



# Машинное обучение

---

НИЯУ МИФИ, КАФЕДРА ФИНАНСОВОГО МОНИТОРИНГА

КУРС ЛЕКЦИЙ

В.Ю. РАДЫГИН. Д.Ю. КУПРИЯНОВ

ЛЕКЦИЯ 1

# Знакомство с лектором

---

**Радыгин Виктор Юрьевич** — к.т.н., доцент кафедры финансового мониторинга (75), начальник департамента цифрового развития НИЯУ МИФИ.

Электронная почта: [vyradygina@mephi.ru](mailto:vyradygina@mephi.ru)

Лучше всего отправлять письма через портал [home.mephi.ru](http://home.mephi.ru)!

Аудитория, где можно задать вопросы лично: Г-328 (понедельник-пятница). Если есть вопросы, не бойтесь заходить, но желательно не более 1-2-х человек сразу.

Рабочий телефон: +7 (495) 788 56 99, доб. 7064



# Другие преподаватели курса

---

**Куприянов Дмитрий Юрьевич** — к.т.н., доцент кафедры финансового мониторинга (75), начальник отдела цифровых и мобильных сервисов и услуг управления информационно-методического обеспечения образовательного процесса департамента цифрового развития НИЯУ МИФИ. Электронная почта: [dykupriyanov@mephi.ru](mailto:dykupriyanov@mephi.ru). Аудитория: Г-328 (понедельник-пятница).



**Манаенкова Татьяна Андреевна** — старший преподаватель кафедры финансового мониторинга (75), Электронная почта: [TAManaenkova@mephi.ru](mailto:TAManaenkova@mephi.ru).

# Часть 1

---

ЦЕЛЬ И ЗАДАЧИ ДИСЦИПЛИНЫ

# Машинное обучение

---

Машинное обучение (Machine Learning) представляет собой набор методов обучения компьютерных систем, основанных на статистических моделях, логических операциях и нейросетевых алгоритмах и решающих задачи не напрямую по заранее известному алгоритму, а за счет применения решений множества схожих задач.

# Виды данных

---

Структурированные данные – данные, которые имеют строго определённую структуру (например, таблицы в базе данных).

Неструктурированные данные – данные, не имеющие predetermined структуры или формата (например, тексты, изображения).

# Датасет

---

Основой машинного обучения является предварительно собранная информация. Совокупность обработанной и структурированной информации образует набор данных или датасет. Например, набор структурированной информации об основных результатах обследования различных пациентов врача кардиолога.

Более формально, **датасет, или набор данных**, – это совокупность данных, систематизированных в определённом формате, представляющих собой базовый элемент для работы с данными во многих отраслях.

Датасет может быть и неструктурированным. В таком случае, с ним нельзя работать без предварительной подготовки.

# Наблюдение

---

**Наблюдение** – это отдельный экземпляр в датасете, обычно представленный одной строкой.

Например, данные одного из пациентов кардиолога.

На жаргоне наблюдение часто называют строкой.



# Признак

---

**Признак** – это переменная, описывающая одну из характеристик каждого наблюдения в датасете.

Грубо говоря, это один из столбцов датасета (если он представлен в виде таблицы).

Например, возраст пациентов кардиолога.

На жаргоне признак называют атрибутом, столбцом и т.д.

# Подготовка датасета

---

- ❑ Сбор – разные способы получения набор данных, включая непосредственное наблюдение, опросы, интервью, анализ уже существующих данных, и т.д.
- ❑ Очистка – процесс выявления и удаление ошибок и несоответствий данных с целью улучшения их качества.
- ❑ Обработка и трансформация – этот шаг включает преобразование информации в формат, который можно использовать для анализа. Это может включать преобразование неструктурированных данных в структурированные, нормализацию, кодирование категориальных признаков и т.д.

# Датасет должен быть надежным

---

Свойства надежного датасета:

- ❑ Точность – данные представляют реальность без ошибок и искажений.
- ❑ Полнота – все необходимые данные присутствуют в датасете (нет ситуаций, не рассмотренных в датасете).
- ❑ Согласованность – данные согласованы и не имеют противоречий.
- ❑ Актуальность – данные отражают текущую реальность, они своевременны.
- ❑ Достоверность – данные можно проверить и подтвердить.

