Обратите внимание: не используйте внешний модуль Python в этом решении, то есть не импортируйте какой-либо модуль для решения! В частности, не используйте модуль math или модуль itertools.

Часть I: Итераторы и генераторы

Эта часть состоит из нескольких независимых задач. Целостность ввода можно принять во всех задачах. Все функции должны быть включены в один файл ex8\_iterators.py.

1. Напишите класс с именем reverse\_iter, который выполняет обратный итератор для списков. Пример использования:

>>> it **=** reverse\_iter([1, 2, 3, 4])  
>>> it**.**next()  
4  
>>> it**.**next()  
3  
>>> it**.**next()  
2  
>>> it**.**next()  
1  
>>> it**.**next()

(В конце итератора возвращается None).

1. Только для целей этого раздела вы можете использовать случайный пакет, с которым вы столкнулись ранее. Запустите генератор с именем roll\_pair\_of\_dice, который возвращает tuple со случайным броском из двух кубов. Пример использования:

>>> roll\_iter = roll\_pair\_of\_dice()

>>> print(next(roll\_iter))

(2, 4)

>>> print(next(roll\_iter))

(6, 5)

1. Реализуйте функцию range, в форме генератора. Имя функции будет myRange. Конечно, сам range или что-то еще не следует использовать, чтобы сделать реализацию тривиальной. Также обратите внимание, что вы применяете все варианты range (включая, например, итерацию от высокого к низкому путем отрицательных скачков). Вы можете предположить, что вход правильный (целые числа, а скачок не равен нулю). Когда дополнительных объектов нет, итератор должен выкинуть ошибку, как обычный оператор, используя команду

raise StopIteration()

**Часть II: Программирование второго порядка**

Вы должны представить все необходимое для реализации этой части в файле ex8\_2nd\_order.py

1. Использует функцию power\_function (base), которая возвращает функцию, которая вычисляет значение basis в аргументе аргумента. Пример использования:

>>> my\_function = power\_function(2)

>>> print(my\_function(4))

16 # since 2\*2\*2\*2 equals 16.

1. Использует функцию diff\_function (g, h), которая возвращает функцию, являющуюся разницей между функциями: f(x) = g(x) - h(x)

# Identity function

def g(x):

return x

# Squared function

def h(x):

return x\*x

diff = diff\_function(g,h)

print(“g(5)-h(5) = “, diff(5))

1. Использует функцию get\_contained\_area\_func(x0, x1, num\_segments) которая возвращает функцию contained\_area (G, H), которая вычисляет площадь поверхности между этими двумя функциями в поле [x0, x1] с использованием метода сумм Римана:

1). Сначала разделите сегмент на равные сегменты num\_sements

2). Для каждого сегмента (например, [x0, x0 + delta]) мы вычисляем разницу между g и h функциями в конце сегмента. Примечание. Разница может быть отрицательной, если g ниже h, но область должна быть положительной.

3). Это среднее значение, умноженное на ширину сегмента (дельта в примере), будет близко к области, заключенной между двумя функциями.

4). Общая площадь будет вычисляться по схеме во всех сегментах, и это значение будет возвращено функцией.

// f1,g1,f2,g2,... are some mathematical functions that are defined (at least) in the range [-5,5]

area\_func = get\_contained\_area\_func(x0=-5, x1=5, num\_segments=100)

print(“The area between f1 and g2 is: “, area\_func(f1, g1))

print(“The area between g1 and g2 is: “, area\_func(f2, g2))

...

1. Используйте функцию binary\_search (f, x0, x1, epsilon), которая получает две точки x0 и x1, одна из которых является положительной и одна отрицательная, и возвращает точку в поле [x0, x1], где f равно 0 с точностью до epsilon. Решение намного более эффективно, чем переход от x0 к x1 в прыжках epsilon

Запуск тестирования для проверки кода:

import ex8tests

ex8tests.main()