COBRA

An Elegant Language for a more Sophisticated Age

Chase Arrington
Adam Barrett

Basic Functionality

- Functional and Simple Arithmetic
- Variable Storage and Recall
- Function Definition
- List Comprehension
- Java Execution
- Streaming

Basic Functionality | Arithmetic

- As you would write it.

Syntax : a (+, -, *, /) b

```
>>> 1 + 2
3
>>> 2 * 5
10
>>> 9 / 3
3
```

Basic Functionality | Variables

Syntax : x = a

```
>>> a = 6
6
>>> print a
6
>>> a + 7
13
```

Basic Functionality | Defining a Function

Syntax : def *name*(*vars*){*function*}

```
>>> def add(a b){a + b}
add
>>> add(1 2)
3
>>> def circAr(r){3.14 * r * r)
circAr
```

Code | Python Closure

```python
class function(object):
    def f(self):
        data = {
            'varVals': {},
            '$varVals': lambda x: data.update({'varVals': x}),
            'varL': [],
            '$varL': lambda x: data.update({'varL': x}),
            'equation': ['a', '+', 'b'],
            '$equation': lambda x: data.update({'equation': x}),
        }
        def cf(self, d):
            if d in data:
                return data[d]
            else:
                return None
        return cf
    run = f(1)

s1 = function()
```

Code | Python Closure Call

```python
def p_defFunc(p):
    '''call : FUNC item LPAREN items RPAREN LCURLY item M
            | FUNC item LPAREN items RPAREN LCURLY item M
            | FUNC item LPAREN items RPAREN LCURLY EXEC i
    '''
    functions[p[2]] = function()
    a = {}
    b = []
    c = []
    for i in p[4]:
        a[i] = 0
        b.append(i)
    functions[p[2]].run('$varVals')(a)
    functions[p[2]].run('$varL')(b)
    c = p[7 : len(p) - 1]
    functions[p[2]].run('$equation')(c)
    p[0] = p[2]
```

```python
def p_callFunc(p):
    'call : item LPAREN items RPAREN'
    if p[1] in functions:
        funString = ''
        a = functions[p[1]].run('varVals')
        b = functions[p[1]].run('varL')
        c = functions[p[1]].run('equation')
        for i in range (0, len(p[3])):
            a[b[i]] = p[3][i]
        for i in c:
            if i in a:
                i = a[i]
            funString += str(i)
        funString += '**'
        p[0] = funString
    else:
        print 'nope nope nope'
```

Basic Functionality | Java Execution

Syntax : exec *func* [*args*]

```
>>> exec Math.max [24, 70]
70
>>> exec Math.min [800, 2]
2
```

Basic Functionality | Map

Syntax : map *x* [*list*] *func*

```
>>> map x [1 2 3 4 5] x + 1
[2 3 4 5 6]
>>> a = [1 2 3 4 5]
>>>map x a x + 1
>>> print a
[2 3 4 5 6]
```

Code | Java Execution

```
def p_exec(p):
    'item : EXEC items'
    from java.lang import Math
    s = ''
    for i in p[2]:
        i = str(i)
        i = i.replace('[', '(')
        i = i.replace(']', ')')
        s += str(i)
    p[0] = eval(compile(s, 'None', 'single'))
```

Basic Functionality | List Storage

Syntax : x = [[a,b][c,#]]

```
>>> s_dept = [[ID, NAME, REGION_ID'],[10, Finance, 1],[31, Sales, 1],[32, Sales, 2],[33, Sales, 3],[34, Sales, 4],[35, Sales, 5],[41, Operations, 1],[42, Operations, 2],[43, Operations, 3],[44, Operations, 4],[45, Operations, 5],[50, Administration, 1]]
```

```
[['ID', 'NAME', 'REGION_ID'], [10, 'Finance', 1], [31, 'Sales', 1], [32, 'Sales', 2], [33, 'Sales', 3], [34, 'Sales', 4], [35, 'Sales', 5], [41, 'Operations', 1], [42, 'Operations', 2], [43, 'Operations', 3], [44, 'Operations', 4], [45, 'Operations', 5], [50, 'Administration', 1]]
```

```
>>> print s_dept
```

```
[['ID', 'NAME', 'REGION_ID'], [10, 'Finance', 1], [31, 'Sales', 1], [32, 'Sales', 2], [33, 'Sales', 3], [34, 'Sales', 4], [35, 'Sales', 5], [41, 'Operations', 1], [42, 'Operations', 2], [43, 'Operations', 3], [44, 'Operations', 4], [45, 'Operations', 5], [50, 'Administration', 1]]
```

