My$_S tudy$

Выпуск 1.0

нояб. 06, 2020

Оглавление

1 Главная навигация по сайту **3**

1.1 Главная . . . 3

1.1.1 Что это? . . . 3

1.1.2 Содержание . . . 3

1.1.3 Какой во всём этом смысл? . . . 3

1.1.4 Как этим пользоваться? . . . 4

1.1.5 Как со мной связаться? . . . 4

1.2 Разделы . . . 4

1.2.1 Операционные системы . . . 4

1.2.1.1 Конспекты . . . 5

1.2.2 Математика . . . 28

1.2.2.1 Курсы . . . 28

1.2.2.2 Конспекты . . . 33

1.2.3 Web-технологии . . . 33

1.2.3.1 Конспекты . . . 33

1.2.3.2 Курсы . . . 33

1.2.4 Python . . . 42

1.2.4.1 Курсы . . . 42

1.2.5 Scratch . . . 52

1.2.5.1 Курсы . . . 52

Здесь я храню и разрабатываю конспекты и статьи по различным темам - сам пишу учебный материал как лично для себя, так и для других, собираю полезные и интересные материалы по той или иной теме.

Изначально, весь материал разрабатывался в формате markdown-файлов, которые может отображать GitHub. Но сейчас, репозиторий разрабатывается как сайт, сгенерированный Sphinx - Python Documentation Generator и размещённый с помощью Read The Docs.

Глава 1

Главная навигация по сайту

1.1 Главная

1.1.1 Что это?

Здесь я храню и разрабатываю конспекты и статьи по различным темам - сам пишу учебный материал как лично для себя, так и для других, собираю полезные и интересные материалы по той или иной теме.

Изначально, весь материал разрабатывался в формате markdown-файлов, которые может отображать GitHub. Но сейчас, репозиторий разрабатывается как сайт, сгенерированный Sphinx - Python Documentation Generator и размещённый с помощью Read The Docs.

1.1.2 Содержание

- Операционные системы
- Математика
- Web-технологии

1.1.3 Какой во всём этом смысл?

Цель проекта — максимально последовательно и простым языком, насколько это возможно, объяснить сложные темы.

Таким образом я сам наиболее эффективно изучаю интересную мне тему, ведь нужно очень хорошо подумать: осмыслить как сами принципы, прийти к пониманию, так и разобраться в частностях и практике — это достаточно непростая задачка, работая над которой, ты сможешь действительно овладеть всем этим в полной мере.

Изучать так, будто ты будешь должен объяснить это кому-то ну совершенно не сведущему.

Ну и другим будет полезно, кто интересуется и тоже жаждет знаний.

1.1.4 Как этим пользоваться?

Всё просто! Каждый раздел — это отдельная тематика, в которой находятся соответствующие материалы.

Важно: Могу ли я в этом поучаствовать? Помочь?

Конечно! Вы можете внести вклад в развитие дела как просто поддержкой в виде **звёздочки** для этого репозитория, так и инициируя какие-либо дельные запросы/вопросы/правки/предложения через **Issue**. Непосредственно поучаствовать в разработке с помощью **Pull Requests**.

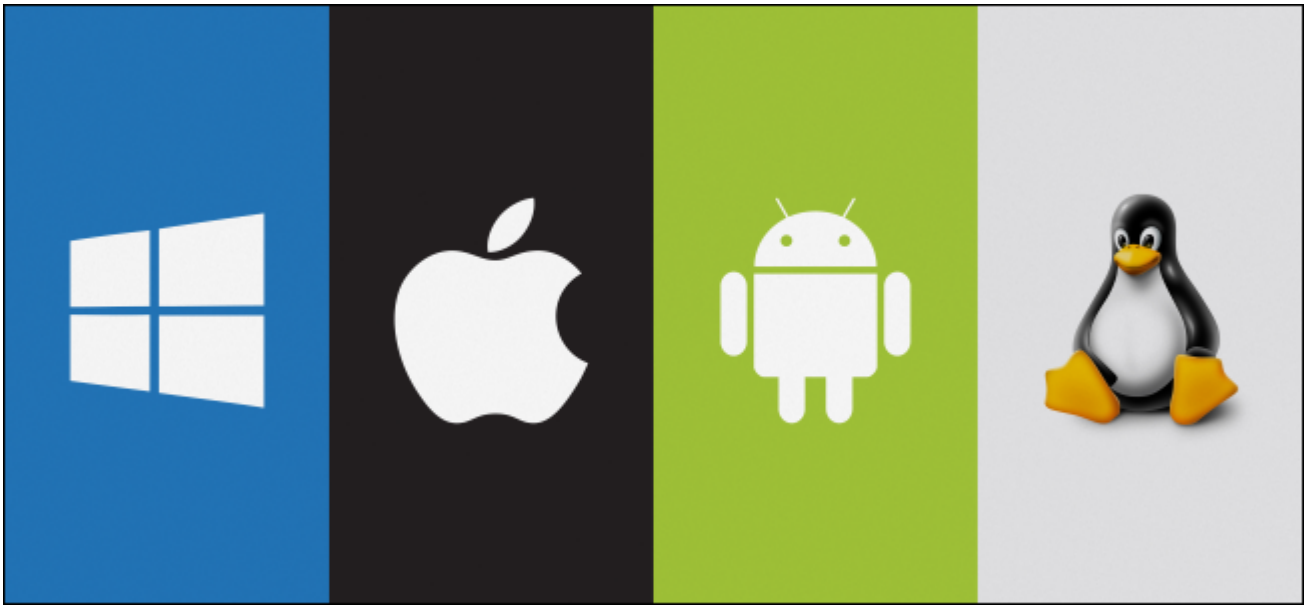
1.1.5 Как со мной связаться?

- email –> steindvart.st@gmail.com
- Telegram –> @Steindvart
- Discord –> Steindvart#1423

1.2 Разделы

В раждом разделе хранятся материалы по соответствующей теме

1.2.1 Операционные системы



1.2.1.1 Конспекты

Доработанные конспекты-статьи по лекциям. Можно использовать как совместно с промотром лекции, для закрепления, так и отдельно от лекции.

Примечание: В какой-то степени конспекты даже лучше: читать быстрее, более подробная и структурированная информация с ссылками на другие источники, обилие поясняющих схем.

Что такое операционная система и как она работает?

Примечание: Основа конспекта, лекция - Что такое операционная система и как она работает Автор конспекта/статьи немного переработал и дополнил некоторые моменты: структуру и повествование, постарался дать более глубокое и последовательное разъясление, добавил как скаченные, так и созданные им лично схемы.

Цель конспекта — последовательно рассмотреть и объяснить принципы устройства и функционирования операционной системы, её основных компонентов и абстракций.

Содержание

- *Что такое операционная система и как она работает?*
  - *Введение*
    - *Немного истории*
  - *Зачем нужна Операционная Система?*
    - *Фунции ОС*
    - *Основные абстракции ОС*
  - *Положение ОС в многоуровневой иерархии организации компьютера*
  - *Как операционная система загружается в компьютер?*
  - *Что делает ядро ОС?*
  - *Прерывания*
  - *Как приложения взаимодействуют с ОС?*
  - *Как оборудование взаимодействует с ОС?*
    - *Что это значит?*
  - *Сервисы ОС*
    - *Основные*
    - *Дополнительные*
  - *Основные абстракции*
    - *Процесс*
    - *Поток*

        * *Файл*
    - *Реализация многозадачности*
        * *Переключение контекста*
        * *Критические секции и блокировки*
        * *Deadlock - взаимная блокировка*
    - *Адреса и управление памятью*
    - *Итог*
        * *Основные механизмы (сервисы)*
        * *Абстракции*
        * *Заключение*

Введение

Операционные системы окружают нас повсюду – это основное программное обеспечение персональных компьютеров, серверов, мобильных устройств, сетевых устройств (роутеры, коммутаторы) и даже современных автомобилей (борт-компьютер), телевизоров и прочего. Перечислять можно очень долго, ведь они требуются практически в каждой компьютерной системе.

Любой компьютер представляет собой **связанную совокупность: процессора, памяти и устройств ввода-вывода**.

Сама по себе, аппаратура умеет делать только очень простые, базовые операции - по типу: *сложить два числа, перейти к адресу, записать по адресу и тд.*

Например, **процессор** умеет выполнять только четыре базовых типа инструкции:

- Чтение инструкций/данных из памяти (read)
- Выполнение интрукции (execute)
- Запись результата в память (write)
- Прерывание (interrupt)

Получается, что непосредственное **создание и управление** сложными процессами (приложениями) на аппаратуре становится крайне неэффективным и неудобным. То есть, например, создать и запустить на исполнение программу-браузер исключительно с помощью данных инструкций становится крайне сложной задачей. Особенно при условии, что помимо этого процесса (браузера) существуют и другие процессы, которые также пользуются ресурсами вычислительной машины.

Возникает вопрос — *Как заставить всё это слаженно и эффективно работать, сделав пользование компьютером удобным как для обычного человека, так и для прикладного программиста?*

Чтобы ответить на этот вопрос более последовательно, немного заглянем туда, откуда всё начиналось.

Немного истории

На заре компьютерной эпохи, первые компьютеры представляли собой огромные блоки (занимавшие большие комнаты), в которых размещались основные его компоненты: **процессор, память и устрой-**

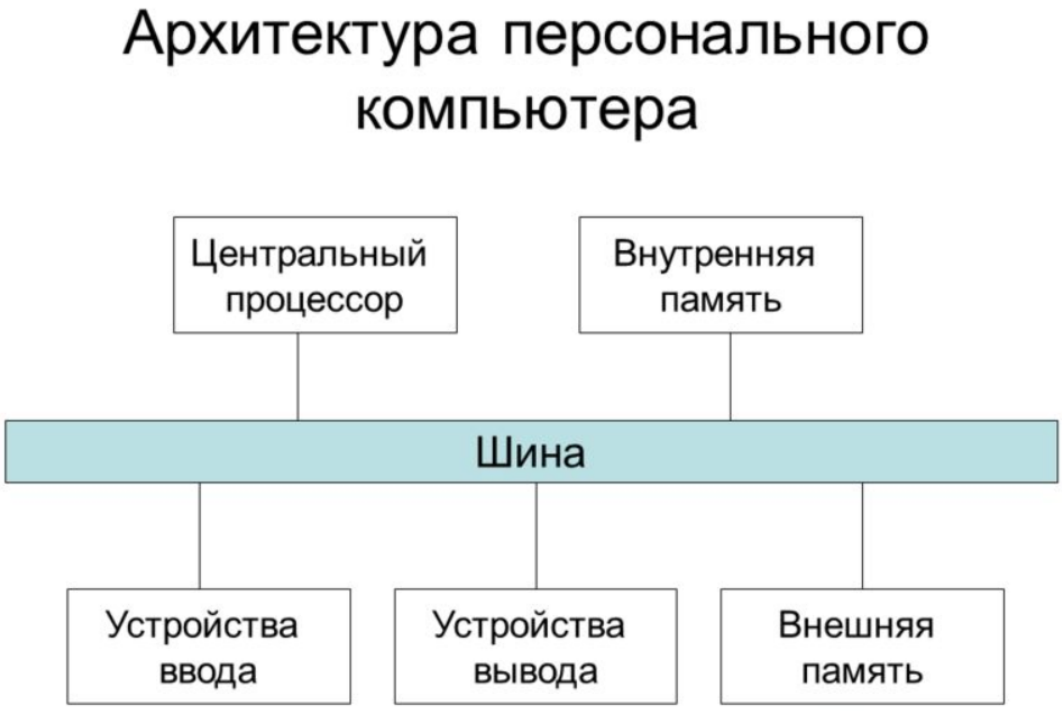


Рис. 1: Рис. 1. Общее представление архитектуры компьютера
Более подробная структура архитектуры ПК

ства ввода-вывода. И всего можно было выделить *два состояния*, в котором, в реальном времени находится компьютерная система:

- Ввод/Вывод
- Вычисление

Внимание: Важная идея! Так как вычисления производятся быстрее, чем непосредственный ввод-вывод данных, разработчикам пришла идея о том, что к ресурсам можно допускать не одного пользователя (процесс), а множество, предоставляя им способ независимо друг от друга загружать (ввод) и получать (вывод) данные через отдельные терминалы, чтобы более эффективно использовать ресурсы компьютера и вычислительные модули не простаивали в ожидании ввода/вывода.

Идея многопользовательского режима в использовании ресурсов компьютера нашла свою реализацию в понятии **процесс**. То есть, каждый процесс - **это пользователь ресурсов компьютера**.

Эта идея положила начало созданию такой системы, которую мы теперь называем операционной - **программной системы, которая управляет ресурсами компьютера, а следовательно осуществляет доступ к этим ресурсам и управляет процессами - пользователями этимх ресурсов.**

Примечание: Далее, термины: *процесс, приложение* идут как синонимы термину **пользователь ресурсов**.

Зачем нужна Операционная Система?

Операционная система - **программная система, которая управляет ресурсами компьютера, а следовательно осуществляет доступ к этим ресурсам, управляет процессами, вводит и реализует различные абстракции, выполняет другие системные функции.**

Существует три ключевых элемента операционной системы:

1. Абстракции (процессы, потоки, файлы, сокеты, адресное пространство).
2. Функции (создание, управление, открытие, запись, распределение).
3. **Конкретные реализации**
    - Архитектуры: монолитные, модульные, гибридные;
    - Алгоритмы: LRU, EDF;

Фунции ОС

- **Управление ресурсами и процессами, а также совместное использование вычислительных ресурсо**
    - Scheduler — планировщик. Механизм управляющий процессами и реализующий многозадачность.
    - Memory manager — менеджер памяти. Механизм выделяющий память и управляющий ею.
- **Абстракция оборудования для удобства и переносимости**
    - то есть реализация единого интерфейса для разного, но схожего по функциям оборудования.
- Изоляция ошибок приложений друг от друга (и от ядра ОС)
- **Переносимость данных между приложениями (процессами)**
    - Inter Process Communication (IPC) — Механизм межпроцессного взаимодействия
    - файлы и файловая система

Основные абстракции ОС

- Процессы и потоки - программы, что находятся в оперативной памяти и обрабатываются процессором. Другими словами - исполнение программы.
- Файлы и файловые системы - некоторая структура данных и связи между этими структурами, универсальный системный интерфейс.
- Адресное пространство и память - распределение и управление памятью.
- Сокеты, протоколы, устройства - интерфейсы взаимодействия.

Положение ОС в многоуровневой иерархии организации компьютера

Современный компьютер можно представить в виде иерархии уровней (от двух и более), где на каждом уровне выделяются свои абстракции и набор возможных функций.

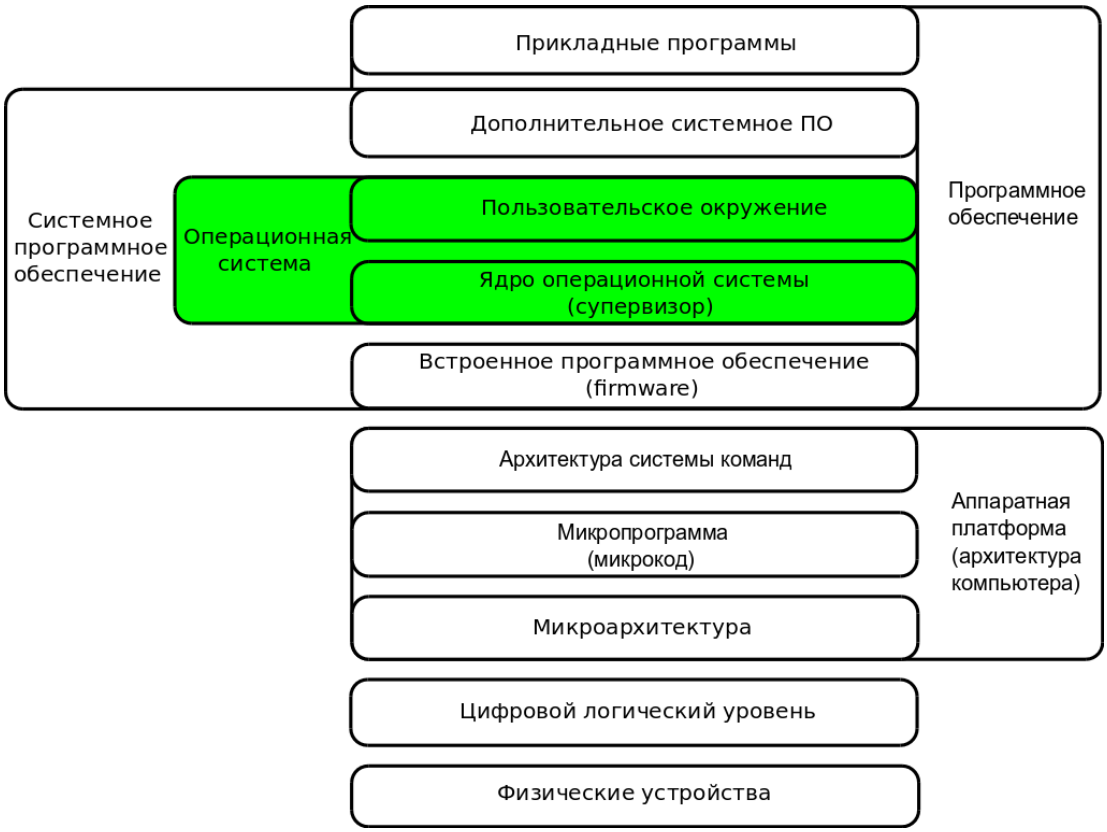


Рис. 2: Рис. 2. Основные уровни устройства ПК

Операционная система является одним из таких уровней и представляет собой **интерфейс** («прослойку») между пользователем ресурсов компьютера и самими ресурсами, управляющий взаимодействиями как между *пользователь-ресурс*, так и *пользователь-пользователь*, *устройство-устройство*.