# Разметка датасета

---

**Разметка датасета** (аннотирование данных) – это процесс добавления тэгов в сырые данные, чтобы показать модели машинного обучения целевые атрибуты (ответы), которые она должна предсказывать. Метка (label) или тэг (tag) — это описательный элемент, сообщающий модели, чем является отдельный элемент данных, чтобы она могла учиться на примере.

Например, добавление в датасет пациентов нового столбца «диагноз», где каждому больному сопоставлено заболевание сердца или информация о полном отсутствии кардиологических проблем.

# Виды машинного обучения

---

- ❑ Обучение с учителем – при таком виде обучения имеется заранее размеченный набор данных (датасет), где у каждого наблюдения есть значение целевой функции (грубо говоря, ответ). На основе данного набора обучается определенная вычислительная модель. После чего мы можем применять её для получения новых ранее неизвестных результатов. Например, автоматически предсказывать диагноз для новых пациентов по их результатам обследования.
- ❑ Обучение без учителя – при таком виде обучения у нас есть только возможность сравнивать наблюдения друг с другом. Именно из таких сравнений и получаются новые знания.

# Классические задачи, решаемые с помощью машинного обучения

---

- ☐ Классификация
- ☐ Кластеризация
- ☐ Регрессия
- ☐ Понижение размерности данных
- ☐ Восстановление плотности распределения вероятности по набору данных
- ☐ Построение ранговых зависимостей
- ☐ Обнаружение аномалий
- ☐ Визуализация данных

# Примеры задач, решаемых с помощью машинного обучения

---

## **Интернет-технологии:**

- персонализация посетителей веб-сайтов;
- поиск случаев мошенничества с кредитными картами;
- Web Mining: Web content mining, Web structure mining и Web usage mining.

## **Торговля:**

- анализ рыночных корзин и секвенциальный анализ.

## **Телекоммуникации:**

- анализ доходности и риска потери клиентов;
- защита от мошенничества;
- выявление категорий клиентов с похожими стереотипами пользования услугами и разработка привлекательных наборов цен и услуг.

# Примеры задач, решаемых с помощью машинного обучения

---

## **Промышленное производство:**

- прогнозирование качества изделия в зависимости от измеряемых параметров технологического процесса.

## **Медицина и биология:**

- построение диагностической системы;
- исследование эффективности хирургического вмешательства.

## **Биоинформатика:**

- изучение генов, разработка новых лекарств.

## **Банковское дело:**

- оценка кредитоспособности заемщика.



# Примеры задач, решаемых с помощью машинного обучения

---

## **Страховой бизнес:**

- привлечение и удержание клиентов, прогнозирование финансовых показателей.

## **Розничная торговля:**

- анализ деятельности торговых точек, построение профиля покупателя, управление ресурсами.

## **Биржевые трейдеры:**

- выработка оптимальной торговой стратегии, контроль рисков.

## **Ученые и инженеры:**

- построение эмпирических моделей, основанных на анализе данных, решение научно-технических задач.

# Цель и задачи дисциплины

---

## *Цель:*

Получить знания и навыки работы с современными технологиями машинного обучения.

## *Задачи дисциплины:*

1. Получить навыки работы с современными технологиями машинного обучения и анализа данных. В частности, изучить язык **Python** и основные математические библиотеки\*.
2. Получить теоретические знания в области построения интеллектуальных систем.
3. Научиться решать ряд простейших задачи машинного обучения в области потенциальной профессиональной деятельности самостоятельно от начала и до конца.

\* Из задачи 1 вытекает подзадача научиться программировать на языке Python. Поэтому первые занятия будут посвящены, именно, обучению обычному программированию.

