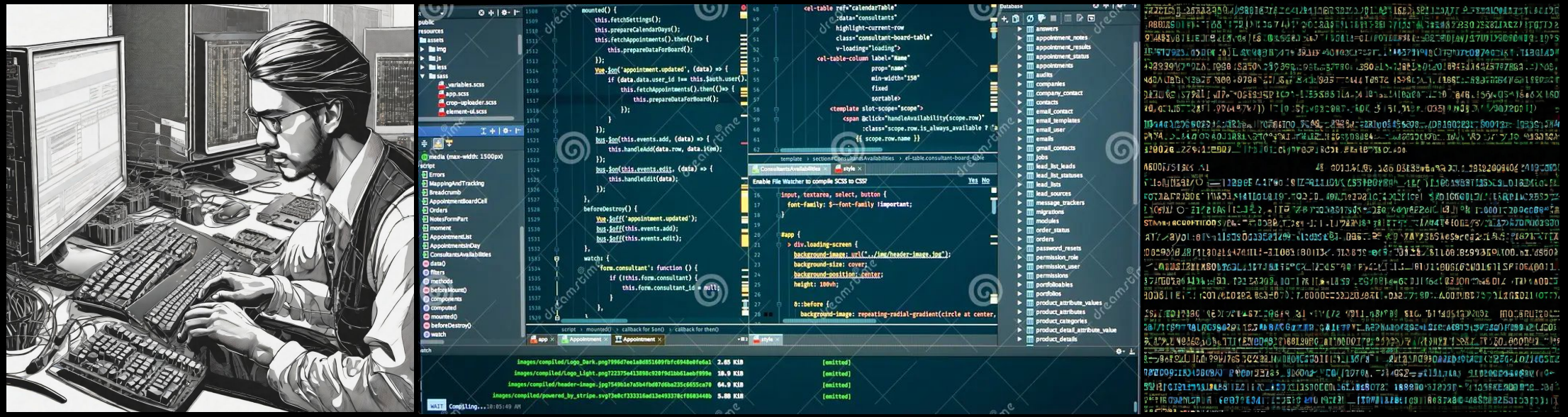
Парадигмы программирования

- Визуальная
- Функциональная
- Процедурная
- Объектно-Ориентированная

Определение термина “парадигма программирования”

Парадигмы программирования — это совокупность методов, концепций, принципов, техник и инструментов, которые определяют способ организации программы на языке программирования и ход её выполнения.

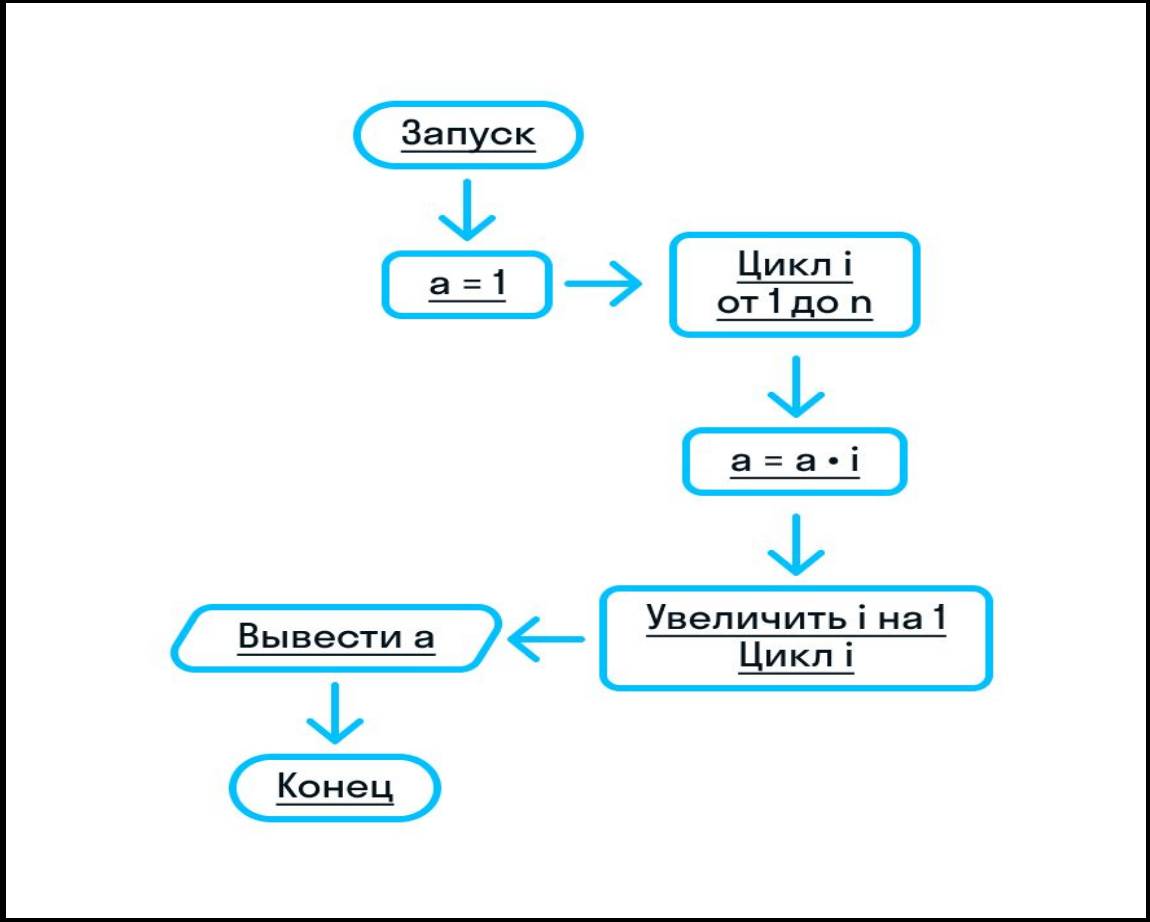
Ещё парадигмами называют **запрет на определённые действия** внутри кода программы. Его придумал Роберт Мартин — международный консультант в области разработки, известный среди разработчиков как «дядя Боб». С его точки зрения, парадигмы — это ограничения на определённые языковые конструкции, которые вынуждают использовать определённый стиль. Например, процедурное программирование накладывает запрет на прыжки по коду программы, а функциональное — на прямое изменение памяти компьютера.



Визуальная парадигма.