В целом, *общей схемой* это можно отобразить так:

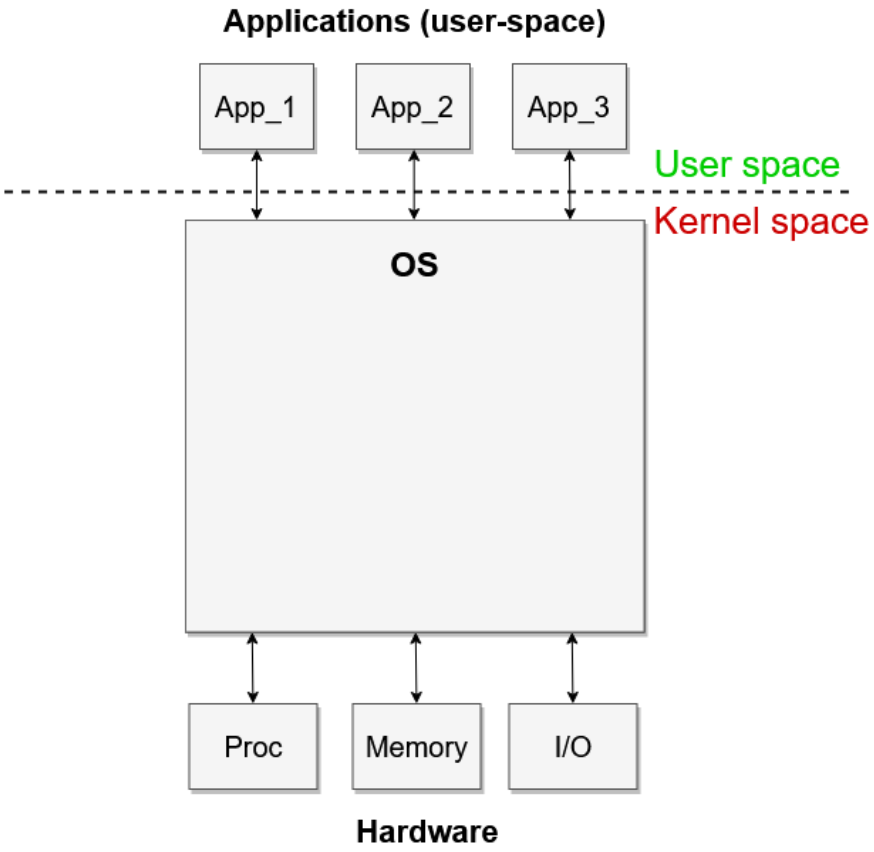


Рис. 3: Рис. 3. Место ОС в компьютерной системе

Примечание:

- App - пользователь ресурсов;
- Hardware - оборудование;
- Proc - Процессор;
- Memory - Оперативная память;
- I/O - Устройства ввода/вывода;

Интерфейс — набор правил и средств взаимодействия двух систем. Иными словами способ взаимодействия.

Kernel space — адресное пространство ядра ОС, в котором процессы имеют привилегированный доступ к ресурсам компьютера и другим процессам.

User space — адресное пространство, отведённое для пользовательских процессов (приложений), то есть не имеющих привилегированный доступ к ресурсам.

Как операционная система загружается в компьютер?

Процесс загрузки операционной системы и вообще компьютера имеет несколько этапов, основные из которых:

1. Запуск компьютера – на процессор подаётся напряжение и его элекрические компоненты начинают работу.
2. Процессор начинает исполнять инструкции с **фиксированного, аппаратно зашитого в него адреса.**
3. По этому адресу находится специальная программа POST – **Power On Self Test**. Которая проверяет работоспособность основных компонентов вычислительной системы.
4. Далее, управление передаётся BIOS'у — Basic Input Output System (Базовая система ввода-вывода), которая инициализирует основные устройства ввода-вывода: загрузочные устройства (раличные хранители информации: HDD, SSD, Flash и так далее), клавиатура, монитор и прочее.
5. BIOS обращается к загрузочному устройству и читает первый блок данных, на котором **должен находиться загрузчик**. Загружает его в память и передаёт ему управление.
6. Загрузчик загружает в память и инициализирует основные компоненты операционной системы и передаёт ей управление.
7. Операционная система запускает таймер, который будет возвращать управление операционной системе каждый, заранее установленный разработчиками ОС, **квант** времени. Это делается для реализация Scheduler'а - планировщика, чтобы ОС могла управлять и контролировать процессы.
8. Операционная система создаёт первый **процесс-пользователя** и дальше от него начинают множится другие процессы.

Что делает ядро ОС?

Ядро ОС – центральная часть операционной системы. По сути, это и есть ОС.

Внимание: Это реакционный механизм, то есть его работа заключается исключительно в реакции на какие-либо события для их последующей обработки.

- **Обрабывает запросы приложений**
  - системные вызовы (программные прерывания)
- **Обрабывает запросы оборудования**
  - прерывания
- **Обрабатывает исключительные ситуации**
  - Разного рода ошибки
- **Обеспечивает диспетчеризацию процессов (scheduling)**
  - **реализация многопользовательского режима доступа к ресурсам**
    - время работы процессора делится на фрагменты и они распределяются по процессам

Примечание: Процессорное время измеряется в тиках или секундах. Часто бывает полезно измерение процессорного времени в *процентах* от мощности процессора, которое называется **загрузкой процессора**.

```
top - 16:54:41 up  5:12,  2 users,  load average: 0.75, 0.77, 0.35
Tasks: 116 total,   1 running, 115 sleeping,   0 stopped,   0 zombie
Cpu(s):  5.3%us,  2.7%sy,  0.0%ni, 91.7%id,  0.3%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:    775540k total,   758548k used,    16992k free,    13920k buffers
Swap:   787144k total,    34724k used,   752420k free,   443552k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 6938 funalien  15   0 70012  29m  18m S  4.0  3.9  10:40.43 ktorrent
 5375 root      15   0 79060  55m 6616 S  2.3  7.3   4:45.84 Xorg
 7869 funalien  15   0 30400  15m  13m S  1.0  2.0   0:00.99 ksnapshot
 5600 funalien  18   0 15252 9700 4528 S  0.3  1.3   0:19.03 pypanel
 5605 funalien  15   0  9704 3592 2968 S  0.3  0.5   1:20.99 conky
 7802 funalien  15   0  228m  75m  23m S  0.3  9.9   0:36.56 firefox-bin
    1 root      15   0  2952 1852  532 S  0.0  0.2   0:01.33 init
    2 root      11  -5     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 migration/0
    4 root      34  19     0    0    0 S  0.0  0.0   0:00.10 ksoftirqd/0
    5 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 watchdog/0
    6 root      10  -5     0    0    0 S  0.0  0.0   0:00.16 events/0
    7 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 khelper
   26 root      12  -5     0    0    0 S  0.0  0.0   0:00.00 kblockd/0
   27 root      20  -5     0    0    0 S  0.0  0.0   0:00.00 kacpid
   28 root      20  -5     0    0    0 S  0.0  0.0   0:00.00 kacpi_notify
  108 root      10  -5     0    0    0 S  0.0  0.0   0:00.00 kseriod
```

Рис. 4: Вывод программы top. Процессорное время каждого процесса (task) указано в колонке «TIME+», «CPU%» - загружаемость процессора относительно его «мощности».

Прерывания

Примечание: Эта часть больше относится непосредственно к аппаратной части, но этот механизм стоит освятить, так как именно это основной аппаратный механизм реализации ОС.

Прерывание – сигнал остановки последовательного выполнения программы, для обработки запроса или реакции на событие.

Чтобы получить код обработки прерывания, в памяти расположена специальная **таблица обработчиков прерываний**, в которой для каждого типа прерывания содержится указатель на тот участок памяти, где расположен соответствующий код обработки данного прерывания.

Инициализация данной таблицы первично осуществялется BIOS'ом в соответствии с архитектурой процессора. После, её инициализирует операционная система для дополнения этой таблицы какими-либо своими прерываниями.

Виды и примеры прерываний:

- **Аппаратные прерывания – с помощью специального контроллера прерываний.**
  - Нажатие на кнопку

  - Заполнение памяти сетевой карты
  - И тд.
- **Программные прерывания (системные вызовы) – вызывается самой программой для вызова т**
  - Открытие/закрытие файла
- Прерывания таймера – для реализации планировщика ОС
- **Исключение**
  - Разного рода ошибки

Как приложения взаимодействуют с ОС?

Взаимодействие процессов с ОС осуществляется с помощью **системных вызовов**.

Примечание: Механизм системных вызовов — это интерфейс, который предоставляет ядро ОС (kernel space) пользовательским процессам (user space).

Системный вызов – программное прерывание, обращение пользовательского процесса к ядру операционной системы для выполнения какой-либо операции.

Например, чтобы выполнить обычное действие, с точки зрения прикладного программиста, – вывод строки в консоль, необходимо загрузить исполнимый код в оперативную память и передать его процессору. С помощью *системных вызовов*, **запускающий** процесс (уже запущенный процесс, из которого вызывается новый процесс — одни процессы порождают другие) обращается к соответствующим сервисам ОС и передаёт им управление для выполнения этих функций.

То есть с помощью **системных вызовов** выполняются те рутинные действия, которые раньше осуществлялись вручную, — загрузка кода программы в память, передача его на исполнение процессору и прочее.

Схема организации ОС расширяется добавлением интерфейса для взаимодействия приложений с ядром ОС — механизмом системных вызовов:

Как оборудование взаимодействует с ОС?

Оборудование взаимодействует с ОС с помощью **аппаратных прерываний**. И одна из функций ОС — **абстрагирование оборудования**.

Что это значит?

У каждого оборудования есть свой фиксированный интерфейс. Например, операции с флешкой, жестким диском, сетевой платой и многими другими будут похожи по своему типу - «записать/считать данные». Но у каждого устройства для этого, тем не менее, будет свой особенный и отличный от других интерфейс. То есть эти **однотипные** действия нужно будет выполнять для разных устройств по разному.

ОС должка выполнять одни и те же операции над разными типами устройств. И чтобы она выполняла их однообразно — нужно чтобы был **общий интерфейс**. Реализацией этого общего интерфейса занимаются специальные программы - **драйверы устройств**. То есть, ОС обращается к драйверам

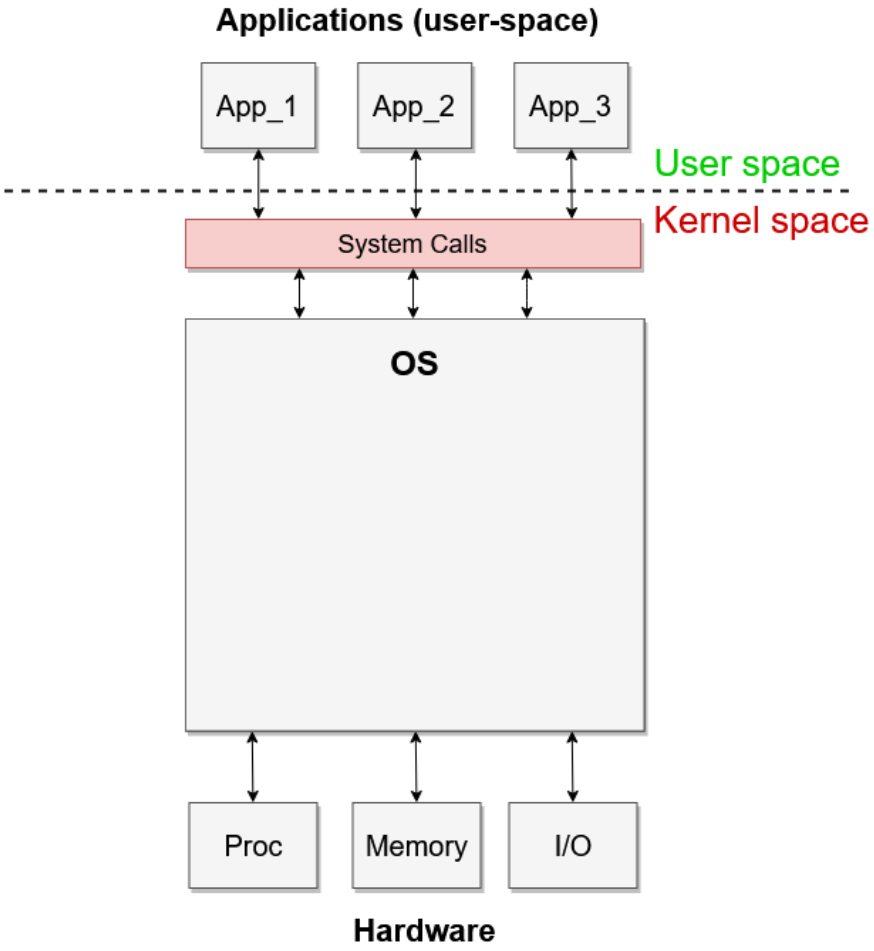


Рис. 5: Рис 4. Интерфейс системных вызовов

устройств используя однотипные команды «отправить команду/считать/записать», а драйвера уже превращает эти команды в то, что понимает конкретное устройство.

Схема организации ОС расширяется добавлением интерфейса взаимодействия ОС и оборудования - специальные программы «драйвера»:

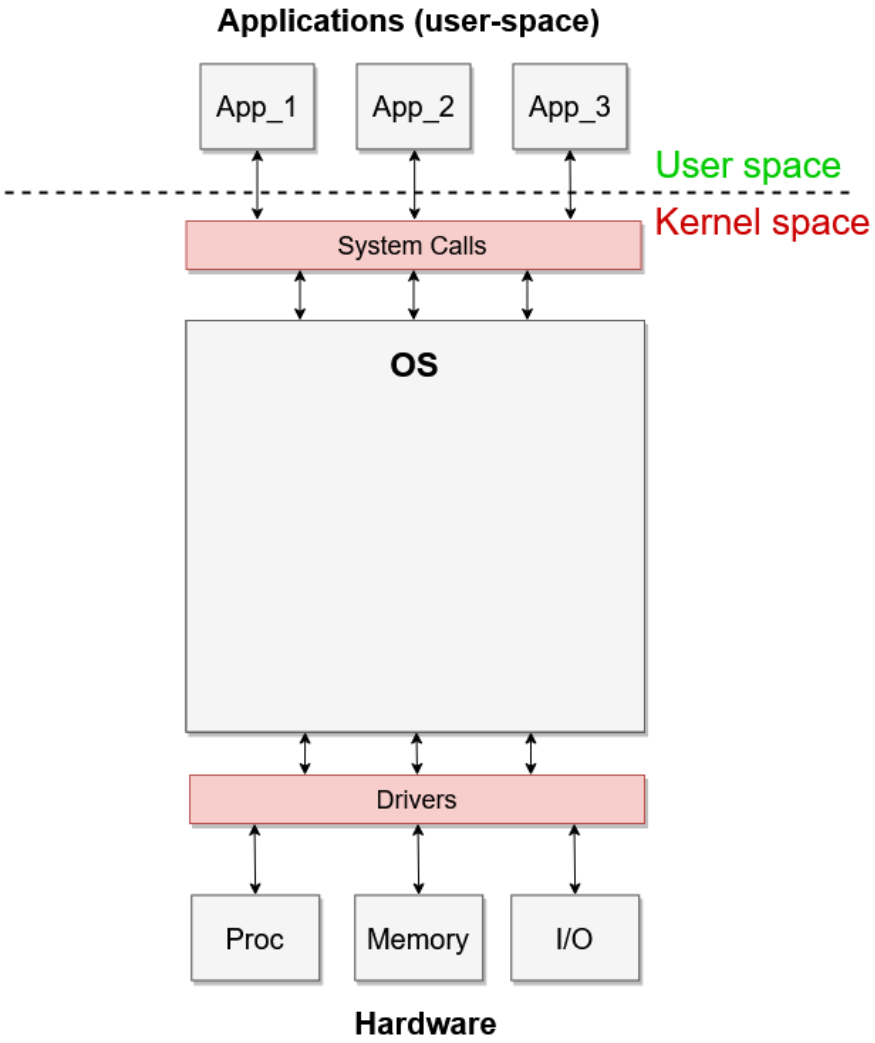


Рис. 6: Рис 5. Интерфейс драйверов

Сервисы ОС

Функции ОС заключены в её сервисах (модулях). Реализация организации которых зависит от архитектуры ядра. Рассмотрим на примере монолитного ядра:

Основные

- **Управление процессами (Process scheduler - планировщик)**
  - Запуск (помещение на процессор, выделение процессорного времени)
  - Приостановка (заморозка)

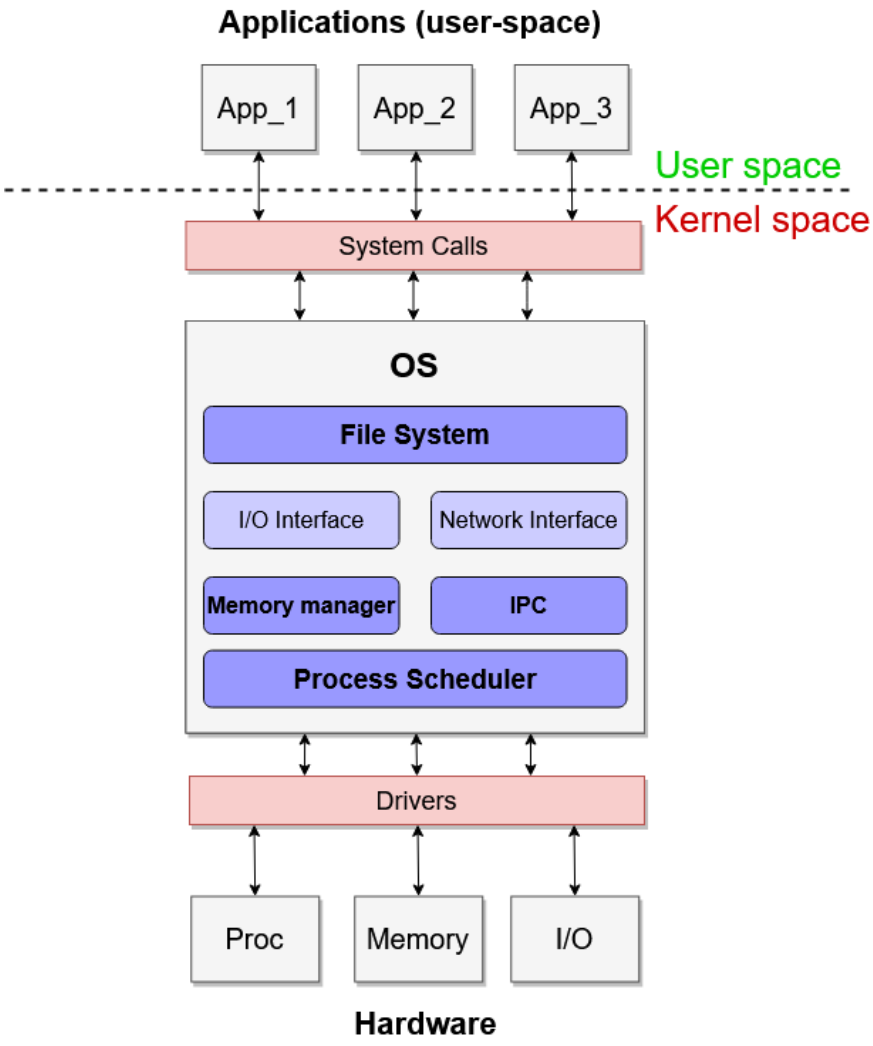


Рис. 7: Рис 6. Основные компоненты ОС

  - Завершение
  - Изменение приоритета

Примечание: Как говорилось в части о загрузке ОС, реализация планировщика осуществляется с помощью **прерывания по таймеру** — каждый **квант времени** происходит прерывание, которое передаёт управление ОС и она анализирует состояние всех процессов и что с каким процессом сделать: запустить, приостановить, завершить или изменить приоритет.

