Основы алгоритмизации и программирования

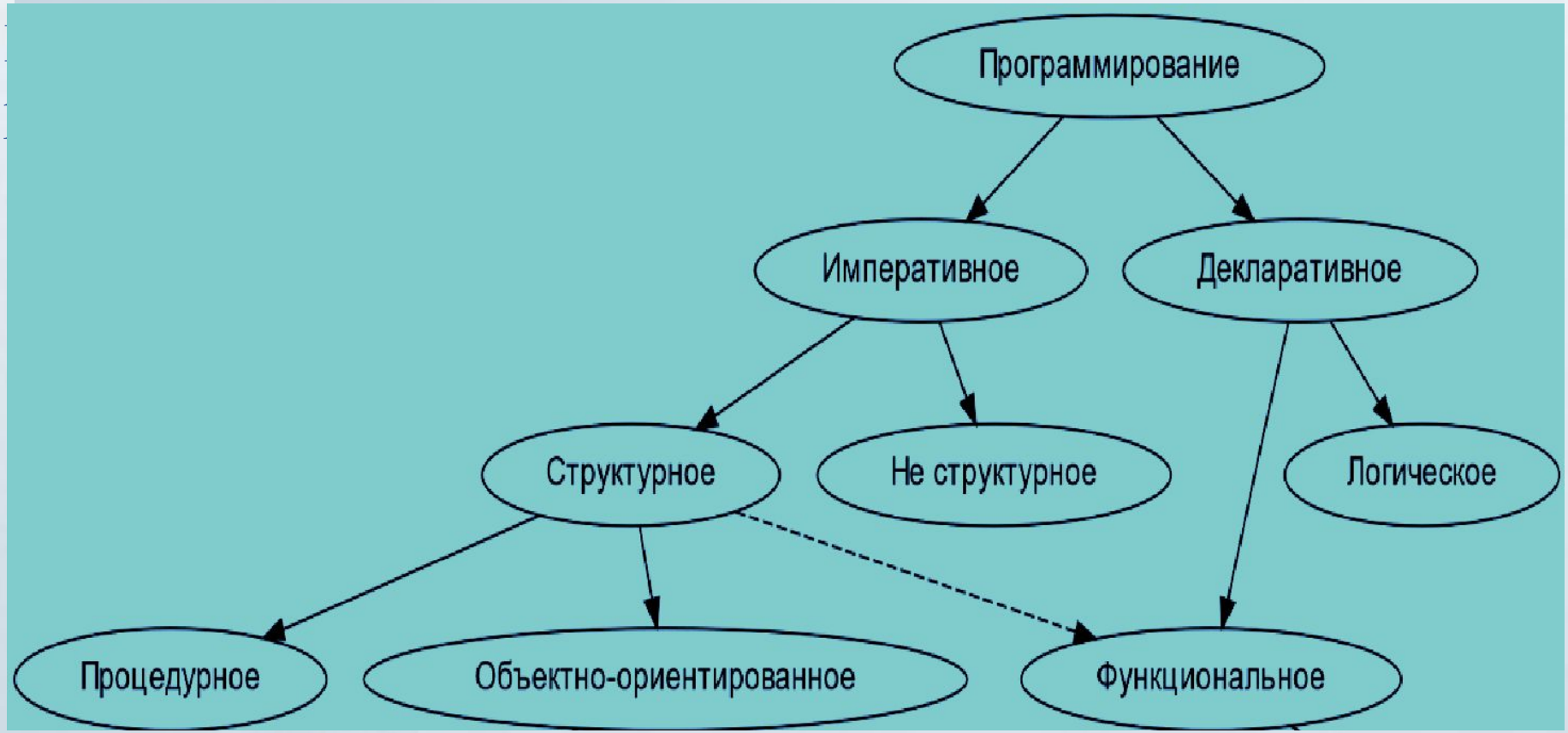
Парадигмы программирования

Преподаватель Ставицкая Е.А.

Содержание

Определение

Парадигмы программирования - системы базовых понятий (способа мышления), определяющие способ написания компьютерных программ.



