**2023/10/21 00:00|Лекция. Практика. Часть 1.**

**Практика. Часть 1**

На этом занятии рассмотрим ситуацию, когда вы придумали крутую фишку и хотите облегчить жизнь себе и вашим коллегам.  
Представим, что вы решили сделать некую надстройку в проекте, то есть создать свой отдельный модуль. В нем будут находиться алгоритмы для сортировки списков. Таких алгоритмов на самом деле достаточно много. Несмотря на то, что они придуманы за нас, все равно их рассмотрим.  
Возьмём 2 самых простых для изучения алгоритма — это пузырьковая сортировка и сортировка выбором. Данные алгоритмы не являются наилучшим решением и имеют 1 из самых неприятных сложностей в плане выполнения по системе оценки “О” большая.

**Big O**— это определённая система, которая показывает, как изменяется сложность в зависимости от входных данных. Именно эта система оценки зачастую интересует разработчиков при выборе того или иного алгоритма. Например, когда мы собираемся перебрать список обычный с помощью цикла for, скорость выполнения или сложность данного алгоритма будет линейной, то есть O(n). В зависимости от количества элементов в списке у нас будет изменяться сложность. Однако в случае с пузырьковой сортировкой и сортировкой выбором у нас будет сложность O(n^2), так как будем использовать вложенные циклы. Это не очень хорошо, но все равно попытаемся.

**Давайте приступим к написанию функции.**  
Для начала сделаем массив. Он будет содержать в себе какие-то тестовые данные, изначально не отсортированные.

data = [5, 2, 3, 4, 1]

Именно с ним и будем работать. Создадим функцию, назовём её **bubble\_sort (пузырьковая сортировка).**

Для начала с помощью цикла for с последнего элемента по самый 1 будем перебирать наш список, однако внутри у нас будет ещё 1 цикл, который будет зависеть от значения i(рис.1).

**for** i **in** range(len(ls) - 1, 0, -1):

**for** j **in** range(i):

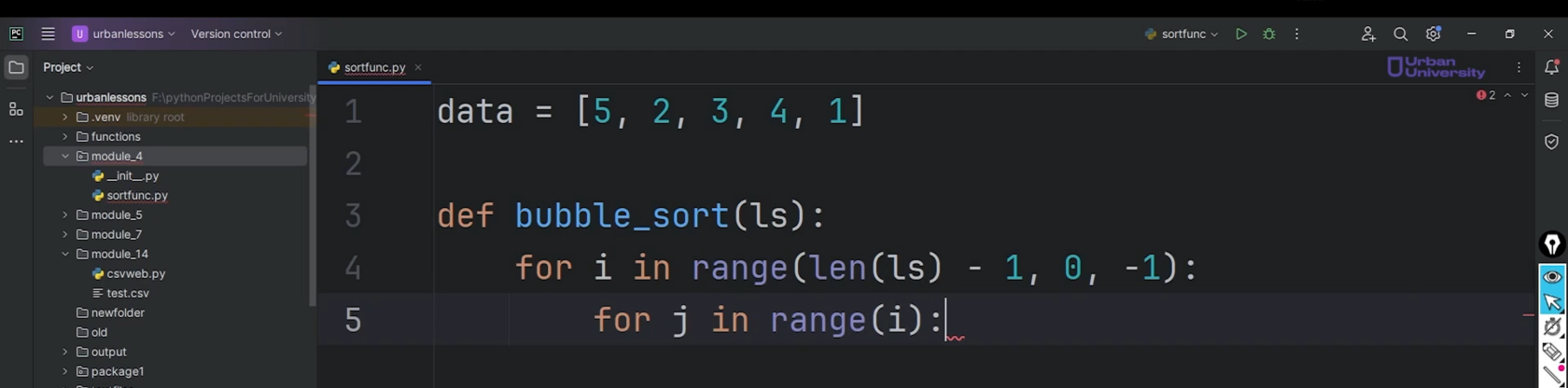


Рис.1

Дальше все достаточно просто. Берём 2 элемента, то есть какой-то элемент и стоящий за ним элемент. Если это удовлетворяет нашим условиям, то просто их меняем местами(рис.2).

