## Занятия 1 Современное программирование

План Занятия:

* Экскурс в историю
* Операционные системы
* Языки программирования
* Компилятор и интерпретатор
* Бэкенд и Фронтенд
* Python
* Установка, настройка, PyCharm
* Режим консоли и запуск файлов
* Print(‘Hello world!’)

# Ход занятия:

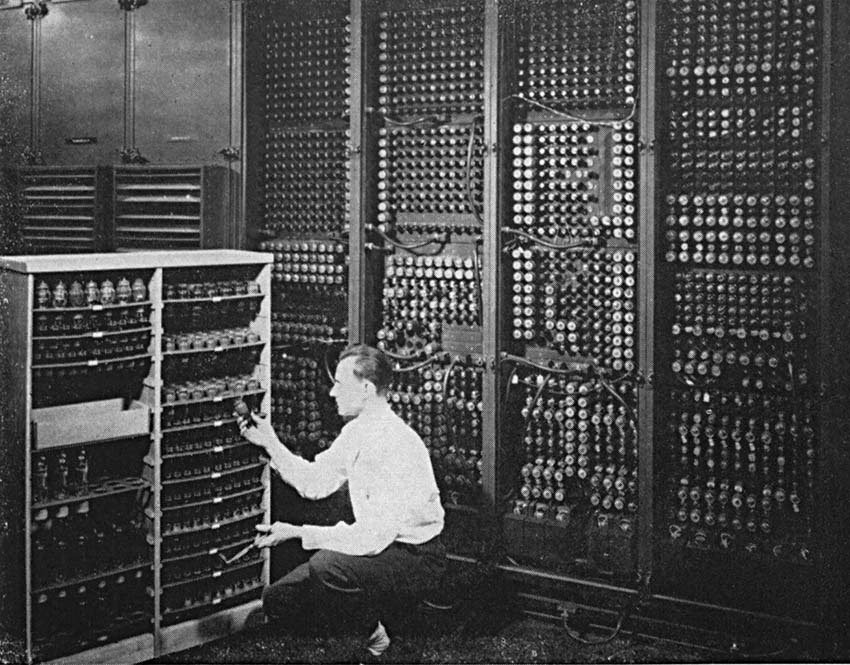
1. Организационный момент:

Представление преподователи и знакомство с учениками

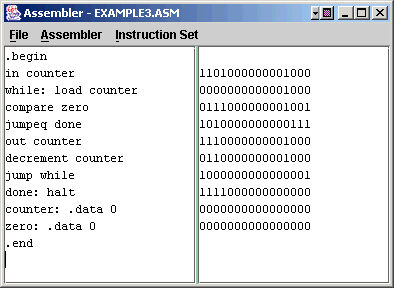
Проверка учеников по списку

1. Переход к Плану занятия:

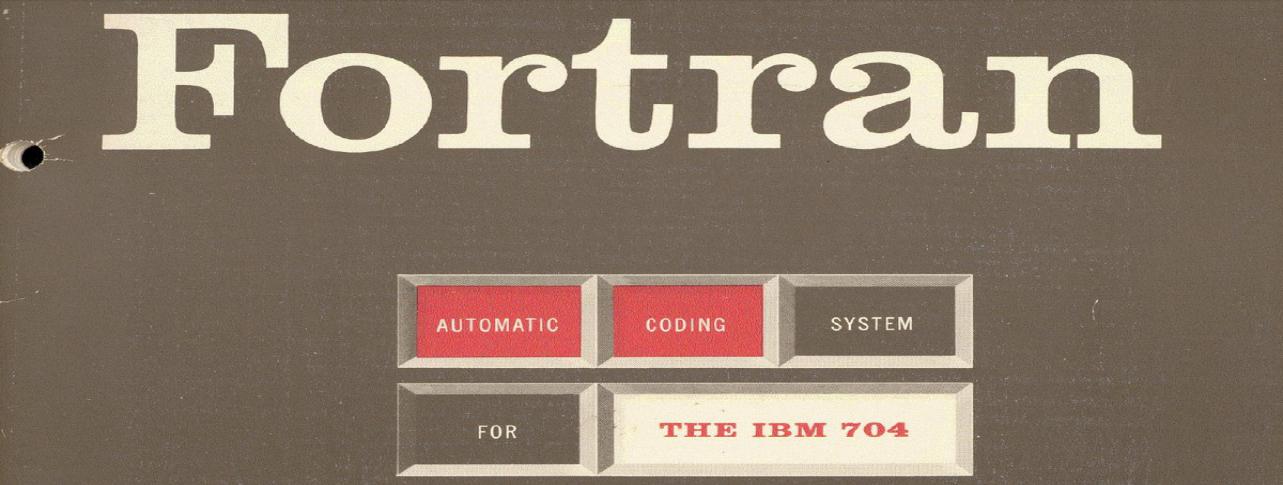
* Экскурс в историю

Языки программирования создавались и разрабатывались в течение многих лет. Некоторые из них до сих пор находятся в процессе развития, так как потребности потребителя растут, разрабатывается новое «железо» и программистам приходится искать новые решения, а также принципы работы с инновациями. Само понятие «язык программирования» можно трактовать как формальную знаковую систему, которую программисты используют для написания компьютерных программ. Однако, о языке машин инженеры стали задумываться намного раньше, чем у них появилась возможность писать на нем и создавать программы. Идею о создании языка машин впервые в 19 веке высказал ученый Чарльз Бэббидж. Его считают основателем и отцом первого в мире компьютера. На тот момент его идеи были лишь предположениями. Он не знал о мониторах и микросхемах, но ему удалось точно описать основные принципы работы вычислительных машин. Со временем инженеры сделали грандиозный вывод о том, что эффективное использование компьютеров заключается в правильно подобранных алгоритмах.

