**Описание полученного задания**

*Номер варианта задания*: 255

*Номер условия задачи*: 3

*Объёмная (трёхмерная) геометрическая фигура*:

1. Шар (целочисленный радиус)

2. Параллелепипед (три целочисленных ребра)

3. Правильный тетраэдр (длина ребра – целое)

*Общая переменная*: плотность материала фигуры (действительное число)

*Общая функция*: вычисление объёма (действительное число)

*Номер дополнительной функции*: 19

*Описание функции*: Удалить из контейнера те элементы, для которых значение, полученное с использованием функции, общей для всех альтернатив, меньше, чем среднее арифметическое для всех элементов контейнера, полученное с использованием этой же функции.

*Входные данные:* программа запускается через командную строку, входные данные передаются через пробел в виде «python\_main <inputFileName> [<outputFileName1>] [<outputFileName2>]» (где inputFileName – путь к считываемому файлу, outputFileName1 (опционально) – путь к выводимому файлу до выполнения дополнительной функции, outputFileName2 (опционально) – путь к выводимому файлу после выполнения дополнительной функции) для считывания из файла или «python\_main <number> [<outputFileName1>] [<outputFileName2>]» (где number – количество генерируемых фигур, outputFileName1 (опционально) – путь к выводимому файлу до выполнения дополнительной функции, outputFileName2 (опционально) – путь к выводимому файлу после выполнения дополнительной функции)

*Выходные данные*: выводятся в путь, указанный при запуске программы: первый файл содержит количество фигур, их типы, характеристики и объём; второй – содержит фигуры, которые подходят под условие дополнительной функции, их типы, характеристики и объём.

|  |  |  |  |
| --- | --- | --- | --- |
| **Память программы** | **Таблица имён** | **Память данных** | |
| main.py | ifile | file | filename |
| str | string | “…” |
| strArray | list | […] |
| cont | list | […] |
| container | module | Container.py |
| ofile | file | Filename |
| fig\_num | int | <number> |
| container.py | module | extender.py |
| input\_file\_name | string | “…” |
| output\_file\_name | string | “…” |

|  |  |  |  |
| --- | --- | --- | --- |
| **Память программы** | **Таблица имён** | **Память данных** | |
| container.py |  |  |  |
| def read | cont | list | […] |
| str\_array | list | […] |
| i | int | <number> |
| fig\_num | int | <number> |
| str | string | “…” |
| key | int | <number> |
| ball\_item | list | [ball, int, int] |
| parallelepiped\_item | list | [parallelepiped, int, int, int, int] |
| tetrahedron\_item | list | [tetrahedron, int, int] |
| ball | module | ball.py |
| parallelepiped | module | parallelepiped.py |
| tetrahedron | module | tetrahedron.py |
| def random\_generate | container\_item | list | […] |
| count | int | <number> |
| fig\_num | int | <number> |
| I | int | <number> |
| key | int | <number> |
| ball\_item | list | [ball, int, int] |
| parallelepiped\_item | list | [parallelepiped, int, int, int, int] |
| tetrahedron\_item | list | [tetrahedron, int, int] |
| ball | module | ball.py |
| parallelepiped | module | parallelepiped.py |
| tetrahedron | module | tetrahedron.py |
| def write | cont | list | […] |
| ostream | TextIO | open(output\_file\_name1/2) |
| shape | list | [ball, int, int]  [parallelepiped, int, int, int, int]  [tetrahedron, int, int] |
| ball\_item | list | [ball, int, int] |
| parallelepiped\_item | list | [parallelepiped, int, int, int, int] |
| tetrahedron\_item | list | [tetrahedron, int, int] |
| ball | module | ball.py |
| parallelepiped | module | parallelepiped.py |
| tetrahedron | module | tetrahedron.py |
| def volume | cont | list | […] |
| volume\_value | int | <number> |
| shape | list | [ball, int, int]  [parallelepiped, int, int, int, int]  [tetrahedron, int, int] |
| ball | module | ball.py |
| parallelepiped | module | parallelepiped.py |
| tetrahedron | module | tetrahedron.py |
| def volume\_filter | cont | list | […] |
| volume\_value | int | <number> |
| new\_container | list | […] |
| shape | list | [ball, int, int]  [parallelepiped, int, int, int, int]  [tetrahedron, int, int] |
| ball | module | ball.py |
| parallelepiped | module | parallelepiped.py |
| tetrahedron | module | tetrahedron.py |

|  |  |  |  |
| --- | --- | --- | --- |
| **Память программы** | **Таблица имён** | **Память данных** | |
| ball.py |  |  |  |
| def read | ball | list | [ball, int, int] |
| str\_array | string | “…” |
| i | int | <number> |
| def read\_random | ball | list | [ball, int, int] |
| def write | ball | list | [ball, int, int] |
| ostream | TextIO | open(output\_file\_name1/2) |
| def volume | ball | list | [ball, int, int] |

|  |  |  |  |
| --- | --- | --- | --- |
| **Память программы** | **Таблица имён** | **Память данных** | |
| parallelepiped.py |  |  |  |
| def read | parallelepiped | list | [parallelepiped, int, int, int, int] |
| str\_array | string | “…” |
| i | int | <number> |
| def read\_random | parallelepiped | list | [parallelepiped, int, int, int, int] |
| def write | parallelepiped | list | [parallelepiped, int, int, int, int] |
| ostream | TextIO | open(output\_file\_name1/2) |
| def volume | parallelepiped | list | [parallelepiped, int, int, int, int] |

|  |  |  |  |
| --- | --- | --- | --- |
| **Память программы** | **Таблица имён** | **Память данных** | |
| tetrahedron.py |  |  |  |
| def read | tetrahedron | list | [tetrahedron, int, int] |
| str\_array | string | “…” |
| i | int | <number> |
| def read\_random | tetrahedron | list | [tetrahedron, int, int] |
| def write | tetrahedron | list | [tetrahedron, int, int] |
| ostream | TextIO | open(output\_file\_name1/2) |
| def volume | tetrahedron | list | [tetrahedron, int, int] |

*Число модулей реализации*: 6

*Общий размер исходных текстов*: 10,4 КБ

*Размер исполняемого кода:* 7,46 КБ

*Тест 1*: 0.09375 с

*Тест 2*: 0.125 с

*Тест 3*: 0.078125 с

*Тест 4*: 0.09375 с

*Тест 5*: 0.078125 с

В сравнении с другими реализациями программы динамическая типизация позволяет использовать одну и ту же переменную для разных типов данных, универсальность, помогает избежать дублирования кода для разных типов и легко программировать.

В то же время статическая типизация является более надёжной, код более читабелен, проверки типов происходят только один раз — на этапе компиляции, а значит – статически типизированные языки быстрее динамически типизированных, пресекание потенциальных ошибок.