ЦИФРОВАЯ КУЛЬТУРА
УНИВЕРСИТЕТ ИТМО

# Технологии программирования

Высшая школа цифровой культуры
Университет ИТМО
dc@itmo.ru

# Содержание

# 1 Введение в технологии программирования

Здравствуйте! Рады приветствовать вас в курсе «Цифровая культура» и на этой неделе мы доступно и информативно расскажем о неотъемлемом компоненте этого направления, а именно о современных технологиях программирования. Для понимания картины в целом и дальнейшего изучения материалов необходимо ответить на несколько вопросов: чем же все-таки занимаются программисты, как именно они это делают и самое главное для чего это вообще нужно.

Начнем с того, как профессия программиста воспринимается в обществе. В интернете существует достаточно большое количество шуток на тему того, чем на самом деле занимаются специалисты в области программирования. К разочарованию большинства рядовых пользователей, не особо вникающих в детали технологий программирования, область IT слишком обширна и отдельно взятый специалист обычно не может разбираться во всех тонкостях. При этом получение навыков ремонта бытовой техники вообще не предусмотрены в образовательной программе подготовки программистов. Такое искаженное восприятие достаточно прочно закрепилось в обществе. И для понимания реального положения дел как раз-таки необходимо иметь хотя бы общие представления о современной цифровой культуре. Для успешной реализации различных проектов в области разработки программного обеспечения необходимо, чтобы все участники этого процесса обладали хотя бы базовыми навыками в этой области. Т.е. даже если в обязанности сотрудника не входит разработка ПО, он должен уметь поставить перед программистом техническое задание, грамотно формулируя задачи своей предметной области.

С другой же стороны хороший специалист в области разработки обязан обладать рядом определенных качеств для успешной деятельности, а именно:

1. определенный склад ума для формализации и решения прикладных задач;
2. желание постоянно совершенствоваться и изучать новые технологии;
3. работать в коллективе;
4. обладать определенной креативностью;
5. грамотно и понятно излагать свои идеи.

Что же такое программирование? Для понимания этого вопроса обратимся к простой задаче, с который сталкивался каждый из вас – приготовление бутерброда. Попробуйте описать все действия, необходимые для его приготовления, учтите все нюансы.

А теперь представим, что вы уронили нож, испачкались, какова дальнейшая последовательность действий? Такой сценарий рассматривался? Таким образом можно говорить о том, что программирование – это точное описание всех действий и возможных событий на которые необходимо реагировать. Программист же должен уметь написать такую последовательность инструкций, разбить сложные задачи на более простые составляющие и учесть дополнительные параметры чтобы не переделывать решение каждый раз. В отличии от бытовых задач, этапы разработки программного обеспечения редко заранее репетируют, а программисты подходят к решению составляющих задач по отдельности и тестируют их работоспособность. Для передачи таких инструкций и существуют различные языки программирования, которые интерпретируют те или иные команды в двоичный код понятный компьютеру. По сути язык программирования мало чем отличается от любого языка, на которым мы говорим и сводится к знанию правил написания, конструкций и согласования, без которых нас не поймут вовсе, или поймут неправильно.

Чему нас учит программирование и зачем уметь программировать? Один из ответов - мыслить, ведь компьютеры и программы созданы человеком, а изучая их язык и логику мы все больше узнаем о самом обществе, которое функционирует во многом благодаря сложным технологиям. Все больше процессов и задач выполняется с помощью автоматизированных средств, роботов и программных комплексов.

Также программирование помогает нам развиваться. Как люди изучают иностранный язык, в том числе самостоятельно, то теперь и языки программирования, которые развивают логику и мышление. Все это позволяет нам быть более конкурентоспособными в различных сферах деятельности, в том числе где изначально не требуются навыки программирования.

Все это приводит нас к конечной цели, программирование и как следствие программы позволяют нам экономить время. Развитие технологий позволяет автоматизировать все больше процессов, с которыми сталкивается любой человек. К примеру, разослать приглашения на свадьбу лично всем по почте займет длительное время, а программа автоматической рассылки сделает это за секунды. Именно поэтому так популярны различные боты.

Развитие технологий программирования также облегчает и их изучение. Все больше можно видеть истории из развивающихся стран, так в Индии 12-летние девочки сделали приложение для онлайн-очереди за водой, чтобы не тратить больше времени на ожидание. Таким образом программирование в том или ином виде используется почти повсеместно и становится частью нашей культуры. В последующий лекциях мы уделим внимание различным языкам программирования, их классификации и безопасности.

# 2 Выбор первого языка

В этой лекции мы постараемся классифицировать языки программирования по различным сферам деятельности и помочь вам, слушателям, с выбором своего первого языка или просто сориентироваться в мире современных технологий. В любом случае, затрагивая тему программирования, стоит задаться вопросом, для каких целей вы собираетесь его изучать. Выбирая первый язык, стоит внимательно отнестись к оценке следующих факторов:

1. рынок труда;
2. долгосрочные перспективы языка;
3. сложность изучения языка;
4. что именно вы сможете создать в процессе изучения, и, показать окружающим, поддержав тем самым свою мотивацию.

Прежде чем говорить о каких-либо языках программирования, уточним, что Ваш путь индивидуален, и можно долго спорить о том, что некий язык объективно лучше любого другого. Тем более, разработчики, в конечном счёте, изучают больше одного языка.

Многое конечно зависит от направления вашей подготовки, кто-то пройдет весь путь и познает основы всех языков, кто-то будет изучать программирование в рамках предметов по информационным технологиям, кто-то захочет изучить многое самостоятельно. В любом случае программирование и личностное развитие в рамках этого направления связано с самоконтролем и саморазвитием.

С чего же начать?

- Первое направление, актуальное в современном цифровом обществе, это если вы хотите изучить язык программирования ради своего развития, ради интереса и понимания данной сферы. Самым простым в изучении принято считать Python. Если вас не смущают трудности на пути, и вы серьезно нацелены, то можно выбрать между Java и C. Выбор в чем то схож с тем, как автолюбители выбирают между автоматической и ручной коробкой передач. Выбирая между удобством и возможно некоторыми ограничениями, и между тонкой настройкой и возможностью держать все под своим контролем.

- Второе направление так или иначе будет связано с карьерой и уже вашими предпочтениями в рамках платформы и сферы:

    1. Веб – состоит из двух направлений фронтэнд и бэкэнд:

   (a) Фронтенд-разработчик – иногда с этим связана некоторая путаница среди как вакансий, так и в целом понимания чем должен заниматься специалист в этой области. А отметить можно два типа таких разработчиков и чем они занимаются: верстальщик и фронтенд-разработчик. Верстальщик боец узкого фронта и его задача сверстать полученный от дизайнера макет, используя HTML+CSS. Фронтенд-разработчик не просто верстает макеты. Он хорошо знает JavaScript, разбирается во фреймворках и библиотеках, понимает, что происходит на серверной стороне. Его не пугают препроцессоры и сборщики LESS, SASS, GRUNT он умеет работать с DOM, API, SVG-объектами, AJAX может составлять SQL-запросы и работать с данными.