**if** ls[j] > ls[j + 1]:

ls[j], ls[j +1] = ls[j +1], ls[j]

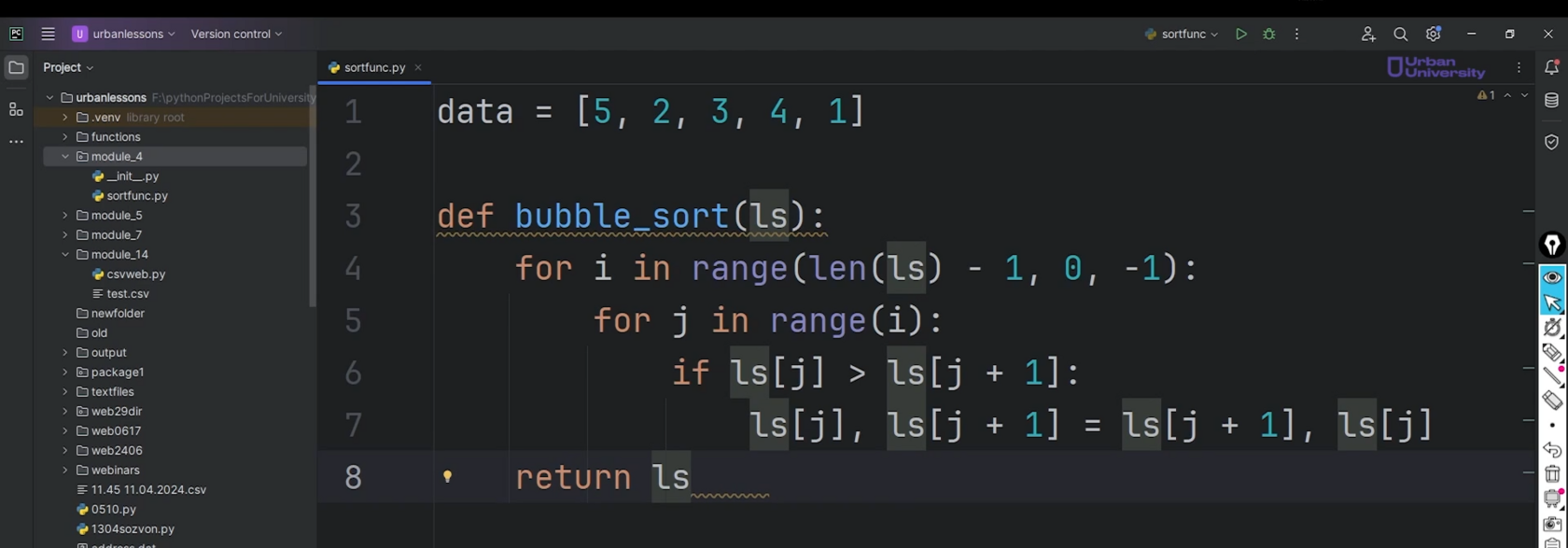


Рис.2

**Вот этот приём следует запомнить**, потому что часто можно встретить, когда хотим поменять значение в переменных. Например, у нас есть переменная ‘а’ = 5, переменная ‘b’ = 1. Если хотим поменять значения в этих переменных, то можем написать: a,b = b,a. Таким образом, значение ‘а’ будет принимать значение, которое находится в ‘b’, а значение ‘b’ будет принимать значение, которое находится в ‘a’.

Здесь будем использовать именно этот способ, то есть менять элементы местами. После выполнения этих 2 циклов будем возвращать по итогу отсортированный список. Давайте убедимся, что все работает, так как надо. У нас есть неотсортированный список(рис.3). В результате ожидаем увидеть отсортированный список. Запускаем и видим, что все отработало правильно(рис.4).

