**Функциональное программирование: что это такое, преимущества и недостатки**

Функциональное программирование на сегодняшний день является одним из приоритетных направлений развития кода. Оно кардинально отличается от императивной, объектно-ориентированной парадигмы. К примеру, если в императивном подходе используются инструкции, то в функциональном – функции, набор правил, которые необходимо выполнять без строгой последовательности действий.

## Что такое функциональные системы программирования

Практически каждый программист первым делом изучал объектно-ориентированную методологию разработок. Обычно вхождение в эту специальность предполагает знакомство с [**языками Java**](https://gb.ru/blog/yazyk-programmirovaniya-java/?from=blog_stati_ankor) или C++, а в лучшем случае Ruby, Python или C#. Такой разработчик уже точно будет иметь представление о классах, объектах и т.д. Но вот основы функционального программирования, скорее всего, ему еще не будут знакомы. Эта парадигма существенно отличается не только от объектно-ориентированного подхода, но и от других методологий (процедурная, прототипно-ориентированная и др.).

Функциональное программирование становится все более востребованным. Это обусловлено рядом факторов. Нельзя сказать, что это новая парадигма. Так, еще в девяностых годах появился язык программирования Haskell, который и сейчас активно используется для ФП. В эту же категорию могут быть отнесены также Erlang, Scala, Clojure. Все подобные языки программирования объединяет одно очень важное преимущество. С их помощью можно писать конкурентные программные продукты, поэтому при их использовании отпадают такие проблемы, как взаимные блокировки и потокобезопасность.

Суть функциональной парадигмы программирования заключается в том, что разработчик должен задавать не последовательность требуемых команд, а описать принцип их взаимодействия с подпрограммами. Это напоминает работу в ООП, но в этом случае реализация осуществляетсК примеру, в объектно-ориентированном программировании разработчик должен задать объекты и описать алгоритмы их взаимодействия друг с другом, но, при этом, есть возможность для написания кода, не связанного с ними. Такой код выглядит обособленным и оказывает влияние на работу всей программы. Он направляет некоторые объекты для взаимодействия между собой, осуществляет обработку определенных результатов и т.д.я на уровне всего программного продукта.

Функциональная парадигма в этом плане продвинулась глубже. При таком программировании код описывает правила работы с данными. Разработчик лишь формирует необходимые условия, а код уже непосредственно определяет их практическое применение.

Сравнивая процедурное и функциональное программирование, можно отметить совпадения только в наличии команд, которые выполняет язык. Все остальные аспекты этих парадигм отличаются.

Понятие функционального программированиясвязано с парадигмами, при которых приложения разрабатываются путем последовательного использования функций. А при процедурном подходе применяются инструкции.

## Преимущества и недостатки функционального программирования

Особенности функционального программирования обеспечивают более высокую чистоту кода и его простую читаемость. Благодаря таким принципам составления программ, появляется возможность предотвратить ненужные абстракции с непредсказуемыми действиями. Таким образом, можно обеспечить высокую предсказуемость работы программы и сократить численность потенциальных ошибок.

Методы функционального программирования неэффективны для разработки алгоритмов, которые построены на графах. Это связано с более медленной работой приложения. Можно сказать, что парадигмы функционального программирования не ориентированы на использование для решений, которые на протяжении многих лет базировались на императивных принципах.

Основной момент заключается в том, что языки функционального программирования не имеют действенного неупорядоченного словаря множества. Если сравнивать с хэш-таблицами, то словари языков, построенных на функциях, работают не так эффективно, что может стать преградой для работы ряда предложений. Но, такие минусы имеют отношение только к императивным языкам.

## 5 принципов функционального программирования

Теперь поговорим о преимуществах. Основное — это возможности функционального программирования по максимуму использовать ресурсы процессора. Это обусловлено конкурентным поведением и определяется принципами, которые, впрочем, не все можно назвать обязательными. Есть много языков, в которых такие принципы поддерживаются не в полном объеме. Это обусловлено их теоретическим характером. Принципы функционального программирования устанавливаются, чтобы максимально точно дать определение функциональной парадигмы. Основные 5 выглядят следующим образом:

### Применяются исключительно чистые функции

Это базовый принцип функционального программирования. Чистыми считаются функции, которые соответствуют 2-м условиям:

* Функции, которые вызывает одинаковым аргументом, обязательно должны возвращать одно и то же значение.
* В процессе выполнения функции не должны иметь место побочные эффекты.

В соответствии с первым условием, к примеру, при вызове функции (подпрограммы) sum(2, 3), мы понимаем, что всегда в результате получим 5. Но, если идет обращение к подпрограмме rand() или к переменным, которые не определены функцией, то условие чистоты не соблюдается, а это является недопустимым моментом для рассматриваемой парадигмы.

Условие номер два в отношении недопустимости побочных эффектов является более обширным. Что такое побочный эффект? В контексте данной парадигмы это изменение какого-либо элемента, не связанного с функцией, исполняемой в конкретный момент. Примерами таких эффектов, которые приводят к нарушению условия чистоты, могут служить: вызовы переменной вне функции, вывод в консоль, вызов исключения, чтение файловых данных.

На первый взгляд такое ограничение кажется очень строгим, но при наличии уверенности в том, что обращение к функции не спровоцирует «внешних» изменений, подпрограмму можно применять во всех сценариях. Эта особенность обеспечивает конкурентность программирования многопоточных программ.

### В данной парадигме предусматривается применение функций первого класса и высшего порядка

Такой же принцип, помимо языков функционального программирования, действует в Javascript, PHP и в ряде других систем. Но для ФП он является обязательным.

Чтобы функция могла считаться первоклассной, должна присутствовать возможность для ее объявления в виде переменной. Благодаря такому условию можно управлять подпрограммой так же, как и обычными типами данных, а, при этом, и исполнять ее.

Неизменность переменных

Подпрограммы высшего порядка допускают принятие в качестве аргумента других функций. При этом, они могут и возвращать функцию. Яркими примерами подобных подпрограмм можно назвать map и filter.

### Неизменность переменных

Этот принцип достаточно прост. Используя функциональную модель программирования, нельзя менять переменную после инициализации. Можно создавать новую, но изменение существующей недопустимо. Благодаря этому принципу мы получаем уверенность, что переменные не будут изменены.

### Относительная прозрачность подпрограмм

Этот принцип объяснить намного сложнее. Наиболее соответствующее истине определение выглядит так: функция считается относительно прозрачной, при условии, что при замене вызова функции на возвращаемое значение, состояние не изменится.