***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.**

Изучить критерии выбора языка программирования.

1. Язык программирования Язык, предназначенный для записи компьютерных программ. Язык определяет набор правил, которые определяют внешний вид и действия программы. Он предназначен для написания программ, которые представляют собой набор правил, позволяющих компьютеру выполнить тот или иной вычислительный процесс. Сам язык предназначен для управления ЭВМ. Алфавит языка программирования – это символы или их комбинации, которые используются при программировании на этом языке. Семантика формализация языков программирования посредством построения их формальных математических моделей. Стандарт языка программирования – это предмет документации, который определяет язык программирования, чтобы пользователи и разработчики языка могли согласовывать, что означают программы на данном языке. У языков программирования присутствует классификация, например: процедурные, логические, структурные или объектно-ориентированные языки. На начальном этапе создания программы так или иначе становится вопрос выбора языка программирования. Кто-то выбирает язык только из личных предпочтений, кто-то только потому, что знает только этот язык.\

Критерии выбора:   
1. *Скорость работы конечного продукта*.   
Требовательным к скорости выполнения могут быть программы с большим объемом математических вычислений, например языки: ассемблер, С/С++, фортран. После сборки программа не требует лишних действий и содержит в себе машинные команды, которые выполняются без лишних задержек. Схема их работы не замысловата: 1) программа исполняется сразу; 2) программа кроме своего кода содержит вызовы библиотек с машинным кодом, поэтому, кроме исполнения собственно своих команд, программа вызывает функции из библиотек; 3) программа может работать через прослойку драйверов, которые написаны на языках низкого уровня.   
2. *Объем занимаемой оперативной памяти*.   
Данное требование появляется, когда программа разрабатывается для встраиваемых систем, мобильных платформ и так далее. В данных случаях, чем

меньше памяти расходует программа – тем лучше. К таким языкам относятся ассемблер, С/С++, Objective-C и другие. Чем меньше функциональных блоков в схеме исполнения, тем меньше занимается и памяти компьютера.   
3.*Скорость разработки программы*.   
Важна тогда, когда требуется срочность. Тогда выбор падает на высокоуровневые языки с максимально человеколюбивым синтаксисом. Например, Java, Flash и подобные. На данных языках время разработки может существенно сокращаться из-за обилия сторонних библиотек, довольно понятного синтаксиса, и подобных вещей.   
4.*Кроссплатформенность*.   
Кроссплатформенность – возможность работы программы на различных платформах, в различных ОС с минимальными изменениями. Облегчают задачу кроссплатформенные библиотеки, например, Qt, которые позволяют добиться принципа «один код на все платформы», однако на каждую платформу нужно программу собирать отдельно.   
5.*Скорость скорость тестирования*.  
Если в проект по ходу разработки вносится много изменений, тогда стоит выбрать высокоуровневые языки, где любой функциональный блок можно быстро переписать.

1. Обосновать выбор языка программирования для решения поставленной задачи в соответствии с индивидуальным заданием.

Для написания программы была выбрана среда программирования CSS. Данная среда выгодно отличается эффективностью и надежностью. В качестве такой платформы была выбрана среда . PyCharm. Среда разработки PyCharm предоставляет необходимый инструментарий для эффективного и быстрого создания приложений с графическим интерфейсом.

Обосновать выбор языка программирования для решения поставленной задачи в соответствии с  индивидуальным заданием.

Для написания программы была выбрана среда программирования PyCharm основанная на языке программирования Python. Данная среда выгодно отличается эффективностью и надежностью.

Для решения поставленной задачи необходимо использовать функциональную, эффективную и удобную платформу для разработки, позволяющую применять принципы объектно-ориентированного программирования. В качестве такой платформы была выбрана среда SaaS.

Среда разработки PyCharm, вместе с SaaS, предоставляет необходимый инструментарий для эффективного и быстрого создания приложений с графическим интерфейсом.

**Задание 2. Изучение парадигм программирования**

* 1. Изучить различные парадигмы программирования.

Парадигма программирования — это совокупность идей и понятий, определяющих стиль написания компьютерных программ (подход к программированию). Это способ концептуализации, определяющий организацию вычислений и структурирование работы, выполняемой компьютером.

Парадигма программирования не определяется однозначно языком программирования; практически все современные языки программирования в той или иной мере допускают использование различных парадигм.

