

Модуль 1.1 Введение в Data Science

План:

1. Unix-подобные ОС. Экскурс по командной строке.
Основные команды.
2. Понятие репозитория. Version Control System (VCS). Git.
3. Package Management. Pip. Conda. Virtual Environments.
4. Экскурс в язык Python

1. Unix-подобные ОС. Экскурс по командной строке. Основные команды.

Unix-подобные ОС.

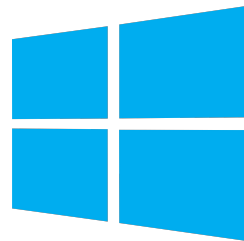
Большинство операционных систем можно разделить на 2 группы (семьи):

1. **Microsoft Windows NT** и их потомки (e.g. Windows, Xbox OS, Windows Server)
2. **Unix / Unix-подобные ОС** (MacOS X, Linux, Android, Chrome OS, PS5 OS и т.д).

Древо: [Operating Systems: Timeline and Family Tree](#)



macOS



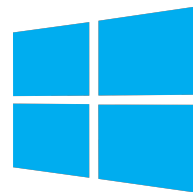
Терминал.



- **Gnome Terminal**
- Tilda
- Terminator
- .
- .
- .
- И многие другие

macOS

- **MacOs Terminal**



- **PowerShell**
- **CMD**
- **WSL2**

date

which

nano

less

grep

tr

cal

ls

rm

tac

xargs

source

ncal

cd

rmdir

rev

chmod

export

echo

pwd

mv

head

sudo

crontab

man

touch

cp

tail

su

tee

help

mkdir

cat

tree

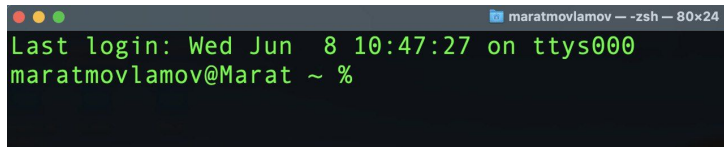
chown

sort

Диалоговое окно консоли (Prompt).

Когда вы откроете ваш терминал, вы увидите диалоговое окно, которое будет выглядеть:

\$username@yourmachine

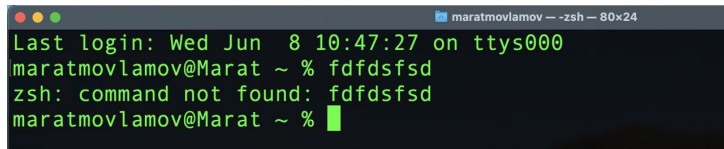
A terminal window titled 'maratmovlamov -- zsh -- 80x24'. The output shows 'Last login: Wed Jun 8 10:47:27 on ttys000' followed by the prompt 'maratmovlamov@Marat ~ %'.

Диалоговое окно (Prompt) это то, что вы будете видеть каждый раз запуская терминал.

Все что необходимо - описать команду и нажать enter.

В случае ввода несуществующей команды, терминал выдаст ошибку:

"command not found".

A terminal window titled 'maratmovlamov -- zsh -- 80x24'. The output shows 'Last login: Wed Jun 8 10:47:27 on ttys000', followed by the prompt 'maratmovlamov@Marat ~ %'. The user enters 'fdfdsfsd', and the terminal responds with 'zsh: command not found: fdfdsfsd' followed by the prompt 'maratmovlamov@Marat ~ %' and a cursor.

Начало работы с терминалом. Первые команды.

Команда date

Не самая полезная команда, выбрана для того чтобы проиллюстрировать концепции построения команд и понять работу терминала.

Выполните данную команду и нажмите enter. Вы должны будете увидеть текущую дату в формате **timestamp** в вашей консоли.

A screenshot of a terminal window with a dark blue background. At the top left, there are three colored window control buttons: red, yellow, and green. The prompt '\$date' is shown in blue text. Below it, the output is displayed in a monospaced font: 'Thursday 09 June 2022 11:11:11 AM PDT', where the day of the week and time are in white, and the date is in orange.

```
$date
Thursday 09 June 2022 11:11:11 AM PDT
```


Общая структура команд

В целом, общая структура команд в терминале имеет следующий вид:

```
$ command [-flag(s)] [-option(s) [value]] [argument(s)]
```

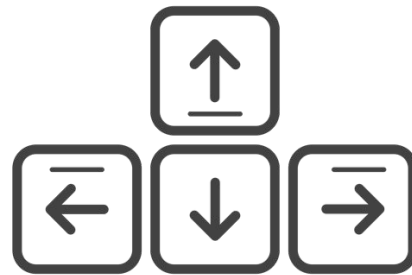
Options (параметры) - параметры встроенные в сценарий команды, которые изменяют поведение команды. Некоторые команды можно использовать без каких-либо параметров или аргументов (далее мы увидим команды - ls, pwd), но некоторые команды обычно требуют аргумент (e.g. less, more).

Начало работы с терминалом. Первые команды.

Кнопки навигации

Используя клавиши стрелок на клавиатуре легко передвигаться по предыдущим командам в “истории” консоли.

Попробуйте вернуться на предыдущую команду и исполните ее еще раз.



Начало работы с терминалом. Структура команды.



```
$command -options arguments
```

Большинство команд поддерживают параметры (options), которые изменяют либо модифицируют их изменение.

Также многие команды принимают аргументы (например в качестве аргумента может выступать файл, адрес сервера и так далее).

Начало работы с терминалом. Аргументы.

Аргументы. Команда echo

Понятия "аргумента" - "флага" очень часто используются взаимозаменяемо. Вы можете от разных людей услышать их "собственное название".

Команда echo довольно простая. В качестве аргумента она **принимает ввод** пользователя и **выводит его в терминале** в качестве строкового значения.



```
$echo Data Science Foundations  
Data Science Foundations
```

Начало работы с терминалом. Аргументы.

Аргументы. Команда ncal

Команда ncal принимает аргументы, в которых можно указать месяц (месяцы) и год.

Если указать год, то команда выведет все календарные месяцы в указанном году.

Если указать месяц и год, то команда выведет только тот месяц года, который вы указали.

```
$ncal 2022
```

	January	February	March	April
Su	2 9 16 23 30	6 13 20 27	6 13 20 27	3 10 17 24
Mo	3 10 17 24 31	7 14 21 28	7 14 21 28	4 11 18 25
Tu	4 11 18 25	1 8 15 22	1 8 15 22 29	5 12 19 26
We	5 12 19 26	2 9 16 23	2 9 16 23 30	6 13 20 27
Th	6 13 20 27	3 10 17 24	3 10 17 24 31	7 14 21 28
Fr	7 14 21 28	4 11 18 25	4 11 18 25	1 8 15 22 29
Sa	1 8 15 22 29	5 12 19 26	5 12 19 26	2 9 16 23 30

	May	June	July	August
Su	1 8 15 22 29	5 12 19 26	3 10 17 24 31	7 14 21 28
Mo	2 9 16 23 30	6 13 20 27	4 11 18 25	1 8 15 22 29
Tu	3 10 17 24 31	7 14 21 28	5 12 19 26	2 9 16 23 30
We	4 11 18 25	1 8 15 22 29	6 13 20 27	3 10 17 24 31
Th	5 12 19 26	2 9 16 23 30	7 14 21 28	4 11 18 25
Fr	6 13 20 27	3 10 17 24	1 8 15 22 29	5 12 19 26
Sa	7 14 21 28	4 11 18 25	2 9 16 23 30	6 13 20 27

	October	November	December
	2 9 16 23 30	6 13 20 27	4 11 18 25
	3 10 17 24 31	7 14 21 28	5 12 19 26
	4 11 18 25	1 8 15 22 29	6 13 20 27
	5 12 19 26	2 9 16 23 30	7 14 21 28
	6 13 20 27	3 10 17 24	1 8 15 22 29
	7 14 21 28	4 11 18 25	2 9 16 23 30
	1 8 15 22 29	5 12 19 26	3 10 17 24 31

```
$ncal june 2022
```

June 2022

Su	5 12 19 26
Mo	6 13 20 27
Tu	7 14 21 28
We	1 8 15 22 29
Th	2 9 16 23 30
Fr	3 10 17 24
Sa	4 11 18 25

Начало работы с терминалом. Аргументы.

Аргументы. Команда sort. Параметры.

Команда `sort` принимает в качестве аргумента путь к файлу и выводит отсортированное содержание данного файла.

Для сортировки файла в котором данные представлены текстом, команда отсортирует их в алфавитном порядке.

В случае с числами, сортировка будет происходить по возрастанию.

A dark-themed terminal window with three colored window control buttons (red, yellow, green) in the top-left corner. The command `$sort some_file.txt` is entered in a light blue monospaced font.

Начало работы с терминалом. Параметры.

Аргументы. Команда `sort`. Параметры.

Вам доступно 2 файла `colors.txt` и `numbers.txt`

Отсортируйте 2 файла и покажите в консоли.

Начало работы с терминалом. Параметры.

Аргументы. Команда sort. Параметры.

Вам доступно 2 файла colors.txt и numbers.txt

Отсортируйте 2 файла и покажите в консоли.



The image shows two overlapping terminal window screenshots. The top window shows the command `$sort colors.txt` being executed, resulting in the output: `blue`, `cyan`, `yellow`, `n`, and `leon`. The bottom window shows the command `$sort -g numbers.txt` being executed, resulting in the output: `1`, `10`, `14`, `15`, `45`, `50`, and `51`. Both windows have a dark blue background and three colored window control buttons (red, yellow, green) in the top left corner.

```
$sort colors.txt
blue
cyan
yellow
n
leon
```

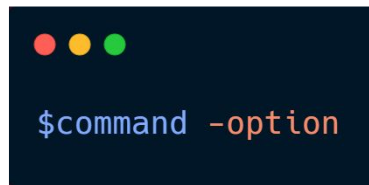
```
$sort -g numbers.txt
1
10
14
15
45
50
51
```


Начало работы с терминалом. Параметры.

Параметры.

Почти каждая команда поддерживает набор собственных параметров. Параметры служат для того, чтобы модифицировать исполнение команды для нужд пользователя.

Обычно параметры определяются путем дефиса (-option) либо двойного дефиса (--O).



```
$command -option
```



```
$ncal -j
```



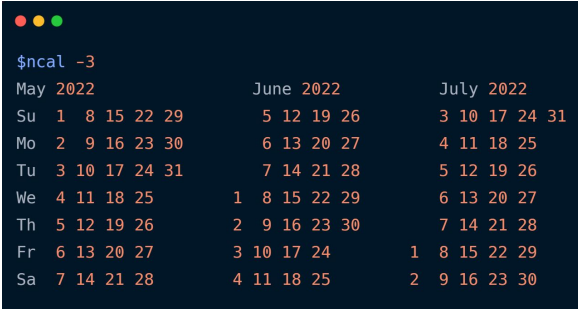
```
$sort -r colors.txt
```

Начало работы с терминалом. Параметры.

Параметры. Команда ncal

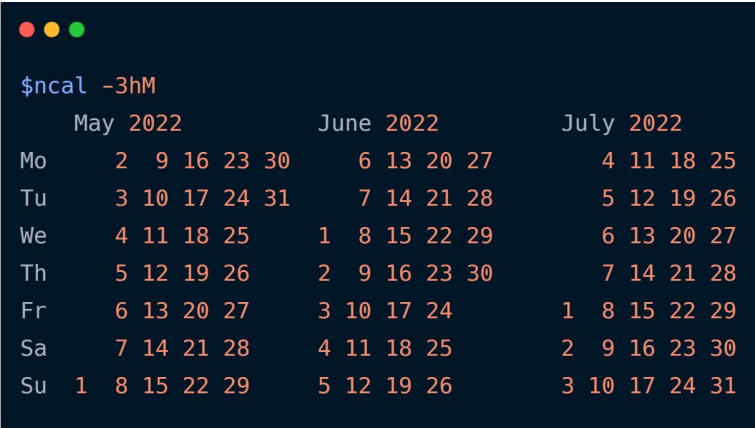
Используя параметр -3 в команде ncal пользователь тем самым указывает, что необходимо вывести предыдущий, текущий и следующий месяц.

Также можно комбинировать параметры "одной строкой".



```
$ncal -3
```

May 2022	June 2022	July 2022
Su 1 8 15 22 29	5 12 19 26	3 10 17 24 31
Mo 2 9 16 23 30	6 13 20 27	4 11 18 25
Tu 3 10 17 24 31	7 14 21 28	5 12 19 26
We 4 11 18 25	1 8 15 22 29	6 13 20 27
Th 5 12 19 26	2 9 16 23 30	7 14 21 28
Fr 6 13 20 27	3 10 17 24	1 8 15 22 29
Sa 7 14 21 28	4 11 18 25	2 9 16 23 30



```
$ncal -3hM
```

May 2022	June 2022	July 2022
Mo 2 9 16 23 30	6 13 20 27	4 11 18 25
Tu 3 10 17 24 31	7 14 21 28	5 12 19 26
We 4 11 18 25	1 8 15 22 29	6 13 20 27
Th 5 12 19 26	2 9 16 23 30	7 14 21 28
Fr 6 13 20 27	3 10 17 24	1 8 15 22 29
Sa 7 14 21 28	4 11 18 25	2 9 16 23 30
Su 1 8 15 22 29	5 12 19 26	3 10 17 24 31

Начало работы с терминалом. Параметры.

