**Class 25:**
Python, Objects, Bombs, and Inheritance

---

**Menu**

- PS5: end-auction! running time
- Lists in Python
- Inheritance

---

**end-auction!**

```scheme
(define (end-auction!)
  (mmap
   (lambda (item-entry)
     (let ((item-name (list-get-element item-entry (table-field-number items 'item-name))))
       (let ((high-bid (get-highest-bid item-name)))
         (if (null? high-bid)
             (printf "No bids on ~a.\n" (list-get-element item-entry (table-field-number items 'item-name)))
           (printf "Congratulations ~a! You have won the ~a for $~a.\n" (list-get-element high-bid (table-field-number bids 'bidder-name)) item-name (list-get-element high-bid (table-field-number bids 'amount))))
           (list-get-element high-bid (table-field-number bids 'amount)))))))
  (table-entries items)))
```

---

**end-auction!**

```scheme
(define (end-auction!)
  (mmap
   (lambda (item-entry)
     (let ((item-name (list-get-element item-entry (table-field-number items 'item-name))))
       (let ((high-bid (get-highest-bid item-name)))
         (if (null? high-bid)
             (printf "No bids on ~a.\n" (list-get-element item-entry (table-field-number items 'item-name)))
           (printf "Congratulations ~a! You have won the ~a for $~a.\n" (list-get-element high-bid (table-field-number bids 'bidder-name)) item-name (list-get-element high-bid (table-field-number bids 'amount))))
           (list-get-element high-bid (table-field-number bids 'amount)))))))
  (table-entries items)))
```

---

**Python Lists**

Built-in datatypes for both mutable lists `[]` and immutable tuples `()`

```python
>>> m = range(1, 1000)
>>> m[0]
1
>>> m[-1]
999
>>> len(m)
999
>>> m[1:]
[2, ..., 999]
```

---

**Is m[1:] like mcdr?**

```python
>>> m1 = m[1:]
>>> m1[0]
2
>>> m1[1]
2
>>> m1[0] = 3
>>> m1[1]
2
```

---

**WRONG: need to be careful what N means!**
Implementing list-map in Python

```python
def schemish_list_map(f, p):
    if not p:
        return []
    else:
        return [f(p[0])] + schemish_list_map(f, p[1:])
```

Running time is in $\Theta(N^2)$ where $N$ is number of elements in $p$.

“Literal” translation...not a good way to do this.

Note: there is a built-in `map` in Python.

Pythonic Mapping

```python
def mlist_map(f, p):
    for i in range(0, len(p)):
        p[i] = f(p[i])
    return p
```

Unlike the previous one, this mutates $p$.

There are many kinds of Dogs...

```python
class Dog:
    def __init__(self, n):
        self.name = n
    def bark(self):
        print "wuff wuff wuff wuff"

class TalkingDog(Dog):
    def speak(self, stuff):
        print stuff
```

Every Dog has its Day

```python
class Dog:
    def __init__(self, n):
        self.name = n

class TalkingDog(Dog):
    def speak(self, stuff):
        print stuff
```

Running time is in $\Theta(N^2)$ where $N$ is number of elements in $p$.

“Inheritance

There are many kinds of Dogs...

```python
class Dog:
    def __init__(self, n):
        self.name = n
    def bark(self):
        print "wuff wuff wuff wuff"

class TalkingDog(Dog):
    def speak(self, stuff):
        print stuff
```

Every Dog has its Day

```python
class Dog:
    def __init__(self, n):
        self.name = n

class TalkingDog(Dog):
    def speak(self, stuff):
        print stuff
```

Running time is in $\Theta(N^2)$ where $N$ is number of elements in $p$.

“Inheritance

There are many kinds of Dogs...

```python
class Dog:
    def __init__(self, n):
        self.name = n
    def bark(self):
        print "wuff wuff wuff wuff"

class TalkingDog(Dog):
    def speak(self, stuff):
        print stuff
```

Every Dog has its Day

```python
class Dog:
    def __init__(self, n):
        self.name = n

class TalkingDog(Dog):
    def speak(self, stuff):
        print stuff
```

Running time is in $\Theta(N^2)$ where $N$ is number of elements in $p$.

“Inheritance

There are many kinds of Dogs...

```python
class Dog:
    def __init__(self, n):
        self.name = n
    def bark(self):
        print "wuff wuff wuff wuff"

class TalkingDog(Dog):
    def speak(self, stuff):
        print stuff
```

Every Dog has its Day

```python
class Dog:
    def __init__(self, n):
        self.name = n

class TalkingDog(Dog):
    def speak(self, stuff):
        print stuff
```

Running time is in $\Theta(N^2)$ where $N$ is number of elements in $p$.

