FIT1045/FIT1053 Algorithmic Problem Solving – Tutorial 4.
Solutions

Objectives

After this tutorial, you should be able to:

- Sort a list using selection sort.
- Sort a list using insertion sort.
- Explain the important components of selection and insertion sort.
- Describe how functions interact.

Prepared Question

1. `x == ('Buffy', '04#', 5)` and `y == ('Tiffany', '04#', 5)`
2. The second implementation is better as it does not alter our contact list. It is preferable that we do not lose our contact list each time this function is called!

Warm-up

Do not solve these questions by running in Python.

Multiple assignment: What does `x` yield after executing the code block?

- `x, y, z = 'abc'`
- `w, x = ['horse', 'donkey']`

Mutability: What has changed after executing each function call?

- ```
  def remove_letters(lst):
      count = 0
      for i in range(len(lst)):
          if lst[i].isalpha():
              count += 1
              lst[i] = None
      return count

  my_list = ['3', 'a', '5', '2', 'b', 'd']
  num_letters = remove_letters(my_list)
  ```
- ```
  def shuffle_min(lst):
      min = lst[0]
      i = 1
      while i < len(lst):
          if lst[i] < min:
              lst[i], lst[0] = lst[0], lst[i]
              min = lst[0]
          i += 1
      return min

  my_list = [3, 6, 2, 9, 1, 4]
  minimum = shuffle_min(my_list)
  ```

Lists: What does `x` yield at the end of the code block?

- ```
  my_list = ['purple', 'blue', 'grey']
  x = my_list
  my_list.append('red')
  ```
- ```
  x = ['breakfast', 'lunch', 'dinner']
  hobbit_meals = x
  hobbit_meals.insert(1, 'elevenses')
  ```

Deep Copying: Draw a pointer/reference diagram of the *Frames* and *Objects* when the following code is executed, and answer the following:

- Is `'t2'` a deep copy of `'t1'` after execution?
- Is `'t2'` a shallow copy of `'t1'` after execution?

```
from copy import deepcopy

t1 = [['a', 'b'],
              ['c', 'd']]

t2 = deepcopy(t1)
t2[1] = t1[1]
```

Selection sort: What is the state of the list after the *first* swap that occurs when applying selection sort? (If you feel confident, practice sorting the list using selection sort.)

- `[3,6,1,8,2]`
- `[2,6,3,1,8]`

Insertion sort: What is the state of the list after the *first* swap that occurs when applying insertion sort? (If you feel confident, practice sorting the list using insertion sort.)

- `[3,6,1,8,2]`
- `[2,6,3,1,8]`

Introduction to Sorting

Selection Sort

It is possible to change step 5 so that steps 2 to 4 are repeated while `k < len(lst)-1`. Once k has reached the last element in the list, that element is by definition the smallest element in the list from k onwards. If step 2 found the minimum item in the whole list then the list would not sort. Instead, each item in the list would be swapped with the minimum item, and the minimum item would end the algorithm in the last position in the list (assuming step 5 is not changed). It is not necessary to return the sorted list as lists are *mutable*, meaning the original list will be changed by this process.

Assuming a list called `lst`, the instructions may be translated into Python as follows:

1. `k = 0`
2.
```
m = k
for i in range(k, len(lst)):
    if lst[i] < lst[m]:
        m = i
```
3. `lst[k], lst[m] = lst[m], lst[k]`
4. `k += 1`
5. `while k < len(lst):`

Insertion Sort

It is possible to change step 1 so that k starts at the second element in the list. This removes a redundant check, as there is no reason to check whether the first item in the list has a smaller item to its left. In a list already sorted in ascending order the swap in step 3 would never occur, as no item would have a bigger item on its left. In a list sorted in descending order the swap in step 3 would occur the maximum possible number of times, as every item would swap all the way to the start of the list.

Assuming a list called `lst`, the instructions may be translated into Python as follows:

1. `k = 0`
2. `j = k`
3.
```
if lst[j-1] > lst[j]:
    lst[j], lst[j-1] = lst[j-1], lst[j]
    j -= 1
```
4. `while j > 0 and lst[j-1] > lst[j]:`
5. `k += 1`
6. `while k < len(lst):`

Note that if these steps were combined, there would be redundancies.

You are the Function

No solution for this question.

You should have seen `main` call `list_names`, and `list_names` return a list of names; then `main` gives each name

in the list of names to `get_letter`, and `get_letter` gets a single letter from the given name, and passes the letter to `get_letter_position`, which turns the letter into a number that is then passed back to `get_letter`, who passes it back to `main`; `main` adds the number to a list and passes the full list to `sum` and then returns `sum`'s result.

Minimalist Sorting

It is likely you have designed an inefficent version of the algorithm Bubble sort. This algorithm is not assessable. A possible implementation is as follows:

```
def minimalist_sort(lst):
    i = 0
    while i < len(lst):
        if i > 0:
            if lst[i] < lst[i-1]:
                lst[i], lst[i-1] = lst[i-1], lst[i]
                i -= 1
            else:
                i += 1
        else:
            i += 1
```

Checkpoint

1. What is the definition of a *method*?
2. What are two *mutable* object types in Python, and two *immutable* object types?
3. Sort the following list using selection sort, showing the state of the list after each iteration of the main loop: `[9,7,9,0,6,4]`
4. Sort the following list using insertion sort, showing the state of the list after each iteration of the main loop: `[6,3,1,5,2,4]`

Solutions

1. "A *method* is a function that belongs to an object." Note: wording may be different, but key underlined concepts should be present.
2. Mutable types include: lists, sets. Immutable types include: booleans, integers, floats, tuples, strings.
3. `[9,7,6,0,6,4]`
   `[0,7,6,9,6,4]`
   `[0,4,6,9,6,7]`
   `[0,4,6,9,6,7]`
   `[0,4,6,6,9,7]`
   `[0,4,6,6,7,9]`
4. `[6,3,1,5,2,4]`
   `[3,6,1,5,2,4]`
   `[1,3,6,5,2,4]`
   `[1,3,5,6,2,4]`
   `[1,2,3,5,6,4]`
   `[1,2,3,4,5,6]`