**AI LAB 2**

**Exercise 2.1.**

Complete the following code which will perform a selection sort in Python. ”...” denotes missing code that should be filled in:

|  |
| --- |
| def selection\_sort(items):      # Sorts a list of items into ascending order using selection sort algorithm      for step in range(len(items)):          # Find the location of the smallest element in items[step:]          location\_of\_smallest = step          for location in range(step, len(items)):              # Determine location of the smallest              if items[location] < items[location\_of\_smallest]:                  location\_of\_smallest = location          # Exchange items[step] with items[location\_of\_smallest]          items[step], items[location\_of\_smallest] = items[location\_of\_smallest], items[step]        return items  # Example usage of selection\_sort()  items = [64, 25, 12, 22, 11]  sorted\_items = selection\_sort(items)  print(sorted\_items)  # Output: [11, 12, 22, 25, 64] |

**Exercise 2.2.**

Considering the exercise 2.1 now convert the code for character list: [‘P’,‘Y’,‘T’,‘H’,‘O’,‘N’] and apply same selection sort.

|  |
| --- |
| def selection\_sort(chars):      n = len(chars)      # Traverse through all array elements      for i in range(n):          # Find the minimum element in remaining unsorted array          min\_idx = i          for j in range(i+1, n):              if chars[min\_idx] > chars[j]:                  min\_idx = j          # Swap the found minimum element with the first element          chars[i], chars[min\_idx] = chars[min\_idx], chars[i]      return chars  # Test the function with the given list of characters  char\_list = ['P', 'Y', 'T', 'H', 'O', 'N']  sorted\_chars = selection\_sort(char\_list)  print("Sorted list of characters using Selection Sort:", sorted\_chars) |

**Exercise 2.3.**

Write a function dups to find all duplicates in the list.

1. Create a list to read elements.
2. Pass the list as parameter to dup function.
3. Define function dup to identify duplicate elements in the list.
4. Return the final list to called function.
5. Display the result

|  |
| --- |
| **def dups(lst):**  **duplicates = []**  **for i in range(len(lst)):**  **for j in range(i+1, len(lst)):**  **if lst[i] == lst[j] and lst[i] not in duplicates:**  **duplicates.append(lst[i])**  **return duplicates**  **lst = [1, 2, 3, 4, 3, 5, 2, 6, 7, 7]**  **print(dups(lst))** |

**Exercise 2.4.**

Implement the following instructions.

1. Create a list a which contains the first three odd positive integers and a list b which contains the first three even positive integers.
2. Create a new list **c** which combines the numbers from both lists (order is unimportant).
3. Create a new list d which is a sorted copy of c, leaving c unchanged.
4. Reverse d in-place.
5. Set the fourth element of c to 42.
6. Append 10 to the end of d.
7. Append 7, 8 and 9 to the end of c.
8. Print the first three elements of c.
9. Print the last element of d without using its length.
10. Print the length of d.

|  |
| --- |
| # Step 1  a = [1, 3, 5]  b = [2, 4, 6]  # Step 2  c = a + b  # Step 3  d = sorted(c)  # Step 4  d.reverse()  # Step 5  c[3] = 42  # Step 6  d.append(10)  # Step 7  c.extend([7, 8, 9])  # Step 8  print(c[:3])  # Step 9  print(d[-1])  # Step 10  print(len(d)) |

**Exercise 2.5.**

1. Create a list a which contains three tuples. The first tuple should contain a single element, the second two elements and the third three elements.
2. Print the second element of the second element of a.
3. Create a list b which contains four lists, each of which contains four elements.
4. Print the last two elements of the first element of b.

|  |
| --- |
| # Step 1  a = [(1), (2, 3), (4, 5, 6)]  # Step 2  print(a[1][1])  # Step 3  b = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]]  # Step 4  print(b[0][-2:]) |

**Exercise 2.6**

Create a list whose elements contain the integers from 100,000 through 300,000, inclusive (be careful! what should the range be?) and set the variable big\_list to contain that list.

NOTE: Put the assignment that creates big\_list in your python file outside the scope of a function. Do **NOT** assign to big\_list in the timer or search functions because it will invalidate the timing and search results.

1. Time how long it takes the search function above to find each of the following in big\_list:
   1. element at front (100000)
   2. element in middle (200000)
   3. element at back (300000)
   4. element not in the list (3)

How long does each take?

1. If you repeat the same searches, does it take exactly the same amount of time? Why do you think this is?
2. How would you describe the relationship between the position of the item you are searching for in the list and how long it takes to find that item? (Is the search time independent of the position, or is there some mathematical relationship between the two quantities?)

Creating a list containing a range of elements

list(range(0, 10))

Linear search function to find a list element equal to the key. Note that the same structure can be used to find a list element having some other property (e.g., is even, odd, positive, negative, etc.).

def search(integer\_list, key):

for i in range(0, len(integer\_list)):

if integer\_list[i] == key:

return i

return None

Timing linear search for 700 in list(range(100,1000000)) by using time()

import time

def timer():

start = time.time()

"""

insert the code to be timed here, e.g. search(list, key)

Make sure to remove the quote marks!

"""

end = time.time()

return end – start

big\_list = list(range(100000, 300001))

|  |
| --- |
| def search(integer\_list, key):      for i in range(0, len(integer\_list)):          if integer\_list[i] == key:              return i      return None  import time  def timer():      start = time.time()        # insert the code to be timed here      search(big\_list, 200000)        end = time.time()      return end - start  #search for element at front (100000)  print(timer())  # output: 8.106231689453125e-07 (in seconds) |