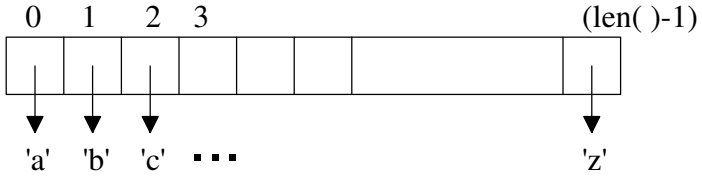
1. Most programming languages have a built-in array data structure to store a collection of same-type items. Arrays are implemented in RAM memory as a contiguous block of memory locations. Consider an array X that contains the odd integers:

address	Memory	
4000	1	X[0]
4004	3	X[1]
4008	5	X[2]
4012	7	X[3]
4016	9	X[4]
4020	11	X[5]
4024	13	X[6]
⋮		

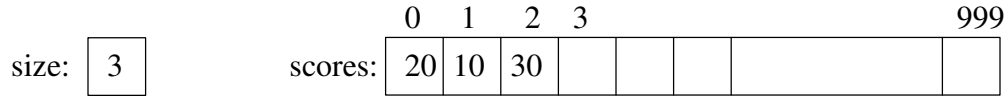
a) Any array element can be accessed randomly by calculating its address. For example, address of X[5] = 4000 + 5 * 4 = 4020. What is the general formula for calculating the address of the ith element in an array?

b) A Python list uses an array of references (pointers) to list items in their implementation of a list. For example, a list of strings containing the alphabet:



Since a Python list can contain heterogeneous data, how does storing references in the list aid implementation?

2. Arrays in most HLLs are static in size (i.e., cannot grow at run-time), so arrays are constructed to hold the "maximum" number of items. For example, an array with 1,000 slots might only contain 3 items:



a) The *physical size* of the array is the number of slots in the array. What is the physical size of scores?

b) The *logical size* of the array is the number of items actually in the array. What is the logical size of scores?

c) The *load factor* is faction of the array being used. What is the load factor of scores?

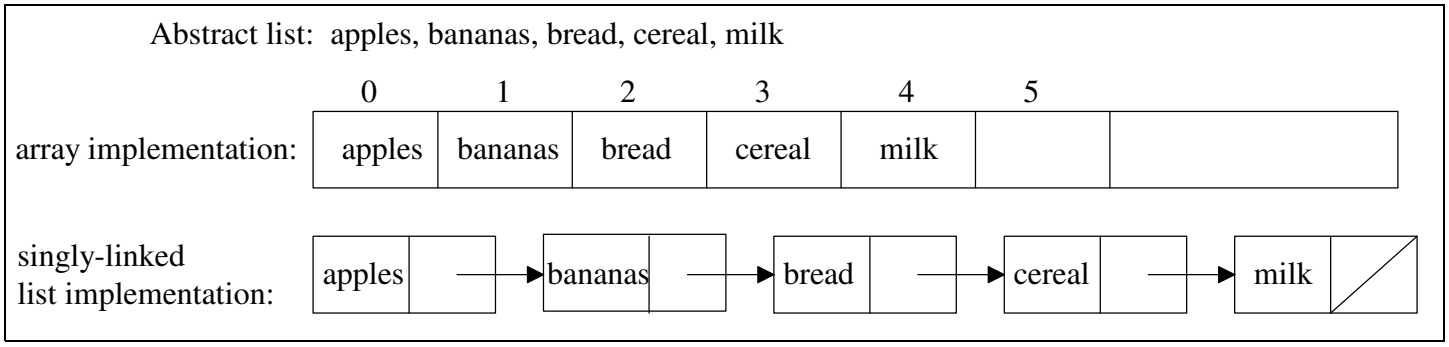
d) During run-time if an array fills up and we want to add another item, the program can usually:

- Create a bigger array than the one that filled up
- Copy all the items from the old array to the bigger array
- Add the new item
- Delete the smaller array to free up its memory

When creating the bigger array, how much bigger than the old array should it be?

e) What is the *O*( ) of moving to a larger array?

3. The items in a list can be stored in an array (/list) or as dynamically allocated nodes in a linked structure.



a) Assume a **sorted list** of "n" items. What are the time complexity for the following operations?

Operation	Array implementation		Singly-linked list implementation	
	Worst case	Average case	Worst case	Average case
Search for target item				
Retrieve item at the ith position				
Delete target item				
Delete item at the ith position				
Insert new item				
Traverse the list ("process" each item)				
Size - number of items in list				
Test for equality				

b) Assume an **unsorted list** of "n" items. What are the time complexity for the following operations?

Operation	Array implementation		Singly-linked list implementation	
	Worst case	Average case	Worst case	Average case
Search for target item				
Retrieve item at the ith position				
Delete target item				
Delete item at the ith position				
Insert new item				
Traverse the list ("process" each item)				
Size - number of items in list				
Test for equality				

c) What advantages does an array implementation have over a linked list implementation?

d) What advantages does a linked list implementation have over an array implementation?

e) How does the space utilization compare between the array and linked list implementations?