

# Системы и технологии интеллектуальной обработки данных

Лектор: Сухорукова Ирина Геннадьевна  
ст. преподаватель кафедры программной инженерии

Контакты: ауд.408 к.1  
[sig@belstu.by](mailto:sig@belstu.by)

Лекции – планируется 2 тематических теста, контроль посещаемости

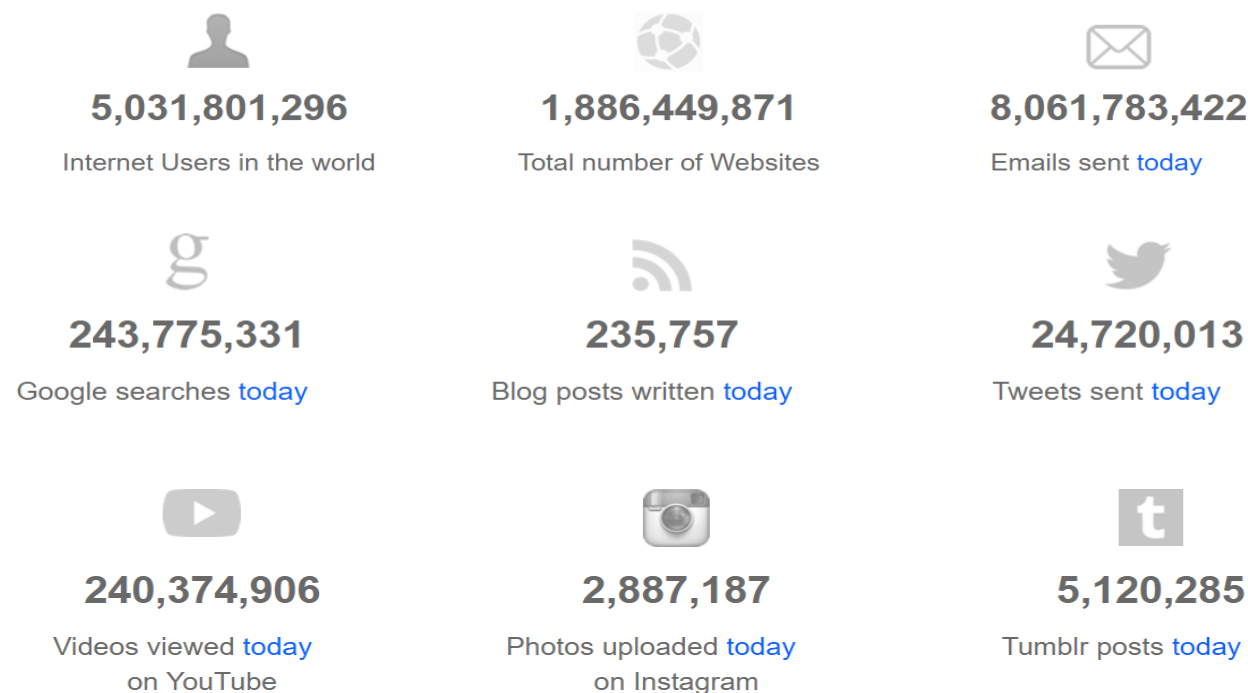
Лабораторные занятия – Python, PyCharm, ( Jupyter Notebook )

Экзамен – залог успеха на экзамене: посещение лекций, успешно написанные тесты, хорошая теория при сдаче лабораторных.

Выступления на лекциях с интересными проектами из рассматриваемой области очень хорошо поощряются при сдаче экзамена и при сдаче лабораторных работ.

В 2020 году общий объём сгенерированных данных **составил 64,2 зеттабайта\***. Сообщается, что для дальнейшего использования было сохранено менее 2% информации. Основная часть данных имела временный характер. (интересно, что в 2016г. на 2020г. **прогнозировали 44 зеттабайта**)

Сайт [internetlivestats.com](https://internetlivestats.com) показывает, что происходит в интернете в режиме реального времени




Из-за огромного количества информации очень малая ее часть будет когда-либо увидена человеческим глазом. **Единственная надежда понять и найти что-то полезное в этом океане информации — широкое применение методов Data Mining.**

\* 1 зеттабайт — это миллиард терабайтов

## Выдача кредита

German credit data set (UCI репозиторий)



1	6	4	12	5	5	3	4	1	67	3	2	1	2	1	0	0	1	0	0	1	0	0	1	1
2	48	2	60	1	3	2	2	1	22	3	1	1	1	1	0	0	1	0	0	1	0	0	1	2
4	12	4	21	1	4	3	3	1	49	3	1	2	1	1	0	0	1	0	0	1	0	1	0	1
1	42	2	79	1	4	3	4	2	45	3	1	2	1	1	0	0	0	0	0	0	0	0	1	1
1	24	3	49	1	3	3	4	4	55	3	2	2	1	1	1	0	1	0	0	0	0	0	1	2
4	36	2	91	3	3	3	4	4	35	3	1	2	2	1	0	0	1	0	0	0	0	1	0	1
4	24	2	28	3	3	3	4	2	55	3	1	1	1	1	0	0	1	0	0	1	0	0	1	1
2	36	2	12	1	3	3	3	3	27	3	1	1	1	1	0	0	1	0	0	0	0	0	0	1
4	12	2	12	1	3	3	3	3	27	3	1	1	1	1	0	0	1	0	0	0	0	0	0	1
2	30	2	12	1	3	3	3	3	27	3	1	1	1	1	0	0	1	0	0	0	0	0	0	2
2	12	2	12	1	3	3	3	3	27	3	1	1	1	1	0	0	1	0	0	0	0	0	0	2
1	48	1	48	1	3	3	3	3	27	3	1	1	1	1	0	0	1	0	0	0	0	0	0	2
2	12	2	12	1	3	3	3	3	27	3	1	1	1	1	0	0	1	0	0	0	0	0	0	1
1	24	1	24	1	3	3	3	3	27	3	1	1	1	1	0	0	1	0	0	0	0	0	0	2
1	15	1	15	1	3	3	3	3	27	3	1	1	1	1	0	0	1	0	0	0	0	0	0	1
1	24	1	24	1	3	3	3	3	27	3	1	1	1	1	0	0	1	0	0	0	0	0	0	2
4	24	4	24	1	3	3	3	3	27	3	1	1	1	1	0	0	1	0	0	0	0	0	0	1
1	30	1	30	1	3	3	3	3	27	3	1	1	1	1	0	0	1	0	0	0	0	0	0	1
2	24	2	24	1	3	3	3	3	27	3	1	1	1	1	0	0	1	0	0	0	0	0	0	2
4	24	4	24	1	3	3	3	3	27	3	1	1	1	1	0	0	1	0	0	0	0	0	0	1
4	9	4	9	1	3	3	3	3	27	3	1	1	1	1	0	0	1	0	0	0	0	0	0	1
1	6	1	6	1	3	3	3	3	27	3	1	1	1	1	0	0	1	0	0	0	0	0	0	1
1	10	1	10	1	3	3	3	3	27	3	1	1	1	1	0	0	1	0	0	0	0	0	0	1
2	12	2	12	1	3	3	3	3	27	3	1	1	1	1	0	0	1	0	0	0	0	0	0	1

**Attribute 1:** Status of existing checking account

1 : ... < 0 DM

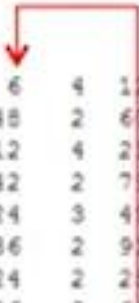
2 : 0 <= ... < 200 DM

3 : ... >= 200 DM / salary assignments for at least 1 year

4 : no checking account

# Выдача кредита

German credit data set (UCI репозиторий)



1	6	4	12	5	5	3	4	1	67	3	2	1	2	1	0	0	1	0	0	1	0	0	1	1
2	48	2	60	1	3	2	2	1	22	3	1	1	1	1	0	0	1	0	0	1	0	0	1	2
4	12	4	21	1	4	3	3	1	49	3	1	2	1	1	0	0	1	0	0	1	0	1	0	1
1	42	2	79	1	4	3	4	2	45	3	1	2	1	1	0	0	0	0	0	0	0	0	1	1
1	24	3	49	1	3	3	4	4	53	3	2	2	1	1	1	0	1	0	0	0	0	0	1	2
4	36	2	91	5	3	3	4	4	35	3	1	2	2	1	0	0	1	0	0	0	0	1	0	1
4	24	2	29	3	5	3	4	2	53	3	1	1	1	1	0	0	1	0	0	1	0	0	1	1
2	36	2	2	1	3	3	3	3	27	3	1	1	1	1	0	1	1	0	1	0	0	0	0	1
4	12	2	2	1	3	3	3	3	27	3	1	1	1	1	0	0	1	0	0	1	0	1	0	1
2	30	4	2	1	3	3	3	3	27	3	1	1	1	1	1	0	1	0	0	1	0	0	0	2
2	12	2	13	1	2	2	1	3	25	3	1	1	1	1	1	0	1	0	1	0	0	0	1	2
1	48	2	43	1	2	2	4	2	24	3	1	1	1	1	0	0	1	0	1	0	0	0	1	2
2	12	2	16	1	3	2	1	3	22	3	1	1	2	1	0	0	1	0	0	1	0	0	1	1
1	24	4	12	1	5	3	4	3	60	3	2	1	1	1	1	0	1	0	0	1	0	1	0	2
1	15	2	14	1	3	2	4	3	28	3	1	1	1	1	1	0	1	0	1	0	0	0	1	1
1	24	2	13	2	3	2	2	3	32	3	1	1	1	1	0	0	1	0	0	1	0	1	0	2
4	24	4	24	5	5	3	4	2	53	3	2	1	1	1	0	0	1	0	0	1	0	0	1	1
1	30	0	81	5	2	3	3	3	25	1	3	1	1	1	0	0	1	0	0	1	0	0	1	1
2	24	2	126	1	5	2	2	4	44	3	1	1	2	1	0	1	1	0	0	0	0	0	0	2
4	24	2	34	3	5	3	2	3	31	3	1	2	2	1	0	0	1	0	0	1	0	0	1	1
4	9	4	21	1	3	3	4	3	48	3	3	1	2	1	1	0	1	0	0	1	0	0	1	1
1	6	2	26	3	3	3	3	1	44	3	1	2	1	1	0	0	1	0	1	0	0	0	1	1
1	10	4	22	1	2	3	3	1	48	3	2	2	1	2	1	0	1	0	1	0	0	1	0	1
2	12	4	18	2	2	3	4	2	44	3	1	1	1	1	0	1	1	0	0	1	0	0	1	1