# История машинного обучения

---

- ❑ 1958 год – Фрэнк Розенблатт создал свой персептрон. Но эта технология не получила развития в 60-е годы.
- ❑ 1959 год – Артур Самуэль создал первую программу по игре в шашки, которая умела играть сама с собой и обучаться самостоятельно.
- ❑ Конец 80-х годов – создание машины для игры в шахматы ChipTest аспирантами университета Карнеги-Меллон.
- ❑ 1996 год – Deep Blue победил Гарри Каспарова.
- ❑ 2010-е года – появление подразделений ИИ у супергигантов IT-индустрии.

# Часть 2

---

ЯЗЫК PYTHON. ВВЕДЕНИЕ

# В качестве эпиграфа

---

- Что это? – спросила она. – Мы так и остались под этим деревом! Неужели мы не стронулись с места ни на шаг?*
- Ну, конечно, нет, – ответила Королева. – А ты чего хотела?*
- У нас, – сказала Алиса, с трудом переводя дух, – когда долго бежишь со всех ног, непременно попадёшь в другое место.*
- Какая медлительная страна! – сказала Королева. – Ну, а здесь, знаешь ли, приходится бежать со всех ног, чтобы только остаться на том же месте! Если же хочешь попасть в другое место, тогда нужно бежать по меньшей мере вдвое быстрее!*
- Ах, нет, я никуда не хочу попасть! – сказала Алиса. – Мне и здесь хорошо. Очень хорошо!*

Льюис Кэрролл «Алиса в Зазеркалье»

# Язык программирования Python

---

**Python** – это свободный высокоуровневый объектно-ориентированный язык программирования, появившийся относительно давно (в 1991 году), но получивший огромную популярность на текущее время.

Основным преимуществом языка Python перед другими языками программирования является мощная база математических библиотек, в том числе и для решения задач Data Mining. Данный факт признан ведущими университетами и научными организациями мира. К примеру, вот как характеризуется Python Berkeley [1]: “Python has become the *de facto* superglue language for modern scientific computing”.

# Простота Python

---

| C++   | Java  |
|---|---|
| <pre>#include &lt;iostream&gt;  using namespace std;  int main(){     cout &lt;&lt; "Hello World!" &lt;&lt; endl;     return 0; }</pre> | <pre>class Hello {     public static void main( String args[] ) {         System.out.println( "Hello World!" );     } }</pre> |
| Ruby  | Python  |
| <pre>puts "Hello World!"</pre>  | <pre>print("Hello World")</pre>   |

Как видно из данного примера в простоте использования большинство популярных языков уступает Python. Исключение составляет лишь небольшое число языков, например, язык Ruby.

# Версии Python

---

Язык Python в последнее время развивается очень стремительно. В данном курсе мы будем опираться на актуальную на 1 сентября 2024 года версию 3.12 [2], который доступен в релизе 3.12.5 (или более старших версиях семейства 3.12). Быстрое изменение языка обуславливает серьезные отличия его современных версий от более ранних. При этом Python не обладает обратной совместимостью. В частности, программа, написанная для Python версии 2 в большинстве случаев не будет работать в Python версии 3 и наоборот.

*Будьте осторожны, читая документацию по языку Python в сети Интернет! Лучше всегда добавлять цифру 3 в поисковом запросе, чтобы не попасть на документацию для версии 2!*



# Документация

---

Python быстро развивается поэтому хорошую документацию для последних версий языка на русском языке найти практически нельзя. Есть разные варианты переводов официальной документации, но выполнены они преимущественно энтузиастами, а не официальным разработчиком.

Официальная документация для языка Python на английском языке доступна на его официальном сайте [3].

# Синтаксис языка Python

---

Язык Python иногда называют «языком с двумерным синтаксисом». Это название происходит от того, что отделение основных конструкций языка от их содержимого осуществляется не за счет фигурных скобок (как, например, в C++ и Java) или конструкций do-end (как, например, в языке Ruby), а за счет отступов.

Познакомимся с данной особенностью языка на примере оператора while.