“Inheritance

There are many kinds of Dogs...

```python
class Dog:
    def __init__(self, n):
        self.name = n
    def bark(self):
        print "wuff wuff wuff wuff"

class TalkingDog(Dog):
    def speak(self, stuff):
        print stuff
```

Every Dog has its Day

```python
class Dog:
    def __init__(self, n):
        self.name = n

class TalkingDog(Dog):
    def speak(self, stuff):
        print stuff
```

Running time is in $\Theta(N^2)$ where $N$ is number of elements in $p$.

“Inheritance

There are many kinds of Dogs...

```python
class Dog:
    def __init__(self, n):
        self.name = n
    def bark(self):
        print "wuff wuff wuff wuff"

class TalkingDog(Dog):
    def speak(self, stuff):
        print stuff
```

Every Dog has its Day

```python
class Dog:
    def __init__(self, n):
        self.name = n

class TalkingDog(Dog):
    def speak(self, stuff):
        print stuff
```

Running time is in $\Theta(N^2)$ where $N$ is number of elements in $p$.

“Inheritance

There are many kinds of Dogs...

```python
class Dog:
    def __init__(self, n):
        self.name = n
    def bark(self):
        print "wuff wuff wuff wuff"

class TalkingDog(Dog):
    def speak(self, stuff):
        print stuff
```

Every Dog has its Day

```python
class Dog:
    def __init__(self, n):
        self.name = n

class TalkingDog(Dog):
    def speak(self, stuff):
        print stuff
```

Running time is in $\Theta(N^2)$ where $N$ is number of elements in $p$.

“Inheritance

There are many kinds of Dogs...

```python
class Dog:
    def __init__(self, n):
        self.name = n
    def bark(self):
        print "wuff wuff wuff wuff"

class TalkingDog(Dog):
    def speak(self, stuff):
        print stuff
```

Every Dog has its Day

```python
class Dog:
    def __init__(self, n):
        self.name = n

class TalkingDog(Dog):
    def speak(self, stuff):
        print stuff
```

Running time is in $\Theta(N^2)$ where $N$ is number of elements in $p$.

“Inheritance

There are many kinds of Dogs...

```python
class Dog:
    def __init__(self, n):
        self.name = n
    def bark(self):
        print "wuff wuff wuff wuff"

class TalkingDog(Dog):
    def speak(self, stuff):
        print stuff
```

Every Dog has its Day

```python
class Dog:
    def __init__(self, n):
        self.name = n

class TalkingDog(Dog):
    def speak(self, stuff):
        print stuff
```

Running time is in $\Theta(N^2)$ where $N$ is number of elements in $p$.

“Inheritance

There are many kinds of Dogs...

```python
class Dog:
    def __init__(self, n):
        self.name = n
    def bark(self):
        print "wuff wuff wuff wuff"

class TalkingDog(Dog):
    def speak(self, stuff):
        print stuff
```

Every Dog has its Day

```python
class Dog:
    def __init__(self, n):
        self.name = n

class TalkingDog(Dog):
    def speak(self, stuff):
        print stuff
```

Running time is in $\Theta(N^2)$ where $N$ is number of elements in $p$.

“Inheritance

There are many kinds of Dogs...

```python
class Dog:
    def __init__(self, n):
        self.name = n
    def bark(self):
        print "wuff wuff wuff wuff"

class TalkingDog(Dog):
    def speak(self, stuff):
        print stuff
```

Every Dog has its Day

```python
class Dog:
    def __init__(self, n):
        self.name = n

class TalkingDog(Dog):
    def speak(self, stuff):
        print stuff
```

Running time is in $\Theta(N^2)$ where $N$ is number of elements in $p$.
Speaking about Inheritance

Inheritance is using the definition of one class to define another class.

TalkingDog \textit{inherits} from Dog.

TalkingDog is a \textit{subclass} of Dog.

The \textit{superclass} of TalkingDog is Dog.

These all mean the same thing.

PS6

Make an adventure game programming with objects

Many objects in our game have similar properties and behaviors, so we use inheritance.

PS6 Classes

PS6 Objects

Object-Oriented Summary

- An \textit{object packages state and procedures}.
- A \textit{class} provides procedures for making and manipulating a type of object.
- The procedures for manipulating objects are called \textit{methods}. We invoke a method on an object.
- Inheritance allows one class to refine and reuse the behavior of another. This is a good thing.
- Friday: Excursion on Exponential Growth
  — Please ready Tyson essay before Friday!