# Выдача кредита

German credit data set (UCI репозиторий)

1	6	4	12	5	5	3	4	1	67	3	2	1	2	1	0	0	1	0	0	1	0	0	1	0	0	1	1
2	48	2	60	1	3	2	2	1	22	3	1	1	1	1	0	0	1	0	0	1	0	0	1	0	0	1	2
4	12	4	21	1	4	3	3	1	49	3	1	2	1	1	0	0	1	0	0	1	0	1	0	0	1	1	
1	42	2	79	1	4	3	4	2	45	3	1	2	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1
1	24	3	49	1	3	3	4	4	53	3	2	2	1	1	1	0	1	0	0	0	0	0	0	0	0	1	2
4	36	2	91	5	3	3	4	4	35	3	1	2	2	1	0	0	1	0	0	0	0	0	0	0	0	1	1
4	24	2	28	3	3	3	4	2	53	3	1	1	1	1	0	0	1	0	0	1	0	0	0	0	0	1	1
2	36	2	69	1	3	3	2	3	35	3	1	1	2	1	0	1	1	0	1	0	0	0	0	0	0	0	1
4	12	2	31	4	4	1	4	1	61	3	1	1	1	1	0	0	1	0	0	1	0	0	1	0	0	1	1
2	30	4	52	1	1	4	2	3	28	3	2	1	1	1	1	0	1	0	0	1	0	0	0	0	0	0	2
2	12	2	13	1	2	2	1	3	25	3	1	1	1	1	1	0	1	0	1	0	0	0	0	0	1	2	
1	48	2	43	1	2	2	4	2	24	3	1	1	1	1	0	0	1	0	1	0	0	0	0	0	1	2	
2	12	2	16	1	3	2	1	3	22	3	1	1	2	1	0	0	1	0	0	1	0	0	0	0	0	1	1
1	24	4	12	1	3	3	4	3	60	3	2	1	1	1	1	0	1	0	0	1	0	0	0	0	0	1	2
1	15	2	14	1	3	2	4	3	28	3	1	1	1	1	0	0	1	0	0	1	0	0	0	0	0	1	1
1	24	2	13	2	3	2	2	3	32	3	1	1	1	1	0	0	1	0	0	1	0	0	0	0	0	1	2
4	24	4	24	5	3	3	4	2	53	3	2	1	1	1	0	0	1	0	0	1	0	0	0	0	0	1	1
1	30	0	81	5	2	3	3	3	25	1	3	1	1	1	1	0	0	1	0	0	1	0	0	0	0	0	1
2	24	2	126	1	3	2	2	4	44	3	1	1	2	1	0	1	1	0	0	0	0	0	0	0	0	0	2
4	24	2	34	3	3	3	2	3	31	3	1	2	2	1	0	0	1	0	0	1	0	0	0	0	0	1	1
4	9	4	21	1	3	3	4	3	48	3	3	1	2	1	1	0	1	0	0	1	0	0	0	0	0	1	1
1	6	2	26	3	3	3	3	1	44	3	1	2	1	1	0	0	1	0	1	0	0	0	0	0	0	1	1
1	10	4	22	1	2	3	3	1	48	3	2	2	1	1	1	0	1	0	1	0	0	0	0	0	0	1	1

Answer: 1 – Good, 2 - Bad

Answer: 1 – Good, 2 - Bad

# Интеллектуальный анализ данных

В узком смысле это попытка адекватного русского перевода термина **Data Mining**, который ввёл в обиход Григорий Пятецкий-Шапиро в 1992 году. Согласно его определению, *Data Mining — это процесс обнаружения в сырых данных ранее неизвестных, нетривиальных, практически полезных, доступных интерпретации знаний, необходимых для принятия решений в различных сферах человеческой деятельности.*

Дословный перевод «раскопки (или добыча) данных» следует, по всей видимости, считать неудачным.

Английское словосочетание «data mining» не имеет устоявшегося перевода на русский язык, в русском языке как правило используется термин **интеллектуальный анализ данных**. Более полным и точным является словосочетание «обнаружение знаний в базах данных» (англ. **knowledge discovery in databases, KDD**).

**Data mining** —это автоматизированный поиск данных, основанный на анализе огромных массивов информации. За цель берется идентификация тенденций и паттернов, которая при обычном анализе невозможна. Для сегментации данных и оценки вероятности последующих событий используются сложные математические алгоритмы.

В широком смысле это современная концепция анализа данных, предполагающая, что

- ✓ данные могут быть неточными, неполными (содержать пропуски), противоречивыми, разнородными, косвенными, и при этом иметь гигантские объёмы; поэтому понимание данных в конкретных приложениях требует значительных интеллектуальных усилий;
- ✓ сами алгоритмы анализа данных могут обладать «элементами интеллекта», в частности, способностью обучаться по прецедентам, то есть делать общие выводы на основе частных наблюдений; разработка таких алгоритмов также требует значительных интеллектуальных усилий;
- ✓ процессы переработки сырых данных в информацию, а информации в знания уже не могут быть выполнены по старинке вручную, и требуют нетривиальной автоматизации.

*В настоящее время **data mining** является частью большего понятия – **Big data**, которое помимо обработки данных включает в себя их сбор и хранение.*

**Data Mining** — собирательное название, используемое для обозначения **совокупности методов обнаружения в данных ранее неизвестных, нетривиальных, практически полезных и доступных интерпретации знаний, необходимых для принятия решений в различных сферах человеческой деятельности.**

**Знания должны быть новые, ранее неизвестные.** Затраченные усилия на открытие знаний, которые уже известны пользователю, не окупаются. Поэтому ценность представляют именно новые, ранее неизвестные знания.

**Знания должны быть нетривиальны.** Результаты анализа должны отражать неочевидные, неожиданные закономерности в данных, составляющие так называемые скрытые знания. Результаты, которые могли бы быть получены более простыми способами (например, визуальным просмотром), не оправдывают привлечение мощных методов Data Mining.

**Знания должны быть практически полезны.** Найденные знания должны быть применимы, в том числе и на новых данных, с достаточно высокой степенью достоверности. Полезность заключается в том, чтобы эти знания могли принести определенную выгоду при их применении.

**Знания должны быть доступны для понимания человеку.** Найденные закономерности должны быть логически объяснимы, в противном случае существует вероятность, что они являются случайными. Кроме того, обнаруженные знания должны быть представлены в понятном для человека виде.

# Области практического применения Data Mining

Сейчас технология Data Mining используется практически во всех сферах деятельности человека, где накоплены ретроспективные данные. Это бизнес, производство, медицина, наука, Web-направление и др.

БАНКОВСКОЕ ДЕЛО: анализ кредитоспособности клиента, привлечение новых клиентов, мошенничество с карточками, сегментация.

ТЕЛЕКОММУНИКАЦИИ: удержание клиента, выявления определенных групп клиентов, и разработка наборов услуг, наиболее привлекательных именно для них.

МАРКЕТИНГ: В сфере маркетинга Data Mining находит очень широкое применение. Основные вопросы маркетинга "Что продается?", "Как продается?", "Кто является потребителем?"

ИНТЕРНЕТ-ТЕХНОЛОГИИ: формирование рекомендательных систем, планирование маркетинговой политики в соответствии с обнаруженными интересами и потребностями клиентов. Технология Data Mining для электронной коммерции тесно связана с технологией Web Mining.

ТОРГОВЛЯ: анализ рыночных корзин с целью регулирования предложениями, выделение групп потребителей со схожими стереотипами поведения, т.е. сегментирование рынка.

МЕДИЦИНА ...

ФАРМАЦЕВТИКА...

МОЛЕКУЛЯРНАЯ ГЕНЕТИКА и ГЕННАЯ ИНЖЕНЕРИЯ...

.....



# WEB

Data Mining используется каждый день тысячи раз в секунду — каждый раз, когда кто-то использует Google или другие поисковые системы (search engines) на просторах Интернета.

**Web Content Mining** подразумевает автоматический поиск и извлечение качественной информации из разнообразных источников Интернета, перегруженных "информационным шумом". Здесь также идет речь о различных средствах кластеризации и аннотирования документов.

**Web Usage Mining** подразумевает обнаружение закономерностей в действиях пользователя Web-узла или их группы. Анализируется следующая информация: · какие страницы просматривал пользователь; · какова последовательность просмотра страниц.

**Text Mining** охватывает новые методы для выполнения семантического анализа текстов, информационного поиска и управления. Программы, реализующие эту задачу, должны некоторым образом оперировать естественным человеческим языком и при этом понимать семантику анализируемого текста.

Технология **Call Mining** объединяет в себя распознавание речи, ее анализ и Data Mining. Ее цель - упрощение поиска в аудио-архивах, содержащих записи переговоров между операторами и клиентами. При помощи этой технологии операторы могут обнаруживать недостатки в системе обслуживания клиентов, находить возможности увеличения продаж, а также выявлять тенденции в обращениях клиентов.

**Data Mining** носит **мультидисциплинарный характер**, поскольку включает в себя элементы численных методов, математической статистики и теории вероятностей, теории информации и математической логики, искусственного интеллекта и машинного обучения.



Фундаментально data mining основывается на 3-х понятиях:

- ✓ Математическая статистика – является основой большинства технологий, используемых для data mining, например, кластерный анализ, регрессионный анализ, дискриминирующий анализ и пр.;
- ✓ Искусственный интеллект – воспроизведение нейронной сети мышления человека в цифровом виде;
- ✓ Машинное обучение – совокупность статистики и искусственного интеллекта, способствующая пониманию компьютерами данных, которые они обрабатывают для выбора наиболее подходящего метода или методов анализа.

