TODAY

- Review: Dispatch Dictionaries
- Review: OOP Implementation
- Scheme, a new language

REVIEW: DISPATCH DICTIONARIES

Idea: We will allow ourselves one kind of data structure. In particular, we will represent an object by a dispatch dictionary, where the messages are the keys.

def make_pair(first, second):
    pair_dict = {'first': first, 'second': second}
    return pair_dict

How do we create and use “pair objects” now?
```python
def make_instance(cls):
    if base_class is None:
        return cls
    else:
        return cls['get'](name)

def set_value(name, value):
    if name in attributes:
        attributes[name] = value
    else:
        return cls['get'](name)

def get_value(name):
    if name in attributes:
        return attributes[name]
    elif base_class is not None:
        return base_class['get'](name)
    else:
        return cls['get'](name)

DISPATCH_DICTIONARIES

>>> p['first']
1

>>> p['first'] = 3

REVIEW: OOP IMPLEMENTATION: INHERITANCE
Modified solution:
def make_class(attributes={}, base_class=None):
    def get_value(name):
        if name in attributes:
            return attributes[name]
        elif base_class is not None:
            return base_class['get'](name)
    def set_value(name, value):
        attributes[name] = value
    cls = {'get': get_value, 'set': set_value}
    return cls

To find the value of a class attribute...
... check if it is already in the dictionary of attributes...
Otherwise, if there is a parent class, check if the parent class has the class attribute.

A class is still a dictionary! The two new messages get and set allow us to use the general getter and setter functions.

REVIEW: OOP IMPLEMENTATION: OBJECTS
def make_instance(cls):
    def get_value(name):
        if name in attributes:
            return attributes[name]
        else:
            value = cls['get'](name)
            return bind_method(value, instance)
    def bind_method(value, instance):
        if callable(value):
            return value(instance, *args)
        else:
            return value
    def set_value(name, value):
        if name in attributes:
            attributes[name] = value
        else:
            return cls['get'](name)
    attributes = {}  # messages still allow setting attributes
    instance = {'get': get_value, 'set': set_value}

    return instance

To find the value of an instance attribute...
... check if it is already in the dictionary of attributes...
Otherwise, check if the class has a value for the attribute.

An instance is a dictionary! The two new messages get and set allow us to use the general getter and setter functions.

REVIEW: OOP IMPLEMENTATION: OBJECTS

All the methods are defined in the class. We might get one of these: if we do, bind the first argument to the current instance to produce a bound method for the instance.

If the value provided is callable (for example, a function),...
... make the new bound method...
... where the first argument is bound to the instance given, and all the other arguments remain the same...
... and return this bound method.

If the value provided is not callable, return it as is.
```

```
```
REVIEW: OOP IMPLEMENTATION: INSTANTIATION AND INITIALIZATION

We add an extra message that our classes can understand, which will allow us to create new objects:

def make_class(attributes={}, base_class=None):
    ...
def new(*args):
    return init_instance(cls, *args)

cls = {'get': get_value, 'set': set_value, 'new': new}
return cls

ANNOUNCEMENTS: MIDTERM 2

- Midterm 2 is on Wednesday, July 25.
  - Where? 2050 VLSB.
  - When? 7PM to 9PM.
- Closed book and closed electronic devices.
- One 8.5” x 11” ‘cheat sheet’ allowed.
- Group portion is 15 minutes long.
- If you have a conflict, please let us know by the end of today, July 23.

SCHEME: A DIALECT OF LISP

- The language you will implement for Project 4.
- One of the oldest languages still in use today!
- Easiest way to learn Scheme is to start by seeing some examples and then play with it, so that’s what most of today will be.

ANNOUNCEMENTS:

- Homework 10 is due Today.
- Project 3 is due Thursday, July 26.
- Homework 11 is due Friday, July 27.
  - This will be released sometime tonight.

Please ask for help if you need to. There is a lot of work in the weeks ahead, so if you are ever confused, consult (in order of preference) your study group and Piazza, your TAs, and Jom.

Don’t be clueless!

**REVIEW: OOP IMPLEMENTATION: USAGE**

def make_pokemon_class():
    def __init__(self, name, owner, hp):
        self['set']('name', name)
        self['set']('owner', owner)
        self['set']('hp', hp)
    def increase_hp(self, amount):
        old_hp = self['get']('hp')
        self['set']('hp', old_hp + amount)
    ...
    return make_class({"__init__":__init__,
                        'increase_hp':increase_hp, ...})

>>> Pokemon = make_pokemon_class()
>>> ashs_pikachu = Pokemon['new']('Pikachu', 'Ash', 300)
>>> ashs_pikachu['get']('hp') 300
>>> ashs_pikachu['get']('owner') 'Ash'
>>> ashs_pikachu['get']('increase_hp')(50)
>>> ashs_pikachu['get']('hp') 350

**SCHEME: A DIALECT OF LISP**

- The language you will implement for Project 4.
- One of the oldest languages still in use today!
- Easiest way to learn Scheme is to start by seeing some examples and then play with it, so that’s what most of today will be.