Параметры с дополнительными значениями.

Некоторые параметры могут работать с дополнительными значениями. Например, в команде `ncal` параметр `-A int` указывает какое количество месяцев показать после (`-B int` - количество месяцев до).

```
$ ncal -A1 -B1 -h
May 2022      June 2022      July 2022
Su  1  8 15 22 29      5 12 19 26      3 10 17 24 31
Mo  2  9 16 23 30      6 13 20 27      4 11 18 25
Tu  3 10 17 24 31      7 14 21 28      5 12 19 26
We  4 11 18 25      1  8 15 22 29      6 13 20 27
Th  5 12 19 26      2  9 16 23 30      7 14 21 28
Fr  6 13 20 27      3 10 17 24      1  8 15 22 29
Sa  7 14 21 28      4 11 18 25      2  9 16 23 30
```

Начало работы с терминалом. Задание.

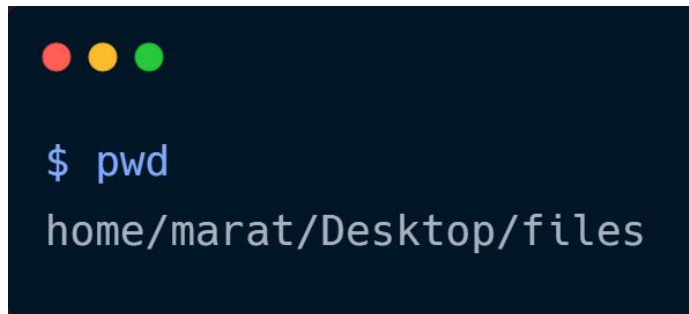
**Вывести в консоль месяц и год вашего рождения с
месяцем до месяца вашего рождения и месяцем после.**

Терминал. Команды навигации.

Терминал. Команды навигации.

Команда pwd

print working directory (pwd) показывает “местонахождение” пользователя, то есть выводит полный путь текущей рабочей директории начиная с корневого каталога /

A terminal window with a dark blue background and three colored window control buttons (red, yellow, green) in the top left corner. The prompt '\$' is followed by the command 'pwd' in blue. The output 'home/marat/Desktop/files' is shown in white below the command.

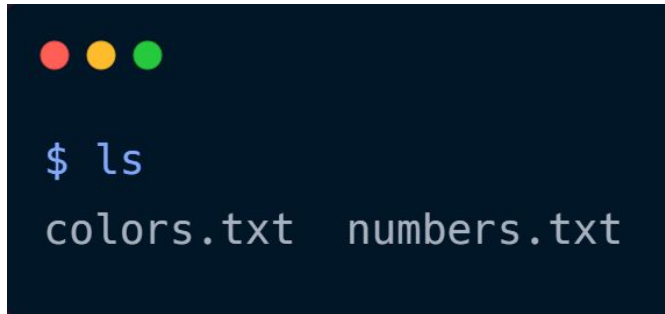
```
$ pwd  
home/marat/Desktop/files
```

Терминал. Команды навигации.

Команда ls

ls показывает пользователю что находится внутри его директории. Без аргументов и параметров показывает все файлы и папки, расположенные внутри директории.

Так же можно посмотреть что находится внутри другой директории указав ее путь.



```
$ ls  
colors.txt  numbers.txt
```

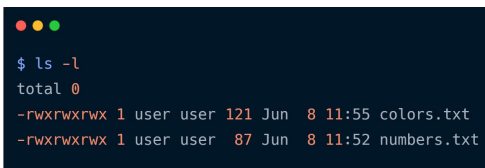


```
$ ls /bin  
bash          false         nc.openbsd   ss  
btrfs         fgconsole    netcat       static-sh  
btrfs-debug-tree fgrep        netstat      stty  
btrfs-find-root findmnt      networkctl   su  
btrfs-image   fsck.btrfs   nisdomainname sync
```

Терминал. Команды навигации.

Параметры ls

Два важных параметра -l и -a



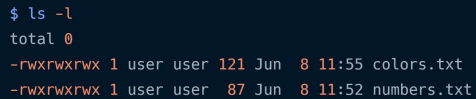
```
$ ls -l
total 0
-rwxrwxrwx 1 user user 121 Jun  8 11:55 colors.txt
-rwxrwxrwx 1 user user  87 Jun  8 11:52 numbers.txt
```

-l выводит все файлы в формате длинного списка. Показывает больше информации о каждом файле/папке

Терминал. Команды навигации.

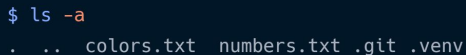
Параметры ls

Два важных параметра -l и -a



```
$ ls -l
total 0
-rwxrwxrwx 1 user user 121 Jun  8 11:55 colors.txt
-rwxrwxrwx 1 user user  87 Jun  8 11:52 numbers.txt
```

-l выводит все файлы в формате длинного списка. Показывает больше информации о каждом файле/папке



```
$ ls -a
.  ..  colors.txt  numbers.txt  .git  .venv
```

-a выводит все файлы а также скрытые файлы которые начинаются с .

Терминал. Команды навигации.

Параметры ls

Два важных параметра -l и -a

```
$ ls -l
total 0
-rwxrwxrwx 1 user user 121 Jun  8 11:55 colors.txt
-rwxrwxrwx 1 user user  87 Jun  8 11:52 numbers.txt
```

-l выводит все файлы в формате длинного списка. Показывает больше информации о каждом файле/папке

```
$ ls -a
.  ..  colors.txt  numbers.txt  .git  .env
```

-a выводит все файлы а также скрытые файлы которые начинаются с .

```
$ ls -la
total 0
drwxrwxrwx 1 user user 4096 Jun  8 11:53 .
drwxrwxrwx 1 user user 4096 Jun  8 14:00 ..
-rwxrwxrwx 1 user user 287 Jun  8 12:52 .env
-rwxrwxrwx 1 user user 287 Jun  8 12:52 .git
-rwxrwxrwx 1 user user 121 Jun  8 11:55 colors.txt
-rwxrwxrwx 1 user user  87 Jun  8 11:52 numbers.txt
```

-la комбинация обоих параметров

Терминал. Команды навигации.

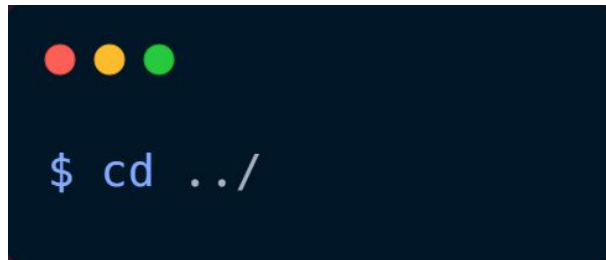
Команда **cd**

change directory

Команда для смены рабочей директории.

Сокращения `cd ..`

`cd ../../`

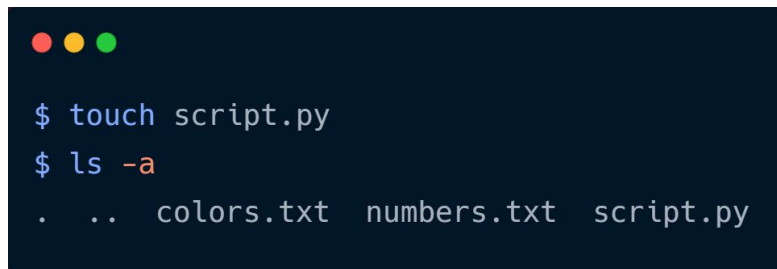


Терминал. Команды взаимодействия с файлами и папками.

Команда touch

Используется для создания нового файла из командной строки. Необходимо указать имя файла после touch. Например touch script.py создаст в вашей рабочей директории файл скрипта для Python.

Если вы попытаетесь создать уже существующий файл то команда просто обновит дату доступа и дату модификации файла до текущего момента времени.

A terminal window with a dark blue background and three colored window control buttons (red, yellow, green) in the top left corner. It shows two commands being executed: '\$ touch script.py' and '\$ ls -a'. The output of the second command is displayed on the next line: '. .. colors.txt numbers.txt script.py'.

```
$ touch script.py
$ ls -a
.  ..  colors.txt  numbers.txt  script.py
```

Терминал. Команды взаимодействия с файлами и папками.

Команда mkdir

Используется для создания новой директории (make directory).
Можно задать созданий одной или несколько директорий за один раз.



```
$ mkdir tests imgs output input data
```

Терминал. Команды взаимодействия с файлами и папками.

Команда rm

Используется для удаления файлов (навсегда).

rm -rf удаление директории (recursively and forcibly)

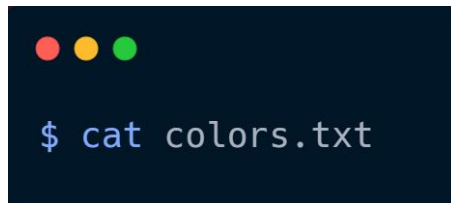


```
$ rm -rf ./tests/test_one/
```

Терминал. Работа с файлами.

Команда cat

Выводит все содержимое файла в консоль. Например `cat instructions.txt` выведет содержимое файла в терминал.

A dark-themed terminal window with three colored window control buttons (red, yellow, green) in the top-left corner. The prompt character is a dollar sign (\$), followed by the command `cat colors.txt` in a light blue monospaced font.

```
$ cat colors.txt
```

Команда head

Выводит первые 10 строк содержимого файла в консоль. Например `cat instructions.txt` выведет содержимое файла в терминал до 10 строки. Аналог `tail` (последние 10 строк). Параметр `-n int` используется для контроля вывода (`-n 5` вывести только первые/последние 5 строк). Параметр `-c` используется для вывода количества байт.

A dark-themed terminal window with three colored window control buttons (red, yellow, green) in the top-left corner. The prompt character is a dollar sign (\$), followed by two commands, `head colors.txt` and `tail colors.txt`, each on a new line in a light blue monospaced font.

```
$ head colors.txt  
$ tail colors.txt
```

Терминал. Pipelines.

Pipes



```
$ python3 script.py > logs.txt | head logs.txt | less
```



```
$ cat colors.txt words.txt | tee combined.txt | wc
```


Терминал. Environment Variables.


Shell & Environment Variables

Командная оболочка поддерживает и хранит определенный набор информации для нашего сеанса. Весь этот набор информации известен как среда (environment). Это просто набор ключ-значение, которые могут определять такие свойства как:


- Нашу рабочую директорию
- Нашу домашнюю директорию
- Название нашей командной оболочки
- Названия и логи пользователя
- Пути для приложений и прочее

Чтобы посмотреть настройки окружения введите команду **printenv**

Для внесения изменений в переменные окружения используют **команду export**



```
$ printenv | less
```



```
$ export PYTHONPATH=/usr/local/bin/python3.7  
$ echo $PYTHONPATH  
/usr/bin/python3.7
```

Терминал. Проверка:

1. Назовите основные команды навигации.

Терминал. Проверка:

1. Назовите базовые команды навигации.
2. Назовите базовые команды взаимодействия с файлами и папками.

Терминал. Проверка:

1. Назовите базовые команды навигации.
2. Назовите базовые команды взаимодействия с файлами и папками.
3. Как правильно организовывать пайплайны команд?

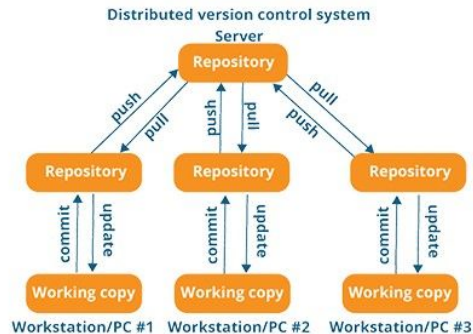
2. Понятие репозитория. Version Control System (VCS). Git.

Version Control System (VCS)

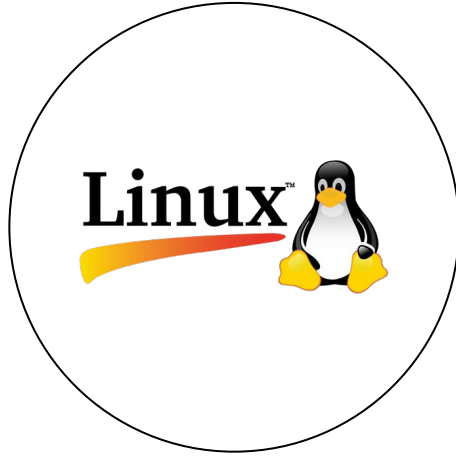
Система контроля версий (VCS) - система, записывающая изменения в файл или набор файлов в течение времени и позволяющая вернуться к определенной версии изменения.

