Министерство образования и науки Российской федерации

Федеральное государственное автономное образовательное учреждение высшего образования

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Лабораторная работа №1 по дисциплине

“ Системное программное обеспечение”

Вариант №3

Выполнили:

студенты группы R3335

Кузьминов А. Ю.

Никитенко И. Р.

Гринжола Д. Ю.

Преподаватель:

кандидат технических наук

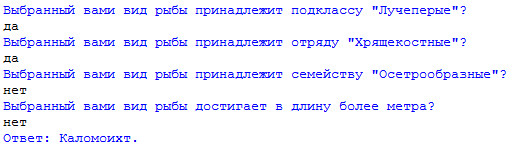
Капитонов А. А.

Санкт-Петербург

2018

Задание 1. Требовалось реализовать алгоритм фасетного поиска для определенной группы объектов, в нашем случае – «рыбы». Реализовать алгоритм не менее чем для 16 объектов. Пользователь может отвечать на вопросы только «да» и «нет».

Пример работы реализованной программы:

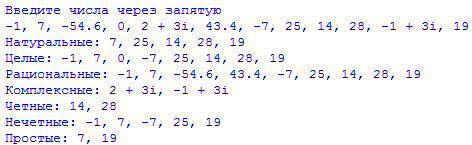


Код программы:

|  |
| --- |
| *#вывод вариантов, объяснение задания*  fishes = ['Белуга', 'Осетр', 'Многопёр', 'Каломоихт',  'Панцирник', 'Лещ', 'Сазан', 'Карась',  'Сом', 'Лосось', 'Щука', 'Целакант',  'Латимерия', 'Протоптер', 'Чешуйчатник', 'Рогозуб']  **print**('Дано:', end = ' ')  i = 0  **while** i < 15:  **print**(fishes[i], end = ', ')  i += 1  **print**(fishes[i], end = '.')  **print**()  **print**('Выберите один из представленных видов рыб, на вопросы отвечайте "да" или "нет".')  **print**()    *#вводим текст ошибки (в случае неправильно введенного пользователем ответа)*  error = 'Ошибка, попробуйте еще раз, вводите "да" или "нет".'    *#вопросы да/нет*  **while** 1:  **print**('Выбранный вами вид рыбы принадлежит подклассу "Лучеперые"?')  a = input()  **if**(a == 'да'):  **print**('Выбранный вами вид рыбы принадлежит отряду "Хрящекостные"?')  a = input()  **if**(a == 'да'):  **print**('Выбранный вами вид рыбы принадлежит семейству "Осетрообразные"?')  a = input()  **if**(a == 'да'):  **print**('Выбранный вами вид рыбы включен в Красную книгу?')  a = input()  **if**(a == 'да'):  **print**('Ответ: Белуга.')  **break**  **elif**(a == 'нет'):  **print**('Ответ: Осетр.')  **break**  **else**:  **print**(error)  **elif**(a == 'нет'):  **print**('Выбранный вами вид рыбы достигает в длину более метра?')  a = input()  **if**(a == 'да'):  **print**('Ответ: Многопёр.')  **break**  **elif**(a == 'нет'):  **print**('Ответ: Каломоихт.')  **break**  **else**:  **print**(error)  **else**:  **print**(error)  **elif**(a == 'нет'):  **print**('Выбранный вами вид рыбы принадлежит инфраклассу "Костные ганоиды"?')  a = input()  **if**(a == 'да'):  **print**('Ответ: Панцирник.')  **break**  **elif**(a == 'нет'):  **print**('Выбранный вами вид рыбы принадлежит надотряду "Костнопузырные"?')  a = input()  **if**(a == 'да'):  **print**('Выбранный вами вид рыбы принадлежит отряду "Карпообразные"?')  a = input()  **if**(a == 'да'):  **print**('Выбранный вами вид рыбы чаще всего обитает стаями?')  a = input()  **if**(a == 'да'):  **print**('Ответ: Лещ.')  **break**  **elif**(a == 'нет'):  **print**('Выбранный вами вид рыбы является всеядным?')  a = input()  **if**(a == 'да'):  **print**('Ответ: Сазан.')  **break**  **elif**(a == 'нет'):  **print**('Ответ: Карась.')  **break**  **else**:  **print**(error)  **else**:  **print**(error)  **elif**(a == 'нет'):  **print**('Ответ: Сом.')  **break**  **else**:  **print**(error)  **elif**(a == 'нет'):  **print**('Выбранный вами вид рыбы принадлежит отряду "Лососеобразные"?')  a = input()  **if**(a == 'да'):  **print**('Ответ: Лосось.')  **break**  **elif**(a == 'нет'):  **print**('Ответ: Щука.')  **break**  **else**:  **print**(error)  **else**:  **print**(error)  **else**:  **print**(error)  **elif**(a == 'нет'):  **print**('Выбранный вами вид рыбы принадлежит отряду "Целакантообразные"?')  a = input()  **if**(a == 'да'):  **print**('Выбранный вами вид рыбы является вымершим?')  a = input()  **if**(a == 'да'):  **print**('Ответ: Целакант.')  **break**  **elif**(a == 'нет'):  **print**('Ответ: Латимерия.')  **break**  **else**:  **print**(error)  **elif**(a == 'нет'):  **print**('Выбранный вами вид рыбы долгое время сохранял рекорд по размеру генома?')  a = input()  **if**(a == 'да'):  **print**('Ответ: Протоптер.')  **break**  **elif**(a == 'нет'):  **print**('Выбранный вами вид рыбы преимущественно обитает в водоемах со стоячей водой?')  a = input()  **if**(a == 'да'):  **print**('Ответ: Чешуйчатник.')  **break**  **elif**(a == 'нет'):  **print**('Ответ: Рогозуб.')  **break**  **else**:  **print**(error)  **else**:  **print**(error)  **else**:  **print**(error)  **else**:  **print**(error) |

