# ООП. Определение операторов

# Калорийность

|  |  |
| --- | --- |
| Ограничение времени | 1 секунда |
| Ограничение памяти | 64Mb |
| Ввод | стандартный ввод или test.py |
| Вывод | стандартный вывод или output.txt |

Марина заботится о здоровом питании. Она попросила вас написать класс **FoodInfo**, экземпляры которого будут описывать пищевую ценность продуктов, а при сложении — возвращать новый экземпляр, описывающий суммарную пищевую ценность его составляющих.

Интерфейс класса:

fi = FoodInfo(proteins, fats, carbohydrates) — инициализировать экземпляр заданным количеством белков, жиров и углеводов. Все три параметра определяются весом в граммах (целым числом).

fi.get\_proteins() — вернуть количество белков.

fi.get\_fats() — вернуть количество жиров.

fi.get\_carbohydrates() — вернуть количество углеводов.

fi.get\_kcalories() — вернуть число килокалорий в пище по формуле (4 \* белки + 9 \* жиры + 4 \* углеводы).

fi\_sum = fi1 + fi2 — результат сложения fi\_sum должен быть новым объектом FoodInfo, описывающим суммарную пищевую ценность продуктов fi1 + fi2. fi1 и fi2 не должны измениться.

### Пример 1

| Ввод | Вывод |
| --- | --- |
| from solution import FoodInfo  food1 = FoodInfo(100, 100, 100)  food2 = FoodInfo(50, 60, 70)  food3 = food1 + food2  print(food1.get\_proteins(), food1.get\_fats(),  food1.get\_carbohydrates(), food1.get\_kcalories())  print(food2.get\_proteins(), food2.get\_fats(),  food2.get\_carbohydrates(), food2.get\_kcalories())  print(food3.get\_proteins(), food3.get\_fats(),  food3.get\_carbohydrates(), food3.get\_kcalories()) | 100 100 100 1700  50 60 70 1020  150 160 170 2720 |

### Пример 2

| Ввод | Вывод |
| --- | --- |
| from solution import FoodInfo  food1 = FoodInfo(1, 2, 3)  food2 = FoodInfo(10, 20, 30)  food3 = food1 + food2  print(food3.get\_proteins(), food3.get\_fats(),  food3.get\_carbohydrates(), food3.get\_kcalories()) | 11 22 33 374 |

### Пример 3

| Ввод | Вывод |
| --- | --- |
| from solution import FoodInfo  food1 = FoodInfo(1, 2, 3)  food2 = FoodInfo(10, 20, 30)  food3 = food1 + food2  food4 = food2 + food1  print(food3.get\_proteins(), food3.get\_fats(),  food3.get\_carbohydrates(), food3.get\_kcalories())  print(food4.get\_proteins(), food4.get\_fats(),  food4.get\_carbohydrates(), food4.get\_kcalories()) | 11 22 33 374  11 22 33 374 |

# Список в обратном порядке

Kлассная работа

макс. 1 балл.

|  |  |
| --- | --- |
| Ограничение времени | 1 секунда |
| Ограничение памяти | 64Mb |
| Ввод | стандартный ввод или test.py |
| Вывод | стандартный вывод или output.txt |

Напишите класс **ReversedList**, который будет при инициализации экземпляра принимать список и реализовывать доступ к элементам этого списка в обратном порядке.

rl = ReversedList(lst) — создание обратного списка.

len(rl) — число элементов в обратном списке.

rl[i] — доступ к элементам в обратном порядке. rl[0] — последний элемент первоначального списка, rl[1] — предпоследний и так далее.

### Пример 1

| Ввод | Вывод |
| --- | --- |
| from solution import ReversedList  rl = ReversedList([10, 20, 30])  for i in range(len(rl)):  print(rl[i]) | 30  20  10 |

### Пример 2

| Ввод | Вывод |
| --- | --- |
| from solution import ReversedList  rl = ReversedList([])  print(len(rl)) | 0 |

### Пример 3

| Ввод | Вывод |
| --- | --- |
| from solution import ReversedList  rl = ReversedList([10])  print(len(rl))  print(rl[0]) | 1  10 |

## Примечания

Функцию len переопределять не нужно, используйте специальный метод.

