Данный документ является результатом работы компании - avasite, Глотова Жеки, Валуйской Яны, Бротиковской Даны, Ковальчука Саши и Галеева Данилы, а так же википедии, и трудов всех людей, которые делали различного сорта теормины до нас.

си, с++, c#, java, Objective C, Python, JavaScript, Visual Basic, PHP ада83, ада95, Оберон 1/2, Delphi, Modula 1/2, Турбо Паскаль, Lisp, SmallTalk

Prolog, Cobol, Eiffel, Erlang, F#, Groovy, D, Haskel, Perl, Nemerle, Ruby, Scala, Tcl, Ocaml

**Терминология**

1. Парадигмы
   1. **Императивная** - противоположность декларативному. Императивный язык должен описывать не столько саму задачу, сколько её решение. Несмотря на различие декларативных и императивных языков в конечном счёте скомпилированная программа представляет собой набор чётких и недвусмысленных инструкций, то есть императивна.
   2. **Объектно-ориентированная** - объектно-ориентированный язык позволяет использовать три парадигмы Объектно-ориентированное программирование: наследование, инкапсуляцию и полиморфизм.
   3. **Рефлексивная** - наличие в языке мощных механизмов интроспекции, функции eval. Возможность программы на данном языке оперировать собственным кодом как данными.
   4. **Функциональная** - позволяет записывать программу как композицию функций. В чистом функциональном языке нет переменных. Так как функции не имеют побочных эффектов, они могут выполняться в любом порядке.
   5. **Обобщённое программирование** - обобщенное программирование позволяет записывать алгоритмы, принимающие данные любого типа.
   6. **Логическая** – использует логику предикатов для описания баз данных и процедур логического вывода и принятия решений.
   7. **Доказательная** - направлен на разработку алгоритмов и программ с доказательствами их правильности с использованием спецификаций программ.
   8. **Декларативная** - противоположность императивному. Декларативный язык описывает не столько решение задачи, сколько саму задачу, а решение уже должен определять компьютер. Следует заметить, что различие между императивными и декларативными языками условно: на любом языке можно написать процедуру решения какой-либо типовой задачи и затем описывать такие задачи в виде структур данных, обрабатываемых этой процедурой.
   9. **Распределённая** - язык, содержащий специальные конструкции для поддержки распараллеливания программы на несколько компьютеров.
2. Типизация
   1. **Статическая типизация** - переменные и параметры методов/функций связываются с типами в момент объявления и не могут быть изменены позже.
   2. **Динамическая типизация** - переменные и параметры методов/функций связываются с типами в момент присваивания значения (или передачи параметра в метод/функцию), а не в момент объявления переменной или параметра. Одна и та же переменная в разные моменты может хранить значения разных типов.
   3. **Явная типизация** - типы переменных и параметров указываются явно.
   4. **Неявная типизация** - типы переменных и параметров не указываются явно. Неявная типизация может быть и статической, в таком случае типы переменных и параметров вычисляются компилятором.
   5. **Явное приведение типов** - для использования переменной какого-то типа там, где предполагается использование переменной другого типа, нужно (возможно) явно выполнить преобразование типа.
   6. **Неявное приведение типов без потери данных** - неявное приведение типов в таких ситуациях, где не происходит потери данных — например, использование целого числа там, где предполагалось использование числа с плавающей точкой.
   7. **Неявное приведение типов с потерей данных** - неявное приведение типов в таких ситуациях, где может произойти потеря данных — например, использование числа с плавающей точкой там, где предполагалось использование целого числа.
   8. **Неявное приведение типов в неоднозначных ситуациях** - например, использование строки там, где предполагалось число или наоборот. Классический пример: сложить число 1 со строкой «2» — результат может быть как число 3, так и строка «12». Другой пример — использование целого числа там, где ожидается логическое значение (boolean).
   9. **Алиасы типов** - возможность определить видимый глобально (за пределами единицы компиляции) алиас типа, полностью эквивалентный исходному типу. Например, typedef в Си. Директива using в C# не подходит под этот критерий из-за локальной области действия.
   10. **Вывод типов переменных из инициализатора** - возможность не указывать явно тип переменной, если для неё задан инициализатор. Если возможность действует для локальных переменных, но не действует для полей класса, все равно ставьте +. Характеристика не применима к языкам с динамической типизацией.
   11. **Вывод типов переменных из использования** - возможность не указывать явно тип переменной, если её тип может быть выведен из дальнейшего использования. Если возможность действует для локальных переменных, но не действует для полей класса, все равно ставьте +. Характеристика не применима к языкам с динамической типизацией.
   12. **Вывод типов-аргументов при вызове метода** - возможность не указывать явно типы-аргументы при вызове generic-метода, если они могут быть выведены из типов обычных аргументов.
   13. **Вывод сигнатуры для локальных функций** - может ли сигнатура локальной функции быть выведена из использования. Неприменимо для языков с динамической типизацией. Ставьте -, если язык не поддерживает локальных функций.
   14. **Параметрический полиморфизм** - [наличие типобезопасного параметрического полиморфизма (aka generic types). Подразумевает возможность указывать constraints или type classes для типов-параметров.](http://en.wikipedia.org/wiki/Type_classes)
   15. **Параметрический полиморфизм с ковариантностью** - наличие ко- и контравариантных type parameters. В некоторых языках может быть лишь частичная поддержка (например, только в интерфейсах и делегатах). В таком случае, отмечайте +/-.
