In [computing](https://en.wikipedia.org/wiki/Computing), aspect-oriented programming (AOP) is a [programming paradigm](https://en.wikipedia.org/wiki/Programming_paradigm) that aims to increase [modularity](https://en.wikipedia.org/wiki/Modularity_(programming)) by allowing the [separation of](https://en.wikipedia.org/wiki/Separation_of_concerns) [cross-cutting concerns](https://en.wikipedia.org/wiki/Cross-cutting_concern). It does so by adding additional behavior to existing code (an [advice](https://en.wikipedia.org/wiki/Advice_(programming))) without modifying the code itself, instead separately specifying which code is modified via a "[pointcut](https://en.wikipedia.org/wiki/Pointcut)" specification, such as "log all function calls when the function's name begins with 'set'". This allows behaviors that are not central to the [business logic](https://en.wikipedia.org/wiki/Business_logic) (such as logging) to be added to a program without cluttering the code, core to the functionality. AOP forms a basis for [aspect-oriented software development](https://en.wikipedia.org/wiki/Aspect-oriented_software_development).

В вычислительной технике аспектно-ориентированное программирование (АОП) - это парадигма программирования, направленная на повышение модульности за счет разделения сквозных задач. Он делает это, добавляя дополнительное поведение к существующему коду (совет) без изменения самого кода, вместо этого отдельно указывая, какой код модифицируется с помощью спецификации «pointcut», например, «записывать все вызовы функций, когда имя функции начинается с 'set'. ". Это позволяет добавлять в программу поведение, не являющееся центральным для бизнес-логики (например, ведение журнала), не загромождая код, являющийся основным для функциональности. АОП формирует основу для аспектно-ориентированной разработки программного обеспечения.

AOP includes programming methods and tools that support the modularization of concerns at the level of the source code, while "aspect-oriented software development" refers to a whole engineering discipline.

АОП включает в себя методы и инструменты программирования, которые поддерживают модульность задач на уровне исходного кода, в то время как «аспектно-ориентированная разработка программного обеспечения» относится ко всей инженерной дисциплине.

Aspect-oriented programming entails breaking down program logic into distinct parts (so-called concerns, cohesive areas of functionality). Nearly all programming paradigms support some level of grouping and [encapsulation](https://en.wikipedia.org/wiki/Encapsulation_(computer_science)) of concerns into separate, independent entities by providing abstractions (such as functions, procedures, modules, classes, methods) that can be used for implementing, abstracting and composing these concerns. Some concerns "cut across" multiple abstractions in a program, and defy these forms of implementation. These concerns are called cross-cutting concerns or horizontal concerns.

Аспектно-ориентированное программирование влечет за собой разбиение логики программы на отдельные части (так называемые задачи, связанные функциональные области). Практически все парадигмы программирования поддерживают определенный уровень группировки и инкапсуляции проблем в отдельные независимые объекты, предоставляя абстракции (например, функции, процедуры, модули, классы, методы), которые можно использовать для реализации, абстрагирования и объединения этих проблем. Некоторые проблемы «пересекают» несколько абстракций в программе и бросают вызов этим формам реализации. Эти проблемы называются сквозными или горизонтальными.

[Logging](https://en.wikipedia.org/wiki/Data_logging) exemplifies a crosscutting concern because a logging strategy necessarily affects every logged part of the system. Logging thereby crosscuts all logged classes and methods.

Ведение журнала представляет собой комплексную проблему, поскольку стратегия ведения журнала обязательно влияет на каждую регистрируемую часть системы. Таким образом, ведение журнала пересекает все зарегистрированные классы и методы.

All AOP implementations have some crosscutting expressions that encapsulate each concern in one place. The difference between implementations lies in the power, safety, and usability of the constructs provided. For example, interceptors that specify the methods to express a limited form of crosscutting, without much support for type-safety or debugging. [AspectJ](https://en.wikipedia.org/wiki/AspectJ) has a number of such expressions and encapsulates them in a special class, an [aspect](https://en.wikipedia.org/wiki/Aspect_(computer_science)). For example, an aspect can alter the behavior of the base code (the non-aspect part of a program) by applying [advice](https://en.wikipedia.org/wiki/Advice_in_aspect-oriented_programming) (additional behavior) at various [join points](https://en.wikipedia.org/wiki/Join_point) (points in a program) specified in a quantification or query called a [pointcut](https://en.wikipedia.org/wiki/Pointcut) (that detects whether a given join point matches). An aspect can also make binary-compatible structural changes to other classes, like adding members or parents.

Все реализации АОП имеют несколько пересекающихся выражений, которые объединяют каждую проблему в одном месте. Разница между реализациями заключается в мощности, безопасности и удобстве использования предоставленных конструкций. Например, перехватчики, которые определяют методы для выражения ограниченной формы пересечения, без особой поддержки безопасности типов или отладки. AspectJ имеет несколько таких выражений и инкапсулирует их в специальный класс, аспект. Например, аспект может изменить поведение базового кода (не аспектная часть программы), применяя совет (дополнительное поведение) в различных точках соединения (точках в программе), указанных в количественной оценке или запросе, называемом pointcut ( который определяет, соответствует ли данная точка соединения). Аспект также может вносить бинарно-совместимые структурные изменения в другие классы, такие как добавление членов или родителей.