Существуют **3 типа** систем контроля версий:

1. Локальные
2. Централизованные
3. Распределенные (Git)



Version Control System (VCS). История

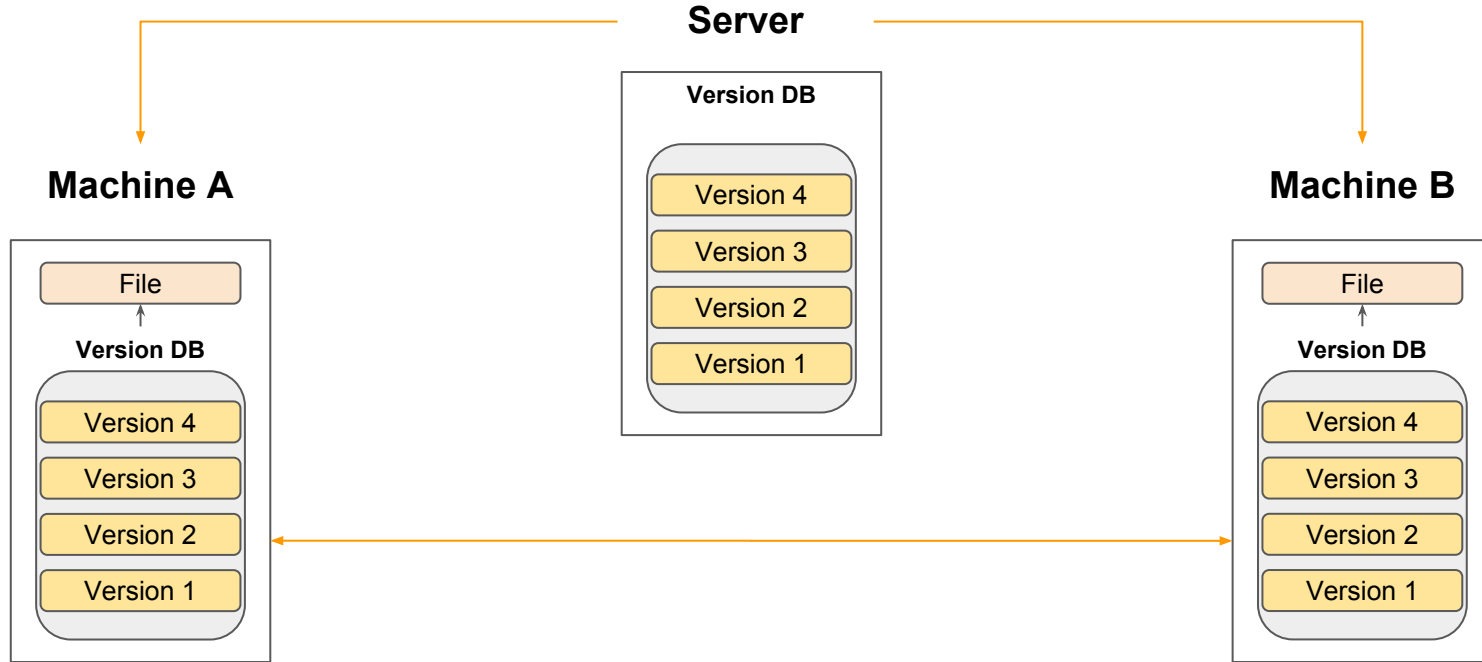


Задача - Разработка
системы контроля и
управления ядром Linux

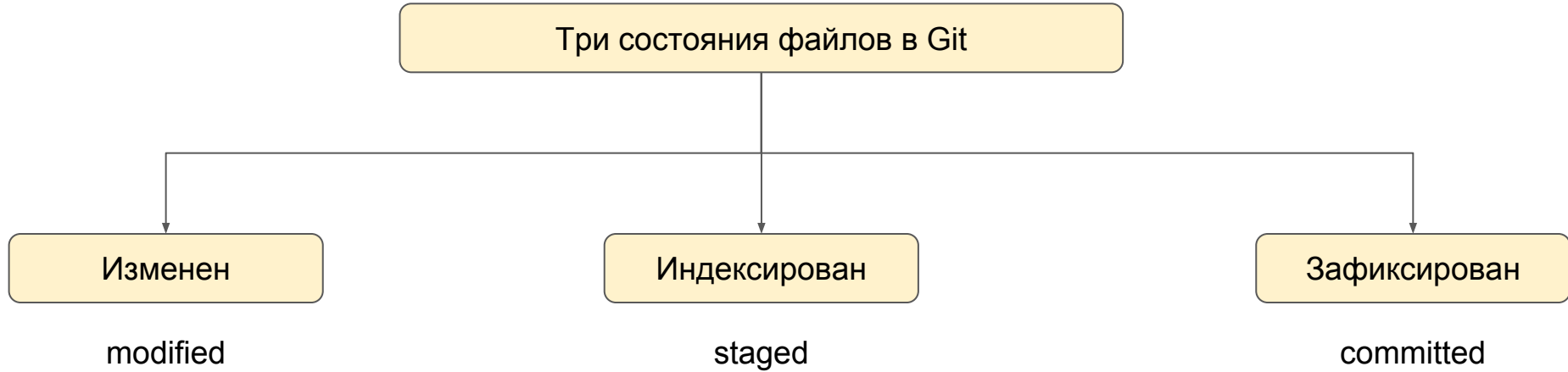


Распределенная
система контроля
версий Git, 2005 год

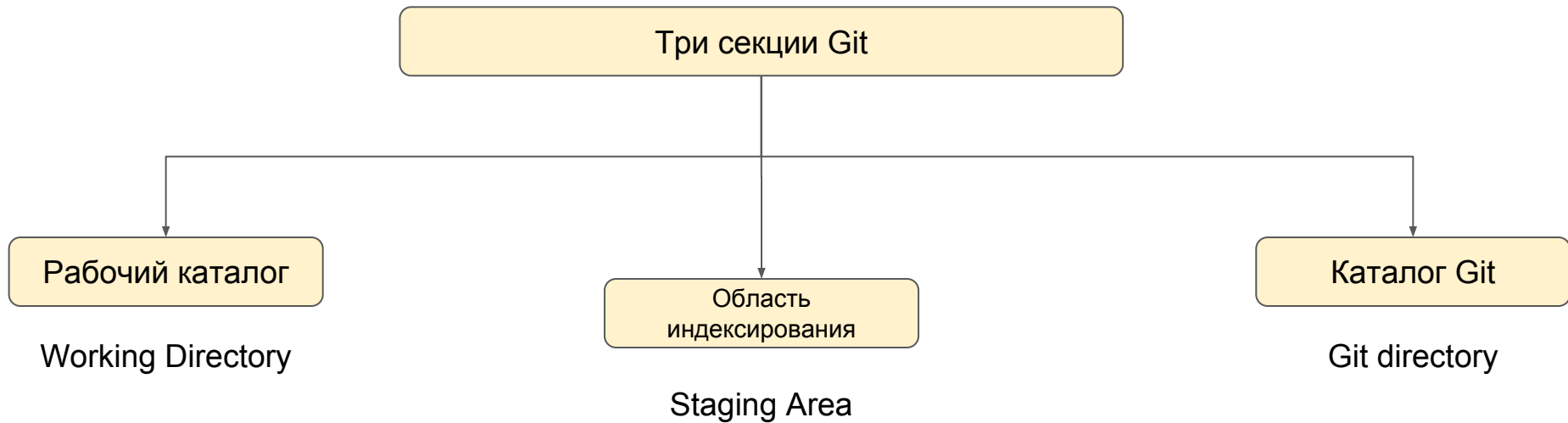
Version Control System (VCS). Распределенный тип



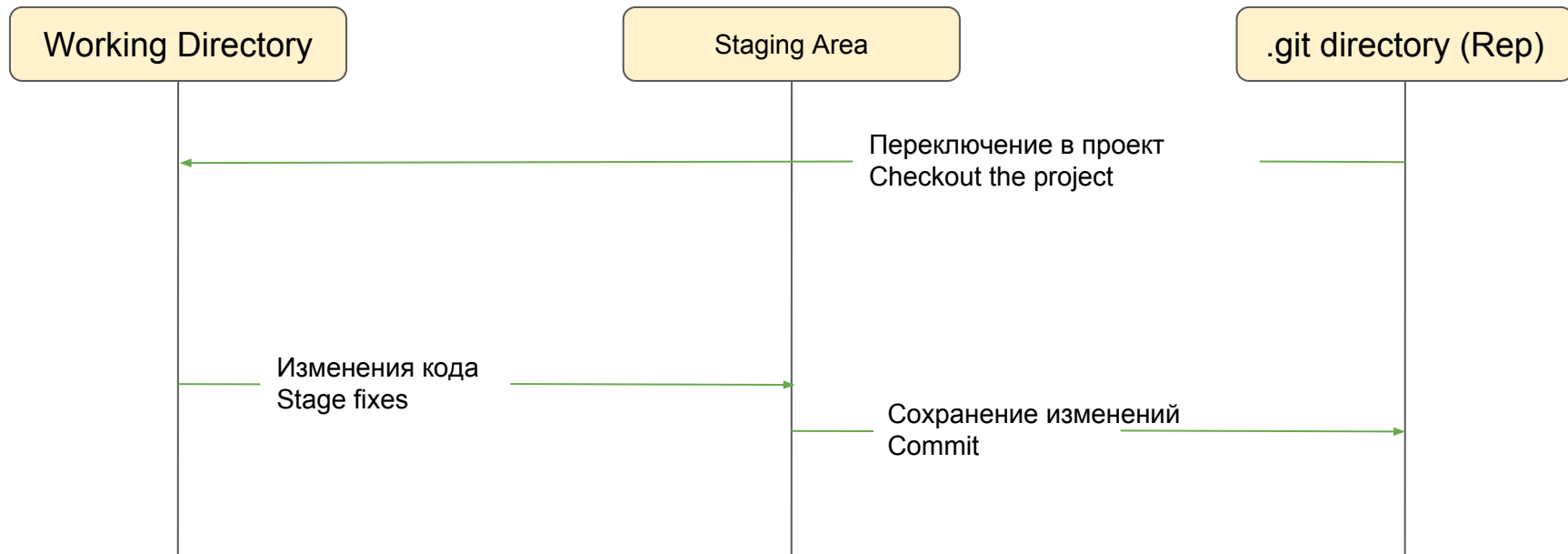
Git. Состояние файлов



Git. Секции.



Git. Секции.

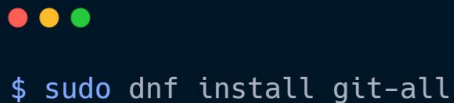


Git. Установка.

Все команды git выполняются в командной строке. Существуют также графические интерфейсы, которые для простоты реализуют часть функциональности git (e.g. Sourcetree) либо инструменты, встроенные в ваш IDE.

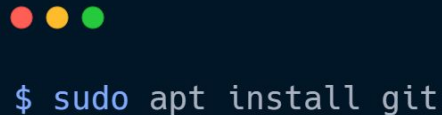
Linux

Для Fedora или схожего дистрибутива:



```
$ sudo dnf install git-all
```

Для Ubuntu или схожего дистрибутива:



```
$ sudo apt install git
```

MacOS

Вариант 1

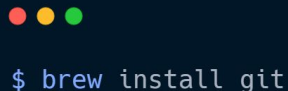
Скачиваем Xcode Command Line Tools. В терминале git -version. Если не установлен автоматически будет предложено установить.

Вариант 2

Качаем бинарный установщик с официального сайта Git.

Вариант 3

Установка через пакетный менеджер brew



```
$ brew install git
```

Git. Установка.

Windows

Вариант 1

Официальная сборка доступна для скачивания на официальном сайте Git. Перейдите и загрузка начнется автоматически

FYI: это отдельный проект, называемый Git для Windows. Чтобы получить больше информации что это такое перейдите <https://gitforwindows.org>

Вариант 2

Для автоматической установки вы можете использовать пакет Git Chocolatey.


Git. Настройка.

В состав Git входит утилита **git config**, которая **позволяет просматривать, настраивать и изменять параметры, контролирующие все аспекты Git**. Эти параметры могут быть в 3 местах:

1. Файл **[path]/etc/gitconfig** содержит значения, **общие для всех пользователей системы и для всех репозиториях**. Если при запуске **git config** указать параметр **--system**, то параметры будут читаться и сохраняться в этот файл. Так как этот файл системный, то для внесения изменений в него вам понадобятся права суперпользователя (sudo).
2. Файл **~/.gitconfig** или **~/.config/git/config** хранит **настройки конкретного пользователя**. Этот файл используется при указании параметра **--global** и применяется **ко всем репозиториям**, с которыми вы работаете в текущей системе.
3. Файл **config** в каталоге **Git** (т.е. **.git/config**) **репозитория**, который вы используете в настоящий момент, хранит настройки конкретного репозитория. Вы можете заставить Git читать и писать в этот файл с помощью параметра **--local** (по умолчанию).


Git. Настройка.

Чтобы посмотреть все установленные настройки и узнать, где именно они заданы:



```
$ git config --list --show-origin
```

Первое что **необходимо** сделать - **указать никнейм и адрес электронной почты (желательно того сервера где будет храниться репозиторий GitHub/GitLab и т.д.)**
Это важно, так как каждый коммит в Git содержит эту информацию. Информация включена в коммиты, передаваемые вами и не может быть далее изменена:



```
$ git config --global user.name "UserName"  
$ git config --global user.email UserName@gmail.com
```

Git. Настройка.

