Пояснительная записка

# Описание полученного задания (вариант 12, функция 11):

Создать консольное приложение, обрабатывающее данные в контейнере.

* Обобщенный артефакт, используемый в задании - животные.
* Базовые альтернативы и уникальные параметры - рыбы (место проживания – перечислимый тип: река, море, озеро…), птицы (отношение к перелету: перелетные, остающиеся на зимовку – булевская величина), звери (хищники, травоядные, насекомоядные… – перечислимый тип).
* Общие для всех альтернатив переменные - название (строка символов), вес в

граммах (целое).

* Функция для обработки данных в контейнере - упорядочить элементы контейнера по убыванию используя сортировку с помощью прямого выбора (Straight Selection). В качестве ключей для сортировки и других действий используются результаты функции, общей для всех альтернатив.
* Функция, общая для всех альтернатив - частное от деления суммы кодов

незашифрованной строки на вес (действительное число).

# Структура проекта:

project/  
 .idea/

\_\_pychache\_\_/   
tests/

main.py container.py animal.py

# Работа программы:

Пользователь может ввести данные в контейнер двумя способами:

* Задать входной файл с тестовыми данными с помощью команды

-f \*input path\* \*output path\* , где \*input path\* и \*output path\* - пути входного и выходного файлов соответственно.

* Задать параметры для генерации тестовых данных:

-n \*size\* \*files path\* , где \*size\* - количество элементов в контейнере, \*files path\* - путь, куда будут сохранены сгенерированный файл с входными данными (\*\*\*\_generated.txt) и с обработанными выходными данными (\*\*\*\_generated.output.txt).

Результаты тестирования программы сохранены в папке /tests. Тесты 1-7 с использованием входных данных из файла, тесты 8-14 с использованием сгенерированных входных данных.

Примеры использованных команд:

./project -f /tests/test1 /tests/test1\_output

./project -n 100 /tests/test14

# Основные характеристики программы:

* Число модулей = 3
* Общий размер исходных текстов ≈ 10.3 КБ
* Время работы программы для различных тестовых наборов данных:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Тип входных данных | Кол-во элементов в контейнере | Время работы программы с использованием динамической типизации Python (мс) | Время работы программы со структурами C (мс) | Время работы программы с классами C++ (мс) |
| Входные данные из файла | 10 | 1.386 | 0.385 | 0.442 |
| 20 | 3.801 | 0.418 | 0.521 |
| Входные данные генерируются | 100 | 43.298 | 0.929 | 0.989 |
| 1000 | 2630.548 | 25.607 | 33.843 |
| 5000 | 63081.304 | 723.708 | 763.432 |
| 10000 | 212349.851 | 3162.394 | 3538.942 |

Python – интерпретируемый язык программирования, в отличие от рассмотренных ранее компилируемых. Несомненно, это сильно влияет на время работы программы, так как интерпретатор выполняет текст программы без предварительного перевода, он должен анализировать каждый оператор в программе каждый раз, когда он выполняется, а затем выполнить желаемое действие, тогда как скомпилированный код просто выполняет действие в фиксированном контексте, определяемом компиляцией.

В отличие от статической типизации, используемой в С и С++, где при объявлении переменной необходимо указывать ее тип, в Python используется динамическая типизация.

var = 5

var = 4.9

var = “five”

В ходе работы программы переменная может менять свой тип, поэтому при динамической типизации тратится дополнительное время на проверку типов переменных.

# Инструментальные средства:

* Виртуальная машина Oracle VM VirtualBox
  + Класс ОС: Linux
  + Версия ОС: Ubuntu (64-bit)
  + Кол-во процессоров виртуальной машины: 2
  + Оперативная память: 4096 МБ
* Языки программирования: Python 3.8
* Импортируемы модули: sys, time, random, string

Интегрированная среда разработки: PyCharm

# Структура классов:

**Bird**

+migration: bool

+def \_\_init\_\_(self, name, weight, migration)  
+def \_\_str\_\_(self)  
+def out\_generated(self, input\_g\_file)

@staticmethod  
+def in\_from\_file(input\_file)  
+def in\_rnd() -> 'Bird'

**Subtype(Enum)**

**Area(Enum)**

**Kind(Enum)**

**Container**

+animals: list

+def \_\_init\_\_(self)  
+def \_\_str\_\_(self)  
+def add(self, animal) -> bool  
+def out(self, output\_file)  
+def out\_generated(self, input\_g\_file)  
+def straight\_selection(self)

@staticmethod  
+def in\_from\_file(input\_file)  
+def in\_rnd(amount)

**Animal**

+name: str  
+weight: int

+def \_\_init\_\_(self, name, weight)  
+def \_\_str\_\_(self)  
+def count\_quotient(self) -> float

@staticmethod  
+def in\_from\_file(input\_file)  
+def in\_rnd()  
+def rand\_name() -> str

@abstractmethod   
+def out\_generated(self, input\_g\_file)

**Beast**

+subtype: Subtype

+def \_\_init\_\_(self, name, weight, subtype)  
+def \_\_str\_\_(self)  
+def out\_generated(self, input\_g\_file)

@staticmethod  
+def in\_from\_file(input\_file)  
+def in\_rnd() -> 'Beast'

**Fish**

+area: Area

+def \_\_init\_\_(self, name, weight, area)  
+def \_\_str\_\_(self)  
+def out\_generated(self, input\_g\_file)

@staticmethod  
+def in\_from\_file(input\_file)  
+def in\_rnd() -> 'Fish'