   16. **Параметрический полиморфизм высших порядков** - [Возможность создавать type constructors высших порядков (как в Scala). См. Towards Equal Rights for Higher-kinded Types](http://www.cs.kuleuven.be/~adriaan/files/higher.pdf)
   17. **Информация о типах в runtime** - возможность узнать точный тип объекта в runtime.
   18. **Информация о типах-параметрах в runtime** - возможность узнать в runtime информацию о типе, с которым инстанциирован generic-тип. Если язык не поддерживает generic-типы, то ставьте -. Если информация о типах стирается в runtime (используется erasure), то ставьте -.
3. Компилятор/интерпретатор
   1. **Open-source компилятор (интерпретатор)** - наличие полноценного open-source компилятора (для интерпретируемых языков — интерпретатора). Если существует open-source компилятор, но он поддерживает не все возможности языка, то ставьте +/- или -/+.
   2. **Возможность компиляции** - возможность компиляции в нативный код или в byte-код с возможностью JIT-компиляции. Если язык компилируется в код на другом языке (например, C), который потом компилируется в нативный код, то тоже ставьте +.
   3. **Bootstrapping** - наличие полноценного bootstrapping-компилятора (то есть компилятора, написанного на том же языке, который он компилирует, и успешно компилирующего самого себя). Если существует bootstrapping-компилятор, но он поддерживает не все возможности языка, то ставьте +/- или -/+.
   4. **Многопоточная компиляция** - возможность компилятора на многопроцессорных системах использовать несколько потоков для ускорения компиляции. Если язык не поддерживает компиляцию, то ставьте x (неприменимо).
   5. **Интерпретатор командной строки** - возможность вводить инструкции языка строка за строкой с их немедленным выполнением. Может использоваться в качестве калькулятора.
   6. **Условная компиляция** - возможность включать/выключать части кода в зависимости от значения символов условной компиляции (например, с помощью #if … #endif в C++)
   7. **Раздельная трансляция -**  означает то, что программа разбивается на части — физические модули или единицы компиляции. Каждая единица может или обязана транслироваться отдельно от остальных.
4. Управление памятью
   1. **Объекты на стеке** - возможность создавать экземпляры объектов не в куче, а на стеке.
   2. **Неуправляемые указатели** - наличие неуправляемых указателей, адресная арифметика, прямой доступ к памяти.
   3. **Ручное управление памятью** - возможность явного выделения и освобождения памяти в куче (например, с помощью операторов new и delete в C++).
   4. **Сборка мусора** - возможность использовать автоматический процесс сборки мусора (освобождения памяти в куче, занятой неиспользуемыми объектами).
   5. **Строгие ссылки и указатели** - когда в языке они могут использоваться только для адресации анонимных объектов в динамической памяти, и у них нету адресной арифметики.
5. Управление потоком вычислений
   1. **Инструкция goto** - [поддержка инструкции goto (безусловный переход на метку).](http://ru.wikipedia.org/wiki/GOTO)
   2. **Инструкция break без метки** - поддержка инструкции break без метки (безусловный выход из ближайшего цикла), и соответствующей инструкции continue. Наличие в языке инструкции break, относящегося к switch или другой конструкции, не влияет на это поле.
   3. **Инструкция break с меткой** - поддержка инструкции break с меткой (безусловный выход из цикла, помеченного меткой), и соответствующей инструкции continue. Наличие в языке инструкции break, относящегося к switch или другой конструкции, не влияет на это поле.
   4. **Поддержка try/catch** - поддержка обработки исключений с помощью try/catch или эквивалентной конструкции.
   5. **Блок finally** - поддержка блока finally при обработке исключений или эквивалентной конструкции.
   6. **Блок else (исключения)** - поддержка блока else при обработке исключений (действия, выполняющиеся при завершении блока try без исключения).
   7. **Перезапуски** - исключения, не раскручивающие стек вызовов. Возможность из места перехвата исключения вернуться в место установки перезапуска.
   8. **Легковесные процессы** - эмуляция многопоточности рантаймом самого языка. В пределах одного (или нескольких) потока ОС выполняется множество потоков исходного кода
6. Типы и структуры данных
   1. **Многомерные массивы** - наличие встроенных в язык многомерных массивов. Если язык поддерживает только массивы массивов, ставьте +/-
   2. **Динамические массивы** - наличие встроенных в язык динамических массивов (способных изменять свой размер во время выполнения программы). Если динамические массивы представлены только векторами (то есть только одномерными массивами) или векторами векторов, ставьте +/-
   3. **Ассоциативные массивы** - наличие встроенных в язык ассоциативных массивов или хэш-таблиц.
   4. **Цикл foreach** - наличие возможности перебрать все элементы коллекции с помощью цикла foreach. Если в языке есть эквивалентная или более сильная возможность (наподобие list comprehensions), ставьте +.
   5. **List comprehensions** - наличие List comprehensions (или аналога).
   6. **Кортежи** - возможность вернуть из функции/метода кортеж (tuple) — неименованный тип данных, содержащий несколько безымянных полей произвольного типа.
   7. **Целые числа произвольной длины** - поддержка целых чисел неограниченной разрядности. Должна быть возможность записать сколь угодно большое целое число с помощью литерала.