# Квадратичная функция

Kлассная работа

макс. 2 балл.

|  |  |
| --- | --- |
| Ограничение времени | 1 секунда |
| Ограничение памяти | 64Mb |
| Ввод | стандартный ввод или test.py |
| Вывод | стандартный вывод или output.txt |

Реализуйте класс **SquareFunction**, экземпляры которого при инициализации получают коэффициенты a, b, c. При вызове объекта как функции с аргументом x должно возвращаться значение выражения ax2 + bx + c.

### Пример 1

| Ввод | Вывод |
| --- | --- |
| from solution import SquareFunction  sf = SquareFunction(1, 0, 0)  print(sf(-2))  print(sf(-1))  print(sf(-0))  print(sf(1))  print(sf(2))  print(sf(10)) | 4  1  0  1  4  100 |

### Пример 2

| Ввод | Вывод |
| --- | --- |
| from solution import SquareFunction  sf = SquareFunction(1, 2, 1)  print(sf(-2))  print(sf(-1))  print(sf(-0))  print(sf(1))  print(sf(2))  print(sf(10)) | 1  0  1  4  9  121 |

### Пример 3

| Ввод | Вывод |
| --- | --- |
| from solution import SquareFunction  sf = SquareFunction(0, 0, 1)  print(sf(-2))  print(sf(-1))  print(sf(-0))  print(sf(1))  print(sf(2))  print(sf(10)) | 1  1  1  1  1  1 |

# Вычитание дат

Kлассная работа

макс. 2 балл.

|  |  |
| --- | --- |
| Ограничение времени | 1 секунда |
| Ограничение памяти | 64Mb |
| Ввод | стандартный ввод или test.py |
| Вывод | стандартный вывод или output.txt |

Реализуйте класс **Date**, экземпляры которого при инициализации принимают месяц и день.

При вычитании дат (d1 - d2) должно возвращаться число дней между d1 и d2.

Число дней должно быть: равно нулю, если d1 и d2 — одна и та же дата, быть больше нуля, если d1 позже d2, быть меньше нуля, если d1 раньше d2.  
Считайте, что все даты указаны в пределах одного и того же не вискосного года (в феврале 28 дней).

### Пример 1

| Ввод | Вывод |
| --- | --- |
| from solution import Date  jan5 = Date(1, 5)  jan1 = Date(1, 1)  print(jan5 - jan1)  print(jan1 - jan5)  print(jan1 - jan1)  print(jan5 - jan5) | 4  -4  0  0 |

### Пример 2

| Ввод | Вывод |
| --- | --- |
| from solution import Date  mar5 = Date(3, 1)  jan1 = Date(1, 1)  print(mar5 - jan1)  print(jan1 - mar5)  print(jan1 - jan1)  print(mar5 - mar5) | 59  -59  0  0 |

# Точки на плоскости

Kлассная работа

макс. 3 балл.

|  |  |
| --- | --- |
| Ограничение времени | 1 секунда |
| Ограничение памяти | 64Mb |
| Ввод | стандартный ввод или test.py |
| Вывод | стандартный вывод или output.txt |

Определите класс **Point**. При инициализации экземпляру передаются координаты x и y.

При сравнении двух экземпляров оператор == должен возвращать True, если координаты точек равны, и False — если нет.  
При сравнении оператором != должно возвращаться True, если координаты точек не равны, и False — если равны.

### Пример 1

| Ввод | Вывод |
| --- | --- |
| from solution import Point  p1 = Point(1, 2)  p2 = Point(5, 6)  if p1 == p2:  print("Equal True")  else:  print("Equal False")  if p1 != p2:  print("Not equal True")  else:  print("Not equal False") | Equal False  Not equal True |

### Пример 2

| Ввод | Вывод |
| --- | --- |
| from solution import Point  p1 = Point(0, 0)  p2 = Point(0, 0)  if p1 == p2:  print("Equal True")  else:  print("Equal False")  if p1 != p2:  print("Not equal True")  else:  print("Not equal False") | Equal True  Not equal False |

### Пример 3

| Ввод | Вывод |
| --- | --- |
| from solution import Point  p1 = Point(0, 10)  p2 = Point(0, 0)  if p1 == p2:  print("Equal True")  else:  print("Equal False")  if p1 != p2:  print("Not equal True")  else:  print("Not equal False") | Equal False  Not equal True |

# Разреженный массив

Kлассная работа

макс. 2 балл.

|  |  |
| --- | --- |
| Ограничение времени | 1 секунда |
| Ограничение памяти | 64Mb |
| Ввод | стандартный ввод или test.py |
| Вывод | стандартный вывод или output.txt |

Периодически в некоторых задачах возникает надобность в разреженных массивах. Разреженный массив характеризуется тем, что подавляющее большинство значений в нём равны нулю, поэтому можно хранить только ненулевые значения. Это позволяет создавать разреженные массивы очень большого размера без лишних затрат памяти.