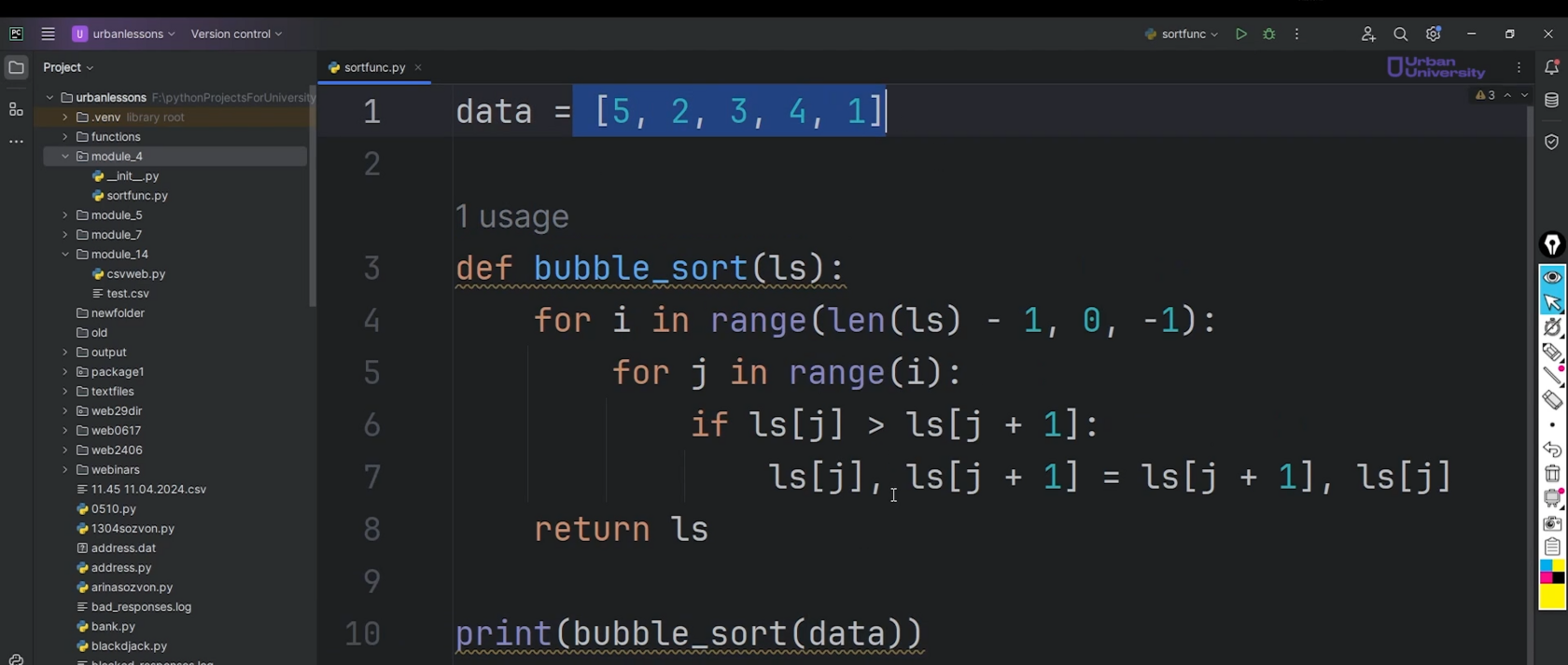


Рис.3



Рис.4

**Представим пошагово.**Для начала создали цикл. Он начинается с конца, то есть будет принимать индекс последнего элемента. Затем внутренний цикл, который будет опираться на вот это самое значение. Далее берём элемент, например, там 5 и сравниваем с 2. 5 больше, чем 2? Да, 5 двигается на второе место, 2 на первое и так далее до тех пор, пока все не встанет на свои места. Это **пузырьковая сортировка.** Сложность данного алгоритма O(n^2).

**Сортировка выбором (selection\_sort)**примерно такая же. Сложность у неё абсолютно одинаковая, но принцип работы немного другой. Циклом for от 0 до конца нашего списка - 1, как бы запускаем наш первый цикл. Затем в качестве минимального индекса сохраняем первое значение этого внешнего цикла. После этого нужно запустить второй цикл. Он будет проверять оставшиеся элементы(рис.5).