   8. **Целые числа с контролем границ** - возможность определить тип, значениями которого могут быть целые числа только определенного интервала, например [-5..27], при этом присвоение переменной такого типа значения, выходящего за указанные рамки, должно вызывать ошибку.
7. Объектно-ориентированные возможности
   1. **Интерфейсы** - семантическая и синтаксическая конструкция в коде программы, используемая для специфицирования услуг, предоставляемых классом. См. Интерфейс в ООП.
   2. **Множественное наследование** - возможность наследовать класс сразу от нескольких классов (не интерфейсов).
   3. **Мультиметоды** - динамическая (run time) диспетчеризация функции в зависимости от типов нескольких аргументов. В языках с «message passing» ООП похожая функциональность реализуется паттерном «Visitor».
   4. **Переименование членов при наследовании** - возможность в наследнике изменить имя поля/метода предка.
   5. **Решение конфликта имен при множественном наследовании** - при множественном наследовании — решение для случая ромбовидного наследования (B потомок A, C потомок A, D потомок B и C). Решение может приниматься как для всего класса, так и для каждого поля/метода в отдельности.
8. Функциональные возможности
   1. **First class functions** - функции в данном языке являются объектами первого класса.
   2. **Лексические замыкания** - возможность использовать локальную или лямбда-функцию (анонимный делегат) за пределами функции-контейнера с автоматическим сохранением контекста (локальных переменных) функции-контейнера
   3. **Частичное применение** - возможность фиксировать часть аргументов функции, то есть имея функцию f:(AxB) -> C; P(f, a): B -> C, где (P(f,a))(b) = f(a,b). Не следует путать с каррированием (оператор каррирования — один из вариантов реализации частичного применения).
9. Разное
   1. **Макросы** - наличие в языке макро-системы, обрабатывающей код программы до времени её компиляции и/или выполнения. Например, макросы Лиспа, препроцессор Си или шаблоны С++.
   2. **Шаблоны/Generics** - наличие в данном статически типизированном языке инструмента для обобщенного программирования, наподобие templates в C++ или generics в C#.
   3. **Поддержка Unicode в идентификаторах** - возможность включения Unicode-символов (например, букв национальных алфавитов) в идентификаторы.
   4. **Перегрузка функций** - возможность перегрузки функций/методов по количеству и типам параметров.
   5. **Перекрытие имён** - это перекрытие имён по областям видимости.
   6. **Замещение (override)** - это замещение функцией виртуальной функции базового класса
   7. **Динамические переменные** - возможность создавать переменные, имеющие динамическую область видимости (англ.).
   8. **Именованные параметры** - возможность при вызове функции/метода указывать имена
   9. **Значения параметров по умолчанию** - возможность при вызове функции/метода опускать некоторые параметры, чтобы при этом подставлялось значение по умолчанию, указанное при определении функции.
   10. **Локальные функции** - возможность определять локальную функцию внутри другой функции/метода. Подразумевается возможность использовать внутри локальной функции локальные переменные из внешнего блока.
   11. **Сопоставление с образцом** - наличие сопоставления с образцом.
   12. **Контрактное программирование** - возможность задавать пред- и пост-условия для методов и инварианты для классов. Подробнее см. Design by contract (англ.).
   13. **Скрытый тип данных** - когда есть доступ к имени типа, но нету доступа к его члену (Как абстрактный тип данных (как класс, с приватными членами))
   14. **Семантика возобновления -** после обработки исключения управление может вернуться непосредственно в точку, где возникло исключение (В языке с++ сделано иначе, но можно смоделировать)
   15. **Абстрактная функция** (в с++ это чисто виртуальная функция (без тела))
   16. **Generic в аде** тоже что и  **шаблон** то же что и **статический полиморфизм**
   17. **Родовая функция, класс (generic (кроме ада))** - если всё унаследовано от Obect, то можно хранить Object\* и с ней работать, в то время, как сам указатель указывает на что угодно в языке, и поэтому это как бы налог шаблонов, но реализованный через **динамический полиморфизм**.
   18. **Явная реализация интерфейса** означает, что вызов метода интерфейса может происходить только через ссылку на интерфейс, но не 25 может происходить через ссылку на класс, реализующий интерфейс. Перед вызовом интерфейсного метода необходимо явно преобразовать ссылку на объект реализующего класса к ссылке на интерфейс. Концепция явной реализации полезна, например, при конфликте имен между унаследованными интерфейсами (В ответах встречалось в c#)
   19. **Сопрограмма и подпрограмма** - При вызове сопрограммы управление передается в точку, непосредственно следующую за местом, где оно покинуло сопрограмму. В подпрограммах управление всегда начинается с первого оператора, и это не зависит от того, в какой точке управление покинуло подпрограмму в прошлый раз.
   20. **sealed (c#), final (java)** - нельзя замещать виртуальные методы в производных классах, и нельзя наследоваться от класса, если перед ним sealed.
   21. **Абстрактный тип данных (АТД)** — это тип с полностью инкапсулированной структурой. Использовать объекты АТД возможно только при помощи явно определенных в интерфейсе типа операций.