Свойства парадигм программирования

Парадигма	Ключевой концепт	Программа	Выполнение программы	Результат
Императивная (процедурный, структурный, модульный подходы)	Команда	Последовательность команд	Исполнение команд	Итоговое состояние памяти
Функциональная	Функция	Набор функций	Вычисление функций	Значение главной функции
Логическая	Предикат	Логические формулы	Логическое доказательство	Результат доказательства
Объектно-ориентированная	Объект	Набор классов объектов	Обмен сообщениями между объектами	Результирующее состояние объектов
Аспектно-ориентированная	Аспект	Комбинация советов и точек соединения	Применение совета в точках соединения, определенных некоторым срезом	Сквозная функциональность

Императивная парадигма

Это наиболее понятный и логичный подход, когда задаётся последовательность действий (команд, инструкций), необходимых для получения результата – изменения состояния (информации, данных, памяти) программы с помощью переменных, операторов присваивания и составных выражений.

Подходы	Языки программирования	Годы создания
операторный	машинный язык, автокоды (ассемблер)	
процедурный	COBOL, ALGOL, FORTRAN, PL/1, FORT, Макроассэмблер	
структурный	Paskal, C	

Язык ассемблера (автокод) — язык программирования низкого уровня. В отличие от языка машинных кодов, позволяет использовать более удобные для человека мнемонические (символьные) обозначения команд. Для перевода с языка ассемблера в понимаемый процессором машинный код требуется специальная программа, называемая ассемблером.

Императивная парадигма

Особенности:

- ✔ Данная парадигма предполагает описания вычислений в форме инструкций, которые постепенно меняют состояние программы.
- ✔ В языках низкого уровня, например в ассемблерах, состояниями могут являться память, регистры и флаги. Инструкциями же выступают команды, которые поддерживаются целевым процессором.
- ✔ В языках более высокого уровня, например в Си, состоянием является исключительно память. Инструкции в таком случае могут быть более сложными, а также приводить к выделению и высвобождению памяти по мере своего функционирования.

Функциональная парадигма

Функциональный подход — противоположность императивному, в котором программист задает программе четкий порядок действий по шагам: программа сама решает, как и в каком порядке исполнять действия, а программист описывает правила взаимодействия и связи между компонентами.

Функциональное программирование распространено при работе с важными данными или при решении задач, где нужны сложные вычисления.

Функциональность определяется подходом: весь код описывается как правила работы с информацией, и они могут

Оптимизация кода - различные методы преобразования кода ради улучшения его характеристик и повышения эффективности

условиях в течение заданного периода времени с достаточно большой возможности для оптимизации.
вероятностью.

Функциональная парадигма

Пример языков, ориентированных на функциональное программирование:

- ✔ Haskell, F#, OCaml, ELM, серия языков Lisp, а также Erlang и его потомок Elixir.
- ✔ Scala и Nemerle, хотя эти языки дают возможность программировать и в функциональном, и в императивном стилях. Они старые и сейчас применяются не так часто, как большинство современных.
- ✔ Несколько узкоспециализированных языков: язык вычислительной платформы Wolfram, языки J и K для финансового анализа, язык R для статистических целей. На функциональном языке APL базируется язык научной математической среды MATLAB. Также сюда можно отнести языки, которые используются при работе с электронными таблицами, а еще, частично, SQL — язык запросов для работы с базами данных.

Функциональная парадигма

Особенности функционального подхода:

- ✔ отсутствие жесткой последовательности. Об этом мы уже говорили. Разработчик задает правила, а компилятор кода сам решает, в какой последовательности их выполнять. Жесткий порядок действий программист не задает. Его выбирает сама программа.
- ✔ Использование «чистых функций», удовлетворяющих двум условиям:
  - при одинаковых входных данных функция всегда вернет одинаковый результат. То есть, функция, возвращающая сумму a и b, может быть чистой, а функция, возвращающая случайное число, — нет;
  - когда функция выполняется, не возникают побочные эффекты - так называют действия, которые влияют на что-то за ее пределами. Например, изменение переменной, чтение данных или вывод в консоль — это побочные эффекты.

Функциональная парадигма

Особенности функционального подхода:

- ✔ неизменные переменные - функциональном программировании нет переменных в привычном виде.

  В нем все объявленные переменные неизменны - то есть фактически это константы.

  Если с какой-то переменной нужно провести вычисления, она не изменяется: создается новая переменная, и результат вычислений записывается в нее. А исходная остается прежней — ее значение не меняется

Функциональная парадигма

Особенности функционального подхода:

- ✔ «первоклассные» функции высшего порядка. Все функции в функциональном программировании должны быть первого класса и высшего порядка.