- **Межпроцессное взаимодействие (IPC - Inter Process Communication)**
  - Общая память для нескольких процессов (shared memory)
  - Способы обмена данными через те или иные механизмы (file, pipe, signals)
  - Сетевое взаимодействие
  - Механизмы предотвращения коллизий и синхронизации (семафоры, мьютексы)
- **Управление памятью (Memory manager)**
  - Динамическое выделение памяти (Memory allocation)
  - Создание иллюзии уникальности адресного пространства для каждого процесса
  - Механизм виртуальной памяти

Дополнительные

- **Файловая система (File system)**
  - Файлы и их содержимое
  - Каталоги и директории
- **Доступ к оборудованию и управление им**
  - Прерывания
  - Драйвера
- **Модель безопасности**
  - Пользователи («юзеры») и их группы
  - Права доступа
- **Разное**
  - Интерфейс ввода-вывода (I/O Interface)
  - Сетевой интерфейс (Network Interface)

Основные абстракции

Процесс

Процесс — совокупность инструкций и данных, что находятся в оперативной памяти и обрабатываются процессором. Другими словами - исполнение программы в целом (не путать с *потоком исполнения*).

Примечание: Компьютерная программа сама по себе — лишь пассивная последовательность инструкций. В то время как процесс — непосредственное выполнение этих инструкций.

В рамках ОС, это абстракция, которая предоставляет иллюзию *персональной машины*. То есть то, что данный исполнимый код полностью владеет всеми вычислительными ресурсами машины.

Состояние (контекст) процесса

***С внешней стороны*, процесс можно описать следующим:**

- **Состояние**
  - Состояние памяти
  - Содержимое регистров процессора
- Адрестное пространство — у каждого процесса своё.
- Состояние исполнения — то, исполняется ли этот процесс на процессоре в данный момент или ожидает чего-либо.
- CPU - величина использовния процессорного времени.

Изнутри, процесс можно условно разделена на четыре части: **Stack, Heap (кучу), Text (код) и данные (Data)**.

Состояния исполнения

Когда процесс выполняется, он проходит через разные состояния. Эти этапы могут различаться в разных операционных системах.

Общая картина выглядит так:

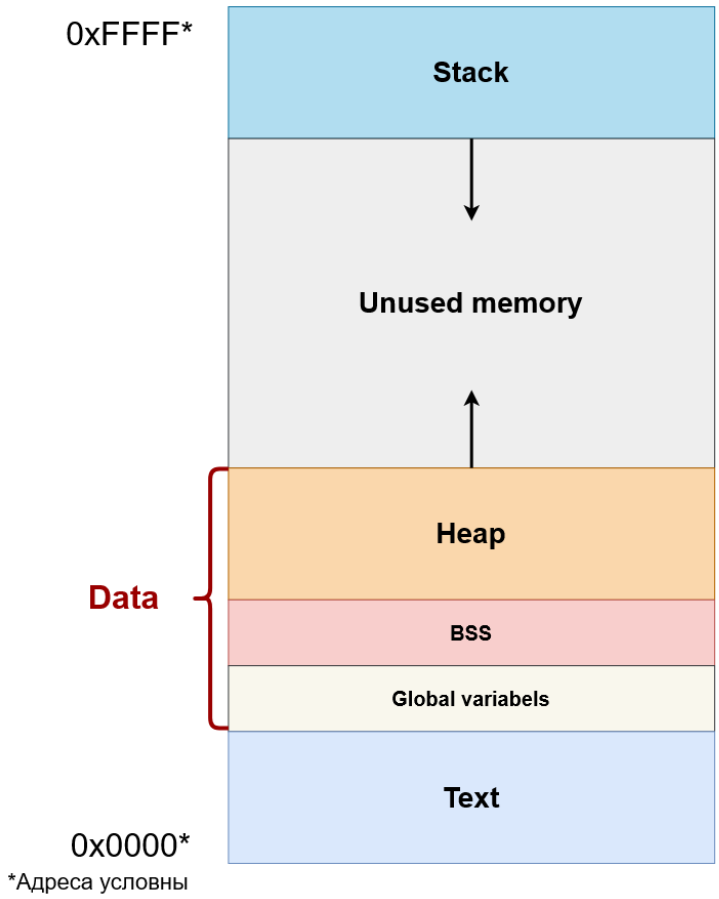
Примечание:

- Новый: начальное состояние при создании процесса.
- Готов: процесс ожидает исполнения на процессоре. В течение работы процессор может переключаться между процессами, переводя одни в режим готовности, другие – в режим исполнения.
- Исполнение: непосредственное выполнение инструкций на процессоре.
- Ожидает: процесс переходит в состояние ожидания. Например, ждёт ввода данных или получения доступа к файлу.
- Завершен: как только процесс завершится, он перейдёт в это состояние и будет ожидать удаления.

Информация о процессе

Вся информация о процессе содержится в специальной структуре данных, поддерживаемой операционной системой для каждого процесса – PCB (Process Control Block) - Блок управления процессов.

Примечание:



Stack

Временные данные – параметры функции, адрес возврата и локальные переменные Растёт от старших адресов к младшим (одна из популярнейших реализаций).

Сегмент данных – Data

Heap (куча) – динамически выделяемая под объекты память. Растёт противоположно ст

BSS – глобальные переменные без инициализации.

Global variables – инициализированные глобальные переменные.

Сегмент кода – Text

Инструкции для процессора, а также: *Состояние регистров, состояние программного счетчика, режим работы процессора, незавершенные операции ввод вывода, информация о выполненных системных вызовах.*

Рис. 8: Рис 7. Сегменты памяти процесса

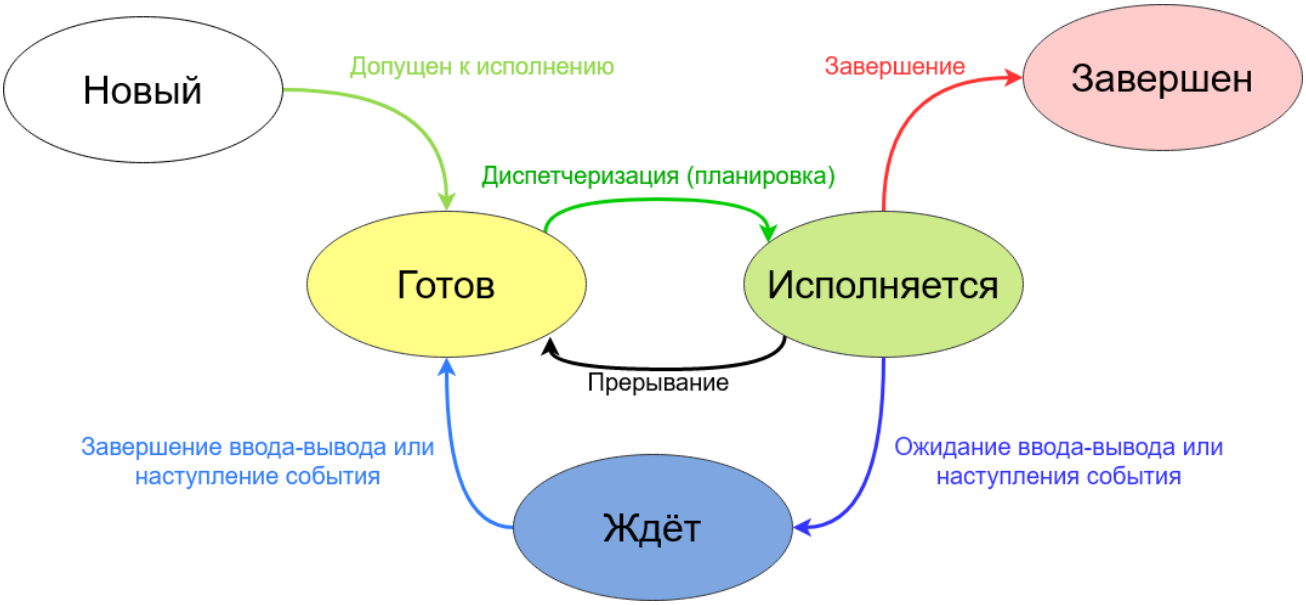


Рис. 9: Рис 8. Состояния исполнения процесса

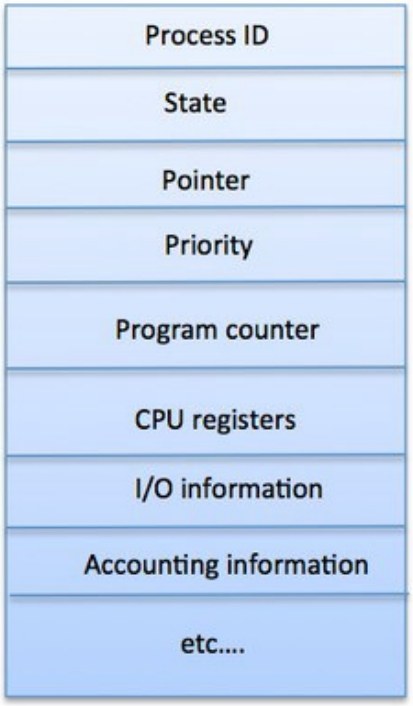


Рис. 10: Рис 9. Process Control Block

- Process ID: идентификатор каждого из процессов в ОС.
- State: текущее состояние процесса.
- Privileges: разрешения доступа к системным ресурсам.
- Pointer: указатель на родительский процесс.
- Priority: приоритет процесса и другая информация, которая требуется для планирования процесса.
- Program Counter: указатель на адрес следующей команды, которая должна быть выполнена.
- CPU registers: регистры процессора, необходимые для состояния исполнения.
- Accounting Information: уровень нагрузки на процессор, статистика и другие данные.
- I/O Information: список ресурсов, использующих чтение и запись.

Информацию о процессах в целом, ОС хранит в специальной таблице процессов.

Поток

Процесс может делиться на **потоки (threads)**. Они обеспечивают параллелизм, то есть одновременное исполнение нескольких потоков инструкций, на уровне программы.

Поток выполнения (нить, thread) — последовательность исполнения инструкций. Ход исполнения программы**.

Процесс является **контейнером ресурсов** (адресное пространство, процессорное время и тд), а **поток – последовательность инструкций, которые исполняются внутри этого контейнера**.

Примечание: Реализация потоков выполнения и процессов в разных операционных системах отличается друг от друга, но в большинстве случаев поток выполнения находится внутри процесса.

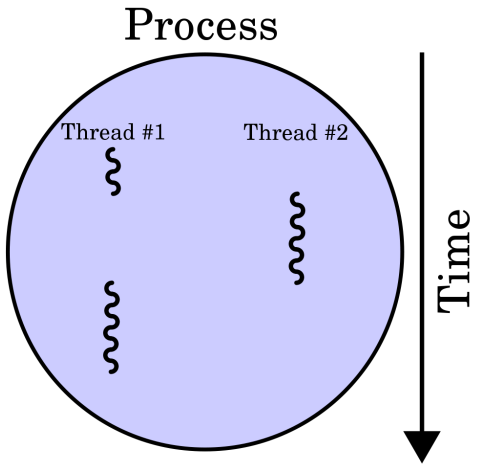


Рис. 11: Рис 10. Многопоточный процесс

Потоки, существующие в рамках одного процесса, в его адресном пространстве, могут совместно использовать ресурсы процесса, например такие как память или файл. Тогда как процессы не разделяют этих ресурсов, так как каждый существует в своём адресном пространстве.

Также поток называют **легковесный процесс**.

Сегодня потоки широко применяются в работе серверов и многопроцессорных устройств с общей памятью.

Рассмотрим на примере утилиты htop.

Примечание: **PID** — Process ID; Уникальное число идентификатор для каждого процесса

TGID — Tread Group ID; Индентификатор группы потоков

На скриншоте, *процесс 2881* имеет множество потоков, отношение которых к нему можно определить по тому, что **TGID у этих потоков имеет значение PIDa этого процесса - 2881**. Таким образом, один процесс разбивается на множество потоков, в которых инструкции исполняются параллельно.

Чем хороши потоки

- Они минимизируют время переключения контекста (процессора).
- Их использование обеспечивает параллелизм процесса.
- Они эффективно общаются между собой.
- Потоки позволяют использовать многопроцессорные архитектуры в большем масштабе.

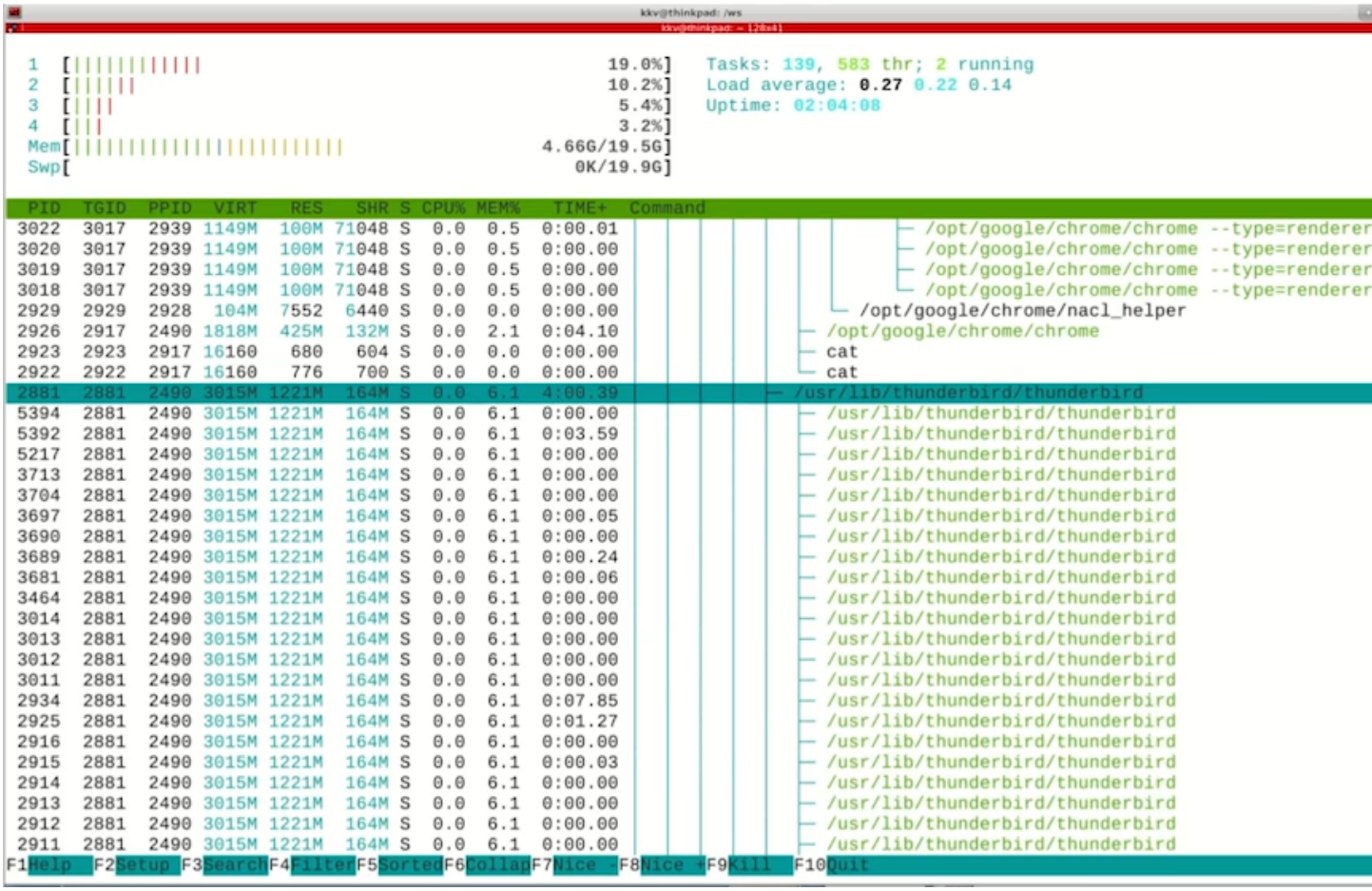


Рис. 12: Рис 11. Вывод утилиты мониторинга процессов htop

Файл

Это очень широкое и многогранное понятие. Но если выделить наиболее общее, то получится, что **файл** — это универсальный системный интерфейс для обращения к тем или иным данным.

А **файловая система** — это система имён. То есть возможность выделять те или иные объекты данных и присваивать им имена, а также выделять иерархию.

По другому, файл и файловую систему можно также представить как структуру данных и связи между этими структурами.

Реализация многозадачности

Осуществляется при помощи следующих механизмов:

- **Прерывание таймера**
  - когда при запуске, ОС программирует таймер на то, чтобы он каждый **квант времени** передавал управление ОС.
- **Переключение контекста**
  - сохранение состояния исполнения (регистров) процесса и установка на процессор контекста другого процесса.
- План блокировок, при наличии нескольких CPU
- Освобождение ресурсов при завершении процесса

Переключение контекста

Контекст процесса — это состояние регистров, при его выполнении на процессоре.

Следовательно, **переключение контекста** — это смена контекста одного процесса, на контекст другого, без потери данных сменяемого процесса - то есть, чтобы его потом можно было восстановить с того момента, где он был переключён.

Примечание: Например, у нас на процессоре в данный момент времени выполняется «процесс 1» - в регистрах хранятся данные, которые относятся к этому процессу.

Но происходит прерывание и «процесс 1» снимается с выполнения на процессоре, чтобы вместо него выполнялся «процесс 2». Следовательно, нужно заполнить регистры уже теми данными, что относятся к «процессу 2».

Однако, «процесс 1» ещё не выполнился полностью, и для дальшейнего исполнения ему нужны те данные, что хранились в регистрах при прерываний, то есть необходим его контекст. Операционная система должна обеспечивать подобные смены контекстов без потери данных.