   22. **Охрана типа** - (where в c# или with в ada или with Oberon-2) - добавление ограничений на шаблонный параметр (например, что тип должен иметь конкретный метод) (generic \n type T is private; \n with function “+”(x,y:T) return T (<>); ...) (В обероне - "WITH v: T1 DO S1 | v: T2 DO S2 ELSE S3 END" - в зависимости от типа объекта v будет выполнен один из опеаторов S1, S2, S3, в обероне называется "страж типа")
   23. **Перечислимый тип данных** - ко всем можно применить сравнение
       1. **Паскаль** - Каждый тип отображается на некоторое число из int, Доступ к именам - глобальный, а не через название типа. Неявные преобразования в целый тип и обратно запрещены. (используются ord и values)
       2. **C#** - Тип может отображаться опционально на byte, short, int, long и все unsigned. + Для доступа обязательно нужно указать имя Перечислимого типа (Color.Red)
       3. **c++** - Недостатки - константы перечислимого типа имеют ту же область действия что и сам enum (как в паскале) ("**Неявный импорт имён**"). Фиксированное педставление - всегда реализуется на основе int (как в Паскале)
       4. **Java** - перечислимый тип - это просто класс (внешне похож на c#). Неявные преобразования в целый тип и обратно запрещены. (используются ordinal и values)
   24. **Упаковкой (boxing) -** называется процесс преобразования значения простого типа значения в экземпляр соответствующего класса-оболочки.

**Распаковкой (unboxing) -** называется, очевидно, процесс преобразования экземпляра класса-оболочки в значение соответствующего простого типа. (C# и java (c 5.0), немного JavaScript - там число - это объект, унаследованный от типа Number)

* 1. **static\_cast (c++) (впринципе почти во всех языка есть приведения как в си) -** преобразует выражения одного статического типа в объекты и значения другого статического типа. Поддерживается преобразование численных типов, указателей и ссылок по иерархии наследования как вверх, так и вниз. Проверка производится на уровне компиляции, так что в случае ошибки сообщение будет получено в момент сборки приложения или библиотеки.
  2. **dynamic\_cast ("as" в c# (приводит тип) или "is" в c# (проверяет можно ли привести тип, т.е. как typeid в с++))** (в delphi аналогично c#) (в java - getClass и instanceof) (в ада in - аналог других is) - Используется для динамического приведения типов во время выполнения. В случае неправильного приведения типов для ссылок вызывается исключительная ситуация std::bad\_cast, а для указателей будет возвращен 0. Использует систему RTTI (Runtime Type Information). Безопасное приведение типов по иерархии наследования, в том числе для виртуального наследования.
  3. **const\_cast -** Снимает qualifiers — const и volatile (антоним квалификатора register)
  4. **reinterpret\_cast -** Приведение типов без проверки. reinterpret\_cast — непосредственное указание компилятору. Применяется для приведения указателя к указателю, указателя к целому и наоборот.
  5. **Частичная специализация шаблона** - возможность "перегрузки" шаблонов, т.е. возможность создать например несколько функций или классов, в которых будет например <value1, value2> и <int, value2>, и тогда при вызове функции (создании класса), будет выбран тот, который более специфичный.
  6. **Явная реализация интерфейса** (Есть **только в c#**) - когда мы унаследовали интерфейсы, а потом, что бы переопределить какой-то метод, нужно сделать преобразование к базовому типу, и для вызова тоже нужно. (пример переопределения метода - void ISomeInterface.F() {}) (пример вызова - ((ISomeInterface)c).F(); (и только так, если была явная реализация интерфейса))

Метод остаётся виртуальным

* 1. **Неявная реализация интерфейса** - когда всё прозрачно, и не нужно заморачиваться с Явной реализацией интерфейса. Метод остаётся виртуальным
  2. **Динамическое связывание подпрограмм** - определение выполняемой подпрограммы на ходу (например это наблюдается во всех языках, где есть virtual, кроме Оберона, в котором нету полиморфизма, но там можно использовать указатель на функцию, которому присваивать нужные функции (фактически, это когда мы сами вручную управляем таблицей виртуальных функций))
  3. **Операция индексирования** - операция, возвращающая ссылку на некоторый объект. Может быть переопределена как operator [] в с++ и c# (а может и ещё где-то)

**Сноски**

1. Императивный/Haskell. Монады позволяют выполнять императивные действия.
2. ***1*** ***2*** ООП/Erlang. Можно провести параллель между процессами в Эрланге и объектами в определении Алана Кея [1].
3. ООП/Javascript. Прототипная модель ООП.
4. [ООП/Haskell. Классы типов и семейства типов перекрывают возможности ООП.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-3)
5. Логический/Haskell. Изначально инструментов для логического программирования не встроено, но есть сторонние библиотеки. Существует академический функционально-логический язык Curry, берущий Haskell за основу.
6. [Логический/Common Lisp. Логическая парадигма изначально в язык не встроена, но реализуется средствами языка.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-5)
7. LINQ
8. В языке существует множество декларативных конструкций, и, более того, возможность создавать свои, с помощью макросов.
9. [Декларативный/Perl. Только регулярные выражения.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-8)
10. Распределённый/Ada. См. Annex E. Distributed Systems.
11. ***1*** ***2*** Распределённый/C и C++. Многие распространённые компиляторы поддерживают директивы для распараллеливания в рамках технологий MPI и OpenMP.
