**Лабораторная работа №15**

# Выбор и обоснование выбора среды разработки программы. Изучение различных стилей программирования, правил формирования листинга программы.

**Цель работы:**

1. Изучить критерии выбора языка программирования. Научиться обосновывать выбор среды разработки в соответствии с критериями выбора языка программирования.
2. Изучить основные парадигмы программирования.
3. Изучить правила формирования листинга программы.

# Порядок выполнения работы:

* 1. Выполните задания 1-3.

# Задание 1. Критерии выбора языка программирования.

* + 1. -Скорость работы конечного продукта.

-Объем занимаемой оперативной памяти.

-Скорость разработки программы.

-Ориентированность на компьютер или человека.

-Кроссплатформенность.-Скорость внесения изменений, скорость тестирования.

Для создания backend части мы будем использовать язык программирования Python. Для создания frontend части мы будем использовать JavaScript. Для управления базой данных будем использовать PostgresSQL.

2.

# Задание 2. Изучение парадигм программирования

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Название парадигмы** | **Сущность парадигмы, основные идеи, принципы, объекты.** | **Языки**  **поддерживаю щие данную парадигму. (2-3 языка)** | **Достоинства** | **Недостатки** | **Примеры программ** |
| **1.** | [**.**](https://ru.wikipedia.org/wiki/%D0%98%D0%BC%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)[**программирование**](https://ru.wikipedia.org/wiki/%D0%98%D0%BC%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) | * в исходном коде программы записываются [инструкции](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) (команды); * инструкции должны выполняться последовательно; * данные, получаемые при выполнении предыдущих инструкций, могут читаться из памяти последующими инструкциями; * данные, полученные при выполнении инструкции, могут записываться в память. | С++, С#, Java | Это самый простой метод программирования для изучения и понимания, поэтому обычно его обучают первым. Также проще отлаживать императивную программу, потому что вы управляете всеми шагами: сообщаете программе, что вы хотите и как это получить.  Главное преимущество императивного программирования состоит в том, что вы можете контролировать сложность программы. | Он основан на порядке выполнения программы, поэтому в некоторых случаях его сложно организовать и требует большего количества кода для выполнения простых задач, что означает, что для более сложных программ код становится очень объемным и теряет свою ясность из-за его размер.  При изменении кода, например, для добавления функций или его оптимизации, высок риск ошибок.  В императивном программировании мы описываем, что делать и как этого добиться. Следовательно, необходимо описать все слои программы. | Python  L = [1, 2, 3, 4, 5]  sum = 0  for x in L:  sum += x |
| **2.** | [**Декларативное**](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BA%D0%BB%D0%B0%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)[**программирование**](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BA%D0%BB%D0%B0%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) | [парадигма программирования](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%B4%D0%B8%D0%B3%D0%BC%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), в которой задаётся [спецификация](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B5%D1%86%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F) решения задачи, то есть описывается, что представляют собой проблема и ожидаемый результат. | * Agda, FP, Mercury. | Проекты разрабатываются на готовых шаблонах, поэтому проекты разрабатывать проще. | Проекты получаются менее функциональными |  |
| **3.** | [**Структурное**](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)[**программирование**](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) | [парадигма программирования](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%B4%D0%B8%D0%B3%D0%BC%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), в основе которой лежит представление программы в виде иерархической структуры [блоков](https://ru.wikipedia.org/wiki/%D0%91%D0%BB%D0%BE%D0%BA_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)). | С, Pascal, Basic | 1) повышается надежность программ (благодаря хорошему структурированию при проектировании, программа легко поддается тестированию и не создает проблем при отладке);  2) повышается эффективность программ (структурирование программы позволяет легко находить и корректировать ошибки, а отдельные подпрограммы можно переделывать (модифицировать) независимо от других);  3) уменьшается время и стоимость программной разработки;  4) улучшается читабельность программ. | Главный недостаток структурного подхода заключается в следующем: процессы и данные существуют отдельно друг от друга (как в модели деятельности организации, так и в модели программной системы), причем проектирование ведется от процессов к данным. Таким образом, помимо функциональной декомпозиции, существует также структура данных, находящаяся на втором плане. | **Pascal**  **Program** arr\_2;  **var** m, n, sum : integer;  arr : **array** [1..10, 1..10] **of** integer;  **begin**  **repeat**  writeln('Введите размер двумерного массива');  read(m);  read(n);  **until** (m > 0) **and** (m <= 10) **and** (n > 0) **and** (n <= 10);  sum := 0;  **for** m := 1 **to** m **do begin**  **for** n := 1 **to** n **do begin**  // randomize;  arr[m, n] := random(50);  write(arr[m, n] : 4);  **if** (m **mod** 2 = 0) **then**  sum := sum + arr[m, n];  **end**;  writeln();  **end**;  writeln('Сумма элементов равна = ', sum);  **end**. |