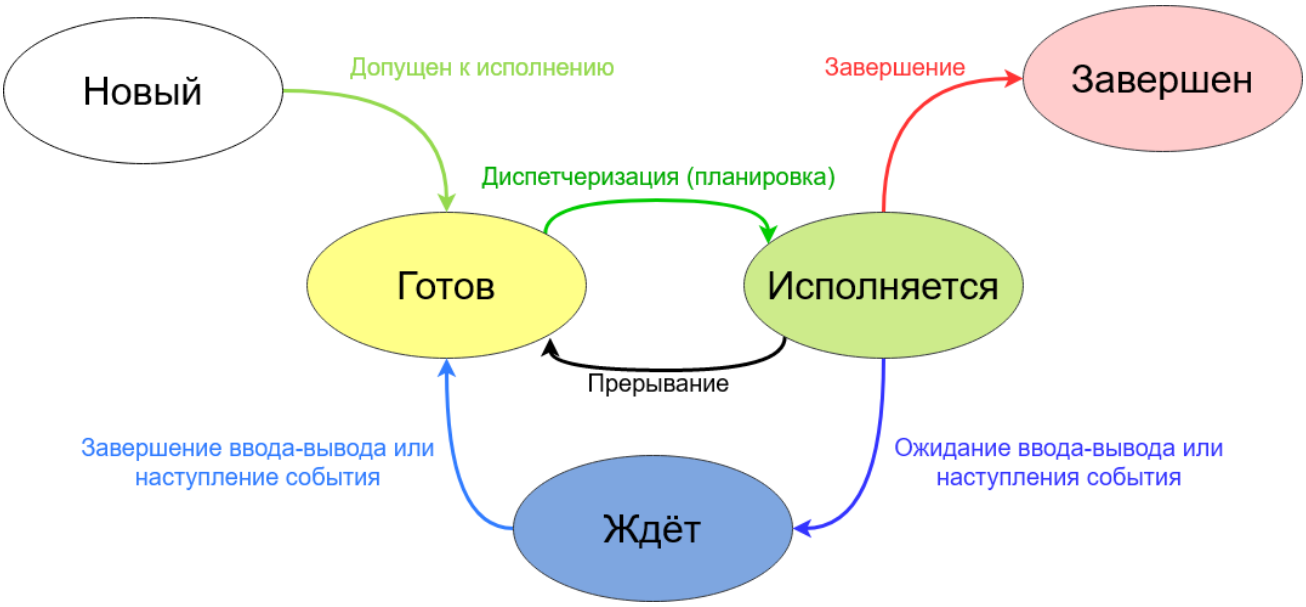
При переключений контекста возникает три важных вопроса:

1. Как?
2. Когда?
3. Между кем и кем?

Как?

1. Значения регистров процесса записываются в Stack этого же процесса в оперативной памяти. Таким образом, процесс в своей памяти хранит свой же контекст.
2. Контекст планировщика появляется на процессоре, выполняет анализ имеющихся процессов.
3. Переключает процессор на контекст уже другого, нового процесса.

В целом, смена контекста происходит между состояниями «Готов», «Ожидает» и «Исполняется».



Критические секции и блокировки

Одна из основных проблем с которыми может столкнуться такая система с вытесняющей многозадачностью — **порядок доступа процессов к их общим ресурсам**.

Возникают, так называемые, критические секции – участки исполняемого кода программы, в которых производится доступ к общему ресурсу (данным или устройству), который не должен быть одновременно использован более чем одним **потоком выполнения**.

Может возникнуть такая ситуация, когда один поток, **«потребитель»**, начинает использовать данные, которые должен подготовить другой поток, **«производитель»**, но этот производитель ещё не закончил их подготовку и снялся с исполнения. Таким образом, «потребитель» использует некорректные данные, что с высокой долей вероятности приведёт к ошибке.

Данная проблема решается с помощью **механизма блокировок** – когда поток, получивший доступ к ресурсу, блокирует его, не давая другим потокам пользоваться этим захваченным ресурсом до разблокировки. То есть, если один поток хочет захватить (заблокировать) ресурс, а он уже занят другим потоком, то первый будет ожидать пока этот другой поток-владелец сам не освободит этот ресурс.

Примечание: Все эти механизмы обеспечиваются операционной системой

Чтобы лучше это понять, можно обратиться к аналогии с туалетом — им может пользоваться только один человек. Если другой хочет им воспользоваться, то ему нужно дождаться когда его освободит уже им пользующийся.

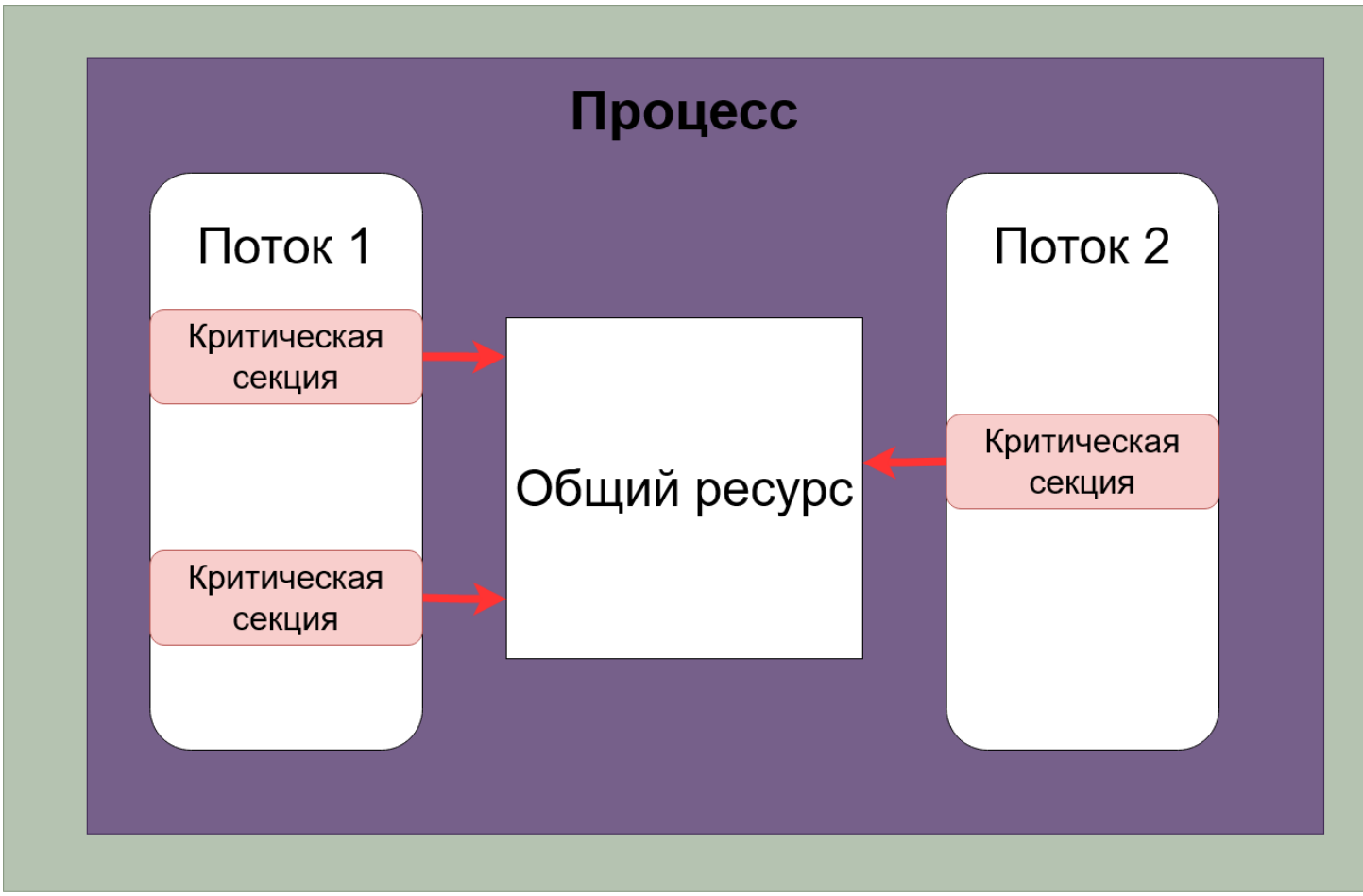


Рис. 13: Рис 12. Критические секции в потоках процесса

Deadlock - взаимная блокировка

Из-за блокировки, захвата ресурса может возникнуть другая проблема: **Deadlock** - взаимная блокировка.

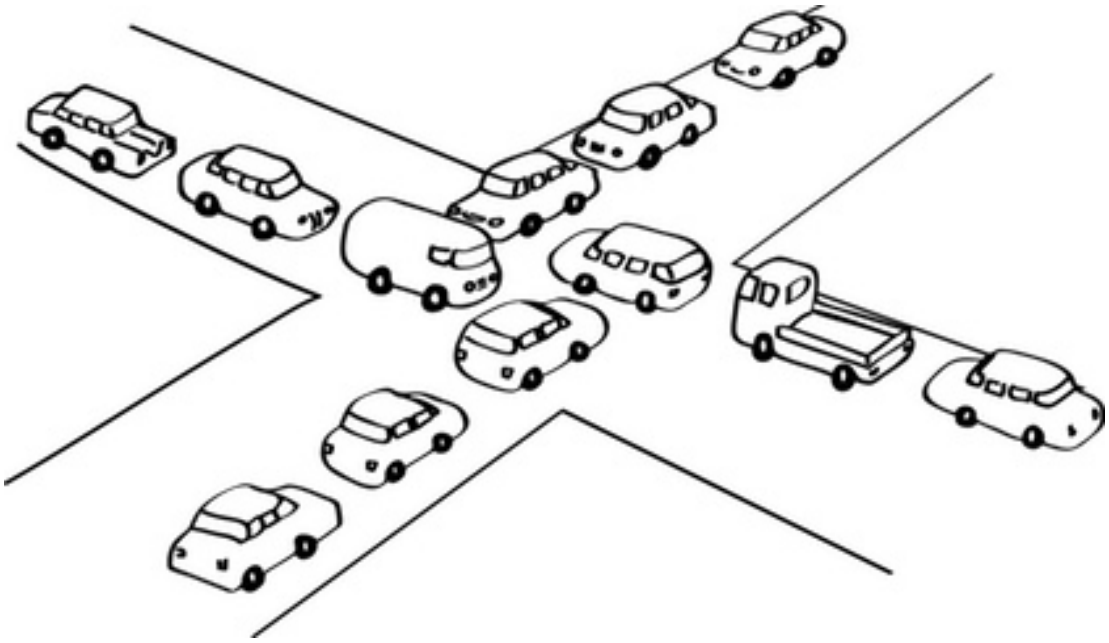


Рис. 14: Рис 13. Аналогия пробки на перекрёстке с Deadlock

Deadlock — ситуация, при которой несколько потоков находятся в состоянии ожидания ресурсов, занятых друг другом, и ни один из них не может продолжать свое выполнение.

Пример Deadlock'а на псевдокоде

Шаг	Поток 1	Поток 2
0	Хочет захватить A и B, начинает с A	Хочет захватить A и B, начинает с B
1	lock(A) — Захват A	lock(B) — Захват B
2	lock(B) — Ожидает освобождения ресурса B	lock(A) — Ожидает освобождения ресурса A
...	**DEADLOCK –> Далее код не выполнится, так как произошел Deadlock в коде выше**	
n	unlock(A) — освобждение A	unlock(B) — освобждение B
n+1	unlock(B) — освобждение B	unlock(A) — освобждение A

Схематично, Deadlock можно изобразить так:

Проблема Deadlock-ов решаема и существует множество способов сделать это - начиная от недопущения подобных ситуаций в самом коде программы, заканчивая механизмами операционной системы, которые разрешают подобные ситуации. Но, однозначно исключить вероятность возникновения подобных ситуаций нельзя. Однако, это отдельная достаточно большая тема.

Адреса и управление памятью

Тема адресации очень сложна, поэтому здесь я лишь проведу краткий обзор общей технологии.

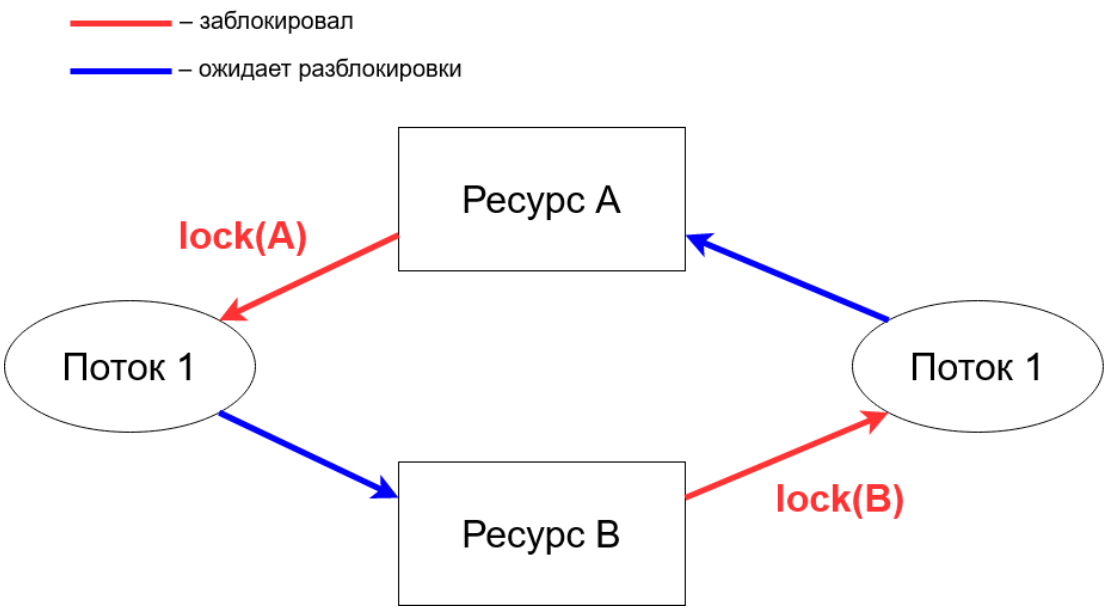


Рис. 15: Рис 14. Deadlock

Существует два вида адресов:

- Физический адрес – непосредственно адрес внутри микросхемы памяти, который передаётся по шине памяти.
- Логический адрес – тот адрес, которым оперирует процесс.

Чтобы отобразить логический адрес в физический, существует специальный **аппаратный механизм**.

В соответствии с этим, процессор может работать в двух режимах:

1. Реальный режим
2. Защищенный режим

Итог

Начав с базовой архитектуры и небольшой истории развития компьютера, мы разобрали причины появления такого комплекса программ как **операционная система** и выделили главную её цель – **обеспечение согласованного доступа к ресурсам компьютера множеству пользователям этих ресурсов, а также управление как самими ресурсами, так и пользователями**.

Основные механизмы (сервисы)

Рассмотрели основные механизмы реализации этой цели: *Scheduler (планировщик), Inter Process Communication (межпроцессное взаимодействие), Memory manager (управление памятью) и другие.*

Абстракции

Ряд абстракций, которые вводит ОС: *Process (процесс), Thread (поток исполнения), File (файл).*

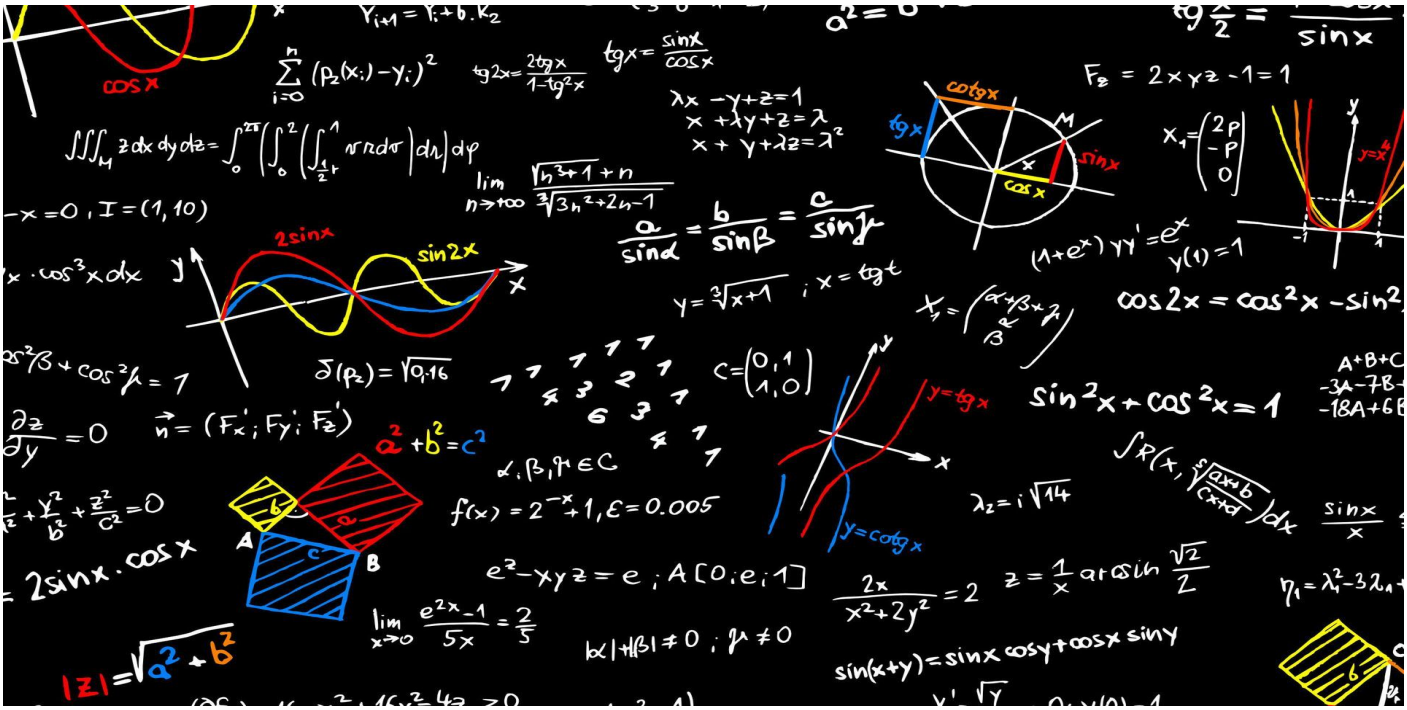
Заключение

Ух, и вот наконец-то я закончил писать этот материал. Надеюсь, вам было интересно и полезно.

Если вы хотите как-то дополнить материал, дать критику по его содержанию или структуре – пожалуйста, пишите в Issue данного репозитория поднимая ту или иную тему для обсуждения и доработки.

Также, вы можете сделать Fork данного репозитория и после внести свои дополнения с помощью Pull Request. Спасибо за внимание!

1.2.2 Математика



1.2.2.1 Курсы

Базовая математика

$$\alpha_t(i) = P(O_1, O_2, \ldots O_t, q_t = S_i \lambda)$$

Описание курса

Последовательный разбор основ математики, которые в том или ином объёме даются в общеобразовательной школьной программе.