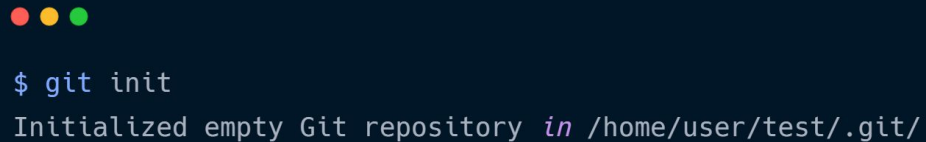
Когда вы **инициализируете репозиторий `git init`**, Git создаёт **ветку с именем `master` (`main`)** по умолчанию (зависит также от настроек при установке). Вы можете задать другое имя для создания ветки по умолчанию. Чтобы установить имя `main` для вашей ветки по умолчанию, используйте команду:



```
$ git config --global init.defaultBranch main
```


Git. Работа с репозиторием.

Репозиторий создается в конкретном каталоге с помощью команды:


A terminal window with a dark blue background and three colored window control buttons (red, yellow, green) in the top-left corner. The prompt '\$' is followed by the command 'git init'. The output line shows 'Initialized empty Git repository' followed by a purple 'in' and the path '/home/user/test/.git/'.

```
$ git init
Initialized empty Git repository in /home/user/test/.git/
```

В результате выполнения команды **git init** появится подкаталог **.git** в котором будут храниться служебные настройки **git'a** и сам **репозиторий**.

Git. Работа с репозиторием.

Git будет следить только за теми файлами, которые вы добавили в **staged area**. Это делается:

A dark-themed terminal window with three colored window control buttons (red, yellow, green) in the top-left corner. The text "\$ git add train.py" is displayed in a light blue monospaced font.

```
$ git add train.py
```

где train.py имя добавляемого файла.

Git. Работа с репозиторием.

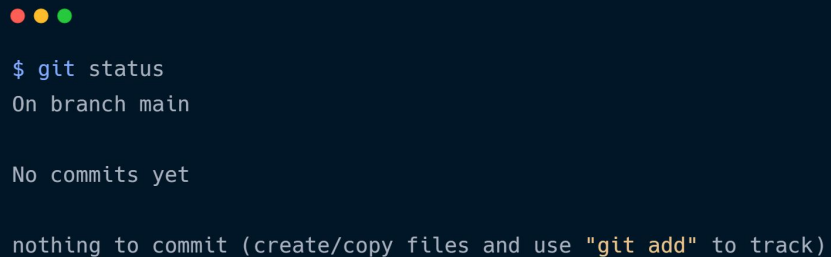
Существует ряд специальных символов (wildcards), которые позволяют упростить процесс работы с файлами и репозиторием.

Специальный символ * означает любую последовательность символов.

- `git add *` добавит в список отслеживаемых файлов все файлы в текущем каталоге и подкаталогах
- `git add *.py` добавит в список отслеживаемых файлов все файлы с расширением `.py` в текущем локальном репозитории

Git. Работа с репозиторием.

Текущий статус. Иногда необходимо отследить текущий статус репозитория:

A terminal window with a dark blue background and three colored window control buttons (red, yellow, green) in the top-left corner. The terminal displays the output of the 'git status' command.

```
$ git status
On branch main

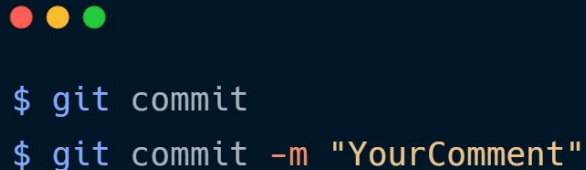
No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

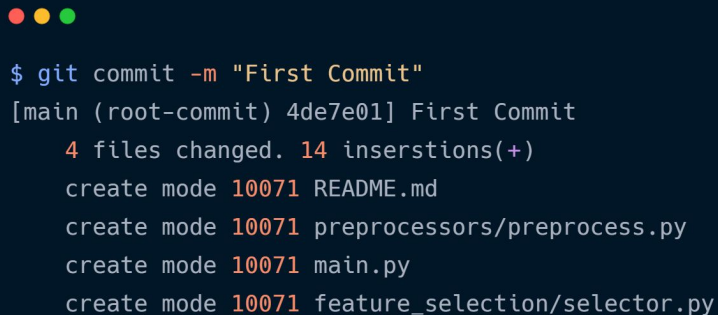
Git. Работа с репозиторием.

Фиксация изменений.

Для фиксации изменений используется команда **git commit** (после это откроется редактор для указания комментария). В редакторе нужно будет нажать Ctrl + O, Ctrl + X для записи и выхода из редактора.



```
$ git commit
$ git commit -m "YourComment"
```

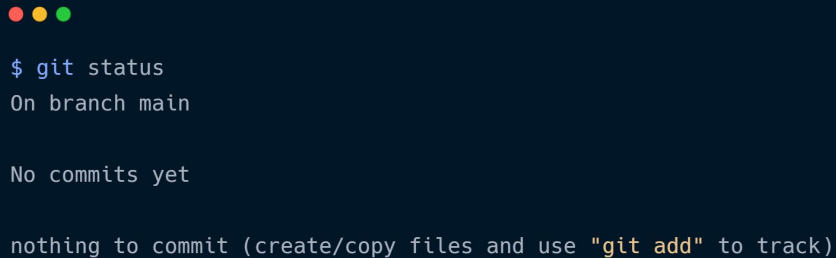


```
$ git commit -m "First Commit"
[main (root-commit) 4de7e01] First Commit
 4 files changed. 14 inserstions(+)
 create mode 10071 README.md
 create mode 10071 preprocessors/preprocess.py
 create mode 10071 main.py
 create mode 10071 feature_selection/selector.py
```

Git. Работа с репозиторием.

После фиксации изменений:

После коммита мы получаем чистое состояние, готовое к новому добавлению изменений и фиксации. Проверить можно командой:

A dark-themed terminal window with three colored window control buttons (red, yellow, green) in the top-left corner. The terminal displays the output of the 'git status' command.

```
$ git status
On branch main


No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

Git. Работа с репозиторием.

История

Посмотреть историю коммитов можно с помощью команды **git log**:

A terminal window with a dark blue background and three colored window control buttons (red, yellow, green) in the top-left corner. The terminal displays the output of the 'git log' command, showing two commits. The first commit is 'b56483e82746975a29b8847495de9f77f9e32ef9' with the message 'rabbitMQ functionality added'. The second commit is '278414a406aefbdc0c75e31beca59805e19e48a5' with the message 'Server side monitor'.

```
$ git log
commit b56483e82746975a29b8847495de9f77f9e32ef9 (HEAD -> master, origin/master)
Author: Marat <maratmovlamov2017@gmail.com>
Date:   Wed Jun 1 14:36:23 2022 +0300

    rabbitMQ functionality added

commit 278414a406aefbdc0c75e31beca59805e19e48a5
Author: Marat <maratmovlamov2017@gmail.com>
Date:   Mon May 30 14:07:03 2022 +0300

    Server side monitor
```

Git. Работа с репозиторием.

Идентификатор коммита. Просмотр информации о коммите.

Посмотреть полную информацию о коммите можно с помощью команды:

A terminal window with a dark blue background and three colored window control buttons (red, yellow, green) in the top-left corner. The terminal displays the output of the 'git show' command for commit 2784. The output shows the commit hash, a series of dots, and a diff summary for a file named '@@'.

```
$ git show 2784
commit 278414a406aefbdc0c75e31beca59805e19e48a5
.....
@@ -0,0 +1,540 @@
```


Git. Работа с репозиторием.

Если что-то пошло не так.

Если вы случайно добавили в stage area файл, который добавлять не нужно, либо добавили, но не сделали до конца свою работу, то удалить его можно **git rm --cached _filename_**:



```
$ git rm --cached train.py
```

Git. Работа с репозиторием.

Если что-то пошло не так.

Если вы залили коммит с ошибкой, то можно создать “зеркальный” коммит, который отменит действие предыдущего:



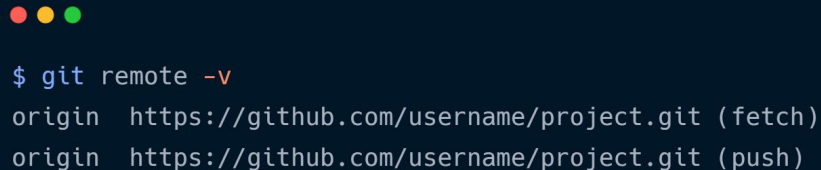
```
$ git revert commit-id
```

commit - id - идентификатор коммита, который можно посмотреть в логах (git log)

GitHub

Команда **git remote** позволяет управлять удалёнными репозиториями (добавлять, удалять, просматривать).

Например



```
$ git remote -v  
origin  https://github.com/username/project.git (fetch)  
origin  https://github.com/username/project.git (push)
```

Общепринято, что первый удаленный репозиторий называют **origin**

GitHub

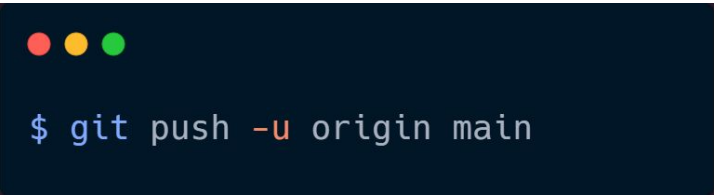
Добавление remote

Например, уже есть локальный репозиторий и мы хотим подключиться к удалённому



```
$ git remote add origin https://github.com/project_name/project.git
```

После того, как поработали локально, отправляем изменения в удаленный репозиторий. Для этого:



```
$ git push -u origin main
```

В случае если в конфигах не указаны учетные данные, попросит ввести пароль и логин. Последнее обновление позволяет вносить изменения только по SSH, подробнее [тут](#).

Git. Работа с репозиторием.

Когда делать git add

Важно запомнить следующий момент: **команда git commit фиксирует только те изменения, которые были добавлены в staging area через git add**

Поэтому если вы сделаете git add для файла, а затем измените его и сделаете git commit, то ваши последние изменения не зафиксируются, так как был пропущен **git add**

Git. Работа с репозиторием.

Gitignore

Иногда в проектах бывают лишние файлы, которые нужно хранить в репозитории, например:

- файлы проектов тестовых редакторов
- директорию виртуального окружения
- сборки, логи, компиляции
- файлы с учетными данными
- файлы автоматически генерируемые операционной системой

FYI: Всё, что хранится в репозитории будет доступно всем.

Для того, **чтобы игнорировать подобные файлы в Git есть специально настраиваемый файл, который называется .gitignore В нём перечисляются файлы и каталоги, которые Git должен игнорировать при работе.**

GitHub/GitLab/BitBucket

Специализированные сервисы, которые позволяют хранить репозиторий на удалённом сервере.

Git позволяет нам привязать к нашему репозиторию удалённый репозиторий, так, чтобы мы могли отправлять туда изменения своего проекта (локального репозитория) и получать обновления (если работают несколько человек над одним проектом).

Создайте аккаунт в любом вам удобном сервисе.

Git. Работа с репозиторием.

Работа в локальном репозитории:

Задача: У вас проект мл модели, состоящий из нескольких файлов и каталогов. Задача состоит в том, чтобы перевести работу с этим проектом в Git.

Общая схема работы:

1. Создание локального репозитория для проекта
2. Добавление файлов (в том числе измененных) в список отслеживания (staged area)
3. Фиксация изменений в локальном репозитории* (commit)

Операции 2 и 3 выполняются в течении всего развития проекта.

* Репозиторий - это хранилище истории и данных (файлов, каталогов, метаданных) вашего проекта.

Git. Работа с репозиторием.

Работа в локальном репозитории:

Задача:

1. Откройте консоль и создайте локальный проект. Проект должен иметь следующую структуру:

—project

- > src
 - > assets
- > model
 - > train
 - > utils
 - > data
- > app
- > experiments
- setup.py
- README.MD

Git. Работа с репозиторием.

Работа в локальном репозитории:

Задача:

1. Откройте консоль и создайте локальный проект. Проект должен иметь следующую структуру.
2. Проинициализируйте свой репозиторий в текущем проекте.

Git. Работа с репозиторием.

Работа в локальном репозитории:

Задача:

1. Откройте консоль и создайте локальный проект. Проект должен иметь следующую структуру.
2. Инициализируйте свой репозиторий в текущем проекте.
3. Добавьте все файлы и папки в staging area.

Git. Работа с репозиторием.

Работа в локальном репозитории:

Задача:

1. Откройте консоль и создайте локальный проект. Проект должен иметь следующую структуру.
2. Инициализируйте свой репозиторий в текущем проекте.
3. Добавьте все файлы и папки в staging area.
4. Посмотрите на статус.

Git. Работа с репозиторием.

Работа в локальном репозитории:

Задача:

1. Откройте консоль и создайте локальный проект. Проект должен иметь следующую структуру.
2. Инициализируйте свой репозиторий в текущем проекте.
3. Добавьте все файлы и папки в staging area.
4. Посмотрите на статус.
5. Добавьте файл .gitignore и сделайте так чтобы репозиторий игнорировал путь **models/data/**

Git. Работа с репозиторием.

