

# Assignment 6: lists, dictionaries, and heap allocation

ECEN 4553 & 5013, CSCI 4555 & 5525  
Prof. Jeremy G. Siek

October 7, 2007

The main ideas introduced in this chapter are

- compound data structures like lists and dictionaries
- heap allocation
- object identity and aliasing
- garbage collection

## 1 Syntax of $P_3$

In this chapter we add two new kinds of objects: lists and dictionaries. A Python list is a sequence of elements. A Python list is not necessarily represented using a linked-list data structure, in fact the standard python interpreter uses an array. A Python dictionary is a mapping from keys to values. The concrete syntax for the  $P_3$  subset of Python is shown in Figure 1. To keep things simple,  $P_3$  does not include list comprehensions (the functionality associated with the `list_iter` non-terminal in the Python grammar) and it does not include slicing.

```
expression ::= "[" expression ("," expression)* "]"
            | "{" key_datum ("," key_datum)* "}"
            | subscription
            | expression "is" expression
key_datum  ::= expression ":" expression
subscription ::= expression "[" expression "]"
simple_statement ::= (target "=")+ expression
target ::= identifier
        | subscription
```

Figure 1: Concrete syntax for the  $P_3$  subset of Python. (In addition to that of  $P_2$ .)

## 2 Semantics of $P_3$

A list may be created with an expression that contains a list of its elements surrounded by square brackets, e.g., `[3,1,4,1,5,9]` creates a list of six integers. The  $n$ th element of a list can be accessed using the subscript notation `l[n]` where  $l$  is a list and  $n$  is an integer (indexing is zero based). For example, `[3,1,4,1,5,9][2]` evaluates to 4. The  $n$ th element of a list can be changed by using a

subscript expression on the left-hand side of an assignment. For example, the following fixes the 4th digit of  $\pi$ .

```
>>> x = [3,1,4,8,5,9]
>>> x[3] = 1
>>> print x
[3, 1, 4, 1, 5, 9]
```

With the introduction of lists and dictionaries, we now have entities in the language where there is a distinction between identity (the `is` operator) and equality (the `==` operator). The following program, we create two lists with the same elements. Changing list `x` does not affect list `y`.

```
>>> x = [1,2]
>>> y = [1,2]
>>> print x == y
True
>>> print x is y
False
>>> x[0] = 3
>>> print x
[3, 2]
>>> print y
[1, 2]
```

Variable assignment is shallow in that it just points the variable to a new entity and does not affect the entity previous referred to by the variable. Multiple variables can point to the same entity, which is called “aliasing”.

```
>>> x = [1,2,3]
>>> y = x
>>> x = [4,5,6]
>>> print y
[1, 2, 3]
>>> y = x
>>> x[0] = 7
>>> print y
[7, 5, 6]
```

**Exercise 2.1.** Read the sections of the Python Reference Manual that apply to  $P_3$ : 3.1, 5.2.4, 5.2.6, 5.3.2, 5.6, 5.9, 6.3

**Exercise 2.2.** Write five programs in the  $P_3$  subset of Python that help you understand the language. Look for corner cases or unusual aspects of the language to test in your programs. Run the programs using the standard python interpreter. Add your test programs to the test directory.

### 3 Compiling lists and dictionaries

Figure 2 shows the Python classes for  $P_3$  ASTs.

List and dictionary objects have a life of their own beyond the variables that they are bound to, which means when compiling to C we need to use the heap to store them. In C, the lifetime of heap-allocated objects is usually under programmer control: the space for the object is allocated (requested from the operating system) with `malloc` and the space is given back to the operating

```

class List(Node):
    def __init__(self, nodes):
        self.nodes = nodes
class Dict(Node):
    def __init__(self, items):
        self.items = items
class Subscript(Node):
    def __init__(self, expr, flags, subs):
        self.expr = expr
        self.flags = flags
        self.subs = subs

```

Figure 2: The Python classes for  $P_3$  ASTs.

system with `free`. On the other hand, in Python the program creates objects, but does not specify when they are no longer needed. Thus, an automated mechanism is needed to reclaim unused objects. The standard python interpreter uses reference counting, but a more general approach is to use a garbage collector. The following is a link to an easy to use garbage collector for C programs.

[http://www.hpl.hp.com/personal/Hans\\_Boehm/gc/](http://www.hpl.hp.com/personal/Hans_Boehm/gc/)

Instead of allocating space with `malloc`, use `GC_malloc`.

To compile list objects to C, we must also decide on a particular data-structure with which to encode lists. For example, we could use a linked-list, or we could use an array. Because Python lists support the subscript operator, it is more time efficient to use an array, as that data-structure provides constant-time access to elements at arbitrary offsets within the array. If we used a linked-list, the subscript operator would need to perform a linear-time traversal of the list.

Similarly, we must decide on a particular data-structure to implement dictionaries. The traditional choice is to use a hash table, though other options include red-black trees and sorted arrays. An implementation of a hashtable is available on the course web site.

**Exercise 3.1.** Extend your Python to C compiler to handle  $P_3$ . Remember to update the `+` operator to support the concatenation of two lists.

## References