1. Создайте в PyCharm новый проект типа Pure Python и **новый файл типа Python file «A\_increase.py».**

2. В файле «A\_increase.py» создайте класс объектов-итераторов (содержащих метод \_\_next\_\_) RangeIterator для прохода по последовательности строк из решеток возрастающей длины:

# класс объектов-итераторов

# для указания текущего элемента

# в последовательности

class RangeIterator:

# при создании создает переменные

# для хранения текущего состояния

def \_\_init\_\_(self, size):

self.x = 0

self.size = size

# единственное, что можно делать с итератором -

# двигать его на следующий элемент последовательности

# это то, что делает итератор итератором

def \_\_next\_\_(self):

self.x += 1

# если последовательность итератора

# не бесконечна - должно быть условие

# для окончания цикла

if self.x > self.size:

# для окончания цикла в итераторе

# вызывается специальное исключение StopIteration

raise StopIteration

# возвращает элементы последовательности -

# строки из решеток заданной длины

return '#' \* self.x

3. В файле «A\_increase.py» создайте класс итерируемых объектов (содержащих метод \_\_iter\_\_) RangeIterable для применения итератора класса RangeIterator в цикле:

# класс итерируемых объектов

class RangeIterable:

def \_\_init\_\_(self, size):

self.size = size

# для прохода по итерируемому объекту с помощью цикла

# в начале цикла вызывается метод \_\_iter\_\_

# для получения итератора

# это то, что делает объект итерируемым

def \_\_iter\_\_(self):

return RangeIterator(self.size)

4. В файле «A\_increase.py» напишите главную программу для создания последовательности (итерируемого объекта класса RangeIterable) из 32 строк возрастающей длины и поэлементного вывода ее с помощью цикла через короткие интервалы времени:

import time

# главная программа с заголовком,

# позволяющим использовать этот файл как модуль:

if \_\_name\_\_ == '\_\_main\_\_':

# создание итерируемого объекта

main\_iter = RangeIterable(32)

# проход по итерируемому объекту с помощью цикла

for line in main\_iter:

# вывод текущего элемента (который возвращает итератор)

print(line)

# задержка между выводами

time.sleep(0.25)

5. Запустите написанную в файле «A\_increase.py» программу, полюбуйтесь рисунком в консоли.

6. На практике для упрощения кода часто объединяют классы итератора и итерируемого объекта (в одном классе располагают методы \_\_iter\_\_ и \_\_next\_\_). Сделаем это **в новом файле «B\_decrease.py».**

7. В файле «B\_decrease.py» создайте для последовательностей строк из решеток убывающей длины класс RangeIterableIterator, который будет одновременно и итерируемым объектом и итератором по самому себе:

# класс объектов, которые одновременно являюстся и

# итерируемыми (есть метод \_\_iter\_\_),

# и итераторами (есть метод \_\_next\_\_);

# преимущества:

# 1. один класс вместо двух - меньше писать и выдумывать названий

# 2. одно общее состояние - не надо передавать переменные

class RangeIterableIterator:

def \_\_init\_\_(self, size):

self.x = size

# раз сам себе итератор - сам себя и возвращает

def \_\_iter\_\_(self):

return self

def \_\_next\_\_(self):

self.x -= 1

if self.x <= 0:

raise StopIteration

return '#' \* self.x

8. В файле «B\_decrease.py» повторите главную программу из файла «A\_increase.py» с итерируемым объектом нового класса RangeIterableIterator.

9. Запустите написанную в файле «B\_decrease.py» программу, полюбуйтесь рисунком в консоли.

10. Дальнейшим упрощением работы с итерируемыми объектами является применение генераторов вместо итераторов. Для работы с генераторами создайте **новый файл «C\_sinus.py».**

11. В файле «C\_sinus.py» создайте для бесконечных последовательностей строк из решеток длиной, определяемой функцией синус, класс SinusIterableWithGenerator, который будет итерируемым объектом, возвращающим в методе \_\_iter\_\_ генератор с помощью оператора yield:

# класс итерируемых объектов,

# которые возвращают генератор вместо итератора

class SinusIterableWithGenerator:

def \_\_init\_\_(self, steps, width):

# запоминаем растянутость синусоиды по высоте и ширине

self.steps = steps

self.width = width

# метод получения итератора

def \_\_iter\_\_(self):

# локальная переменная внутри метода (не класса)

# для хранения состояния

x = 0.0

# в данном случае - бесконечный цикл,

# но мог бы быть с условием окончания последовательности

while True:

x += math.pi \* 2 / self.steps

length = self.width / 2 + self.width / 2 \* math.sin(x) + 1

# вместо return используется yield

# для создания генератора - объекта, который

# сохраняет переменные (х) и логику (цикл while)

# текущего метода (\_\_iter\_\_); логика выполняется

# при вызове метода \_\_next\_\_ у созданного генератора;

# преимущества:

# 1. не надо писать метод \_\_next\_\_ самим;

# 2. не надо хранить переменные в классе, дописывая self.

yield '#' \* int(length)

12. В файле «C\_sinus.py» повторите главную программу из файла «A\_increase.py» с итерируемым объектом нового класса SinusIterableWithGenerator.