Работа в локальном репозитории:

Задача:

1. Откройте консоль и создайте локальный проект. Проект должен иметь следующую структуру.
2. Инициализируйте свой репозиторий в текущем проекте.
3. Добавьте все файлы и папки в staging area.
4. Посмотрите на статус.
5. Добавьте файл .gitignore и сделайте так чтобы репозиторий игнорировал путь **models/data/**
6. Отмените коммит и сделайте новый

Git. Работа с репозиторием.

Работа в локальном репозитории:

Задача:

1. Откройте консоль и создайте локальный проект. Проект должен иметь следующую структуру.
2. Инициализируйте свой репозиторий в текущем проекте.
3. Добавьте все файлы и папки в staging area.
4. Посмотрите на статус.
5. Добавьте файл .gitignore и сделайте так чтобы репозиторий игнорировал путь **models/data/**
6. Отмените коммит и сделайте новый
7. Создайте удаленный репозиторий и свяжите его с текущим репозиторием.

Git. Работа с репозиторием.

Работа в локальном репозитории:

Задача:

1. Откройте консоль и создайте локальный проект. Проект должен иметь следующую структуру.
2. Инициализируйте свой репозиторий в текущем проекте.
3. Добавьте все файлы и папки в staging area.
4. Посмотрите на статус.
5. Добавьте файл .gitignore и сделайте так чтобы репозиторий игнорировал путь **models/data/**
6. Отмените коммит и сделайте новый
7. Создайте удаленный репозиторий и свяжите его с текущим репозиторием.
8. Залейте весь проект на удаленный репозиторий.

3. Package Management. Pip.
Conda. Virtual Environments.

PIP.

Стандартный менеджер пакетов для Python. Позволяет устанавливать пакеты не входящие в стандартную библиотеку. Устанавливается сразу с интерпретатором. Python обычно поставляется со всеми "батарейками", но в виду развитого комьюнити существует множество сторонних модулей и библиотек, которые необходимы разработчику. Данные модули и библиотеки публикуются на Python Package Index сервисе, известном также как PyPI (ПайПи, ПайПиАй).

PIP. Создание виртуального окружения.

Виртуальное окружение (среда) - среда в Python, в которой интерпретатор, скрипты, библиотеки, фреймворки, установленные в ней изолированы от тех, что установлены в других виртуальных окружениях (средах) и ограничены от системных стандартных библиотек Python.



```
$ python -m venv venv
$ source venv/bin/activate
```

PIP. Установка модулей и библиотек.

Если приходится работать с модулем, который отсутствует в стандартной библиотеке Python, то его можно установить в виртуальное окружение, используя команду:

```
(venv) $ python -m pip install scikit-learn
(venv) $ python -m pip install scikit-learn numpy pandas statsmodels
```

PIP. Просмотр модулей и библиотек.

Чтобы посмотреть установленные библиотеки и модули:

```
(venv) $ python -m list
Package      Version
-----
certifi      x.y.z
numpy        x.y.z
statsmodels  x.y.z
pip          x.y.z
requests     x.y.z
setuptools   x.y.z
urllib3      x.y.z
```

PIP. Использование Альтернативного Источника.

Редко, но бывает такое, что необходимо хранить и загружать модули и пакеты с сервера компании, чтобы не нарушать конфиденциальность и не распространять лицензионное ПО. Для этого можно поменять источник индекса, тем самым все модули будут загружаться с вашего источника:

```
(venv) $ python -m pip config list -vv
# pip.conf

[global]
index-url = https://test.pypi.org/simple/
```

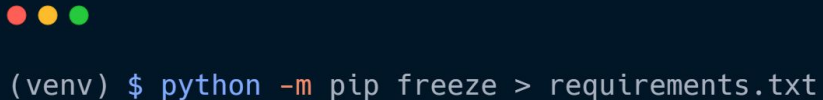
PIP. Установка с GitHub.

Пакетный менеджер также позволяет устанавливать модули и пакеты с GitHub:

```
(venv) $ python -m pip install git+https://github.com/somepackage/somemodule
```

PIP. Requirements.

Бывает, что вы работаете над проектом с другим разработчиком, либо отгружаете проект в продакшн, необходимо указать какие версии пакетов вы используете, чтобы не возникало конфликтов с кодом. Часто скачивая чужой код с репозитория вы найдете файл requirements.txt в нем хранятся записи о версии пакетов и модулей. Чтобы сохранить версии пакетов для своего проекта, необходимо:



```
(venv) $ python -m pip freeze > requirements.txt
```

Установка пакетов происходит следующим образом:



```
(venv) $ python -m pip install -r requirements.txt
```

Conda.

Вторая по популярности система управления пакетами, зависимостями и виртуальным окружением.

Может упаковывать и распространять ПО **для любого языка программирования**.

При помощи Conda также можно выполнять работу по созданию виртуального окружения, особенность - можно также контролировать версию языка, управлять зависимостями других языков и прочее.

Рассмотрим базовые команды пакетного менеджера, которые вам понадобятся на протяжении всего курса, если вы будете использовать Conda.

Conda.

- Проверить, что conda действительно установлена в системе можно просто набрав в консоли:

\$ conda -V

- Старайтесь чтобы у вас всегда стояла последняя версия пакетного менеджера, как убедиться:

\$ conda update conda

- Создаем виртуальное окружение с указанием версии языка:

\$ conda create -n _envname_ python=x.x anaconda

- После того, как создали виртуальное окружение активируем при помощи команды:

\$ source activate _envname_ или \$ conda activate _envname_

Conda.

- Установка модулей и пакетов происходит при помощи следующей команды

```
$ conda install -n _envname_ [package]
```

- Деактивация виртуального окружения происходит следующим образом

```
$ conda deactivate _envname_ или source deactivate
```

- Удалить виртуальное окружение можно при помощи команды:

```
$ conda remove -n _envname_ -all
```


4. Экскурс в язык Python.

Python.

Простой синтаксис

```
def monte_carlo_pi(nsamples):  
    acc = 0  
    for i in range(nsamples):  
        x = random.random()  
        y = random.random()  
        if (x ** 2 + y ** 2) < 1.0:  
            acc += 1  
    return 4.0 * acc / nsamples
```

```
#include <bits/stdc++.h>  
#define INTERVAL 10000  
using namespace std;  
  
int main()  
{  
    int interval, i;  
    double rand_x, rand_y, origin_dist, pi;  
    int circle_points = 0, square_points = 0;  
    srand(time(NULL));  
    for (i = 0; i < (INTERVAL * INTERVAL); i++) {  
        rand_x = double(rand() % (INTERVAL + 1)) / INTERVAL;  
        rand_y = double(rand() % (INTERVAL + 1)) / INTERVAL;  
        origin_dist = rand_x * rand_x + rand_y * rand_y;  
        if (origin_dist <= 1)  
            circle_points++;  
        square_points++;  
        pi = double(4 * circle_points) / square_points;  
        cout << rand_x << " " << rand_y << " "  
            << circle_points << " " << square_points  
            << " - " << pi << endl  
            << endl;  
        if (i < 20)  
            getchar();  
    }  
    cout << "\nFinal Estimation of Pi = " << pi;  
    return 0;  
}
```

Python.

Динамическая типизация

```
a = 1
b = "1"
print("Тип переменной a и b: \n{}\n {}".format(type(a), type(b)))

>>> Тип переменной a и b:
      <class 'int'>
      <class 'str'>
```

```
a = 1
b = "1"
print("{} + {} = {}".format(a, b, a + int(b)))

>>> 1 + 1 = 2
```

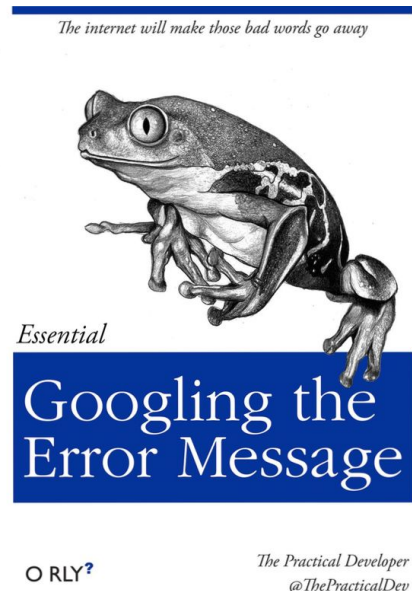
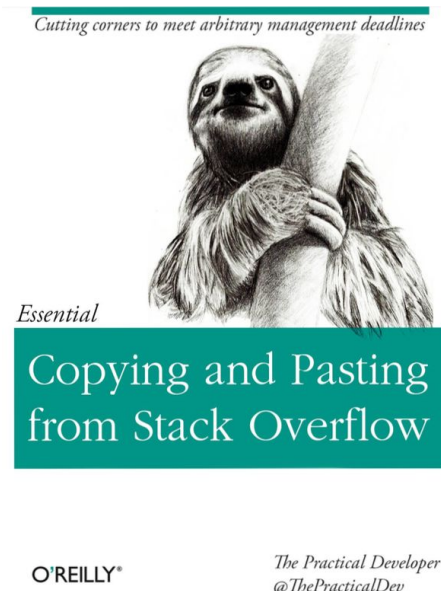
Python.

Популярность

- Существует **огромное количество библиотек** для работы с данными и ML
- Все большие системы и **компании пишут API** и коннекторы (Google Cloud API, Aws API etc.)
- **Базы данных** (paramiko, pymysql, pymongo) и **инструменты для работы с большими данными** (PySpark, AirFlow, Hadoop, DataProc etc).

Python.

Большое КОМЬЮНИТИ



Python. Общие Рекомендации

- » Называйте переменные **интуитивно понятными названиями**
- » **Старайтесь разбивать код** на отдельные функции и (или) классы
- » Каждая функция (класс) должна содержать **описание** и желательно пример её вызова (docstring)
- » Не забывайте про **полезные комментарии**
- » Для каждой задачи **решите какой инструмент наиболее подходящий** и обсудите его с коллегой
- » Строки с параметрами выводить лучше с помощью **format()**

Python. Общие Рекомендации

```
def get_pairs(word):  
    """  
    Return set of symbol pairs in a word.  
    Word is represented as tuple of symbols.  
    Symbols are variable-length strings.  
    """  
    pairs = set()  
    prev_character = word[0]  
    for charecter in word[1:]:  
        pairs.add((prev_character, charecter))  
        prev_character = charecter  
    return pairs
```



```
# write function  
def write(u, w):  
    # for cycle  
    for l in u:  
        # if cycle  
        if l in w:  
            # return l  
            return l  
        else:  
            # return Nothing  
            return None
```



Python. Структура проекта

Непрактично писать весь код в одном файле. Да, это хороший вариант для one-off скриптов, где 100-200 строк кода, но для больших проектов это не вариант.

Большие проекты, как правило, состоят из тысяч строк кода. Python предоставляет нам удобную структуру **package** , которая позволяет разбить группы модулей в проекте.

Package это просто папка, которая содержит специальный файл `__init__.py` который указывает интерпритатору что это пакет.

Python. Структура проекта

Package это просто папка, которая содержит специальный файл **`__init__.py`** который указывает интерпритатору что это пакет.

Пример структуры проекта:

```
example
├── core.py
├── run.py
└── util
    ├── __init__.py
    ├── db.py
    ├── math.py
    └── network.py
```

Python. Как использовать модули и пакеты

Когда разработчик пишет приложение, вполне вероятно, что ему потребуется применить одну и ту же логику в разных его частях.

Например, при написании кода для анализа данных, поступающих из формы, которую пользователь может заполнить на веб-странице, приложение должно будет проверить, содержит ли определенное поле число или нет.

Независимо от того, как написана логика для такого рода проверки, вполне вероятно, что она понадобится более чем для одного поля.

Python. Как использовать модули и пакеты

Плохая практика - дублировать логику кода везде в проекте.

Это нарушает принцип **DRY (Don't repeat yourself)**, который утверждает, что вы должны по максимуму никогда не повторять одну и ту же логику в своем приложении.

Поэтому мы можем спрятать часть логики для последующего использования в **package** либо **упаковать все в библиотеку (library)** - коллекцию функций и объектов.

Python. Built-In Data Types

Быстро вспомним и проверим ваше профессиональное понимание языка программирования.

Абсолютно всё в Python - это объекты.

Что происходит когда создается объект в Python?

Например **a = 42** Не берем пока в расчет области видимости (scopes).

Python. Built-In Data Types

Абсолютно всё в Python - это объекты.

Что происходит когда создается объект в Python?

Например **a = 42** Не берем пока в расчет области видимости (scopes).

1. Объекту присваивается id

Python. Built-In Data Types

Абсолютно всё в Python - это объекты.

Что происходит когда создается объект в Python?

Например **a = 42** Не берем пока в расчет области видимости (scopes).

1. Объекту присваивается id
2. Объекту сопоставляется тип int (type(a))

Python. Built-In Data Types

Абсолютно всё в Python - это объекты.

Что происходит когда создается объект в Python?

Например **a = 42** Не берем пока в расчет области видимости (scopes).

1. Объекту присваивается id
2. Объекту сопоставляется тип int (type(a))
3. Объект ссылается на значение в памяти (42)

Python. Built-In Data Types

Абсолютно всё в Python - это объекты.