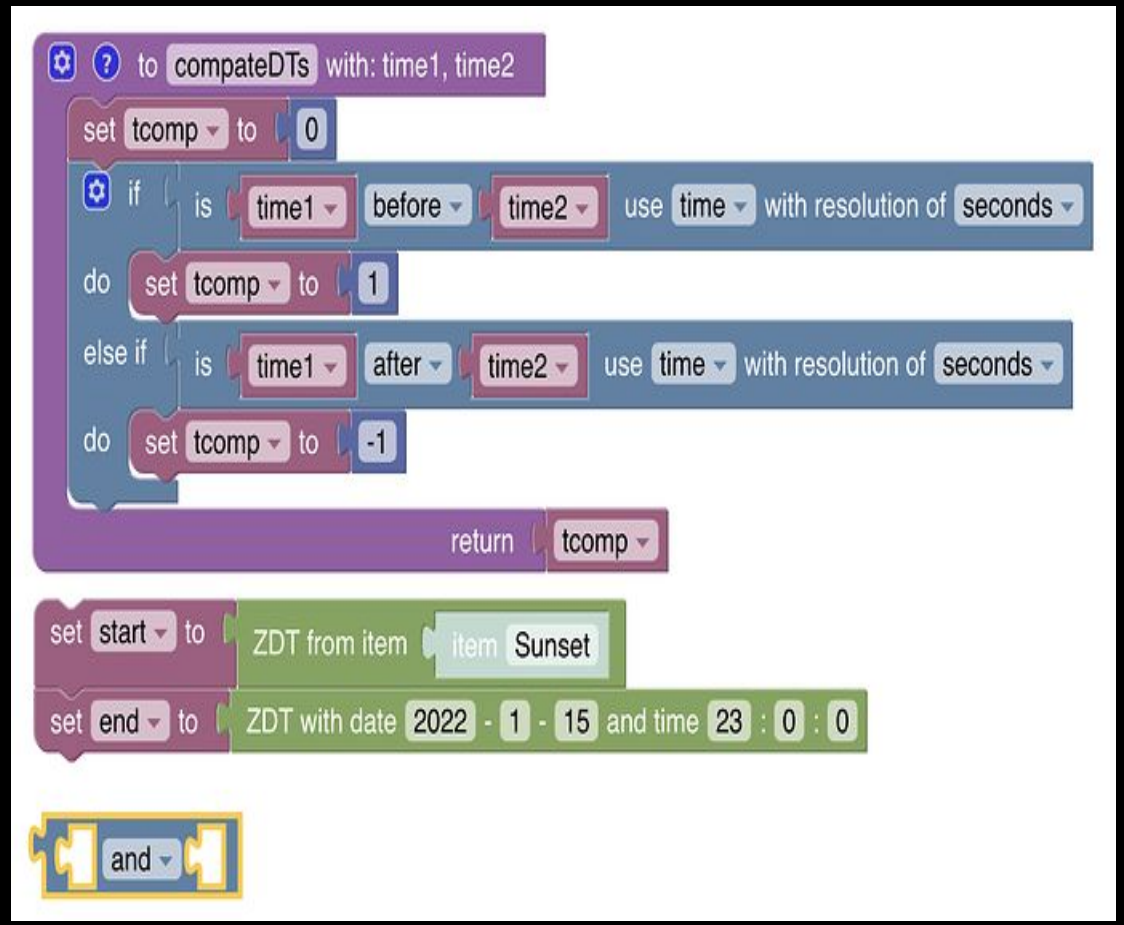
Визуальное программирование — способ создания программы путем манипулирования графическими объектами вместо написания ее текста.

Что такое визуальное программирование, можно догадаться даже из названия. Это технология, которая позволяет создавать программы с помощью графических элементов вместо кода. Так программист может просто описать концепцию приложения и создать алгоритм программы для ЭВМ — электронно-вычислительной техники. При этом он не будет особо вдаваться в техническую сторону процесса, чего не избежать в текстовых языках. Именно поэтому визуальное программирование полезно, чтобы обучаться или описывать данные в виде схем. Например, блок-схемы, с которыми вы могли встречаться на уроках информатики в школе, можно тоже считать видом визуального программирования.



Основные черты визуальной парадигмы программирования.

- **Визуальное представление кода:** Программы создаются с использованием графических элементов, таких как блоки, стрелки и формы, представляющие различные операции и структуры данных.
- **Блочные языки программирования:** Визуальные языки, такие как Scratch или Blockly, предоставляют блоки, которые представляют различные действия и условия. Программист собирает блоки в логические структуры для создания программы.
- **Интуитивный интерфейс:** Визуальные языки обычно ориентированы на упрощение процесса программирования, особенно для начинающих.



Основные преимущества визуальной парадигмы программирования.

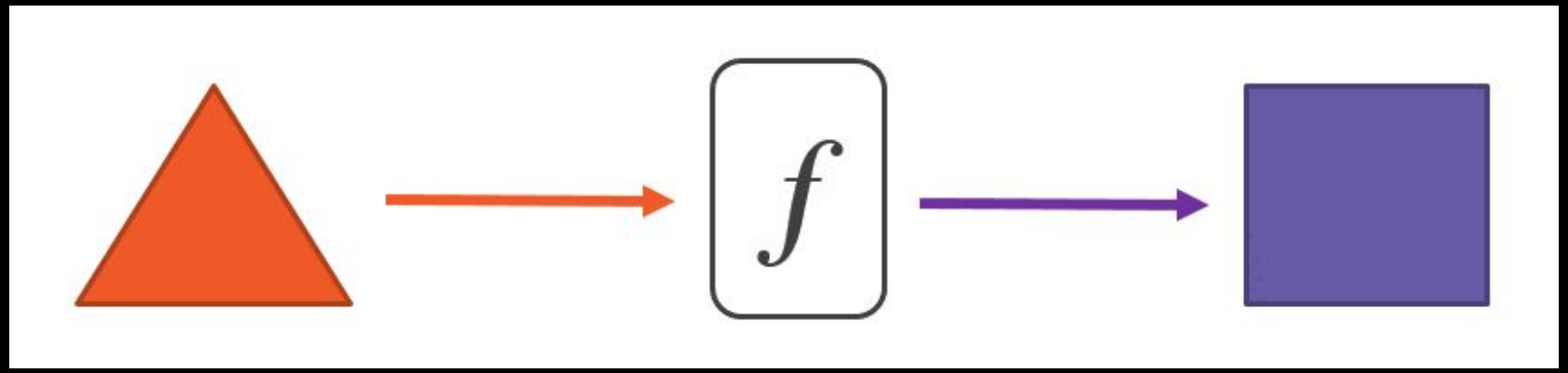
- **Легкость обучения:** Визуальная парадигма делает программирование более доступным для новичков, поскольку не требуется запоминание синтаксиса.
- **Визуализация алгоритмов:** Позволяет лучше понимать логику программы благодаря визуализации структур и потока управления.
- **Быстрая разработка прототипов:** Блочные языки упрощают создание прототипов и экспериментирование с идеями.