Пик развития языков программирования пришелся на 50-е годы 20 века. Тогда, чтобы программировать, инженерам приходилось знать полностью всю архитектуру компьютера и понимать его машинные коды. В 50-е ты не мог быть программистом, если не знал, как собирается компьютер и из чего он состоит. У компьютеров того времени даже клавиатуры не было, но были перфокарты, которые не очень упрощали будни тогдашнего инженера-программиста. Когда человеку нужно упростить свой труд, сделать его более автоматизированным, он начинает придумывать новые идеи и изобретать то, что могло бы облегчить ему жизнь. Так начали появляться первые языки программирования.В то время инженеры-программисты были вынуждены детально изучать все машинные команды, а потом внимательно писать код, каждый раз тщательно его проверяя. Чтобы как-то упростить задачу, программистам требовалось перевести цифровые обозначения операций в буквы. Так появился первый язык программирования – Ассемблер (от английского assemble – собирать). Сейчас его считают языком низкого уровня программирования, но в свое время он произвел настоящий фурор. Программы, написанные на Ассемблере,были достаточно эффективны и работоспособны.



Затем программистам понадобилось создать язык более высокого уровня, который бы ориентировался на написание и запуск алгоритмов. В 1954 году был изобретен новый язык программирования – FORmula TRANslator или просто FORTRAN. Он жив до сих пор и достаточно востребован в Data Science. Существует множество версий Фортрана, но первая и оригинальная из множества версий использовалась еще на перфокартах, как и Ассемблер. Затем внедрение новых типов цифровых устройств побудило программистов не стоять на месте и создавать новые языки машин.



После Фортрана в 1958 году изобрели Algorithmic Language или Algol. Его создали на основе обособленных блоков. Это алгоритмический язык, которым сейчас уже никто не пользуется. По сути его можно назвать мертвым. Однако, в 58 году он был очень востребованным, так как предназначался для научных отчетов и публикаций.

Стремление программистов преобразовать язык программирования в подобие человеческого языка повлияло на создание COBOL в 1959 году. Его синтаксис значительно отличался от Ассемблера и Фортрана и был похож на естественный английский. Это стало новым веянием в мире программирования, так как у программистов появилась возможность работать с текстом и записями, а не машинным кодом.

* Операционные системы

Операционная система — это самая главная программа на ПК, это интерфейс (мост) между «железом» в компьютерном блоке или корпусе телефона ([процессором](https://ktonanovenkogo.ru/voprosy-i-otvety/processor-chto-ehto-takoe.html" \t "/home/newuser/Documents\\x/_blank), памятью, жестким диском) и пользователем.

Она создана для управления всей работой компьютера, эффективного использования его ресурсов. ОС — это «Большой Босс», который выполняет следующие функции:

управляет свободными ресурсами оперативной памяти, жестких дисков или карт памяти;

руководит загрузкой процессора;

распределяет потоки процессов для создания многозадачности (одновременное открытие нескольких программ);

устанавливает, обновляет и удаляет (при помощи пользователя или автоматически) программы и приложения.

Для того чтобы пользователю было комфортно [взаимодействовать с программным обеспечением](https://ktonanovenkogo.ru/voprosy-i-otvety/programmnoe-obespechenie-po-chto-ehto-takoe.html" \t "/home/newuser/Documents\\x/_blank) (браузеры, игры и т.д.), операционные системы имеют графический интерфейс (графическую оболочку).



Работа операционной системы

Все вы слышали фразы «Загрузка компьютера», «Загрузка операционной системы» или что-то вроде этого. Под этими фразами понимается процесс, который происходит при нажатии на кнопку включения компьютера. Во время этого процесса, который занимает минуту или две, компьютер выполняет несколько важных функций:

Проверяет все программы, чтобы убедится, что все они работают правильно;

Проверяет наличие нового оборудования;

Запускается операционная система.

Как только операционная система запустилась, она управляет всем программным и аппаратным обеспечением компьютера.

Наиболее популярные ОС



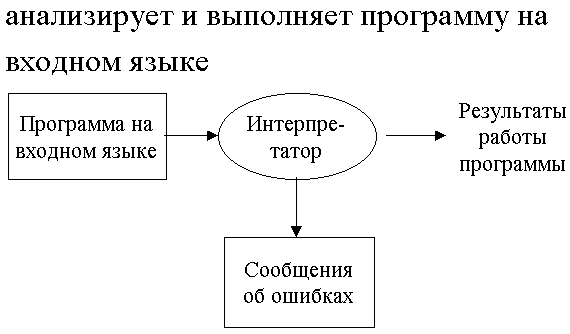
* Языки программирования



Можно писать программы непосредственно на машинном языке, хотя это и сложно. На заре компьютеризации(в начале 1950-х г.г.), машинный язык был единственным языком, большего человек к тому времени не придумал. Для спасения программистов от сурового машинного языка программирования, были созданы **языки высокого уровня** (т.е. немашинные языки), которые стали своеобразным связующим мостом между человеком и машинным языком компьютера. Языки высокого уровня работают через трансляционные программы, которые вводят "исходный код" (гибрид английских слов и математических выражений, который считывает машина), и в конечном итоге заставляет компьютер выполнять соответствующие команды, которые даются на машинном языке. Существует два основных вида трансляторов: интерпретаторы, которые сканируют и проверяют исходный код в один шаг, и компиляторы, которые сканируют исходный код для производства текста программы на машинном языке, которая затем выполняется отдельно.