12. Распределённый/C#. Существуют проекты распределённых модификаций языка, например Parallel C#.
13. [Распределённый/Haskell. Модель языка подразумевает распределённое использование, при этом не требуя от программиста усилий на реализацию распределённости. Один из поддерживающих эту возможность компиляторов — Glasgow Distributed Haskell.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-12)
14. ***1*** ***2*** ANSI стандарт языка предусматривает опциональные декларации типов, которые какие-либо конкретные реализации могут использовать по своему усмотрению. Большинство современных реализаций CL принимают декларации типов в расчет, и используют для статической проверки типов и в целях оптимизации.
15. [Статическая типизация/Perl. С версии 5.6. Только для не встроенных типов.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-14)
16. Статическая типизация/Smalltalk. Возможность статической типизации есть в диалекте Smalltalk — Strongtalk'е.
17. [Динамическая типизация/C#. Посредством специального псевдо-типа dynamic с версии 4.0.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-16)
18. [Динамическая типизация/F#. Компилятор поддерживает синтаксический сахар в виде преобразования использования оператора (?) xml?name в вызов xml.op\_Dynamic("name"), на базе чего может быть реализована имитация динамической типизации.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-17)
19. [Динамическая типизация/Haskell. Обеспечивается модулем Data.Dynamic.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-18)
20. [Динамическая типизация/VB.NET. Контролируемо с помощью Option Strict.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-19)
21. [Динамическая типизация/Delphi. Посредством специального типа Variant.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-20)
22. Явная типизация/Erlang. Можно использовать т. н. type test BIFs. См. [2]
23. [Явная типизация/Perl. См. Prototypes в man perlsub.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-22)
24. [Явная типизация/Python. Частично в Python 3.0.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-23)
25. Явная типизация/Smalltalk. Есть в Strongtalk.
26. [var, dynamic etc.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-25)
27. Неявное приведение типов/Ada. См. 4.6 Type Conversions.
28. [Неявное приведение с потерей данных/Perl. При сложении строки с числом: $a = '5aa'; print $a + 0; Напечатает: 5](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-27)
29. [Неявное приведение в неоднозначных ситуациях/Perl. Не совсем корректно, так как в Perl эти ситуации однозначны: 1 + "2" # 3  и  1 . "2" # "12"](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-28)
30. Макрос DEFTYPE
31. ***1*** ***2*** ***3*** ***4*** Вывод типов/Common Lisp. Некоторые компиляторы Common Lisp, такие как SBCL, поддерживают частичный вывод типов.
32. [Параметрический полиморфизм с ковариантностью/C#. Доступно начиная с C# 4.0 для типов интерфейсов и делегатов.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-31)
33. Информация о типах в runtime/Ada. Точный тип узнать можно (Ada.Tags), но полной поддержки отражения в языке нет. Можно узнать имя, предков, интерфейсы, сериализовать объект, но нельзя запросить список методов.
34. Информация о типах в runtime/С++. Можно сравнить типы на точное совпадение, узнать имя типа (typeid), приводить типы вниз по иерархии наследования.
35. [См. встроенную функцию ref и метод isa](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-34)
36. [Open-source компилятор (интерпретатор)/Smalltalk. В любом диалекте Smalltalk исходники всего, кроме виртуальной машины, (то есть библиотека классов, компилятор в байткод, среда разработки, сторонние библиотеки и пр.) принципиально открыты — это свойство языка. Из основных диалектов исходники виртуальной машины открыты у GNU Smalltalk, Squeak и Strongtalk.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-smalltalk-open-source_35-0)
37. [Open-source компилятор (интерпретатор)/Delphi. FreePascal и Lazarus.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-delphi-open-source_36-0)
38. Возможность компиляции/Erlang. HiPE — High Performance Erlang. Доступен только для \*nix-систем.
39. Существуют PHP-компиляторы, вполне корректно комилирующие любые PHP-скрипты. Например, Roadsend PHP Compiler.
40. [Возможность компиляции/Smalltalk. Стандартная реализация в Smalltalk — это прозрачная компиляция в байт-код (в момент сохранения изменённого исходного кода) с последующим исполнением на виртуальной машине, часто с использованием JIT-компилятора. Однако некоторые диалекты поддерживают прямую компиляцию в машинные коды. В частности, к таким диалектам относятся Smalltalk MT и Smalltalk/X.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-smalltalk-compiling_39-0)
41. [Bootstrapping-компилятор/Java. Java Compiler API появилось в версии 6.0.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-java-compiler-api_40-0)
42. Narcissus.
43. Например, SBCL
44. Bootstrapping-компилятор/Python. Проект PyPy.
45. [Bootstrapping-компилятор/Smalltalk. Компилятор в байт-коды изначально написан на самом Smalltalk и исполняется внутри виртуальной машины. Кроме этого также есть примеры виртуальных машин Smalltalk, написанных на самом Smalltalk — к ним, в частности, относится виртуальная машина Squeak, написанная на подмножестве Smalltalk, которое потом транслируется в C и компилируется в машинные коды. При этом собственно разработка и отладка виртуальной машины Squeak осуществляется внутри работающей системы Squeak.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-smalltalk-bootstrapping-compiler_44-0)
46. Интерпретатор командной строки/Ada. Business Shell (BUSH).
47. ***1*** ***2*** Интерпретатор командной строки/C++. C++ интерпретатор CINT.
