**Программное обеспечение** (Software) - совокупность программ, выполняемых вычислительной системой.

Программное обеспечение — *неотъемлемая часть программно-аппаратного комплекса, поскольку сейчас даже в бытовой технике есть ПО системы*. Оно является управляющим элементом для технических средств, определяющим то, как они будут функционировать. Сфера применения конкретного компьютера определяется созданным для него ПО, однако, архитектура и аппаратная часть тоже важны, бортовой компьютер для автомобиля, самолета или космического корабля – это изделие, назначение которого уже заложено в аппаратной части, и сделать бытовой или бухгалтерский ПК из него либо не получится, либо очень сложно и дорого. Другое дело, когда речь идет о классе вычислительных систем, но и здесь скорее ПО устанавливается под задачи, а также важна аппаратура. Например, если компьютер используется для графических задач, то без специальных аппаратных ресурсов никакое ПО не поможет. Все должно быть в комплексе.

**Разработка ПО включает в себя следующие направления и виды работ:**

* технология проектирования программ (например, нисходящее проектирование, структурное и объектно-ориентированное проектирование и др.);
* методы тестирования программ [ссылка, ссылка];
* методы доказательства правильности программ;
* анализ качества работы программ;
* документирование программ;
* разработка и использование программных средств, облегчающих процесс проектирования программного обеспечения.

**Программа -** упорядоченная последовательности команд.Конечная цель любой компьютерной программы - управление аппаратными средствами.

Состав программного обеспечения ПК называют **программной конфигурацией**. Между программами, как и между физическими узлами и блоками, существует взаимосвязь - многие программы работают, базируясь на программы более низкого уровня. Возможность существования такого интерфейса обеспечивается распределением программного обеспечения на несколько взаимодействующих между собой уровней. **Уровни программного обеспечения** представляют собой иерархическую конструкцию. Каждый уровень опирается на программное обеспечение предшествующих уровней, показано на слайде.

Программное обеспечение можно классифицировать на системное и прикладное ПО. Показано на слайде

В свою очередь, системное программное обеспечение классифицируется на: операционные системы, системные интерфейсы, инструментальные средства, системы технического обслуживания

Итак перейдем к уровням ПО.

**Базовый уровень -** самый низкий уровень программного обеспечения*.* Он отвечает за взаимодействие с *базовыми аппаратными средствами.* Базовые программные средства входят в состав базового оборудования и хранятся в специальных микросхемах, называемых *постоянными запоминающими устройствами* (*ПЗУ).* Программы и данные записываются («прошиваются») в микросхемы ПЗУ на этапе производства и не могут быть изменены в процессе эксплуатации.

В тех случаях, когда изменение базовых программных средств во время эксплуатации является технически целесообразным (например, *BIOS*), вместо микросхем ПЗУ применяют *перепрограммируемые постоянные запоминающие устройства* (*ППЗУ).* В этом случае изменение содержания ПЗУ можно выполнять как непосредственно в составе ПК*,* так и вне её, на специальных устройствах, называемых *программаторами.*

* *ПЗУ отличаются наиболее высокой скоростью обмена;*
* *В ряде случаев, когда речь идет о компьютеризированных устройствах, например о бытовой технике, ПЗУ является единственной памятью в этих устройствах, и все необходимое ПО зашито в них;*

**Инструментальный** **уровень** (трансляторы и компиляторы языков программирования, системы программирования). Создание программ обеспечивают, текстовые редакторы, среды разработки. Трансляторы и компиляторы осуществляют преобразование созданных на языке высоко уровня программ в корректный набор машинных кодов.

**Системный уровень**. Переходный уровень. Программы, работающие на этом уровне, обеспечивают взаимодействие прочих программ компьютерной системы с программами базового уровня и непосредственно с аппаратным обеспечением.

От программного уровня этого уровня зависят эксплуатационные показатели всего компьютера в целом. Другой класс программ, системного уровня отвечает за взаимодействие с пользователем. Именно благодаря им мы получаем возможность вводить данные в компьютер, управлять его работой и получать результат в удобной для себя форме. Эти программные средства называют средствами обеспечения пользовательского интерфейса. От них напрямую зависит удобство работы с компьютером и производительность труда на рабочем месте.

**Служебный уровень**. ПО этого уровня взаимодействует как с программами базового уровня, так и с программами системного уровня. Основное назначение служебных программ (утилит) состоит в автоматизации работ по проверке, наладке и настройке имеющихся программно-аппаратных ресурсов. Во многих случаях они используются для расширения или улучшения функций системных программ. Некоторые служебные программы изначально включают в состав ОС.

**Прикладной уровень.** ПО прикладного уровня представляет собой комплекс прикладных программ, с помощью которых на данном рабочем месте выполняются конкретные задания. Спектр таких заданий очень широк – от производственных до творческих и развлекательно - обучающих. Огромный функциональный диапазон возможных приложений средств обусловлен наличием прикладных программ для разных видов деятельности. Именно прикладное ПО предназначено для решения задач, необходимых для конечного пользователя, или для решения задач на промежуточных этапах этого процесса».

**Задача** - проблема, подлежащая решению в интересах пользователя.

Термин "задача" в программировании означает единицу работы вычислительной системы, требующую выделения вычислительных ресурсов (процессорного времени, оперативной и внешней памяти, файлов и т.п.).

**Приложение** - программная реализация решения задачи на компьютере. Приложение может состоять из одной или нескольких взаимосвязанных и взаимодействующих программ. Условно принято делить программы на небольшие (простые), средней сложности и большие.

Программа считается небольшой как по размерам, так и по другим признакам, если она удовлетворяет следующим признакам:

* решает одну четко поставленную задачу в хорошо известных ограничениях, к тому же, не очень существенную для какой-либо практической или исследовательской деятельности;
* неважно, насколько быстро она работает;
* ущерб от неправильной работы программы - практически нулевой (за исключением возможности обрушения ею системы, в которой выполняются и другие, более важные задачи);
* не требуется дополнять программу новыми возможностями, практически никому не нужно разрабатывать ее новые версии или исправлять найденные ошибки;
* в связи со сказанным выше не очень нужно прилагать к программе подробную и понятную документацию - для человека, который ею заинтересуется, не составит большого труда понять, как ею пользоваться, просто по исходному коду.

Сложные программы, называемые также программными системами, программными комплексами, программными продуктами, отличаются от небольших не только по размерам, но и наличием дополнительных факторов. Эти факторы связаны с их востребованностью и готовностью пользователей платить деньги, как за приобретение самой программы, так и за ее сопровождение и даже за специальное обучение работе с ней.

Обычно сложная программа обладает следующими свойствами:

* программа решает одну или несколько связанных задач;
* существенно, чтобы программа была удобной в использовании. В частности, она должна включать достаточно полную и понятную пользователям документацию, возможно, также специальную документацию для администраторов, а также набор документов для обучения работе с программой;
* низкая производительность программы на реальных данных приводит к значимым потерям для пользователей;
* неправильная работа программы наносит ощутимый ущерб пользователям и другим организациям и лицам, даже если сбои происходят не слишком часто;
* для выполнения своих задач программа должна взаимодействовать с другими программами и программно-аппаратными системами и обеспечивать работу на разных платформах;
* пользователи, работающие с программой, приобретают дополнительные выгоды от того, что программа развивается, в нее вносятся новые функции и устраняются ошибки. Необходимо наличие проектной документации, позволяющей развивать ее, возможно, вовсе не тем разработчикам, которые ее создавали, без больших затрат на обратную разработку (реинжиниринг);
* в разработку программы вовлечено значительное количество людей (более 5-ти человек). Большую программу практически невозможно написать с первой попытки, с небольшими усилиями и в одиночку;
* большая программа имеет намного большее количество ее возможных пользователей по сравнению с небольшими программами, и еще больше тех лиц, деятельность которых будет так или иначе затронута ее работой и результатами.