* Компилятор и интерпритатор

**Интерпретаторы**



Одно, часто упоминаемое преимущество интерпретаторной реализации состоит в том, что она допускает "непосредственный режим". Непосредственный режим позволяет вам задавать компьютеру задачу вроде PRINT 3.14159\*3/2.1 и возвращает вам ответ, как только вы нажмете клавишу ENTER (это позволяет использовать компьютер стоимостью 3000 долларов в качестве калькулятора стоимостью 10 долларов). Кроме того, интерпретаторы имеют специальные атрибуты, которые упрощают отладку. Можно, например, прервать обработку интерпретаторной программы, отобразить содержимое определенных переменных, бегло просмотреть программу, а затем продолжить исполнение.

Больше всего программистам нравится в интерпретаторах возможность получения быстрого ответа. Здесь нет необходимости в компилировании, так как интерпретатор всегда готов для вмешательства в вашу программу. Введите RUN и результат вашего самого последнего изменения оказывается на экране.

Однако интерпретаторные языки имеют недостатки. Необходимо, например, иметь копию интерпретатора в памяти все время, тогда как многие возможности интерпретатора, а следовательно и его возможности могут не быть необходимыми для исполнения конкретной программы.

Слабо различимым недостатком интерпретаторов является то, что они имеют тенденцию отбивать охоту к хорошему стилю программирования. Поскольку комментарии и другие формализуемые детали занимают значительное место программной памяти, люди стремятся ими не пользоваться. Дьявол менее яростен, чем программист, работающий на интерпретаторном Бейсике, пытающийся получить программу в 120К в памяти емкостью 60К. но хуже всего то, что интерпретаторы тихоходны. Ими затрачивается слишком много времени на разгадывание того, что делать, вместо того чтобы заниматься действительно делом.

При исполнении программных операторов, интерпретатор должен сначала сканировать каждый оператор с целью прочтения его содержимого (что этот человек просит меня сделать?), а затем выполнить запрошенную операцию. Операторы в циклах сканируются излишне много.

Рассмотрим программу: на интерпретаторном Python3:

While True:

If True==1:

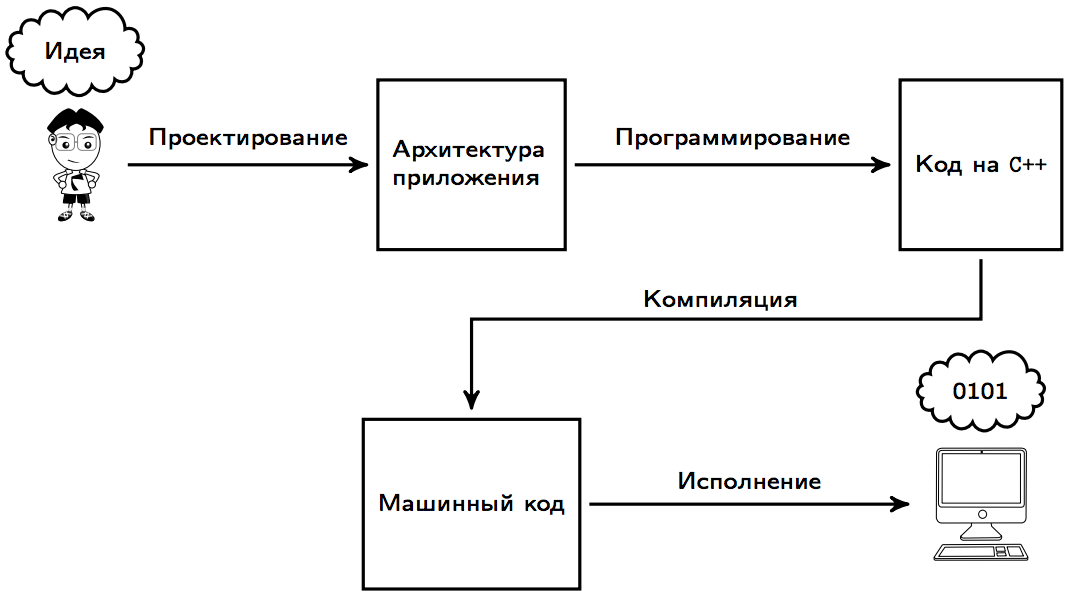
Print(“Верно”)

Else:

Print(‘’Неверно’’)

При втором проходе цикла все это разгадывание повторяется снова, так как абсолютно забыты все результаты изучения этой строки какую-то миллисекунду тому назад. И так во всех следующих безконечных проходах. Совершенно очевидно, что если вам удалось каким-то образом отделить фазу сканирования/понимания от фазы исполнения вы имели бы более быструю программу. И это как раз то, для чего существуют компиляторы.