# Пример 1. Таблица умножения

## C++ Вариант 1

```
#include <iostream>

using namespace std;

int main(){
    int i = 2, j;
    while(i < 10){
        j = 2;
        while(j < 10){
            cout << j << " * " << i << " = " << i * j << "\t";
            j++;
        }
        cout << endl;
        i++;
    }
}
```

## C++ Вариант 2

```
#include <iostream>

using namespace std; int main(){int i = 2, j;while(i < 10){j = 2;
while(j < 10){cout << j << " * " << i << " = " << i * j << "\t";
j++;}cout << endl;i++;}}
```

## Python

```
i = 2
while i < 10:
    j = 2
    while j < 10:
        print(str(j) + " * " + str(i) + " = " + str(i * j), end = "\t")
        j += 1
    print()
    i += 1
```

# Результат работы примера 1

```
In [4]: i = 2
while i < 10:
    j = 2
    while j < 10:
        print(str(j) + " * " + str(i) + " = " + str(i * j), end = "\t")
        j += 1
    print()
    i += 1
```

|            |            |            |            |            |            |            |            |
|------------|------------|------------|------------|------------|------------|------------|------------|
| 2 * 2 = 4  | 3 * 2 = 6  | 4 * 2 = 8  | 5 * 2 = 10 | 6 * 2 = 12 | 7 * 2 = 14 | 8 * 2 = 16 | 9 * 2 = 18 |
| 2 * 3 = 6  | 3 * 3 = 9  | 4 * 3 = 12 | 5 * 3 = 15 | 6 * 3 = 18 | 7 * 3 = 21 | 8 * 3 = 24 | 9 * 3 = 27 |
| 2 * 4 = 8  | 3 * 4 = 12 | 4 * 4 = 16 | 5 * 4 = 20 | 6 * 4 = 24 | 7 * 4 = 28 | 8 * 4 = 32 | 9 * 4 = 36 |
| 2 * 5 = 10 | 3 * 5 = 15 | 4 * 5 = 20 | 5 * 5 = 25 | 6 * 5 = 30 | 7 * 5 = 35 | 8 * 5 = 40 | 9 * 5 = 45 |
| 2 * 6 = 12 | 3 * 6 = 18 | 4 * 6 = 24 | 5 * 6 = 30 | 6 * 6 = 36 | 7 * 6 = 42 | 8 * 6 = 48 | 9 * 6 = 54 |
| 2 * 7 = 14 | 3 * 7 = 21 | 4 * 7 = 28 | 5 * 7 = 35 | 6 * 7 = 42 | 7 * 7 = 49 | 8 * 7 = 56 | 9 * 7 = 63 |
| 2 * 8 = 16 | 3 * 8 = 24 | 4 * 8 = 32 | 5 * 8 = 40 | 6 * 8 = 48 | 7 * 8 = 56 | 8 * 8 = 64 | 9 * 8 = 72 |
| 2 * 9 = 18 | 3 * 9 = 27 | 4 * 9 = 36 | 5 * 9 = 45 | 6 * 9 = 54 | 7 * 9 = 63 | 8 * 9 = 72 | 9 * 9 = 81 |

# Разберём пример

---

В данном примере строится таблица умножения при помощи двух циклов, перебирающих числа от двух до 9 с шагом 1.

Первый цикл осуществляется при помощи оператора `while` и переменной `i`. Всё, что повторяется внутри первого цикла имеет отступ не менее 4 пробелов. Число пробелов не обязательно должно быть равно четырём. Главное требование – чтобы у всех вложенных строк число пробелов было большим, чем у внешнего блока (в данном случае описываемого оператором `while`). Например, мы могли бы поставить у всех строк 1 пробел.

# Разберём пример

---

Второй цикл осуществляется при помощи второго оператора `while` и переменной `j`. Всё, что повторяется внутри второго цикла имеет отступ не менее 8 пробелов.

Большинство конструкций языка Python, имеющих вложенный код (например, такие как операторы `while`, `if`, `for` и т.д.) в соответствии с синтаксисом должны заканчиваться двоеточием, а соответственно вложенный код должен иметь больший отступ, чем сам оператор.

Действие оператора заканчивается тогда, когда отступ возвращается на предыдущий уровень (или в конце файла)!