Ресурсы

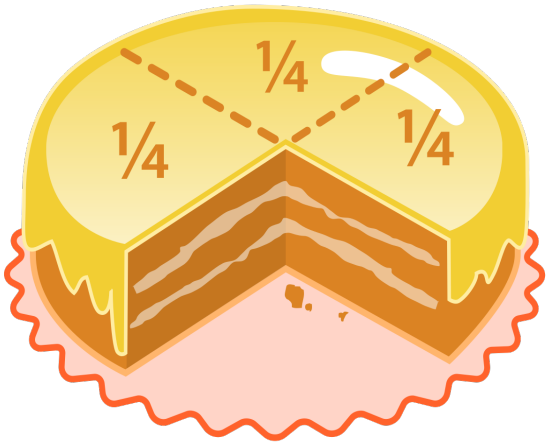
- Базовый курс подготовки к ОГЭ по математике

Структура курса

Обыкновенные дроби

Зачем нужны обыкновенные дроби?

Например, представим, что у вас день рождения, и к вам пришли 4-ро друзей. Разделив торт среди них всех поровну, каждый получит *«одну четвёртую торта»* – $\frac{1}{4}$. Вот вам и дроби в повседневной жизни :)



А вообще, это **важная математическая абстракция**.

Определение

Обыкновенная дробь (*англ. Ordinary fractions*) – это выражения вида $\frac{m}{n}$, где m и n – **натуральные числа**.

То есть, выражаясь математическим языком: $m, n \in \mathbb{N}$.

Примечание: Символ $\in$ – обозначает принадлежность к какому-то множеству.

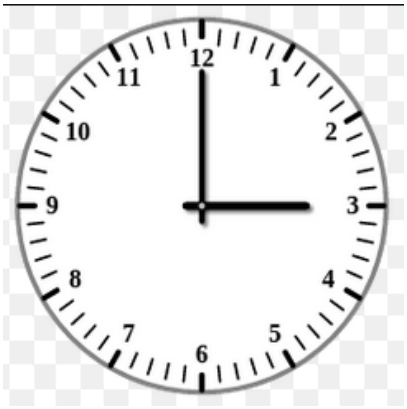
Например, утверждение *«Все кошки (К) есть животные (Ж)»* с помощью математической записи можно запистаь так: $\in$ – «**Кошки**» являются элементом множества «**Животные**».

Где $\frac{m}{n} =$

число над чертой называется *числителем*, а число под чертой — *знаменателем*. Первый играет роль делимого, второй — делителя.

Обыкновенные дроби с целыми числителями и знаменателями образуют поле **рациональных чисел**.

Пример использования обыкновенной дроби:



Допустим, сейчас 3 часа ночи. Какая часть суток прошла?

В сутках 24 часа, а прошло 3 часа из этих 24, то получается *«три двадцатьчетвёртых»* – $\frac{3}{24}$. А если сократить эту дробь на **3**, то получим *«одну восьмую»* – $\frac{1}{8}$. Прошла $\frac{1}{8}$ суток.

Типы дробей

Обыкновенные дроби бывают нескольких видов:

1. **Правильные**
   - $<$.
   - Например числа: $\frac{1}{2}, \frac{3}{4}, \frac{1}{8}$ и так далее.
2. **Неправильные**
   - $>$
   - Например числа: $\frac{10}{9}, \frac{5}{3}, \frac{14}{3}$ и так далее.
3. **Смешанные**
   - **Дроби вида $a + \frac{b}{c}$ называют смешанными. То есть здесь выделяется целая часть.**
     - *Пример:* $1 + \frac{1}{9}$.
   - **Как правило, в записи этих чисел знак «+» опускают и пишут $a\frac{b}{c}$.**
     - *Пример:* $1\frac{1}{9}$.
   - **Всякую смешанную дробь можно представить в виде обыкновенной: $a + \frac{b}{c} = \frac{ac+b}{c}$.**
     - *Пример:* $1 + \frac{1}{9} = \frac{1 \cdot 9 + 1}{9} = \frac{10}{9}$.
   - Например числа: $1\frac{1}{9}, 7\frac{3}{4}$ и так далее.

Основное свойство дроби

При умножении или делении числителя и знаменателя на одно и то же число – значение дроби не меняется.

$$\frac{a}{b} = \frac{a \cdot c}{b \cdot c}$$

Сложение и вычитание дробей

С одинаковыми знаменателями

Если дроби имеют одинаковый знаменатель, то суммой/разностью этих дробей будет дробь, числитель которой равен сумме/разности числителей исходных дробей, а знаменатель равен исходному знаменателю.

$$\frac{a}{c} \pm \frac{b}{c} = \frac{a \pm b}{c}$$

Примечание: Пример вычитания дробей. Найдите значение выражения: $4\frac{2}{3} - \frac{5}{3}$

Решение:

1. **Сначала представим в виде неправильной дроби дробь:** $4\frac{2}{3}$
   - $4\frac{2}{3} = \frac{4 \cdot 3 + 2}{3} = \frac{14}{3}$
2. **Теперь можем вычесть из одной дроби другую:**
   - $4\frac{2}{3} - \frac{5}{3} = \frac{14}{3} - \frac{5}{3} = \frac{9}{3} = 3$
3. Ответ: 3

С разными знаменателями

Если дроби имеют разные знаменатели, то **сначала нужно воспользоваться** *основным свойством дробей* **и привести их к общему знаменателю, а потом просто воспользоваться правилом сложения/вычитания дробей**.

$$\frac{a}{c} \pm \frac{b}{d} = \frac{a \cdot d \pm b \cdot c}{c \cdot d}$$

Примечание: Пример сложения дробей. Сложите дроби: $\frac{2}{5} + \frac{3}{4}$

Решение:

1. **Нам даны дроби с разными знаменателями, поэтому сначала приведем их к общему знаменателю**
   - $\frac{2}{5} = \frac{2 \cdot 4}{5 \cdot 4} = \frac{8}{20}$
   - $\frac{3}{4} = \frac{3 \cdot 5}{4 \cdot 5} = \frac{15}{20}$
2. **Теперь, когда мы получили дроби с одинаковыми знаменателями, можем сложить их:**
   - $\frac{2}{5} + \frac{3}{4} = \frac{8}{20} + \frac{15}{20} = 1\frac{3}{20}$
3. Ответ: $1\frac{3}{20}$

Внимание: При сложении и вычитании смешанных чисел, можно отдельно сложить (вычесть) целые и дробные их части. Например, $2\frac{1}{6} + 5\frac{5}{12} = (2 + 5) + \frac{2+5}{12} = 7\frac{7}{12}$

Умножение и деление дробей

Умножение

Произведение двух обыкновенных дробей есть **дробь, числитель которой равен произведению числителей исходных дробей, а знаменатель произведению знаменателей исходных дробей.**

$$\frac{a}{b} \cdot \frac{c}{d} = \frac{a \cdot c}{b \cdot d}$$

Примечание: Пример умножения дробей. Вычислим: $\frac{3}{5} \cdot 2\frac{1}{4}$

Решение:

1. **Сначала представим дробь $2\frac{1}{4}$ в виде неправильной:**
   - $2\frac{1}{4} = \frac{9}{4}$
2. **Теперь перемножим одну дробь на другую:**
   - $\frac{3}{5} \cdot \frac{9}{4} = \frac{3 \cdot 9}{5 \cdot 4} = \frac{27}{20}$
3. **Выделим целую часть, если того требует задание:**
   - $\frac{27}{20} = 1\frac{7}{20}$
4. **Ответ:**
   - $1\frac{7}{20}$

Деление

Частное двух обыкновенных дробей равно **произведению первой дроби на перевёрнутую вторую дробь.**

$$\frac{a}{b} : \frac{c}{d} = \frac{a \cdot d}{b \cdot c}$$

Внимание: При умножении и делении смешанных чисел необходимо представлять эти числа в виде обыкновенных дробей!

Прочие свойства

- Дроби $\frac{a}{b}\frac{b}{a}$ являются *взаимно обратными*. **Произведение взаимно обратных дробей равно единице:** $\frac{a}{b} \cdot \frac{b}{a} = 1$

Основные формулы

1. **Сложение/вычитание дробей:**
   - $\frac{a}{c} \pm \frac{b}{c} = \frac{a \pm b}{c}$

- $\frac{a}{c} \pm \frac{b}{d} = \frac{a \cdot d \pm b \cdot c}{c \cdot d}$

2. **Умножение/деление дробей:**
   - $\frac{a}{c} \pm \frac{b}{d} = \frac{a \cdot d \pm b \cdot c}{c \cdot d}$
   - $\frac{a}{b} : \frac{c}{d} = \frac{a \cdot d}{b \cdot c}$
3. **Взаимная обратность:**
   - $\frac{a}{b} \cdot \frac{b}{a} = 1$

1.2.2.2 Конспекты

1.2.3 Web-технологии

1.2.3.1 Конспекты

Доработанные конспекты-статьи по лекциям. Можно использовать как совместно с промотром лекции, для закрепления, так и отдельно от лекции.

Примечание: В какой-то степени конспекты даже лучше: читать быстрее, более подробная и структурированная информация с ссылками на другие источники, обилие поясняющих схем.

1.2.3.2 Курсы

Web-технологии. Mail.ru Group

Курс на Stepik.

Описание курса

Курс Web-технологии посвящен разработке web-приложений среднего масштаба, иначе говоря сайтов.

Курс является в большой мере обзорным и знакомит будущих web-разработчиков с широким спектром технологий и общими принципами работы web-приложений.

Что останется за рамками курса?

- Сетевые технологии
- Все используемые языки программирования
- Frontend разработка

Структура курса

Архитектура Web приложений

Web-приложение – *клиент-серверное* приложение, в котором клиент взаимодействует с веб-сервером при помощи браузера.

Разделение между Internet и WWW

Что такое Internet?

Internet – глобальная система объединённых компьютерных сетей. Другими словами - **сеть сетей**.

Инфраструктура, которая позволяет передавать любые типы данных между сетями и устройствами, подключенными к ней.

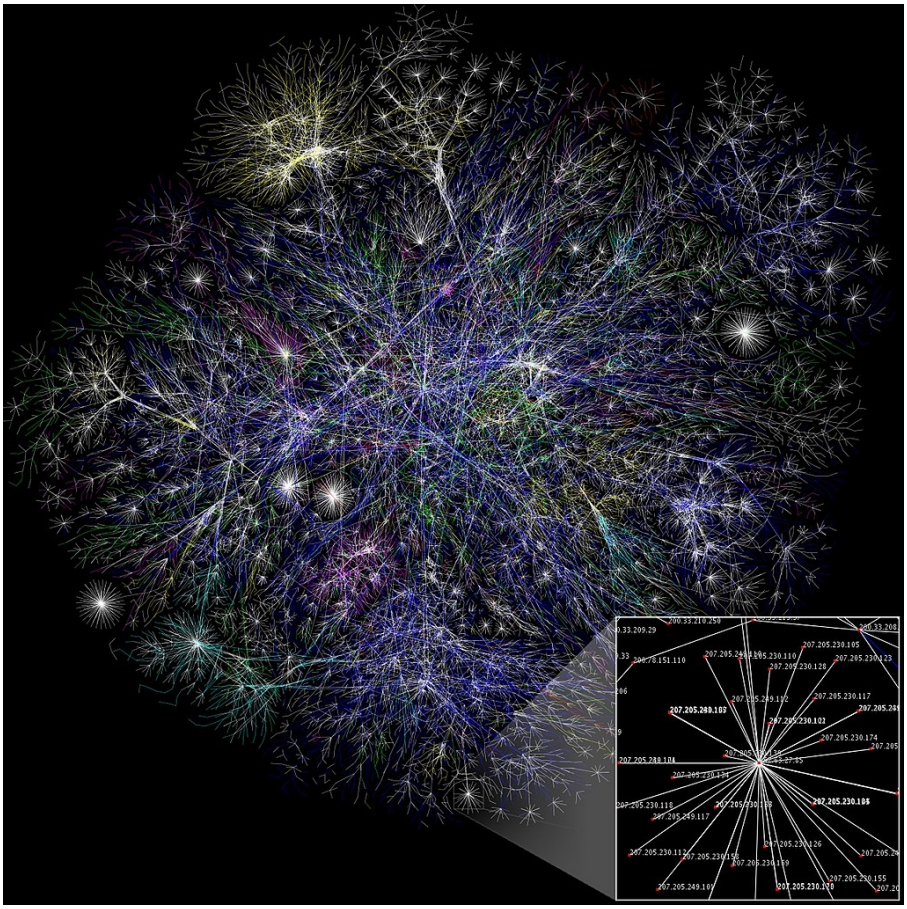


Рис. 16: Частичная карта Интернета

Эта инфраструктура поддерживается **сетевыми протоколами** – алгоритмами, которые реализуют различные сервисы по передачи данных.

Существует несколько стеков сетевых протоколов, основные: **OSI** и **ICP/IP**. Рассмотрим на примере ICP/IP.

Из этого множества протоколов и уровней, в Web'е нас больше всего интересуют протоколы *прикладного* уровня (*приложений*). В частности: **HTTP, FTP, DNS, SSH, P2P**. Но для полноты понимания немного затронем и другие уровни.

Примечание:

Прикладной уровень:

- HTTP – протокол передачи гипертекстовых документов.

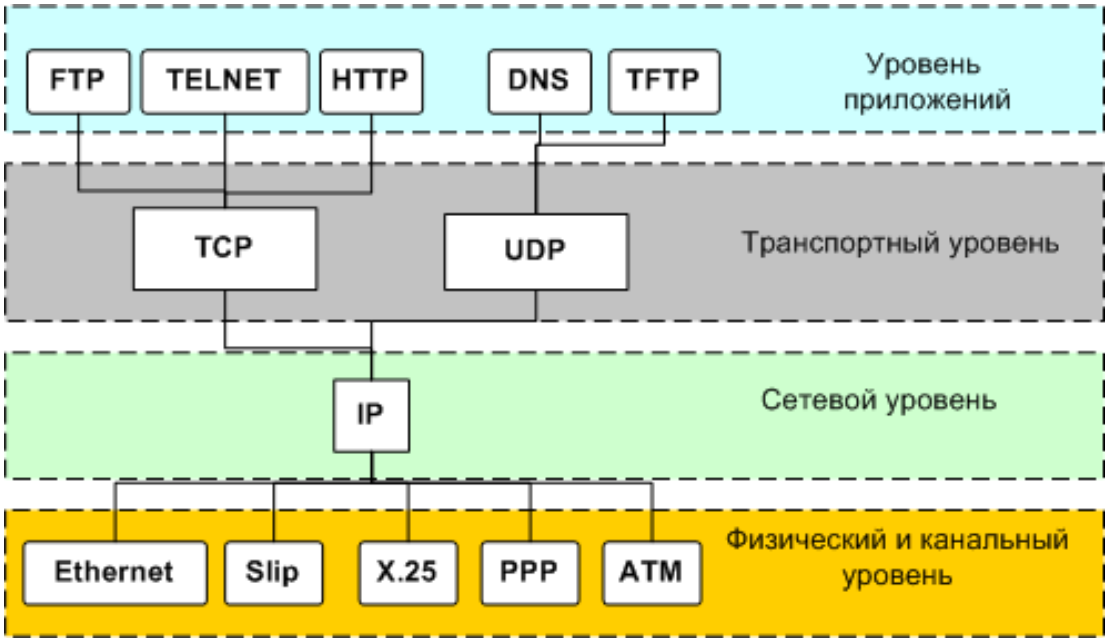


Рис. 17: Стэк TCP/IP (представлены не все протоколы)

- FTP – протокол передачи файлов.
- DNS – доменная система имён.
- SSH – защищенное подключение к консоли, удалённое управление операционной системой.
- P2P (Peer-to-Peer) – каждый узел является как клиентом, так и выполняет функции сервера.

Транспортный уровень:

- TCP – надежная последовательная передача данных. С подтверждением получения.
- UDP – не надежная передача данных. Без подтверждения получения.

Сетевой уровень:

- IP – глобальная адресация, передача в разнородной среде.

Что такое WWW?

World Wide Web – распределённая система взаимосвязанных, посредством гиперссылок, документов и их ресурсов, располагающихся на машинах подключенных к Internet. Эта система посредством протоколов серверного и клиентского ПО обеспечивает доступ к этим вазимосвязанным документам.

Примечание: Распределённая – означает, что физически, документы и их ресурсы могут располагаться на разных серверах.

Примечание: **Историческая справка** – изначально, WWW разрабатывалась как система ссылок в научном сообществе.

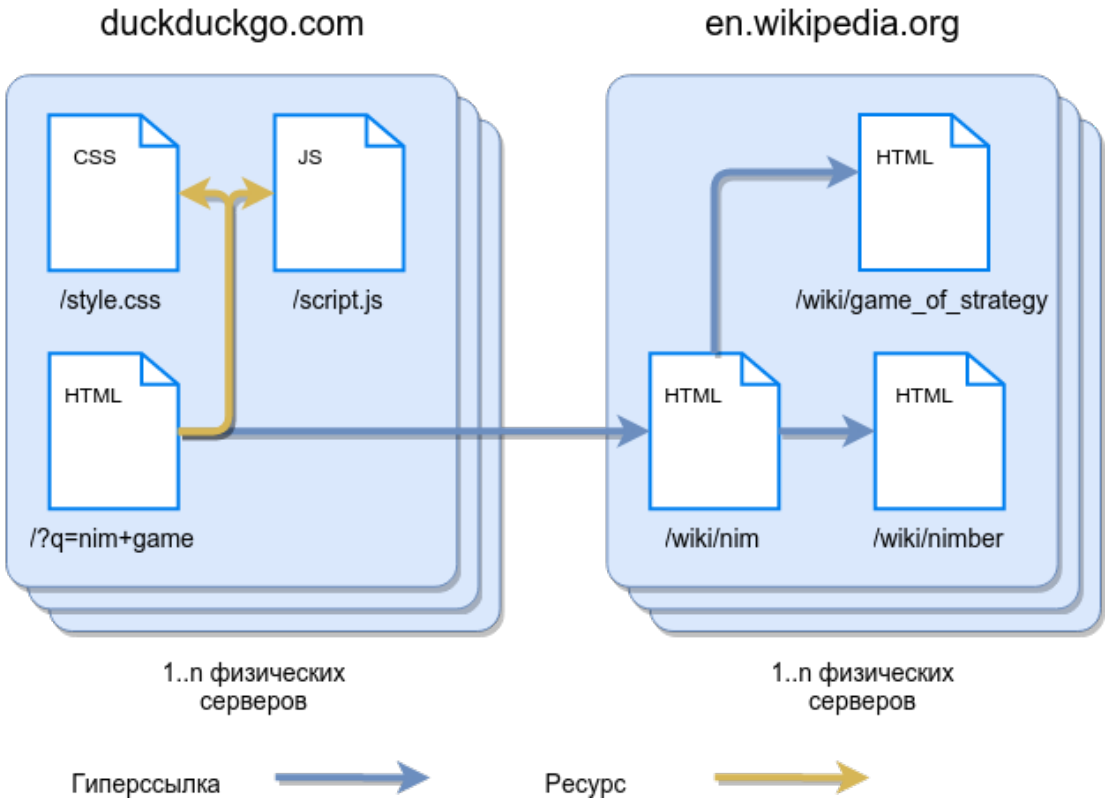


Рис. 18: Связанность документов и ресурсов посредством гиперссылок

Различия

Всё просто.