Вам необходимо реализовать класс разреженного массива **SparseArray**.

arr = SparseArray() — создание пустого разреженного массива.

arr[i] = value — записать значение в массив.

arr[i] — прочитать значение из массива.

### Пример 1

| Ввод | Вывод |
| --- | --- |
| from solution import SparseArray  arr = SparseArray()  arr[1] = 10  arr[8] = 20  for i in range(10):  print('arr[{}] = {}'.format(i, arr[i])) | arr[0] = 0  arr[1] = 10  arr[2] = 0  arr[3] = 0  arr[4] = 0  arr[5] = 0  arr[6] = 0  arr[7] = 0  arr[8] = 20  arr[9] = 0 |

### Пример 2

| Ввод | Вывод |
| --- | --- |
| from solution import SparseArray  arr = SparseArray()  arr[10] = 123  for i in range(8, 13):  print('arr[{}] = {}'.format(i, arr[i])) | arr[8] = 0  arr[9] = 0  arr[10] = 123  arr[11] = 0  arr[12] = 0 |

### Пример 3

| Ввод | Вывод |
| --- | --- |
| from solution import SparseArray  def print\_elem(array, ind):  print('arr[{}] = {}'.format(ind, array[ind]))  arr = SparseArray()  index = 1000000000  arr[index] = 123  print\_elem(arr, index - 1)  print\_elem(arr, index)  print\_elem(arr, index + 1) | arr[999999999] = 0  arr[1000000000] = 123  arr[1000000001] = 0 |

# Сложение многочленов

Kлассная работа

макс. 2 балл.

|  |  |
| --- | --- |
| Ограничение времени | 1 секунда |
| Ограничение памяти | 64Mb |
| Ввод | стандартный ввод или test.py |
| Вывод | стандартный вывод или output.txt |

Реализуйте класс **Polynomial** для вычисления значений многочленов при заданных x, а также для сложения многочленов.

poly = Polynomial(coefficients) — создать новый многочлен с коэффициентами coefficients. coefficients[0] — свободный член, coefficients[1] — множитель при x1, coefficients[2] — множитель при x2, и так далее.

y = poly(x) — вычислить значение многочлена в точке x.

y = c0 + c1 x1 + c2 x2 + ... + cn xn

poly\_sum = poly1 + poly2 — результат сложения двух многочленов должен быть экземляром **Polynomial** и содержать коэфициенты, которые получаются в результате сложения коэффициентов poly1 и poly2 при одинаковых степенях x.

### Пример 1

| Ввод | Вывод |
| --- | --- |
| from solution import Polynomial  poly = Polynomial([10, -1])  print(poly(0))  print(poly(1))  print(poly(2)) | 10  9  8 |

### Пример 2

| Ввод | Вывод |
| --- | --- |
| from solution import Polynomial  poly1 = Polynomial([0, 0, 1])  print(poly1(-2))  print(poly1(-1))  print(poly1(0))  print(poly1(1))  print(poly1(2))  print()  poly2 = Polynomial([0, 0, 2])  print(poly2(-2))  print(poly2(-1))  print(poly2(0))  print(poly2(1))  print(poly2(2))  print()  poly3 = poly1 + poly2  print(poly3(-2))  print(poly3(-1))  print(poly3(0))  print(poly3(1))  print(poly3(2))  print() | 4  1  0  1  4  8  2  0  2  8  12  3  0  3  12 |

### Пример 3

| Ввод | Вывод |
| --- | --- |
| from solution import Polynomial  poly1 = Polynomial([0, 1])  poly2 = Polynomial([10])  poly3 = poly1 + poly2  poly4 = poly2 + poly1  print(poly3(-2), poly4(-2))  print(poly3(-1), poly4(-1))  print(poly3(0), poly4(0))  print(poly3(1), poly4(1))  print(poly3(2), poly4(2)) | 8 8  9 9  10 10  11 11  12 12 |

