Possible solution:
Linked list data abstraction:

```python
def link(first, rest):
    return [first, rest]
def first(s):
    return s[0]
def rest(s):
    return s[1]
def len_link(s):
    x = 0
    while s:
        x = x + 1
        s = rest(s)
    return x
def getitem_link(s, i):
    while i > 0:
        s = rest(s)
        i = i - 1
    return first(s)
```

The result of calling `repr` on a value is what Python prints in an interactive session:

```python
>>> today = datetime.date(2014, 10, 13)
>>> print(repr(today))
2014-10-13
```

`str` and `repr` are both polymorphic; they apply to any object that is the value of the expression `expression`.

Python object system:

```python
class Account:
    def __init__(self, account_holder):
        self.balance = 0
        self.account_holder = account_holder

    def deposit(self, amount):
        self.balance = self.balance + amount
        return self.balance

    def withdraw(self, amount):
        if amount > self.balance:
            return 'Insufficient Funds'
        self.balance = self.balance - amount
        return self.balance
```

```python
class CheckingAccount(Account):
    # A bank account that charges for withdrawals.
    withdraw_fee = 0.05

    def withdraw(self, amount):
        self.balance = self.balance - amount
        return self.balance
```

Assignments with a dot expression on their left-hand side affect the attributes of the object of that dot expression:

```python
# The object is an instance, then assignment sets an instance attribute
>>> jim_account = Account('Jim')
>>> jim_account.balance = 20
>>> jim_account.balance
20
```

To look up a name in a class:

1. If it names an attribute in the class, return the attribute value.
2. Otherwise, look up the name in the base class, if there is one.