Примером большой программы может служить стандартная библиотека классов Java или C#, соответствующих систем программирования.

Строго говоря, ни одно из указанных свойств не является обязательным для того, чтобы программу можно было считать большой, но при наличии двух-трех из них достаточно уверенно можно утверждать, что она большая. На основании некоторых из перечисленных свойств можно сделать вывод, что большая программа или программная система чаще всего представляет собой не просто код или исполняемый файл, а включает еще и набор проектной и пользовательской документации.

Процесс создания программ можно представить как последовательность следующих действий:

* 1. постановка задачи;
* 2. алгоритмизация решения задачи;
* 3. программирование.

**Постановка задачи** - это точная формулировка требований, предъявляемых к работе программы, с описанием входной и выходной информации, и, возможно, описание подходов к решению задачи.

**Алгоритм** - точный набор инструкций, описывающих однозначный порядок действий исполнителя (компьютера), от допустимых исходных данных для достижения результата решения задачи за конечное время.

**Программирование** (programming) - теоретическая и практическая деятельность, связанная с созданием программ.

**Информационная система (ИС)** — система, предназначена для хранения, поиска и обработки информации, и соответствующие организационные ресурсы, которые обеспечивают и распространяют информацию.

Предназначена для своевременного обеспечения потребителей надлежащей [информацией](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D1%8F), то есть для удовлетворения конкретных информационных потребностей в рамках определённой [предметной области](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D0%B4%D0%BC%D0%B5%D1%82%D0%BD%D0%B0%D1%8F_%D0%BE%D0%B1%D0%BB%D0%B0%D1%81%D1%82%D1%8C), при этом результатом функционирования информационных систем является информационная продукция — документы, информационные массивы, [базы данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) и информационные услуги.

Информационная система имеет цель – производство профессиональной информации, связанной с определенной профессиональной деятельностью. Их задача помочь в анализе проблем и создавать новые продукты.

Информационная система - взаимосвязанная совокупность средств, методов и персонала, используемых для хранения, обработки и выдачи информации в интересах достижения поставленной цели.

Следует выделить следующие свойства программного обеспечения:

* **Корректность**
* **Устойчивость**
* **Восстанавливаемость**
* **Надежность**

**Корректность** программного обеспечения — свойство безошибочной реализации требуемого алгоритма при отсутствии таких мешающих факторов, как ошибки входных данных, ошибки операторов, сбои и отказы ЭВМ.

Под **корректностью** понимаются свойства программы, свидетельствующие об отсутствии в ней ошибок, допущенных разработчиком на различных этапах проектирования (спецификации, проектирования алгоритма и структур данных, кодировании). Корректность самой программы понимают по отношению к целям, поставленным перед ее разработкой (т. е. это относительное свойство).

**Устойчивость** — свойство осуществлять требуемое преобразование информации при сохранении выходных решений программы в пределах допусков, установленных спецификацией. Устойчивость характеризует поведение программы при воздействии на нее таких факторов неустойчивости, как ошибки операторов ЭВМ, а также не выявленные ошибки программы.

**Восстанавливаемость** — свойство программного обеспечения, характеризующее возможность приспосабливаться к обнаружению ошибок и их устранению.

**Надежность** можно представить совокупностью следующих характеристик:

* целостностью программного средства (способностью его к защите от отказов);
* живучестью (способностью к входному контролю данных и их проверки в ходе работы);
* завершенностью (бездефектностью готового программного средства, характеристикой качества его тестирования);
* работоспособностью (способностью программного средства к восстановлению своих возможностей после сбоев).

Отличие понятия корректности и надежности программ состоит в следующем:

* надежность характеризует как программу, так и ее "окружение" (качество аппаратуры, квалификацию пользователя и т. п.);
* говоря о надежности программы, обычно допускают определенную, хотя и малую долю ошибок в ней и оценивают вероятность их появления.

Вернемся к понятию корректности. Очевидно, что не всякая синтаксически правильная программа является корректной, т. е. корректность характеризует семантические свойства программ.

С учетом специфики появления ошибок в программах можно выделить две стороны понятия корректности:

* корректность как точное соответствие целям разработки программы (которые отражены в спецификации) при условии ее завершения или частичная корректность;
* завершение программы, т. е. достижение программой в процессе ее выполнения своей конечной точки.

В зависимости от выполнения или невыполнения каждого из двух названных свойств программы различают шесть задач анализа корректности:

1) доказательство частичной корректности;

2) доказательство частичной некорректности;

3) доказательство завершения программы;

4) доказательство не завершения программы;

5) доказательство тотальной (полной) корректности (т. е. одновременное решение 1-й и 3- й задач);

6) доказательство некорректности (решение 2-й или 4-й задачи).

Характеристики программ показаны на слайде

**Мобильность** программных продуктов означает их независимость от технического комплекса системы обработки данных, операционной среды, сетевой технологии обработки данных, специфики предметной области и т.п. Мобильный программный продукт может быть установлен на различных моделях компьютеров и операционных систем, без ограничений на его эксплуатацию в условиях вычислительной сети. Функции обработки такого программного продукта пригодны для массового использования без каких-либо изменений.

**Надежность**работы программного продукта определяется бессбойностью и устойчивостью в работе программ, точностью выполнения предписанных функций обработки, возможностью диагностики возникающих в процессе работы программ ошибок.

**Эффективность** программного продукта оценивается как с позиций прямого его назначения - требований пользователя, так и с точки зрения расхода вычислительных ресурсов, необходимых для его эксплуатации.

**Учет человеческого фактора** означает обеспечение дружественного интерфейса для работы конечного пользователя, наличие контекстно-зависимой подсказки или обучающей системы в составе программного средства, хорошей документации для освоения и использования заложенных в программном средстве функциональных возможностей, анализ и диагностику возникших ошибок и др.

**Модифицируемость** программных продуктов означает способность к внесению изменений, например расширение функций обработки, переход на другую техническую базу обработки и т.п.

**Коммуникативность** программных продуктов основана на максимально возможной их интеграции с другими программами, обеспечении обмена данными в общих форматах представления (экспорт/импорт баз данных, внедрение или связывание объектов обработки и др.).

Программы любого вида характеризуются жизненным циклом, состоящим из нескольких этапов:

1. Проектирование структуры программы

2. Создание, тестирование и отладка программы

3. Распространение программы

4. Эксплуатация программы пользователями, послепродажная поддержка

5. Создание новой версии исходной программы или снятие программы с продажи или производства

В зависимости от условий распространения и использования можно выделить следующие группы ПО:

· **Собственническое (или проприетарное)**

· **Свободное ПО**

· **ПО с открытым кодом**

В программировании выделяют направления:

* Разработка web-приложений
* Разработка desktop-приложений
* Разработка серверных приложений
* Разработка мобильных приложений
* Программирование встраиваемых систем
* Системное программирование
* Разработка игр
* Олимпиадное программирование и решение задач
* Программирование для бухгалтерских и финансовых продуктов
* Программирование баз данных
* Science

**Разработка web-приложений-**Это направление ориентировано на разработку веб-приложений. Web-программирование можно разделить на backend (написание серверных скриптов – PHP, Python, Ruby) и frontend (разработка юзерского интерфейса – Javascript, HTML, CSS).

**Разработка desktop-приложений.** Разработка программного обеспечения для различных операционных систем. Все разнообразие софта, что мы используем в повседневности.

