to create similar games
  - e.g. while graphics.ball_in_zone(): # do something
- Loose ends:
  - Multiple lives
  - Making sure ball isn’t reset in the zone
Creating an Interactive Graphical Game

- ZoneGraphics class handles all of the drawing and provides methods for client
- Client is responsible for main event loop and using the ZoneGraphics methods for gameplay logic
- Abstracting ZoneGraphics away allows us to create similar games
  - e.g. while graphics.ball_in_zone(): # do something
- Loose ends:
  - Multiple lives
  - Making sure ball isn’t reset in the zone
Creating an Interactive Graphical Game

- ZoneGraphics class handles all of the drawing and provides methods for client
- Client is responsible for main event loop and using the ZoneGraphics methods for gameplay logic
- Abstracting ZoneGraphics away allows us to create similar games
  - e.g. while graphics.ball_in_zone(): # do something

- Loose ends:
  - Multiple lives
  - Making sure ball isn’t reset in the zone
Midterm Takeaways
Midterm as a Diagnostic

- Midterm is a “check-in”. This is a good time to get help if you’re still working to understand some midterm topics.
- We will be looking to see your progress from midterm to final!
def pangolin(h):
    return 100 * h

def zoo(c, h, p):
    x = 0
    x += pangolin(c)
    print(x)

zoo(6, 2, 0.50)
def pangolin(h):
    return 100 * h

def zoo(c, h, p):
    x = 0
    x += pangolin(c)
    print(x)

zoo(6, 2, 0.50)
def pangolin(h):
    return 100 * h

def zoo(c, h, p):
    x = 0
    x += pangolin(c)
    print(x)

zoo(6, 2, 0.50)
def pangolin(h):
    return 100 * h

def zoo(c, h, p):
    x = 0
    x += pangolin(c)
    print(x)

zoo(6, 2, 0.50)
def pangolin(h):
    return 100 * h

def zoo(c, h, p):
    x = 0
    x += pangolin(c)
    print(x)

zoo(6, 2, 0.50)
Extend vs. Append

- `append()` is for adding a **single element** to a list
Extend vs. Append

- `append()` is for adding a **single element** to a list
- `extend()` is for adding a **list** to the end of a list
Extend vs. Append

- `append()` is for adding a **single element** to a list
- `extend()` is for adding a **list** to the end of a list
  - same thing as `+=`
List vs. String Functions

- Know what functions you can use with each!
List vs. String Functions

- Know what functions you can use with each!
  - Both strings + lists can do slicing + indexing
List vs. String Functions

- Know what functions you can use with each!
  - Both strings + lists can do slicing + indexing
  - But both have functions that are unique to them.
List vs. String Functions

- Know what functions you can use with each!
  - Both strings + lists can do slicing + indexing
  - But both have functions that are unique to them.
- Use the reference guide when in doubt!
Pattern Matching

- Sometimes a problem will remind you of another you’ve seen before.
Pattern Matching

- Sometimes a problem will remind you of another you’ve seen before.
- *Proceed with caution.*
Pattern Matching

● Sometimes a problem will remind you of another you’ve seen before.
● Proceed with caution.
● We don’t want you to solve by reshaping solution code to fit different problems.
Pattern Matching

● Sometimes a problem will remind you of another you’ve seen before.

● *Proceed with caution.*

● We don’t want you to solve by reshaping solution code to fit different problems.
  ○ Often this approach will result in big conceptual errors.
How can we bundle small amounts of information?
Tuples
What is a tuple?

(‘a’, ‘b’, ‘c’)

(‘karel’, 1)

(‘simba’, ‘lion’, 25)
What is a tuple?

(‘a’, ‘b’, ‘c’)

(‘karel’, 1)

(‘simba’, ‘lion’, 25)

like a list, but written with parentheses () instead of []
What is a tuple?