Что происходит когда создается объект в Python?

Например **a = 42** Не берем пока в расчет области видимости (scopes).

1. Объекту присваивается id
2. Объекту сопоставляется тип int (type(a))
3. Объект ссылается на значение в памяти (42)

Как визуализировать понимание кода: <http://pythontutor.com/>

Python. Built-In Data Types

Что важно понимать в типах данных - это принцип изменяемости (**mutability**) и неизменяемости (**immutability**).

```
>>> age = 42
>>> age
42
```

```
>>> age = 43
>>> age
43
```

Посмотрите на пример выше и проверьте ваше понимание вашего рабочего языка как профессионала. Изменили ли мы значение переменной age?

Python. Built-In Data Types

Какие встроенные типы данных присутствуют в Python?

Python. Built-In Data Types

Какие встроенные типы данных присутствуют в Python?

- Numbers

Python. Built-In Data Types

Какие встроенные типы данных присутствуют в Python?

- Numbers
 - Integers

Python. Built-In Data Types

Какие встроенные типы данных присутствуют в Python?

- Numbers
 - Integers
 - Floating point

Python. Built-In Data Types

Какие встроенные типы данных присутствуют в Python?

- Numbers
 - Integers
 - Floating point
 - Booleans

Python. Built-In Data Types

Какие встроенные типы данных присутствуют в Python?

- Numbers
 - Integers
 - Floating point
 - Booleans
 - Real Numbers

Python. Built-In Data Types

Какие встроенные типы данных присутствуют в Python?

- Numbers
 - Integers
 - Floating point
 - Booleans
 - Real Numbers
 - Complex Numbers

Python. Built-In Data Types

Какие встроенные типы данных присутствуют в Python?

- Numbers
 - Integers
 - Floating point
 - Booleans
 - Real Numbers
 - Complex Numbers
 - Fractions and Decimals

Python. Built-In Data Types

Какие встроенные типы данных присутствуют в Python?

- Numbers
- Strings

Python. Built-In Data Types

Какие встроенные типы данных присутствуют в Python?

- Numbers
- Strings
- Dates and times

Python. Built-In Data Types

Какие встроенные типы данных присутствуют в Python?

- Numbers
- Strings
- Dates and times
- Sequences

Python. Built-In Data Types

Какие встроенные типы данных присутствуют в Python?

- Numbers
- Strings
- Dates and times
- Sequences
- Collections

Python. Built-In Data Types

Какие встроенные типы данных присутствуют в Python?

- Numbers
- Strings
- Dates and times
- Sequences
- Collections
- Mapping Types

Python. Built-In Data Types

Какие встроенные типы данных присутствуют в Python?

- Numbers
- Strings
- Dates and times
- Sequences
- Collections
- Mapping Types

Python. Immutable Sequences

Python. Immutable Sequences

Immutable Sequences: **strings, tuples и bytes**

Python. Immutable Sequences

Strings

Строковые значения в Python - неизменяемая последовательность юникод символов.

```
>>> str1 = 'This is a string. We built it with single quotes.'
```

```
>>> str2 = "This is also a string, but built with double quotes."
```

```
>>> str3 = '''This is built using triple quotes,  
... so it can span multiple lines.'''
```

```
>>> str4 = """This too  
... is a multiline one  
... built with triple double-quotes."""
```

```
>>> str4  
'This too\nis a multiline one\nbuilt with triple double-quotes.'
```

```
>>> print(str4)  
This too  
is a multiline one  
built with triple double-quotes.
```

Python. Immutable Sequences

Strings

Управляя последовательностями, можно получать доступ к определенной части ее (**индексирование/indexing**) или получить из них подпоследовательность (**слайсинг/slicing**). При работе с неизменяемыми последовательностями обе операции доступны только в read-only формате.

```
>>> s = "The trouble is you think you have time."
>>> s[0] # indexing at position 0
'T'

>>> s[5] # indexing at position 5
'r'

>>> s[:4] # slicing, with the stop position
'The '

>>> s[4:] # slicing, we the start position
'trouble is you think you have time.'

>>> s[2:14] # slicing, both start and stop positions
'e trouble is'

>>> s[2:14:3] # slicing, start, stop and step (every 3 chars)
'erb '

>>> s[:] # making a copy
'The trouble is you think you have time.'
```

Python. Immutable Sequences

String Formatting

У нас есть возможность форматирования строковой переменной различными способами. Все способы форматирования строковых переменных доступны в документации языка. Часто используемые:

```
>>> greet_old = 'Hello %s!'
```

```
>>> greet_old % 'Marat'
'Hello Marat!'
```

```
>>> greet_positional = 'Hello {}!'
>>> greet_positional.format('Marat')
'Hello Marat!'
```

```
>>> greet_positional = 'Hello {} {}!'
>>> greet_positional.format('Marat', 'Maulamau')
'Hello Marat Maulamau!'
```

```
>>> greet_positional_idx = 'This is {0}! {1} loves {0}!'
>>> greet_positional_idx.format('Python', 'I')
'This is Python! I love Python!'
```

```
>>> greet_positional_idx.format('Coffee', 'Marat')
'This is Coffee! Marat loves Coffee!'
```

```
>>> keyword = 'Hello, my name is {name} {last_name}'
>>> keyword.format(name='Marat', last_name='Maulamau')
'Hello, my name is Marat Maulamau'
```

Python. Immutable Sequences

Tuples

Tuple (Кортеж) — это последовательность произвольных объектов Python. В объявлении кортежа элементы разделяются запятыми. Кортежи используются в Python повсеместно. Иногда кортежи используются неявно (возврат нескольких объектов из функции, для вывода нескольких элементов и тд):

```
>>> t = ()    # empty tuple

>>> type(t)
<class 'tuple'>

>>> one_element_tuple = (42, )    # you need the comma!

>>> three_elements_tuple = (1, 3, 5)    # braces are optional here

>>> a, b, c = 1, 2, 3    # tuple for multiple assignment

>>> a, b, c    # implicit tuple to print with one instruction
(1, 2, 3)

>>> 3 in three_elements_tuple    # membership test
True
```

Оператор включения используется не только с кортежами, но и со всеми коллекциями (списки, строки, словари и прочими коллекциями и последовательностями)

Python. Mutable Sequences

Lists (Списки)

Списки в Python очень похожи на кортежи, но не имеют ограничений по иммутабельности (они мутабельны). Списки обычно используются для хранения коллекций однородных объектов, но ничто не мешает хранить и разнородные коллекции и объекты. Списки можно создавать разными способами:

```
>>> [] # empty list
[]
```

```
>>> list() # same as []
[]
```

```
>>> [1, 2, 3] # as with tuples, items are comma separated
[1, 2, 3]
```

```
>>> [x + 5 for x in [2, 3, 4]] # List Comprehension
[7, 8, 9]
```

```
>>> list((1, 3, 5, 7, 9)) # list from a tuple
[1, 3, 5, 7, 9]
```

```
>>> list('hello') # list from a string
['h', 'e', 'l', 'l', 'o']
```

Python. Mutable Sequences

Lists (Списки)

Списки обладают огромным количеством методов, которые вы будете использовать все время на протяжении работы с ними. Те методы, которые вам необходимо знать и помнить:

```
>>> a = [1, 2, 1, 3]
>>> a.append(13)  # добавление элемента в конец списка
>>> a
[1, 2, 1, 3, 13]

>>> a.count(1)  # подсчет вхождений
2

>>> a.extend([5, 7])  # добавить другую коллекцию в текущий список
>>> a
[1, 2, 1, 3, 13, 5, 7]

>>> a.index(13)  # позиция элемента в списке (индекс)
4

>>> a.insert(0, 17)  # добавить элемент в определенную позицию списка, добавление по индексу
>>> a
[17, 1, 2, 1, 3, 13, 5, 7]
```

Python. Mutable Sequences

Lists (Списки)

Списки обладают огромным количеством методов, которые вы будете использовать все время на протяжении работы с ними. Те методы, которые вам необходимо знать и помнить:

```
>>> a.pop() # pop (remove and return) last element
7
```

```
>>> a.pop(3) # pop element at position 3
1
>>> a
[17, 1, 2, 3, 13, 5]
```

```
>>> a.remove(17) # remove `17` from the list
>>> a
[1, 2, 3, 13, 5]
```

```
>>> a.reverse() # reverse the order of the elements in the list
>>> a
[5, 13, 3, 2, 1]
```

```
>>> a.sort() # sort the list
>>> a
[1, 2, 3, 5, 13]
```

```
>>> a.clear() # remove all elements from the list
>>> a
[]
```


Python. Mutable Sequences

Set (множества)

2 вида - Set и Frozenset. Set - мутабельная коллекция данных, а frozenset - иммутабельная коллекция. [Хэшируемость](#) - характеристика, которая позволяет использовать объект в качестве члена множества, а также в качестве ключа для словаря.

```
>>> small_primes = set() # empty set

>>> small_primes.add(2) # adding one element at a time

>>> small_primes.add(3)

>>> small_primes.remove(1) # remove element 1 from set

>>> 3 in small_primes # membership test

>>> bigger_primes = set([5, 7, 11, 13]) # faster creation

>>> small_primes | bigger_primes # union operator `|`

>>> small_primes & bigger_primes # intersection operator `&`
{5}

>>> small_primes - bigger_primes # difference operator `-`
{2, 3}
```

Python. Mapping Types: dict

Dictionaries / Dict (словари):

Dict сопоставляет пары ключ-значение (**key-val pairs**). Ключи должны быть хешируемыми объектами, а значения могут быть любого произвольного типа. Словари также являются **изменяемыми** объектами.

```
>>> a = dict(A=1, Z=-1)
```

```
>>> b = {'A': 1, 'Z': -1}
```

```
>>> c = dict(zip(['A', 'Z'], [1, -1]))
```

```
>>> d = dict([('A', 1), ('Z', -1)])
```

```
>>> e = dict({'Z': -1, 'A': 1})
```

```
>>> a == b == c == d == e # что выведет?
```

Python. Mapping Types: dict

Dictionaries / Dict (словари):

Методы, используемые со словарями:

```
>>> d = {}  
>>> d['a'] = 1  
>>> d['b'] = 2  
  
>>> len(d) # Что будет?
```

Python. Mapping Types: dict

Dictionaries / Dict (словари):

Методы, используемые со словарями:

```
>>> d = {}  
>>> d['a'] = 1  
>>> d['b'] = 2
```

```
>>> len(d) # number of pairs in the dictionary
```

```
>>> d['a'] # what is the value of 'a'?  
1
```

Python. Mapping Types: dict

Dictionaries / Dict (словари):

Методы, используемые со словарями:

```
>>> d = {}  
>>> d['a'] = 1  
>>> d['b'] = 2
```

```
>>> len(d)  # number of pairs in the dictionary
```

```
>>> d['a']  # what is the value of 'a'?  
1
```

```
>>> del d['a']  # remove `a`  
>>> d['c'] = 3  # add 'c': 3
```

Python. Mapping Types: dict

Dictionaries / Dict (словари):

Методы, используемые со словарями:

```
>>> d = {}
>>> d['a'] = 1
>>> d['b'] = 2

>>> len(d) # number of pairs in the dictionary

>>> d['a'] # what is the value of 'a'?
1

>>> del d['a'] # remove `a`
>>> d['c'] = 3 # add 'c': 3

>>> 'c' in d # membership is checked against the keys
True
>>> 3 in d # not the values
True
>>> 'e' in d
False
>>> d.clear() # let's clean everything from this dictionary
>>> d
{}
```

Python. Mapping Types: dict - dynamic view

Dictionaries / Dict (словари) - keys, values, items:

- **keys()** - возврат всех ключей из словаря
- **values()** - возврат всех значений из словаря
- **items()** - возврат всех пар ключ-значений из словаря

```
>>> d = dict(zip('hello', range(5)))
```

```
>>> d
```

```
{'h': 0, 'e': 1, 'l': 3, 'o': 4}
```

```
>>> d.keys()
```

```
dict_keys(['h', 'e', 'l', 'o'])
```

```
>>> d.values()
```

```
dict_values([0, 1, 3, 4])
```

```
>>> d.items()
```

```
dict_items([('h', 0), ('e', 1), ('l', 3), ('o', 4)])
```

```
>>> 3 in d.values()
```

```
True
```

```
>>> ('o', 4) in d.items()
```

```
True
```