13. Запустите написанную в файле «C\_sinus.py» программу, полюбуйтесь рисунком в консоли. Для остановки используйте кнопку с изображением красного квадрата.

14. Существует модуль itertools, предназначенный для выполнения распространенных операций над итерируемыми последовательностями, например, их объединения. **В новом файле «D\_chain.py»** импортируйте созданные в предыдущих заданиях итерируемые объекты, объедините их в один и запустите проход циклом по нему:

import time

from A\_increase import RangeIterable

from B\_decrease import RangeIterableIterator

from C\_sinus import SinusIterableWithGenerator

import itertools

# создаем цепочку итераторов из предыдущих заданий

main\_iter = itertools.chain(

RangeIterableIterator(32),

RangeIterable(16),

SinusIterableWithGenerator(64, 32)

)

for line in main\_iter:

print(line)

time.sleep(0.25)

15. Одним из видов итерируемых объектов являются строки текстовых файлов при чтении. Скопируйте в папку с проектом файл «food.csv», содержащий таблицу со строками стихотворения для игры в «съедобное-несъедобное» с названиями предметов в первой колонке и признак съедобности этих предметов во второй. **В новом файле** создайте программу для построчного вывода строк стихотворения в консоль, приема и проверки ответов пользователя:

import io

# главная программа:

# создание итерируемого объекта - строк файла

main\_iter = io.open('food.csv')

# вывод инструкций

print("Игра \"Съедобное-несъедобное\".")

print("Вам будут по очереди выводится названия предметов.")

print("Вводите 0 если предмет несъедобный и 1 - если несъедобный.")

print("Для начала нажмите Enter.")

input()

score = 0

# проход по итерируемому объекту с помощью цикла

for line in main\_iter:

# разбиение строки на ячейки по точкам с запятой

cells = line.split(';')

# вывод текущего элемента

print(cells[0])

# ввод ответа

inbuf = input()

# проверка правильности ответа

if inbuf[0] == cells[1][0]:

print("Правильно!")

score += 1

else:

print("Неправильно!")

# вывод набранных очков

print("Вы набрали " + str(score) + " очков.")

print("КОНЕЦ.")

16. Запустите созданную игру и сыграйте в нее.

17. Еще одним из видов итерируемых объектов являются перестановки и другие последовательности, генерируемые методами математического пакета NumPy. Скопируйте в папку с проектом файл «food2.csv», содержащий таблицу с названиями предметов из предыдущего задания в первой колонке и признак съедобности этих предметов во второй. **В новом файле** создайте программу для создания случайной перестановки номеров строк файла и вывода названий предметов в полученном случайном порядке в консоль, приема и проверки ответов пользователя:

import io

import numpy

# главная программа:

# создание итерируемого объекта - строк файла

load\_iter = io.open('food2.csv')

# преобразуем итератор в списки

lines = []

isFood = []

for line in load\_iter:

# разбиение строки на ячейки по точкам с запятой

cells = line.split(';')

# сохранение текущего элемента в списках

lines += [cells[0]]

isFood += [cells[1]]

# создаем случайную перестановку

# из порядковых номеров загруженных предметов,

# перестановка - итерируемый объект

main\_iter = numpy.random.permutation(len(lines))

# вывод инструкций

print("Игра \"Съедобное-несъедобное\".")

print("Вам будут по очереди выводится названия предметов.")

print("Вводите 0 если предмет несъедобный и 1 - если несъедобный.")

print("Для начала нажмите Enter.")

input()

score = 0

# проход по итерируемому объекту с помощью цикла

for num in main\_iter:

# вывод текущего элемента

print(lines[num])

# ввод ответа

inbuf = input()

# проверка правильности ответа

if inbuf[0] == isFood[num][0]:

print("Правильно!")

score += 1

else:

print("Неправильно!")

# вывод набранных очков

print("Вы набрали " + str(score) + " очков.")

print("КОНЕЦ.")

18. Запустите созданную игру и сыграйте в нее.