Задача 2. Пользователь задает любое количество чисел с экрана, разделяя их запятыми. Реализовать алгоритм, который распределяет числа на натуральные, целые, рациональные, комплексные, четные, нечетные и простые. Обратите внимание, что цифры могут попадать в несколько категорий.

Пример работы реализованной программы:

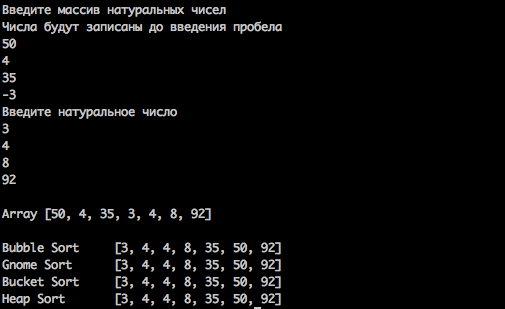


Код программы:

|  |
| --- |
| *#ввод чисел пользователем*  **print**('Введите числа через запятую')  lst = input().split(',')    '''  #заданный список чисел для проверки работоспособности программы  lst = ['-1', '7', '-54.6', '0', '2 + 3i', '43.4', '-7', '25', '14', '28', '-1 + 3i', '19']  '''    *#создание списков*  N = []  Z = []  Q = []  comp = []  even = []  odd = []  simple = []    **for** i **in** lst:  **try**:  *#нахождение натуральных чисел*  **if** float(i) > 0 **and** str(int(float(i))) == i:  N.append(i)  *#нахождение целых чисел*  **if** str(int(float(i))) == i:  Z.append(i)  *#нахождение рациональных чисел*  **if** float(i) != 0:  Q.append(i)  *#нахождение четных чисел*  **if** float(i) % 2 == 0 **and** float(i) != 0:  even.append(i)  *#нахождение нечетных чисел*  **if** float(i) % 2 == 1 **and** float(i) != 0:  odd.append(i)  **except** ValueError:  **pass**    *#нахождение комплексных чисел*  **for** num **in** lst:  **for** sym **in** num:  **if** sym == 'i':  comp.append(num)    *#нахождение простых чисел*  **for** i **in** N:  num = int(i)  div = 2  **while** num > div **and** (num % div) != 0:  div += 1  **if** num == div:  simple.append(i)    *#вывод полученных списков*  **print**('Натуральные: ', end = '')  **print**(', '.join(N))    **print**('Целые: ', end = '')  **print**(', '.join(Z))    **print**('Рациональные: ', end = '')  **print**(', '.join(Q))    **print**('Комплексные: ', end = '')  **print**(', '.join(comp))    **print**('Четные: ', end = '')  **print**(', '.join(even))    **print**('Нечетные: ', end = '')  **print**(', '.join(odd))    **print**('Простые: ', end = '')  **print**(', '.join(simple)) |

