

Пояснительная записка

| Алиева Рената Эдуардовна ФКН ПИ ВШЭ БПИ202.

| Вариант 42. Артефакт номер 14. Функция номер 3.

Описание полученного задания:

Данная программа описывает вычисление максимального расстояния, которое может пройти автомобиль(условие задачи 14) а также выполняет сортировку контейнера пузырьком (Bubble Sort) .В качестве ключей для сортировки и других действий используются результаты функции, вычисляющей максимальное расстояние(обработка данных в контейнере под номером 3). Программа выполнена на динамически типизированном языке Python. В программе реализованы такие классы, как автомобильный транспорт(car), грузовик(truck) – содержит грузоподъемность; емкость топливного бака; расход топлива, автобус(bus) – содержит пассажироместность; емкость топливного бака; расход топлива, легковой автомобиль(automobile) – содержит максимальную скорость; емкость топливного бака; расход топлива.

Работа программы:

Программа ожидает одну из команд:

- `python3 main.py -f infile outfile01 outfile02` ,где `infile` – имя файла в котором хранятся входные данные, `outfile01` – имя файла в котором будут выходные данные,`outfile02` – имя файла в котором будут выходные данные, после выполнения программой функцией(В данном случае Bubble Sort).
- `python3 main.py -n number outfile01 outfile02` – похожая команда, в которой `number` – кол-во артефактов, которые необходимо сгенерировать с помощью функции `rnd()`. Остальные параметры такие же, как и в первой команде.

Ввод в программу в файле реализован следующим образом:

- Первый параметр – целое число от 1 до 3 обозначающее тип машины: 1 – грузовик, 2 – автобус, 3 - автомобиль. Второй параметр – индивидуальный

параметр(описан выше для каждого типа машины),емкость топливного бака, расход топлива.

Программа протестирована на 9 файловых теста(расположены в папке input_tests),результаты которых расположены в папке output_tests(файл типа test01_out.txt и test01_sorted.txt означают файл с заполненными и сортированными элементами соответственно).

Основные характеристики:





- Число интерфейсных модулей - 0
- Число модулей реализации(без main) - 6
- Общий размер текстов - 11 КБ
- Размер результатов тестов - 3280 КБ

Структурная схема:



Python - динамически типизированный язык, что означает что размер, который занимает переменная точно понять нельзя \Rightarrow в следующей таблице в колонке "Размер" будет написано **dynamic size**, то есть неопределенный размер.В колонке тип будет стоять предположительный(то есть тот, который предполагается для работы программы) тип переменной.

Таблица типов.

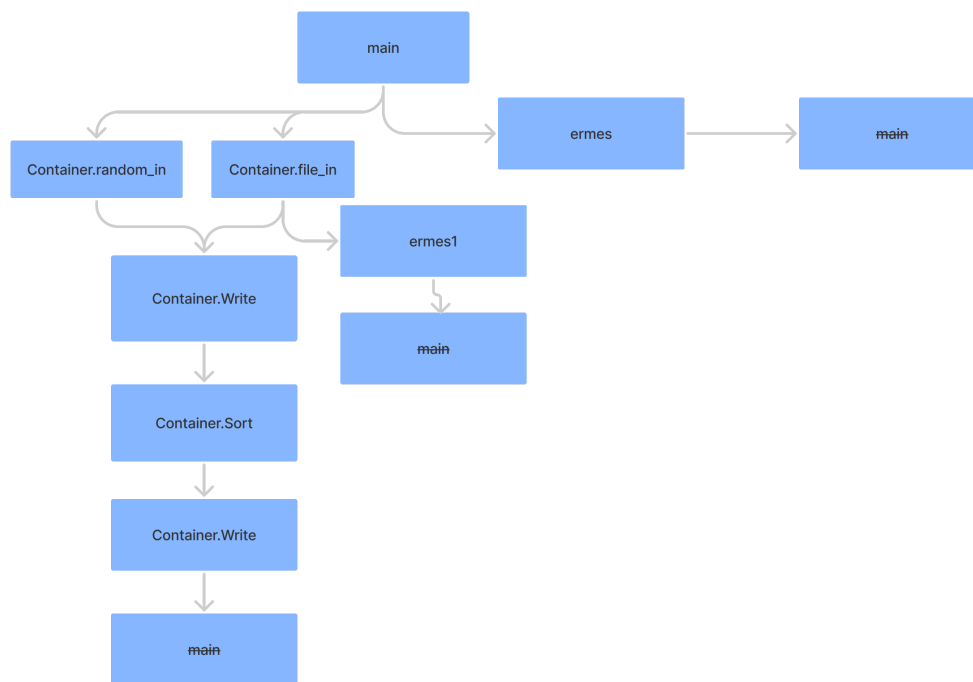
 Класс	 Имя	 Тип	 Размер
<u>Car</u>	fuel_capacity	int	dynamic size
<u>Untitled</u>	fuel_capacity	int	dynamic size
<u>Bus</u>	passenger_capacity	int	dynamic size
<u>Untitled</u>	поля класса Car	int,int	dynamic size
<u>Truck</u>	lifting_capacity	int	dynamic size
<u>Untitled</u>	поля класса Car	int,int	dynamic size

