**Высокоуровневый язык программирования** — язык программирования, разработанный для быстроты и удобства использования программистом. Основная черта высокоуровневых языков — это абстракция, то есть введение смысловых конструкций, кратко описывающих такие структуры данных и операции над ними, описания которых на машинном коде (или другом низкоуровневом языке программирования) очень длинны и сложны для понимания.

Язык **Паскаль**, широко известный в вузовских кругах, в основном в России и в странах бывшего Советского Союза. Он использовался и продолжает использоваться на удивление как учебный язык.

С **Delphi** обычно ассоциируется среда разработки приложений на основе языка **Object Pascal**, разработанного фирмой **Borland**. Этот язык является наследником **Turbo Pascal** с объектно-ориентированными расширениями, который в свою очередь ведет свою историю от **«чистого» Pascal**, созданного Никлаусом Виртом в 1970 году.

В Windows довольно быстро начал развиваться графический интерфейс, а в IDE Turbo Pascal по-прежнему использовался DOS-подобный интерфейс. Постепенно в Microsoft поняли, что нужно двигаться в сторону «визуализации» программирования.

Компания Borland начала дорабатывать его, но вскоре стало понятно: изменения и добавления будут настолько серьезными, что результат этой работы можно считать новым языком программирования. Для нового языка была разработана IDE с графическим интерфейсом. Новая среда программирования и язык получили общее название Delphi.

Впоследствии, в 2002 году, разработчики из компании Borland официально поставили знак равенства между языками Delphi и Object Pascal.

В Delphi 2006, появилась возможность писать приложения для .NET используя стандартную библиотеку классов .NET, VCL для .NET

**Бе́йсик** (от BASIC, **«Beginner’s All-purpose Symbolic Instruction Code» — универсальный код символических инструкций для начинающих**) — семейство высокоуровневых языков программирования. Был разработан **в 1964 году** профессорами Дартмутского колледжа **Томасом Курцем и Джоном Кемени** (оба математик и математик-информатик).

Язык создавался как инструмент, с помощью которого **студенты-непрограммисты** могли самостоятельно создавать компьютерные программы для решения своих задач.

К настоящему моменту претерпел существенные изменения, значительно отойдя от характерной для первых версий **простоты**, граничащей с **примитивизмом**, и превратившись в достаточно **ординарный язык высокого уровня** с типичным набором возможностей. Используется как самостоятельный язык для разработки прикладных программ, главным образом, работающих под управлением **ОС Windows различных версий**.

Язык **Visual Basic** унаследовал дух, стиль и отчасти синтаксис своего предка — языка BASIC, у которого есть немало диалектов. В то же время Visual Basic сочетает в себе **процедуры** и элементы **объектно-ориентированных языков** программирования.

Достоинства

* Высокая скорость создания приложений с графическим интерфейсом для Microsoft Windows.
* Простой синтаксис языка обеспечивает низкий порог вхождения.
* Возможность компиляции как в машинный код, так и в P-код (байт-код, т.е. код для виртуальной машины) (по выбору программиста).
* Безопасность типов обеспечивает защита от ошибок, связанных с применением указателей и доступом к памяти.
* Возможность использования большинства WinAPI функций

Недостатки

* Поддержка операционных систем только семейства Windows и Mac OS X.

**Visual Basic .NET (VB.NET)** — объектно-ориентированный язык программирования, который можно рассматривать как очередной виток эволюции Visual Basic (VB), реализованный на платформе **Microsoft .NET.** VB.NET **не имеет обратной совместимости** с более ранней версией (Visual Basic 6.0).

**Си** является орудием системного программиста и позволяет глубоко влезать в самые тонкие механизмы обработки информации на ЭВМ. Хотя язык требует от программиста высокой дисциплины, он не строг в формальных претензиях и допускает краткие формулировки.

**Си** – современный язык. Он включает в себя те управляющие конструкции, которые рекомендованы теорией и практикой программирования.

**Си** – мощный и гибкий язык. Большая часть операционной системы Unix, компиляторы и интерпретаторы языков Фортран, Паскаль, Лисп, и Бейсик написаны именно с его помощью.

**Си** – удобный язык. Он достаточно структурирован, чтобы поддерживать хороший стиль программирования и вместе с тем не связан жесткими ограничениями. В некотором смысле язык Си – самый универсальный, т.к. кроме набора средств, присущих современным языкам программирования высокого уровня (**структурность, модульность, определенные типы данных**), в него включены средства для программирования практически на уровне ассемблера. Большой набор операторов и средств требуют от программиста осторожности, аккуратности и хорошего знания языка со всеми иго преимуществами и недостатками.

