CS 106
Introduction to Computer Science I

07 / 10 / 2026

Instructor: Michael Eckmann

Today's topics

- More programmer-defined functions
- Choosing the right loop
- Searching (linear and binary search)

Programmer-defined functions

Suppose there's a lottery where the player chooses 1 number from 1 to 72. Grand prize is won if the player matches the first or second number drawn. Second prize if the player matches one of the next three numbers drawn. Nothing is won otherwise.

Let's write a function that takes in the players number, a list of grand prize numbers, and a list of second prize numbers and returns 'Grand prize', 'Second prize', or 'Sorry'

Which loop to use in different situations

1. If don't know the number of times you will iterate:
   - Use a while loop (e.g. loop until user types 'q')
2. If don't need indices
   - Use a for loop to traverse a list without indices:
     e.g. for item in mylist:
3. If need indices
   - Can use a while loop with an index variable to traverse a list
   - But it is usually preferred to use a for loop for the same purpose like:

     for i in range(len(mylist)):

contains

We'll write contains from the lab

- Which kind of loop is best to use here?

Searching

Given an item and a list, determine whether the item is in the list or not.

Linear search: compare the item to the first element of the list, (if not ==) then compare to the second and so on, until we find it (return True) or we go through the entire list and don’t find it (return False)

This is what the contains function from lab does

Searching lists (Linear search)

- Let’s analyze the linear search.
- How many comparisons does it take to find the value or to be able to say it is not found?
  - BEST CASE: What’s the minimum number of comparisons it would take?
  - WORST CASE: What’s the maximum number of comparisons it would take?

Searching

Suppose we have a sorted list (from low to high) can we make fewer comparisons for search in the worst case and on average?

Where's the best place to start your compares?

Searching

Binary search

- Given a sorted list and a key to look for
- Compare the key to the middle element of the list
  - If it is there, return True
  - If key < that middle element, only focus on the left half
  - If key > that middle element, only focus on the right half
  - Do the same thing we did to the whole list where it says to focus on half
  - return False when there is no sublist to search

Binary Search

- Let's discuss some ideas before we get right to the code.
  - What parameters might our function have?
  - What element to compare to first?
    - How do we calculate that index?
  - How do we determine what part of the array to now do a search?
- Let's take a look at an implementation and do an example call.

Binary Search

```
# function to perform binary search of a list
def binary_search(nums_list, key )
      low = 0;                  // low element subscript
      high = len(nums_list) - 1;  // high element subscript
      middle = (low + high) // 2

    # loop until low subscript is greater than high subscript
      while  low <= high:
          # determine middle element subscript
          middle = ( low + high ) // 2

          # if key matches middle element, return middle location
          if  key == nums_list[ middle ]:
              return middle
          elif key < nums_list[ middle ]:
              high = middle - 1
          else:
               low = middle + 1

     return -1
```

Searching lists (Binary search)

- Let’s analyze the binary search.
- To simplify the discussion, we can count the 2 comparisons in the if/elif/else together to be 1 comparison.
- How many comparisons does it take to find the value or to be able to say it is not found?
  - BEST CASE: What’s the minimum number of comparisons it would take?
  - WORST CASE: What’s the maximum number of comparisons it would take?