Internet – это **инфраструктура**.

WWW – это **сервис**.

Клиент-серверная архитектура

Клиент-серверная архитектура – означет, что приложение исполняется одновременно на **двух машинах**: на клиенте и на сервере.

Web-клиенты работают на компьютерах конечных пользователей. Задача Web-клиентов состоит в **формировании и отправки запроса, а далее в получении и отображении полученного ответа - документа**.

Web-сервера работают (как правило) на серверах в датацентрах. Их задача заключается в **хранении (или генерации) и отдачи документов**, то есть **отвечать на запросы**.

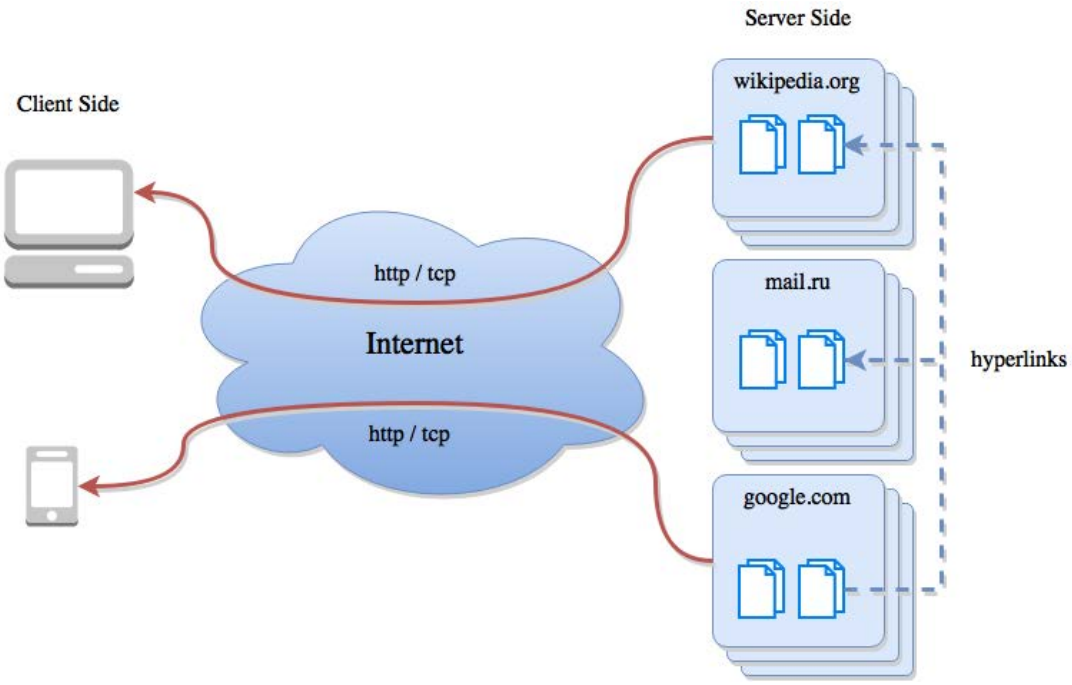


Рис. 19: Схема клиент-серверной модели

Преимущества подхода

- Открытый протокол – очень хорошо документирован и имеет много реализаций.
- Стандартный клиент – самый распространённый клиен - браузер.
- Прозрачный способ взаимодействия приложений – один ресурс может ссылаться на другой.
- Распределенная и масштабируемая система – физически, одно преложение может быть запущенно на нескольких серверах.

Типы Web-документов (MIME-типы)

MIME-типы – типы данных, которые могут быть переданы, посредством сети Интернет, в Web. Нужны для того, чтобы браузер знал как обрабатывать тот или иной документ.

Расширения файлов играют второстепенную роль.

Документы могут быть:

- ***Статические:***
  - Это файлы на дисках сервера
  - Как правило, обладают постоянным адресом
- ***Динамические:***
  - Создаются на каждый запрос
  - Содержимое зависит от параметров запроса. Например, времени и пользователя.
  - Адрес может быть постоянным или меняться

Сокращенный перечень MIME-типов

- **text**
  - **text/html** – основной тип для гипертекстовых документов. Разметка текста, пользовательского интерфейса

```
<!DOCTYPE html>
<html>
    <head>
        <link rel="stylesheet" href="/css/style.css">
        <script src="http://code.jquery.com/jquery-2.1.4.js"></script>
    </head>
    <body>
        <p>
            Some text with <img src="pic/img1.png">
            and <a href="#yes">hyperlinks</a>
        </p>
    </body>
</html>
```

  - **text/css** – стили, которые применяются к html-документам. Определяют внешний вид

```
body {
    width: 70%;
    margin: auto;
}

h1, h2, h3, h4, h5, h6 {
    font-family: sans-serif;
    margin-top: 1.5em;
}
```

  - **text/javascript** – скрипты для документов

```
<script>
    document.write("Hello, world!");
</script>
```

  - **text/xml** – для разметки самих данных. Служит для обмена данными между приложениями

```
<response status="ok">
    <friends>
        <friend id="1" name="v.pupkin"/>
        <friend id="2" name="a.pushkin"/>
        <friend id="3" name="n.tesla"/>
    </friends>
</response>
```

- **image и video**
  - image/png
  - image/jpeg
  - video/mp4
- **application**
  - **application/json** – для разметки самих данных. Служит для обмена данными между приложениями, более совершенная версия, чем xml

```
{
    "status": "ok",
    "friends": [
        { "id": 1, "name": "v.pupkin" },
        { "id": 2, "name": "a.pushkin" },
        { "id": 3, "name": "n.tesla" }
    ]
}
```

  - application/javascript
  - и многие другие...

Полный список MIME-типов

URL - Связующие звено

URL (Uniform Resource Locator) – определитель местонахождения ресурса, документа. Другими словами – **адрес**.

Структура у него следующая:

Рис. 20: Структура URL

Примечание: **Протокол** – с помощью какого протокола загружать.

Параметры – список определённых значений, которые передаются вместе с запросом.

Якорь – определённое место в самом документе (Web-страничке).

Абсолютные и относительные URL

Абсолютный – полный и точный адрес.

Примеры:

- http://server.org/1.html - абсолютный
- //server.org/1.html - schemeless, для гибкой подстройки протокола http или https

Относительный – адрес, который высчитывается браузером в зависимости от текущего адреса.

Примеры:

- /another/page.html?a=1 - путь от корня домена
- pictures/1.png - от URL текущего документа
- ?a=1&b=2 - параметры, от URL текущего документа
- #part2 - якорь, в пределах текущего документа

Правила разрешения URL

Рассмотрим на примере – **жирным** выделено, какая часть добавляется браузером при обработке относительных адресов:

https://site.com/path/page.html - основной документ

+ `http://wikipedia.org` = `http://wikipedia.org`
+ `//cdn.org/jquery.js` = **`https:`**`//cdn.org/jquery.js`
+ `/admin/index.html` = **`https://site.com`**`/admin/index.html`
+ `another.html` = **`https://site.com/path/`**`another.html`
+ `?full=1` = **`https://site.com/path/page.html`**`?full=1`
+ `#chapter2` = **`https://site.com/path/page.html`**`#chaprer2`

Рис. 21: Преобразования относительных URL

Как документы могут ссылаться друг на друга?

HTML - гиперссылки

С помощью специального тега `<a>` и его атрибута `href="URL"`.

Пример:

```
Узнать больше о <a href="https://ru.wikipedia.org/wiki/URL">URL</a>
```

Результат: Узнать больше о URL

HTML - формы

С помщью тега `<form>` и его атрибута `action="URL"` – куда будет отправлен запрос этой формы.

Пример:

```
<form action="https://duckduckgo.com/">
    <input type="text" name="q" value="">
    <input type="hidden" name="ia" value="images">
    <button type="submit">Найти</a>
</form>
```

Результат:

HTML - ресурсы

URL-адреса как ссылки на ресурсы документа.

Пример:

```
<link rel="stylesheet" href="/css/index.css">
<script src="http://code.jquery.com/jquery-2.1.4.js"></script>
<img src="pictures/network.png" width="200">
```

CSS - ресурсы

Подключение ресурсов через использование URL в CSS файлах.

Пример:

```
/* Загрузка фонового изображения*/
.slide {
    background-image: url(../pictures/network.png)
}
/* Загрузка шрифта */
@font-face {
    font-family: Terminus;
    src: url(fonts/terminus.ttf);
}
```

JavaScript - прямое указание URL

Пример:

```
var saveApiUrl = '/items/save/';
var newTitle = 'Duck tales';
    $.ajax({
    type: 'POST',
    url: saveApiUrl,
    data: { id: 10, title: newTitle }
});
```

1.2.4 Python

1.2.4.1 Курсы

Python, Processing и основы программирования для начинающих

Введение в Python и Processing

Содержание

- *Введение в Python и Processing*
  - *Что такое компьютерная программа?*
  - *А что понимает компьютер?*
  - *Python и Processing*
  - *Задания Processing*
  - *Задания стандартного Python*
  - *Рефлексия*

Что такое компьютерная программа?

Компьютерная программа - это набор **инструкций**, следуя которым компьютер выполняет поставленную задачу.

А программировать - значит составлять для компьютера пошаговые инструкций, объясняющие, что и как ему делать.

Это можно сравнить с дрессированной собакой - она умеет исполнять набор однозначных и простых команд. А чтобы она выполнила что-то более сложное, нам нужно будет последовательно давать ей эти самые команды, которые она уже знает.

Также и с компьютером. Вот только компьютеры умеют куда больше дрессированных собак и производят операции над **информацией**.

А что понимает компьютер?

Компьютер понимает специальные команды, которые заложены в том или ином **языке программирования**. Один из которых, Python, мы и будем изучать.

Примечание: На самом деле, компьютеры, как таковые, оперируют и понимают исключительно *нули*

(0) и единицы (1) - это называется двоичный код. Языки программирования - это лишь инструмент, который помогает задавать команды и данные как в более понятной для человека форме, так и с достаточно высокой точностью, чтобы потом этот набор команд, **исходный код**, можно было преобразовать в двоичный код, который и будет исполнять процессор компьютера.

Преобразованием исходного кода в двоичный занимаются специальные программы - компиляторы и интерпретаторы. На следующих занятиях, мы разберёмся что же это такое. Пока что достаточно понимать, что **Python - это интерпретируемый язык программирования**.

Например, вот так выглядит исходный код языка Python.

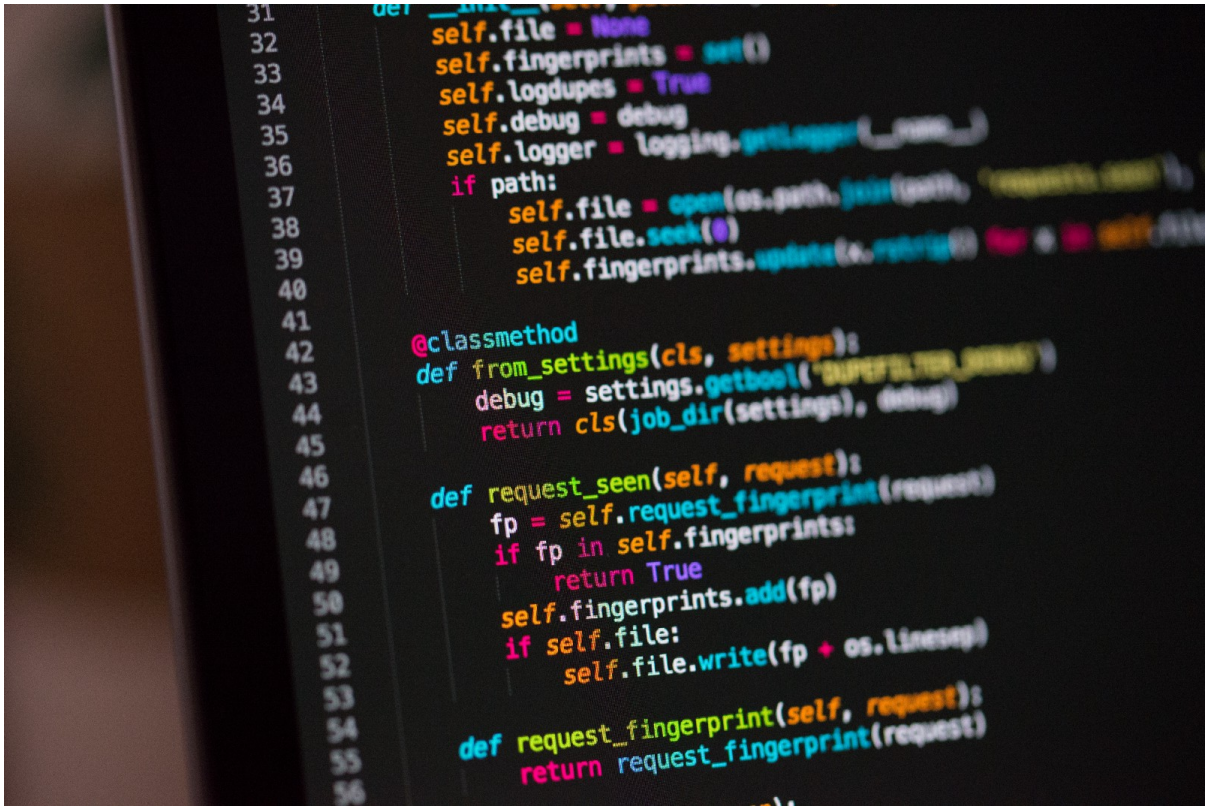


Рис. 22: Исходный код Python

Python и Processing

Python (Питон) - это текстовый язык программирования, он нужен, чтобы человек мог объяснить компьютеру, что делать. Например, он применяется в очень многих сферах: на нём можно писать полезные скрипты для взаимодействия с компьютером, программировать искуственный интеллект, делать скрипты для других программ и даже создавать игры, веб-сайты и многое другое.

Например, вот так выглядит Python-программа, которая выводит надпись `"Привет Вася"` :)

```
name = "Вася"
print("Привет", name)
```

Примечание: Что происходит в этой программе?

Рис. 23: Логотип Python

В программе ты видишь две команды:

`name = "Вася"` — положить в переменную `name` строку `"Вася"`. Программист мог записать туда любое другое имя или слово, то есть строку, и выбрал `"Вася"`.

`print("Привет", name)` — напечатать на экран сообщение `"Привет"` и ту строку, котороя хранится в переменной `name`.

Что такое «переменная» мы разберём в следующих уроках.

Эта программа использует **стандартные средства** языка Python. Мы будем их изучать. Но также, и будем использовать Python в специальной среде, которая называется **Processing** (Процессинг), чтобы можно было уже с первых занятий создавать красивые и интересные изображения, анимации и другую графику.

Processing - это среда, которая представляет собой лёгкий и быстрый инструментарий для программирования изображений, анимаций и интерфейсов. Она не изменяет сам язык программирования, но дополняет его множеством функций, которые можно использовать в этой среде.

Вообще, в Processing'е можно делать очень много красивых и удивительных вещей. Примеры некоторых из них ты можешь посмотреть в «Файл->Примеры».

Рис. 24: Пример графики созданной с помощью Processing

А сейчас давай попробуем всё это в деле и изучим базовые команды Processing'а! В следующем уроке перейдём к стандартным средствам Python и продолжим погружение в Processing.

Чтобы посмотреть базовые команды, открой следующую ссылку в новой вкладке - Нажми на меня колёсиком мыши!

Внимание: Каждую новую работу создавай в отдельном проекте и старайся сохранять свои проекты!

Задания Processing

И вот первые задания:

1. Нарисуй снеговика :)
2. Нарисуй светофор
3. Нарисуй робота :]

Минимализм

Нарисовать кодом Чебурашку и крокодила Гену, либо любого другого персонажа, в стиле «минимализм» (кубики из майкрафта, получается :D), с помощью только прямоугольников.

Например:

Задания стандартного Python

1. Используя функцию (команду) *print()*, создай программу, которая при запуске расскажет историю о тебе!

Рефлексия

А теперь, давай подумаем над тем, как прошло занятие, что ты узнал нового, какие были трудности и что можно было бы сделать лучше.

Переходи по ссылке и заполни форму - рефлексия.

Операторы, типы данных и переменные

Содержание

- *Операторы, типы данных и переменные*
  - *Переменные: место, где мы храним данные*
  - *Числа и математика*
  - *Операторы*
  - *Задания*
  - *Рефлексия*

Мы использовали язык Python для создания интересных программ, рисующих картинки за счет всего нескольких строк кода, но наши программы были ограничены в возможностях. Мы просто запускали их и наблюдали, как они сами создают картинки.

А что если бы мы захотели взаимодействовать с нашими программами на Python? На этом занятии мы узнаем, как заставить Python запросить имя пользователя, а также попробуем применение операторов при рисовании картин, на один большой шаг приблизившись к созданию динамических анимаций!

Переменные: место, где мы храним данные

В предыдущем уроке, мы использовали переменную `name` для записи в неё имени «Вася». Теперь давайте посмотрим, что такое в действительности переменные и как они работают.

Переменная

Переменная — это нечто такое, что нужно запомнить вашему ком-ьютеру на время работы программы.

Когда Python «запоминает» что-либо, он сохраняет эту информацию в памяти компьютера. Python может запоминать значения нескольких *типов данных*, например числовые значения (такие как *7*, *42* или даже дробные числа, вроде *3.1415*) и строки (буквы, символы, слова, предложения — все, что вы можете набрать на клавиатуре). А также и *пользовательские* типы данных, но это уже забегая далеко вперёд.

В Python, как и в большинстве других современных языков программирования, мы *присваиваем* значение переменной с помощью знака равенства (`=`).

Например, в примере с именем «Вася» мы сделали это вот так:

```
name = "Вася"
```

Если мы присваиваем *строку*, то нужно не забыть взять её в кавычки (""")!

Также мы можем присваивать переменным и другие *типы данных*. Например, числа:

```
x = 42
y = 15
PI = 3.1415
```

Если это число, то кавычки ("") не нужны. Если мы возмём в кавычки, например "3.1415", то это уже будет не *число*, а *строка*!

Примечание: Как Python различает типы данных?