(‘a’, ‘b’, ‘c’)

(‘karel’, 1)

(‘simba’, ‘lion’, 25)

like a list, but written with parentheses ( ) instead of []

Definition

Tuple

An immutable data type for storing values in an ordered linear collection.
Tuples vs. Lists
Tuples vs. Lists

- len(), print()
<table>
<thead>
<tr>
<th>Tuples</th>
<th>vs.</th>
<th>Lists</th>
</tr>
</thead>
<tbody>
<tr>
<td>len(), print()</td>
<td></td>
<td>len(), print()</td>
</tr>
<tr>
<td>slicing, indexing</td>
<td></td>
<td>slicing, indexing</td>
</tr>
<tr>
<td>Tuples</td>
<td>vs.</td>
<td>Lists</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>len(), print()</td>
<td></td>
<td>len(), print()</td>
</tr>
<tr>
<td>slicing, indexing</td>
<td></td>
<td>slicing, indexing</td>
</tr>
<tr>
<td>in</td>
<td></td>
<td>in</td>
</tr>
<tr>
<td>Tuples</td>
<td>vs.</td>
<td>Lists</td>
</tr>
<tr>
<td>--------</td>
<td>-----</td>
<td>-------</td>
</tr>
<tr>
<td>• len(), print()</td>
<td>•</td>
<td>• len(), print()</td>
</tr>
<tr>
<td>• slicing, indexing</td>
<td>•</td>
<td>• slicing, indexing</td>
</tr>
<tr>
<td>• in</td>
<td>•</td>
<td>• in</td>
</tr>
<tr>
<td>• foreach loops</td>
<td>•</td>
<td>• foreach loops</td>
</tr>
<tr>
<td>Tuples</td>
<td>vs.</td>
<td>Lists</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>len(), print()</td>
<td></td>
<td>len(), print()</td>
</tr>
<tr>
<td>slicing, indexing</td>
<td></td>
<td>slicing, indexing</td>
</tr>
<tr>
<td>in</td>
<td></td>
<td>in</td>
</tr>
<tr>
<td>foreach loops</td>
<td></td>
<td>foreach loops</td>
</tr>
<tr>
<td>concatenation</td>
<td></td>
<td>concatenation (creates new)</td>
</tr>
<tr>
<td></td>
<td>(creates new)</td>
<td></td>
</tr>
</tbody>
</table>
Tuples vs. Lists

- len(), print()
- slicing, indexing
- in
- foreach loops
- concatenation (creates new)
- immutable

- len(), print()
- slicing, indexing
- in
- foreach loops
- concatenation (creates new)
- mutable
Tuples vs. Lists

- len(), print()
- slicing, indexing
- in
- foreach loops
- concatenation (creates new)
- immutable
  - can’t add elements

- len(), print()
- slicing, indexing
- in
- foreach loops
- concatenation (creates new)
- mutable
  - append(), extend()
Tuples vs. Lists

- len(), print()
- slicing, indexing
- in
- foreach loops
- concatenation (creates new)
- immutable
  - can’t add elements
  - can’t remove elements

- len(), print()
- slicing, indexing
- in
- foreach loops
- concatenation (creates new)
- mutable
  - append(), extend()
  - pop(), remove()
Tuples don’t support item assignment

```python
>>> tup = ('apple', 0.79, 'WA')
```
Tuples don’t support item assignment

```python
>>> tup = ('apple', 0.79, 'WA')
>>> tup[0]
```
Tuples don’t support item assignment

```python
>>> tup = (‘apple’, 0.79, ‘WA’)
```

```python
>>> tup[0]
```

You can index into to view the elements
Tuples don’t support item assignment

```python
>>> tup = ('apple', 0.79, 'WA')

>>> tup[0]
‘apple’
```

You can index into to view the elements

‘apple’
Tuples don’t support item assignment

```python
>>> tup = ('apple', 0.79, 'WA')
```

```python
>>> tup[0]
    'apple'
```

You can index in to view the elements

```python
>>> tup[2] = 'CA'
```
Tuples don’t support item assignment

```python
>>> tup = ('apple', 0.79, 'WA')

>>> tup[0]
'apple'

>>> tup[2] = 'CA'
TypeError
```

You can index in to view the elements

'apple'

TypeError
Tuples don’t support item assignment

```python
>>> tup = ('apple', 0.79, 'WA')

>>> tup[0]
'apple'

>>> tup[2] = 'CA'
TypeError
```