И, наконец, **Си придумали инженеры**. Если Паскаль придумал учёный, то Си придумали **Керниган и Ритчи**, они работали инженерами в Bell. В то время на этих языках (Fortran, COBOL, Algol) ничего системного написать было нельзя. Например, операционную систему, драйвера какие-нибудь, ещё что-то. Эти языки предназначались для математических расчётов, для бизнес-расчётов, для всего такого. А всё остальное писали на Ассемблере. (язык Си появился не сразу от Ассемблера, а через какие-то промежуточные вещи)

Язык C имеет огромное количество недостатков (ну просто вообще огромное) — на нём можно делать вообще всё, в том числе стрелять себе в ногу, стрелять себе в ногу с выдумкой, в другую ногу, одной ногой стрелять в другую ногу, в общем — что угодно делать. Но при этом некоторые архитектурные вещи там делаются довольно сложно — опять же, как и в Ассемблере, **нам приходится всё время следить, где мы, чего и какую память выделили; она там всё время «течёт» куда-то эта память -— то есть мы выделили, забыли удалить, удалили не то, вылезли за пределы памяти, в общем — огребли кучу проблем.**

**C++** создавался сначала как набор дополнений к языку C, который облегчит разработку. В то время стало модно **объектно-ориентированное программирование** и люди решили, что всё можно описать в виде иерархии. C++ в каком-то смысле реализовывал этот объектный подход — это не был первый язык объектно-ориентированного программирования, но он стал достаточно популярным и в нём начали появляться всякие фичи. При этом C++ сохранял почти полную совместимость (на тот момент) с языком C, программа написанная на Си в 99% случаев успешной компилировалась как C++-ная и даже работала также. Это было задумано, чтобы с Си было легко перейти на C++

Помимо объектного подхода (в C++) достаточно быстро появилась стандартная библиотека шаблонов ([STL](https://en.wikipedia.org/wiki/Standard_Template_Library)).

**Язык Java** зародился как часть проекта создания передового программного обеспечения (ПО) для различных бытовых приборов. Реализация проекта была начата на языке С++, но вскоре возник ряд проблем, наилучшим средством борьбы с которыми было изменение самого инструмента - языка программирования. Стало очевидным, что необходим платформо-независимый язык программирования, позволяющий создавать программы, которые не приходилось бы компилировать отдельно для каждой архитектуры и можно было бы использовать на различных процессорах под различными операционными системами. Язык Java потребовался для создания интерактивных продуктов для сети Internet. Фактически, большинство архитектурных решений, принятых при создании Java, было продиктовано желанием предоставить синтаксис, сходный с Си и Cи++. В Java используются практически идентичные соглашения для объявления переменных, передачи параметров, операторов и для управления потоком выполнением кода. **В Java добавлены все хорошие черты C++.**

Три ключевых элемента объединились в технологии языка Java:

- Java предоставляет для широкого использования свои апплеты (applets) — небольшие, надежные, динамичные, не зависящие от платформы активные сетевые приложения, встраиваемые в страницы Web. Апплеты Java могут настраиваться и распространяться потребителям с такой же легкостью, как любые документы HTML.

- Java высвобождает мощь объектно-ориентированной разработки приложений, сочетая простой и знакомый синтаксис с надежной и удобной в работе средой разработки. Это позволяет широкому кругу программистов быстро создавать новые программы и новые апплеты.

- Java предоставляет программисту богатый набор классов объектов для ясного абстрагирования многих системных функций, используемых при работе с окнами, сетью и для ввода-вывода. Ключевая черта этих классов заключается в том, что они обеспечивают создание независимых от используемой платформы абстракций для широкого спектра системных интерфейсов.

**Скриптовые языки из 90-x**

**Perl** был предназначен для обработки текстов — в те времена уже памяти стало настолько много в компьютерах, что туда можно было запихнуть какой-то текст и с этим текстом что-то полезное делать. На мой взгляд, на нём можно только писать, читать его нельзя. Когда я смотрю на код на Перле и пытаюсь что-то понять, я ничего не понимаю. То есть, программки получаются коротки и реально проще переписать заново, чем разобраться с тем, что там есть и исправить.

**PHP** появился как… случайно. Один человек довольно быстро перестал заниматься этим — делал свою страничку какую-то, была у него гостевая книга какая-то, ещё что-то, какие-то штуки. И он написал набор макросов каких-то для Перла, которые были похожи на Си, потому что он на Си умел, просто потому что ему было так удобно. И назвал это **Personal HomePage**. Расшарил и говорит: «Люди, смотрите какую штуку я написал, здесь всё намного понятнее, чем на Перле и можно править». И людям это понравилось.  
 Потом он это дело бросил. В общем, в итоге этот PHP стал жить и стал со временем гораздо популярнее, чем Perl. Но вот эта его «родовая травма» (задумка как набор макросов для Перла) с ним сыграла довольно злую шутку. Язык получился странный. То есть он развивался сам по себе, его никто не проектировал, никто не администрировал процесс развития (ни компания, ни какой-то человек), а было много группок, каждая из которых пилила то, что им нравится. В итоге там функции называются по-разному, даже стиля нет, всё через подчёркивания, в общем как попало, настройки лежат здесь и там, и как всё это будет работать не очень понятно. Зато можно сесть и через два часа писать на PHP, потому что он так задумывался.  
  