Code | List Storage

```python
with open('testerfile.txt', 'r') as content_file:
    content = content_file.read()
s = ''
lines = []
for i in content:
    if i == '\n':
        lines.append(s)
        s = ''
    else:
        s += i
lines.append(s)
for i in lines:
    result = yacc.parse(i)
    if isinstance(result, str):
        if result[len(result) - 2] + result[len(result) - 1] == '**':
            result = yacc.parse(result[0 : len(result) - 2])
    if result != None:
        print (result)
```

```python
def p_list(p):
    'call : LBrack items RBrack'
    p[0] = p[2]
```

Basic Functionality | List Comprehension

Syntax : select a b c list

>>> select 1 2 5 6 s_emp

select LAST_NAME, FIRST_NAME, TITLE, SALARY from s_emp:

[['Martin', 'Carmen', 'President', 4500], ['Jackson', 'Marta', 'Warehouse_Manager', 1507], ['Henderson', 'Colin', 'Sales_Representative', 1400], ['Gilson', 'Sam', 'Sales_Representative', 1490], ['Sanders', 'Jason', 'Sales_Representative', 1515], ['Dameron', 'Andre', 'Sales_Representative', 1450], ['Hardwick', 'Elaine', 'Stock_Clerk', 1400], ['Brown', 'George', 'Stock_Clerk', 940], ['Washington', 'Thomas', 'Stock_Clerk', 1200], ['Patterson', 'Donald', 'Stock_Clerk', 795], ['Bell', 'Alexander', 'Stock_Clerk', 850], ['Smith', 'Doris', 'VP_Operations', 2450], ['Gantos', 'Eddie', 'Stock_Clerk', 800], ['Stephenson', 'Blaine', 'Stock_Clerk', 860], ['Chester', 'Eddie', 'Stock_Clerk', 800], ['Pearl', 'Roger', 'Stock_Clerk', 795], ['Dancer', 'Bonnie', 'Stock_Clerk', 860], ['Schmitt', 'Sandra', 'Stock_Clerk', 1100], ['Norton', 'Michael', 'VP_Sales', 2400], ['Quentin', 'Mark', 'VP_Finance', 2450], ['Roper', 'Joseph', 'VP_Administration', 2550], ['Brown', 'Molly', 'Warehouse_Manager', 1600], ['Hawkins', 'Roberta', 'Warehouse_Manager', 1650], ['Burns', 'Ben', 'Warehouse_Manager', 1500], ['Catskill', 'Antoinette', 'Warehouse_Manager', 1700]]

Code | List Comprehension

```
def p_selectList(p):
    '''call : SELECT ALL item
            | SELECT NUM NUM NUM NUM item
            | SELECT NUM NUM NUM NUM item NUM LESS NUM AND NUM GREAT NUM'''
    if p[2] == 'all':
        print "\nselect * from " + str(p[3]) + ': ,' + str(p[3])
        p[0] = vars[p[3]]
    else:
        if len(p) == 7:
            a=[]
            for i in vars[p[6]][1:]:
                a.append([i[p[2]],i[p[3]],i[p[4]],i[p[5]]])
            print "\n select "+str(vars[p[6]][0][p[2]])+', '+str(vars[p[6]][0][p[3]])+', '\
                  +str(vars[p[6]][0][p[4]])+', '+str(vars[p[6]][0][p[5]])+' from '+str(p[6])+': '
            p[0] = a
        if len(p) == 14:
            b=[]
            if p[8] == '<' and p[12] == '>':
                for i in vars[p[6]][1:]:
                    if i[p[7]] < p[9] and i[p[11]] > p[13]:
                        b.append([i[p[2]],i[p[3]],i[p[4]],i[p[5]]])
                print "\n select "+str(vars[p[6]][0][p[2]])+', '+str(vars[p[6]][0][p[3]])+', '\
                      +str(vars[p[6]][0][p[4]])+', '+str(vars[p[6]][0][p[5]])+' from '+str(p[6])+' where '+\
                      str(vars[p[6]][0][p[7]]), p[8], p[9], p[10], str(vars[p[6]][0][p[7]]),p[12], p[13]
                p[0] = b
```

Cobra | Conclusion

Intuitive. Simple. ~~Python~~ Cobra.