Задание 3. Пользователь задает массив натуральных чисел. Требовалось реализовать для них алгоритмы сортировки следующими методами: пузырьковый, гномий, блочный, пирамидальный. Проанализировать достоинства и недостатки данных методов.

Пример работы реализованной программы:



Код программы:

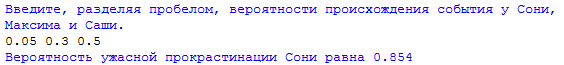
|  |
| --- |
| *#coding=utf8*  **import** random  **import** math    *#Функция ввода массива случайных натуральных чисел*  *#(для удобства тестирования)*  **def** randArr(length):  numbers = []  **for** i **in** range(length):  numbers.append(random.randint(0,100))  **return** numbers    *#Функция ввода массива натуральных чисел пользователем*  **def** arrImput():  **print**("Введите массив натуральных чисел**\n**Числа будут записаны до введения пробела")  numbers = []  **while**(1):  number = raw\_input()  **if**(number == ' '):  **break**  **else**:  **if** (int(number) < 0):  **print**("Введите натуральное число")  **else**:  numbers.append(number)  numbers = list(map(int, numbers))  **return** numbers    *#Пузырьковая сортировка*  *#Элементы от нулевого до предпоследнего сдвигаются по массиву вправо, пока не встанут слева от большего элемента*  **def** bubbleSort(numbers):  length = len(numbers)-1  **for** i **in** range(length):  **for** j **in** range(length-i):  **if**(numbers[j+1]<numbers[j]):  temp = numbers[j+1]  numbers[j+1] = numbers[j]  numbers[j] = temp  **return** numbers    *#Гномья сортировка*  **def** gnomeSort(numbers):  i = 1  *#Массив проверяется, начиная с первого элемента и до последнего*  **while** (i<len(numbers)):  *#Если элемент больше предыдущего, индекс увеливается и происходит шаг по массиву вперед*  **if**(numbers[i]>=numbers[i-1]):  i += 1  **else**:  *#Если элемент меньше предыдущего, они меняются местами*  tmp = numbers[i-1]  numbers[i-1] = numbers[i]  numbers[i] = tmp  *#И происходит шаг назад по массиву*  i -= 1  **if** (i == 0):  i = 1  **return** numbers    *#Блочная сортировка*  **def** bucketSort(numbers, depth = 0):  numSum = 0  flag = 0  *#Создается двухмерный массив блоков*  *#20 блоков для возможности работы с отрицательными числами*  buckets = []  **for** i **in** range(20):  buckets.append([])    *#Каждое число из заданного массива распределяется по блокам*  *#Индекс блока в массиве блоков для данного числа такой же, как цифра данного числа, находящаяся на порядке таком же, как глубина рекурсии*  **for** i **in** range(len(numbers)):  **if**(numbers[i] < 0): *#Для работы с отрицательными числами*  token = (-1)\*numbers[i]  flag = 1  **else**:  token = numbers[i]  flag = 0  remainder = (token // (10 \*\* depth))%10 *#Определение цифры*  numSum += remainder *#Сумма цифр всех чисел на данном уровне рекурсии*  *#Определение индексов чисел*  **if**(flag == 1):  index = 9 - remainder  **else**:  index = 10 + remainder  buckets[index].append(numbers[i])    *#Перезапись массива из блоков*  numIndex = 0  **for** i **in** range(20):  **for** j **in** range(len(buckets[i])):  numbers[numIndex] = buckets[i][j]  numIndex += 1      depth += 1  *#Если в данном порядке цифры всех чисел не равны нулю, переход на следующий уровень рекурсии - седующий порядок*  **if**(numSum != 0):  bucketSort(numbers, depth)    **return** numbers    *#Пирамидальная сортировка*  **def** heapSort (numbers):  *#Свап элементов массива*  **def** swapItems (i1, i2):  **if** numbers[i1] < numbers[i2]:  tmp = numbers[i1]  numbers[i1] = numbers[i2]  numbers[i2] = tmp    *#Спуск больших элементов вниз*  **def** siftDown (parent, limit):  **while** (1):  child = parent \* 2 + 2  *#выбирается наименьший потомок*  **if** child < limit:  **if** numbers[child] < numbers[child - 1]:  child -= 1  *#выбранный потомок свапается с предком*  swapItems(parent, child)  *#за предка принимается потомок*  parent = child  **else**:  **break**  *#Тело функции heapSort*  length = len(numbers)  *#Формирование первичной пирамиды*  **for** i **in** range((length // 2) - 1, -1, -1):  siftDown(i, length)  *#Окончательное упорядочивание*  **for** i **in** range(length - 1, 0, -1):  swapItems(i, 0)  siftDown(0, i)    **return** numbers        *#Ввод массива и вывод результатов сортировок*  numbers = arrImput()    **print**("Array "+str(numbers))    bubbleSorted = bubbleSort(numbers)  **print**("**\n**Bubble Sort **\t**"+str(bubbleSorted))    gnomeSorted = gnomeSort(numbers)  **print**("Gnome Sort **\t**"+str(gnomeSorted))    buckedSorted = bucketSort(numbers)  **print**("Bucket Sort **\t**" + str(buckedSorted))    heapSorted = heapSort(numbers)  **print**("Heap Sort **\t**" + str(heapSorted)) |

