CS 61A CSM
Fall 2017

The Structure and Interpretation of Computer Programs
Kartik Kapur

Lists, For Loops, Data Abstraction

1 Introduction

This worksheet will primarily be going over Lists, For Loops, and Data Abstraction.

2 Lists

Lists are essentially containers for object types. Lists can contain numbers, strings, croissants, pretty much anything. Lists are a pretty handy tool and can be used when you want to store a collection of data or if you want some sort of ordering for said data.

2.1 Basic List Operations

Having plain lists that you couldn't do anything with could get pretty boring, so Python has a few operations that you can perform to lists. The ones that we'll be discussing are adding two lists and splicing.
Adding two lists is literally what it sounds like, putting one list at the end of the other.

```
>>> lst = [1,2,3]
>>> lst2 = [4,5,6]
>>> lst1+lst2
[1,2,3,4,5,6]
```

The list's first values would be the item that is to the left of the addition sign and the later values would be the one on the right of the addition sign. Note that when we add 2 lists, neither of the original lists are affected, as a new list is created when we perform and operation on them.
There is a quick distinction that we need to make, and that is the += operation. Intuitively, it feels that having lst = lst + [1] would be the same thing as lst+=[1]; however these are vastly things

```
>>> lst = [1,2,3,4,5]
>>> lst + [6]
>>> lst
[1,2,3,4,5]
>>> lst+=[6]
>>> lst
[1,2,3,4,5,6]
```

The difference between these 2 methods is quite subtle; however, there is a major difference. Simply adding two lists would create a new list; however, doing the += operation would result in the same list getting modified.
The next operation that we will go over is splicing. Splicing is essentially creating a subset of a list. Splicing takes the form of [before index : after index]. If the before index is left blank, then the default argument would be 0 and if the after index is left blank then the default argument would be the length of the list. Slicing is done as follows.

```
>>> lst = [1,2,3,4,5]
>>> lst[2:4]
[3,4]
```

Note that when we're splicing lists, the beginning index is inclusive whereas the end index is exclusive, so our splice of lst[2:4] would only contain the elements at indices 2 and 3 since we exclude 4.

2.2 More Splicing Operations

When splicing, we can also deal with pretty weird patterns including going backwards, skipping indices, and using negative indices!
The first operation that we go over s

```
>>> lst = [1,2,3,4,5]
>>> lst[::2]
[1,3,5]
```

In this case, we have two colons in a row, what a strange syntax. This new syntax goes as follows: [start index: end index: size of the step]. In our above example, the first step was the default argument, 0 and the second step was the default argument, the length of the list, 5. The final argument is 2, so we go from index 0 to 4 and skip every other step. We then create a new list with the 0th index, 2nd index, and the 4th index.
The next case that we will go over is negative indices. We know that we have the standard indexing that starts at 0 and

ends at the the length of the list -1; however, we can have a weirder case, when the index is negative. Negative indices work as follows, -1 is equivalent to the last index of the list while -size of the list is the first index of the list.

```
>>> lst = [1,2,3,4,5]
>>> lst[-3]
3
```

We start at 5 and see that it is index at -1, we keep moving until we reach -3 whose element is 3.

2.3 Box and Pointers