Основные ограничения визуальной парадигмы программирования.

- **Ограниченная выразительность:** Визуальные языки могут оказаться неэффективными при решении сложных или вычислительно интенсивных задач.
- **Ограниченные возможности:** Возможности визуальных языков могут быть ограничены по сравнению с текстовыми языками, что делает их менее подходящими для крупных и сложных проектов.
- **Неудобство при масштабировании:** При увеличении размера программы визуальное представление может стать сложным для поддержки и понимания, особенно без соответствующих инструментов.
- **Ограниченный контроль над деталями:** Некоторые программисты могут испытывать неудовлетворение из-за ограниченного контроля над деталями программы, который обеспечивается визуальной парадигмой.

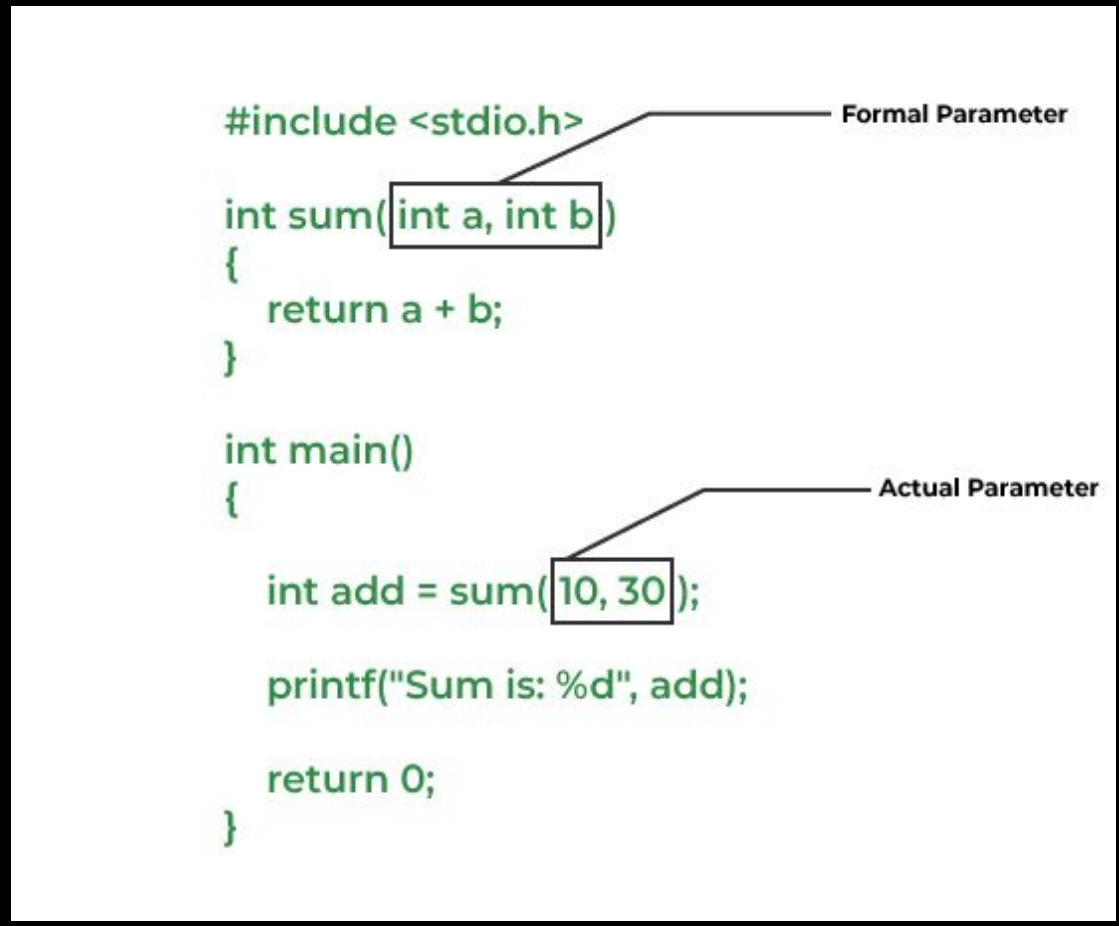
Функциональная парадигма

Функциональная парадигма программирования — это стиль написания программ, ориентированный на использование функций и операций над функциями. Она основана на математическом понятии функций и включает в себя ряд принципов, которые отличают ее от других парадигм, таких как процедурное программирование или объектно-ориентированное программирование.



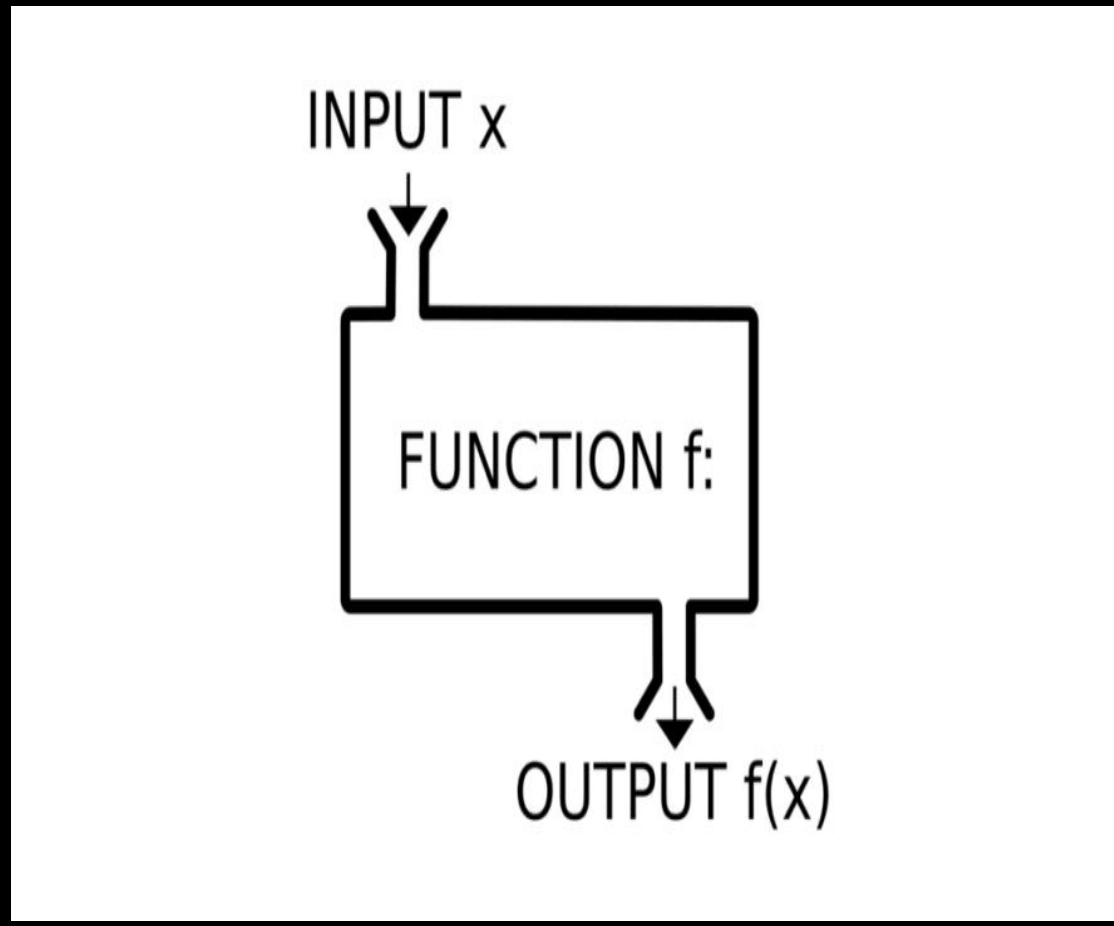
Основные черты функциональной парадигмы программирования.

