List methods

```
lst = [1, 2, 3]
len(lst)            # 3
lst[0]              # 1
lst[0:2]            # [1, 2]
lst[0] = ’howdy’    # lst == [’howdy’, 2, 3]
lst.append(29)      # lst == [’howdy’, 2, 3, 29]
lst.pop()           # lst == [’howdy’, 2, 3], returns 29
lst.pop(1)          # lst == [’howdy’, 3], returns 2
lst.insert(1, 100)  # lst == [’howdy’, 100, 3]
3 in lst            # returns True
```

Dictionary methods

```
d = {’hi’: 4, ’bye’: 100}
d[’hi’]           # ’hi’
d[100]            # raises KeyError!
’hi’ in d         # True
4 in d            # False
d[’howdy’] = 15   # adds new key-value pair
d[’hi’] = -100    # changes a key-value pair
```

Control flow

```
if x == 5:
    y = 1
elif 4 <= 100:
    z = 2
else:
    y = 100

for i in [0, 1, 2, 3]:     # or, "for i in range(4):"
    print(i)

j = 0
while j < 10:
    print(j)
    j = j * 2
```

Class syntax

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def size(self):
        return (self.x ** 2 + self.y ** 2) ** 0.5

p = Point(3, 4)   # constructor
p.x               # attribute access: returns 3
p.size()          # method call: returns 5.0

class MyWeirdClass(Point):
    pass
```

Exceptions

```
raise KeyError
try:
    lst[1000]
except IndexError:
    print(’haha’)
```

Linked lists (recursive)

```
class LinkedListRec:

    def __init__(self, items):
        if len(items) == 0:
            self.first = EmptyValue
            self.rest = None
        else:
            self.first = items[0]
            self.rest = LinkedListRec(items[1:])

    def is_empty(self):
        return self.first is EmptyValue
```

General Trees

```
class Tree:

    def __init__(self, root=EmptyValue):
        self.root = root
        self.subtrees = []

    def is_empty(self):
        return self.root is EmptyValue
```

Binary Search Trees

```
class BinarySearchTree:

    def __init__(self, root=EmptyValue):
        self.root = root    # root value
        if self.is_empty():
            self.left = None
            self.right = None
        else:
            self.left = BinarySearchTree()
            self.right = BinarySearchTree()

    def is_empty(self):
        return self.root is EmptyValue
```