Также существующие парадигмы зачастую пересекаются друг с другом в деталях, поэтому можно встретить ситуации, когда разные авторы употребляют названия из разных парадигм, говоря при этом, по сути, об одном и том же явлении.

*2.* Провести сравнительную характеристику различных парадигм программирования. Результаты  оформить в виде таблицы следующего вида:

1. Дайте характеристику следующим **парадигмам программирования**:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **№** | **Название парадигмы** | **Сущность**  **парадигмы,**  **основные**  **идеи,**  **принципы,**  **объекты.** | **Языки**  **поддерживающие данную  парадигму.**  **(2-3 языка)** | **Достоинства** | **Недостатки** | **Примеры** **программ** |
| **1.** | **Императивное**  **программирование** | 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)).  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. | [Fortran](https://ru.wikipedia.org/wiki/Fortran), [Algol](https://ru.wikipedia.org/wiki/Algol), [COBOL](https://ru.wikipedia.org/wiki/COBOL) и [Basic](https://ru.wikipedia.org/wiki/Basic) | Использование именованных [переменных](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0%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%D0%B5));  использование оператора [присваивания](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D1%81%D0%B2%D0%B0%D0%B8%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5);  использование [подпрограмм](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B4%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0); | Императивный язык должен описывать не столько саму задачу сколько её решение. Некоторыми авторами считается, что данное определение скорее относится к «процедурной» парадигме, которая, помимо императивного, включает в себя функциональное программирование. |  |
| **2.** | **Декларативное**  **программирование** | [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. | yacc.; SQL; CSS | К подвидам декларативного программирования также зачастую относят [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) и [логическое 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). Несмотря на то, что программы на таких языках нередко содержат алгоритмические составляющие, архитектура в императивном понимании (как нечто отдельное от кодирования) в них также отсутствует: схема программы является непосредственно частью исполняемого кода. | декларативные программы не используют понятия состояния, в частности, не содержат [переменных](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0%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%D0%B5)) и операторов [присваивания](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D1%81%D0%B2%D0%B0%D0%B8%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), обеспечивается [ссылочная прозрачность](https://ru.wikipedia.org/wiki/%D0%A1%D1%81%D1%8B%D0%BB%D0%BE%D1%87%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B7%D1%80%D0%B0%D1%87%D0%BD%D0%BE%D1%81%D1%82%D1%8C). |  |
| **3.** | **Структурное**  **программирование** | В соответствии с парадигмой, любая программа, которая строится без использования оператора goto, состоит из трёх базовых управляющих конструкций: последовательность, [ветвление](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D1%82%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%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)), [цикл](https://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D0%BA%D0%BB_(%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)); кроме того, используются [подпрограммы](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B4%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0). При этом разработка программы ведётся пошагово, методом «сверху вниз». | Basic; Pascal; REXX | Методология структурной разработки программного обеспечения была признана «самой сильной формализацией 70-х годов». | Программы становились слишком сложными, чтобы их можно было нормально сопровождать. Поэтому потребовалась систематизация процесса разработки и структуры программ. |  |
| **4.** | **Процедурное**  **программирование** | Программирование на [императивном языке](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), при котором последовательно выполняемые [операторы](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)) можно собрать в [подпрограммы](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B4%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0), то есть более крупные целостные единицы [кода](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4), с помощью механизмов самого языка | 1С; Basic; C | Используя процедурный язык, программист определяет языковые конструкции для выполнения последовательности алгоритмических шагов. | Выполнение программы сводится к последовательному выполнению операторов с целью преобразования исходного состояния памяти, то есть значений исходных данных, в заключительное, то есть в результаты |  |