- **Ориентация на функции:** Программирование в функциональной парадигме строится вокруг создания и композиции функций. Функции рассматриваются как основные строительные блоки программы.
- **Отсутствие изменяемого состояния:** Переменные считаются неизменяемыми, и изменение состояния осуществляется созданием новых значений вместо изменения существующих.
- **Рекурсия:** Рекурсия часто используется для решения задач, и циклы заменяются рекурсивными вызовами функций.



Основные преимущества функциональной парадигмы программирования.

- **Простота тестирования и отладки:** Из-за отсутствия изменяемого состояния и побочных эффектов функциональные программы обычно легче тестируются и отлаживаются.
- **Параллелизм:** Функциональные программы легче поддаются параллелизации, что может улучшить производительность в многозадачных средах.
- **Избегание гонок данных:** Отсутствие изменяемого состояния и использование неизменяемых структур данных снижает возможность возникновения гонок данных.
- **Математическая основа:** Функциональная парадигма имеет четкую математическую основу, что делает ее привлекательной для формального доказательства свойств программ.

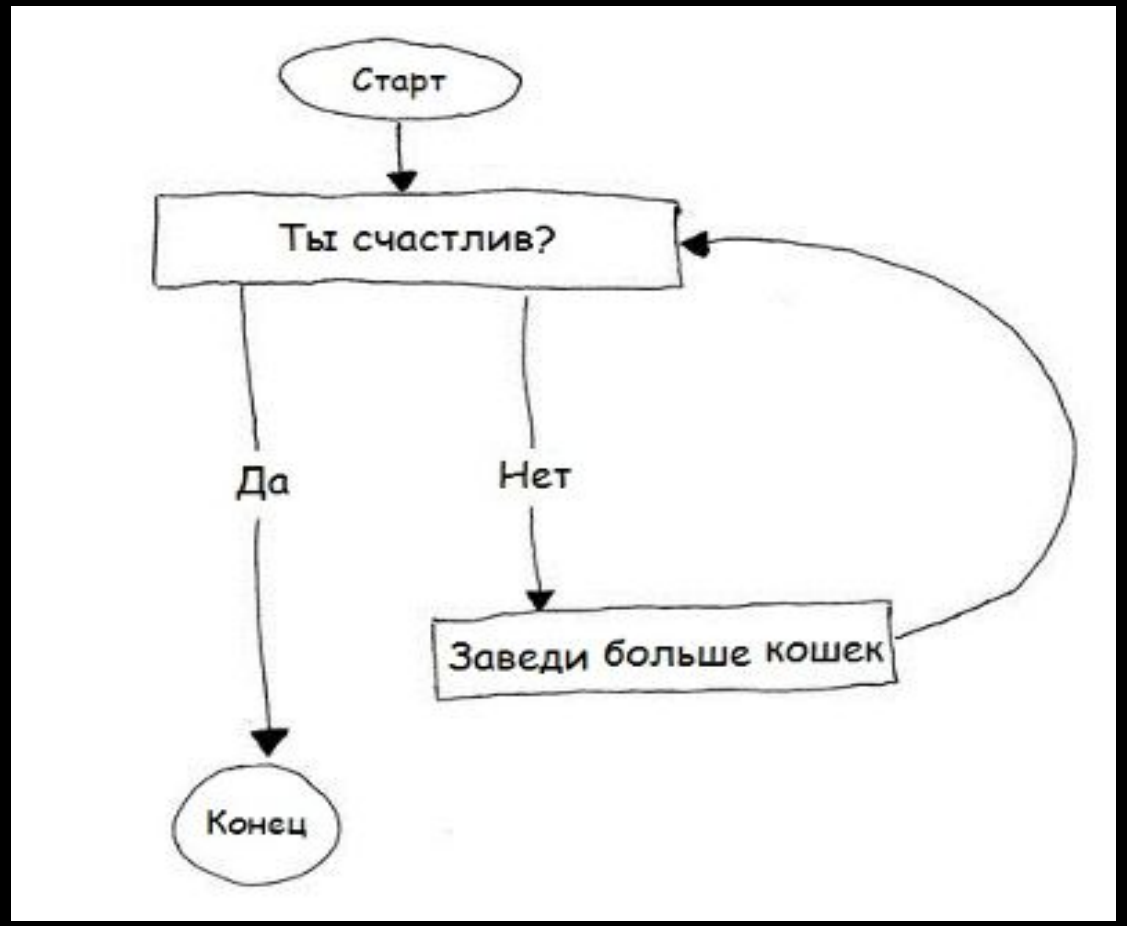


Основные ограничения функциональной парадигмы программирования.

- **Не всегда интуитивна:** Для программистов, привыкших к императивным или объектно-ориентированным языкам, функциональная парадигма может потребовать времени для освоения и понимания.
- **Неэффективность в некоторых случаях:** В некоторых вычислительно интенсивных задачах функциональные программы могут быть менее эффективными по сравнению с императивными аналогами.
- **Неудобство при работе с изменяемыми данными:** Некоторые задачи, связанные с изменяемыми данными, могут быть неудобными для решения в функциональной парадигме.
- **Ограниченная поддержка в некоторых языках:** Не все языки программирования хорошо поддерживают функциональную парадигму, что может создавать ограничения при выборе языка для конкретного проекта.