**for** i **in** **range** (len(ls) -1):

min\_index = i

**for** j **in** **range** (i+1, len(ls)):

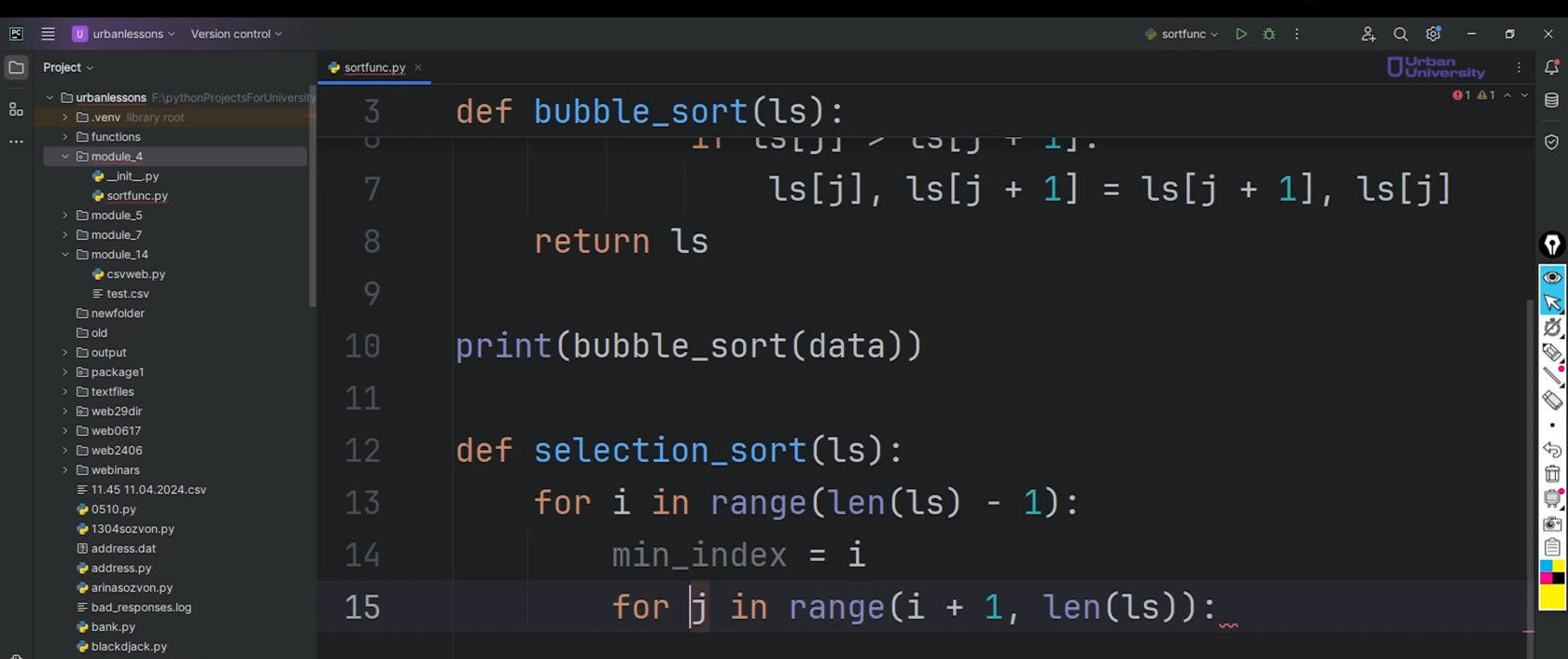


Рис.5

Далее будем брать элемент с минимальным индексом, сравнивать его с каким-то оставшимся элементом. Если он удовлетворит, скажем так, нашему условию, то есть вот этот минимальный индекс оказался больше, чем элемент из оставшегося списка, то мы переопределяем этот минимальный индекс. Однако эти элементы нам необходимо будет поменять местами(рис.6).