| **4.** | **Процедурное программирование** | Процеду́рное программи́рование — программирование на императивном языке, при котором последовательно выполняемые операторы можно собрать в подпрограммы, то есть более крупные целостные единицы кода, с помощью механизмов самого языка . | Kotlin, Go, Nim | * Любая процедура (функция) может быть вызвана неограниченное количество раз. Все как в жизни – ты один раз “написал” в голове маршрут к любимой пиццерии, а затем просто вызываешь эту “программу” из памяти. * Возможность оперативно решить задачу, в которой отсутствует сложная иерархия. Можно пойти дальше и сказать: если проект не подразумевает создания большого количества классов и объектов, то в ПП совсем нет минусов. | * Риск возникновения множества ошибок при работе над большим проектом. Приходится писать много процедур, и это не может не сказаться на чистоте и работоспособности кода. * Все данные процедуры доступны только внутри нее. Их нельзя вызвать из другого места программы и при необходимости придется писать аналогичный код. А это уже противоречит одному из основополагающих принципов программирования, который звучит как Don’t Repeat Yourself (Не повторяйся). * Сложность изучения для начинающих. Этот недостаток может кому-то показаться притянутым за уши, но простая статистика свидетельствует, что процедурное программирование для большинства новичков дается сложнее, чем объектно-ориентированное. | **Pascal**  **Program** zadproc\_1;  **var** x1, y1, x2, y2, dl1, dl2, dl3, m1, a, b, m : real;  **procedure** dlina(x1, y1, x2, y2 : real; **var** dl : real);  **begin**  dl := sqrt(sqr(x2 - x1) + sqr(y2 - y1));  **end**;    **procedure** mx(a, b : real; **var** m : real);    **begin**  **if**(a > b) **then**  m := a **else**  m := b;  **end**;  **begin**  writeln('Введите 1-ю коордитату');  readln(x1);  writeln('Введите 2-ю коордитату');  readln(x2);  writeln('Введите 3-ю коордитату');  readln(y1);  writeln('Введите 4-ю коордитату');  readln(y2);  writeln('Введите a');  readln(a);  writeln('Введите b');  readln(b);  dlina(x1, y1, 0, 0, dl1);  writeln(dl1);  dlina(x2, y2, 0, 0, dl2);  writeln(dl2);  dlina(a, b, 0, 0, dl3);  writeln(dl3);  mx(dl1, dl2, m1);  writeln('Сравнение первых 2 длин ', m1);  mx(dl3, m1, m);  writeln('Минимальное расстояние ', m);  **end**. |
| **5.** | **Модульное программирование** | **Мо́дульное** **программи́рование** — это организация программы как совокупности небольших независимых блоков, называемых модулями, структура и поведение которых подчиняются определённым правилам. | Assembler, Python, Perl | возможность разрабатывать модули с использованием различных языков программирования;  – модуль является естественной единицей локализации имен;  – локализация места ошибки: обычно исправление ошибки внутри одного модуля не влечет за собой исправление в других модулях (естественно, если это свойство будет выполняться только при хорошей разбивке программы на части, с малым числом связей между модулями); – возможность повторного использования разработанных модулей в других программах | – модули не являются совсем уж независимыми друг от друга: между ними существуют связи, то есть один модуль иногда может завязываться на переменные, константы и программный код из другого модуля;  – перед тем, как будет произведен первый запуск программы, необходимо собрать все нужные модули для программы в особую сборку. Этот процесс достаточно сложен, так как кроме собственно объединения всех модулей в одну программу, необходимо проконтролировать и установить все связи между этими частями;  – программа-компилятор, с помощью которой делается сборка программы, не видит сразу всех модулей и не может проконтролировать, установлены ли правильно связи между модулями. | **Pascal**  **Program** lab\_12;  **uses** lab\_12;  **var** x1, y1, x2, y2, dl1, dl2, dl3, m1, a, b, m, z : real;  **begin**  writeln('Введите 1-ю коордитату');  readln(x1);  writeln('Введите 2-ю коордитату');  readln(x2);  writeln('Введите 3-ю коордитату');  readln(y1);  writeln('Введите 4-ю коордитату');  readln(y2);  writeln('Введите a');  readln(a);  writeln('Введите b');  readln(b);  dlina(x1, y1, 0, 0, dl1);  writeln(dl1);  dlina(x2, y2, 0, 0, dl2);  writeln(dl2);  dlina(a, b, 0, 0, dl3);  writeln(dl3);  mx(dl1, dl2, m1);  writeln('Сравнение первых 2 длин ', m1);  mx(dl3, m1, m);  writeln('Минимальное расстояние ', m);  a:= 2.5;  b:= -7.3;  z := f(a, b) + f(sqr(a), sqr(b)) + f(sqr(a) - 1, b) + f(a - b, b) + f(sqr(a) + sqr(b), sqr(b) - 1);  write('Z = ', z);  **end**. |