Процедурная парадигма.

Процедурная парадигма программирования представляет собой стиль написания программ, в котором основной акцент делается на создании процедур (или функций), которые выполняют конкретные задачи. Процедуры в этом контексте представляют собой набор инструкций, группированных в логические блоки, чтобы выполнять определенные действия.



Основные черты процедурной парадигмы программирования.

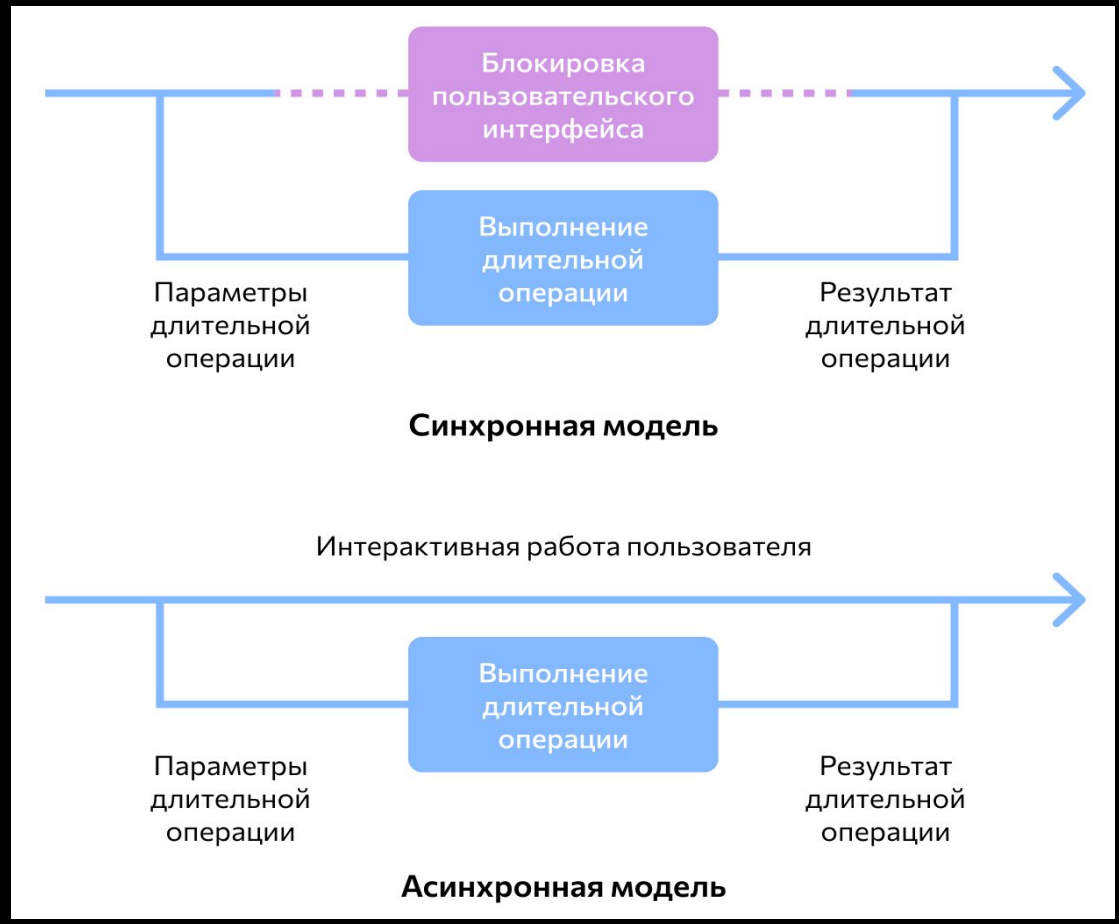
- **Процедуры (функции):** Программа строится вокруг процедур, которые являются наборами инструкций, выполняющими конкретные действия. Процедуры также могут быть названы функциями, подпрограммами или методами в зависимости от языка программирования.
- **Использование переменных:** Программы могут использовать переменные для хранения и управления данными. Эти переменные могут быть изменяемыми, и состояние программы изменяется в процессе выполнения.
- **Структурирование кода:** Код структурируется в виде процедур, что обеспечивает модульность и улучшает читаемость.

Основные преимущества процедурной парадигмы программирования.

- **Простота понимания:** Процедурная парадигма обычно предоставляет простой и интуитивно понятный способ написания программ, что делает ее доступной для большого числа программистов.
- **Легкость отладки:** Использование процедур позволяет локализовать ошибки и упрощает отладку, поскольку каждая процедура выполняет конкретную функцию.
- **Эффективное использование ресурсов:** Процедурные программы могут быть эффективными с точки зрения использования ресурсов, так как управление памятью и другими ресурсами может быть более предсказуемым.
- **Широкое применение:** Множество языков программирования, таких как C и Pascal, предоставляют возможности для процедурного программирования, что делает эту парадигму широко используемой.

Основные ограничения процедурной парадигмы программирования.