   (b) Бэкенд-разработчик же отвечает за серверную часть любого веб-приложения. Из основных технологий используемых для этих целей можно выделить три языка: Python, Ruby и PHP. Если проводить аналогии, то если вам нравится идея конструктора Лего и создавать все из готовых различных кубиков выбирайте Python, если вам нравится творить и создавать самому как из пластилина – Ruby, или останавливайтесь на языке PHP – он как старая игрушка, но которая вам дорога из детства.

2. Игры. 2д и 3д гейминг на различных платформах – как правило связан с языком разработки C++, но все чаще разработчики используют игровые движки и как следствие предлагаемые ими инструменты и поддерживаемые языки программирования. К примеру при разработке в Юнити можно использовать C# или встроенный язык UnityScript.

3. Разработка мобильных приложений как очень актуальное направление будет рассмотрено в отдельной лекции, но отметим что язык в основном зависит от платформы, Java под Android или Objective-C/Swift под iOS.

4. Крупные компании как правило работают в ярко выраженном направлении, и если вы хотите работать в одной из таких, следует уделить внимание определенным языкам программирования, например Google, Facebook – Python, Microsoft – C#, Apple - Objective-C/Swift.

5. Если вы преследуете единственную цель, высокую заработную плату, то можно попробовать свои силы в Java разработке и стажировках от крупных корпораций и фирм.

Многие изучают конкретные языки программирования если их интерес связан с программным обеспечением. Проведем краткий обзор:

1. Google Chrome - Популярный браузер написан на C++, Assembly и Python.
2. Adobe - Все продукты написаны на C/C++, в интерфейсах используется JavaScript.
3. Word, Excel и Powerpoint написаны на C/C++.
4. Windows 10 использовался C++, а ядро операционной системы работает на C.
5. Mac Os X написана на Objective C, а ядро, как и в предыдущем случае, на C.
6. Quora, Reddit, Dropbox – серверная часть написана на Python, интерфейс использует JavaScript.

В целом можно говорить о ряде языков программирования, которые так или иначе являются самыми популярными и чаще всего используемыми в различных сферах, это подтверждают и различные исследования, проводимые как поисковыми системами, так и крупными аналитическими ресурсами, вроде tiobe.com. Где в пятерку самых популярных языков входят Java, C, C++, Python, C#, а большинство сервисов и программного обеспечения используют JavaScript в своих интерфейсах.

## 3 Описание языков

Остановимся подробней на самих языках программирования и их особенностях. Языки принято классифицировать на две группы, низкоуровневые и высокоуровневые.

К низкоуровневым языкам программирования относится машинный код – система команд конкретной вычислительной машины в двоичном коде. Для сокращения записи часто представляется в шестнадцатеричной системе счисления. Для удобства программирования используют символьные мнемоники с некоторыми дополнительными возможностями, что называется Ассемблером (или так называемый класс языков Assembly).

Ассемблер – язык низкого уровня абстракции, прямо отражающий все тонкости и нюансы работы с железом. В итоге получается, что для написания даже простой программы на ассемблере требуется весьма большое количество предварительных знаний – «порог вхождения» здесь намного выше,

чем для языков высокого уровня. Язык знаменит тем, что написать код обычно быстрее, чем разобраться в существующем. На языке ассемблера пишут программы или их фрагменты в тех случаях, когда критически важны:

- объем используемой памяти – это могут быть программы-загрузчики, встраиваемое программное обеспечение, программы для микроконтроллеров и процессоров с ограниченными ресурсами, вирусы, программные защиты;

- либо важно быстродействие – однако оно зависит от понимания того, как работает конкретная модель процессора, тонкостей работы операционной системы. В результате, такие программы работают быстрее, но теряется переносимость и универсальность.

Вторая группа языков – это Высокоуровневые языки программирования.

В таких языках вводятся абстракции и конструкции, кратко описывающие структуры данных и операции над ними. Высокоуровневые языки позволяют транслировать и исполнять код на разных операционных системах и разном оборудовании. Что возможно благодаря компиляции, то есть трансляции программы с высокоуровневого на низкоуровневый язык, близкий к машинному коду без ее выполнения, или интерпретации, когда код обрабатывается и сразу исполняется.

Рассмотрим такие языки:

1. Язык Си уникален с той точки зрения, что именно он стал первым языком высокого уровня, всерьёз потеснившим ассемблер в разработке системного программного обеспечения.

   Си отличается минимализмом, код можно легко писать на низком уровне абстракции, почти как на ассемблере, поэтому его часто называют языком среднего или даже низкого уровня, учитывая то, как близко он работает к реальному железу. Также Си называют «ассемблером высокого уровня», так как язык ассемблера различается для разных платформ, а стандарт языка Си позволяет компилировать код без изменений практически на любой модели компьютера.

   По причине того, что язык Си содержит в себе только наиболее близкие к машинным инструкциям команды, разработка занимает длительный процесс, однако результат будет отзывчивым и быстрым.

   Работы для Си программистов немало, в основном связанной с разработкой операционных систем для медиа приставок, телевизоров и прочих устройств или узкоспециализированного софта в секторе корпораций, например облачные технологии, защита данных и так далее.

Такие компании обычно довольно крупные (Dell, EMC, Dr.Web) с долгосрочными и высокооплачиваемыми проектами. Однако большинство вакансий для состоявшихся программистов и новичкам крайне сложно начать карьеру.

2. Язык C++ унаследован от языка Си. Одним из принципов его разработки было сохранение совместимости с Си и быстроты работы. Тем не менее, C++ не является в строгом смысле надмножеством Си, но изучаются как правило они именно в таком порядке.

   Главным отличием является поддержка объектно-ориентированного программирования. Это подход, когда еще на этапе проектирования все элементы будущего приложения рассматриваются как объекты с некоторыми свойствами и методами – то есть действия которые может делать объект или которые можно совершать над объектом. Например котенок – это объект, его свойства: порода, кличка, цвет, а его методы: спать, кушать, играть.

   C++ задает современные стандарты, в него в первую очередь добавляются новые специфики и потом уже они распространяются по различным языкам. Компиляторы этого языка преобразуют код в машинный код для процессора что сохраняет быстроту работы и эффективное использование ресурсов вычислительной машины.

   Однако есть и сложности, C++ не зря сравнивают с ручной коробкой передач в автомобиле. Программисту необходимо самостоятельно следить за разными вещами, но главное это выделение памяти, ее адресация и очистка.