48. [Планируется к версии 5.0 языка.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-csharp_interpreter_47-0)
49. Rhino Shell.
50. [В диалекте GNU Smalltalk реализована поддержка командной строки.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-smalltalk-command-line_49-0)
51. Условная компиляция/Ada. Поскольку использование препроцессора существенно осложняет работу утилит, отличных от компилятора, работающих с исходными текстами, в стандарт эта возможность не входит. Здесь: Conditional Compilation описывается, как можно организовать условно компилируемый код. В качестве резервного варианта предоставляется препроцессор gnatprep.
52. [Условная компиляция/Java. Утверждения (операторы assert) всегда включаются компилятором в байт-код и могут быть разрешены (по умолчанию запрещены, то есть игнорируются) при запуске виртуальной машины ключом -ea/-enableassertion.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-java-assert_51-0)
53. [3].
54. Макросы лиспа позволяют при компиляции вычислять произвольные выражения, включая, естественно, конструкции ветвлений. Кроме того, имеется также примерный аналог #ifdef из Си.[4][5]
55. [Компилятор должен решать, какие классы будут представлены «простыми» типами и будут, в том числе, размещаться в стеке.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-eiffel_stack_54-0)
56. [Создание объектов на стеке/Haskell. В GHC при помощи Unboxed Types / Unboxed Arrays.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-haskell-stack-objects_55-0)
57. Стандарт языка предусматривает декларацию DYNAMIC-EXTENT, которая может трактоваться компилятором как указание выделить место под объект на стеке.
58. [Создание объектов на стеке/Delphi. В Delphi имеется 2 объектных модели — старая (унаследована из Turbo Pascal) и новая. Создание объектов на стеке возможно только в старой объектной модели.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-delphi-stack-objects_57-0)
59. ***1*** ***2*** ***3*** ***4*** ***5*** ***6*** ***7*** Через FFI (foreign function interface)
60. [Можно с помощью модуля стандартной библиотеки — ctypes.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-python-pointer_59-0)
61. [Неуправляемые указатели/Smalltalk. В Smalltalk есть возможность низкоуровневой работы с памятью, но только в адресном пространстве, предоставляемом виртуальной машиной.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-smalltalk-unmanaged-pointers_60-0)
62. [unsafe + System.Runtime.InteropServices](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-61)
63. [Ручное управление памятью/Smalltalk. При низкоуровневой работе в пространстве памяти, предоставляемом виртуальной машиной, можно вручную создавать и удалять объекты, записывая данные в соответствующие адреса памяти. Аналогично можно вручную управлять размещением объектов в памяти.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-smalltalk-manual-memory-management_62-0)
64. Сборка мусора/Ada. Только на некоторых платформах (.NET и JVM) или при помощи библиотек (AdaCL:GC). Тем не менее, практически все программы на Ada могут работать как с ним, так и без него. В этом смысле к сборке мусора применительно к Аде следует относиться не как к инженерному решению, а как к оптимизации управления памятью.
65. ***1*** ***2*** Сборка мусора/C и C++. В стандарте языка и в стандартных библиотеках нет сборки мусора. Однако существуют сборщики мусора для C и C++ в виде библиотек. Например, BoehmGC (англоязычный раздел).
66. [Сборка мусора/Delphi. Если не считать Delphi.NET.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-delphi-garbage-collector_65-0)
67. [Инструкция goto/Java. Является зарезервированным словом.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-java-goto_66-0)
68. Специальный оператор GO. Все конструкции циклов в CL, фактически, являются макросами-надстройками над этой инструкцией.
69. Инструкция goto/Ruby. В языке goto нет, но есть библиотека реализующая его.
70. Инструкция goto/Smalltalk. В стандарте языка goto нет, но существуют библиотеки, реализующие функциональность goto через управление стеком исполнения. Используются крайне редко, это скорее proof of concept (англ.).
71. Макрос RETURN. Фактически, является частным случаем RETURN-FROM.
72. заменяется исключениями, также реализуется с помощью Camlp4 http://code.google.com/p/ocaml-break-continue/
73. Специальный оператор RETURN-FROM
74. [Есть возможность указать число вложенных циклов, которые нужно прервать](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-73)
75. [Можно либо повторить выполнение метода, либо пробросить исключение далее](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-eiffel-try-cacth_74-0)
76. [Java-style try-catch блок реализуется макросом handler-case. Кроме того, в возможности системы обработки исключений Common Lisp входит система т. н. перезапусков(restarts), которые позволяют обрабатывать исключения «изнутри» без раскрутки стека вызовов функций](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-Lisp-conditions_75-0)
77. [При помощи оператора eval](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-perl-try-cacth_76-0)
78. ***1*** ***2*** ***3*** При использовании библиотеки PBOSL
79. Блок finally/Ada. В стандарте языка finally нет, но существуют библиотеки, реализующие функциональность finally. Используются крайне редко, это скорее proof of concept (англ.).