- Функция первого класса — это такая, которую можно представить как переменную. То есть, ее можно передавать как аргумент другим функциям, возвращать как результат работы других функций, сохранять в переменную или структуру данных.
- Функция высшего порядка — такая, которая принимает в качестве аргументов функции или возвращает их в качестве результата.

Функциональная парадигма

Особенности функционального подхода:

- ✔ относительная прозрачность. Это не означает, что функция должна выдавать одинаковый результат во всех случаях — только при одинаковых входных данных.
- ✔ рекурсия вместо циклов. Очень многие алгоритмы в функциональном подходе построены на рекурсии — функциях, вызывающих себя. Так реализованы многие действия, где что-то нужно выполнить несколько раз.

- Цикл — несколько выполнений одной и той же части кода подряд;
- Рекурсия — явление, когда функция вызывает сама себя, но с другими аргументами.

Функциональная парадигма

Особенности функционального подхода:

- ✔ лямбда-исчисление - особая математическая система, которая используется в функциональных языках программирования. Ее название также иногда пишут как λ-исчисление.

Лямбда-исчисление находит применение во многих различных областях, таких как математика, философия, лингвистика и информатика. Лямбда-исчисление сыграло важную роль в развитии теории языков программирования.

Лямбда-исчисление:

- это правила построения и вычисления безымянных функций;
- это не язык программирования, а формальный аппарат, способный определить в своих терминах любую языковую конструкцию или алгоритм.

Логическая парадигма

Логическое программирование – это парадигма программирования, основанная на математической логике и формальной логике.

В логическом программировании программа представляет собой набор логических утверждений, из которых система может выводить ответы на запросы.

Основные понятия:

- логические утверждения. Логические утверждения в логическом программировании записываются с помощью предикатов и правил. Предикаты – это утверждения о свойствах объектов или отношениях между объектами.
- факты - это простые утверждения о свойствах объектов. Факты представляют собой базу знаний, на основе которой система может делать выводы.

Логическая парадигма

Основные понятия:

- правила – это утверждения, которые определяют отношения между объектами или выводы, которые можно сделать на основе предикатов.
- запросы – это вопросы, которые задаются системе на основе предикатов. Система использует базу знаний и правила для поиска ответов на запросы. Запросы могут быть простыми или составными, и система будет искать все возможные ответы на запросы.
- унификация – это процесс сопоставления предикатов и переменных для нахождения значений, которые удовлетворяют условиям. Унификация используется для проверки правил и поиска ответов на запросы

Логическая парадигма

Принципы логического программирования:

- ✔ Декларативность.
- ✔ Логическая инверсия
- ✔ Рекурсия
- ✔ Недетерминизм
- ✔ Логическое следование

Логическое программирование – это подход к программированию, основанный на использовании логических выражений и правил вывода. Оно позволяет описывать отношения между объектами и задавать логические условия для их вывода. Логическое программирование имеет свои преимущества, такие как декларативность и возможность автоматического вывода решений, но также и недостатки, такие как сложность в понимании и ограниченность в выразительности.

Логическое следование – это процесс вывода новых фактов или правил на основе имеющихся в базе знаний. Система может использовать правила и факты для вывода новых фактов или для проверки условий. Логическое следование позволяет системе автоматически находить новые знания и использовать их для решения задач.

Логическая парадигма

Применение логического программирования

- ✔ Искусственный интеллект

Логическое программирование широко используется в области искусственного интеллекта для решения задач, таких как автоматическое доказательство теорем, экспертные системы, обработка естественного языка и машинное обучение. Пролог, один из наиболее популярных языков логического программирования, часто используется для реализации таких систем.

- ✔ Базы знаний и базы данных

Логическое программирование может быть использовано для создания баз знаний и баз данных, где факты и правила могут быть представлены в виде логических утверждений. Это позволяет эффективно организовывать и обрабатывать информацию, а также выполнять запросы и поиск по базе данных.

Логическая парадигма

Применение логического программирования

- ✔ Разработка программного обеспечения

Логическое программирование может быть использовано для разработки программного обеспечения, особенно в области символьных вычислений и автоматического доказательства теорем. Логические языки программирования позволяют описывать проблему в терминах фактов и правил, а затем автоматически генерировать решение на основе этих описаний.