| **5.** | **Модульное**  **программирование** | Это организация программы как совокупности небольших независимых блоков, называемых модулями, структура и поведение которых подчиняются определённым правилам | Assembler, RPG, Aga | Использование модульного программирования позволяет упростить тестирование программы и обнаружение ошибок. Аппаратно-зависимые подзадачи могут быть строго отделены от других подзадач, что улучшает мобильность создаваемых программ.  Модульное программирование может быть осуществлено, даже когда синтаксис языка программирования не поддерживает явное задание имён модулям. | Модульный подход иногда требует большего времени ЦП. Эта проблема возникает прежде всего в тех случаях, когда программа отличается наличием большого числа подпрограмм, написанных на языках высокого уровня. Для обеспечения модульности может также потребоваться больше времени ЦП, если части программы с командами ввода-вывода совершенно отделены от ее вычислительных частей; входная запись может быть несколько раз передана подпрограммам, прежде чем начнется ее фактическая обработка. |  |
| **6.** | **Объектно**  **ориентированное**  **программирование** | [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 наследовании | [C++](https://ru.wikipedia.org/wiki/C%2B%2B), [Java](https://ru.wikipedia.org/wiki/Java) , [Smalltalk](https://ru.wikipedia.org/wiki/Smalltalk) | Управляемость для иерархических систем предполагает минимизацию избыточности данных (аналогичную [нормализации](https://ru.wikipedia.org/wiki/%D0%9D%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%84%D0%BE%D1%80%D0%BC%D0%B0)) и их целостность, поэтому созданное удобно управляемым — будет и удобно пониматься. Таким образом, через тактическую задачу управляемости решается стратегическая задача — транслировать понимание задачи программистом в наиболее удобную для дальнейшего использования форму. | То есть фактически речь идёт о прогрессирующей организации информации согласно первичным семантическим критериям: «важное/неважное», «ключевое/подробности», «родительское/дочернее», «единое/множественное». |  |
| **7.** | **Функциональное**  **программирование** | [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 процесс [вычисления](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5) трактуется как вычисление значений [функций](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) в математическом понимании последних (в отличие от [функций](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%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%D0%B5)) как подпрограмм в [процедурном программировании](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D0%B4%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)). | [Лисп](https://ru.wikipedia.org/wiki/%D0%9B%D0%B8%D1%81%D0%BF)  [APL](https://ru.wikipedia.org/wiki/APL_(%D1%8F%D0%B7%D1%8B%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%D1%8F)) —  [ML](https://ru.wikipedia.org/wiki/ML)  [F#](https://ru.wikipedia.org/wiki/F_Sharp)  [Miranda](https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D1%80%D0%B0%D0%BD%D0%B4%D0%B0_(%D1%8F%D0%B7%D1%8B%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%D1%8F)) [Nemerle](https://ru.wikipedia.org/wiki/Nemerle) | Функциональное программирование предполагает обходиться вычислением результатов функций от исходных данных и результатов других функций, и не предполагает явного хранения состояния программы. Соответственно, не предполагает оно и изменяемость этого состояния (в отличие от [императивного](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), где одной из базовых концепций является [переменная](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0%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%D0%B5)), хранящая своё значение и позволяющая менять его по мере выполнения [алгоритма](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC)). | На практике отличие математической функции от понятия «функции» в императивном программировании заключается в том, что императивные функции могут опираться не только на аргументы, но и на состояние внешних по отношению к функции переменных, а также иметь [побочные эффекты](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B1%D0%BE%D1%87%D0%BD%D1%8B%D0%B9_%D1%8D%D1%84%D1%84%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)) и менять состояние внешних переменных. |  |
| **8.** | **Логическое**  **программирование** | [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. | [Planner](https://ru.wikipedia.org/wiki/Planner), [QA-4](https://ru.wikipedia.org/w/index.php?title=QA-4&action=edit&redlink=1), [QLISP](https://ru.wikipedia.org/w/index.php?title=QLISP_(%D1%8F%D0%B7%D1%8B%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%D1%8F)&action=edit&redlink=1). | Первым языком логического программирования был язык [Planner](https://ru.wikipedia.org/wiki/Planner), в котором была заложена возможность автоматического вывода результата из данных и заданных правил перебора вариантов (совокупность которых называлась планом). Planner использовался для того, чтобы понизить требования к вычислительным ресурсам (с помощью бэктрекинга — [поиска с возвратом](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B8%D1%81%D0%BA_%D1%81_%D0%B2%D0%BE%D0%B7%D0%B2%D1%80%D0%B0%D1%82%D0%BE%D0%BC)) и обеспечить возможность вывода фактов, без активного использования [стека](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B5%D0%BA). Затем был разработан язык [Prolog](https://ru.wikipedia.org/wiki/Prolog), который не требовал плана перебора вариантов и был, в этом смысле, упрощением языка [Planner](https://ru.wikipedia.org/wiki/Planner). | Из-за недостатка в инвестициях и простом внимании, логические языки слабо развиваются;  Если предстоит иметь дело с вычислительными операциями, то логические языки программирования - не лучший выбор. |  |