Аа Класс	☰ Имя	☰ Тип	▼ Размер
<u>Automobile</u>	max_speed	int	dynamic size
<u>Untitled</u>	поля класса Car	int,int	dynamic size
<u>Container</u>	store	list[]	dynamic size

Таблица классов

Аа Класс	☰ Имя
<u>Car</u>	def Distance(self) def __init__(self fuel_capacity fuel_consumption)
<u>Bus</u>	def Print(self) def Write(self ostream) def __init__(self passenger_capacity fuel_consumption fuel_capacity)
<u>Truck</u>	def Print(self) def Write(self ostream) def __init__(self lifting_capacity fuel_consumption fuel_capacity)
<u>Automobile</u>	def Print(self) def Write(self ostream) def __init__(self max_speed fuel_consumption fuel_capacity)
<u>Container</u>	def Print(self) def Sort(self) def Write(self ostream) def __init__(self) def file_in(self strArray) def random_in(self figureNumbers)

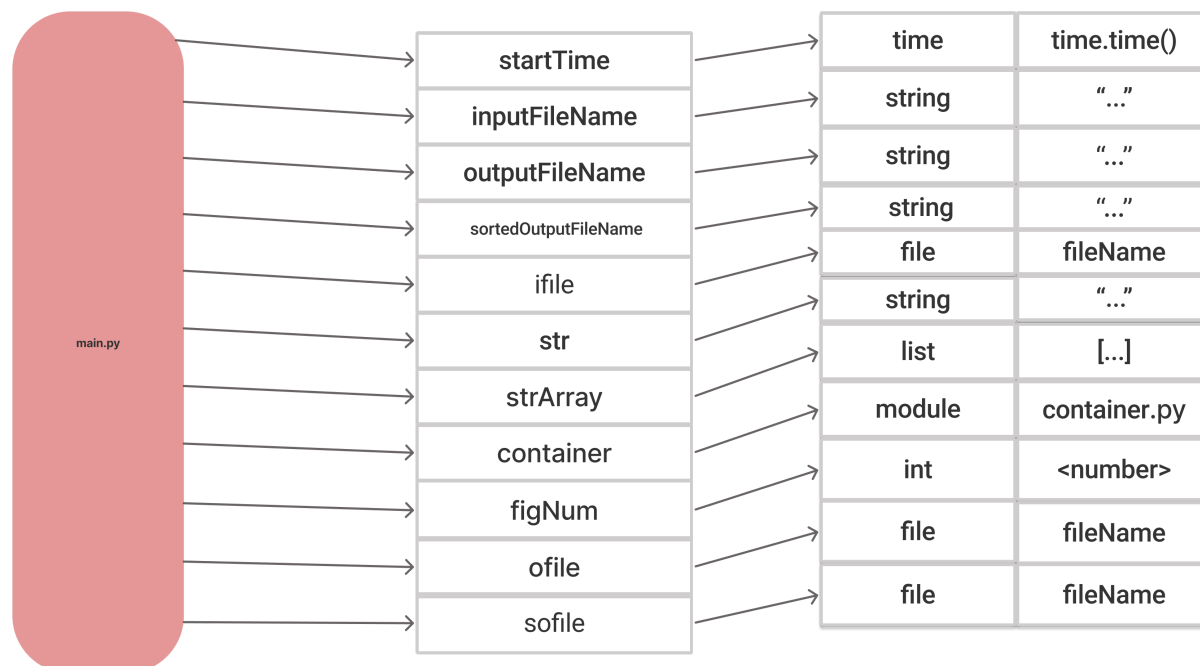
Стек(Main())