   Работа, связанная с языком C++ также, как и Си связана с разработкой операционных систем, драйверов устройств, но также и прикладными программами, и играми. В сути своей чаще требуются специалисты со знанием именно связки языков C/C++. Таким образом знания этих двух языков программирования является более конкурентным на рынке труда.

3. Java изначально поддерживает только объектно-ориентированную разработку. Главной особенностью в отличие от C++ является то, что приложения транслируются в специальный код виртуальной машины Java, которая может быть запущена на любой компьютерной архитектуре. Также это повышает безопасность, так как любые превышения полномочий программы вызовут остановку виртуальной машины.

   Применение виртуальной машины Java влечет и ряд недостатков, а именно:

- большой размер программы в памяти, что связано с тем, что каждая программа запускается в рамках другой программы или виртуальной машины;
- скорость работы программы страдает, так как программа выполняется на виртуальной машине, которая уже выполняется на реальном процессоре.

Java облегчает и некоторые аспекты разработки, например работу с памятью, например не нужно заботится о том, чтобы освобождать выделенную память, так как существует механизм ее очистки.

С точки зрения трудоустройства, Java является очень обширной сферой, где большое число вакансий и претендентов. Из основных направлений можно выделить:

- мобильная разработка – ОС Android работает именно на Java;
- веб-приложения – большое количество мобильных версий веб-сайтов написано именно на Java;
- десктопное ПО – широчайший диапазон применения от игр (Minecraft) до офисных программ (Open Office);
- финансовая область – такие гиганты, как Barclays, Citigroup, Goldman Sachs активно используют Java для внутренних электронных систем и в качестве инструмента для создания пользовательских приложений;
- большие данные – очень актуальное на сегодняшний день направление, связанное с обработкой, прогнозированием и моделированием информации.

Существует большое число стажировок от крупных компаний, так что начать карьеру не так сложно, но всегда требуются знания языка и выполнение различных тестовых заданий.

4. C# является, как и Java чистым ООП языком, и, в частности, схож с ним в 80% случаев. В отличие от Java не сильно нагружен различными функциями. Также основан на виртуальной машине .Net от Microsoft или mono для Windows, Linux и macOS. Обычно используется в качестве инструмента для создания корпоративных систем, приложений Windows и веб-сайтов на платформе .Net.

   Характерной особенностью является наличие различных библиотек для создания интерфейса приложения в отличие от Java, что позволяет реализовать приложение в рамках одной технологии.

Работа так или иначе связана с корпоративным сектором, в частности Microsoft или другими крупными корпорациями. Либо другими компаниями, работающими с базами данных, и потоками информации.

5. Python – один из наиболее популярных «неклассических» языков программирования и продолжающий набирать популярность. Он идеально подходит на роль «вводного» языка, все чаще используется в качестве первого при изучении программирования в университетах. Благодаря лаконичности или даже минималистичности он быстро усваивается.

   Кроме того, Python может применяться практически в любой области разработки ПО (приложения, клиент-серверы, веб-приложения) и в любой предметной области. Он легко интегрируется с другими компонентами, что позволяет внедрять Python в уже написанные приложения. Проекты, написанные на этом языке, обладают свойством кроссплатформенности, то есть при необходимости их можно быстро и безболезненно перенести с одной операционной системы на другую.

   Однако Python является более медленным по сравнению с Java и C#. Так как интерпретируется сам текст и затрачивается время на разбор написанных выражений и конструкций. В рассмотренных же ранее языках исполняются готовые инструкции, которые на этапе компиляции уже были обработаны. Немаловажным фактом является то, что код можно легко украсть и проанализировать при декомпиляции.

   Язык Python набрал большую популярность, его легко изучать, и он применяется в различных сферах. Крайне популярен в научных сообществах, особенно в сфере искусственного интеллекта, больших данных, машинного обучения. Помимо приложений, позволяет создавать и веб-приложения, такие как YouTube, Spotify, Instagram.

   Популярность языка влечет за собой высокую конкурентность среди соискателей при поиске работы. Также всегда требуются дополнительные знания, в частности html/js и просто прикладных сферах, например ИИ.

   В целом изучение языка Python может стать даже полезным хобби, и позволит узнать новые концепции программирования и как программировать в целом. Он позволяет реализовывать различные вычисления, создавать программы быстро.

6. Swift язык нативный для разработки приложений под iOS и macOS. Язык Swift по замыслу его создателей призван не только заменить Objective-C в качестве базового для разработки приложений для iOS, но и заменить C во всех остальных сферах, в том числе IoT. Однако

об этом еще рано говорить, так как язык был представлен лишь в 2014 году, и для разработки на Swift пока требуются знания и C также.

Apple активно развивает направление по интерактивному обучению языку Swift через приложение Swift Playgrounds, позволяющее обучаться в игровом формате. Вакансии чаще связаны с платформой iOS, разработкой и поддержкой приложений. Или всегда можно начать свой стартап и разработать прорывное приложение.

7. Ruby, язык который поклонники называют красивым и искусным. И в то же время они говорят, что он удобный и практичный. Ruby очень гибкий язык, так как он позволяет его пользователям свободно менять, удалять и переопределять его части по желанию. Ruby старается ни в чём не ограничивать пользователя.

   Ruby как язык имеет несколько разных имплементаций, так как он полностью открытый и его можно изменять. Это обеспечивает интеграцию с другими языками или окружениями. Например:

   - JRuby это Ruby реализованный на Java Virtual Machine.
   - MacRuby это Ruby, который тесно интегрирован с библиотеками Cocoa от Apple для Mac OS X. Позволяет вам проще писать приложения для Mac OS X.
   - IronRuby это имплементация “тесно интегрированная с .NET Framework”.
   - Популярным является Фреймворк Ruby on Rails написанный на языке Ruby, а в свою очередь на нем созданы такие веб-сервисы как Airbnb, Github, Hulu. Так чаще всего требуются специалисты знающие именно Фреймворк Ruby on Rails, так как на нем разрабатываются многие современные веб-приложений.

8. PHP - скриптовый язык общего назначения, интенсивно применяемый для разработки веб-приложений и специально для нее сконструированный, и его код может внедряться непосредственно в HTML. Также входит в стек классических технологий веб-разработки. Однако с течением времени все чаще применяют другие технологии.

   На PHP написано крупные фреймворки, такие как Zend, Codeigniter, системы управления контентом и интернет-магазинами, что упрощает разработку таких решений.