# Задачи Data Mining

Методы Data Mining помогают решить многие задачи, с которыми сталкивается аналитик. Из них основными являются: **классификация, регрессия, поиск ассоциативных правил и кластеризация.**

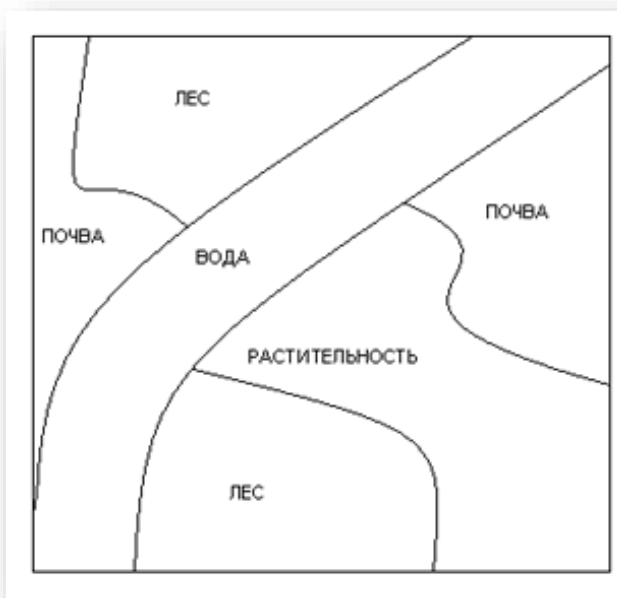
- ✓ **Задача классификации** сводится к определению класса объекта по его характеристикам. Необходимо заметить, что в этой задаче множество классов, к которым может быть отнесен объект, известно заранее. *Примеры: есть ли на фотографии кот, болен ли пациент раком, обнаружение спама*
- ✓ **Задача регрессии** подобно задаче классификации позволяет определить по известным характеристикам объекта значение некоторого его параметра. В отличие от задачи классификации значением параметра является не конечное множество классов, а множество действительных чисел. *Примеры: цена квартиры, стоимость ценной бумаги по прошествии полугода, ожидаемый доход магазина на следующий месяц*
- ✓ **При поиске ассоциативных правил** целью является нахождение частых зависимостей (или ассоциаций) между объектами или событиями. Найденные зависимости представляются в виде правил и могут быть использованы как для лучшего понимания природы анализируемых данных, так и для предсказания появления событий. *Пример: анализ продуктовой корзины*
- ✓ **Задача кластеризации** заключается в поиске независимых групп (кластеров) и их характеристик во всем множестве анализируемых данных. Решение этой задачи помогает лучше понять данные. Кроме того, группировка однородных объектов позволяет сократить их число, а следовательно, и облегчить анализ. *Пример: разделение всех клиентов мобильного оператора по уровню платёжеспособности*

Фактически задачи Data Mining являются элементами, из которых можно «собрать» решение большинства реальных задач.

1

К какой задаче относятся рисунки?

- поиск ассоциативных правил
- классификация,
- регрессия,
- или кластеризация?

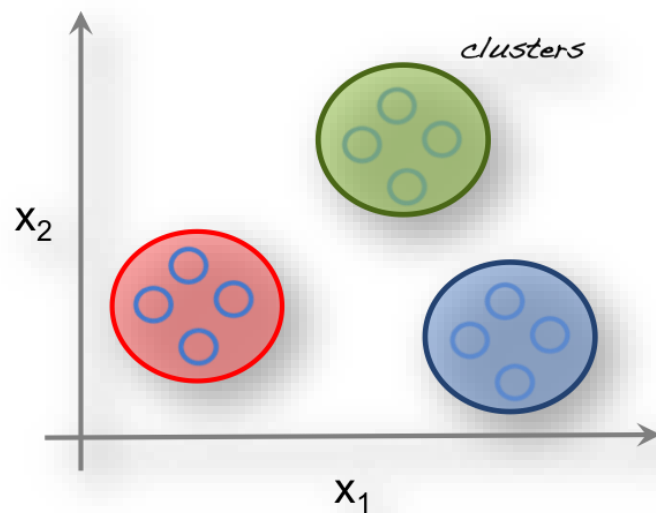


2

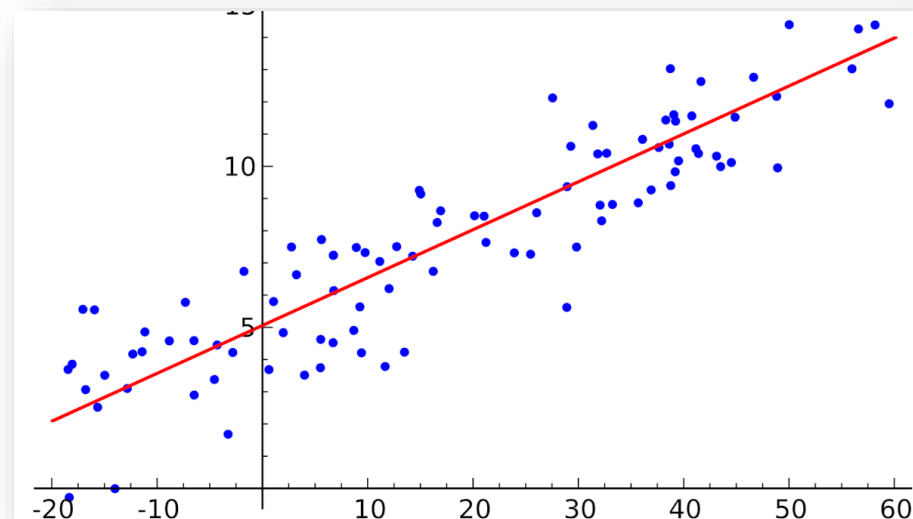
TID	Приобретенные покупки
100	Хлеб, молоко, печенье
200	Молоко, сметана
300	Молоко, хлеб, сметана, печенье
400	Колбаса, сметана
500	Хлеб, молоко, печенье, сметана

Вывод: Если клиент купил хлеб, то он купит и молоко с вероятностью 100%.

3



4

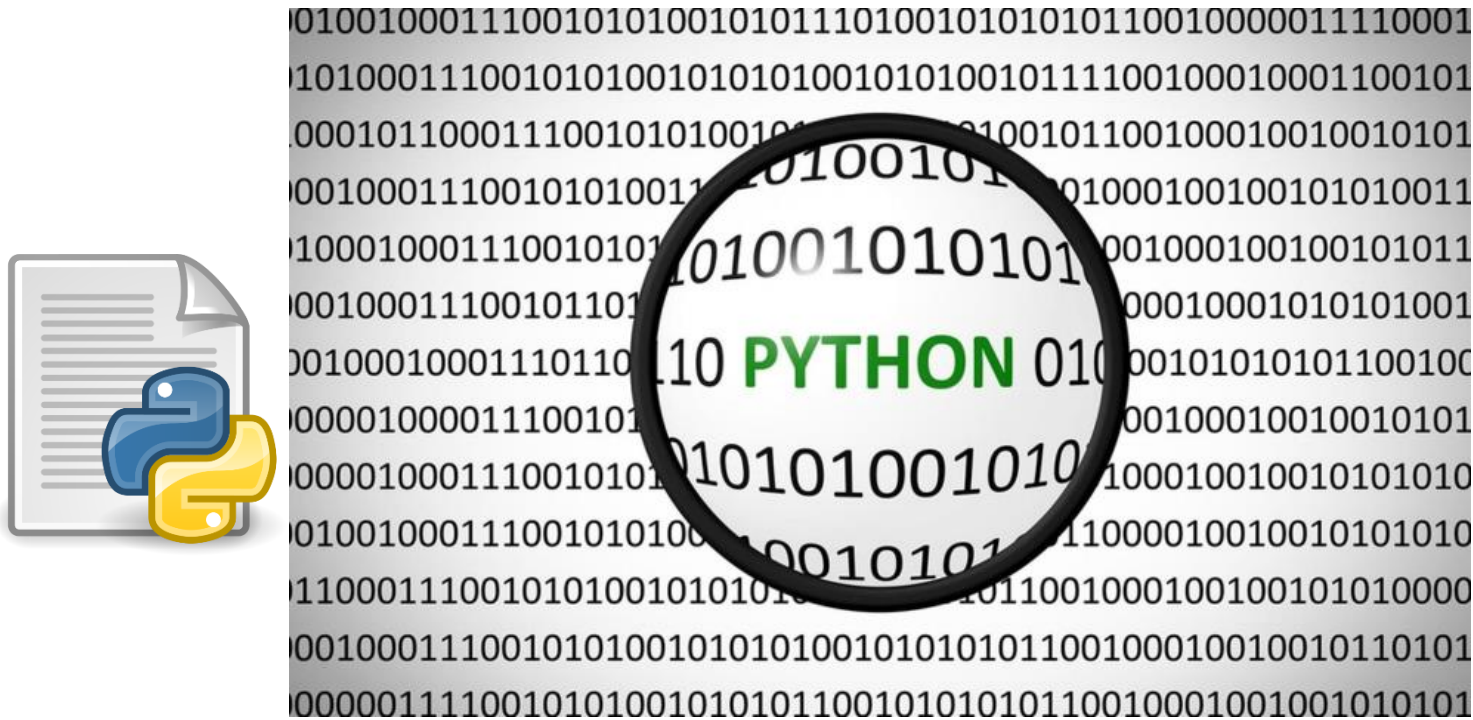


Задачи Data Mining *по назначению* делятся на **описательные** и **предсказательные**.

- ✓ **Описательные** (descriptive) задачи уделяют внимание улучшению понимания анализируемых данных, обнаружение закономерностей. К такому виду задач относятся кластеризация и поиск ассоциативных правил.
- ✓ При решении **предсказательных** (predictive) задач на основании набора данных с известными результатами **строится модель**. Затем она используется для предсказания результатов на основании новых наборов данных. К данному виду задач относят задачи классификации и регрессии.

По *способам решения* задачи разделяют на **обучение с учителем** (supervised learning) и **обучение без учителя** (unsupervised learning). Такое название произошло от термина Machine Learning (машинное обучение), часто используемого в англоязычной литературе и обозначающего все технологии Data Mining.