80. MDN — MDC
81. Специальный оператор UNWIND-PROTECT
82. реализуется на camlp4 http://bluestorm.info/camlp4/dev/try/pa\_tryfinally.ml.html
83. [При помощи нескольких последовательных catch](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-java-try-cacth-catch_82-0)
84. [Java-style try-catch блок реализуется макросом handler-case. Кроме того, в возможности системы обработки исключений Common Lisp входит система т. н. перезапусков(restarts), которые позволяют обрабатывать исключения «сверху» без раскрутки стека вызовов функций](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-autogenerated3_83-0)
85. [При помощи eval or {…}](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-perl-try-cacth-or_84-0)
86. Частично реализуются нестандартным модулем Runops::Resume
87. Конструкции yield return, запросы LINQ, в FCL 4.0 войдёт тип Lazy.
88. [Seq-генераторы, модуль Lazy стандартной библиотеки F#.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-fsharp-lazy_87-0)
89. [Однако, данную возможность можно реализовать на макросах](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-88)
90. [Данная возможность реализована на макросах](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-89)
91. [Конструкции Linq.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-.D0.9A.D0.BE.D0.BD.D1.81.D1.82.D1.80.D1.83.D0.BA.D1.86.D0.B8.D0.B8_Linq_90-0)
92. [модуль Lazy стандартной библиотеки Ocaml.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-ocaml-lazy_91-0)
93. setcontext et al. (UNIX System V and GNU libc)
94. Реазилуется сторонними библиотеками, например cl-cont
95. Только для байт-кода http://okmij.org/ftp/Computation/Continuations.html#caml-shift
96. [Легковесные процессы/Java. Вплоть до Java 1.1.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-java-green-threads_95-0)
97. [Только в некоторых реализациях.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-96)
98. Следует заметить что это не стандартные легкие процессы [6]
99. Легковесные процессы/Python. Используя Stackless Python.
100. [Монадические потоки выполнения, реализованы в библиотеке Lwt](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-ocaml-green-threads_99-0)
101. Кортежи/C++. Реализуются в стандартной библиотеке (появились в TR1 (англоязычный раздел), до этого в boost).
102. ***1*** ***2*** Кортежи/.NET. С помощью типов System.Tuple<T,...> введёных в FCL 4.0.
103. ***1*** ***2*** Алгебраические типы данных/Ada и Delphi. Через механизм вариантных записей.
104. ***1*** ***2*** ***3*** ***4*** ***5*** ***6*** ***7*** ***8*** В динамических языках механизм алгебраических типов данных не имеет смысла.
105. ***1*** ***2*** Массивы/Haskell. С помощью Data.Array.
106. [Динамические массивы/C. «Из коробки» данной возможности нет, однако похожий функциональность можно реализовать, используя функцию realloc.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-105)
107. [Динамические массивы/Java. С помощью java.util.Vector (в стандартной библиотеке).](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-106)
108. [map и hash\_map в стандартной библиотеке](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-107)
109. [Ассоциативные массивы/Java. С помощью java.util.HashMap (в стандартной библиотеке).](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-108)
110. [Ассоциативные массивы/Haskell. С помощью Data.Map](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-109)
111. [Контроль границы массивов/С++. Для массивов контроля нет, однако в STL входит шаблон std::vector в который может быть встроен контроль границ (в перегрузку оператора индексации).](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-110)
112. ***1*** ***2*** ***3*** Контроль границ массивов/Perl, PHP и JavaScript. В языке нет массивов со статическими границами, присваивание элементу за текущими границами массива просто расширяет границы массива.
113. [Контроль границ массивов/Ocaml. Можно отключить на этапе компиляции с помощью ключа -unsafe](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-112)
114. [Цикл foreach/Ada. Методы Iterate и Reverse\_Iterate различных контейнеров, входящих в библиотеку Ada.Containers.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-ada-foreach_113-0)
115. [Цикл foreach/C++. Алгоритм for\_each из библиотеки STL.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-.D0.A1.2B.2B-foreach_114-0)
116. [Цикл foreach/Erlang. В виде функции foreach/3 из модуля lists.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-erlang-foreach_115-0)
117. Цикл foreach/JavaScript. С версии 1.6[7].
118. Цикл foreach/Lisp. Макрос LOOP в составе стандартной библиотеки. Представляет собой «язык в языке» с большим количеством возможностей.
119. [List comprehensions/C#. «Query Comprehension» можно считать за List Comprehension только с большой натяжкой.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-118)
120. [LOOP et al.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-119)
121. ***1*** ***2*** ***3*** Целые числа произвольной длины/.NET. Посредством типа System.Numerics.BigInteger, включенного в FCL версии 4.0.
122. [Целые числа произвольной длины/Java. С помощью классов BigInteger и BigDecimal.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-121)
123. [Целые числа произвольной длины/Scala. С помощью классов BigInteger и BigDecimal.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-122)
124. [Целые числа произвольной длины/OCaml. В помощью модуля Num и Big\_int.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-123)
125. [Пример: Тип (INTEGER 0 9) включает в себя все цифры от 0 до 9](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-124)
126. [Целые числа произвольной длины/Perl. С помощью модуля Tie::Scalar.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-125)
127. Похожая функциональность реализуется макросами и средствами CLOS.
128. Через множественное наследование от классов с методами-заготовками. См. [8]
129. [Эмуляция через dynamic](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-128)
130. ***1*** ***2*** ***3*** ***4*** Реализуется сторонними библиотеками
131. [появятся(?) в Perl 6](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-130)
132. [Через множественное наследование и/или изменение атрибутов произвольного объекта во время выполнения](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-131)
133. Подмешивание реализации интерфейсов через ключевое слово implements. См. страницы 10-7 и 10-8 в Object Pascal Guide.