9. JavaScript – поддерживает различные стили программирования. В основном применяется для создания интерактивных интерфейсов, как

веб-приложений, так и прикладных программ. Развитие фреймворков также позволяет использовать в качестве серверного языка. Так что JS везде :)

Это влечет за собой то что язык популярен в различных сферах, в связках с другими языками программирования и упоминается почти во всех вакансиях. Также язык рекомендуется многими к изучению в качестве первого, по нему достаточно литературы, различных онлайн ресурсов. А даже изучив основы, можно создать что то наглядное и интерактивно.

10. Под конец отметим, что языков программирования свыше двух тысяч, и постоянно создаются и развиваются новые. Одним из примеров является язык программирования Kotlin, разработанный петербургской компанией JetBrains, а командой Kotlin руководит выпускник Университета ИТМО. Как вы уже могли понять, языки программирования обладают своими плюсами и минусами, а задачу сделать универсальный язык уже никто перед собой не ставит. Как правило новые языки программирования разрабатываются программистами или компаниями с огромным опытом и обладают совместимостью с другими языками, но при этом упрощают и делают работу с ними проще. Так язык Kotlin полностью совместим с Java, тем самым стал официальным языком разработок для Android.

Мы рассмотрели исключительно популярные языки программирования, изучив основы любого из них вы будете лучше разбираться в сфере ИТ, а в дальнейшем углубляться в изучение технологий в рамках конкретного направления интересно вам.

## 4 Инструменты разработчика

В этой части лекции, мы рассмотрим основные программные средства, необходимые для написания своей первой программы и повышения эффективности при работе с кодом.

Для выполнения любого программного кода, необходимо либо наличие установленных пакетов языка программирования, либо наличие интегрированной среды разработки, либо программное обеспечение для его исполнения, а также любой текстовый редактор для написания кода.

Изучение программирования может быть сложным, но подходящие инструменты немного облегчают процесс. Начав писать прекрасный код, вы можете так увлечься, что забудете обо всем на свете. Но этот опыт может быть испорчен неправильным инструментарием.

Рассмотрим в первую очередь редакторы для написания кода, которые повысят вашу продуктивность. Конечно, писать код можно и в блокноте, или другом его аналоге, но что же отличает обычный текстовый редактор, от редактора кода? Отметим основные возможности:

- автодополнение кода – позволит не ошибаться в написании команд, предложит основные свойства объектов, а главное позволит писать код быстрее;
- подсветка синтаксиса и форматирование – облегчает чтение кода за счет использования различных цветов, шрифтов, начертаний и отступов;
- настройка интерфейса – написание кода процесс не быстрый, так что интерфейс реактора не должен отвлекать, а цветовая схема должна быть комфортна для работы, обычно используют темный фон рабочей области.

В первую очередь отметим кроссплатформенные и бесплатные редакторы, которые удовлетворяют ранее озвученным критериям, а использование каждого из них либо дело вкуса, либо тонких особенностей редакторов, связанных с продвинутыми инструментами разработки и интеграции.

Следующий инструмент в руках программиста это среда разработки. IDE отличается от редактора кода 4 вещами:

- поддержка большого числа языков программирования. Как правило, всех в рамках веба, мобильной или десктоп разработки;
- наличие компилятора, интерпретатора для преобразования кода высокого уровня в машинный код и его исполнения;
- встроенные утилиты для автоматизации процесса подключения библиотек, шаблонов и других пакетов;
- наличие дебаггера для обнаружения ошибок, опечаток и прочих рекомендаций по коду.

В целом для новичка в программировании может хватить и редактора кода, подходящего под конкретный язык, а ошибки искать лучше в ручном режиме, чтобы видеть их последствия и учиться на них. Однако в дальнейшем без хорошей IDE не обойтись. Хорошая – значит поддерживает различные языки, кроссплатформенна и бесплатна.

Перечисленные инструменты хороши для первых шагов в разработке, так как не придется тратить много времени на их изучение. При этом они перекрывают возможности большинства платных конкурентов, поэтому могут

использоваться и в более серьезных разработках. Отметим что существует большое число и других IDE, как платных, так и нацеленных на конкретные языки разработки и сферы применения. При этом многие предлагают бесплатные лицензии для студентов, чем вы можете пользоваться.

Отдельно отметим различные онлайн инструменты, которые помогут решать задачи в браузере без установки ПО на компьютер. О них также полезно знать хотя бы для того, чтобы быстро показать результат или поделиться ссылкой на код, с возможностью сразу его исполнить и увидеть результат. Среди самых универсальных. Популярны также и узко ориентированные сервисы для языков JS, Python, PHP и других.

Далее рассмотрим хоть и не обязательный, но очень важный инструмент разработчика, в лице системы управления версиями Git. Git – это набор консольных утилит, которые отслеживают и фиксируют изменения в файлах, работают они полностью локально, сохраняя данные в папках на жестком диске, которые называются репозиторием. Тем не менее, вы можете хранить копию репозитория онлайн, это сильно облегчает работу над одним проектом для нескольких людей. Для этого используются сайты вроде GitHub и bitbucket. Новичков часто пугает большое количество команд Git. Но для начала все они и не нужны, а онлайн сервисы вроде GitHub имеют приложения, где их знание вообще не требуется. О GitHub можно говорить как о социальной сети для разработчиков, через него можно найти работу. Ваш профиль может заинтересовать работодателя или конкретный проект. Поэтому даже только начиная изучать языки стоит сохранять свои проекты и свою активность, как разработчика.

И напоследок отметим средства для проектирования или точней инструменты и подходы, которые можно применять, когда мы пытаемся разложить некую задачу на язык последовательности действий. Так можно выделить три основных подхода:

Блок схемы. Один из простых подходов, но при этом стандартизированный по ГОСТ 19.701-90 и изучаемый в рамках любого направления связанного с программированием. Таким образом язык блок-схем является понятным каждому. Но нельзя говорить о необходимости представлять все в виде блок-схем на самом низком уровне разработки, скорей разумно использовать их для представления информации о взаимосвязи между компонентами, функциями, либо общей логики работы алгоритма.

Унифицированные языки моделирования. Среди них можно выделить UML и IDEF0 – оба языка моделирования являются графическими и также стандартизированными. Однако в первую очередь их используют для описания бизнес-процессов, проектирования и отображения организационных структур. При этом они предусматривают диаграммы для описания как структуры программы, так и к примеру самих классов, в случае объектно

ориентированной разработки. Применяются они уже в случае крупных проектов для документирования, или в качестве элементов технического задания.

И в третьих можно отметить более гибкий инструмент, необходимый скорей для записи заметок, идей и результатов мозгового штурма. Это интеллект-карта, диаграмма связей, карта мыслей или майндмэп. Все эти термины обозначают способ фиксации процесса мышления, наиболее похожий на то, как рождаются и развиваются мысли и идеи в нашем мозгу. Создания майнд карт процесс простой, так что программное обеспечение отличается лишь конечным результатом стиля.

