1 Lists

To construct a list, one can use the list constructor, so \( l = \text{list()} \) returns an empty list. The constructor also takes any iterable object in Python and constructs a list from it. For example \( \text{list(range(5))} \) returns a new list equal to \([0, 1, 2, 3, 4]\) and \( \text{list("cow")} \) returns a new list equal to \(['c', 'o', 'w']\). One can use the square bracket notation to create lists too, so \([3, 1, 4, 1, 5, 9]\) returns an appropriate list of length 6.

Operations

Lists, like strings, are sequences of objects, so they support the sequence operations:

- indexing,
- slicing, and
- length.

These operations are not side-effecting—they keep However, there are many differences between lists and strings:

- Lists are mutable, which means that we can change the contents of the list several of its methods. If \( l \) is a list, then the following operations are all popular methods for manipulating \( l \):
  
  - **index assignment**: \( l[i] = \text{obj} \) means replace the object at index \( i \) of \( l \) with \( \text{obj} \).
  - **appending**: \( l.\text{append}(\text{obj}) \) means append \( \text{obj} \) to \( l \) so that the length of \( l \) increases by one.
  - **inserting**: \( l.\text{insert}(i, \text{obj}) \) means insert \( \text{obj} \) at index \( i \) of \( l \); the length of the list increases by one.
  - **popping**: \( l.\text{pop}(i) \) means delete the the object at index \( i \) of \( l \); \( l.\text{pop()} \) means delete the last object.
  - **deleting**: \( \text{del } l[i] \) means delete the object at index \( i \) of \( l \); this decreases the length of the list by one.
  - **removing**: \( l.\text{remove}(\text{obj}) \) means remove the first item in \( l \) that equals \( \text{obj} \).

- Sort lists using the sort() method.

- Lists are heterogeneous, which means they can simultaneously store objects of different type.

- Lists are really adjustable arrays, which we will examine in detail later.

- Lists support list comprehensions, which allow you to make new lists from other iterables. For example, to generate the first five non-negative multiples of 5, one could write:

\[
\{5*i \text{ for } i \text{ in } \text{range}(10)\}
\]

Let \( l = \text{list(range(10))} \). What does \( l \) equal after the following operations?

\[
\text{l.append(11)} \\
\text{del l[0]} \\
\text{l.remove(1)}
\]

Let \( l = \text{list('sub pop')} \). What does \( l \) equal after the following operations?

\[
\text{l.insert(3, '*')} \\
\text{l[len(l)-2] = 'u'} \\
\text{l.append('!')} \\
\text{l.append(l.pop())}
\]
2 Searching

A fundamental operation in computer science is search. Suppose we have a list of strings

```python
```

and we want to be able to find a string in the list that begins with a certain prefix. Call this function `find_startswith(last, searchstr)` and consider its natural definition below:

```python
def find_startswith(lst, searchstr):
    for s in lst:
        if s.startswith(searchstr):
            return s
    return None
```

**Question 1.** In the worst case, if `lst` has `n` elements, how many elements will `find_startswith` examine?

```python
def find_startswith(lst, searchstr):
    low = 0
    high = len(lst) - 1
    while (low < high):
        mid = (high + low) // 2
        if lst[mid].startswith(searchstr):
            return lst[mid]
        elif lst[mid] < searchstr:
            low = mid + 1
        else:
            high = mid - 1
    return None
```

**Question 2.** In the worst case, if `lst` has `n` elements, how many elements will `find_startswith` examine?