134. [Только совместное использование посредством виртуального наследования](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-cpp-dual-inherit_133-0)
135. [Для каждого члена класса — выбор дублирование (через переименование), или слияние (иначе, если не было переопределения)](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-eiffel-dual-inherit_134-0)
136. CLHS: Section 4.3.5
137. Functions — D Programming Language 2.0 — Digital Mars
138. [в форме указателей на функции](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-137)
139. [указатели на методы](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-138)
140. ***1*** ***2*** ***3*** Появились в Delphi2009, как анонимные функции.
141. [Анонимные делегаты присутствуют в языке с версии 2.0. В C# 3.0 появились полноценные анонимные функции.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-140)
142. Макрос LAMBDA
143. [Через анонимные классы](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-142)
144. [Начиная с версии 5.3](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-143)
145. [Только для функций двух переменных, через binder1st и binder2nd в стандартной библиотеке](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-144)
146. Реализуется сторонними библиотеками, например Sub::Curry и Sub::Curried
147. [functools.partial в стандартной библиотеке начиная с Python 2.5](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-146)
148. [Proc#curry, появился в Ruby 1.9](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-147)
149. ***1*** ***2*** Макросы/C. Посредством препроцессора C.
150. [Макросы/Haskell. Template Haskell — препроцессор, встроенный в GHC.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-149)
151. Фильтры [9], в т. ч., C/C++ препроцессор Filter::cpp
152. [Встроенны в Visual Studio (нет в Express Edition )](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-151)
153. [Штатный препроцессор camlp4](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-152)
154. ***1*** ***2*** ***3*** ***4*** ***5*** ***6*** ***7*** Неприменимо в языках с динамической типизацией.
155. [Generics/Haskell. Прямых аналогов шаблонов в языке нет, однако имеются не менее мощные средства обобщенного программирования.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-154)
156. [Generics/Delphi. Доступно начиная с Delphi 2009.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-155)
157. [Unicode в идентификаторах/C. Доступно в компиляторах gcc начиная с 4.2](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-156)
158. [Unicode в идентификаторах/C++. Доступно в компиляторах от MS, начиная с MSVS++ 2005 и в gcc начиная с 4.2](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-157)
159. [В большинстве современных реализаций](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-158)
160. [Unicode в идентификаторах/Python. Доступно начиная с Python 3.0.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-159)
161. [Unicode в идентификаторах/Ruby. Доступно начиная с Ruby 1.9.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-160)
162. [Перегрузка функций/JavaScript. Можно сымитировать, используя проверку передаваемых параметров с помощью рефлексии.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-161)
163. [Обобщенные функции можно перегружать по типам или значениям нескольких параметров](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-162)
164. Только перегрузка операторов [10].
165. ***1*** ***2*** Перегрузка функций и сопоставление с образцом/Python. Реализовано в сторонней библиотеке PEAK-rules.
166. implicit-parameters
167. макросы DEFVAR и DEFPARAMETER, а также декларация SPECIAL, создают динамические биндинги.
168. ***1*** ***2*** Именованные аргументы и параметры по умолчанию/C#. Доступно начиная с C# 4.0.
169. [Именованные параметры/JavaScript. Можно сымитировать, передав в качестве параметра функции объект: f ({param1: "value1", param2: "value2"}).](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-168)
170. [Спецификатор «&key» в списке аргументов объявляемой функции объявляет именованный параметр.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-169)
171. [Именованные параметры/Smalltalk. Можно называть методы в стиле сделатьЧтоНибудьС:используя:и: — в таком случае двоеточия обозначают места, куда будут подставляться параметры при вызове метода, например сделатьЧтоНибудьС: парам1 используя: парам2 и: парам3. Названия подбирают таким образом, чтобы при вызове было понятно, для чего будут использоваться параметры.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-170)
172. [Именованные параметры/Delphi: Могут использоваться при вызове OLE: Word.Openfile(filename='1.doc')](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-171)
173. [Значения параметров по умолчанию/Erlang. Можно сымитировать с помощью арности функции.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-172)
174. [«&key» и «&optional» параметры допускают значения по умолчанию](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-173)
175. Значения параметров по умолчанию/Perl. Можно элементарно сымитировать, см. [11].
176. Локальные функции/С. Поддерживаются в компиляторе gcc как нестандартное расширение языка, см. [12].
177. [Локальные функции/С++. Поддерживаются в виде методов локальных классов](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-176)
178. [Локальные функции/Java. Внутри метода можно определять безымянные (анонимные) локальные классы, которые фактически позволяют создавать экземпляры объектов, перекрывающие методы своего класса.](http://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%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#cite_ref-java-anonymous-class_177-0)
179. Cпециальный оператор LABELS
180. Макрос DESTRUCTURING-BIND и EQL спецификатор в обобщенных функциях можно рассматривать как аналоги некоторых подмножеств функциональности сопоставления с образцом.
181. ***1*** ***2*** ***3*** Посредством библиотеки Code Contracts из состава FCL 4.0.
182. Контрактное программирование/Java. На основе аннотаций Java 5, используя библиотеку OVal и аспектный компилятор AspectJ.
183. Контрактное программирование/Haskell. Посредством библиотеки QuickCheck.