Все рассмотренные технологии и программное обеспечение лишь краткий обзор, и зачастую выбор зависит от личных предпочтений, либо от имеющегося опыта. Можем лишь дать совет, пока вы учитесь, экспериментируйте. Каждый в конечном счете сформирует свои личные предпочтения.

# 5 Мобильная разработка

## 5.1 Зачем нужны мобильные приложения

Для чего нам нужны мобильные приложения? Ответ на этот вопрос достаточно многогранен. Во-первых, мобильные приложения создают удобства пользователям. Для быстрого доступа к информации, без сомнения, можно использовать браузер, однако даже мобильные версии сайтов не всегда работают быстро, особенно при медленном Интернет-соединении и зачастую функциональность таких страниц резко отличается от полных версий.

Рост среднестатистической диагонали современного смартфона, конечно налицо, но кто из нас не был в ситуации, когда палец упорно не хочет попадать на нужную ссылку и нажимает соседнюю.

Во-вторых, различные компании всячески стараются продвигать свои мобильные приложения из соображений маркетинга. Сегодня недостаточно создать свой сайт, в надежде, что отбоя от клиентов не будет. Свой сервис нужно продвигать, причем достаточно активно. Большая часть мобильной аудитории использует маркет плейсы для загрузки и покупки приложений (app-store, playmarket). Поэтому имея свое приложение, можно претендовать на новых клиентов. При этом качество приложения – очень важная характеристика, т.к. чем выше у него количество скачиваний, чем больше хороших отзывов, тем более высокие шансы попасть в различные списки «хитов» или «лучших приложений», которые формируются магазином приложений, а значит выше шансы заполучить новую аудиторию.

Использование мобильных приложений позволяет упростить бизнес процессы (например, при заказе еды или покупках в интернет магазинах) нет

нужды звонить оператору, диктовать данные и т.п. Продавцу же при этом нет смысла иметь сотрудников колл-центра, принимающих заказы. Такая оптимизация увеличивает скорость и удобство, а также уменьшает издержки.

Кроме того, существует огромное количество приложений, предлагающих различные сервисы от электронного списка для похода в магазин, до программ по 3д моделированию.

В качестве существенного плюса можно отметить, что разрабатывать приложение (не всегда, но часто), можно не обладая широкими знаниями в области программирования. Существуют примеры, когда простое по своей реализации, но воплощающее оригинальную идею приложение становилось крайне популярным. Немалая доля современных стартапов ориентирована на создание мобильных приложений. Если задуматься, то это прекрасная возможность создать нечто по-настоящему важное и полезное, что-то, чем могут пользоваться миллионы людей по всему миру.

## 5.2 Как разрабатывать

Технологий разработки мобильных приложений существует достаточно большое количество. Мы постараемся очертить некоторую классификацию, хотя и она на претендует на догматичность, т.к. могут встречаться различные «гибридные» подходы.

Начнем с самого простого, а именно с конструктора мобильных приложений. На сегодняшний день на рынке присутствуют десятки различных вариантов таких конструкторов. К очевидным преимуществам можно отнести простоту и скорость создания приложения. Т.е. если стоит задача создать приложение, обладающее каким-то стандартным функционалом и подходящее под определенный шаблон (Интернет магазин, ресторан, доставка, служба такси, дизайнерские студии и т.п.). Разработка такого приложения может занять буквально несколько минут. Потом останется только его добавить в магазин приложений для проверки, после которой его уже может установить любой желающий. К недостаткам такого подхода можно отнести ограниченность возможностей. Из кубиков детского конструктора можно создавать определенные объекты и совсем невозможно получить правдоподобное, например, обручальное кольцо. Так и здесь: для типовых задач метод хорош, но для уникальных и сложных проектов явно не годится.

## 5.3 Кроссплатформенность

Более сложным подходом является полноценная кроссплатформенная разработка. Различные мобильные ОС хоть и обладают похожим функционалом, но все же достаточно сильно различаются как внутренней архитектурой, так и интерфейсами взаимодействия с внешними программами. То, как именно мобильное приложение будет использовать процессор и память

устройства существенно влияет на скорость работы и производительность. Функциональные возможности операционной системы предоставляются разработчикам посредством API. Но дело в том, что у различных операционных систем различные интерфейсы, поэтому среды разработки (IDE), в которых работают программисты, содержат библиотеки, обеспечивающие доступ к функциональным возможностям различных операционных систем. Такие библиотеки разработчика выполняют роль переводчика между кодом, который написал программист и функциями конкретной операционной системы.

Они обычно скомпилированы под разные архитектуры процессоров и операционные системы. API функции также могут отличаться. При этом сама библиотека разработки будет выглядеть одинаково для программиста, т.е. иметь одинаковые названия функций, свойства и методы. Однако на уровне машинного представления это будет код с большими отличиями. Такие отличия накладываются не только со стороны архитектуры под которую компилируется код, но и со стороны инструментальных средств разработки.

Если используются инструменты разработки, которые код на языке программирования компилируют именно в машинный код, тогда программа будет работать только под одной архитектурой процессора и запустить такую программу под другой архитектурой будет невозможно. Поэтому часто разработку ведут с использованием виртуальных машин исполнения, принципы которых присущи java и C#. Тогда код выполняет система исполнения, а ее уже выполняет ОС и соответственно процессор. При таком подходе с разработчика снимается задача, чтобы версия программы было скомпилирована под разные архитектуры. Однако это реализуемо только в том случае, если в такую программу не добавляются какие-либо библиотеки (сторонние или собственные), которые реализуются нативно для процессора или ОС.

В таком случае мы получаем гибридную программу. Особенностью таких программ будет то, что весь код, написанный с помощью интерпретируемого языка будет переносим, но медлителен, а код созданный с использованием нативного подхода и состоящий из машинных инструкций, хоть и не будет переносим на другую архитектуру, зато обеспечит очень быструю работу в местах своего вызова.

К преимуществам такого подхода можно отнести то, что один и тот же код, написанный разработчиком можно использоваться для создания приложений для различных ОС и различных архитектур процессоров. Существенным недостатком является то, что универсальность неизбежно связана с компромиссами. Из-за использования «посредников» в виде библиотек разработки страдают объем и быстродействие конечных приложений.

## 5.4 Разработка под ОС

При разработке под определенную ОС программист намеренно отказывается от многих инструментов разработки или использует непосредственно те, которые ориентированы на определенную ОС. В таком случае программы получаются меньшего размера, причем как следствие, значительно быстрее. При еще более дифференцированном подходе используется ориентация под определенную архитектуру процессора, потому что в таком случае можно достичь максимальной стабильности и производительности при меньших объемах.