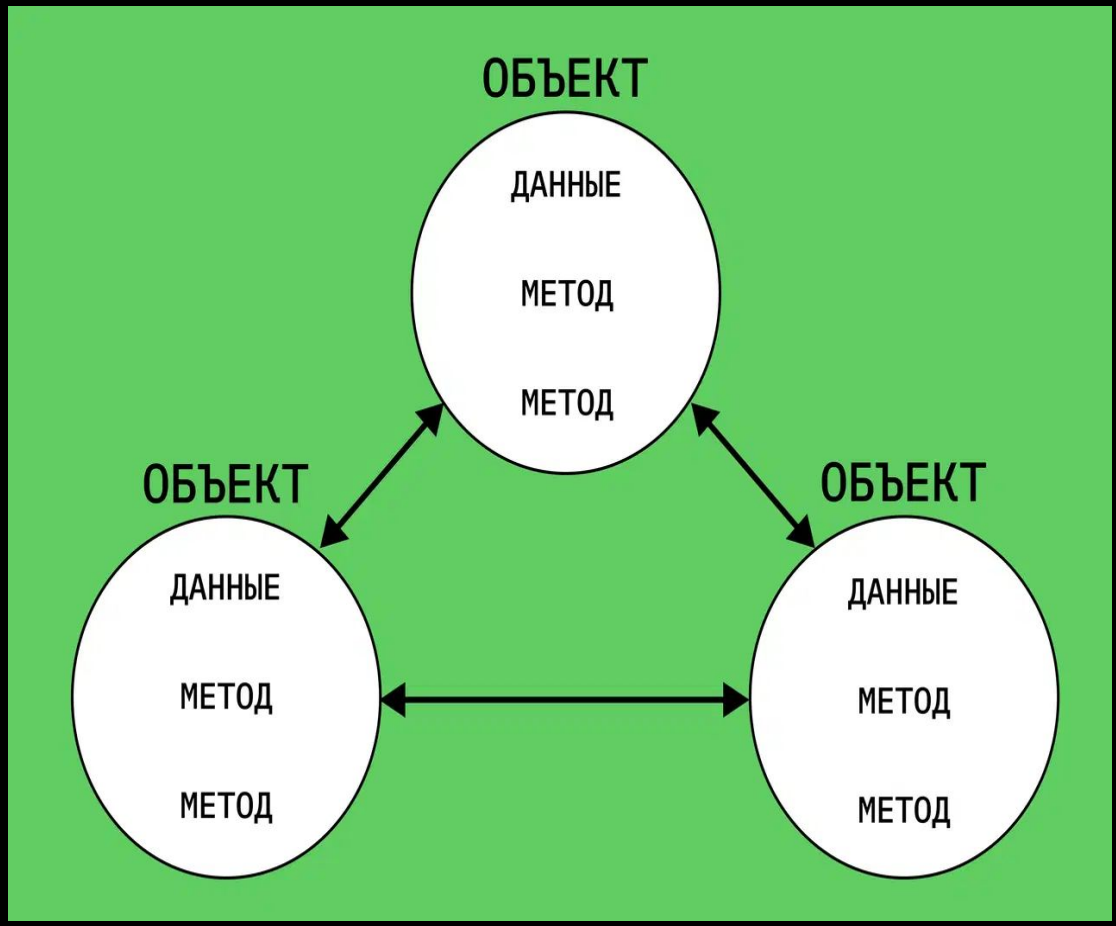
1. **Сложность поддержки больших проектов:** При разработке больших программ процедурный код может стать сложным для поддержки и расширения, так как все процедуры могут быть взаимосвязаны.
2. **Ограниченная переиспользуемость кода:** Процедурный код не всегда обеспечивает легкость переиспользования, поскольку процедуры обычно тесно связаны с конкретным контекстом.
3. **Сложность асинхронного программирования:** В процедурной парадигме сложно эффективно работать с асинхронными событиями, что может быть проблемой при разработке интерактивных приложений или при программировании в средах событийно-ориентированных систем.
4. **Ограниченная гибкость:** Процедурная парадигма может оказаться менее гибкой по сравнению с другими парадигмами, такими как объектно-ориентированная или функциональная, в определенных сценариях программирования.



Объектно-ориентированная парадигма.

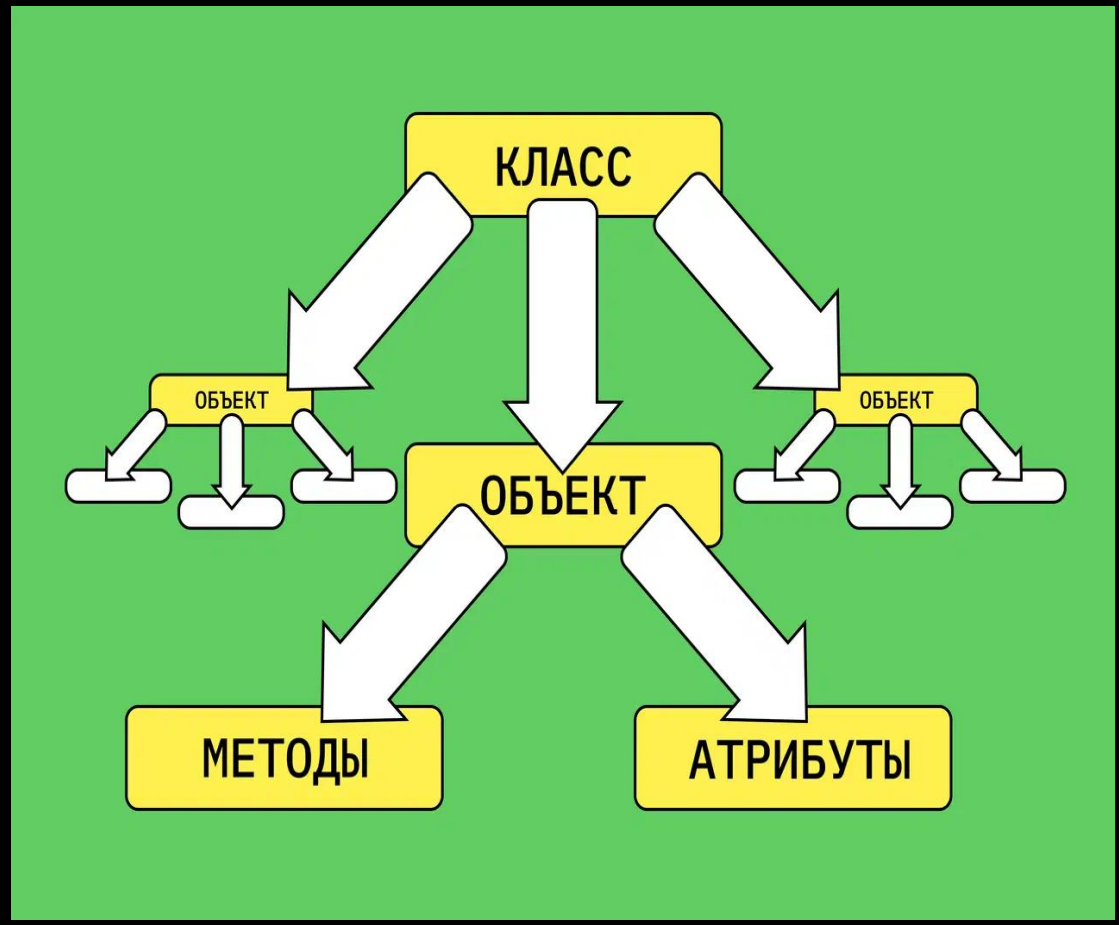
Суть понятия объектно-ориентированного программирования в том, что все программы, написанные с применением этой парадигмы, состоят из объектов. Каждый объект — это определённая сущность со своими данными и набором доступных действий.

Например, нужно написать для интернет-магазина каталог товаров. Руководствуясь принципами ООП,
в первую очередь нужно создать объекты: карточки товаров. Потом заполнить эти карточки данными: названием товара, свойствами, ценой. И потом прописать доступные действия для объектов: обновление, изменение, взаимодействие.