- ✓ В случае **обучения с учителем** строится модель анализируемых данных — классификатор. Затем классификатор подвергается обучению. К этому типу задач относят задачи классификации и регрессии.
- ✓ **Обучение без учителя** объединяет задачи, выявляющие описательные модели, например закономерности в покупках. Очевидно, что если эти закономерности есть, то модель должна их представить и неуместно говорить об ее обучении. Достоинством таких задач является возможность их решения без каких-либо предварительных знаний об анализируемых данных. К этим задачам относятся кластеризация и поиск ассоциативных правил.



Python не задумывался создателями как язык для анализа данных. Однако сегодня это один из самых лучших языков для статистики, машинного обучения, прогнозной аналитики, а также стандартных задач по обработке данных.

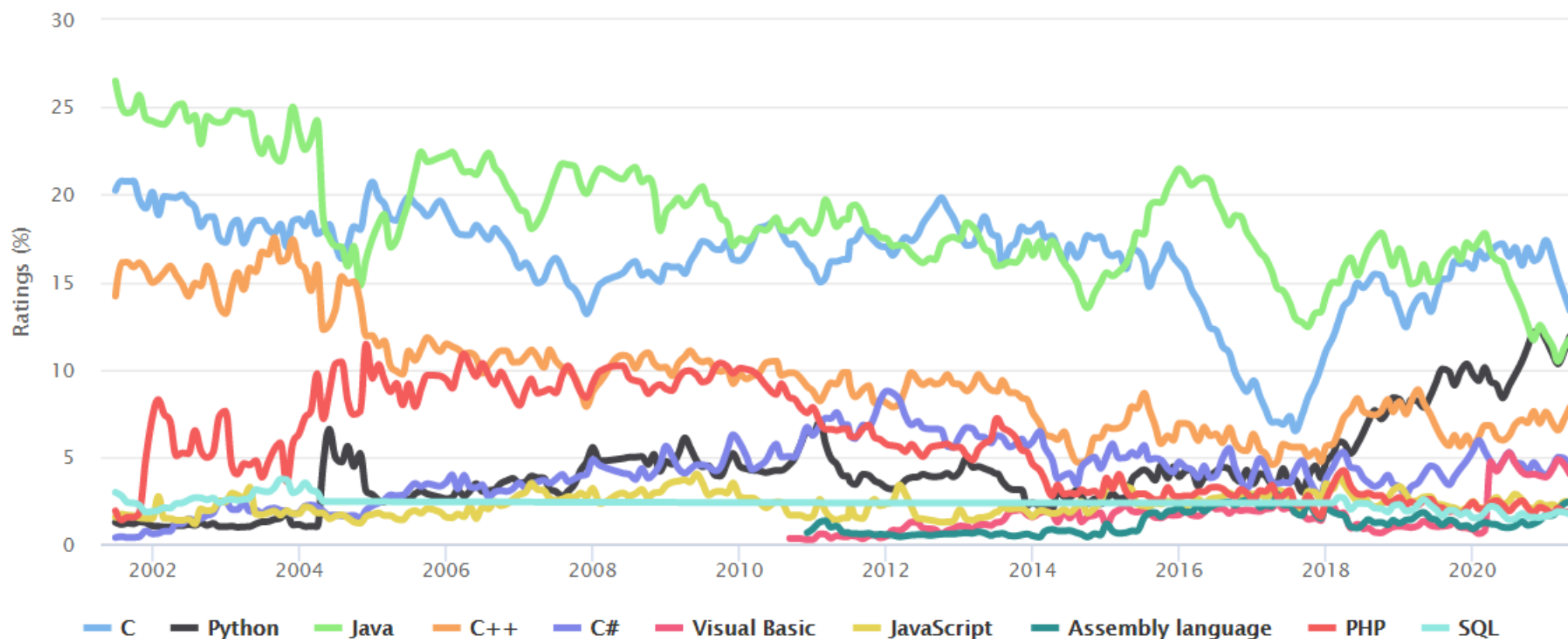
Python — язык с открытым кодом, и специалисты по data science стали создавать инструменты, чтобы более эффективно выполнять свои задачи.

Индекс TIOBE — индекс, оценивающий популярность языков программирования, на основе подсчёта результатов поисковых запросов, содержащих название языка.

May 2021	May 2020	Change	Programming Language	Ratings	Change
1	1		C	13.38%	-3.68%
2	3	▲	Python	11.87%	+2.75%
3	2	▼	Java	11.74%	-4.54%
4	4		C++	7.81%	+1.69%
5	5		C#	4.41%	+0.12%
6	6		Visual Basic	4.02%	-0.16%
7	7		JavaScript	2.45%	-0.23%

## TIOBE Programming Community Index

Source: [www.tiobe.com](http://www.tiobe.com)



### Язык года

- 2020 Python
- 2019 C
- 2018 Python
- 2017 C



В современном мире активно используются десятки языков программирования, и их количество только растёт. В числе самых популярных: *C* и *C++*, *C#* и *Go*, *Java* и *JavaScript*, *Python* и *Ruby*. У каждого из них есть свои особенности и преимущества.

## Какие преимущества есть у Python

Python невероятно эффективен: программы, написанные на нем, делают больше, чем многие на других языках и в меньшем объеме кода.

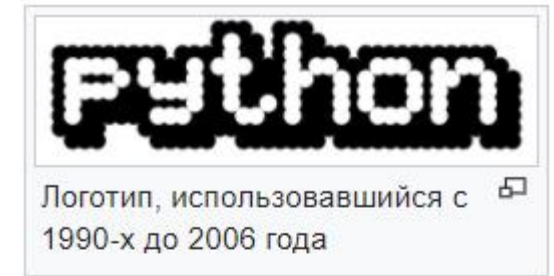
### Его просто учить

Главной целью основателя *Python*, Гвидо ван Россума, было создать простой и понятный широкому кругу людей язык программирования. Изучение любого языка требует усидчивости и дисциплины. Но *Python* в этом смысле считается одним из самых комфортных, особенно для новичков. Простой синтаксис позволяет легко учиться и читать код.

### Он очень распространенный

*Python* универсален благодаря богатой стандартной библиотеке (набору функций), поэтому его применяют в самых разных областях:

- веб-разработке;
- machine Learning и AI;
- Big Data;
- разработке игр.





# Для чего используется Python



Web разработка



Desktop  
приложения



Анализ данных  
и визуализация



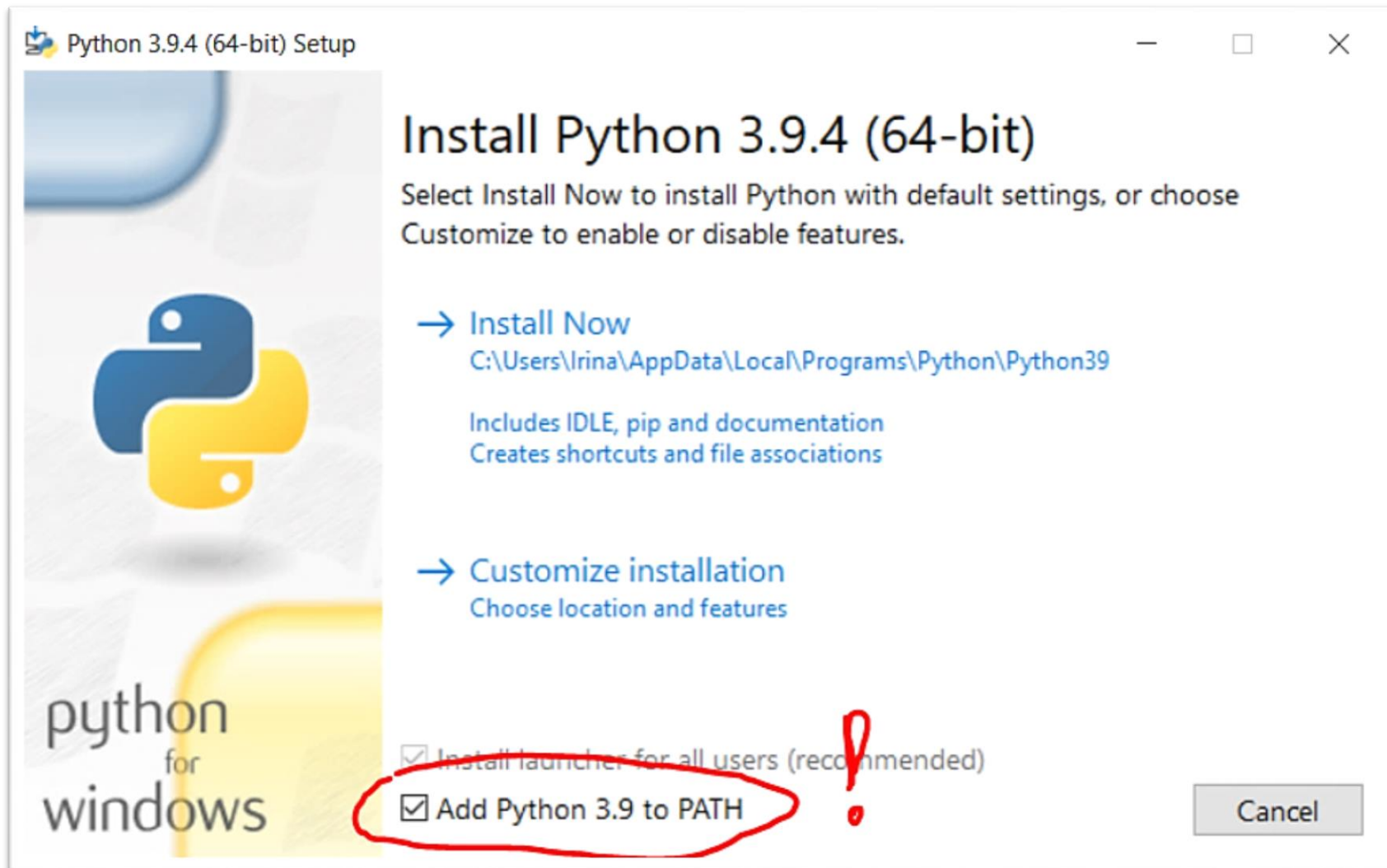
Разработка игр



Data Science



Написание  
скриптов



## Запуск кода Python в интерактивном режиме в командной строке

```
Администратор: C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.19042.928]
(с) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\Irina>python
```

```
Администратор: C:\WINDOWS\system32\cmd.exe - python
Microsoft Windows [Version 10.0.19042.928]
(с) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\Irina>python
Python 3.9.4 (tags/v3.9.4:1f2e308, Apr 6 2021, 13:40:21) [MSC v.1928 64 bi
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