К существенным недостаткам можно отнести время, затрачиваемое на разработку, т.к. под каждую ОС создается свое приложение практически «с нуля». При этом для эффективного использования функций ОС необходима хорошая квалификация непосредственно разработчика.

## 5.5 Перспективы

Сложно предполагать, в каком направлении будет развиваться разработка мобильных решений. На сегодняшний день все указывает на то, что их количество будет расти. Некоторые эксперты предрекают слияние мобильных и настольных ОС. В таком видении есть рациональное зерно. Начиная с 80-ых годов хабом для остальных цифровых устройств являлся персональный компьютер. Но с ростом как числа мобильных устройств, так и функциональности мобильных решений происходит перенос функции устройства, объединяющего остальные, на смартфон.

Уже сейчас подавляющее число гаджетов и бытовой техники, начиная от фитнесс-браслетов, умных весов, пылесосов и заканчивая элементами умного дома, управляются при помощи смартфона. Такой подход порождает запрос на создание крупных интегрированных приложений для координации работы большого количества устройств.

# 6 Методологии

Здравствуйте, в этом видео мы поговорим о методологиях разработки программного обеспечения. Тема достаточно актуальная, допускающая различные трактовки и дискуссии. Что с одной стороны хорошо, ибо как сказал Сократ: «в споре рождается истина», с другой стороны когда нет определенности новичку становится сложнее ориентироваться. Мы рассмотрим некоторые основные подходы и их особенности, не претендуя на строгую академичность и полноту. Предложенную классификацию на самом деле можно изменять, дополнять, расширять.

Что же такое методология разработки ПО. Строго говоря это описание

процессов и взаимодействий, необходимых для создания программного продукта. Т.е. что, кому, как и в какой последовательности нужно делать, чтобы создать программный продукт.

Самой старой, но проверенной моделью разработки программного обеспечения является **каскадная**. Иногда ее еще называют «водопад». Идея крайне проста и понятна. Для начала определяются требования к разрабатываемому продукту, потом наступает стадия проектирования. Т.е. более детальное описание того, каким функционалом будет обладать продукт, из каких компонентов состоит и как эти компоненты будут взаимодействовать между собой.

Далее следуют стадии дизайна, кодирования, тестирования и поддержки. Существенными преимуществами такого подхода является то, что до начала разработки можно более-менее точно определить сроки выполнения того или иного этапа, а также финальный бюджет.

Из минусов можно отметить, что каскадная модель эффективна лишь в том случае, когда есть четкое представление того, что должно получиться на выходе, т.к. внесение правок в изначальное ТЗ на финальных стадиях влечет серьезные временные и финансовые затраты. Существуют и сложности со стороны исполнителя. Проектированием и, например, тестированием занимаются обычно разные люди, поэтому если в работе находится несколько проектов возможны простои как работников (продукт завис на предыдущей стадии), так и продуктов (в работе находится другой продукт).

**V образная модель** тоже использует step by step подход, однако обладает некоторым особенностями, а именно крайне серьезно отношение к тестированию. В классической каскадной модели тестирование выполняется после завершения процесса кодирования. В V образной модели тестирование – это часть процесса кодирования. Т.е. модули и компоненты проверяются еще на этапе разработки. Такой подход оправдан, если программное обеспечение должно работать бесперебойно. Например сервисы госструктур, военных ведомств, медицинский учреждений и т.п.

**Инкрементная модель** предполагает абсолютно противоположный подход по сравнению с каскадной. Продукт проходит те же стадии, однако они более короткие, но повторяются несколько раз. Такая модель предполагает получение рабочей версии продукта с той или иной функциональностью уже после первой итерации. К преимуществам можно отнести то, что в достаточно короткие сроки продукт, хоть и с ограниченным функционалом можно выпускать. Внесение изменений на этапе разработки не оказывает серьезный негативных последствий. Заказчик может остановить разработку в любой момент и использовать актуальную рабочую версию. К недостаткам можно отнести то, что в первых версиях продукта функционал приносится в жертву скорости разработки, а также неизвестна финансовая стоимость конечного

продукта.

**RAD** это один из типов инкрементной модели. Основная особенность заключается в том, что отдельные компоненты проекта разрабатываются параллельно несколькими командами разработчиков. Полученные модули затем интегрируются в один прототип и представляются заказчику для уточнения требований или вывода продукта на рынок. Такой подход является достаточно затратным, но быстрым. RAD-модель обычно предлагается для заказов от крупных компаний, давно работающих в той или иной сфере, у которых есть потребность в скорейшей реализации проекта.

Методологии типа **Agile** это общее название нескольких подтипов (kanban, XP, scrum и т.д.) которые в основе своей философии подразумевают гибкость в разработке. Такой подход наиболее эффективен в условиях, когда требования к продукту достаточно часто меняются. Обычно это происходит не из-за того, что заказчик переменчивая или ветреная натура, а в силу объективных причин, например, динамичностью рынка. Такой подход также актуален для больших проектов, которые необходимо оперативно подстраивать под вновь появляющиеся требования.

Для обеспечения непрерывного взаимодействия как с заказчиком, так и между всеми членами команды используются регулярные встречи (обычно несколько раз в месяц), которые называются sprint. Исполнители также используют ежедневные короткие встречи, которые называются Scrum, где обсуждается, что было сделано с прошлого scruma, что планируется сделать в рамках текущего и проблемы, с которыми столкнулись.

**Итерационная модель** похожа на инкрементную в том, что разработка ведется поэтапно и на каждой итерации предполагается наличие работающей версии продукта. Однако существует концептуальное различие в подходе к непосредственной реализации. В инкрементной модели разработка ведется по частям, и продукт представляет собой набор модулей, добавляя которые можно увеличивать функциональность. В итерационной модели продукт даже на первой стадии разрабатывается весь продукт, со всеми отдельными модулями, но с ограниченной функциональностью. На следующих итерациях этот «скелет» обрастает «мясом функциональности». Очевидно, что на каждой следующей итерации требования уточняются и учитываются при разработке.

**Спиральная модель** – это по сути инкрементная, в которой акцент смещен не на добавление новых функциональных возможностей, хотя их конечно тоже, а на анализ рисков при их добавлении. Такой подход используется при разработке систем, во главу угла которых ставится стабильность и отказоустойчивость. Например, банковские системы перед внедрением новых возможностей должные протестировать последствия и проанализировать риски крайне ответственно. Иначе ввод новых функциональных возможно-

стей может привести к краху системы, что недопустимо.

Подводя итог, методологии разработки - это лишь общие концепции, которые достаточно редко встречаются в чистом виде. В большинстве своем на деле используются некоторые комбинации или гибридные версии таких моделей. При этом подходы могут часто видоизменяться в зависимости от требований к проекту, финансовых условий и сроков исполнения, от команды разработчиков, руководителя проекта и заказчика.