**Компиляторы**



Компилятор-это транслятор текста на машинный язык, который считывает исходный текст. Он оценивает его в соответствии с синтаксической конструкцией языка и переводит на машинный язык. Другими словами, компилятор не исполняет программы, он их строит. Интерпретаторы невозможно отделить от программ, которые ими прогоняются, компиляторы делают свое дело и уходят со сцены. При работе с компилирующим языком, таким как Турбо-Бейсик, вы придете к необходимости мыслить о ваших программах в признаках двух главных фаз их жизни: периода компилирования и периода прогона. Большинство программ будут прогоняться в четыре - десять раз быстрее их интерпретаторных эквивалентов. Если вы поработаете над улучшением, то сможете достичь 100-кратного повышения быстродействия. Оборотная сторона монеты состоит в том, что программы, расходующие большую часть времени на возню с файлами на дисках или ожидание ввода, не смогут продемонстрировать какое-то впечатляющее увеличение скорости.

* Бэк и Фронт

Любое веб приложение состоит из двух частей: Frontend и Backend. Соответственно и программистов делят также, по сути это две несвязанные между собой касты разработчиков.

**Фронтэнд**

Frontend – это разработка пользовательского интерфейса и функций, которые работают на клиентской стороне веб-сайта или приложения. Это всё, что видит пользователь, открывая веб-страницу, и с чем он взаимодействует.

Frontend-разработчик сотрудничает с дизайнерами, программистами  
и UX-аналитиками, чтобы создавать удобный и востребованный продукт.

Чтобы наглядно понять frontend-разработку, откройте страницу любого сайта – перед собой вы увидите интерфейс.



Компоненты frontend-разработки

* **HTML (HyperText Markup Language)** – язык разметки документов для создания структуры страницы: заголовки, абзацы, списки и так далее.
* **CSS (Cascading Style Sheets)** – язык для описания и стилизации внешнего вида документа. Благодаря CSS-коду браузер понимает, как именно отображать элементы. CSS задаёт цвета и параметры шрифтов, определяет, как будут располагаться разные блоки сайта, и так далее. Ещё он позволяет выводить один и тот же документ в разных стилях, например, для печати (обычной или шрифтом Брайля), вывода передачи на экран или чтения голосом.
* **JavaScript** – это язык, который создавался, чтобы оживить веб-страницы. Его задача–реагировать на действия пользователя, обрабатывать клики мышкой, перемещения курсора, нажатия клавиш. Ещё он посылает запросы на сервер и загружает данные без перезагрузки страницы, позволяет вводить сообщения и многое другое.

**Backend**

Backend-разработка – это набор аппаратно-программных средств, при помощи которых реализована логика работы сайта. Попросту говоря, это то, что скрыто от глаз пользователя и происходит вне его браузера и компьютера.

Например, когда вы вводите запрос на странице поисковика и жмёте клавишу Enter, frontend заканчивается и начинается backend. Ваш запрос отправляется на сервер Google или «Яндекса», где расположены алгоритмы поиска. Именно там случается всё «волшебство». Как только на мониторе появилась информация, которую вы искали, – вновь происходит возвращение в зону frontend.

По большому счёту, сервер – это тот же компьютер, только более мощный. Он хранит данные и отвечает на запросы пользователей.

Компоненты backend-разработки

Backend-разработчик применяет те инструменты, что доступны на его сервере. Он вправе выбрать любой из универсальных языков программирования, например, Ruby, PHP, Python, Java. Всё зависит от конкретного проекта и задачи заказчика.

Также для backend-разработки используются системы управления базами данных:

* MySQL;
* PostgreSQL;
* SQLite;
* MongoDB.

В зависимости от продукта обязанности backend-разработчика сильно меняются. Такой специалист может создавать и интегрировать базы данных, обеспечивать безопасность или настраивать технологии резервного копирования и восстановления.

**Как взаимодействуют frontend и backend?**

Взаимодействие *frontend* и *backend* происходит по кругу:

* *frontend* отправляет пользовательскую информацию в *backend*;
* информация обрабатывается;
* и возвращается обратно, приняв понятную форму.

Этими работами занимаются разные специалисты, но каждому из них желательно понимать принципы, по которым работают коллеги. Даже дизайнеру интерфейсов важно хотя бы в общих чертах знать, что представляет собой backend проекта, которым он занимается. Это поможет адекватно оценить, какие технические возможности есть у сайта или приложения.

Существует несколько вариантов взаимодействия *frontend*и*backend*:

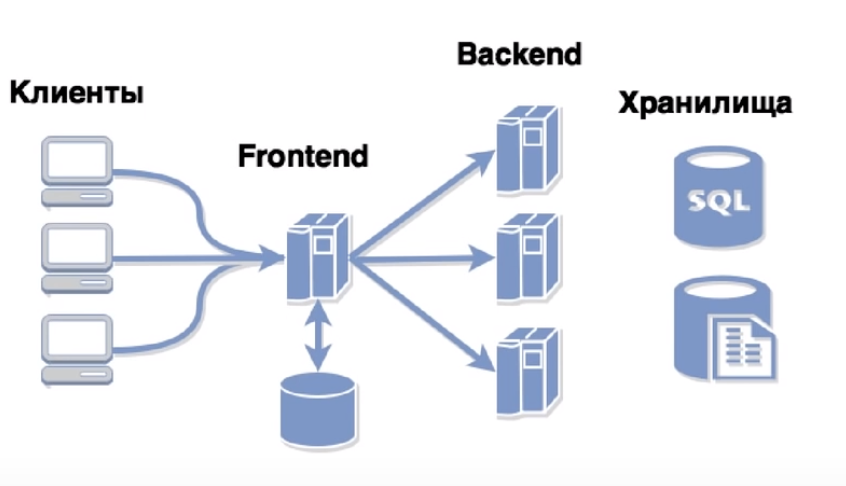
Сегодня существует несколько основных архитектур, определяющих, как будут взаимодействовать ваши бэкенд и фронтенд.