```
D:\IBA>python p1.py
Hello, World!
Hello, Python!
Hello, everybody!
```

```
>>> print("Hello World!")
Hello World!
>>>
```



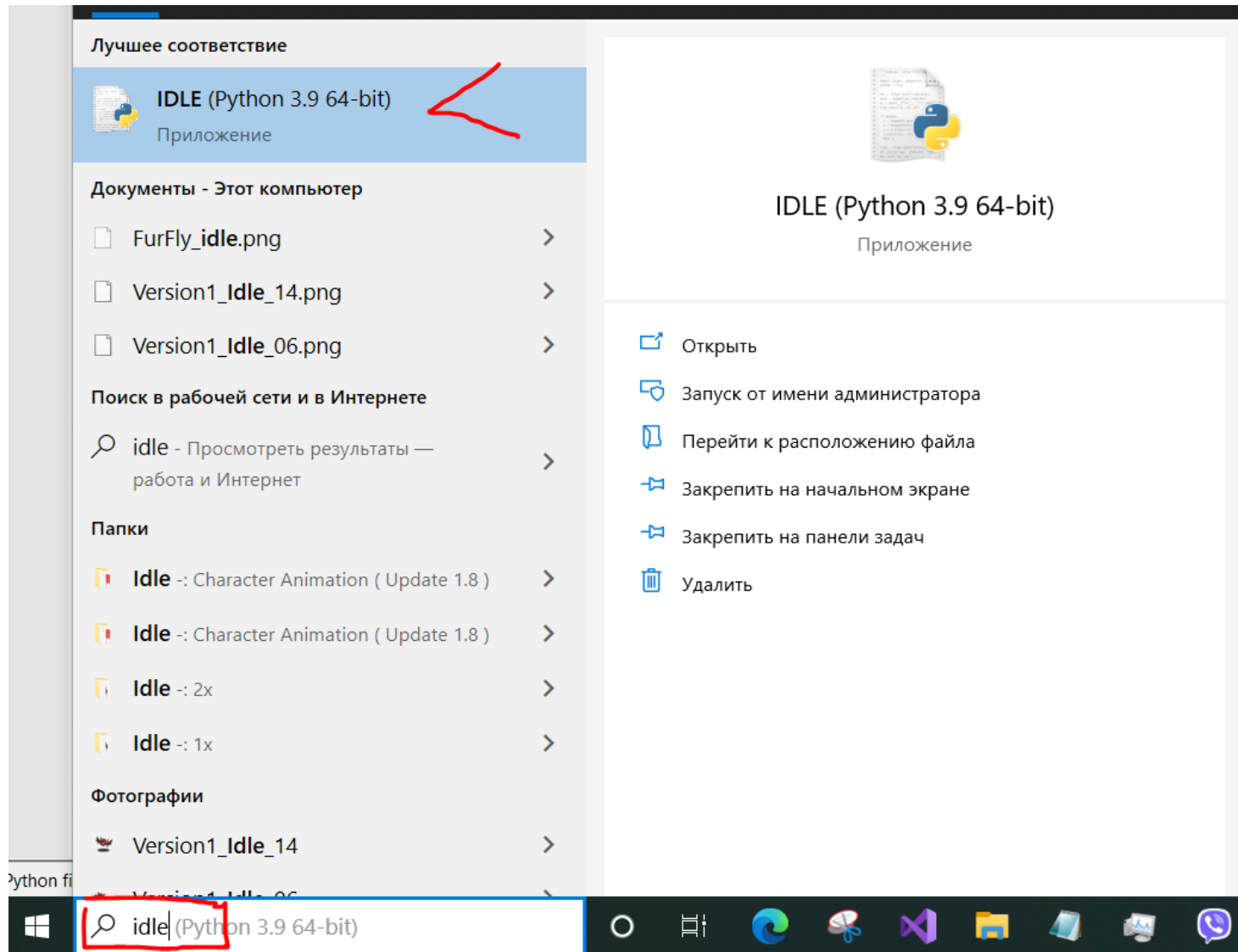
Python из командной строки можно использовать и в качестве калькулятора, выполняя простые вычисления.

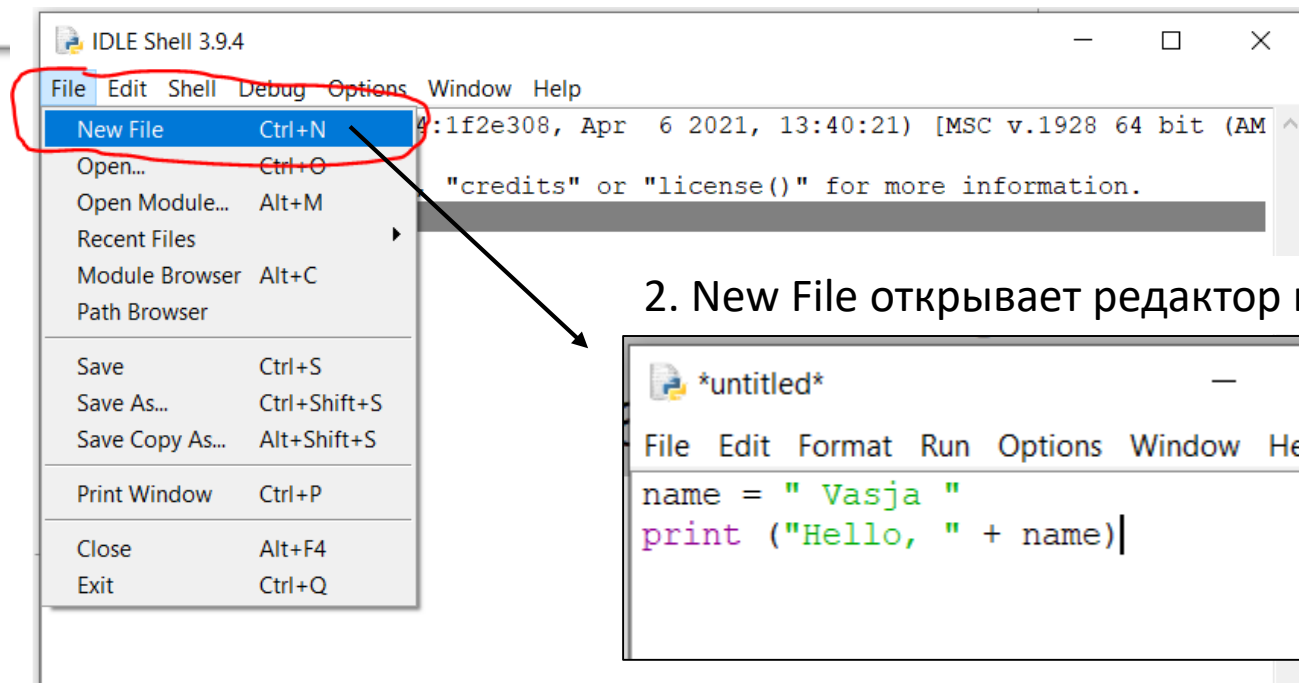
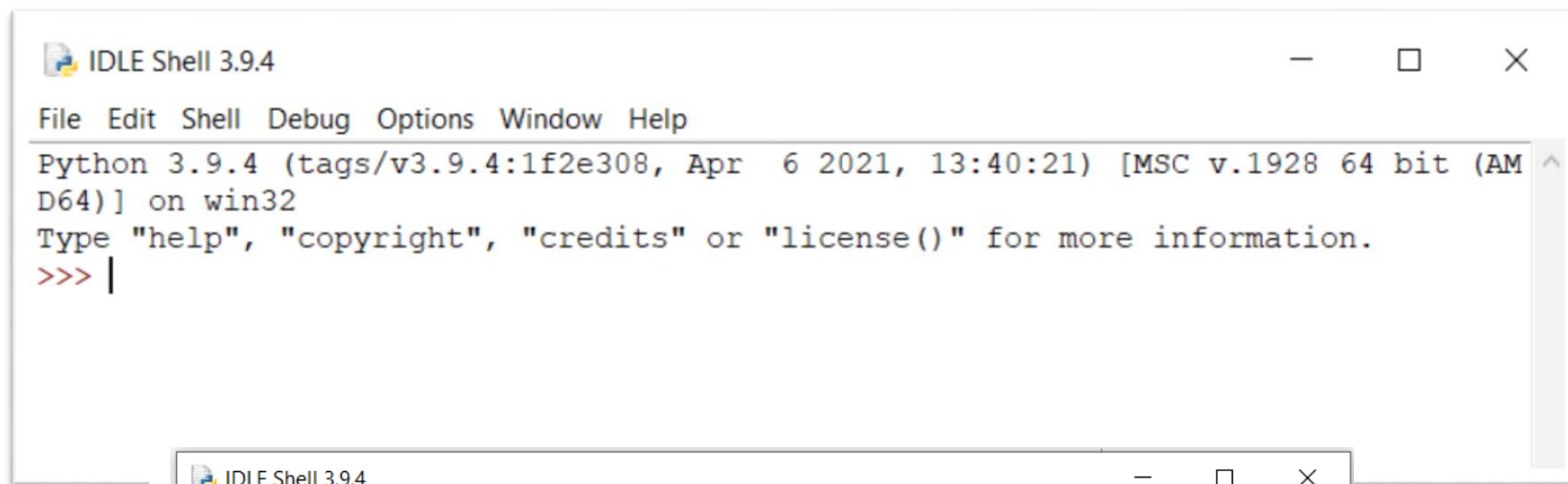
```
C:\Users\Irina>python
Python 3.9.4 (tags/v3.9.4:1f2e308, Apr
Type "help", "copyright", "credits" or
>>> 3+4
7
>>> 4+6*(12+34)
280
>>>
```

$x + y$	Сложение
$x - y$	Вычитание
$x * y$	Умножение
$x / y$	Деление
$x // y$	Получение целой части от деления
$x \% y$	Остаток от деления
$-x$	Смена знака числа
$\text{abs}(x)$	Модуль числа
$x ** y$	Возведение в степень

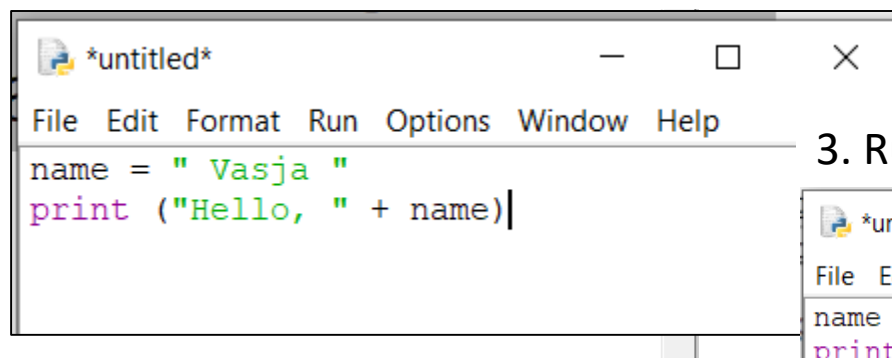
**IDLE** (*Integrated Development and Learning Environment*) — это **интегрированная среда разработки и обучения** на языке Python. Поставляется вместе с Python.

*Официально — искажение **IDE**, но на самом деле названа в честь Эрика Айбла (англ. Eric Idle) из Монти Пайтон.*

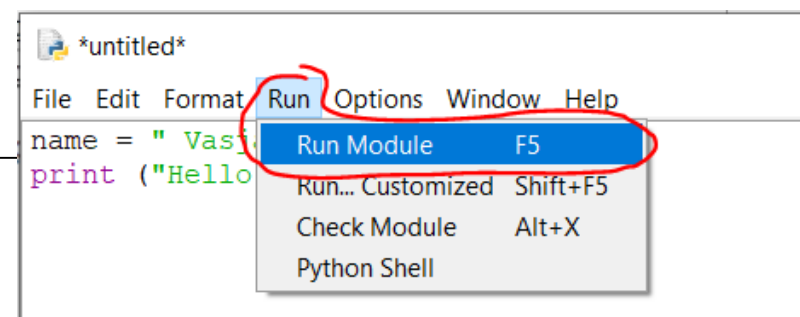




2. New File открывает редактор кода



3. Run – запускает на выполнение



## Преобразование типов

Выражение	Результат выполнения	
<code>int ('56')</code>	56	Строка преобразуется в целое число
<code>int (4.03)</code>	4	Вещественное число преобразуется в целое число
<code>int ("comp 486")</code>	О ш и б к а	Символы алфавита нельзя преобразовать в число
<code>str (56)</code>	'56'	Целое число преобразуется в строку
<code>str (4.03)</code>	'4.03'	Вещественное число преобразуется в строку
<code>float (56)</code>	56.0	Целое число преобразуется в вещественное число
<code>float ("56")</code>	56.0	Строка преобразуется в вещественное число

## Функция округления `round()`

Запись в интерпретаторе