- ✔ **Лингвистика и обработка естественного языка**

Логическое программирование может быть использовано для моделирования языковых структур и обработки естественного языка. Оно позволяет описывать грамматические правила и семантические отношения, а также выполнять анализ и генерацию текста на основе этих описаний.

Логическое программирование

Понятие	Определение	Свойства
Логическое программирование	Парадигма программирования, основанная на формализме логического вывода	- Декларативность – программист описывает, что нужно получить, а не как это сделать - Недетерминизм – возможность нескольких решений для одной задачи - Рекурсия – возможность определения функций через их самих
Принципы логического программирования	Логическое следование – вывод новых фактов из имеющихся	- Унификация – процесс сопоставления и объединения термов - Резольвента – текущее состояние программы, содержащее цель и базу знаний
Преимущества логического программирования	- Простота и наглядность кода - Мощные возможности для решения задач логического вывода - Возможность автоматического поиска решений	- Недостатки логического программирования - Ограниченная эффективность в некоторых задачах - Сложность отладки и анализа программ
Применение логического программирования	- Искусственный интеллект - Экспертные системы - Базы знаний	

Объектно-ориентированная парадигма

Объектно-ориентированное программирование (ООП) представляет собой подход, который рассматривает программу в качестве набора объектов, взаимодействующих между собой. Каждый объект — это определённая сущность со своими данными и набором доступных действий.

ООП помогает ускорить процесс написания кода и сделать его более читаемым.

Основными составляющими объектно-ориентированной методологии являются объектно-ориентированный анализ, объектно-ориентированное проектирование и объектно-ориентированное программирование.

Составляющие ООП методологии

- Объектно-ориентированный анализ предполагает создание объектно-ориентированной модели предметной области - использовании аппарата объектно-ориентированной методологии для представления реальной системы.
- Объектно-ориентированное проектирование представляет собой процесс описания классов будущего программного обеспечения с использованием формальных методов (как правило, графических), а также определения взаимодействия классов и объектов.
- Объектно-ориентированное программирование - непосредственное создание классов и объектов, а также определение связей между ними, выполняемое с использованием одного из языков объектно-ориентированного программирования.