# 7 Будущее разработки

Напоследок, рассмотрим вопрос связанный с будущем профессии программиста. Что ждет профессию разработчика с развитием современных технологий, таких как искусственный интеллект, нейросети, блокчейн, BigData, квантовые вычисления, адаптивная и виртуальная реальность.

Масштабы глобальности применения технологий с каждым днем нарастают. Поэтому выбирая ту или иную сферу деятельности, можно посоветовать начинающим разработчикам ориентироваться как на изучение фундаментальных языков программирования, так и на охват различных технологий и средств в рамках языка. Для этого ставьте перед собой задачи, которые можно решить с помощью разработки программы или приложения и, к примеру, с интеграцией простейшей нейронной сети.

Можно говорить с уверенностью, что программист будущего, это человек, способный не только использовать свои навыки программирования, но и всегда открыт к изучению новых языков программирования, новых принципов и подходов, готов экспериментировать. Однако важен и баланс, можно потратить месяцы на поиск и чтение информации о подходах и техниках, но часто больше знаний можно получить при решении конкретной задачи. И даже если выбор технологий вас разочарует, полученные знания облегчат изучения новых.

Развитие технологий в целом, влечет за собой как сокращение вакантных рабочих мест в одной сфере, так и появление новых направлений, связанных с новыми особенностями разработки.

Например, система Pix2Code может писать интерфейсы приложений под Android и iOS, а также веб-сайты с помощью нейронной сети, которая генерирует код на основе макетов интерфейса. Система DeepCoder, разработанная Microsoft умеет сама писать код. Для этого искусственный интеллект комбинирует строчки кода, взятые из различных источников.

Такие системы, конечно же не оставят программистов без работы. Благодаря им, разработчики смогут посвящать время более творческой и сложной работе, а простые задачи решать в автоматическом режиме.

Таким образом еще раз стоит отметить необходимость изучения технологий, вместе с изучением языков программирования. Все чаще необходимы программисты:

- способные обучить нейронную сеть – то есть те, кто не столько программирует нейронную сеть или другие сложные системы машинного обучения с нуля, сколько умеет их обучать. Такому специалисту главное не уметь писать весь код самому, а главное умение понимать чужой код, модифицировать его под необходимые задачи и главное – правильно обучить сеть, для последующего использования;

- знатоки Open source проектов. Те, кто хорошо разбирается в открытом коде, знает о существовании различных проектов и алгоритмов, которые можно адаптировать под другие задачи или интегрировать готовые решения в виде компонента целостной системы. И снова важным является умение читать и разбираться в коде, нежели изобретать велосипед с нуля;

- специалисты по различным API интерфейсам, так как все больше крупных сервисов предоставляют API. Таким образом программист должен знать, что происходит в мире, какие новые интерфейсы и где открываются, как их можно применить и как с ними работать. Системы авторизации через социальные сети, интеграция сервисов карт, данные из википедии – одни из простейших примеров таких интеграций.

Широкое развитие open source проектов, интеграции различных сервисов, сделала популярными среди разработчиков сервисы создания контейнеров вроде Docker и Packer и др. Такие сервисы позволяют разработчикам оперативно создавать и делиться контейнерами, включая в них необходимые службы, компоненты, библиотеки, системные инструменты. Вы можете отделить ваше приложение от инфраструктуры и обращаться с инфраструктурой как управляемым приложением. Все это экономит время, например упрощает развертку сервиса на множестве серверов. С появлением и развитием контейнеров, развиваются и появляются новые языки, например язык Go, у которого возможностей работы с ними куда больше, чем у тех же Java и C++. К примеру, контейнеры и язык Go, позволяют изящно реализовать параллельные вычисления.

Мы рассмотрели лишь общие сведения крайне обширной сферы программирования и разработки. В последующих разделах вы узнаете больше про современные технологии, такие как искусственный интеллект, нейросети, блокчейн, квантовые вычисления и другие.

И в последнюю очередь, развеем еще один миф, многим кажется, что разработка программного обеспечения похожа на строительство космического

корабля. Где все очень точно спроектировано, продумано, сконструировано, протестировано на все возможные проблемы и запущено в космос. Однако, на самом деле разработка сложных программ больше напоминает починку несущегося болида формулы 1, где единственный материал, который использовали для его сборки это синяя изолента.

И не стоит думать, что в один прекрасный день вы будете целиком и полностью понимать как работает нечто большое, над чем трудится не один разработчик. Могут возникать моменты, когда вы понятия не имеете что и зачем вы делаете, и это не редкость, но главное выполнять свою работу качественно.

# 8 Начало программирования в Python

## 8.1 Основные идеи

В этом фрагменте мы рассмотрим основные идеи создания простейших компьютерных программ и проверим их на практике. В качестве первого языка для изучения очень часто рекомендуют использовать Python, воспользуемся этим советом.

Говоря об основных идеях создания программы, мы имели в виду то, что написание кода является в некотором смысле диалогом между программистом и компьютером. В этом смысле последний похож на собаку, обученную вполне ограниченному набору трюков. При этом, если собака не понимает, что от нее хотят, то она и не сможет ничего продемонстрировать. При исполнении программного кода, компьютер выполняет ровно то, что ему было велено делать, ни больше, ни меньше. Если он столкнулся с тем, что программа написана некорректно, выдается сообщение об ошибке, часто с указанием конкретного места, где она произошла. Даже самые опытные программисты могут в процессе создания кода совершать ошибки, в основном по невнимательности, в этом нет ничего страшного, просто чем больше опыта, тем реже такое случается. Давайте рассмотрим базовые возможности языка Python.

## 8.2 Константы

Не вдаваясь в подробности, константы можно разделить на числовые и текстовые. Для вывода на экран будем использовать команду `print()`, где в скобках указывается то, что мы хотим вывести на экран, например:

```
print(567) #567
print(10.8) #10.8
print('Hello World!') #Hello World!
```

Отметим, что текстовые данные заключаются в одинарные или двойные кавычки. А разделителем между дробной и целой частью числа является точка. Все что написано после символах # в переделах одной строки является комментарием и при выполнении кода пропускается.

## 8.3 Переменные

Переменные используются для хранения данных внутри программы. Для визуализации можно представить переменную как контейнер, в который можно что-то положить. В нашем случае это будет число или текст. При этом все контейнеры должны быть подписаны, т.е. иметь свое уникальное имя. Например, поместим в переменную `a` значение `5`, а в переменную `b` значение `8`. Далее выведем на экран переменную `result`, в которой будет вычисляться сумма `a + b`:

```
a = 5
b = 8
result = a + b
print(result)
```

В качестве замечаний можно отметить:

- в данном случае необязательно пользоваться 3-ей переменной, так как сумму a + b можно посчитать прямо в функции `print()`, т.е. `print(a + b)`;
- использование пробелов является «правилом хорошего тона», но записи типа `a=5` и `result=a+b` будут выполнены машиной точно также как и с пробелами.