```
>>> round(5.76543, 2)  
5.77
```

```
>>> round(5.76543)  
6
```

```
>>> round(5.63557/3, 3)  
1.879
```

Запись в коде

```
x = round(5.76543, 2)  
print(x)
```

```
x=5.76543  
print(round(x,2))
```

Результат

5.77

5.77

# Работа со строками в Python

Строки можно *индексировать* ( *индексировать* ), причем первый символ имеет индекс 0. Отдельного типа символа не существует; символ - это просто строка единичного размера. Индексы также могут быть отрицательными числами, чтобы начать отсчет справа:

```
>>> word = 'Python'
>>> word[0]  # character in position 0
'P'
>>> word[5]  # character in position 5
'n'
```

```
>>> word[-1]  # last character
'n'
>>> word[-2]  # second-last character
'o'
```

Помимо индексации, также поддерживается нарезка . В то время как индексация используется для получения отдельных символов, нарезка позволяет получить подстроку:

```
>>> word[0:2]
'Py'
>>> word[2:5]
'tho'
```

```
>>> word[:2]
'Py'
>>> word[4:]
'on'
```

```
>>> word[:2] + word[2:]
'Python'
>>> word[:4] + word[4:]
'Python'
```

Строки Python нельзя изменить - они неизменяемы.

Для изменения строки нужно создать новую.

```
>>> word[0] = 'J'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

```
>>> 'J' + word[1:]
'Jython'
>>> word[:2] + 'py'
'Pypy'
```



# Работа со строками в Python

Функция **len()** возвращает длину строки:

```
a = "Hello, World!"  
print(len(a))
```

можно писать и в интерактивном режиме:

```
>>> s = 'supercalifragilisticexpialidocious'  
>>> len(s)  
34
```

Чтобы проверить, присутствует ли в строке определенная фраза или символ, мы можем использовать ключевое слово **in**.

```
txt = "The best things in life are free!"  
print("free" in txt)
```

Метод **upper()** возвращает строку в верхнем регистре:

Метод **lower()** возвращает строку в нижнем регистре.

```
a = "Hello, World!"  
print(a.upper())
```

Метод **capitalize()** преобразует в верхний регистр первую букву в строке:

```
txt = "hello, world."  
x = txt.capitalize()  
print (x)
```

## Конкатенация

```
a = "Hello"  
b = "World"  
c = a + b  
print(c)
```

Method	Description
<u>capitalize()</u>	Converts the first character to upper case
<u>casefold()</u>	Converts string into lower case
<u>center()</u>	Returns a centered string
<u>count()</u>	Returns the number of times a specified value occurs in a string
<u>encode()</u>	Returns an encoded version of the string
<u>endswith()</u>	Returns true if the string ends with the specified value
<u>expandtabs()</u>	Sets the tab size of the string
<u>find()</u>	Searches the string for a specified value and returns the position of where it was found
<u>format()</u>	Formats specified values in a string
<u>format_map()</u>	Formats specified values in a string
<u>index()</u>	Searches the string for a specified value and returns the position of where it was found
<u>isalnum()</u>	Returns True if all characters in the string are alphanumeric
<u>isalpha()</u>	Returns True if all characters in the string are in the alphabet
<u>isdecimal()</u>	Returns True if all characters in the string are decimals
<u>isdigit()</u>	Returns True if all characters in the string are digits
<u>isidentifier()</u>	Returns True if the string is an identifier
<u>islower()</u>	Returns True if all characters in the string are lower case

Существует  
большое  
количество  
методов для  
работы со  
строками,  
смотрите  
документацию

# Операторы сравнения

Оператор	Значение	Выражение
<code>==</code>	Равно	<code>A==B</code>
<code>!=</code>	Не равно	<code>A!=B</code>
<code>&gt;</code>	Больше	<code>A&gt;B</code>
<code>&lt;</code>	Меньше	<code>A&lt;B</code>
<code>&gt;=</code>	Больше или равно	<code>A&gt;=B</code>
<code>&lt;=</code>	Меньше или равно	<code>A&lt;=B</code>

Примеры:

`x = 12 - 5` *# это не логическая операция, а операция присваивания переменной x результата выражения 12 — 5*

`x == 4` *# x равен 4*

`x == 7` *# x равен 7*

`x != 7` *# x не равен 7*

`x != 4` *# x не равен 4*

`x > 5` *# x больше 5*

`x < 5` *# x меньше 5*

`x >= 6` *# x больше или равен 6*

`x <= 6` *# x меньше или равен 6*

Примеры кода:

```
x = 5
y = 3

print(x == y)

# returns False
```

```
x = 5
y = 3

print(x != y)

# returns True
```

## Условия: if, else, elif. Блоки, отступы

Ключевое слово **if** – если, условный оператор.

```
a = 33
b = 200

if b > a:
    print("b больше чем a")
```

```
if b > a:
    print("b is
```

```
a = 33
b = 200

if b > a:
    c = b // a
    print(c)

print("The end")
```

Ключевое слово **elif** - это способ Python сказать: «Если предыдущие условия не были истинными, попробуйте это условие».

```
a = 33
b = 33
if b > a:
    print("b больше чем a")
elif a == b:
    print("a и b равны")
```

Ключевое слово **else** перехватывает все, что не отловлено предыдущими условиями.

```
a = 200
b = 33
if b > a:
    print("b больше чем a")
elif a == b:
    print("a и b равны")
else:
    print("a больше чем b")
```

```
a = 200
b = 33
if b > a:
    print("b больше чем a")
else:
    print("b не больше чем a")
```

Если у вас есть только один оператор для выполнения, то можно записывать в одну строку

```
if a > b: print("a больше чем b")
```

```
a = 2  
b = 330  
print("A") if a > b else print("B")
```

Логический оператор **and** (И):

```
a = 200  
b = 33  
c = 500  
if a > b and c > a:  
    print("Оба условия верны")
```

Логический оператор **or** (ИЛИ) :

```
a = 200  
b = 33  
c = 500  
if a > b or a > c:  
    print("Одно из условий верно")
```

# Математические функции

Python имеет набор встроенных математических функций, включая обширный математический модуль, который позволяет выполнять математические задачи с числами.

```
x = min(5, 10, 25)
y = max(5, 10, 25)

print(x)
print(y)
```

```
x = abs(-7.25)

print(x)
```

Python также имеет встроенный модуль **math**, который расширяет список математических функций.

```
import math

x = math.sqrt(64)
print(x)
```

```
import math

x = math.pi
print(x)
```

```
x=3.14
import math
print(math.sin(x))
```

## Math Methods

### Method

[math.acos\(\)](#)

[math.acosh\(\)](#)

[math.asin\(\)](#)

[math.asinh\(\)](#)

[math.atan\(\)](#)

[math.atan2\(\)](#)

[math.atanh\(\)](#)

[math.ceil\(\)](#)

[math.cos\(\)](#)

[math.cosh\(\)](#)

[math.degrees\(\)](#)

[math.dist\(\)](#)

[math.erf\(\)](#)

[math.erfc\(\)](#)

[math.exp\(\)](#)

[math.expm1\(\)](#)

[math.fabs\(\)](#)

[math.factorial\(\)](#)

# Циклический оператор **while**

Цикл **while** позволяет выполнять набор операторов, пока выполняется некоторое условие.

```
i = 1           # присваиваем i начальное значение
while i < 6:     # пока i < 6 (проверем условие)
    print(i)     # печать i (если условие соблюдено)
    i += 1       # увеличиваем i
```

Результат

1  
2  
3  
4  
5

Строки, которые  
относятся к циклу  
имеют отступ

```
i = 1
while i < 6:
    print(i)
    i += 1
```

```
i = 1
while i < 6:
    i += 1
    print(i)
```

Что  
напечатает  
этот код?