**Разработка серверных приложений.** Это различные игровые сервера (ваша любимая Дотка, CS: GO), IM-сервисы (серверная часть Skype, ICQ, MSN), банковские базы данных.

### Разработка мобильных приложений Множество Java-приложений. VK, Viber, Яндекс.Карты, переводчики, электронные читалки.

### Программирование встраиваемых систем Отрасль программирования для различной домашней техники: пылесосы, холодильники, стиральные машины, плееры, навигаторы, электронные весы. Здесь задействованы научные разработки с использованием специализированных языков, типа MATLAB.

### Системное программирование. Написание различных драйверов для оборудования, программирования «ядра» операционных систем. Кстати, создание компиляторов и интерпретаторов для языков программирования относятся сюда же.

### Разработка игр Гигантская отрасль. Сюда включается разработка игр и для ПК, и для консолей, и для мобильный устройств.

### Олимпиадное программирование и решение задач Программирование на различных «непрактичных» и не распространенных языках для решения каких-то оригинальных задач.

### Программирование для бухгалтерских и финансовых продуктов «1С: Предприятие». Вся бухгалтерия в России завязана на этом продукте. Но недостаточно знать лишь сам язык, важно понимать основы бухгалтерского учета. Плюс в том, что работы очень много, и без хлеба вы не останетесь.

### Программирование баз данных Серьезное направление. способность хранить миллиарды строк информации.

### Science Наука и этим все сказано. Нейронные сети, моделирование структуры ДНК, запуск спутников, моделирование Большого Взрыва.

*Язык программирования* - формальная знаковая система, предназначенная для описания алгоритмов. Язык программирования определяет набор лексических, синтаксических и семантических правил, используемых при составлении компьютерной программы. Он позволяет программисту точно определить то, на какие события будет реагировать компьютер, как будут храниться и передаваться данные, а также какие именно действия следует выполнять над этими данными при различных обстоятельствах.

Языки программирования подразделяются на *низкоуровневые* и *высокоуровневые* языки.

*Низкоуровневый язык программирования* - язык программирования, близкий к программированию непосредственно в машинных кодах.

Как правило, использует особенности конкретного семейства процессоров. Общеизвестный пример низкоуровнего языка - язык *ассемблера*.

*Высокоуровневый язык программирования* - язык программирования, разработанный для быстроты и удобства использования программистом. Язык предназначен для решения абстрактных высокоуровневых задач и оперирует не инструкциями к оборудованию, а логическими понятиями и абстракцией данных. Это позволяет быстрее программировать сложные задачи и обеспечивает относительную независимость от оборудования. Использование разнообразных трансляторов и интерпретаторов обеспечивает связь программ, написанных при помощи языков высокого уровня, с различными операционными системами и оборудованием, в то время как их исходный код остаётся, в большей части, неизменным.

Такого рода оторванность высокоуровневых языков от аппаратной реализации компьютера помимо множества плюсов имеет и минусы. В частности, она не позволяет создавать простые и точные инструкции к используемому оборудованию. Программы написанные на языках высокого уровня, проще для понимания программистом, но гораздо менее эффективны, чем их аналоги, создаваемые при помощи низкоуровневых языков. Одним из следствий этого стало добавление поддержки того или иного языка низкого уровня (язык ассемблера) в большинство современных профессиональных высокоуровневых языков программирования.

Наиболее распространёнными языками подобного типа являются *C++*, *Visual Basic*, *Java*, *Python*, *Ruby*, *Perl*, *Delphi*, *PHP*.

Языки программирования также можно разделить на *компилируемые* и *интерпретируемые*.

Программа на компилируемом языке при помощи специальной программы *компилятора* преобразуется (компилируется) в набор инструкций для данного типа процессора (машинный код) и далее записывается в исполняемый файл, который может быть запущен на выполнение как отдельная программа. Другими словами, компилятор переводит программу с языка высокого уровня на низкоуровневый язык, понятный процессору.

Если программа написана на интерпретируемом языке, то *интерпретатор* непосредственно выполняет (интерпретирует) её текст без предварительного перевода. При этом программа остаётся на исходном языке и не может быть запущена без интерпретатора. Можно сказать, что процессор компьютера - это интерпретатор машинного кода.

Кратко говоря, *компилятор* переводит программу на машинный язык сразу и целиком, создавая при этом отдельную программу, а *интерпретатор* переводит на машинный язык прямо во время исполнения программы.

Разделение на компилируемые и интерпретируемые языки является несколько условным.

Так, для любого традиционно компилируемого языка, как, например, *Java*, можно написать интерпретатор. Кроме того, большинство современных «чистых» интерпретаторов не исполняют конструкции языка непосредственно, а компилируют их в некоторое высокоуровневое промежуточное представление (например, с разыменованием переменных и раскрытием макросов).

Для любого интерпретируемого языка можно создать компилятор - например, язык *PHP*, изначально интерпретируемый, может компилироваться без каких бы то ни было ограничений. Создаваемый во время исполнения программы код может так же динамически компилироваться во время исполнения.

Рассмотрим подробнее наиболее популярные языки программирования.



C - Деннис Макалистэйр Ритчи, американский компьютерный специалист, занимался созданием языка программирования C в 1967 — 1973 годах в AT&T Bell Labs. Он является предшественником таких языков программирования как С++, Java, C#, JavaScript и Perl. По этой причине изучение этого языка ведет к понимаю и других языков.

С - компилируемый статически типизированный язык программирования общего назначения.

Язык C всё ещё весьма популярен. Язык С используется для разработки низкоуровневых приложений, так как считается ближе всего к аппаратному, т.е. на этом языке обычно программируются микросхемы.

Деннис Ритчи, кстати, создал ещё и всемирно известную операционную систему Unix. Сделал он это вместе со своим давним коллегой Кеном Томпсоном.



C++ - Бьёрн Страуструп родился 30 Декабря 1950 года в городе Орхус, в Дании. Он известен благодаря созданию и разработке широко используемого языка программирования C++.

C++, представляет собой язык нового поколения, следующего за языком C, который отличается от предшественника только поддержкой возможностей по объектно-ориентированному программированию.

C++ всё ещё остаётся одним из самых популярных языков и является одним из самых распространенных языков в мире.

Он, например, широко используется при разработке систем высокочастотного трейдинга из-за того, что код, написанный на C++, довольно-таки близок к системным возможностям компьютеров, и из-за того, что этот язык поддерживает популярные в наше время объектно-ориентированные возможности. Например, приложения такие, как: Google Chrome, Mozilla Firefox, Winamp и линейка продуктов Adobe были разработаны с помощью С++. Кроме того, некоторые современные игры и операционные системы были разработаны на С++ из-за быстрого процессинга и компиляции.

Преимущества С++

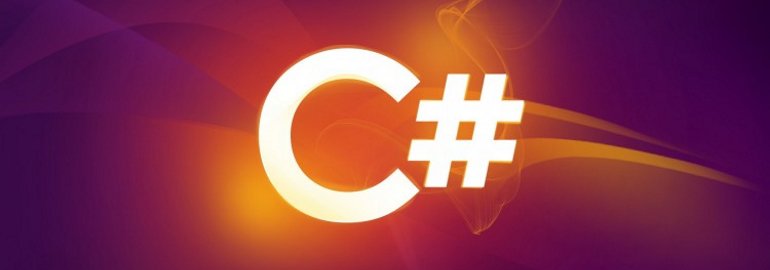
**Компилятор**. Множество компиляторов, консольные команды, этапы сборки программы.  
 Код программы – это просто текст, который сам по себе не заработает.