You can index in to view the elements

`'apple'`

You can’t index in to set one of the elements

TypeError
Tuple packing/unpacking

```python
>>> tup = ('apple', 0.79, 'WA')
```
Tuple packing/unpacking

```python
>>> tup = ('apple', 0.79, 'WA')
```

Parentheses not necessary for functionality but sometimes better for readability
Tuple packing/unpacking

```python
>>> tup = ('apple', 0.79, 'WA')
>>> tup
```

Parentheses not necessary for functionality but sometimes better for readability
Tuple packing/unpacking

```python
>>> tup = ('apple', 0.79, 'WA')
>>> tup
('apple', 0.79, 'WA')
```

Parentheses not necessary for functionality but sometimes better for readability.
Tuple packing/unpacking

```python
>>> tup = ('apple', 0.79, 'WA')
>>> tup
('apple', 0.79, 'WA')
>>> food, price, location = tup
```

Parentheses not necessary for functionality but sometimes better for readability
Tuple packing/unpacking

```python
>>> tup = ('apple', 0.79, 'WA')
>>> tup
('apple', 0.79, 'WA')
>>> food, price, location = tup
```

Parentheses not necessary for functionality but sometimes better for readability

unpack tuple contents into other variables
Tuple packing/unpacking

```python
>>> tup = ('apple', 0.79, 'WA')
>>> tup
('apple', 0.79, 'WA')
>>> food, price, location = tup
>>> price
```

Parentheses not necessary for functionality but sometimes better for readability

unpack tuple contents into other variables
Tuple packing/unpacking

```python
>>> tup = ('apple', 0.79, 'WA')
>>> tup
('apple', 0.79, 'WA')
>>> food, price, location = tup
>>> price
0.79
```
Assigning multiple variables simultaneously

```python
>>> a, b = (1, 2)
```
Assigning multiple variables simultaneously

```python
>>> a, b = 1, 2
```
Assigning multiple variables simultaneously

```python
>>> a, b = 1, 2
```

*Style note*

**parentheses**
Use parentheses with tuples when it improves readability
Assigning multiple variables simultaneously

```python
>>> a, b = 1, 2
```

packs 1, 2 into tuple and unpacks into variables b, a
Assigning multiple variables simultaneously

```python
>>> a, b = 1, 2
```

packs 1, 2 into tuple and unpacks into variables b, a
Assigning multiple variables simultaneously

```python
>>> a, b = 1, 2
```

Packs 1, 2 into tuple and unpacks into variables b, a
Assigning multiple variables simultaneously

```python
>>> a, b = 1, 2
>>> a
1
```

packs 1, 2 into tuple and unpacks into variables b, a
Assigning multiple variables simultaneously

```python
>>> a, b = 1, 2
>>> a
1
>>> b
```
Assigning multiple variables simultaneously

```python
>>> a, b = 1, 2
>>> a
1
>>> b
2
```

packs 1, 2 into tuple and unpacks into variables b, a
Assigning multiple variables simultaneously

```python
>>> a, b = 1, 2
>>> a
1
>>> b
2
```

packs 1, 2 into tuple and unpacks into variables b, a

Note: length of left-hand side must = length of right-hand side!
Tuples for swapping variables

```python
>>> a, b = 1, 2
```
Tuples for swapping variables

```python
>>> a, b = 1, 2
>>> b, a = a, b
```
Tuples for swapping variables

```python
>>> a, b = 1, 2
>>> b, a = a, b
```

packs a, b into tuple and unpacks into variables b, a
Tuples for swapping variables

```python
g>>> a, b = 1, 2
g>>> b, a = a, b
g>>> b
```

packs `a, b` into tuple and unpacks into variables `b, a`
Tuples for swapping variables

```python
>>> a, b = 1, 2
>>> b, a = a, b
>>> b
1
```

packs a, b into tuple and unpacks into variables b, a
Tuples for swapping variables

```python
>>> a, b = 1, 2
>>> b, a = a, b
>>> b
1
>>> a
```

packs a, b into tuple and unpacks into variables b, a
Tuples for swapping variables

```python
>>> a, b = 1, 2
>>> b, a = a, b
>>> b
1
>>> a
2
```
Tuples vs. Lists
Tuples vs. Lists