## Оператор **break**

```
i = 1
while i < 6:
    print(i)
    if i == 3:
        break
    i += 1
```

Результат

Выход из цикла,  
когда i будет  
равно 3:

1  
2  
3

## Оператор **continue**

```
i = 0
while i < 6:
    i += 1
    if i == 3:
        continue
    print(i)
```

Результат

1  
2  
4  
5  
6

# Циклический оператор for

Цикл **for** в Python используется для перебора некоторой последовательности, например строки, списка, кортежа, словаря.

**Строки** являются перечисляемыми объектами и содержат последовательность символов:

```
for x in "banana":  
    print(x)
```

Результат

b  
a  
n  
a  
n  
a

## break

```
for x in "crocodile":  
    if x == "d":  
        break  
    print(x)
```

## continue

```
for x in "banana":  
    if x == "a":  
        continue  
    print(x)
```

Функция **range ()** возвращает последовательность чисел.

от 0 до 6 с шагом 1

```
for x in range(6):  
    print(x)
```

0 1 2 3 4 5

от 2 до 6 с шагом 1

```
for x in range(2, 6):  
    print(x)
```

2 3 4 5

от 1 до 10 с шагом 2

```
for x in range(1, 10, 2):  
    print(x)
```

1 3 5 7 9



# Структуры данных в Python

# Списки

**Список** — это структура данных, которая может хранить разные типы данных.

Списки используются для хранения нескольких элементов в одной переменной.

Элементы списка проиндексированы, первый элемент имеет индекс [0], второй элемент - индекс [1] и т. д.

Списки являются изменяемым типом, что значит, что можно добавлять или удалять их элементы.

Элементы списка допускают повторяющиеся значения.

## Рекомендации по работе со списками

- Списки создаются с помощью **квадратных скобок** [ ].
- Элементы списка **нужно** разделять запятыми.
- Правила синтаксиса, характерные для определенных типов данных, нужно соблюдать внутри списка. Так, если у строки должны быть кавычки, то их нужно использовать и внутри списка.

```
fruits_list = ["apple", "banana", "cherry"]  
print(fruits_list)
```

```
===== RESTART: D:/IBA/les3-list.py =====  
['apple', 'banana', 'cherry']  
>>>
```

```
list_datatypes = ["апельсин", 23, 51.0, False, "False", "22"]  
print(list_datatypes)
```

```
===== RESTART: D:/IBA/les3-list.py =====  
['апельсин', 23, 51.0, False, 'False', '22']  
>>> |
```

# Списки

Можно получить доступ к элементам списка, указав номер индекса:

```
thislist = ["apple", "banana", "cherry"]  
print(thislist[1])
```

Негативная индексация. Пример печатает последний элемент списка:

```
thislist = ["apple", "banana", "cherry"]  
print(thislist[-1])
```

Можно проверить наличие элемента в списке ключевым словом in:

```
thislist = ["apple", "banana", "cherry"]  
if "apple" in thislist:  
    print("Yes, 'apple' is in the fruits list")
```

При указании диапазона возвращаемое значение будет новым списком с указанными элементами. В примере видно, что при указании диапазона [2:5] индекс 2 включается, а индекс 5 – не включается:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]  
print(thislist[2:5])
```



```
['cherry', 'orange', 'kiwi']  
>>> |
```

# Методы списков

**.append()** —используется для добавления элементов в конец списка.

```
>>> p_datatypes = ["Python", "апельсин"]
>>> p_datatypes.append("BMW")
>>> print(p_datatypes)
["Python", "апельсин", "BMW"]
```

**.insert()** —используется для вставки элемента в список по индексу.

insert() принимает два аргумента: индекс, куда нужно вставить новый элемент, и сам элемент.

```
>>> p_datatypes = ["Python", "апельсин"]
>>> p_datatypes.insert(1, "BMW")
>>> print(p_datatypes)
["Python", "BMW", "апельсин"]
```

**.index()** помогает определить индекс элемента.

```
>>> lst = [1, 33, 5, 55, 1001]
>>> a = lst.index(55)
>>> print(a)
3
```

**.remove()** удаляет конкретный элемент списка.

```
>>> lucky_numbers = [5, 55, 4, 3, 101, 42]
>>> lucky_numbers.remove(101)
>>> print(lucky_numbers)
[5, 55, 4, 3, 42]
```

**.clear()** удаляет все элементы списка.

```
lucky_numbers.clear()
```

## Методы списков

**.count()** используется, чтобы подсчитать, как часто конкретный элемент встречается в списке.

```
>>> lucky_numbers = [5, 55, 4, 3, 101, 42]
>>> lucky_numbers.count(4)
```

1

```
>>> thislist = ["apple", "banana", "cherry", "kiwi", "orange", "kiwi"]
>>> thislist.count("kiwi")
```

2

**.reverse()** разворачивает порядок элементов в списке.

```
>>> lucky_numbers = [5, 55, 4, 3, 101, 42]
>>> lucky_numbers.reverse()
>>> print(lucky_numbers)
[42, 101, 3, 4, 55, 5]
```

**.pop()** извлекает последний элемент из списка.

```
>>> thislist
['apple', 'banana', 'cherry', 'kiwi']
>>> thislist.pop()
'kiwi'
>>> thislist.pop()
'cherry'
>>> thislist
['apple', 'banana']
>>>
```

**.sort()** сортирует список.

```
>>> lucky_numbers = [5, 55, 4, 3, 101, 42]
>>> lucky_numbers.sort()
>>> lucky_numbers
[3, 4, 5, 42, 55, 101]
```

```
>>> thislist = [ "banana", "cherry", "apple", "orange", "kiwi"]
>>> thislist.sort()
>>> thislist
['apple', 'banana', 'cherry', 'kiwi', 'orange']
```

## Функции списков:

```
>>> lucky_numbers = [5, 55, 4, 3, 101, 42]
>>> min(lucky_numbers)
3
>>> max(lucky_numbers)
101
>>> sum(lucky_numbers)
210
>>> .
```

**List Comprehension** трудно перевести правильно на русский, но так как он генерирует новый список, можно называть его просто генератором списков.

```
>>> squares = []
>>> for x in range(10):
...     squares.append(x**2)
...
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```



**List Comprehension** обеспечивает краткий способ создания списков.

```
squares = [x**2 for x in range(10)]
```

Так же как и в обычных циклах в генераторах можно применять условия:

```
>>> odds = [x for x in range(10) if x % 2 != 0]
# [1, 3, 5, 7, 9]
```

# Кортежи (Tuples)

Кортежи очень похожи на списки, но имеют одно важное отличие — они неизменяемые. Кортежи создаются с помощью круглых скобок ().

```
>>> mytuple = ("apple", "cherry", 44, True)
>>> len(mytuple)
4
>>> mytuple[0]
'apple'
>>> mytuple[-1]
True
>>> mytuple.index(44)
2
>>>
>>> len(mytuple)
4
```

```
>>> tup=(2,4,12,6,32,4,21,4,0)
>>> tup.count(4)
3
>>> sum(tup)
85
>>> min(tup)
0
>>> max(tup)
32
```

## Зачем нужны кортежи, если есть списки?

1. Кортеж защищен от изменений (намеренных или случайных).
2. Кортеж имеет меньший размер, чем список.

```
>>> a=[1,2,3,4,5,6]
>>> b=(1,2,3,4,5,6)
>>> a.__sizeof__()
136
>>> b.__sizeof__()
72
>>>
```

`__sizeof__()` возвращает  
размер объекта в байтах.

Создать кортеж можно из итерируемого объекта функцией **tuple()**

```
>>> tuple("hello")
('h', 'e', 'l', 'l', 'o')
```

Создать список можно из итерируемого объекта функцией **list()**

```
>>> list('word')
['w', 'o', 'r', 'd']
```

Больше информации о кортежах смотрите в официальной документации

<https://docs.python.org/3.6/tutorial/datastructures.html#tuples-and-sequences>

```
>>> t = 12345, 54321, 'hello!'
>>> t[0]
12345
>>> t
(12345, 54321, 'hello!')
>>> # Tuples may be nested:
... u = t, (1, 2, 3, 4, 5)
>>> u
((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
>>> # Tuples are immutable:
... t[0] = 88888
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> # but they can contain mutable objects:
... v = ([1, 2, 3], [3, 2, 1])
>>> v
([1, 2, 3], [3, 2, 1])
```



# Словари (dict)

**Словари** - неупорядоченные коллекции произвольных объектов с доступом по ключу. Их иногда ещё называют **ассоциативными массивами**.

Словарь состоит из пар ключ-значение (**key** - **value**), которые разделяются запятыми. Внутри каждой пары значение отделяется от ключа двоеточием.

`dict = { k: v }`

`p_ages = {"Андрей": 32, "Виктор": 29, "Максим": 18}`

`p_ages = {32: "Андрей", 29: "Виктор", 18: "Максим"}`

В отличие от списков и кортежей **у словарей нет определенного порядка**. Можно представить, что пары из ключа и значения перемешаны в мешке. И в нем не существует первого, второго или последнего элементов — они просто случайно существуют. Такая структура нацелена на увеличение производительности и предполагает доступ к значению по ключу.