# Почему 4 пробела?

---

Как уже было сказано, можно было использовать не 4 и 8 пробелов для отступов внутри цикла, а, например, 1 и 2. Но «правильные» программисты следуют общепринятым соглашениям оформления программ на Python. Наиболее популярные соглашения описаны в PEP 8 (аббр. от Python Enhancement Proposal, Предложение по улучшению Python) [4], который называется Style Guide for Python Code (Руководство по стилю для кода Python).

Считается, что 4 пробела хорошо отличимы друг от друга (то есть, строка с 4 пробелами, легко отличима от строки с 8 и т.д.). При этом 4 пробела гораздо менее громоздки, чем 8 (число пробелов по умолчанию для знака табуляции).

# Основные типы данных

---

В Python любое значение – это объект. Число, строка, кортеж, булева величина, функция – все это объект. У каждого объекта есть свой тип.

Тип объекта можно узнать при помощи функции `type(объект)`.



# Посмотрим какие бывают типы

---

```
► In [13]: print(type('This is test!'))  
           print(type(-1))  
           print(type(100))  
           print(type("Text"))  
           print(type(True))  
           print(type(False))  
           print(type(-5.6))
```

```
<class 'str'>  
<class 'int'>  
<class 'int'>  
<class 'str'>  
<class 'bool'>  
<class 'bool'>  
<class 'float'>
```

int – целые числа

str – строки в кодировке Unicode. Записываются в одинарных или двойных кавычках. Разницы между кавычками нет!

bool – логический тип True или False (на самом деле это целое число: 1 или 0).

float – действительное число.

Одноименные функции позволяют осуществлять преобразования типов.

# Преобразования типов

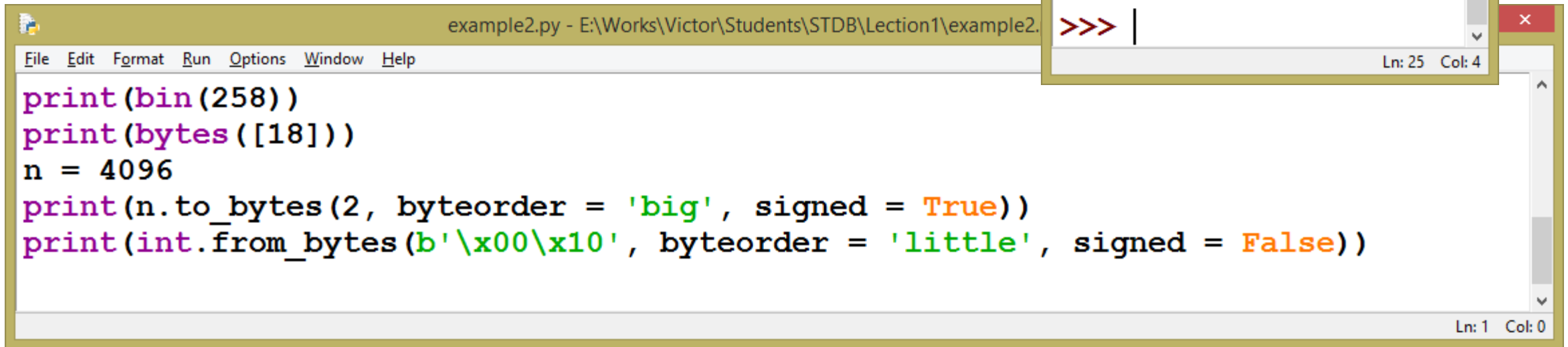
```
► In [18]: print(bool(0))  
           print(bool("True"))  
           print(bool(1))  
           print(str(123))  
           print(int("456"))  
           print(int(345.56))  
           print(float("22.2"))  
           print(int(True))  
           print(int("12b", 16))
```

```
False  
True  
True  
123  
456  
345  
22.2  
1  
299
```

Последний пример показывает перевод из строки, записанной в 16-й системе, в десятичное число.