* Компилятор – это отдельная программа, которой надо указать, что и как сделать с исходным кодом, чтобы он превратился в исполняемый файл.
* Текстовый редактор – это тоже отдельная программа, предназначенная для написания исходного кода.
* Существуют опции сборки, и существует не один компилятор.
* Исходный код, написанный программистом, может быть предварительно обработан и изменён (например, препроцессором).

**Статическая типизация.** На примере языка со статической типизацией проще понять, что такое тип данных, зачем он нужен и от чего зависит. Видно, что собой представляет объявление, определение и инициализация. Использование языка C++ даёт это явно увидеть, что способствует дальнейшему пониманию того, как работают эти механизмы в других языках.

**Высокоуровневые и низкоуровневые средства.** Использование таких средств, как указатели и динамическое выделение памяти, позволяет понять, что такое стэк, куча, стэк вызовов, раскрутка стэка и т.д. Помимо этого, на практике закрепляется понимание концепции адресов и адресной арифметики. Необходимо выделять память, освобождать, потому что память не бесконечная, существуют утечки памяти.

**Реализация ООП.** Это относительно чистая реализация ООП. Чётко разграниченные уровни доступа к объектам класса, возможность множественного наследования и динамический полиморфизм дают возможность быстро усвоить основные концепции ООП (абстракция, наследование, инкапсуляция и полиморфизм).



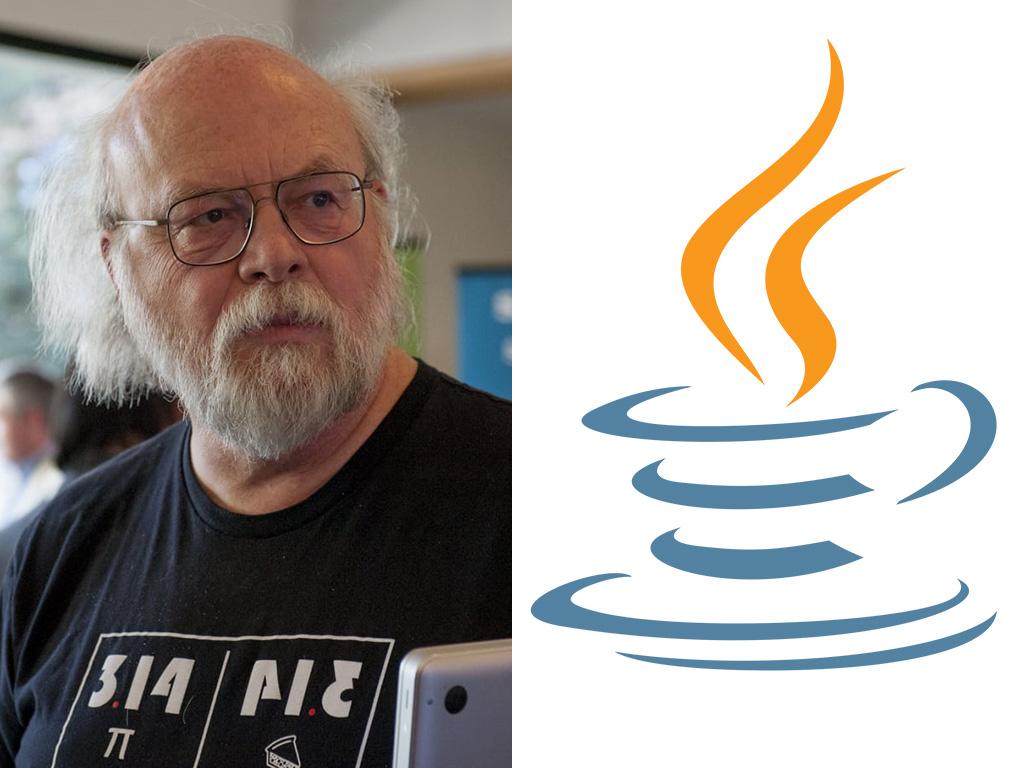
C# - Компилируемый Объектно-ориентированный язык программирования. Разработан в 1998—2001 годах группой инженеров под руководством Андерса Хейлсберга в компании Microsoft как язык разработки приложений для платформы Microsoft .NET Framework. C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок Java.

Преимущества языка:

* Язык программирования C# претендует на полную объектную ориентированность.
* Компонентно-ориентированный подход к программированию, способствующий меньшей машинно-архитектурной зависимости результирующего программного кода, гибкости, переносимости и легкости повторного использования (фрагментов) программ;
* Унифицированная система типизации;
* Расширенная поддержка событийно-ориентированного программирования

Недостатки языка:

* Мало свежих концептуальных идей (приблизительно менее чем 10% конструкций языка);
* Относительно невысокая производительность (намного медленнее, чем язык C, но сравним с Java);
* C# очень легко дизассемблируется. Это означает, что с большой долей вероятности твой код будет получен и изучен конкурентами. Конечно же, есть специальные инструменты, которые могут усложнить этот процесс, но на 100% защититься от этого практически невозможно.



Java — это один из самых популярных и успешных языков программирования. Java создал доктор Джеймс Артур Гослинг. Он известен как «отец Java».

Изначально разработкой и поддержкой Java занималась компания Sun Microsystems. С января 2010 года, после того, как компанию Sun купила Oracle, этим занимается Oracle.

В основу Java положен принцип WORA («Write Once Run Anywhere», «Написано один раз, работает везде»). Один из главных плюсов Java, является кроссплатформенность разработанного приложения. Кроссплатформенность, т.е. возможность выполнения в любой операционной среде, достигается благодаря виртуальной Java машине, и приложение (jar, war) работают в этой виртуальной среде. Достаточно установить на компьютер JDK и запустить приложение.

Java - Строго типизированный, компилируемый, объектно-ориентированный язык программирования.

Другой важной особенностью технологии Java является гибкая система безопасности, в рамках которой исполнение программы полностью контролируется виртуальной машиной.

Изначально язык назывался Oak («Дуб») разрабатывался Джеймсом Гослингом для программирования бытовых электронных устройств. Впоследствии он был переименован в Java и стал использоваться для написания клиентских приложений и серверного программного обеспечения.

В настоящее время Java — это один из самых популярных языков.

Примеры использования Java: Java используется при разработке backend сервера, который будет работать с БД и отвечать на запросы frontend серверу, можно использовать в браузерах в виде java-applet, хотя этот способ программирования сейчас становиться не актуальным, ввиду безопасности исполняемого кода. В большинстве случаев Java, конечно же выступает в виде backend сервера.



JavaScript — это скриптовый язык, в котором используются механизмы прототипного наследования. Это — динамический язык со слабой типизацией, который поддерживает работу с функциями высшего порядка. JavaScript создал Брендан Эйх, работавший во время разработки этого языка в Netscape Communications Corporation.

JavaScript-код может выполняться в браузерах, либо на сервере(NodeJS).

Большинство сценариев JavaScript призваны "оживить" HTML, т. е. цель создания сценария заключается в том, чтобы продемонстрировать, как будет меняться вид страницы при изменении значений параметров HTML-тегов. В браузере они подключаются напрямую к HTML и, как только загружается страничка – тут же выполняются.

Когда создавался язык JavaScript, у него изначально было другое название: «LiveScript». Но тогда быстро набирал популярность язык Java, и маркетологи решили, что схожее название сделает новый язык более популярным.

Планировалось, что JavaScript будет эдаким «младшим братом» Java. Однако, история распорядилась по-своему, JavaScript сильно вырос, и сейчас это совершенно независимый язык, со своей спецификацией и к Java не имеет никакого отношения.

JavaScript используется на практически каждом серьёзном сайте. Для JavaScript был специально разработан JQuery, для облегчения управления элементами, доступа к элементам и работы с данными.