Так как доступ осуществляется по ключу, то **требуется, чтобы ключи были уникальными** (в пределах одного словаря).

```
>>> tel={'sape': 4139, 'guido': 4127, 'jack': 4098}
>>> tel['jack']
4098
```

# задаем словарь

# ищем значение по ключу

```
>>> tel['mark'] = 7137
>>> tel
{'sape': 4139, 'guido': 4127, 'jack': 4098, 'mark': 7137}
```

# добавляем новые ключ-значение

## Создать словарь можно:

1. с помощью литерала {}

```
>>> tel={'sape': 4139, 'guido': 4127}
>>> tel
{'sape': 4139, 'guido': 4127}
```

2. с помощью функции **dict**:

```
>>> d=dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
>>> d
{'sape': 4139, 'guido': 4127, 'jack': 4098}
```

или так:

```
>>> dict(sape=4139, guido=4127, jack=4098)
{'sape': 4139, 'guido': 4127, 'jack': 4098}
```

3. с помощью метода **fromkeys**:

```
>>> d = dict.fromkeys(['a', 'b'])
>>> d
{'a': None, 'b': None}
```

4. с помощью генераторов словарей, которые очень похожи на генераторы списков.

```
>>> d = {a: a ** 2 for a in range(7)}
>>> d
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36}
```

# Методы словарей в Python

Перечислим основные словарные методы, которые помогут вам при работе с этим типом данных.

**clear()** – очищает заданный словарь, приводя его к пустому.

**get()** – отдаёт значение словаря по указанному ключу. Если ключ не существует, а в качестве дополнительного аргумента передано значение по умолчанию, то метод вернет его. Если же значение по умолчанию опущено, метод вернет None.

**items()** – возвращает словарные пары ключ:значение, как соответствующие им кортежи.

**keys()** – возвращает ключи словаря, организованные в виде списка.

**values()** – подобным образом, возвращает список значений словаря.

**pop()** – удалит запись словаря по ключу и вернет её значение.

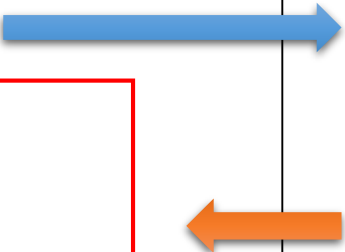
**popitem()** – выбрасывает пару ключ:значение из словаря и возвращает её в качестве кортежа. Такие пары возвращаются в порядке LIFO.

**update()** – реализует своеобразную операцию конкатенации для словарей. Он объединяет ключи и значения одного словаря с ключами и значениями другого. При этом если какие-то ключи совпадут, то результирующим значением станет значение словаря, указанного в качестве аргумента метода update.

**copy()** – создает полную копию исходного словаря.

**.get()** — полезный метод для получения значений из словаря по ключу.

```
>>> p_ages = {"Андрей": 32, "Виктор": 29, "Максим": 18}
>>> p_ages.get("Виктор")
29
>>> p_ages.get("Анатолий", "Не найдено")
'Не найдено'
>>> p_ages["Анатолий"]
Traceback (most recent call last):
  File "<pyshell#43>", line 1, in <module>
    p_ages["Анатолий"]
KeyError: 'Анатолий'
```



Если добавить в метод второй параметр, то он вернет переданное значение в случае, когда ключ не будет найден.  
Применение `.get()` позволяет избежать исключения (ошибки), как в случае простого поиска по ключу

**.items()** возвращает список кортежей, каждый из которых является парой из ключа и значения. Метод `.items()` пригодится при необходимости использовать индексацию для доступа к данным.

```
>>> tel={'sape': 4139, 'guido': 4127, 'jack': 4098}
>>> tel.items()
dict_items([('sape', 4139), ('guido', 4127), ('jack', 4098)])
```

**.update([other])** - обновляет словарь, добавляя пары (ключ, значение) из `other`. Одинаковые ключи перезаписываются значениями из второго словаря.

```
>>> p_ages.update(tel)
>>> p_ages
{'Андрей': 32, 'Виктор': 29, 'Максим': 18, 'sape': 4139, 'guido': 4127}
```

Начиная с версии Python 3.9, в языке появились новые операторы, которые облегчают процесс слияния словарей.

**Merge (|):** этот оператор позволяет объединять два словаря с помощью одного символа |. При этом создает новый словарь, объединяя два:

```
>>> p_pages = {"Андрей": 32, "Виктор": 29, "Максим": 18}
>>> tel={'sape': 4139, 'guido': 4127}
>>> dict = p_pages|tel
>>> dict
{'Андрей': 32, 'Виктор': 29, 'Максим': 18, 'sape': 4139, 'guido': 4127}
```

**Update (|=):** с помощью такого оператора можно обновить первый словарь значением второго. При этом новый словарь не создается:

```
>>> p_pages = {"Андрей": 32, "Виктор": 29, "Максим": 18}
>>> tel={'sape': 4139, 'guido': 4127}
>>> tel|=p_pages
>>> tel
{'sape': 4139, 'guido': 4127, 'Андрей': 32, 'Виктор': 29, 'Максим': 18}
```

Оператор merge (|) упрощает процесс объединения словарей и работы с их значениями. Оператор update (|=) используется для обновления словарей.

## Перебор словаря

Перебор по ключу:

```
iter_dict = {'key_b': 1, 'key_d': 0, 'key_e': -2, 'key_c': 95, 'key_a': 13}
for key in iter_dict:
    print(key, end=' ')
```

key\_b key\_d key\_e key\_c key\_a

Перебор по значению:

```
iter_dict = {'key_b': 1, 'key_d': 0, 'key_e': -2, 'key_c': 95, 'key_a': 13}
for v in iter_dict.values():
    print(v, end=' ')
```

1 0 -2 95 13

С использованием метода **.items()**. В этом случае на каждой итерации, пара ключ: значение будет возвращаться к нам в виде кортежа ('ключ', значение):

```
iter_dict = {'key_b': 1, 'key_d': 0, 'key_e': -2, 'key_c': 95, 'key_a': 13}
for item in iter_dict.items():
    print(item, end=' ')

> ('key_b', 1) ('key_d', 0) ('key_e', -2) ('key_c', 95) ('key_a', 13)
```

# Множество (Set)

**Множество (набор)** - "контейнер", содержащий не повторяющиеся и неизменяемые элементы в случайном порядке.

Создаём множества разными способами

1

```
>>> a = set()
>>> a
set()
```

2

```
>>> a = set('hello')
>>> a
{'h', 'o', 'l', 'e'}
```

Во множестве элементы не повторяются.

3

```
>>> a = {'a', 'b', 'c', 'd'}
>>> a
{'b', 'c', 'a', 'd'}
```

Множества неупорядочены, поэтому вы не можете быть уверены, в каком порядке будут отображаться элементы.

4

```
>>> a = {i ** 2 for i in range(10)} # генератор множеств
>>> a
{0, 1, 4, 81, 64, 9, 16, 49, 25, 36}
```

```
>>> a = {} # А так нельзя!
>>> type(a)
<class 'dict'>
```

Множества удобно использовать для удаления повторяющихся элементов:

```
>>> words = ['hello', 'daddy', 'hello', 'mum']
>>> set(words)
{'hello', 'daddy', 'mum'}
```

### Уже знакомые нам методы:

**len(s)** - число элементов в множестве (размер множества).

**x in s** - принадлежит ли x множеству s.

**.copy()** - копия множества.

**.add(elem)** - добавляет элемент в множество.

**.remove(elem)** - удаляет элемент из множества.

**.pop()** - удаляет первый элемент из множества. Так как множества не упорядочены, нельзя точно сказать, какой элемент будет первым.

**.clear()** - очистка множества.

**set.update(other)** – добавляет в **other** элементы из **set**.

```
thisset = {"apple", "banana", "cherry"}  
tropical = {"pineapple", "mango", "papaya"}  
  
thisset.update(tropical)
```

```
{'pineapple', 'banana', 'apple', 'papaya', 'mango', 'cherry'}
```



## НОВЫЕ ВОЗМОЖНОСТИ МНОЖЕСТВ: объединение, пересечение и т.д.:

**set.isdisjoint(other)** - истина, если set и other не имеют общих элементов.

**set.issubset(other)** или **set <= other** - все элементы set принадлежат other.

**set.issuperset(other)** или **set >= other** - аналогично.

**set.union(other, ...)** или **set | other | ...** - объединение нескольких множеств.

**set.intersection(other, ...)** или **set & other & ...** - пересечение.

**set.difference(other, ...)** или **set - other - ...** - множество из всех элементов set, не принадлежащие ни одному из other.

**set.symmetric\_difference(other); set ^ other** - множество из элементов, встречающихся в одном множестве, но не встречающиеся в обоих.

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

x.intersection_update(y)

print(x)
```

```
=====
{'apple'}
>>>
```

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

# s = y ^ x
s = x.symmetric_difference(y)
print(s)
```

```
===== RESTART: D:/IBA/set.py
{'google', 'microsoft', 'banana', 'cherry'}
>>> |
```

```
x.symmetric_difference_update(y)
print(x)
```

## Обобщение свойств встроенных коллекций в сводной таблице:

Тип коллекции	Изменяемость	Индексированность	Уникальность	Как создаём
<b>Список</b> (list)	+	+	-	<code>[]</code> <code>list()</code>
<b>Кортеж</b> (tuple)	-	+	-	<code>()</code> , <code>tuple()</code>
<b>Строка</b> (string)	-	+	-	<code>"</code> <code>" "</code>
<b>Множество</b> (set)	+	-	+	<code>{elm1, elm2}</code> <code>set()</code>
<b>Неизменяемое множество</b> (frozenset)	-	-	+	<code>frozenset()</code>
<b>Словарь</b> (dict)	+ элементы - ключи + значения	-	+ элементы + ключи - значения	<code>{}</code> <code>{key: value,}</code> <code>dict()</code>

<https://habr.com/ru/post/319164/>

Примечание для словаря (dict):

сам словарь изменяем — можно добавлять/удалять новые пары ключ: значение;

значения элементов словаря — изменяемые и не уникальные;

а вот ключи — не изменяемые и уникальные, поэтому, например, мы не можем сделать ключом словаря список, но можем кортеж. Из уникальности ключей, так же следует уникальность элементов словаря — пар ключ: значение.