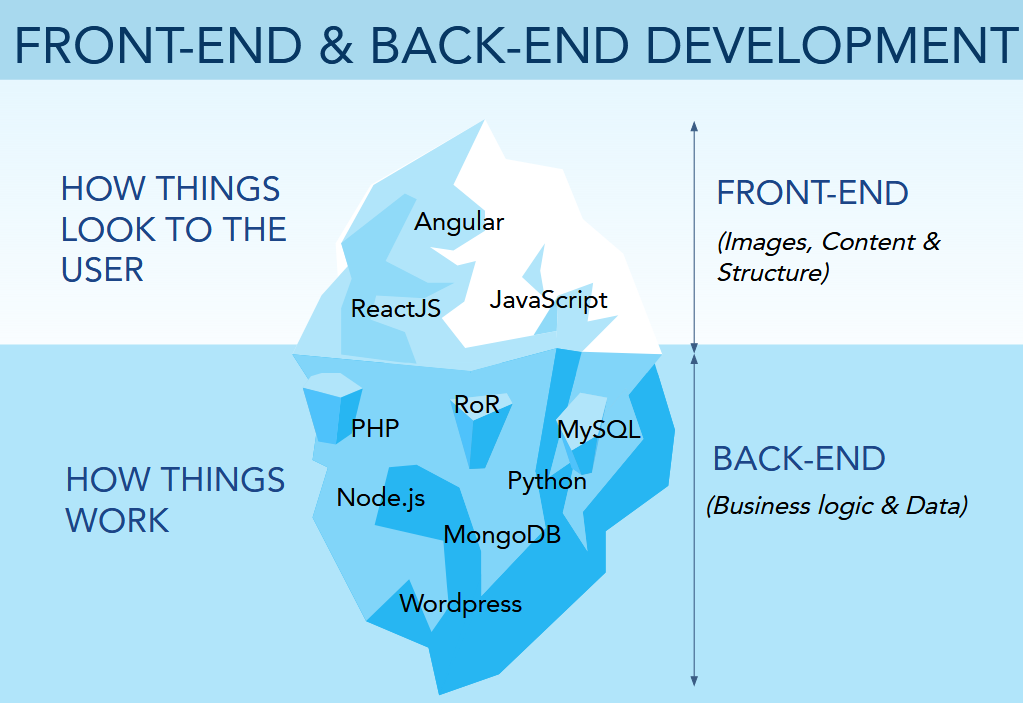
* **Серверные приложения** –HTTP-запросы отправляются напрямую на сервер приложения, а сервер отвечает HTML-страницей.Между получением запроса и ответом сервер обычно ищет по запросу информацию в базе данных и встраивает ее в шаблон (ERB, Blade, EJS, Handlebars).

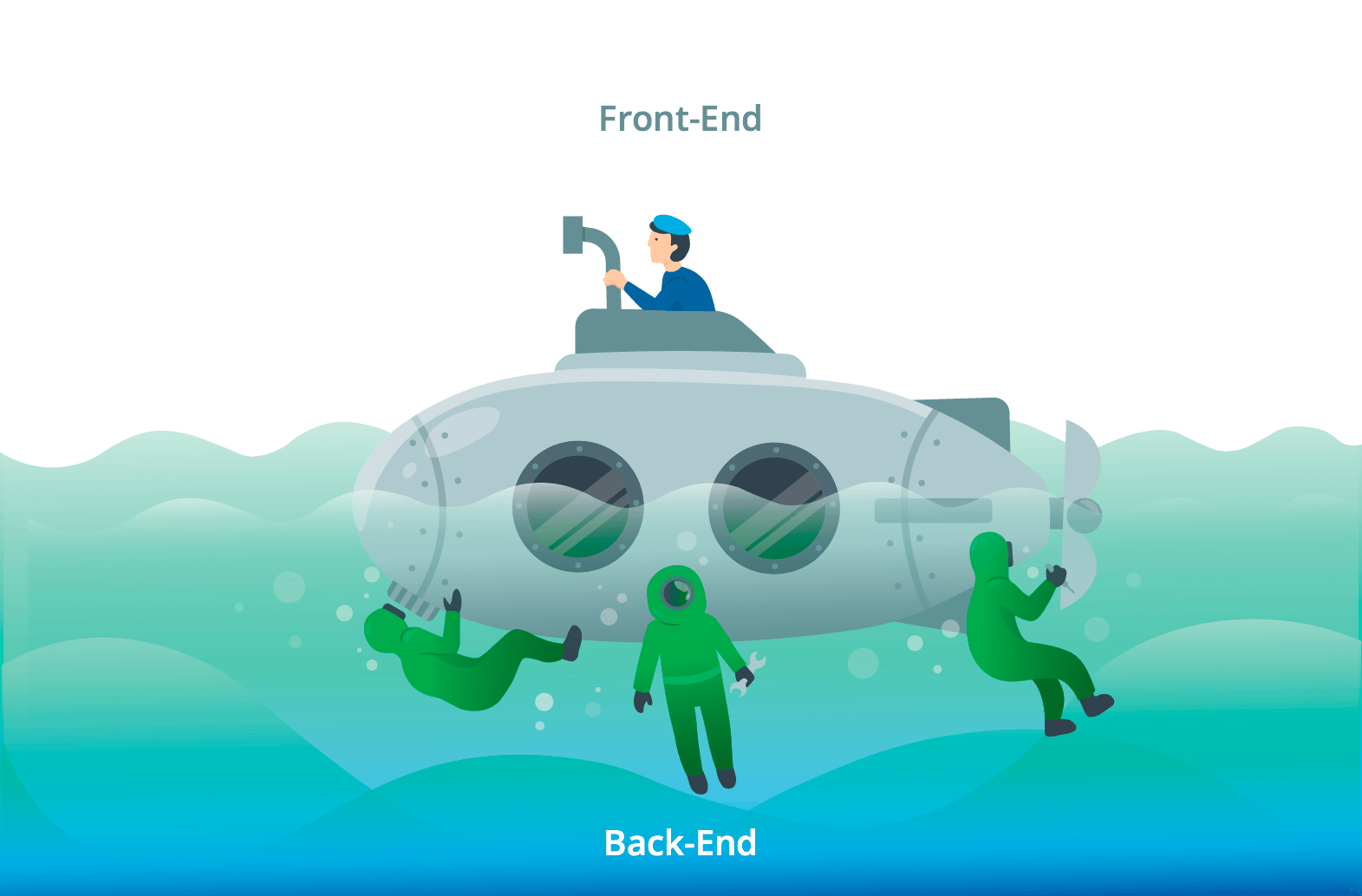
Когда страница загружена в браузере, HTML определяет, что будет показано, CSS – как это будет выглядеть, а JS – всякие особые взаимодействия.