Пример на языке JavaScript

<!DOCTYPE html>  
<html>  
<head>  
<meta charset="utf-8">  
</head>  
<body>  
<p>Это обычный HTML документ</p>  
<script>  
document.write("<b>Это и есть JavaScript!</b>")  
</script>  
<p>Выходим обратно в HTML</p>  
</body>  
</html>



Python - Язык Python разработал Гвидо ван Россум из Центра математики и информатики (CWI). Python — это высокоуровневый язык общего назначения. При его проектировании особое внимание уделялось хорошей читабельности кода. Синтаксис ядра Python минималистичен. Код в Python организовывается в функции и классы, которые могут объединяться в модули. Его синтаксис считается чистым и выразительным.

В США Python почти заменил Java в научной среде. В частности, это выражается в том, что современные студенты начинают осваивать программирование с изучения Python, а не C или Java, как это было раньше.

Python широко используется в разных сферах, в частности, в веб-разработке и в области информационной безопасности. Этот язык применяется в таких компаниях, как Google, Yahoo и Spotify.

В отличии от других языков программирования, у Python есть целый ряд абстрактных интерфейсов. Все остальные языки можно строго разделить на компилируемый и интерпритируемые языки программирования, с Python нельзя так сказать и вот почему.

#### Python интерпретируемый или компилируемый?

Во-первых. Python ***–***это интерфейс. Существует [спецификация](http://docs.python.org/2/reference/index.html), описывающая, что должен делать Python, и как он должен себя вести. Существует несколько имплементаций.

Во-вторых. Интерпретируемый и компилируемый - свойства имплементации, но не интерфейса.

Т.о. вопрос не совсем корректен.

В случае с самой распространенной реализацией (CPython: написанный на C, часто называемый просто “Python”, и, конечно, именно тот, который все используют, просто скачав и установив Python) ответ: интерпретируемый, с некоторой компиляцией. CPython компилирует\* исходный код на Питоне в байткод, а затем интерпретирует этот байткод, запуская его в процессе.

\*это не совсем “компиляция” в традиционном смысле. Обычно “компиляция” это конвертация из высокоуровневого языка в машинный код. Тем не менее – в некотором роде это “компиляция”.

Изучим этот ответ получше.

#### Байткод или машинный код

Рассмотрим пример:

— С компилируется в машинный код, который впоследствии запускается напрямую процессором. Каждая инструкция заставляет процессор производить разные действия.  
— Java компилируется в байткод, который впоследствии запускается на Виртуальной машине Java (Java Virtual Machine, JVM), абстрактном компьютере, который запускает программы. Каждая инструкция обрабатывается JVM, который взаимодействует с компьютером.

Сильно упрощая: машинный код намного быстрее, но байткод лучше переносим и защищен. Машинный код может отличаться в зависимости от машины, тогда как байткод одинаковый на всех машинах. Можно сказать, что машинный код оптимизирован под вашу конфигурацию.

Возвращаясь к CPython, цепочка операций выглядит следующим образом:  
1.CPython компилирует ваш исходный код на Питоне в байткод.  
2. Этот байткод запускается на виртуальной машине CPython.

Рассмотрим и другие реализации Python

*По умолчанию* реализация Python это CPython, т.е. на С.

Одна из реализаций Python, это реализация JPython на Java. Эта реализация использует JVM. Jython генерирует байткод Java для запуска на JVM.

На рисунке 4 показаны компиляторы: на C и на Java.

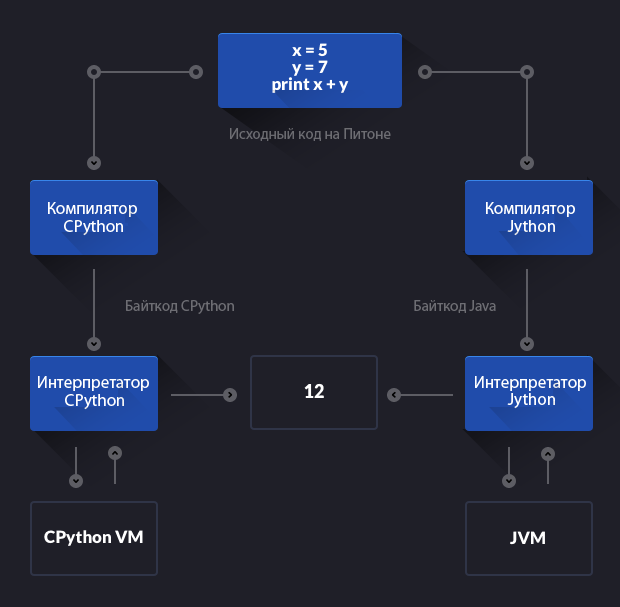


Рис.4. Компилятор CPython и Jython.

Для чего же нужны такие трудности с разными реализациями Python?Ответ: разные реализации хорошо ладят с разными наборами технологий.

CPython упрощает написание C-расширений для кода на Python, т.к. в конце он запускается интерпретатором Cи. Jython в свою очередь упрощает работу с другими программами на Java: вы можете импортировать любые Java-классы без дополнительных усилий, призывая и используя ваши Java-классы из программ на Jython.

Приведем пример:

[Java HotSpot(TM) 64-Bit Server VM (Apple Inc.)] on java1.6.0\_51  
>>> from java.util import HashSet  
>>> s = HashSet(5)  
>>> s.add("Foo")  
>>> s.add("Bar")  
>>> s  
[Foo, Bar]

В Python изначально нет понятия HashSet, благодаря Jython можно импортировать библиотеку из Java.

**IronPython** это другая реализация Python, написанная полностью на C# и предназначенная для .NET. В частности, она запускается на виртуальной машине .NET, если ее можно так назвать, на **Common Language Runtime (CLR),** от Майкрософт, сравнимым с JVM.

Можно сказать, что Jython: Java :: IronPython: C#. Они работают на соответствующих виртуальных машинах, есть возможность импортировать классы C# в код IronPython и классы Java в код Jython, и так далее.

Еще одна имплементация Python это имплементация написанная на Python (PyPy),

рассмотрим компиляцию на лету (just-in-time или JIT)

**JIT**

Нативный машинный код намного быстрее байткода. А что, если бы можно было компилировать часть байткода и запускать его как нативный код? Пришлось бы потратить некоторое количество времени за компиляцию байткода, но если результат будет работать быстрее, то этим можно принебречь! Этим и мотивируется JIT-компиляция, гибридная техника, которая совмещает в себе преимущества интерпретаторов и компиляторов. В двух словах – JIT старается использовать компиляцию, чтобы ускорить систему интерпретации.

Распространенный подход JIT:

1. Определить байткод, который запускается часто.
2. Скомпилировать его в нативный машинный код.
3. Закэшировать результат.
4. Всегда когда необходимо запустить тот же самый байткод, использовать уже скомпилированный машинный код и пожинать плоды (в частности, прирост скорости).

В этом и состоит вся суть имплементации PyPy. На рисунке 5 показана схема компилятора на Python.