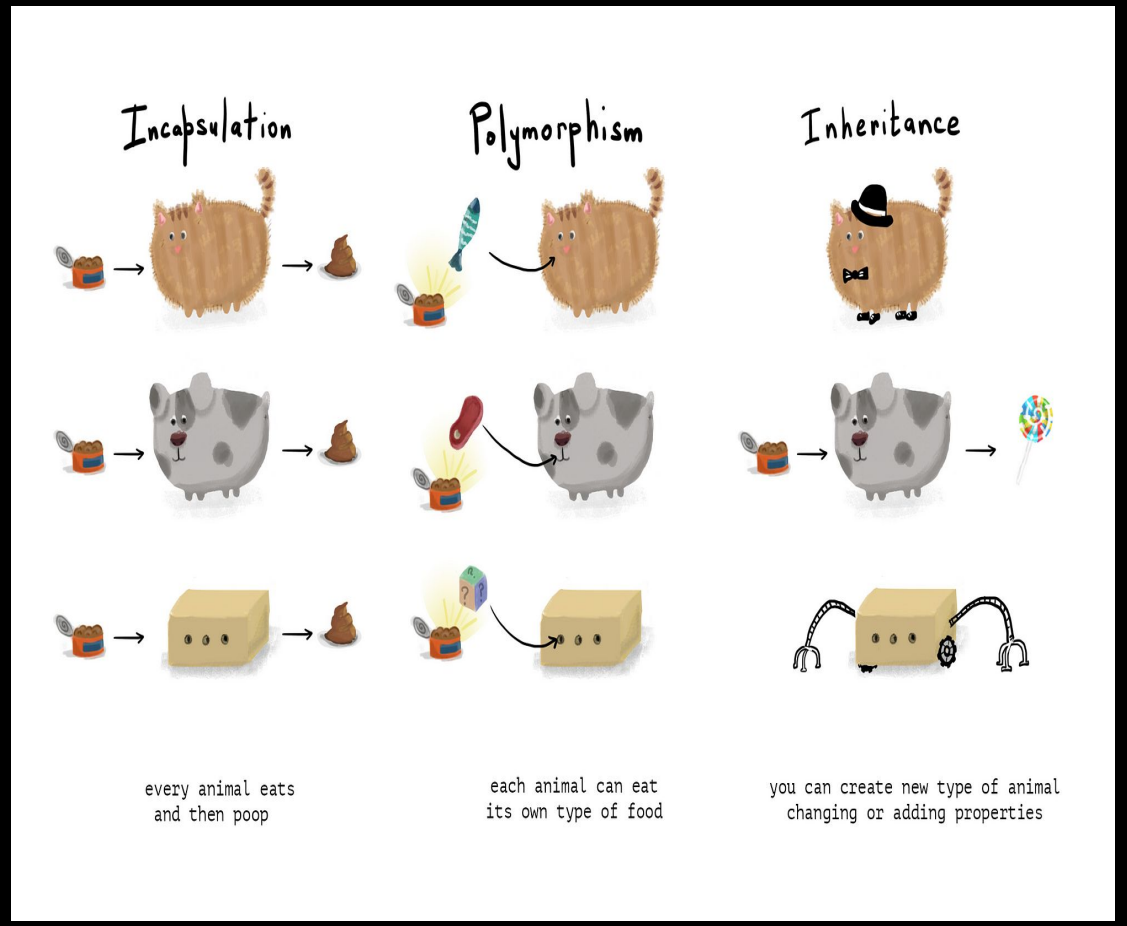
Основные черты объектно-ориентированной парадигмы программирования.

- **Объекты и классы:**
  Программа структурируется вокруг объектов, которые являются экземплярами классов. Классы определяют структуру объектов.
- **Инкапсуляция:**
  Инкапсуляция позволяет объединять данные и методы, обрабатывающие эти данные, в единый компонент. Детали реализации объекта скрыты от внешнего мира.
- **Наследование:**
  Наследование позволяет создавать новые классы на основе существующих, наследуя их свойства и методы. Это способствует повторному использованию кода и организации программы.
- **Полиморфизм:**
  Полиморфизм позволяет объектам разных типов использоваться в общих контекстах, что упрощает код и повышает гибкость программы.



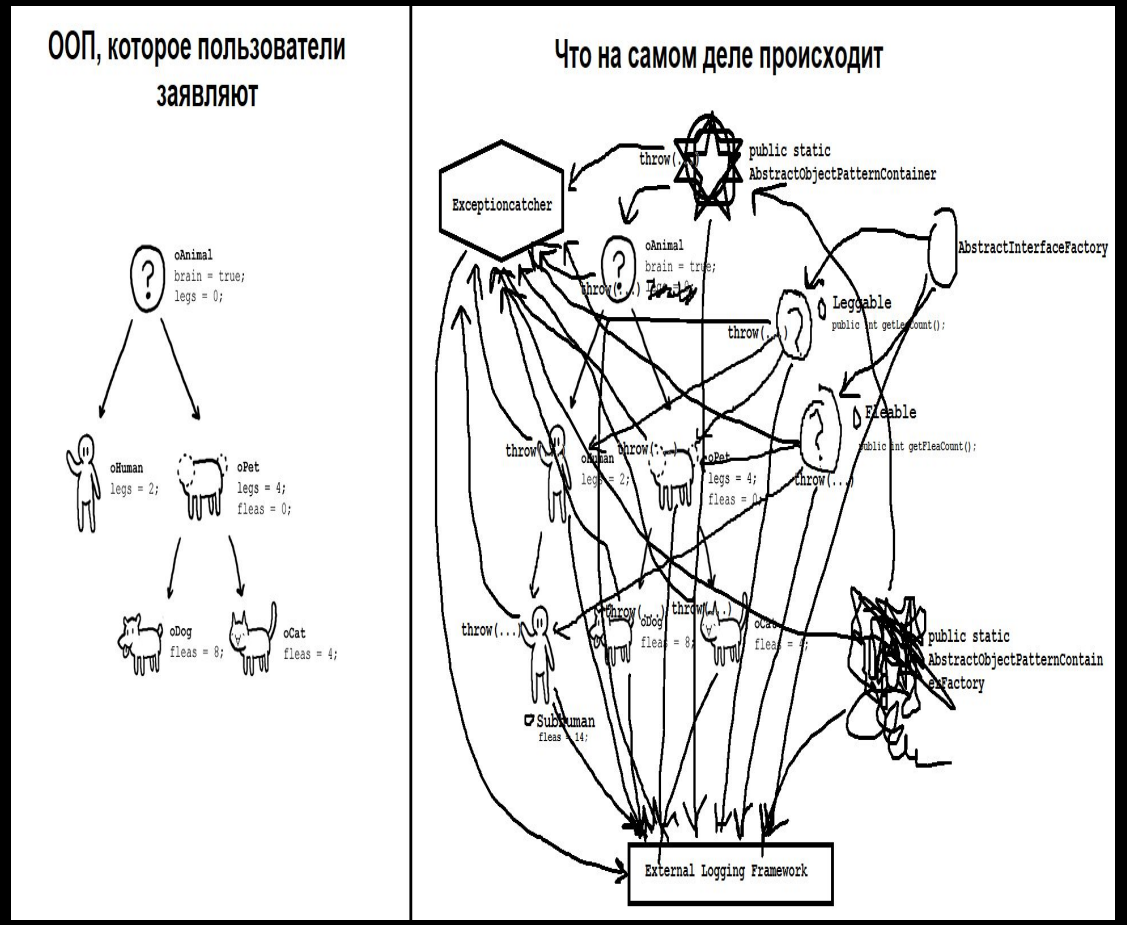
Основные преимущества объектно-ориентированной парадигмы программирования.