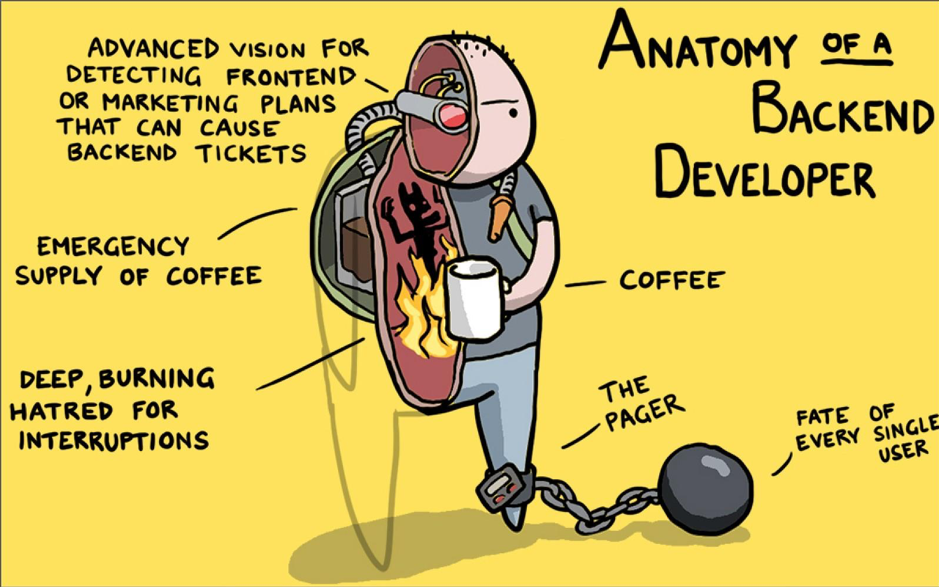
* **Связь с использованием AJAX** –(Asynchronous JavaScript and XML – асинхронный JavaScript и XML). Это означает, что JavaScript, загруженный в браузере, отправляет HTTP-запрос (XHR, XML HTTP Request) изнутри страницы и (так сложилось исторически) получает XML-ответ. Сейчас для ответов также можно использовать формат JSON. Это значит, что у вашего сервера должна быть конечная точка, которая отвечает на запросы JSON- или XML-кодом. Два примера протоколов, используемых для этого – REST и SOAP.
* **Клиентские (одностраничные) приложения** –AJAX позволяет вам загружать данные без обновления страницы. Больше всего это используется в таких фреймворках, как Angular и Ember. После сборки такие приложения отправляются в браузер, и любой последующий рендеринг выполняется на стороне клиента (в браузере). Такой фронтенд общается с бэкендом через HTTP, используя JSON- или XML-ответы.
* **Универсальные/изоморфные приложения** – **н**екоторые библиотеки и фреймворки, например, React и Ember, позволяют вам исполнять приложения как на сервере, так и в клиенте. В этом случае для связи фронтенда с бэкендом приложение использует и AJAX, и обрабатываемый на сервере HTML.

Обязанности frontend- и backend-разработчиков, как правило, разделены, но бывают моменты, когда программист решает проблемы как на стороне сервера, так и в клиентской части. Оба вида разработки подразумевают и технические, и творческие компоненты. Нередко на рынке встречаются специалисты, которые уверенно чувствуют себя как во frontend, так и в backend и могут совмещать их.