Python - язык с *динамичекой типизацией*. Это означает, что он может сам, то есть без указания программиста, *динамически* адаптировать те или иные переменные к нужному типу данных. То есть, например, если мы сначала запишем в переменную число `var = 42`, то ничто нам не помешает потом записать в неё строку `var = "Hello"` - Python всё сделает!

Имена переменных

Каждый раз при присвоении переменной какого-либо значения сначала (слева от знака равенства) необходимо указать имя этой переменной (чтобы мы могли к ней потом обращаться), а затем (справа от знака равенства) — присваиваемое значение.

Как правило, переменные называются простыми именами, описывающими содержащееся в переменной значение (как в случае с переменной `name`, хранящей имя).

Такой подход упрощает понимание предназначения переменных и их последующее использование. Однако следует запомнить несколько правил, которым должны соответствовать имена создаваемых переменных.

1. Используевые буквы должны быть латинскими.
2. Остальные символы в имени переменной должны быть *буквами, цифрами или нижним подчеркиванием (“_“)*. Внутри имени переменной пробелы не допускаются (например, имя *my name* приведет к появлению сообщения о синтаксической ошибке, так как Python подумает, что вы перечислили две пе ременные и разделили их пробелом).
3. Имена переменных в Python чувствительны к регистру (БОЛЬШИЕ или маленькие буквы). Это значит, что если в имени переменной мы использовали только буквы нижнего регистра (например, `abc`), то воспользоваться значением, сохраненном в переменной, мы можем, только указав имя переменной в точно таком же виде, в том же регистре. Таким образом, переменная `My_Name` отличается от `my_name`, а `MY_NAME` от них обеих.

Использование переменных

Использовать переменные очень легко! Как присвоить (записать) значение переменной ты уже знаешь. А чтобы использовать её, то есть обратиться к тому значению, что в нём находится - нужно просто использовать её имя.

Например, как в программе приветствия Васи.

```
name = "Вася"
print("Привет", name)
```

`print("Привет", name)` - здесь мы ввели строку `"Привет"` вместе с тем, что хранилось в `name`.

Чтобы лучше понять этот механизм - запусти *интерпретатор Python* и попробуй создать свои переменные и взять из них значения.

Например,

```
>>> num = 42
>>> num
>>> 42
>>> hi = "hello"
>>> hi
>>> hello
```

Пользовательский ввод

Чтобы улучшить нашу программу, можно сделать так, чтобы в переменную сохранялось то значение, которое введёт пользователь.

Выглядеть это будет так:

```
name = input("Введите своё имя: ")
print("Привет", name)
```

Примечание: `name = input("Введите своё имя: ")` — создаём переменную `name` и записываем в неё результат выполнения команды (функции) `input`, а в скобочках к команде `()` передаём то, какую строку вывести в консоль при запросе.

`print("Привет", name)` — вывести в консоль «Привет» вместе с `name`, вставив между ними пробел.

Подводя итог:

- Переменные нужны, чтобы запоминать фрагменты данных, используемых в программе.
- Использовать в имени переменной мы можем только латинские буквы, цифры, знак подчёркивания (`_`) и имя должно начинаться с буквы.
- Python может *динамически* адаптировать те или иные переменные для нужного *типа данных*.

Числа и математика

Компьютеры очень хороши в математических операциях (сложение, вычитание и так далее). Ваш компьютер способен совершать более одного миллиарда (1'000'000'000, или тысячи миллионов) математических операций в секунду!

Язык Python дает нам доступ к этому мощному ресурсу математических вычислений, предоставляя для этого числа двух основных видов:

- Целые числа. Например, *1, 2, 42, 1234* и тд.
- Вещественные (дробные) числа. Например, *1.5, 3.1415, 2.67842356* и тд. Их ещё называют *числа с плавающей точкой*

Но чтобы от чисел было больше пользы, мы должны уметь ими совершать с ними *операции*. И здесь Python предоставляет нам множество возможностей!

Операторы

Оператор — это символ, или несколько символов — в отличие от имени переменной или функции (команды). Например, «+» умеет брать два числа слева и справа от себя(они называются операнды), складывать и подставлять вместо себя и операндов то, что получается.

Математические символы, такие как + (плюс) и (минус), являются операторами, так как они оперируют (или выполняют вычисления) числами в уравнении. Когда мы говорим вслух «4 + 2», чтобы ввести это на калькуляторе, мы хотим сложить числа 4 и 2, чтобы получить их сумму, 6.

В языке Python используется большинство операторов, которыми вы пользуетесь на уроках математики, в том числе +, -, скобки () и другие, которые показаны в таблице ниже.

Математический символ	Оператор Python	Операция	Пример	Результат
+	+	Сложение	`4 + 2`	`6`
−	-	Вычитание	`4 - 2`	`2`
×	*	Умножение	`4 * 2`	`8`
÷	/	Деление	`4 / 2`	`2.0`
4^2	**	Экспонента или степень	`4 ** 2`	`16`
()	()	Скобки (группировка)	`(4 + 2) * 3`	`18`

Задания

Ну, самое время попробовать это всё в деле! Далее будут задания, с помощью которых ты сможешь опробовать и закрепить полученные знания.

Processing

1. Напиши код, который объяснит Processing, как нарисовать дом с крышей, окном и дверью. Можешь добавить больше деталей: печную трубу, траву, облака — выбор не ограничен. Не забудь раскрасить! Чтобы было легче позиционировать объекты, используй переменные и операторы.
2. По желанию, можешь попробовать изобразить более сложные или перейти к другим заданиям.

Стандартный Python

Чепуха

Вместо того чтобы спрашивать имя пользователя, мы попросим ввести *прилагательное*, *существительное* и *глагол прошедшего времени мужского рода единственного числа* и сохраним введенные

данные в трех разных переменных точно так же, как мы поступали с переменной `name` в исходной программе. Потом мы выведем на печать какое-нибудь предложение, например «Этот `прилагательное существительное глагол` на ленивую рыжую собаку».

Результат работы программы должен выглядеть примерно следующим образом:

```
>>>
Введите прилагательное: умный
Введите существительное: учитель
Введите глагол прошедшего времени: чихнул
Этот умный учительно чихнул на ленивую рыжую собаку.
>>>
```

Попробуй придумать ещё что-нибудь забавное ;)

И не забудь сохранить свою программу!

Новая чепуха!

Давайте сделаем нашу игру чуть более интересной.

1. Создай новый файл для кода, сохрани под тем именем, которое тебе больше понравится.
2. Добавьте строку, которая бы запрашивала ввод названия животного с предлогом «на».
3. Затем измени команду `print()`, убрав слово `собаку` и добавив переменную `animal` после предложения, взятого в кавычки (не забудь в команде `print()` добавить запятую перед новой переменной).

При желании вы можете изменить предложение еще сильнее. К примеру, `Эта весёлая меловая доска рыгнула на ленивого коричневого геккона!`, или что-нибудь еще более забавное!

И не забудь сохранить свою программу!

Пиццерия

Давайте представим, что мы не только программисты, но и владельцы нашей собственной пиццерии.

Чтобы нам было удобно расчитывать общую стоимость заказа, с учетом налога, напишем небольшую программу, которая подсчитает это.

Предположим, что у нас заказывают одну или несколько пицц одинаковой стоимости, а налог в нашей стране составляет 8% с общей стоимости покупки. То есть за каждый рубль, уплаченный за пиццу, мы должны заплатить восемь копеек налога с продаж. На словах мы можем смоделировать данную программу следующим образом:

1. Спросить человека, сколько пицц он желает заказать.
2. Запросить стоимость каждой пиццы, указанную в меню.
3. Подсчитать стоимость каждой пиццы как подытог.
4. Подсчитать начисляемый налог с продаж по ставке 8 процентов от подытога.
5. Добавить сумму налога с продаж к подытогу и получить итоговую стоимость.
6. Показать пользователю общую сумму к оплате, в том числе налог.

Рефлексия

А теперь, давай подумаем над тем, как прошло занятие, что ты узнал нового, какие были трудности и что можно было бы сделать лучше.

Переходи по ссылке и заполни форму - рефлексия.

- Введение в Python и Processing
- Операторы и переменные
- Processing
- Рефлексия
- GitHub Start Up
- SoloLearn - приложение на смартфон для изучения программирования
- Python, Processing и основы программирования для начинающих

1.2.5 Scratch

1.2.5.1 Курсы

Креативное программирование в Scratch

Начало

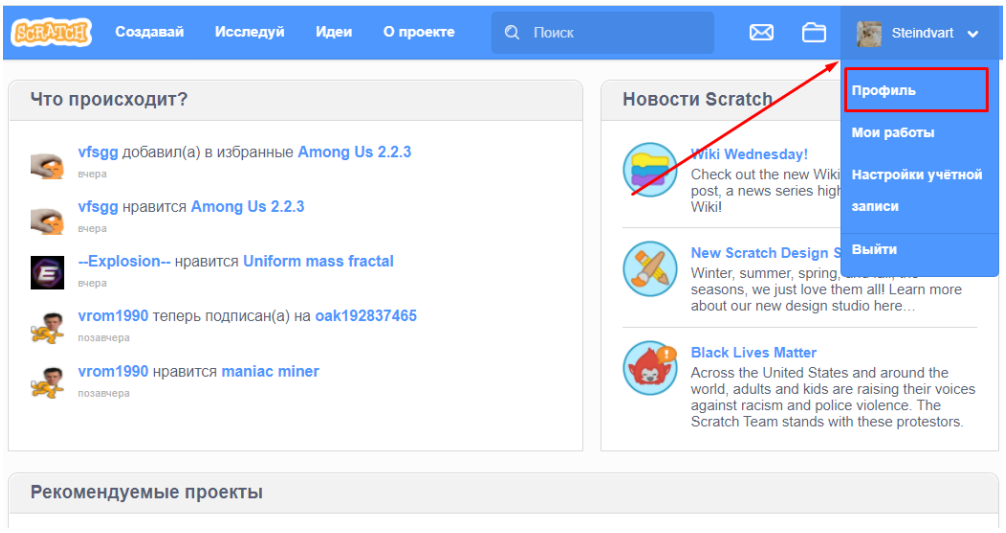
Приветствую тебя на курсе основ программирования в Scratch, дорогой друг!

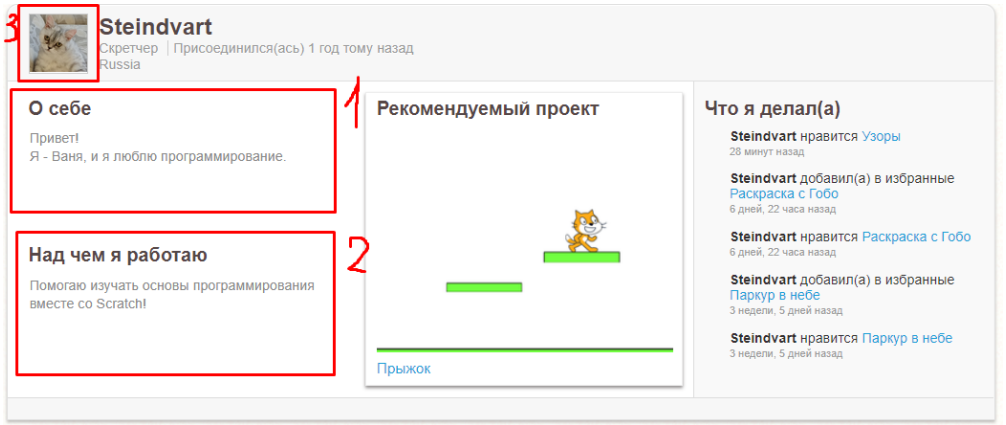
Сегодня, мы более подробно разберёмся в том, что же такое программирование и Scratch, а также настроим профиль и поделимся одним из своих первых проектов!

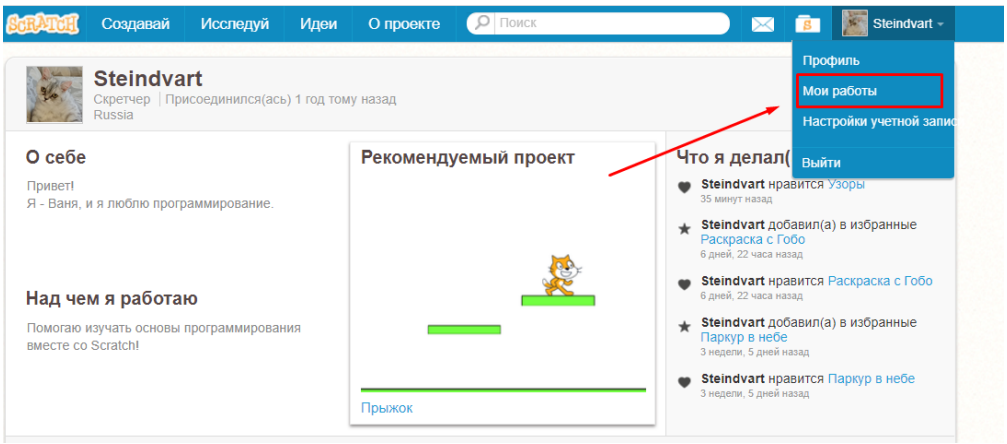
Задания

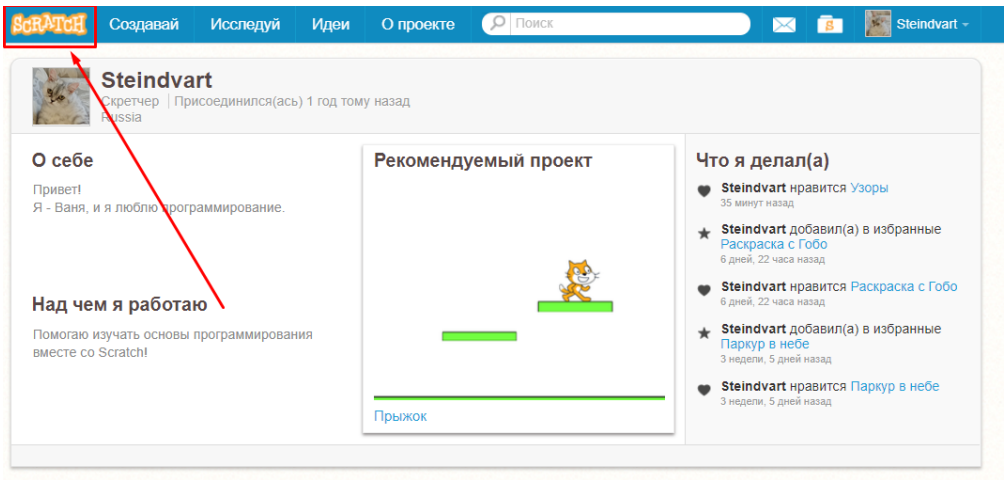
Знакомимся с сайтом Scratch и заполняем профиль

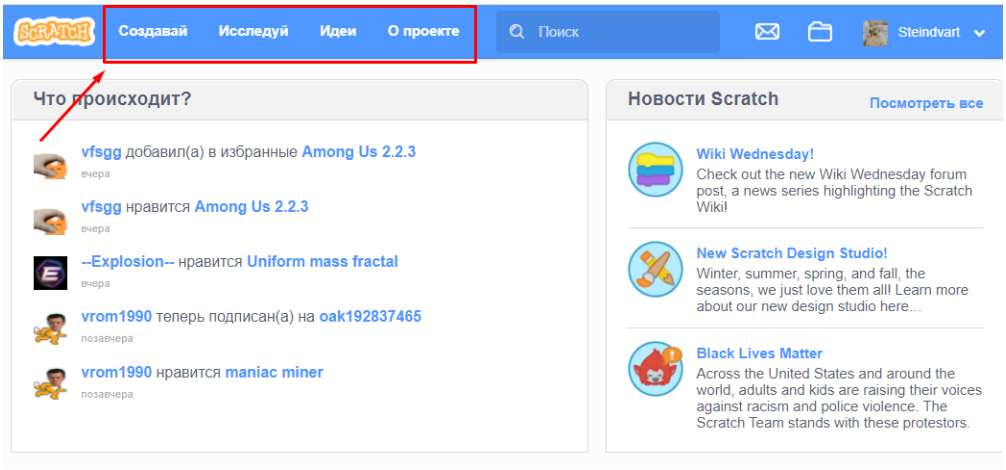
1. Зайди в свой профиль. Для этого нажми на свой никнейм (псевдоним) в правом верхнем углу сайта и в выпавшем меню выбери «Профиль».
2. Найди раздел «О себе» и кликни по надписи «расскажи о себе». Напиши туда что-нибудь о себе — расскажи будущим посетителям то, что считаешь самым важным.
3. Выбери свой аватар — картинку не больше 512 килобайт и 500 на 500 пикселей. Наведи курсор мыши на картинку котика над разделом «О себе». Нажми на появившуюся на картинке кнопку Change и выбери подготовленный аватар, чтобы загрузить на сайт.
4. Снова нажми на свой никнейм в правом верхнем углу сайта, как в пункте 1. Но выбери в выпадающем меню пункт «Мои работы», он следующий после «Профиль». Запомни, как сюда попасть — здесь будут все проекты, которые ты сделаешь.
5. Нажми на логотип Scratch в левом нижнем углу. Откроется главная страница Scratch. Какие разделы на ней есть? Запомни их, они подарят много идей.
6. Справа от логотипа сайта Scratch есть кнопка «Создавай». Если нажать на неё, создашь новый проект. Но рядом с ней ещё есть кнопки «Исследуй», «Идеи» и «О проекте». Посмотри, что откроется, если нажать на них.