# Очередь и все-все-все

Kлассная работа

макс. 3 балл.

|  |  |
| --- | --- |
| Ограничение времени | 1 секунда |
| Ограничение памяти | 64Mb |
| Ввод | стандартный ввод или testmodule.py |
| Вывод | стандартный вывод или output.txt |

Программист Николай, изучая различные структуры данных, решил погрузиться в теорию очередей, построенных на базе списков Python.

Напомним, что очередь – это такая структура данных, работа с которой определяется принципом FIFO (First In First Out). Наблюдать этот принцип можно у кассы в магазине. Если, конечно, никто не нарушает правила :-)

Николай предлагает Вам реализовать класс Queue, инкапсулирующий (то есть моделирующий работу) очередь.

Однако, Николай не ищет легких путей и хочет усложнить Вам работу: **очередь может быть инициализирована**различным числом элементов.

Например,

q = Queue(1, 2, 3, 4, 5) – создаёт очередь [1 -> 2 -> 3 -> 4 -> 5], а

q1 = Queue(1, 2, 3) – очередь [1 -> 2 -> 3].

Но всё же есть гарантия того, что для инициализации передается хотя бы один параметр.

**Класс Queue реализует следующие методы:**

append(∗values)append(∗values) – добавляет несколько значений в конец очереди (как минимум одно).

copy()copy() – создаёт копию данной очереди, то есть возвращает новую очередь, полностью аналогичную исходной.

pop()pop() – вытаскивает и возвращает первый элемент очереди, при этом этот элемент из очереди удаляется. Если очередь пуста, то возвращает None.

extend(queue)extend(queue) – расширяет данную очередь другой, то есть приклеивает вторую очередь к первой.

next()next() – возвращает новую очередь, начинающуюся со второго элемента текущей.

**Также требуется реализовать следующие операторы и встроенные** **функции:**

queue1+queue2queue1+queue2 – склейка очередей создаёт новую увеличенную очередь.

queue1+=queue2queue1+=queue2 – расширяет первую очередь второй.

queue1==queue2queue1==queue2 – проверяет очереди на равенство всех элементов. Возвращает True или False.

queue>>Nqueue>>N – создаёт новую очередь без первых N (вышедших) элементов. В случае, когда N превышает количество элементов очереди, следует вернуть пустую очередь.

str(queue)str(queue) – приводит очередь к строке вида [q1 -> q2 -> q3 -> ... -> qn]. Пустая или ошибочная очередь – это []

next(queue)next(queue) – аналогичное действие методу next()next().

## Пример

| Ввод | Вывод |
| --- | --- |
| from solution import Queue  q1 = Queue(1, 2, 3)  print(q1)  q1.append(4, 5)  print(q1)  qx = q1.copy()  print(qx.pop())  print(qx)  q2 = q1.copy()  print(q2)  print(q1 == q2, id(q1) == id(q2))  q3 = q2.next()  print(q1, q2, q3, sep = '\n')  print(q1 + q3)  q3.extend(Queue(1, 2))  print(q3)  q4 = Queue(1, 2)  q4 += q3 >> 4  print(q4)  q5 = next(q4)  print(q4)  print(q5) | [1 -> 2 -> 3]  [1 -> 2 -> 3 -> 4 -> 5]  1  [2 -> 3 -> 4 -> 5]  [1 -> 2 -> 3 -> 4 -> 5]  True False  [1 -> 2 -> 3 -> 4 -> 5]  [1 -> 2 -> 3 -> 4 -> 5]  [2 -> 3 -> 4 -> 5]  [1 -> 2 -> 3 -> 4 -> 5 -> 2 -> 3 -> 4 -> 5]  [2 -> 3 -> 4 -> 5 -> 1 -> 2]  [1 -> 2 -> 1 -> 2]  [1 -> 2 -> 1 -> 2]  [2 -> 1 -> 2] |

## Примечания

В файле решения необходимо предоставить только созданный класс. Никаких вызовов делать не надо.

Данная задача дополнительно проверяется преподавателем.