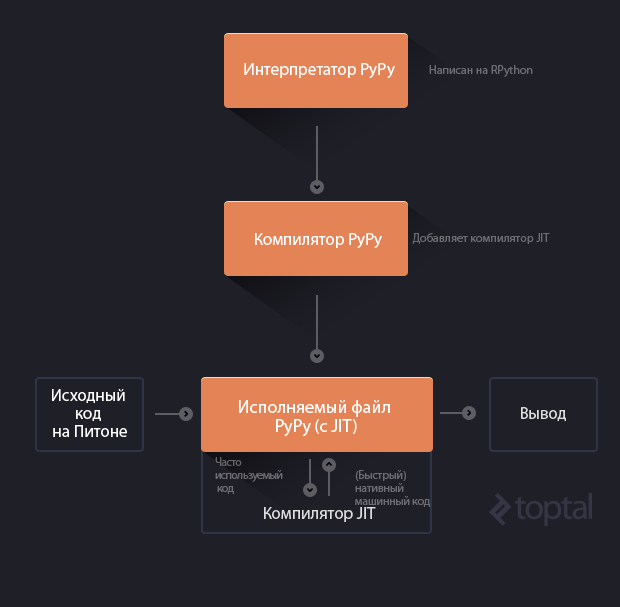
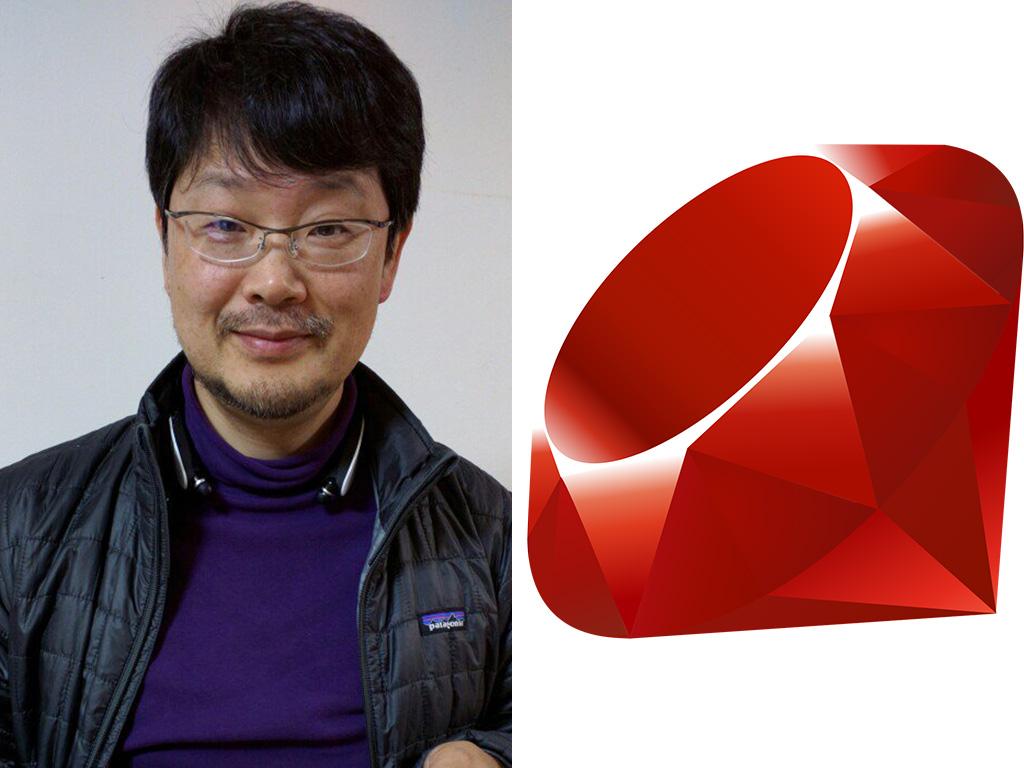


Рис.5.



Rubby - Язык Ruby был создан Юкихиро Мацумото в середине 1990-х в Японии. Программирование на Ruby — занятие увлекательное.

Воздействие на Ruby оказали такие языки, как Perl, Ada, Lisp, Smalltalk. Этот язык создан в расчёте на то, чтобы писать программы на нём можно было бы быстро и приятно. Язык обладает независимой от операционной системы реализацией многопоточности, строгой динамической типизацией, сборщиком мусора.

Основное назначение Ruby — создание простых и в то же время понятных программ, где важна не скорость работы программы, а малое время разработки, понятность и простота синтаксиса. Язык следует принципу «наименьшей неожиданности»: программа должна вести себя так, как ожидает программист.

Ruby, в основном, используется при разработке веб-приложений. В частности, этот язык применяется в таких компаниях, как Twitter, Hulu и Groupon.

В Ruby есть так-же как и в Python имплементации, рассмотрим одну из них это JRuby., т.е. опять наша любимая Java,приведем пример:

java\_import 'java.util.HashMap'

hash = HashMap.new

hash[:test1]=1

puts hash

>> {:test1=>1}

Ruby - **мощный язык**. Перечислю основные достоинства:

- Позволяет обрабатывать исключения в стиле Java и Python.

- Не поддерживает множественное наследование, но вместо него может использоваться

концепция «примесей», основанная в данном языке на механизме модулей.

- Содержит автоматический сборщик мусора. Он работает для всех объектов Ruby, в том

числе для внешних библиотек.

- Поддерживает замыкания с полной привязкой к переменным.

- Поддерживает блоки кода (код заключается в { … } или do … end). Блоки могут

использоваться в методах или преобразовываться в замыкания.

- Целые переменные в Ruby автоматически конвертируются между типами Fixnum (32-разрядные) и Bignum (больше 32 разрядов) в зависимости от их значения, что позволяет производить целочисленные математические расчёты со сколь угодно большой точностью.

- Может динамически загружать расширения, если это позволяет операционная система.

- Имеет независимую от ОС поддержку невытесняющей многопоточности.

- Перенесён на множество платформ. Он разрабатывался на Linux, но работает на многих версиях Unix, DOS, Microsoft Windows (в частности, Win32), Mac OS, BeOS, OS/2 и т. д.

- С официальной версией интерпретатора Ruby поставляется командная оболочка Ruby (Interactive Ruby Shell). Запускаемая командой irb в окне терминала (интерфейсе командной строки), она позволяет тестировать код программы очень быстро.



PHP - Язык PHP (изначально он назывался Personal Home Page Tools, что переводится как «Инструменты для создания персональных веб-страниц») создал в 1995 году Расмус Лердорф.

PHP является интерпретируемым языком общего назначения с открытым исходным кодом (скриптовый язык). PHP создавался специально для ведения web-разработок и код на нем может внедряться непосредственно в HTML-код. Синтаксис языка берет начало из C, Java и Perl, и является легким для изучения.

Основной целью PHP является предоставление web-разработчикам возможности быстрого создания динамически генерируемых web-страниц, однако область применения PHP не ограничивается только этим.

В настоящее время эталонная реализация PHP разрабатывается силами The PHP Group. PHP соперничал с технологиями Microsoft Active Server Pages (ASP) и Java Server Pages (JSP) и в итоге стал гораздо более востребованным, чем они. PHP используется такими интернет-гигантами, как Facebook, Wikipedia, Wordpress и Joomla.

**Основные классы инструментальных средств**

В настоящее время выделяют три основных класса инструментальных сред разработки и сопровождения ПС(программных средств) (схема 1):

* инструментальные среды программирования,
* рабочие места компьютерной технологии,
* инструментальные системы технологии программирования.

инструментальные среды разработки и сопровождения ПС

инструментальные среды программирования

рабочие места компьютерной технологии

инструментальные системы технологии программирования

Схема 1.

*Инструментальная среда программирования* предназначена в основном для поддержки процессов программирования (кодирования), тестирования и отладки ПС. Она не обладает рассмотренными выше свойствами комплексности, ориентированности на конкретную технологию программирования, ориентированности на коллективную разработку и, как правило, свойством интегрированности, хотя имеется некоторая тенденция к созданию интегрированных сред программирования (в этом случае их следовало бы называть *системами программирования*). Иногда среда программирования может обладать свойством специализированности. Признак же ориентированности на конкретный язык программирования может иметь разные значения, что существенно используется для дальнейшей классификации сред программирования.