**Python**

**Достоинства языка.**

Несомненным достоинством является то, что **интерпретатор Python** реализован практически на всех платформах и операционных системах. (прям как С, только примитивы с одинаковым размером на разных машинах)

Следующая немаловажная черта - расширяемость языка, этому придается большое значение и, как пишет сам автор, язык был задуман именно как расширяемый. Это означает, что имеется возможность совершенствования языка всеми всеми заинтересованными программистами. Интерпретатор написан на С и исходный код доступен для любых манипуляций. В случае необходимости, можно вставить его в свою программу и использовать как встроенную оболочку. Или же, написав на C свои дополнения к Python и скомпилировав программу, получить "расширенный" интерпретатор с новыми возможностями.

Следующее достоинство - наличие большого числа подключаемых к программе модулей, обеспечивающих различные дополнительные возможности. Такие модули пишутся на С и на самом Python и могут быть разработаны всеми достаточно квалифицированными программистами. В качестве примера можно привести следующие модули:

**Numerical Python** - расширенные математические возможности, такие как манипуляции с целыми векторами и матрицами;

**Tkinter** - построение приложений с использованием графического пользовательского интерфейса (GUI) на основе широко распространенного на X-Windows Tk-интерфейса;

**OpenGL** - использование обширной библиотеки графического моделирования двух- и трехмерных объектов Open Graphics Library фирмы Silicon Graphics Inc. Данный стандарт поддерживается, в том числе, в таких распространенных операционных системах как Microsoft Windows 95 OSR 2, 98 и Windows NT 4.0.

**Недостатки языка.**

Недостатком является сравнительно невысокая скорость выполнения Python-программы, что обусловлено ее интерпретируемостью. Однако, на наш взгляд, это с лихвой окупается достоинствами языка при написании программ не очень критичных к скорости выполнения.

**Обзор особенностей**

Python, в отличие от многих языков (Pascal, C++, Java, и т.д.), не требует описания переменных. Они создаются в месте их инициализации, т.е. при первом присваивании переменной какого-либо значения. Значит, тип переменной определяется типом присваиваемого значения.

Тип переменной не является неизменным. Любое присваивание для нее корректно и это приводит лишь к тому, что типом переменной становится тип нового присваиваемого значения.

Одним из особенностей Python является то, как происходит присваивание одной переменной другой, т.е. когда по обе стороны от оператора "=" стоят переменные.

Присваивание в Python происходит следующим образом: если присваеваемый объект является экземпляром таких типов как числа или строки, то действует семантика копирования, если же в правой части стоит экземпляр класса, список, словарь, то действует семантика указателей. Пример:

a = 2; b = a; b = 3

print ' семантика копирования: a=', a, 'b=', b

a = [2,5]; b = a; b[0] = 3

print ' семантика указателей: a=', a, 'b=', b

Результат:

семантика копирования: a= 2 b= 3

семантика указателей: a= [3,5] b= [3,5]

 дать объекту "список **[2,5]**" имя **a** ;

 дать этому объекту еще одно имя - **b** ;

 модифицировать нулевой элемент объекта.  
 Так и появилась **Java (Ява)**. Там тоже связана довольно забавная история, как название появилось у этого языка. Там программисты, они всё время пьют кофе и в то время было модным пить кофе, который на острове Ява рос. Язык задумывался как язык для встраиваемых приборов, в частности для кофе-машины. Вот так и получилось название…  
Что вообще началось с неё, что в ней было хорошее и почему же она завоевала большую популярность? Во-первых, они избавились от наследия Сишного, полностью. Никаких указателей, гораздно меньше способов отстрелить себе какую-нибудь часть тела и всё сломать. Во-вторых, они внедрили гораздо более свежие идеи в плане объектной модели — то есть C++ появился значительно раньше, чем Java и использовал более архаичную, «дикую» модель объектную. Ну а здесь она была уже более продуманной тогда уже, и в теории люди думали, и на практике применяли и сделали всё намного круче.  
+И наконец, третье. У нас программки на Джаве собирались не в машинный код, а в код для виртуальной машины. То есть, у вас жила виртуальная машина (VM) JVM — Javовская. Ваши программки собирались в какое-то промежуточное представление и затем, с помощью это машины уже выполнялись. Что это дало? Во-первых, оно тормозило, во-вторых, оно жрало память со страшной силой, в третьих оно было переносимо куда угодно (теоретически) — хоть на кофеварку, хоть на кофемолку, хоть на компьютер, хоть на телефон мобильный. Это, с одной стороны хорошо, то есть вы написали просто реализацию виртуальной машины, потом свои джавовские программы запускаете везде. Но, с другой стороны, плохо, что на том же телефоне тогда было мало памяти, была низкая производительность и всё это ещё дополнительно начинало тупить и тормозить.  
  