Важно понимать, как пользоваться переменными. Например, если в переменной уже находится какая-то информация и туда поместить новую, то старая информация просто перезапишется.

```
a = 5
b = 8
print(a + b) #13
a = 'hello'
print(a) #hello
print(a + b) #ERROR!
```

Отдельно стоит упомянуть, что переменные лучше именовать таким образом, чтобы по названию было понятно, для чего эта переменная используется и какая информация там хранится. Например: speed, force, normal-acceleration, boss-month-salary и т.п. При этом пробелы в названиях переменных использовать нельзя, а переменные boss, Boss и BOSS будут разными. Рассмотрим еще такой пример, иллюстрирующий работу с переменными:

```
a = 3
a = a ** 2 + (a - 1) * 5
print(a)
```

Заметим, что во второй строчке сначала вычисляется то, что стоит справа от знака равно (операция присвоить), при этом вместо `a` подставляется значение `3`, а лишь затем вычисленное значение помещается в переменную `a`. При этом операция `**` – это возведение в степень, а приоритеты выполнения операций такие же, как в обычной арифметике.

## 8.4 Условия

Возможность ветвления – один из существенных мотивов использования программирования как такового. Связка ЕСЛИ – ТО (IF – THEN) является основополагающей в автоматизации различных процессов. Рассмотрим, как работает оператор `if`.

```
a = 10
if a < 12:
  print('smaller than 12')
print('Done')
```

После слова `if` следует логическое выражение, которое может быть истиной или ложью. Если это выражение верно (истина), то выполняется тот код, который следует после двоеточия и имеет отступ (пробелы относительно самого слова `if`). Если же логическое выражение ложно, то код, входящий в `if`, игнорируется, а последующие команды исполняются.

В качестве логических выражений чаще всего используются операции сравнения:

| Python | Meaning |
|---|---|
| < | Less than |
| <= | Less than or Equal To |
| == | Equal to |
| >= | Greater than or Equal to |
| > | Greater than |
| != | Not Equal to |

Рис. 1: Операции сравнения

При этом важно не путать операцию присваивания `=` и операцию сравнения `==`.

Рассмотрим пример программы, которая находит корни квадратного уравнения. Пусть $a, b, c$ – коэффициенты в квадратном уравнении $ax^2 + bx + c = 0$:

```
a = 1
b = -2
c = 2
if a != 0:
  disc = b ** 2 - 4 * a * c
  if disc < 0:
    print('No real roots')
  if disc == 0:
    x = -b / (2 * a)
    print('Double root x = ', x)
  if disc > 0:
    print ('Two different roots:')
    x_1 = (-b + disc ** 0.5) / (2 * a)
    x_2 = (-b - disc ** 0.5) / (2 * a)
    print('x_1 = ', x_1)
    print('x_2 = ', x_2)
else:
  print('It is not a Quadratic Equation')
print('We hope you've enjoyed')
```

Обращаем ваше внимание на то, что все проверки дискриминанта находятся внутри главного условия `if a != 0:`. При этом в условии еще присутствует ветка `else:`, которая будет выполнена, если a все-таки равно нулю. Попробуйте поменять коэффициенты и посмотрите, что будет получаться.

## 8.5 Циклы

Одно из основных преимуществ компьютера заключается в том, что ему можно поручить выполнение монотонной рутинной работы, требующей большого количества повторений. Поэтому использование циклов приходится как нельзя кстати. Мы рассмотрим циклы `for` и `while`. Начнем с `for`. В таком цикле указывается переменная и множество значений, которые она способна принимать. Способы задания множества значений могут быть весьма различными. Например:

```
for i in 4, 20, -3, 'word':
  print(i)
```

В данном случае `i` – переменная, а числа `4, 20, -3` и слово `word` – значения, которые должна принять переменная. Дословно можно перевести так: «Для каждого `i` из множества ...»

Задание множества значений перечислением далеко не всегда удобно, поэтому для числовых значений удобно использовать функцию `range()`. Например:

```
for num in range(0, 14, 3):
  print(num)
print('cycle is over')
```

В данном случае переменная `num` будет принимать значения от `0` включительно до `14` не включительно с шагом в `3` значения, т.е. `0, 3, 6, 9` и `12`.

Обращаем внимание, что тело цикла, т.е. то, что выполняется внутри этого цикла также, как и в случае с условным оператором, выделяется отступами.

Цикл `while` реализуется несколько другим образом. Сначала указывается некоторое условие (логическое выражение) и пока это условие истинно, будет выполняться то, что написано в теле цикла. Если оно ложно – тело цикла пропускается и выполняется переход к следующей команде.

```
i = 0
while i < 5:
  print(i)
  i += 1
print('cycle is over')
```

При использовании цикла `while` нужно быть крайне внимательным, чтобы не получить бесконечный цикл, например такого вида:

```
i = 0
while True:
  print(i)
  i += 1
print('cycle is over')
```

Для принудительного перехода к следующей итерации или для выхода из цикла используются специальные команды `continue` и `break`, про которые вы можете почитать самостоятельно.

## 8.6 Массивы

Предположим, что в группе 30 человек, и для каждого нам нужно хранить в программе его возраст. Можно создать 30 уникальных переменных, но это крайне неудобно. Для решения подобной задачи удобно использовать массив. Массив – это множество переменных, рассматриваемое как единое целое. Возвращаясь к аналогии с контейнерами, массив – это большой контейнер, имеющий название и содержащий другие контейнеры, названиями

которых являются числа. Отсчет начинается с нуля. Массив задается именем и перечислением своих элементов через запятую, заключенных в квадратные скобки. Обращение к конкретному элементу массива происходит по его индексу.

```
array = [13, -2, 6, 'car', 3]
print(array[0])
print(array[3])
print(array[-1])
```

Весьма органичным является совместное использование циклов и массивов. При этом, если длина массива, т.е. количество элементов заранее не известно, удобно использовать функцию `len()` для того, чтобы знать, когда заканчивать цикл. При этом с элементами массива можно работать, как и с обычными переменными. Допустим на каждом шаге выводить на экран квадрат элемента массива.

```
array = [4, -2, 6, 7, 3]
for i in range(0,len(array)):
  print(array[i] ** 2)
print('cycle is over')
```

**Онлайн компиляторы**:
`https://www.onlinegdb.com/online_python_compiler`
`https://repl.it/languages/python3`

**Интерактивный учебник**:
`https://pythontutor.ru/lessons/ifelse/`