**ANNOUNCEMENTS**

- Homework 10 is due Today.
- Project 3 is due Thursday, July 26.
- Homework 11 is due Friday, July 27.
  - This will be released sometime tonight.

Please ask for help if you need to. There is a lot of work in the weeks ahead, so if you are ever confused, consult (in order of preference) your study group and Piazza, your TAs, and Jom.

Don’t be clueless!
SCHEME: SIMPLE EXAMPLES

```
STk> 5  
STk> (= 3 22) #f
STk> (= 3 3) #t
STk> '(hello tom) (hello tom)
STk> (+ 5 7) 12
STk> (* 3 22) 66
```

SCHEME: SIMPLE EXAMPLES

```
STk> (if (= 3 3) 'true 'false) true
STk> (if (= 3 4) 'true)
STk> (first 'word) w
STk> (butfirst 'word) ord
STk> (last 'word) d
STk> (butlast 'word) wor
STk> (word 'race 'car) racecar
```

SCHEME: SIMPLE EXAMPLES

```
STk> (cond ((=< 5 0) 'zero) ((= (remainder 5 2) 0) 'even) (else 'odd))
odd
STk>
```

SCHEME: VARIABLES

```
STk> (define toms-number 5)
toms-number
STk> toms-number 5
STk> y
*** Error: unbound variable: y
...
STk> (+ toms-number 66)
```

SCHEME: FUNCTIONS

```
STk> (define (square x) (* x x))
square
STk> (square 5)
25
STk> square
```

SCHEME: PREFIX NOTATION

```
In Python:
4 + 7

In Scheme
(+ 4 7)
```

SCHEME: PREFIX NOTATION

Notice that the parentheses go **BEFORE** the function or operation.

Scheme uses **prefix notation** where all operations appear before the expressions used by the operation. There is **absolutely no exceptions** to this notation.

This makes the language much simpler, since we don’t have a bunch of different special cases for each type of operation.
**SCHEME: FUNCTIONS**

Scheme has lambdas! Unlike Python, anything you can do in a named function you can also do in a lambda.

```scheme
STk> (lambda (x) (+ x 1))
#<closure arglist=(x) 23bbc8>
STk> ((lambda (x) (* x x)) 3) 9
STk> (define (call-with-2 fn) (fn 2))
call-with-2 STk> ((call-with-2 (lambda (x) (+ x 1))) 3)
STk> (lambda () 3)
#<closure arglist=() 23bbc8>
STk> (((lambda () 3)) 3)
```

**SCHEME: REPETITION**

```scheme
STk> (define (fact n)
    (if (<= n 1)
        1
        (* n (fact (- n 1))))

fact
STk> (fact 5)
120
```

**SCHEME: MORE EXAMPLES**

```scheme
STk> (define (twice f)
    (lambda (x) (f (f x))))
twice
STk> (((twice (twice twice)) 1+) 0) 16
STk> (define (make-adder x)
    (lambda (y) (+ x y)))
make-adder
STk> ((make-adder 7) 22) 29
```

**SCHEME: PAIRS AND LISTS**

```scheme
STk> (cons 2 5)
(2 . 5)
STk> (list 1 2 3 4 5 6)
(1 2 3 4 5 6)
STk> (car '(1 2 3))
1
STk> (cdr '(1 2 3))
(2 3)
STk> (cons 1 '())
(1)
STk> (cons 1 (cons 2 (cons 3 '())))
(1 2 3)
```

**SCHEME: LISTS**

```scheme
STk> (append '(1 2 3) '(4 5 6))
(1 2 3 4 5 6)
STk> (define (map fn lst)
    (if (null? lst)
        ()
        (cons (fn (car lst))
             (map fn (cdr lst)))))
STk> (map (lambda (x) (* x 5)) '(1 2 3))
(5 10 15)
STk> (map (lambda (x) (* x 22)) '(1 2 3))
(22 44 66)
```
PRACTICE: SCHEME
What does each expression evaluate to?
STk> (square (square 5))
625
STk> '(square 5)
(square 5)
STk> (cons (car '(3 . 4)) (cdr '(5 . 6)))
(3 5 6)
STk> (cons (car '(3 . 4)) (cdr '(5 6)))
(3 5 6)

CONCLUSION
Scheme: It’s awesome, it’s simple, it’s what you’re implementing in project 4.
Preview: We will start to talk about how we implement an interpreter.