- fixed number of elements
- unbound number of elements
<table>
<thead>
<tr>
<th>Tuples</th>
<th>vs.</th>
<th>Lists</th>
</tr>
</thead>
<tbody>
<tr>
<td>• fixed number of elements</td>
<td>• unbound number of elements</td>
<td></td>
</tr>
<tr>
<td>• know # ahead of time</td>
<td>• sometimes large number</td>
<td></td>
</tr>
</tbody>
</table>
Tuples vs. Lists

- fixed number of elements
- know # ahead of time
- sometimes different types

- unbound number of elements
- sometimes large number
- usually all elements are the same type
Tuples vs. Lists

- fixed number of elements
- know # ahead of time
- sometimes different types

- unbound number of elements
- sometimes large number
- usually all elements are the same type

*not enforced by Python but good style*
Who wore it best: List vs. Tuple

- store many website urls
- store x, y, z coordinates together
- store many pixels
- store “name” and sunet id together
- store filename of mp3 file and its length in seconds
Who wore it best: List vs. Tuple

- store many website urls
- store x, y, z coordinates together
- store many pixels
- store “name” and sunet id together
- store filename of mp3 file and its length in seconds

- list
Who wore it best: List vs. Tuple

- store many website urls
- store x, y, z coordinates together
- store many pixels
- store “name” and sunet id together
- store filename of mp3 file and its length in seconds

- list
- tuple
Who wore it best: List vs. Tuple

- store many website urls
- store x, y, z coordinates together
- store many pixels
- store “name” and sunet id together
- store filename of mp3 file and its length in seconds

- list
- tuple
- list
Who wore it best: List vs. Tuple

- store many website urls
- store x, y, z coordinates together
- store many pixels
- store “name” and sunet id together
- store filename of mp3 file and its length in seconds

- list
- tuple
- list
- tuple
Who wore it best: List vs. Tuple

- store many website urls
- store x, y, z coordinates together
- store many pixels
- store “name” and sunet id together
- store filename of mp3 file and its length in seconds

- list
- tuple
- list
- tuple
- tuple
print_hungriest_animal()
(tuples in action!)
Tuples as return values

def animal_min_feedings(animal_dict):
    name, min_feedings = None, float('inf')
    for animal, num_feedings in animal_dict.items():
        if num_feedings < min_feedings:
            name, min_feedings = animal, num_feedings
    return name, min_feedings
Tuples as return values

def animal_min_feedings(animal_dict):
    name, min_feedings = None, float('inf')
    for animal, num_feedings in animal_dict.items():
        if num_feedings < min_feedings:
            name, min_feedings = animal, num_feedings
    return name, min_feedings
Tuples as return values

def animal_min_feedings(animal_dict):
    name, min_feedings = None, float('inf')
    for animal, num_feedings in animal_dict.items():
        if num_feedings < min_feedings:
            name, min_feedings = animal, num_feedings
    return name, min_feedings

packs name, min_feedings into a tuple and returns!
def print_hungriest_animal(filename):
    animal_dict = get_animal_feedings(filename)
    animal, num = animal_min_feedings(animal_dict)
    print(animal, 'eats', num, 'times a day. ')

unpacks name, min_feedings
tuple into two variables
Sorting lists with tuples

```python
>>> lst = [('mango', 3), ('apple', 6), ('lychee', 1), ('apricot', 10)]
```
Sorting lists with tuples

```python
>>> lst = [('mango', 3), ('apple', 6), ('lychee', 1), ('apricot', 10)]
>>> sorted(lst)
```
Sorting lists with tuples

```python
>>> lst = [(‘mango’, 3), (‘apple’, 6), (‘lychee’, 1), (‘apricot’, 10)]
>>> sorted(lst)
[(‘apple’, 6), (‘apricot’, 10), (‘lychee’, 1), (‘mango’, 3)]
```
Sorting lists with tuples

```python
>>> lst = [('mango', 3), ('apple', 6), ('lychee', 1), ('apricot', 10)]
>>> sorted(lst)
[('apple', 6), ('apricot', 10), ('lychee', 1), ('mango', 3)]
```

sorts by the first element in each tuple
Tuples + SimpleImage
What’s next?
List Comprehensions

- Can we make our code run faster?
- Can we write prettier loops?
- What other programming paradigms exist?