Создавай
Исследуй
Идеи
О проекте
Поиск
Steindvart
Профиль
Мои работы
Настройки учётной записи
Выйти
Что происходит?
vfsgg добавил(а) в избранные Among Us 2.2.3
вчера
vfsgg нравится Among Us 2.2.3
вчера
--Explosion-- нравится Uniform mass fractal
вчера
vrom1990 теперь подписан(а) на oak192837465
позавчера
vrom1990 нравится maniac miner
позавчера
Новости Scratch
Wiki Wednesday!
Check out the new Wiki
post, a news series high
Wiki!
New Scratch Design S
Winter, summer, spring
seasons, we just love them all! Learn more
about our new design studio here...
Black Lives Matter
Across the United States and around the
world, adults and kids are raising their voices
against racism and police violence. The
Scratch Team stands with these protestors.
Рекомендуемые проекты


3
Steindvart
Скретчер | Присоединился(ась) 1 год тому назад
Russia
О себе
Привет!
Я - Ваня, и я люблю программирование.
Над чем я работаю
Помогаю изучать основы программирования
вместе со Scratch!
1
Рекомендуемый проект
2
Прыжок
Что я делал(а)
Steindvart нравится Узоры
28 минут назад
Steindvart добавил(а) в избранные
Раскраска с Гобо
6 дней, 22 часа назад
Steindvart нравится Раскраска с Гобо
6 дней, 22 часа назад
Steindvart добавил(а) в избранные
Паркур в небе
3 недели, 5 дней назад
Steindvart нравится Паркур в небе
3 недели, 5 дней назад


Создавай
Исследуй
Идеи
О проекте
Поиск
Steindvart
Профиль
Мои работы
Настройки учетной запис
Выйти
Steindvart
Скретчер | Присоединился(ась) 1 год тому назад
Russia
О себе
Привет!
Я - Ваня, и я люблю программирование.
Над чем я работаю
Помогаю изучать основы программирования
вместе со Scratch!
Рекомендуемый проект
Прыжок
Что я делал(
Steindvart нравится Узоры
35 минут назад
Steindvart добавил(а) в избранные
Раскраска с Гобо
6 дней, 22 часа назад
Steindvart нравится Раскраска с Гобо
6 дней, 22 часа назад
Steindvart добавил(а) в избранные
Паркур в небе
3 недели, 5 дней назад
Steindvart нравится Паркур в небе
3 недели, 5 дней назад

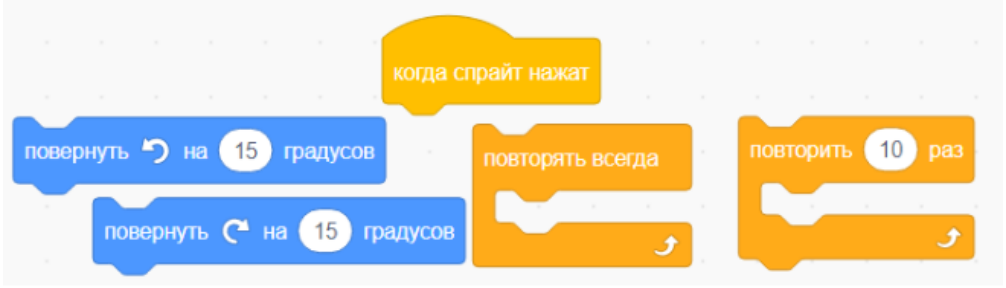




Кручусь-верчусь!

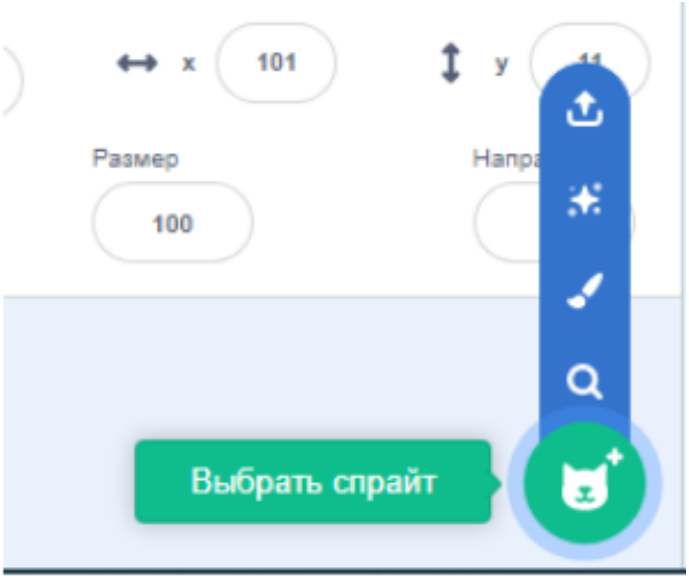
Давай сделаем проекты с весёлыми вращающимися спрайтами!

1. Создай новый проект — на главной странице сайта Scratch нажми кнопку «Создавай».
2. С помощью блоков ниже сделай, чтобы твой спрайт вращался, когда ты на него нажимаешь. Как всё заработает, попробуй потом добавить и другие блоки!



3. Теперь наведи курсор на кнопку «Выбрать спрайт» в правом нижнем углу. Она выглядит как синий кружок с котиком. Тебе выпадет несколько вариантов — выбери один

из них, чтобы выбрать новый спрайт из библиотеки Scratch, загрузить с компьютера, нарисовать свой или выбрать случайный.



4. Добавь ещё 2-3 спрайта, которые тоже будут вращаться, когда на них нажимают. С одинаковой ли они скоростью вращаются? Попробуй сделать, чтобы с разной.
5. Добавь в проект подходящий фон.
6. Добавь в проект спрайт, который при нажатии будет вращаться сначала в одну сторону, а потом в другую.

Ещё задания!

1. Сделай проект про спиннер (вид сверху)— нарисуй его или вставь картинку из интернета. Вместо спиннера может быть юла, карусель и т.д.
2. Наведи на кнопку «добавить спрайт», выбери «нарисовать». Нарисуй спрайт. Нарисуй второй спрайт, но с «дырой» в середине — как раз, чтобы первый мог в ней поместиться. Например, нарисуй круг или квадрат. Сделай так, чтобы при нажатии эти спрайты вращались, но в разные стороны. Как тебе?
3. Создай спрайт. В верхней, синей полосе найди кнопку «Руководства». Выбери интересную тему и попробуй сделать проект по видео.
4. Зайди на сайт Скретч. Поищи интересные проекты. Когда будешь находить интересный, нажимай «сердечко» под проектом и «Звёздочку». Когда сделаешь это три раза, зайди в свой профиль. Что изменилось?
5. Придумай тему проекта сам и попробуй сделать. Не получилось? Значит, ты придумал сложный и интересный проект, и именно для того, чтобы получалось, ты пришёл к нам.
6. Перейди по ссылке на страницу с проектами для начинающих: я ссылка - жми на меня! Нажимай на картинки, чтобы открыть проект. Выбери интересный проект, нажми на зелёную кнопку «ремикс». Всё! Теперь это твоя копия этого интересного проекта. Переделай её как хочешь
7. Найди любой другой интересный проект на сайте Scratch. Сделай его ремикс и переделай, как тебе хочется.

10 блоков

Сегодня, мы продолжим погружение в программирование вместе с Scratch и попробуем сделать интересный и весёлый проект с помощью 10 блоков!

Но для начала, давай поиграем в игру, чтобы разогреть свои прекрасные и умные мозги. Ведь они нам очень понадобятся!

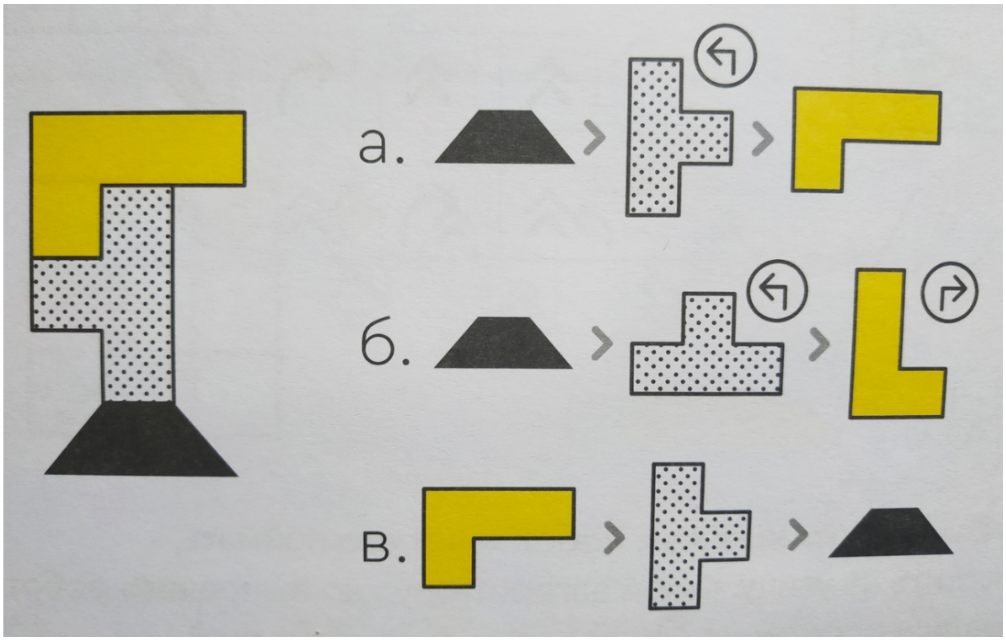
Игра «Действуй по алгоритму»

Робот должен построить башню по проекту. Есть команды для управления роботом:

- поворот направо на 90 градусов
- поворот налево на 90 градусов
- поворот на 180 градусов.

Команды над кубиками показывают, как их нужно повернуть перед тем, как построить башню.

Выбери алгоритм, с помощью которого получится построить башню как на рисунке слева. Какой из алгоритмов составлен верно? Ответы можешь записать в тетрадке или просто на листочке, чтобы потом разобрать их вместе с учителем.

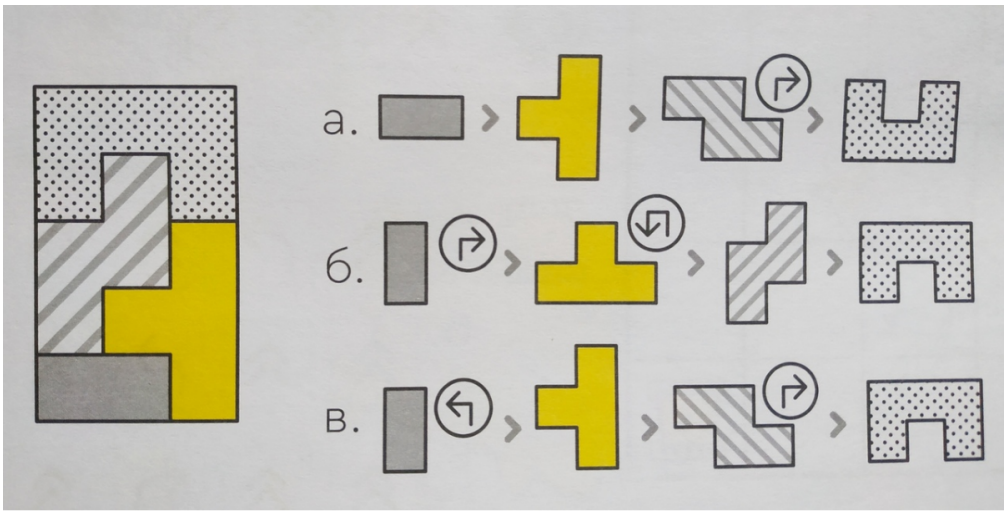


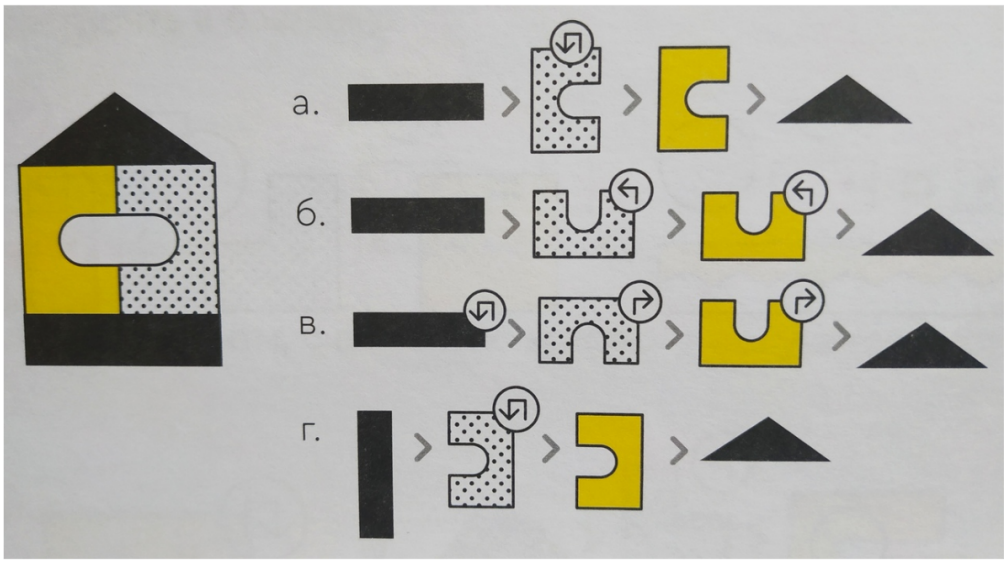
Ну как? Мозги заряжены и готовы к работе! :)

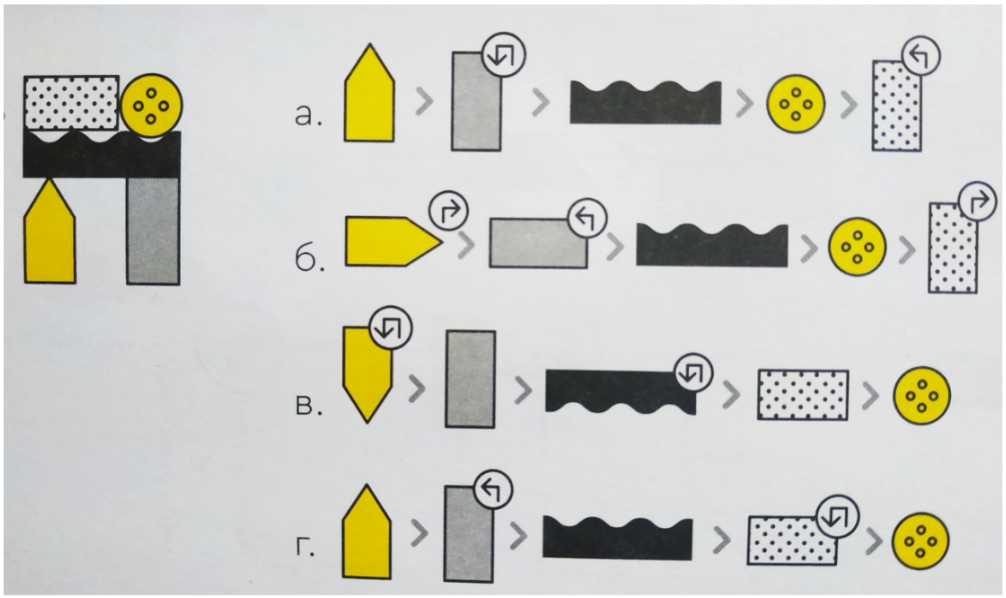
Задание «10 блоков»

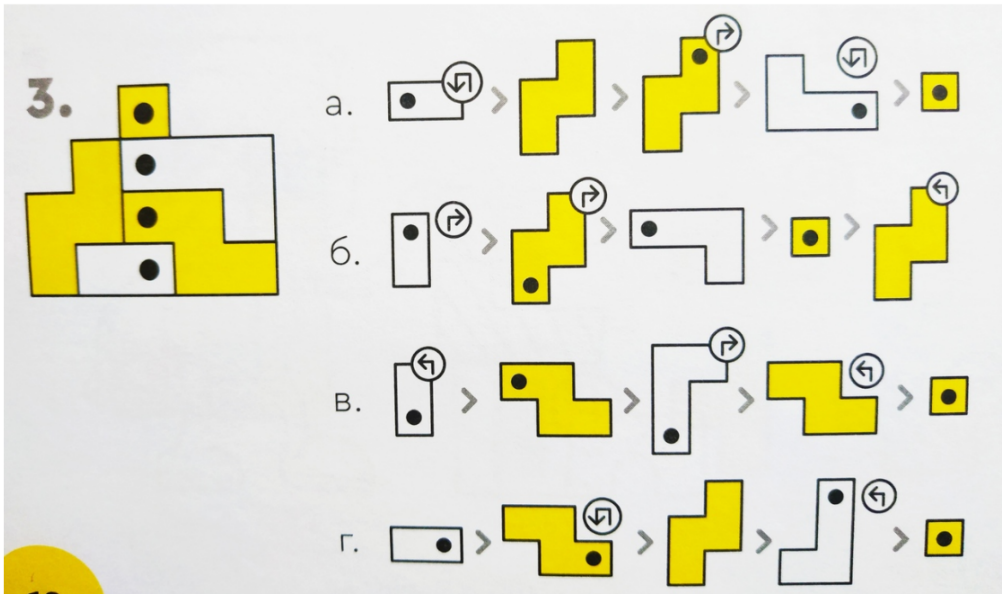
Попробуй сделать проект используя только эти 10 блоков Скретч.

Применяйте их один, два или более раз, но используйте каждый блок хотя бы раз.


а.
б.
в.


а.
б.
в.
г.


а.
б.
в.
г.


3.
а.
б.
в.
г.

когда спрайт нажат
повторить 10 раз
ждать 1 секунд
плыть 1 секунд в точку x: 0 y: 0
перейти в x: 0 y: 0
говорить Привет! 2 секунд
показаться
спрятаться
установить размер 100 %
играть звук Мяу до конца

Идеи для воплощения

Прятки

- Создай спрайт с человеком, который ищет кота. Этот спрайт плавно перемещается в центр экрана и зовет кота (видим надпись на экране “Ты где?) слышится звук “Мяу” (3 раза).
- Создай спрайт Кот в центре экрана, который уменьшается когда на него нажмут и исчезает в кустах.
- Используй все 10 новых блоков.

День рождения Котика

- Создай спрайт Котика, который медленно перемещается в центр экрана и входит в комнату с шарами (фон шары), звучит музыка, кот от восхищения говорит “Мяу” (видим надпись на экране).
- Создай спрайт Шарик, при нажатии на который он взлетает наверх и уменьшается в размерах, затем исчезает.
- Используй все 10 новых блоков.

Исследование космоса

- Создай спрайт Космонавт, который проводит исследования в космосе (подбери космический фон) и плавно перемещается по станции. Вдруг появляется надпись “Замените баллон”.
- Создай спрайт Инопланетянин, при нажатии на который он начинает увеличиваться, приближается и приветствует Космонавта на инопланетном языке.
- Используй все 10 новых блоков.

Творческое задание

- Придумай два спрайта и историю, в которой будут использованы все 10 новых блоков не менее 1 раза. Поделись своей идеей с соседом и спроси его идею.
- Используй все 10 новых блоков.

Если захочешь зарядиться новыми идеями и посмотреть, как с этим заданием справлялись другие (сможешь сделать лучше и интереснее?) переходи по ссылке - *тыкай колёсиком мыши! <https://scratch.mit.edu/studios/475480/>*

Попробуй сделать ремикс какого-нибудь проекта!

Рефлексия

А теперь, давай подумаем над тем, как прошло занятие, что ты узнал нового, какие были трудности и что можно было бы сделать лучше.

Переходи по ссылке и заполни форму - рефлексия.

- Знакомство

- Создаём первые проекты, заполняем профиль
- 10 блоков
- SoloLearn - приложение на смартфон для изучения программирования