*Рабочее место компьютерной технологии* ориентировано на поддержку ранних этапов разработки ПС (системного анализа и спецификаций) и автоматической генерации программ по спецификациям. Оно существенно использует свойства специализированности, ориентированности на конкретную технологию программирования и, как правило, интегрированности. Более поздние рабочие места компьютерной технологии обладают также свойством комплексности. Что же касается языковойориентированности, то вместо языков программирования они ориентированы на те или иные формальные языки спецификаций. Свойством ориентированности на коллективную разработку указанные рабочие места в настоящее время, как правило, не обладают.

*Инструментальная система технологии программирования* предназначена для поддержки всех процессов разработки и сопровождения в течение всего жизненного цикла ПС и ориентирована на коллективную разработку больших программных систем с продолжительным жизненным циклом. Обязательными свойствами ее являются комплексность, ориентированность на коллективную разработку и интегрированность. Кроме того, она или обладает технологической определенностью или получает это свойство в процессе расширения (настройки). Значение признака языковой ориентированности может быть различным, что используется для дальнейшей классификации этих систем.

**API**

API (Application programming interface) — это контракт, который предоставляет программа. «Ко мне можно обращаться так и так, я обязуюсь делать то и это». Т.е. другими словами, API – это представление(view) предоставляемого функционала. Специально разработана для запросов в определенный удаленный сервис.

API определяет функциональность, которую предоставляет программа ([модуль](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C_(%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%D0%B5)), [библиотека](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_(%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%D0%B5))), при этом API позволяет абстрагироваться от того, как именно эта функциональность реализована. На рисунке 6. Показана схема обращения с API

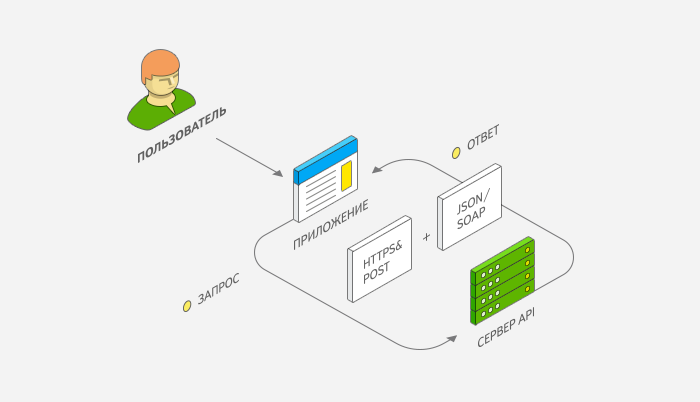


Рис.6.

Если программу (модуль, библиотеку) рассматривать как [чёрный ящик](https://ru.wikipedia.org/wiki/%D0%A7%D1%91%D1%80%D0%BD%D1%8B%D0%B9_%D1%8F%D1%89%D0%B8%D0%BA), то API — это множество «ручек», которые доступны пользователю данного ящика и которые он может вертеть и дёргать.

Программные компоненты взаимодействуют друг с другом посредством API. При этом обычно компоненты образуют иерархию — высокоуровневые компоненты используют API низкоуровневых компонентов, а те, в свою очередь, используют API ещё более низкоуровневых компонентов.

По такому принципу построены [протоколы передачи данных](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB_%D0%BF%D0%B5%D1%80%D0%B5%D0%B4%D0%B0%D1%87%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) по [Интернету](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82). Стандартный стек протоколов ([сетевая модель OSI](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_OSI)) содержит 7 уровней (от физического уровня передачи бит до уровня протоколов приложений, подобных протоколам [HTTP](https://ru.wikipedia.org/wiki/HTTP) и [IMAP](https://ru.wikipedia.org/wiki/IMAP)). Каждый уровень пользуется функциональностью предыдущего («нижележащего») уровня передачи данных и, в свою очередь, предоставляет нужную функциональность следующему («вышележащему») уровню.

Важно заметить, что понятие протокола близко по смыслу к понятию API. И то, и другое является абстракцией функциональности, только в первом случае речь идёт о передаче данных, а во втором — о взаимодействии приложений.

API библиотеки функций и классов включает в себя описание сигнатур и семантики функций.

*Сигнатура функции* — часть общего объявления функции, позволяющая средствам трансляции идентифицировать функцию среди других. В различных языках программирования существуют разные представления о сигнатуре функции, что также тесно связано с возможностями перегрузки функций в этих языках.

Иногда различают *сигнатуру вызова* и *сигнатуру реализации* функции. Сигнатура вызова обычно составляется по синтаксической конструкции вызова функции с учётом сигнатуры области видимости данной функции, имени функции, последовательности фактических типов аргументов в вызове и типе результата. В сигнатуре реализации обычно участвуют некоторые элементы из синтаксической конструкции объявления функции: спецификатор области видимости функции, её имя и последовательность формальных типов аргументов.

Например, в языке программирования C++ простая функция однозначно опознаётся компилятором по её имени и последовательности типов её аргументов, что составляет сигнатуру функции в этом языке. Если функция является методом некоторого класса, то в сигнатуре будет участвовать и имя класса.

В языке программирования Java сигнатуру метода составляет его имя и последовательность типов параметров; тип возвращаемого значения в сигнатуре не участвует.

*Семантика функции* — это описание того, что данная функция делает. Семантика функции включает в себя описание того, что является результатом вычисления функции, как и от чего этот результат зависит. Обычно результат выполнения зависит только от значений аргументов функции, но в некоторых модулях есть понятие состояния. Тогда результат функции может зависеть от этого состояния, и, кроме того, результатом может стать изменение состояния. Логика этих зависимостей и изменений относится к семантике функции. Полным описанием семантики функций является исполняемый код функции или математическое определение функции.

API используется практически в любых современных сервисах. API для доступа почтовый программы к почтовому сервису, различным онлайн картам, сервисы по предоставлению погоды и многие другие. Рассмотрим пример использования API.

**API яндекс карт.**

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title>Быстрый старт. Размещение интерактивной карты на странице</title>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<script src="https://api-maps.yandex.ru/2.1/?apikey=ваш API-ключ&lang=ru\_RU" type="text/javascript">

</script>

<script type="text/javascript">

ymaps.ready(init);

function init(){

var myMap = new ymaps.Map("map", {

center: [55.76, 37.64],

zoom: 7

});

}

</script>

</head>

<body>

<div id="map" style="width: 600px; height: 400px"></div>

</body>

</html>

**API государственной думы**

var token = "10e644926dae6a63f3fb0a2f47b0e0ac0e544804";

$.ajax({

"url":"http://api.duma.gov.ru/api/" + token + "/search.json",

"data": {

"status":6,//законопроекты,рассмотрение которых завершено

"registration\_start": "2011-01-01",

"registration\_end":"2011-03-31",

"limit": 10//вывести первые 10 найденных законопроектов

},

"dataType": "jsonp",

"success": function(data) {

// возникла ошибка, выводим код и сообщение об ошибке

if (data.code) {

$(".ajax").remove();

alert("Error #" + data.code + ": " + data.text);

return false;

}

// отправляем данные в шаблон и получаем конечный html

var resultHtml = tmpl("column1\_tmpl", { data: data });

// выводим на странице результат

$("#column-1 .ajax").replaceWith(resultHtml);

}

});

В самом общем случае для создания программы на выбранном языке программирования (языке системного программирования) нужно иметь следующие компоненты:

1. Текстовый редактор для создания файла с исходным текстом программы.