**If** ls[min\_index] > ls[j]:

min\_index = j

ls[min\_index, ls[j] = ls[j], ls[min\_index]

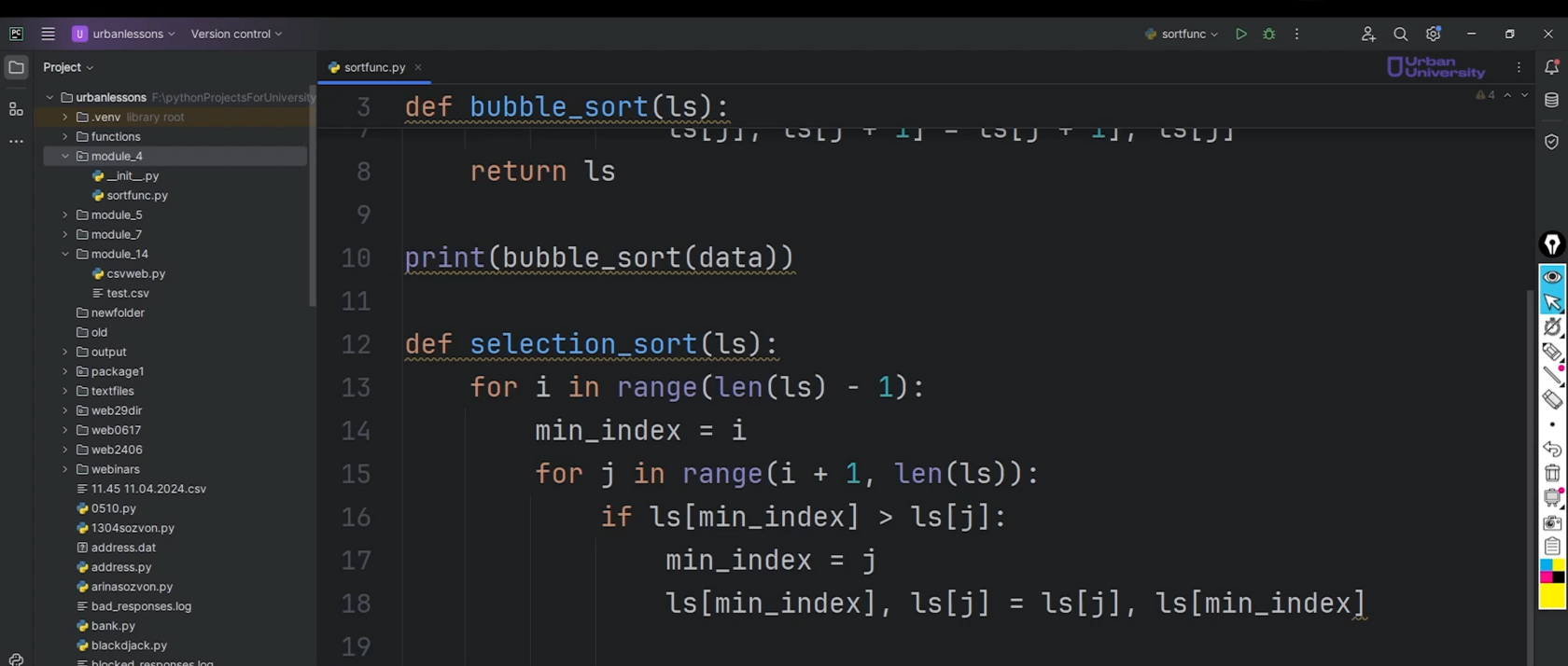


Рис.6

**Суть сортировки выбором**заключается в том, что мы опираемся на какой-то элемент и представляем, что это опорная точка. Затем проверяем элементы, идущие после него. Если находится элемент, который, допустим, меньше или больше в зависимости от того, в каком направлении вы будете выполнять сортировку, вы будете их менять местами. Постепенно будет образовываться отсортированная часть и неотсортированная. Это и есть сортировка выбора.

Давайте посмотрим после всех этих 2 циклов. Вернём отсортированный список и посмотрим, как он сработает. **В итоге получаем, что при пузырьковой сортировке, что при сортировке выбором одинаковый результат — отсортированный список**(рис.7). Значит эти две функции отрабатывают так, как надо.



Рис.7