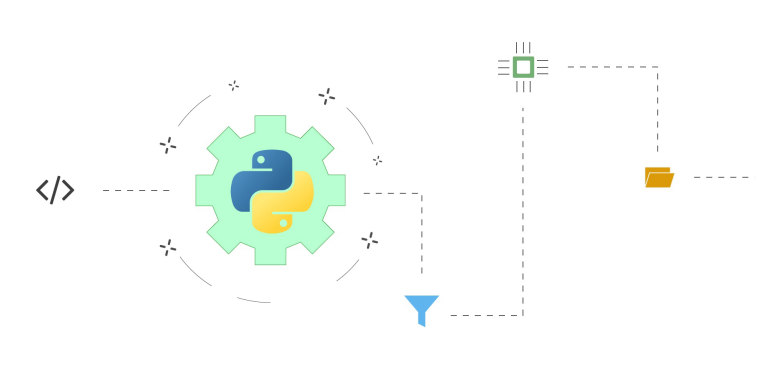




****

****

* Python3



Python – это универсальный современный ЯП высокого уровня, к преимуществам которого относят высокую производительность программных решений и структурированный, хорошо читаемый код.  Синтаксис Питона максимально облегчен, что позволяет выучить его за сравнительно короткое время. Ядро имеет очень удобную структуру, а широкий перечень встроенных библиотек позволяет применять внушительный набор полезных функций и возможностей. ЯП может использоваться для написания прикладных приложений, а также разработки WEB-сервисов.

Python может поддерживать широкий перечень стилей разработки приложений, в том числе, очень удобен для работы с ООП и функционального программирования.Один из самых популярных интерпретаторов языка –  CPython, написанный [на Си](https://techrocks.ru/2019/01/20/migration-from-c-plus-plus-to-python/). Распространяется эта среда разработки бесплатно по свободной лицензии. Интерпретатор поддерживает большинство популярных платформ.

Типы данных:

Синтаксис Python не подразумевает обязательное определение типа данных для переменных, констант, массивов, списков и т.д. Основные типы ничем не отличаются от других языков с жестко заданной типизацией.

Самые важные типы:

Числовые: целые, дробные, вещественные с плавающей точкой, комплексные.

Логические: тип для хранения значений алгебры логики – «истина» или «ложь».

Строковые: содержат символы Юникода, в том числе, html-код.

Списки – упорядоченные массивы переменных.

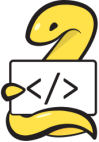
Кортежи – массив упорядоченных констант, т.е. значений, которые не могут изменяться в процессе работы.

Множества – массивы неупорядоченных данных.

Словари – специализированный массив, состоящий из пары – «ключ» — «значение».

Байты, массивы байтов – поименованные области памяти для хранения изображений (jpg, gif и т.д.), pdf-документов и других файлов.

Здесь объектами считаются практически все конструкции. А потому собственные типы имеют, в том числе, классы, файлы или методы.



Краткий перечень возможностей:

Любой описанный класс единовременно представляет из себя и

* объект.
* Функция множественного наследования.
* Поддержка виртуальных функций.
* Возможность легко управлять именами скрывать их особыми метками.
* Возможность жизнью объекта и распределение памяти.
* Управление работы операторов как символьных, так и логических.
* Возможность имитировать поле
* Управление полями – как прямой, так и частичный доступ.
* Контроль над самыми распространенными операциями. От глубокого до итерации по объекту.
* Возможно создавать триггеры и классы.

Разработка языка Python была начата в конце [1980-х годов](https://ru.wikipedia.org/wiki/1980-%D0%B5" \o "1980-е) сотрудником голландского института CWI [Гвидо ван Россумом](https://ru.wikipedia.org/wiki/%D0%93%D0%B2%D0%B8%D0%B4%D0%BE_%D0%B2%D0%B0%D0%BD_%D0%A0%D0%BE%D1%81%D1%81%D1%83%D0%BC" \o "Гвидо ван Россум). Для распределённой ОС [Amoeba](https://ru.wikipedia.org/wiki/Amoeba_(%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0)" \o "Amoeba (операционная система)) требовался расширяемый [скриптовый язык](https://ru.wikipedia.org/wiki/%D0%A1%D1%86%D0%B5%D0%BD%D0%B0%D1%80%D0%BD%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA" \o "Сценарный язык), и Гвидо начал писать Python на досуге, позаимствовав некоторые наработки для языка [ABC](https://ru.wikipedia.org/wiki/ABC_(%D1%8F%D0%B7%D1%8B%D0%BA_%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)" \o "ABC (язык программирования)) (Гвидо участвовал в разработке этого языка, ориентированного на обучение программированию). В феврале [1991 года](https://ru.wikipedia.org/wiki/1991_%D0%B3%D0%BE%D0%B4" \o "1991 год) Гвидо опубликовал исходный текст в [группе новостей](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D1%83%D0%BF%D0%BF%D0%B0_%D0%BD%D0%BE%D0%B2%D0%BE%D1%81%D1%82%D0%B5%D0%B9" \o "Группа новостей) alt.sources. С самого начала Python проектировался как [объектно-ориентированный язык](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%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" \o "Объектно-ориентированный язык программирования).

Наличие дружелюбного, отзывчивого сообщества пользователей считается, наряду с дизайнерской интуицией Гвидо, одним из факторов успеха Python. Развитие языка происходит согласно чётко регламентированному процессу создания, обсуждения, отбора и реализации документов PEP ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA" \o "Английский язык) Python Enhancement Proposal) — предложений по развитию Python.