 Но даже не это главное, для чего вообще язык придумывался. Язык Джава придумывался чтобы снизить требования к квалификации программистов. То есть более плохие программисты могут писать хорошие программы на Java, потому что она не позволяет писать плохие программы — там нет средств, чтобы писать программы плохо. Там можно писать только хорошо, программы. Хорошо, в понимании создателей языка.  
  
 То есть, если на Си, на C++, на Питоне, на чём угодно мы можем развести из своего проекта жуткую помойку какую-то, где у нас всё лежит вперемешку, собирается часами и там ещё чего-то. То в Java помойку развести тоже можно, но для этого уже надо приложить какие-то усилия. То есть, по умолчанию, там получается не “помойка”, получаются другие проблемы, что там что-то пронаследовали-пронаследовали — в общем на одну осмысленную строку получается десять не очень осмысленных. Зато, может такой, средней квалификации, программист писать довольно качественный код.  
 У нас, следующее, что появилось — это **.Net (дотнет)**, ну и в частности нас интересует **C#** (почти то же самое [Java], то есть там различия в деталях, если будете выбирать между ними — смотрите, где денег больше платят).  
  
 И ещё одна штука — **JavaScript**. Не имеет никакого отношения к языку Java, появился в том же году — слово было модное, они лицензировали торговую марку, чтобы использовать.  
  
 На что, главное, нужно обратить внимание? (рисует стрелки от C++ к Java, .Net, C#, JavaScript и PHP). Чтобы написать простенькую программку на одном из этих языков, да и на многих других — если вы знаете C++, вам в общем-то ничего больше знать не надо — вы берете и пишете на C++, а потом добавляете доллары в начале, ещё что-то делаете по мелочи и у вас она начинает работать на чём угодно (лектор показывает на языки, к которым были отведены стрелочки от C++). То есть они предельно похожи в каких-то простых вещах. Если вы решаете какие-то школьные задачи, учебные задачи, ещё что-то (не проектируете большой проект — у вас один файлик, который читает числа, выводит числа в консоли, ещё чего-то делает), то разницы почти никакой нет между этими языками. Понятно, что JavaScript и PHP они специализированные, у них всё немножко по другому. А вот здесь (лектор указывает на Java и C#) вообще предельно мало разницы.

**Go.** Go (часто также Golang) — компилируемый многопоточный язык программирования, разработанный компанией Google.

Язык Go разрабатывался как язык системного программирования для создания высокоэффективных программ, работающих на современных распределённых системах и многоядерных процессорах. Он может рассматриваться как попытка создать замену языкам Си и C++[8]. При разработке уделялось особое внимание обеспечению высокоэффективной компиляции. Программы на Go компилируются в объектный код (хотя доступен и интерпретатор) и не требуют для исполнения виртуальной машины.

Основные возможности языка Go:

* Go — язык со строгой статической типизацией. Доступен автоматический вывод типов, для пользовательских типов — «утиная типизация».
* Полноценная поддержка указателей, но без возможности применять к ним арифметические операции, в отличие от C/C++/D.
* Строковый тип со встроенной поддержкой юникода.
* Использование динамических массивов, хэш-таблиц, срезов (слайсов), вариант цикла для обхода коллекции.
* Средства функционального программирования: неименованные функции, замыкания, передача функций в параметрах и возврат функциональных значений.
* Автоматическое управление памятью со сборщиком мусора.
* Средства объектно-ориентированного программирования, но без поддержки наследования реализации (наследуются только интерфейсы). По большому счёту, Go является процедурным языком с поддержкой интерфейсов.
* Средства параллельного программирования: встроенные в язык потоки (go routines), взаимодействие потоков через каналы и другие средства организации многопоточных программ.
* Достаточно лаконичный и простой синтаксис, основанный на Си, но существенно доработанный, с большим количеством синтаксического сахара.

При этом из языка сознательно исключены:

* Структурная запись обработчиков исключений. Вместо неё предлагается проверка кодов возврата с использованием многозначных функций и использование отложенных (deferred) функций и специального интерфейса error для перехвата исключительных ситуаций.
* Наследование реализации. В качестве замены предлагается анонимное вложение типов.
* Обобщённое программирование.
* Использование утверждений.
* переопределение методов.

Язык продолжает развиваться, и разработчики рассматривают возможность включения в язык средств обобщённого программирования. В «Часто задаваемых вопросах»[5] по языку приводятся аргументы против использования утверждений, а наследование без указания типа, наоборот, отстаивается.