Подробнее о большем количестве методов для словарей в документации

<https://docs.python.org/3/tutorial/datastructures.html#dictionaries>

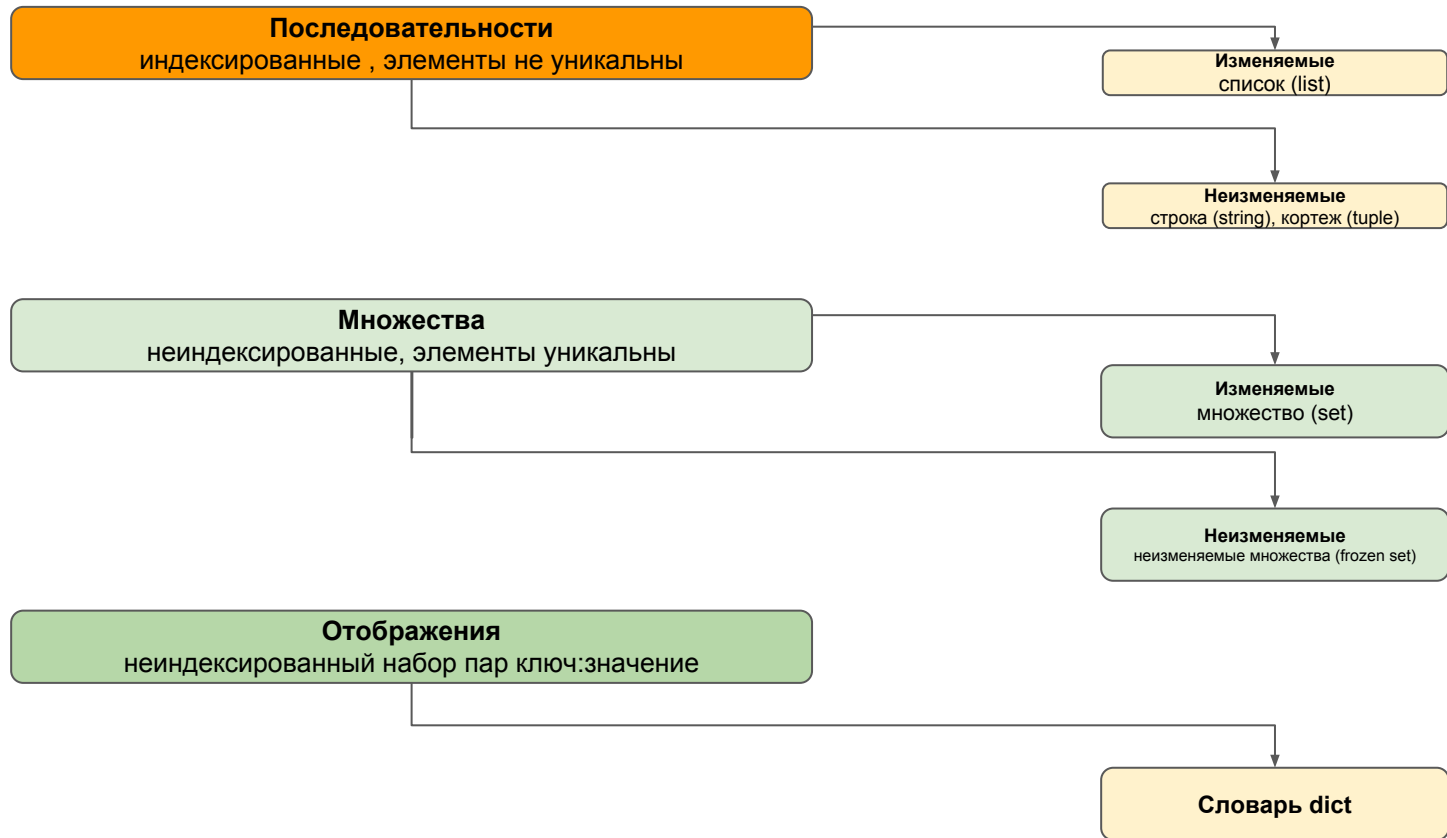
Python. Collections module

Когда встроенных коллекций общего назначения Python (**tuple, list, set и dict**) недостаточно, мы можем найти специализированные типы данных коллекций в модуле **collections**.

- namedtuple()
- deque
- ChainMap
- Counter
- OrderedDict
- defaultdict
- UserDict
- UserList
- UserString

<https://docs.python.org/3/library/collections.html>

Python. Collections Recap.



Conditionals and Iteration

Условные операторы:

1. **if**
2. **if...else**
3. **if...elif...else**
4. **match**

Операторы потока:

1. **for**
2. **while**

```
1. customers = [  
2.     dict(id=1, total=200, coupon_code='F20'), # F20: fixed, £20  
3.     dict(id=2, total=150, coupon_code='P30'), # P30: percent, 30%  
4.     dict(id=3, total=100, coupon_code='P50'), # P50: percent, 50%  
5.     dict(id=4, total=110, coupon_code='F15'), # F15: fixed, £15  
6. ]  
7. discounts = {  
8.     'F20': (0.0, 20.0), # each value is (percent, fixed)  
9.     'P30': (0.3, 0.0),  
10.    'P50': (0.5, 0.0),  
11.    'F15': (0.0, 15.0),  
12. }  
13. for customer in customers:  
14.     code = customer['coupon_code']  
15.     percent, fixed = discounts.get(code, (0.0, 0.0))  
16.     customer['discount'] = percent * customer['total'] + fixed  
17.  
18. for customer in customers:  
19.     print(customer['id'], customer['total'], customer['discount'])
```

```
# primes.else.py  
primes = []  
upto = 100  
for n in range(2, upto + 1):  
    for divisor in range(2, n):  
        if n % divisor == 0:  
            break  
    else:  
        primes.append(n)  
print(primes)
```

Functions

Функция - фрагмент программного кода, к которому можно обратиться из другого места программы. С именем функции связан адрес первой инструкции, входящей в функцию, которой передается управление при обращении к функции.

Функции в Python определяются при помощи **def**, после ключевого слова следует название функции и ее аргументы:

```
def some_function(*args, **kwargs):  
    pass
```

```
def calculate_price_with_vat(price, vat):  
    return price * (100 + vat) / 100
```

Как вы уже знаете, функции могут возвращать, а могут и не возвращать значения. Возврат значения определяется при помощи ключевого слова **return**.

!По дефолту функции в Python возвращают объект **None**.

Functions. Scopes

Scope (область видимости).

Во время исполнения вашего кода интерпретатор Python исполняет код согласно следующей последовательности в пространстве имен (правило **LEGB**):

1. **L**ocal
2. **E**nclosing
3. **G**lobal
4. **B**uilt-in

```
def outer():  
    test = 1 # outer scope  
    def inner():  
        test = 2 # inner scope  
        print('inner:', test)  
    inner()  
    print('outer:', test)  
test = 0 # global scope  
outer()  
print('global:', test)
```

Functions. Передача аругментов

4 основных способа передачи аругментов:

Functions. Передача аругментов

4 основных способа передачи аругментов:

1. Positional arguments

Functions. Передача аругментов

4 основных способа передачи аругментов:

1. Positional arguments
2. Keyword arguments

Functions. Передача аругментов

4 основных способа передачи аругментов:

1. Positional arguments
2. Keyword arguments
3. Iterable unpacking

Functions. Передача аругментов

4 основных способа передачи аругментов:

1. Positional arguments
2. Keyword arguments
3. Iterable unpacking
4. Dictionary unpacking

Functions. Передача аругментов

4 основных способа передачи аругментов:

1. Positional arguments
2. Keyword arguments
3. Iterable unpacking
4. Dictionary unpacking

```
# arguments -> positional  
def func(a, b, c):  
    print(a, b, c)  
func(1, 2, 3) # prints: 1 2 3
```

Functions. Передача аргументов

4 основных способа передачи аргументов:

1. Positional arguments
2. Keyword arguments
3. Iterable unpacking
4. Dictionary unpacking

arguments -> positional

```
def func(a, b, c):
```

```
    print(a, b, c)
```

```
func(1, 2, 3)  # prints: 1 2 3
```

arguments -> keyword

```
def func(a, b, c):
```

```
    print(a, b, c)
```

```
func(a=1, c=2, b=3)  # prints: 1 3 2
```

Functions. Передача аргументов

4 основных способа передачи аргументов:

1. Positional arguments
2. Keyword arguments
3. Iterable unpacking
4. Dictionary unpacking

```
# arguments -> unpack dict
```

```
def func(a, b, c):  
    print(a, b, c)
```

```
values = {'b': 1, 'c': 2, 'a': 42}
```

```
func(**values) # equivalent to func(b=1, c=2, a=42)
```

```
# arguments -> positional
```

```
def func(a, b, c):  
    print(a, b, c)
```

```
func(1, 2, 3) # prints: 1 2 3
```

```
# arguments -> keyword
```

```
def func(a, b, c):  
    print(a, b, c)
```

```
func(a=1, c=2, b=3) # prints: 1 3 2
```

```
# arguments -> unpack iterable
```

```
def func(a, b, c):  
    print(a, b, c)
```

```
values = (1, 3, -7)
```

```
func(*values) # equivalent to: func(1, 3, -7)
```

Functions. Variable keyword parameters

Variable keyword parameters:

Синтаксис похожий на позиционную передачу параметров, за одним исключением - **используем **** вместо *. Переданные аргументы хранятся в виде словаря.

```
# parameters variable
def connect(**options):
    conn_params = {
        'host': options.get('host', '127.0.0.1'),
        'port': options.get('port', 5432),
        'user': options.get('user', ''),
        'pwd': options.get('pwd', ''),
    }
    print(conn_params)
    # we then connect to the db (commented out)
    # db.connect(**conn_params)

connect()
connect(host='127.0.0.42', port=5433)
connect(port=5431,
        ser='Spock',
        pwd='Discovery')
```

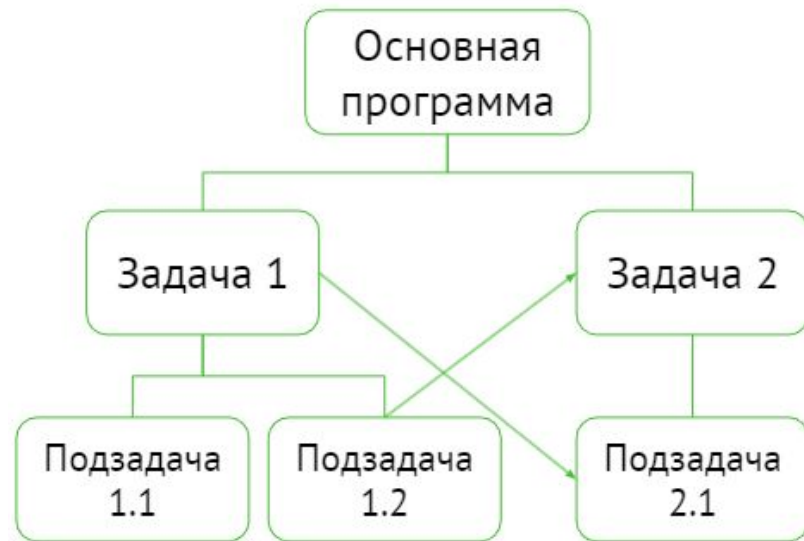
Подходы к программированию

Какие подходы к программированию вам известны?

Процедурное программирование

Процедурное (функциональное) программирование - это парадигма программирования, в которой используется линейный или нисходящий подход.

Он полагается на процедуры (функции) или подпрограммы для выполнения вычислений.

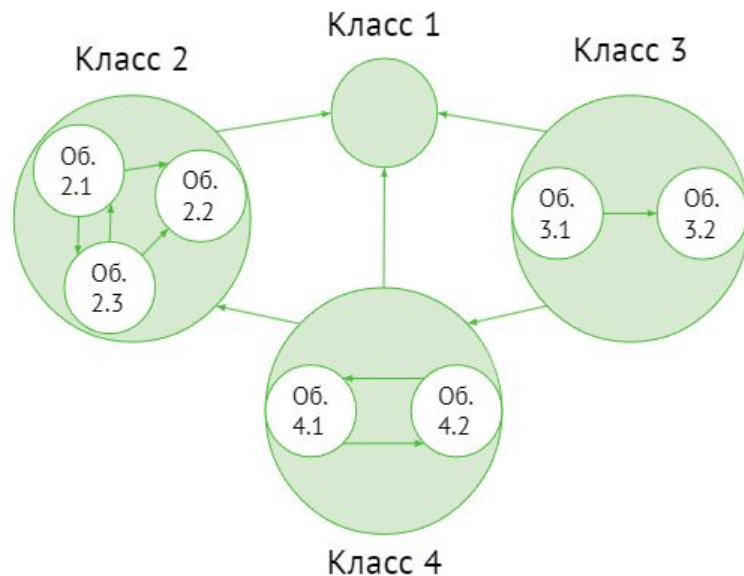


Пример взаимосвязей функций в программе

Объектно-ориентированное программирование

Объектно-ориентированное программирование (ООП) - это парадигма программирования, построенная вокруг объектов.

Она разделяет данные на объекты и описывает содержимое и поведение объекта посредством объявления классов.

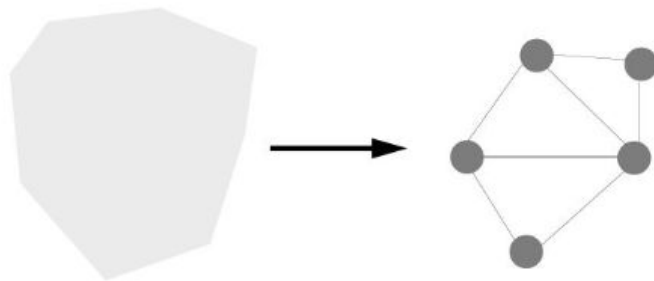


Декомпозиция и абстракция

Декомпозиция и абстракция

От сложного к простому:

- Как нам разделить программу на менее сложные подпрограммы?
- Как выделить значимые характеристики объектов и зачем нам это надо?