Память программы

Таблица имен

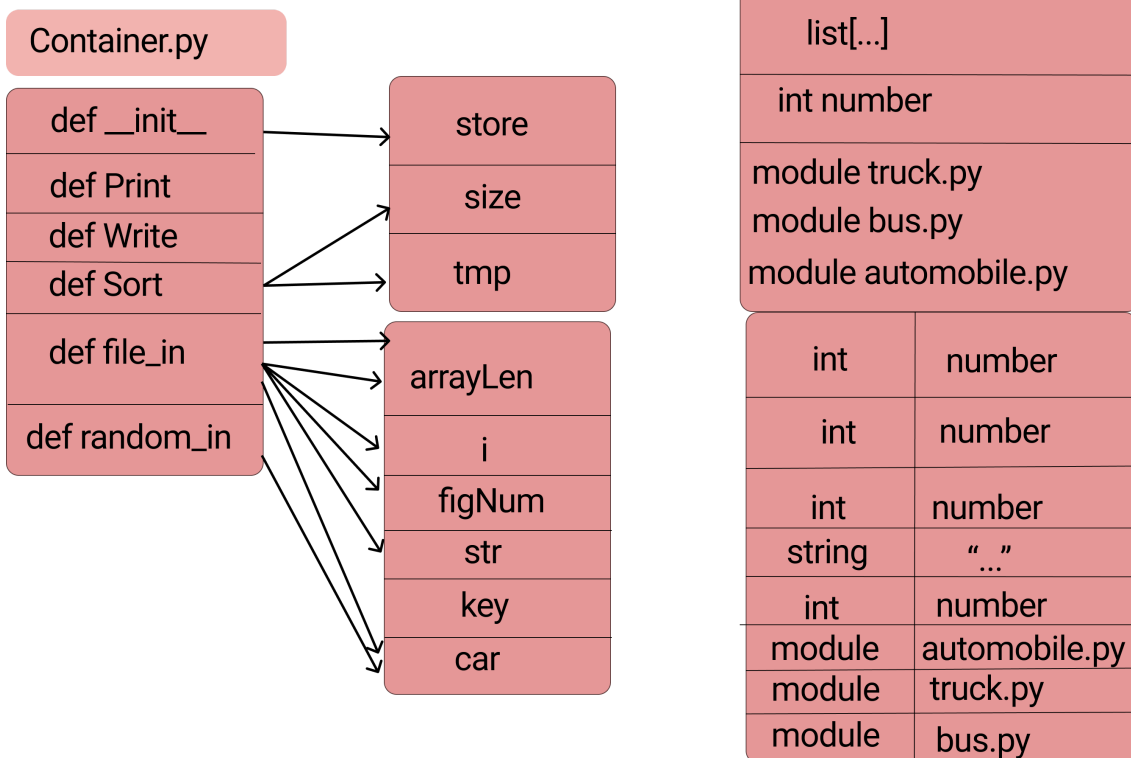
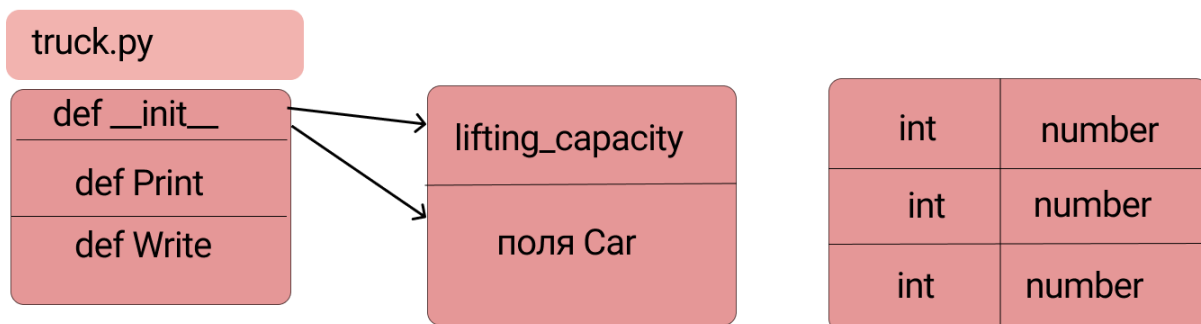
Память данных