| **6.** | [**Объектно-**](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)[**ориентированное**](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)  [**программирование**](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) | **Объе́ктно-ориенти́рованное программи́рование (ООП)** — [методология программирования](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), основанная на представлении программы в виде совокупности [объектов](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), каждый из которых является экземпляром определённого [класса](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), а классы образуют иерархию наследования. | C++, С#, Java | * Возможность легкой модификации (при грамотном анализе и проектировании) * Возможность отката при наличии версий * Более легкая расширяемость * «Более естественная» декомпозиция программного обеспечения, которая существенно облегчает его разработку. * Сокращение количества межмодульных вызовов и уменьшение объемов информации, передаваемой между модулями. * Увеличивается показатель повторного использования кода. | * Требуется другая квалификация * Резко увеличивается время на анализ и проектирование систем * Увеличение времени выполнения * Размер кода увеличивается :interrobang: * Неэффективно с точки зрения памяти (мертвый код - тот, который не используется) :interrobang: * Сложность распределения работ на начальном этапе * Себестоимость больше | Python  class MyClass:  x = int(input('Введите значение '))    def print\_x(self):  print(self.x)    x1 = MyClass()  x1.print\_x() |
| **7.** | [**Функциональное**](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)[**программирование**](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) | [**функциональное программирование**](https://intellect.icu/category/functional-programming#term-funktsionalnoe-programmirovanie)— раздел дискретной математики и парадигма программирования, в которой процессвычисления трактуется как вычисление значений функций в математическом понимании последних (в отличие от функций как подпрограмм в процедурном программировании). | Lisp, Erlang | -Повышение надежности кода;  -Удобство организации модульного тестирования;  -Возможности оптимизации при компиляции;  -Возможности параллелизма | Недостатки функционального программирования вытекают из тех же самых его особенностей. Отсутствие присваиваний и замена их на порождение новых данных приводят к необходимости постоянного выделения и автоматического освобождения памяти, поэтому в системе исполнения функциональной программы обязательным компонентом становится высокоэффективный сборщик мусора. Нестрогая модель вычислений приводит к непредсказуемому порядку вызова функций, что создает проблемы при вводе-выводе, где порядок выполнения операций важен. Кроме того, очевидно, функции ввода в своем естественном виде (например, getchar из стандартной библиотеки языка C) не являются чистыми, поскольку способны возвращать различные значения для одних и тех же аргументов, и для устранения этого требуются определенные ухищрения. | **Pascal**  **Program** z\_20;  **var** a, b, z : real;  **function** f(u, t: real) : real;  **begin**  **if** (u > 0) **and** (t > 0) **then** f := sqr(u) + sqr(t);  **if** (u <= 0) **and** (t <= 0) **then** f := u + sqr(t);  **if** (u > 0) **and** (t <= 0) **then** f := u - t;  **if** (u <= 0) **and** (t > 0) **then** f := u + t;  **end**;  **begin**  a:= 2.5;  b:= -7.3;  z := f(a, b) + f(sqr(a), sqr(b)) + f(sqr(a) - 1, b) + f(a - b, b) + f(sqr(a) + sqr(b), sqr(b) - 1);  write('Z = ', z);  **end**. |
| **8.** | [**Логическое**](https://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)[**программирование**](https://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) | **Логи́ческое программи́рование** — [парадигма программирования](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%B4%D0%B8%D0%B3%D0%BC%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), основанная на автоматическом доказательстве теорем, а также раздел [дискретной математики](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D0%BA%D1%80%D0%B5%D1%82%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0), изучающий принципы логического вывода информации на основе заданных фактов и правил вывода. Логическое программирование основано на теории и аппарате [математической логики](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%BB%D0%BE%D0%B3%D0%B8%D0%BA%D0%B0) с использованием математических принципов резолюций. | Mercury, Visual Prolog, Oz | 1. Операции, совершаемые в логическом программировании всегда понятны; 2. Результат практически всегда не зависит от выбранного пути реализации; 3. Может быть использован в качестве невычислительного языка используя только выражения и факты. | 1. Если брать за пример логического языка программирования Prolog, то на лицо невозможность создания комплексных задач. То есть в реальности логический язык может идти дополнением к процедурному, но самостоятельно используется крайне редко; 2. Из-за недостатка в инвестициях и простом внимании, логические языки слабо развиваются; 3. Если предстоит иметь дело с вычислительными операциями, то логические языки программирования - не лучший выбор. |  |