[3 декабря](https://ru.wikipedia.org/wiki/3_%D0%B4%D0%B5%D0%BA%D0%B0%D0%B1%D1%80%D1%8F" \o "3 декабря) [2008 года](https://ru.wikipedia.org/wiki/2008_%D0%B3%D0%BE%D0%B4" \o "2008 год), после длительного тестирования, вышла первая версия Python 3000 (или Python 3.0, также используется [сокращение](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D0%B1%D1%80%D0%B5%D0%B2%D0%B8%D0%B0%D1%82%D1%83%D1%80%D0%B0" \o "Аббревиатура) Py3k). В Python 3000 устранены многие недостатки архитектуры с максимально возможным (но не полным) сохранением совместимости со старыми версиями Python. На сегодня поддерживается одна ветка развития (Python 3.x), поддержка ветки Python 2.x закончилась в апреле 2020 года.

* Установка и настройка IDE

# Что такое IDE и редакторы кода

IDE (или интегрированная среда разработки) — программа, предназначенная для разработки программного обеспечения. Как следует из названия, IDE включает в себя инструменты, специально предназначенные для разработки программного обеспечения.

Стандартно инструменты содержат :

1. Редактор, предназначенный для обработки кода (например, с подсветкой синтаксиса и автозавершением)
2. Средства сборки, выполнения и отладки
3. Систему контроля версий

Большинство IDE поддерживают множество языков программирования и содержат другие дополнительные функции. Поэтому они требуют большое количество времени для загрузки и установки.

# Требования к среде программирования на Python

Итак, что нужно знать о среде программирования. Списки функций варьируются от приложения к приложению, но есть основной набор функций, который упрощает разработку:

* Сохранять и перезагружать файлы  
  IDE или редактор позволят сохранить работу и открыть ее позже, в том же состоянии, в котором она была до закрытия.
* Запустить код в среде программирования  
  С помощью IDE [запустить код Python](https://PythonRu.com/baza-znanij/gde-skachat-python-i-kak-ego-ustanovit-na-linux-mac-os-windows" \t "/home/newuser/Documents\\x/_blank), будет не сложнее, чем из простого текстового редактора.
* Поддержка отладки  
  Возможность проверить код во время перед запуском — особенностью всех IDE и прочих редакторов кода.
* Подсветка синтаксиса  
  Определение ключевых слов, переменных и символов в коде делает чтение и понимание кода намного проще.
* Автоматическое форматирование кода  
  Любой редактор или IDE будет распознавать двоеточие в конце for или while.

Есть много других функций: управление исходным кодом, модель расширения, инструменты сборки и тестирования, помощь с [синтаксисом языка](https://pythonru.com/uroki/sintaksis-python-uroki-po-python-dlja-nachinajushhih" \t "/home/newuser/Documents\\x/_self) и другие. Но приведенный выше список — основные функции, которые поддерживает хорошая среда редактирования.

### Наш выбор PyCharm

* Режим консоли и запуск файлов

У разработчиков на Python есть возможность использовать его в интерактивном режиме. Введенные с клавиатуры операторы сразу же выполняются и результат выводится на экран (в консоль). Такой режим интерпретатора называется REPL (read-eval-print loop). Работает он следующим образом:

1. read считывает одно выражение и преобразует его в соответствующую структуру данных;
2. eval принимает одну такую структуру и вычисляет соответствующее выражение;
3. print принимает результат и печатает его на экран (в консоль).

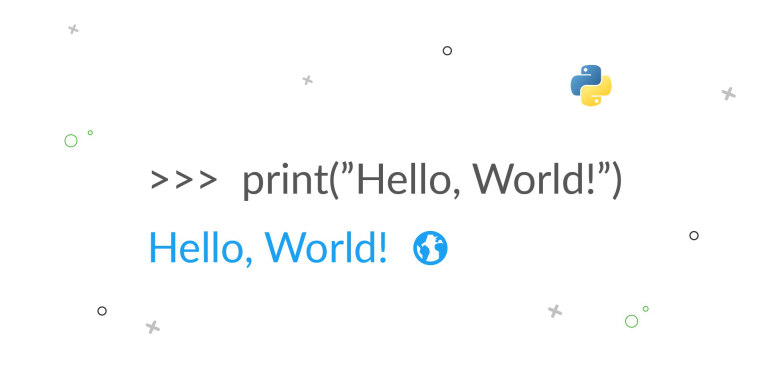
Грубо говоря, в этом режиме вам не нужно создавать файлы с python-кодом. Все что вы напишете в интерактивном режиме будет сразу же выполнено построчно.

Интерактивный режим используется редко, но удобен в ряде случаев:

* при изучении языка;
* для быстрого тестирования отдельных фрагментов кода;
* для быстрого запуска простых команд (например для запуска нескольких bash скриптов в цикле через Python);
* для использования интерпретатора в качестве продвинутого калькулятора;

Второй вариант – использовать среду разработки IDLE. После запуска IDLE появится окно оболочки Python (Python Shell), в котором можно работать с Python в интерактивном режиме.

* Hello World!



3. Подведение итогов занятия, ответы на возникающие вопросы

4. Выдача домашнего задания ученикам:

* Виды типизации данных
* Типы данных в Python3
* Развертывание IDE на персональном компьютере