- **Модульность и повторное использование кода:**
  ООП позволяет создавать модули (классы и объекты), которые могут быть повторно использованы в различных частях программы.
- **Инкапсуляция и контроль доступа:**
  Инкапсуляция помогает скрывать детали реализации объектов, обеспечивая контроль доступа к данным и методам.
- **Гибкость и расширяемость:**
  Наследование и полиморфизм делают код более гибким и позволяют легко добавлять новый функционал или изменять существующий без модификации других частей программы.
- **Облегчение сопровождения:**
  ООП способствует созданию более читаемого и понятного кода, что упрощает сопровождение и отладку программ.



Основные ограничения объектно-ориентированной парадигмы программирования.

- **Перегрузка данных:**
  Использование объектно-ориентированной парадигмы может привести к избыточному использованию памяти, поскольку каждый объект содержит свои собственные данные.
- **Сложность:**
  При неудачном проектировании объектно-ориентированных систем код может стать слишком сложным и трудным для понимания.
- **Переизбыток гибкости:**
  Иногда объектно-ориентированное проектирование может привести к избыточной гибкости, когда слишком много абстракций усложняют программу.



Когда использовать ту или иную парадигму программирования?

- **Визуальная парадигма:**
  Когда использовать: Визуальная парадигма часто подходит для создания программ с простой логикой, где важна визуализация потока данных или бизнес-процессов. Это может быть полезно в случае разработки программ для обучения или моделирования бизнес-процессов.
- **Функциональная парадигма:**
  Когда использовать: Функциональная парадигма отлично подходит для задач, где функции могут рассматриваться как математические объекты, и где избегание изменяемого состояния данных является важным. Это может быть полезно при разработке параллельных или конкурентных систем.
- **Процедурная парадигма:**
  Когда использовать: Процедурная парадигма часто эффективна для разработки приложений с низкоуровневым программированием, манипуляцией памятью и выполнением последовательных шагов. Это может быть полезно при создании встроенных систем, систем управления ресурсами и других задач.
- **Объектно-ориентированная парадигма:**
  Когда использовать: ООП подходит для создания крупных и сложных систем, где важны модульность, повторное использование кода и управление сложностью. Это может быть полезно при разработке приложений с графическим интерфейсом, баз данных, веб-приложений и других проектов.

Важно отметить, что в реальных проектах часто используется смешанный подход, когда различные парадигмы комбинируются в зависимости от конкретных требований различных частей системы. Такой гибкий подход позволяет выбирать инструменты, наилучшим образом подходящие для решения конкретных задач в рамках проекта.

А на чём программировать ?

- **Визуальная парадигма:**
  Scratch: Это блок-ориентированный визуальный язык программирования, созданный для обучения детей основам программирования.
  LabVIEW: Язык, специально предназначенный для визуального программирования в области автоматизации, контроля и измерений.
  Blockly: Библиотека для создания среды визуального программирования, которая может быть встроена в произвольное веб-приложение.
- **Функциональная парадигма:**
  Haskell: Чистый функциональный язык программирования с сильной статической типизацией.
  Scala: Язык программирования, объединяющий функциональное и объектно-ориентированное программирование.
  Clojure: Диалект Lisp, работающий на платформе Java, с акцентом на функциональное программирование.
- **Процедурная парадигма:**
  C: Классический язык программирования, ориентированный на процедуры, часто используемый для системного программирования.
  Fortran: Язык программирования, первоначально разработанный для научных и инженерных вычислений, с акцентом на процедурном программировании.
  Pascal: Язык программирования, разработанный для обучения программированию и разработки приложений, поддерживает процедурное программирование.
- **Объектно-ориентированная парадигма:**
  Java: Объектно-ориентированный язык программирования, разработанный для обеспечения переносимости программного обеспечения через разные платформы.
  C++: Расширение языка C с добавлением объектно-ориентированных возможностей.
  Python: Объектно-ориентированный язык программирования, который также поддерживает множество других парадигм.

Это лишь небольшой обзор языков, и многие из них поддерживают сразу несколько парадигм. Выбор языка программирования зависит от требований конкретного проекта и предпочтений команды разработчиков.

Немного примеров:

- **Визуальная парадигма:**
  MIT App Inventor: Это визуальное программное обеспечение, которое позволяет людям создавать Android-приложения без необходимости написания кода.
  LabVIEW: Широко используется в области автоматизации, измерений и контроля. Программы в LabVIEW создаются визуально с использованием блок-диаграмм.
- **Функциональная парадигма:**
  Haskell Platform: Несколько программ и инструментов написаны на Haskell, включая компилятор GHC (Glasgow Haskell Compiler).
  Erlang: Используется в телекоммуникационных системах и распределенных, параллельных системах. Пример программы - серверы WhatsApp.
- **Процедурная парадигма:**
  Linux Kernel: Ядро операционной системы Linux написано на языке C, который является процедурным языком программирования.
  Windows Command Prompt (cmd.exe): Командная оболочка в операционной системе Windows написана на языке командной оболочки, который следует процедурной парадигме.
- **Объектно-ориентированная парадигма:**
  Java Development Kit (JDK): Весь язык Java и его библиотеки построены на основе объектно-ориентированной парадигмы.
  Microsoft Office Suite: Программы, такие как Microsoft Word и Excel, написаны с использованием объектно-ориентированного программирования.

Спасибо за внимание!