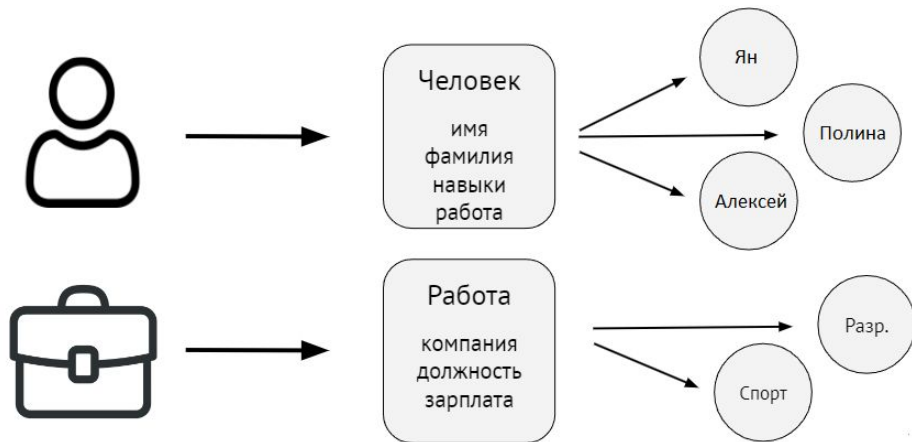


Пример декомпозиции

Декомпозиция и абстракция

Абстракция:

Абстракция - это выделение существенных характеристик объекта, отличающих его от других объектов.



Объекты и Классы

Класс

Объекты и Классы

Класс - способ описания сущности, определяющий состояние и поведение, зависящее от этого состояния, а также правила для взаимодействия с данной сущностью.

`class Person`



```
class Person:
    name = ...
    last_name = ...

    def run(self, time):
        ...
```

Объекты и Классы

Объект

Объекты и Классы

Объект - отдельный представитель класса, имеющий конкретное состояние и поведение, полностью определяемое классом.



```
peter = Person()
```

Атрибуты и методы классов

Атрибут - переменная, которая хранит некоторые данные.

Peter



Peter
Parker
Age 26
Photographer

```
peter = Person()  
peter.name = 'Peter'  
peter.last_name = 'Parker'  
peter.age = 26  
peter.job = 'photographer'
```


Атрибуты и методы классов

Метод - функция, с помощью которой объект может совершать действия и при необходимости изменять свое состояние или состояние других объектов.

Peter

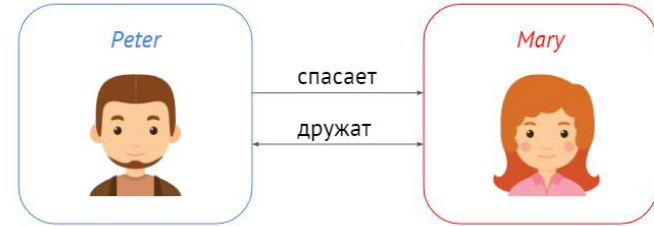


бегает
прыгает
спасает
делает фото

```
peter = Person()  
peter.run(distance=100)  
peter.jump(height=10)  
peter.rescue('Mary Jane')  
peter.photo('Spider Man')
```

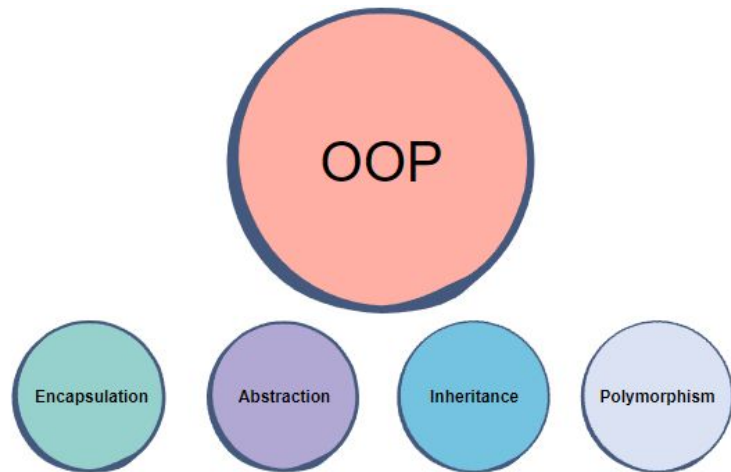
Взаимодействие объектов

Интерфейс - совокупность всех методов, доступных для использования объектами и представителями других объектов.



Принципы ООП

1. Инкапсуляция
2. Наследование
3. Полиморфизм
4. Абстракция



Принципы ООП

Инкапсуляция - это свойство системы, позволяющее объединить данные и методы работающие с ними в классе, и скрыть детали реализации.



Модификаторы доступа

Модификаторы доступа в Python используются для модификации области видимости переменных по умолчанию.

Существуют 3 типа модификаторов доступа в Python ООП:

- `public`
- `_private`
- `_protected`

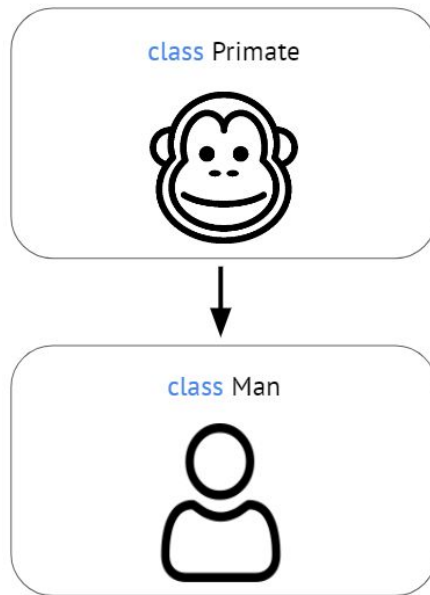
Доступ к переменным с модификаторами публичного доступа открыт из любой точки вне класса, доступ к приватным переменным открыт только внутри самого класса и в случае с защищенными переменными, доступ открыт только внутри класса и дочерних классов.

Но это все работает только на словах.

<https://docs.python.org/3.10/tutorial/classes.html#private-variables-and-class-local-references>

Принципы ООП

Наследование - это свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствованной функциональностью.



```
class Primate:

    def eat(self, food):
        ...

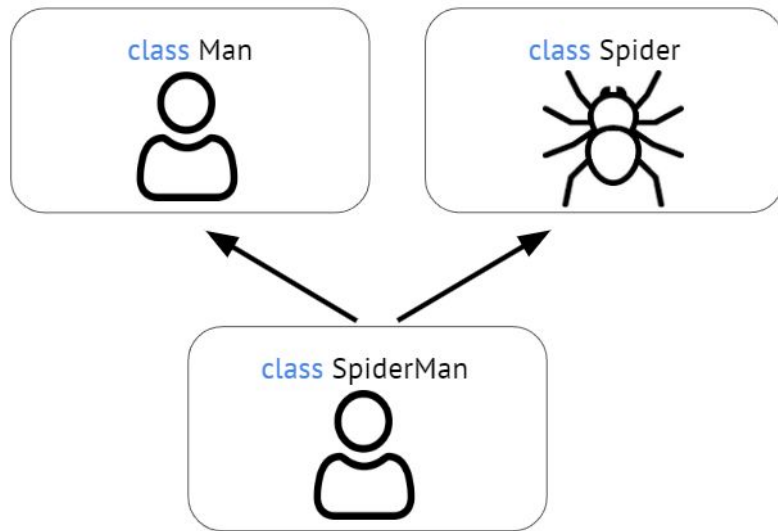
    def run(self, time):
        ...
```

```
class Man(Primate):
    name = ...
    last_name = ...

    def build(self, something):
        ...
```

Принципы ООП

Множественное наследование



MRO

MRO (Method Resolution Order) - порядок, в котором Python ищет метод в иерархии классов.

```
class Primate:
    ...

class Man(Primate):
    ...

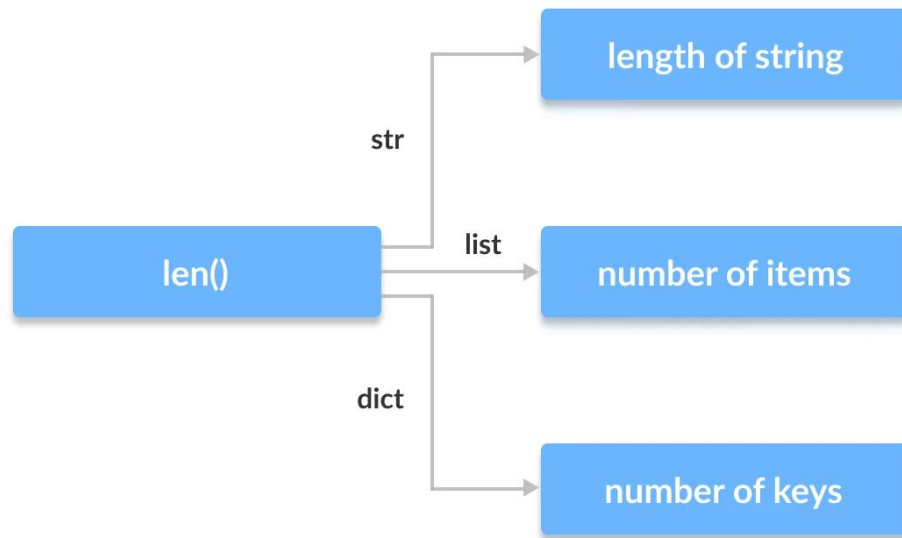
class Spider:
    ...

class SpiderMan(Spider, Man):
    ...

SpiderMan.mro()
```


Принципы ООП

Полиморфизм - это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.



Перегрузка и переопределение методов

Заставляем методы работать по-разному в зависимости от наличия параметров или исходя из того, из какого класса мы их вызываем.

```
class Primate:

    def eat(self, food):
        print(food)

class Man(Primate):

    def eat(self, food,
cooked=False):
        if cooked:
            print('cooked', food)
        else:
            print(food)
```

Магические методы (Dunder methods)

Инициализация и Конструирование:

- `__new__(cls, other)`
- `__init__(self, other)`
- `__del__(self)`

Операторы:

- `__add__(self, other)`
- `__div__(self, other)`
- `__lt__(self, other)`

<https://www.tutorialsteacher.com/python/magic-methods-in-python>

**Погнали в Jupyter Notebook
вспомним базовый синтаксис**

Практика

Напишите функцию, которая определяет попадает ли число в заданный диапазон.

Например:

`check_range(start, end, num)`

`check_range(0, 10, 2)`

`>>> True`

Напишите функцию, которая определяет является ли число совершенным или нет.

Например:

is_perfect(n)

is_perfect(6)

>>> True

Напишите функцию Python для создания и вывода списка, в котором значения являются квадратами чисел от 1 до 50

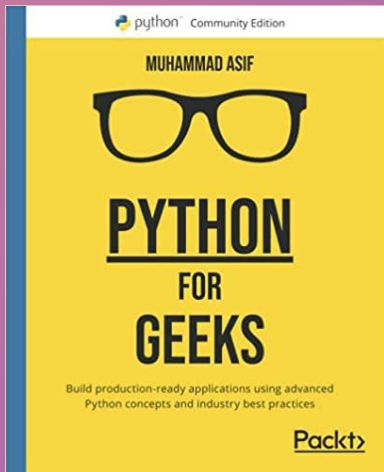
Напишите функцию для создания словаря с уникальными значениями заданного списка в качестве ключей и их частотами в качестве значений.

Подсказка: `from collections import defaultdict`

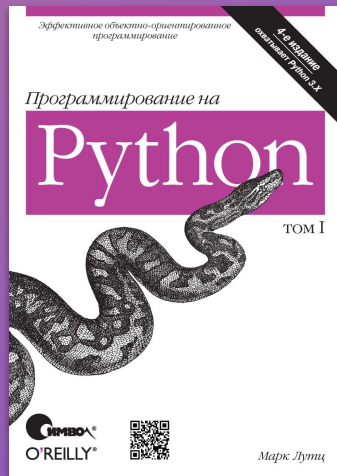
Выводы

- Рассмотрели и повторили основной синтаксис языка программирования Python
- Научились работать в терминале, рассмотрели базовые команды git, научились работать с виртуальным окружением
- Научились работать в Jupyter Notebook и устанавливать необходимые библиотеки для работы с данными

Дополнительная литература



Мухаммед Асиф,
“Python For Geeks”



Марк Лутц,
“Программируем на
Python”
в 2-х томах

Q&A

Полезные материалы:

- **Справочник** [Learnpython.org](https://learnpython.org) даёт исчерпывающее и простое для понимания интро в язык с интерактивными примерами, которые можете запустить прямо в браузере.
- Шпаргалка по Python
- [Git commands that you should now \(EN\)](#)
- Шпаргалка по Git
- Интерактивный тур по Git
- Команды терминала, которые должен знать каждый программист
- Шпаргалка по Conda
- Итераторы, Генераторы, Декораторы
- ООП Python
- [Design Patterns Python](#)

Полезные материалы:

- Введение в командную строку.
- Шпаргалка по командной строке и командам
- Шпаргалка по базовым командам Git
- Работа с ветками в Git
- Трюки и полезные команды для Git
- Визуализация кода Python