Анализ достоинств и недостатков методов сортировки:

|  |  |  |
| --- | --- | --- |
| Сортировка | Достоинства | Недостатки |
| Пузырьковая | Простота реализации | Медленная работа, сложность в худшем случае |
| Гномья | Простота реализации, не содержит второй вложенный цикл | Медленная работа, сложность в худшем случае |
| Блочная | Быстро работает при удачном наборе данных (равномерно распределенных числах) – сложность в этом случае линейная | Более сложная реализация Скорость выполнения сильно деградирует при большом количестве мало отличимых элементов |
| Пирамидальная | Сохранение линейно-логарифмической сложности при любых наборах данных | Сложная реализация  Нечувствительность к уже отсортированным частям массива – время работы не уменьшается |

Задание 4. С Соней, Максимом и Сашей часто происходит событие «ужасная прокрастинация». Зная вероятность, что данное событие происходит с ними N раз в 8 дней (N вводится пользователем при запуске программы), требовалось определить вероятность того, что за 300 дней данная ситуация произойдет только с Соней. Реализовать алгоритм для решения данной задачи.

Пример работы реализованной программы:



Код программы:

|  |
| --- |
| **print**('Введите, разделяя пробелом, вероятности происхождения события у Сони, Максима и Саши.')  C = 8  D = 300    **while** 1 == 1:  *#ввод значений вероятностей*  Sonya, Maksim, Sasha = map(float, input().split())  *#сообщение о некорректно введеных значениях*  **if** ((Sonya > 1) **or** (Maksim > 1) **or** (Sasha > 1) **or** (Sonya < 0) **or** (Maksim < 0) **or** (Sasha < 0)):  **print**('Значения вероятностей должны лежать в пределах от 0 до 1, попробуйте еще раз.')  **continue**  *#подсчет вероятности*  **else**:  result = float((1 - (1 - Sonya)\*\*(D/C))\*(1 - Maksim\*\*(D/C))\*(1 - Sasha\*\*(D/C)))  result = round(result, 3)  **print**('Вероятность ужасной прокрастинации Сони равна', result)  **break** |