Примеры программ:

1) Императивный подход  
Кнопка, которая меняет цвет при нажатии  
const container = document.getElementById(‘container’);

const btn = document.createElement(‘button’);

btn.className = ‘btn red’;

btn.onclick = function(event) {

if (this.classList.contains(‘red’)) {

this.classList.remove(‘red’);

this.classList.add(‘blue’);

} else {

this.classList.remove(‘blue’);

this.classList.add(‘red’);

}

};

container.appendChild(btn);

2) Декларированный подход  
Кнопка, которая меняет цвет при нажатии  
class Button extends React.Component{

this.state = { color: 'red' }

handleChange = () => {

const color = this.state.color === 'red' ? 'blue' : 'red';

this.setState({ color });

}

render() {

return (<div>

<button

className=`btn ${this.state.color}`

onClick={this.handleChange}>

</button>

</div>);

}

}

3)Структурный подход  
Алгоритм Евклида  
Program Evclid;

Var a, b, u, v, w: Integer;

Begin

Read(a,b);

u := a; v := b;

While u <> v do begin

w := u - v;

If w > 0 then u := w else v := -w;

end;

Write(u)

end.

4) Процедурный подход  
Просуммировать элементы  
int sum = 0;

for (int v : vec)

sum += v;

std::cout << sum << std::endl;

6) Функциональный подход  
Просуммировать элементы  
int sum = std::accumulate(vec.begin(), vec.end(), 0);

std::cout << sum << std::endl;

7) ООП

Просуммировать элементы  
class Accumulator

{

public:

void feed(int value) { fSum += value; }

int sum() const { return fSum; }

private:

int fSum = 0;

}

Accumulator acc;

for (int v : vec)

acc.feed(vec);

std::cout << acc.sum() << std::endl;

5) Модульный подход  
Вычислить площадь круга S и длину окружности L по заданному радиусу R.

Программа

program KRUG;

const P=3.14159

var

R,S,L:Real;

begin

Read(R);{ввод значения радиуса}

L:=2\*P\*R;

S:=P\*SQR(R);

Writeln(Длина окружности = ',L,'см');

Write('Площадь круга = ',S,'кв.см');

end.

3. Сделать выводы о том, какие из парадигм программирования используются наиболее часто.

## Процедурное программирование

Этот подход — разновидность императивной парадигмы программирования. Процедурами здесь называют команды, которые применяются в определённом порядке и последовательно меняют состояние памяти. После применения всех команд программа выдаёт результат.

**Метапрограммирование**

Здесь программа, которую вы создаёте, сама генерирует код новой программы или модифицирует свой. С помощью этого подхода часть задач разработчика можно автоматизировать. А ещё он даёт возможность людям, не владеющим языками программирования, создавать программы с помощью графических интерфейсов или словесных команд на естественном языке — программа преобразует их в обычный код.

## Обобщённое программирование

В этой парадигме программист создаёт обобщённые представления для классов и функций. Изначально у них отсутствуют требования типа данных для входных параметров, поэтому шаблоны можно сделать более универсальными.

Преимущество этой парадигмы в том, что можно создавать алгоритмы, которые будут работать с разными типами, и для этого не придётся добавлять реализации для каждого типа отдельно. Такой подход можно совместить как с ООП, так и с другими современными парадигмами программирования.

## Логическое программирование

Логическое программирование — это подвид декларативного. Основан на выводе информации из заданных фактов и логических правил, которые к ним можно применить. При выполнении программ используются правила формальной логики.

4. Для своего индивидуального задания привести обоснование того какие парадигмы  программирования вы будете использовать.

Для индивидуального задания я буду использовать Императивную или процедурную парадигму. Она основана на принципе выполнения инструкций шаг за шагом, как создание рецепта приготовления. Он основан на принципе машины фон Неймана.

Набор инструкций управления потоком выполнения управляет порядком, в котором выполняются инструкции, описывающие шаги. С , то Паскаль, то Fortran и COBOL, являются примерами языка программирования, который реализует императив парадигма .