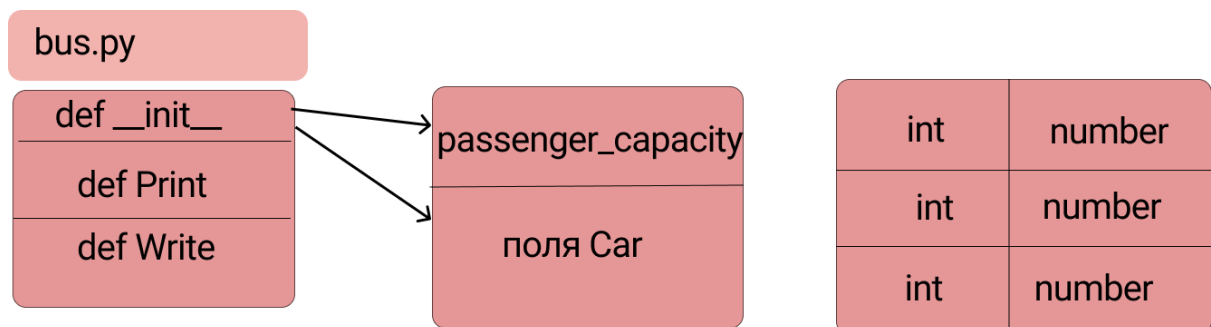
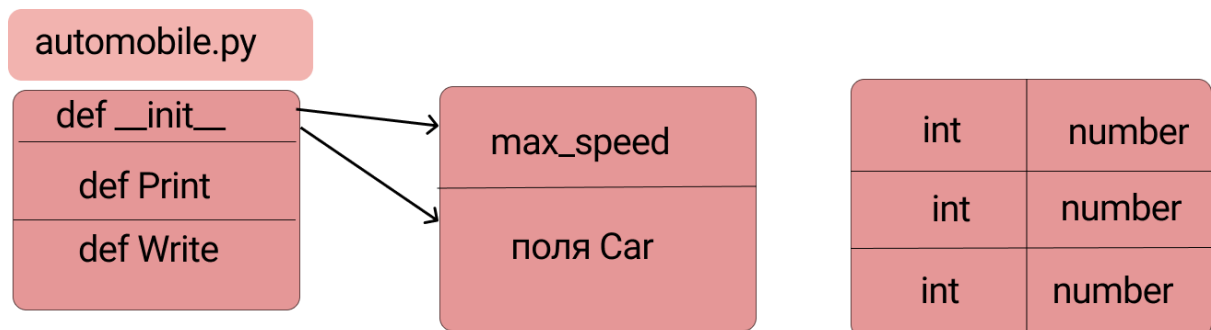
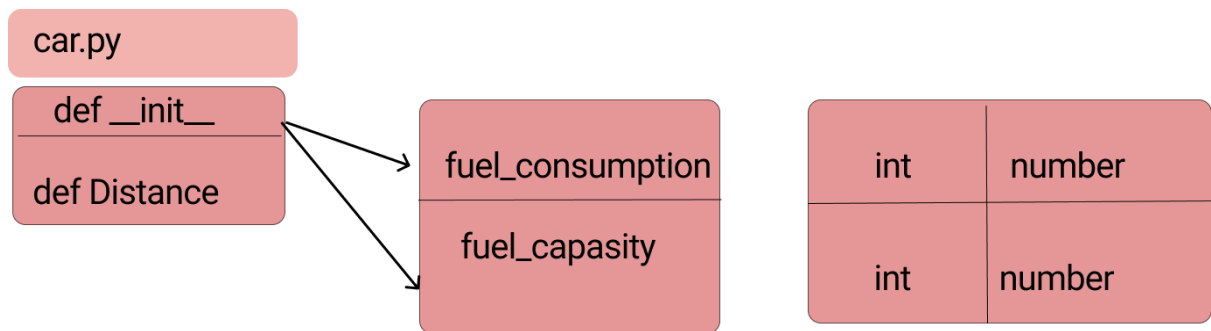