Для нашего проекта мы будем использовать [объектно-ориентированное](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) [программирование](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) т.к. эта система программирования очень проста и благодаря ей можно создавать сложные проекты.

# Задание 3. Изучение правил формирования листинга программ.

# //Разработал: Борис И.Д.

# //Задача: Найти среднее арифметическое элементов массива.

unit Unit1;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls, Grids, ShellAPI;

type

TForm1 = class(TForm)

Label1: TLabel;

Label2: TLabel;

Button1: TButton;

StringGrid1: TStringGrid;

btn1: TButton;

procedure Button1Click(Sender: TObject);

procedure btn1Click(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Form1: TForm1;

implementation

{$R \*.dfm}

procedure TForm1.Button1Click(Sender: TObject); // объявление процедуры

const SIZE = 4; // присваиваем переменной SIZE постоянное значение 4

var

i : Integer;

k : Real;

a : array [1..SIZE] of real; // объявление массива

begin

k := 0; // присваиваем переменной k значение 0

for i := 1 to SIZE do // объявление массива

a[i] := StrToFloat(StringGrid1.Cells[i-1, 0]); // присваиваем StringGrid значения переменной a[i]

for i := 1 to SIZE do

begin

k := (k + a[i]) / 4; // формула нахождения среднего арифметического

end;

label2.Caption := 'Среднее арифметическое ' + FloatToStr(k); // вывод результата в компонент Label

end;.

// Разработал: Борис И.Д.

//Создайте класс чтобы он выводил значение атрибута, введенного пользователем.

class MyClass://Создание класса

x = int(input('Введите значение '))//Ввод значения

def print\_x(self)://Создание метода

print(self.x)//Вывод значения

x1 = MyClass()//Создание экземпляра класса

x1.print\_x()//Вывод метода для экземпляра класса