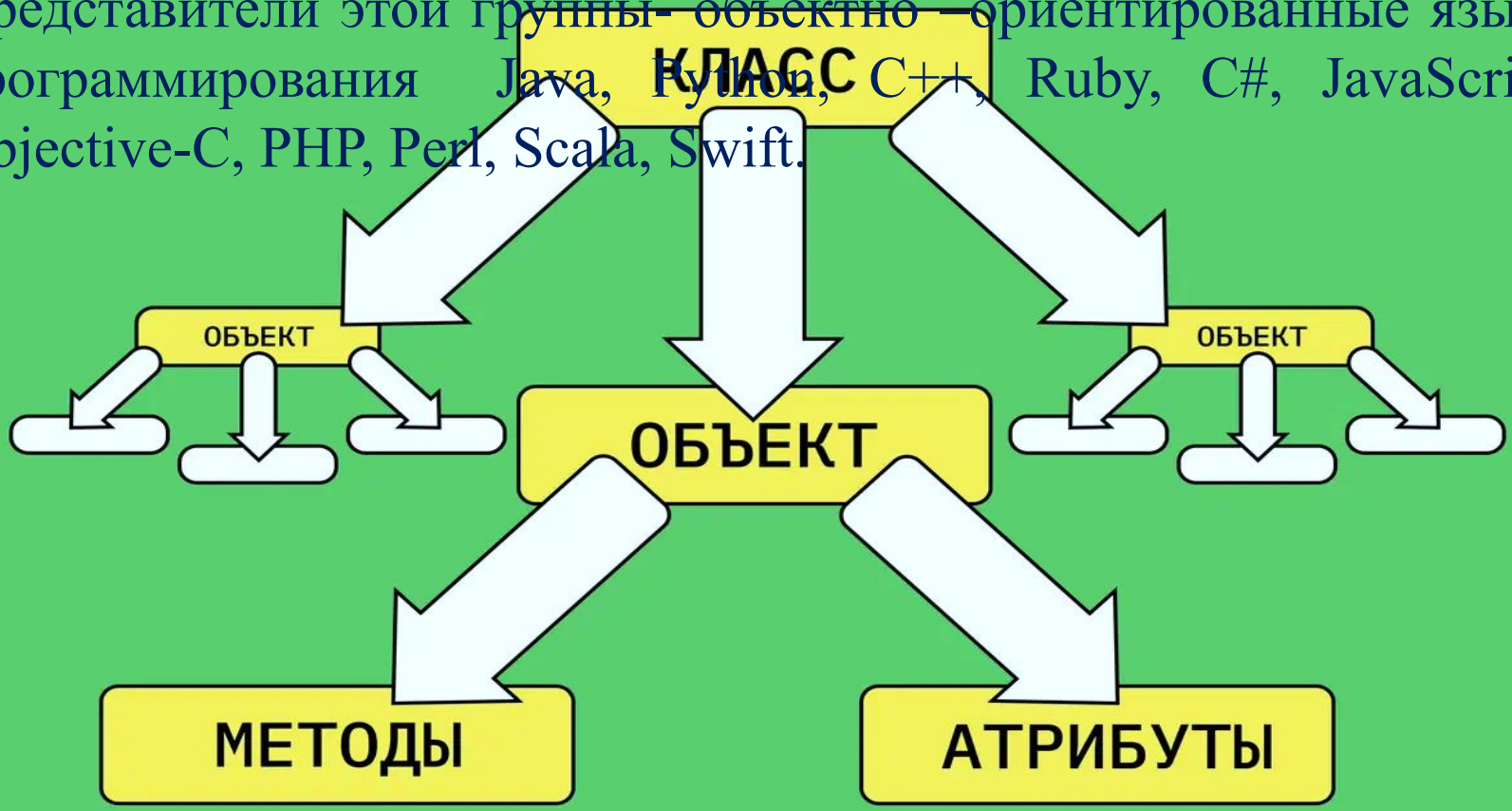
Объектно-ориентированная парадигма

Объектно-ориентированное программирование – это представление программы в виде совокупности взаимодействующих объектов, каждый из которых является экземпляром определенного класса, а классы являются членами определенной иерархии наследования.

- ✔ Объект – структура, которая объединяет данные и методы, которые эти данные обрабатывают. Фактически, объект является основным строительным блоком объектно-ориентированных программ.
- ✔ Класс – шаблон для объектов. Каждый объект является экземпляром какого-либо класса («безклассовых» объектов не существует). В рамках класса задается общий шаблон, структура, на основании которой создаются объекты. Данные, относящиеся к классу, называются полями класса, а программный код для их обработки называется методами класса.

Объектно-ориентированная парадигма

Представители этой группы- объектно –ориентированные языки программирования Java, Python, C++, Ruby, C#, JavaScript, Objective-C, PHP, Perl, Scala, Swift.



Преимущества и недостатки ООП

Преимущества	Недостатки
В парадигме объектов легче писать код. Удобно создать класс или метод и использовать.	Сложность в освоении. ООП требует больше знаний. Необходимо разобраться в классах, наследовании, писать публичные и внутренние функции, изучать взаимодействие объектов.
Читать код гораздо проще. Видны конкретные объекты и методы.	Громоздкость. Требуется создавать классы, объекты, методы и атрибуты даже для маленьких программ.
Код легче обновлять. Изменение в одном месте распространяется на все объекты.	Низкая производительность. Объекты потребляют больше памяти и могут снижать скорость компиляции.
Удобство работы в команде. Разные люди могут отвечать за разные объекты.	
Код можно переиспользовать. Один раз написанный класс можно использовать в разных проектах.	
Шаблоны проектирования. Готовые решения для взаимодействия классов, шаблоны упрощают написание кода.	

Аспектно-ориентированная парадигма

АОП — аспектно-ориентированное программирование — это парадигма, направленная на повышение модульности различных частей приложения за счет разделения сквозных задач. Для этого к уже существующему коду добавляется дополнительного поведение, без изменений в изначальном коде.

Аспектно-ориентированное программирование предназначено для решения сквозных задач, которые могут представлять собой любой код, многократно повторяющийся разными методами, который нельзя полностью структурировать в отдельный модуль.

В качестве примера можно привести применение политики безопасности в каком-либо приложении, а так же логирование.