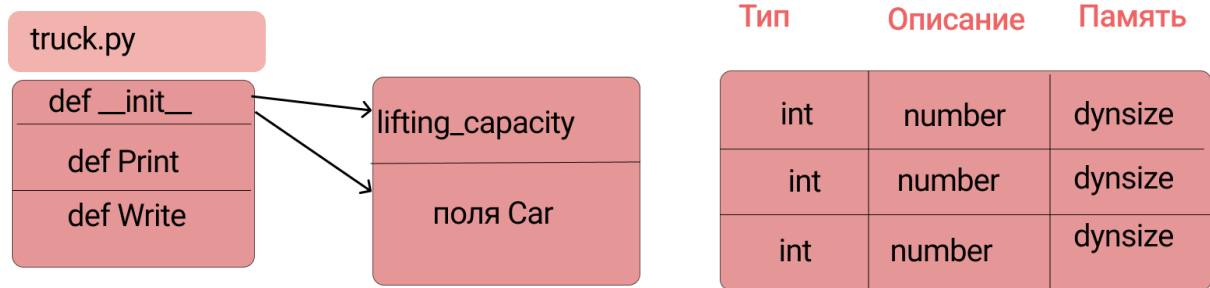
Память программы

Таблица имен

Описание







Метрики

Время работы программы

Тест	Размер	Время в секундах	Время выполнения прошлой программы
test01.txt	8	0.0009973 seconds	0.0013 seconds
test02.txt	12	0.0019951 seconds	0.0123233
test03.txt	1000	1.2845616 seconds	1.5654213
test04.txt	5000	19.7960272 seconds	22.3456123
test05.txt	9999	92.182657 seconds	95.3451123



Анализируя время работы программы написанной в стиле ООП и на динамически типизированном языке, можно заметить что данная программа работает быстрее, в отличие от прошлой. Причиной этому может быть то, что прошлая программа была написана на более низкоуровневом языке.

Вывод

Использование модульного программирования позволяет упростить тестирование программы и обнаружение ошибок, так как структура и поведение подчиняются определённым правилам и модули работают независимо. Естественно на более высокоуровневом языке как Python программисту будет легче писать код, так как в данном случае не надо задумываться о том какой тип нужен той или иной переменной и понимать это сами. Но если проект весьма большой, это может понести ряд проблем, из-за которых будет трудно

отследить какой тип может попасть в ту или иную переменную и решить эту проблему будет труднее.