2. Компилятор или интерпретатор. Исходный текст с помощью программы-компилятора переводится в промежуточный объектный код. Исходный текст большой программы состоит из нескольких модулей (файлов с исходными текстами). Каждый модуль компилируется в отдельный файл с объектным кодом, которые затем надо объединить в одно целое.

3. Редактор связей или сборщик, который выполняет связывание объектных модулей и формирует на выходе работоспособное приложение - исполнимый код.

Исполнимый код - это законченная программа, которую можно запустить на любом компьютере, где установлена операционная система, для которой эта программа создавалась. Как правило, итоговый файл имеет расширение .ЕХЕ или .СОМ.

В последнее время получили распространение визуальный методы программирования (с помощью языков описания сценариев), ориентированные на создание Windows-приложений. Этот процесс автоматизирован в средах быстрого проектирования. При этом используются готовые визуальные компоненты, которые настраиваются с помощью специальных редакторов.

Наиболее популярные редакторы (системы программирования программ с использованием визуальных средств) визуального проектирования:

- Borland Delphi - предназначен для решения практически любых задачи прикладного программирования.

- Microsoft Visual C++ - это средство позволяет разрабатывать любые приложения, выполняющиеся в среде ОС типа Microsoft Windows

- Eclipse – средство позволяет разрабатывать кросплатформенные приложения на языке Java.

Таким образом, сущность программного обеспечения заключается в создании любой исполняемой программы , путем преобразования формально логических выражений в исполняемый машинный код, а также его контроль и корректировка.

Для программного обеспечения, как особой разновидности программного обеспечения, характерны общие и частные специализированные функции:

1. Создание текста разрабатываемой программы с использованием специально установленных кодовых слов (языка программирования), а также определенного набора символов и их расположения в созданном файле - синтаксис программы.

2. Перевод текста создаваемой программы в машинно-ориентированный код, доступный для распознавания ЭВМ. В случае значительного объема создаваемой программы, она разбивается на отдельные модули и каждый из модулей переводится отдельно.

3. Соединение отдельных модулей в единый исполняемый код, с соблюдением необходимой структуры, обеспечение координации взаимодействия отдельных частей между собой.

4. Тестирование и контроль созданной программы, выявление и устранение формальных, логических и синтаксических ошибок, проверка программ на наличие запрещенных кодов, а также оценка работоспособности и потенциала созданной программы.

**Виды инструментального программного обеспечения**

Исходя из задач, поставленных перед инструментальным программным обеспечением, можно выделить большое количество различных по назначению видов инструментального программного обеспечения:

- Текстовые редакторы

- Интегрированные среды разработки

- SDK

- Компиляторы

- Интерпретаторы

- Линковщики(компоновщики)

- Парсеры и генераторы парсеров (см. Javacc)

- Ассемблеры

- Отладчики

- Профилировщики

- Генераторы документации

- Средства анализа покрытия кода

- Средства непрерывной интеграции

- Средства автоматизированного тестирования

- Системы управления версиями и др.

Оболочки для создания прикладных программ создаются также инструментальными программами и поэтому могут быть отнесены к прикладным программам. Рассмотрим кратко назначения некоторых инструментальных программ.

*Текстовые редакторы.*

Текстовый редактор — компьютерная программа, предназначенная для обработки текстовых файлов, такой как создание и внесение изменений.

Условно выделяют два типа редакторов: потоковые текстовые редакторы иинтерактивные.

1) Потоковые текстовые редакторы

Потоковые текстовые редакторы представляют собой компьютерные программы, которые предназначены для автоматизированной обработки входныхтекстовых данных, полученных из текстового файла, в соответствии с заранее заданными пользователями правилами. Чаще всего правила представляют собой регулярные выражения.

2) Интерактивные текстовые редакторы

Интерактивные текстовые редакторы - это семейство компьютерных программ предназначенных для внесения изменений в текстовый файл в интерактивном режиме. Такие программы позволяют отображать текущее состояние текстовых данных в файле и производить над ними различные действия.

*Интегрированная среда разработки*

Интегрированная среда разработки - система программных средств, используемая программистами для разработки ПО. Обычно среда разработки включает в себя:

1) текстовый редактор

2) компилятор и/или интерпретатор

3) средства автоматизации сборки

4) отладчик.

*SDK*.

**SDK** (от англ. Software Development Kit) или «devkit» - комплект средств разработки, который позволяет специалистам по программному обеспечению создавать приложения для определённого пакета программ, программного обеспечения базовых средств разработки, аппаратной платформы, компьютерной системы, видеоигровых консолей, операционных систем и прочих платформ.

Примеры SDK:

1) Adobe Flex

2) DirectX

3) Eclipse

4) Java Development Kit

5) Opera Devices SDK

6) Source SDK

*Компиляторы*

Компилятор описывается следующими свойствами:

1) Программа или техническое средство, выполняющее компиляцию.

2) Машинная программа, используемая для компиляции.

3) Транслятор, выполняющий преобразование программы, составленной на исходном языке, в объектный модуль.

4) Программа, переводящая текст программы на языке высокого уровня в эквивалентную программу на машинном языке.

5) Программа, предназначенная для трансляции высокоуровневого языка в абсолютный код или, иногда, в язык ассемблера. Входной информацией для компилятора (исходный код) является описание алгоритма или программа на проблемно-ориентированном языке, а на выходе компилятора -- эквивалентное описание алгоритма на машинно-ориентированном языке (объектный код).

*Интерпретаторы.*

Интерпретатор (языка программирования):

1) Программа или техническое средство, выполняющее интерпретацию.

2) Вид транслятора, осуществляющего покомандную обработку и выполнение исходной программы или запроса (в отличие от компилятора, транслирующего всю программу без её выполнения).

3) Программа, анализирующая команды или операторы программы и тут же выполняющая их.

4) Языковый процессор, который построчно анализирует исходную программу и одновременно выполняет предписанные действия, а не формирует на машинном языке скомпилированную программу, которая выполняется впоследствии.

*Компоновщик.*

**Компоновщик** (также редактор связей, линкер) - программа, которая производит компоновку, принимает на вход один или несколько объектных модулей и собирает по ним исполнимый модуль.

Для связывания модулей компоновщик использует таблицы имён, созданные компилятором в каждом из объектных модулей. Такие имена могут быть двух типов:

1) Определённые или экспортируемые имена -функции и переменные, определённые в данном модуле и предоставляемые для использования другим модулям.

2) Неопределённые или импортируемые имена -- функции и переменные, на которые ссылается модуль, но не определяет их внутри себя.

*Ассемблер.*

**Ассемблер** (от англ. assembler -- сборщик) -- компьютерная программа, компилятор исходного текста программы, написанной на языке ассемблера, в программу на машинном языке.

*Отладчики*

Отладчик или дебаггер является модулем среды разработки или отдельным приложением, предназначенным для поиска ошибок в программе. Отладчик позволяет выполнять пошаговую трассировку, отслеживать, устанавливать или изменять значения переменных в процессе выполнения программы, устанавливать и удалять контрольные точки или условия остановки и т. д.

*Профилировщики*

**Профилирование** — сбор характеристик работы программы, таких как время выполнения отдельных фрагментов (обычно подпрограмм), число верно предсказанных условных переходов, число кэш-промахов и т. д. Инструмент, используемый для анализа работы, называют профилировщиком или профайлером. Обычно выполняется совместно с оптимизацией программы.

*Генератор документации*

Генератор документации - программа или пакет программ, позволяющая получать документацию, предназначенную для программистов (документация на API) и/или для конечных пользователей системы, по особым образом комментированному исходному коду и, в некоторых случаях, по исполняемым модулям (полученным на выходе компилятора).