Аспектно-ориентированная парадигма

Ключевые понятия АОП:

- ✔ Аспекты (aspects, «что») — это добавочное поведение, которое нужно включить в целевой код.
- ✔ Срезы (pointcuts, «где») указывают место в целевом коде, куда следует внедрить аспект. Теоретически оно может быть любым.
- ✔ Советы (advice, «когда») определяют момент, когда должен выполняться код аспекта, например:
  - before (перед вызовом метода),
  - after (после возврата в точку вызова),
  - around (до вызова метода и при возврате из него),
  - whenThrowing (когда метод вызывает исключение) и так далее.

Аспектно-ориентированная парадигма

Примеры сквозной функциональности

- ✔ Идентификация пользователей/ролей, авторизация операций
- ✔ Функции защиты, логирования, репликации
- ✔ Сериализация/десериализация
- ✔ Трассировка
- ✔ Управление ресурсами
- ✔ Кэширование объектов
- ✔ Декорирование (пул потоков)
- ✔ Ленивая инициализация объектов
- ✔ Обработка исключений
- ✔ Изменение контекста выполнения
- ✔ Мониторинг, аудит

Аспектно-ориентированная парадигма

Достоинства	Недостатки
Улучшение декомпозиции	Недостаточная проработка методологии АОП-разработки программ.
Минимальное связывание, снижение внешних зависимостей	Недостаточное качество реализаций расширений ОО-языков
Улучшение повторного использования	Недостаточная проработка механизма привязки аспектов к компонентам, вносящая фактическую зависимость от аспектов
	Неполная исследованность случаев применения

При разработке программного обеспечения программисты и архитекторы пользуются декомпозицией — представлением объектов и взаимосвязей между ними в виде классов, объектов, их свойств и методов. Проводя декомпозицию, удается получить более точное представление объектов из реальной жизни в виде программного кода

Реализации АОП

Для .Net:

- PostSharp
- Aspect.NET
- Aspect#
- Unity
- Spring.NET AOP
- Seasar.NET

Для Java:

- AspectJ (AJDT plug-in)
- Jakarta Hivemind
- Dynaop
- JBoss AOP
- Reflex
- Seasar
- Spring Framework AOP

Для JavaScript:

- Ajaxpect
- Jquery AOP
- Dojo Toolkit
- Aspectes

Для C/C++:

- AspectC++
- FeatureC++
- The XWeaver Project

Для PHP:

- Seasar.PHP
- GAP: Generic Aspects
- paspect

Для Python:

- Aspyct
- Python AOP
- PEAK
- Pythius
- PyPy

Для Ruby:

- AspectR
- Aquarium

Другие:

- AspectLua
- Fling (ActionScript)
- AspectL (Lisp)
- AspectCocoa

Мультипарадигма

В современном мире многие языки совмещают в себе несколько парадигм программирования.

Все чаще программисты замечают, что границы современных парадигм становятся все более и более размытыми, что приводит к появлению новых парадигм: динамические типы на статическом языке, и статические способности появляются на динамическом языке.

Теперь все основные языки программирования влияют на функциональный язык, все более очевидной тенденцией является «мульти – стильный язык программирования».

Распределение ЯП по парадигмам программирования

Парадигма	Языки программирования
Императивная (процедурный, структурный, модульный подходы)	Языки ассемблера, Basic, Pascal, C, C++, Python, Ruby, Java, C#, PHP, JavaScript
Функциональная	Perl, PHP, Scala, Haskell, Python
Логическая	Prolog, SNOBOL
Объектно- ориентированная	Smalltalk, Ruby, Python, Java, C++
Аспектно-ориентированное	AspectJ, Python
Мультипарадигма	Perl, Python, Ada, C++, C#,

Информационные источники

1. Парадигмы и языки в обучении информатике и программированию. / Большакова Е.И., Баева Н.В., Груздева Н.В., Горячая И.В.; МГУ им. М.В.Ломоносова, факультет ВМК, Москва, ГСП-1, Ленинские горы 1-52,119991
2. Индекс сообщества программистов TIOBE. Источник: https://www.tiobe.com/tiobe-index/
3. HOPL - Генеалогическое древо ЯП (Онлайн – историческая энциклопедия языков программирования, начиная с 18 века) Источник: https://hopl.info/