# Двоичное или байтовое представление

Двоичное или байтовое представление чисел можно получить различными способами.



```
example2.py - E:\Works\Victor\Students\STDB\Lecton1\example2.  
File Edit Format Run Options Window Help  
print(bin(258))  
print(bytes([18]))  
n = 4096  
print(n.to_bytes(2, byteorder = 'big', signed = True))  
print(int.from_bytes(b'\x00\x10', byteorder = 'little', signed = False))  
Ln: 1 Col: 0
```

IDLE Shell 3.9.7

```
File Edit Shell Debug Options Window Help  
0b100000010  
b'\x12'  
b'\x10\x00'  
4096  
>>> |  
Ln: 25 Col: 4
```

# Простейшие операции с числами

| Оператор       | Пример                 | Описание                                     | Тип результата   |
|----------------|------------------------|--|--|
| Унарные +<br>- | +2<br>-(3)             | Не делает ничего!<br>Инвертирует знак числа. | Такой же, как у самого числа                                       |
| +<br>-<br>*    | 2+4<br>5.2-6.8<br>2*22 | Сложение<br>Вычитание<br>Умножение           | Если оба аргумента int, то int. Если хотя бы один float, то float. |
| /              | 10/5                   | Вещественное деление                         | Всегда float   |
| //             | 10//5                  | Целочисленное деление                        | Если оба аргумента int, то int. Если хотя бы один float, то float. |
| %              | 10%6                   | Остаток от деления                           | Если оба аргумента int, то int. Если хотя бы один float, то float. |
| **             | 2**10                  | Возведение в степень                         | Если оба аргумента int, то int. Если хотя бы один float, то float. |

# Простейшие операции с числами

```
In [20]: print(10/3)
print(10//3)
print(10//3.5)
print(2**10)
print(8**0.3)
print(++3)
print(--3)
print(10%4)
print(10%4.12)
print(-3**2)

3.3333333333333335
3
2.0
1024
1.8660659830736148
-3
3
2
1.7599999999999998
-9
```

Обратите внимание на две последних строки примера.

Вычисления 9-й строки должны были дать ответ 1.76, но из-за погрешности представления вещественных чисел на ЭВМ мы получили число чуть меньшее.

При вычислении 10-й строки приоритет операции возведения в степень выше приоритета унарного минуса. Поэтому сначала вычисляется квадрат числа 3, а уже после этого к результату 9 применяется унарный минус.

# Встроенные функции для работы

---

| Функция | Пример                      | Описание  |
|---------|-----------------------------|---|
| abs     | abs(-5)                     | Модуль числа  |
| divmod  | divmod(10, 3)               | Частное и остаток от деления                              |
| round   | round(123.4567, 2)          | Округление (второй аргумент – число знаков после запятой) |
| sum     | sum((1, 2.6, -3.1, 5, 0.4)) | Сумма аргументов  |
| pow     | pow(5, 3)                   | Степень (альтернативный вариант)                          |

# Простейший ввод

---

Для организации простейшего ввода строки данных в Python предусмотрена функция `input`. По умолчанию она считывает строку текста и возвращает её (без символа конца строки) в качестве своего результата.

Кроме того, можно передать функции `input` аргумент – сообщение, поясняющее пользователю, что он должен вводить.

# Пример 2

---

```
► In [16]: a = input()
           b = input()
           print(a, b, a + b)
           c = input('Введите c: ')
           print(len(c))
```

```
12
34
12 34 1234
Введите c: 56
2
```



# Преобразования типов

---

```
► In [17]: a = int(input())  
           b = float(input())  
           c = bool(input())  
           print(a, b, a + b, c)
```

```
12  
34  
True  
12 34.0 46.0 True
```

Для ввода значений других типов данных используем функции преобразования типов.

# Строки в Python

---

Строки в Python 3 – это последовательности символов, записанных в кодировке Unicode. Для задания строки можно использовать одинарные или двойные кавычки. Оба варианта эквивалентны. Единственное отличие, что в двойных кавычках нужно экранировать символ двойной кавычки, а в одинарных – символ одинарной.

Строки относятся к типу `str`. Методы работы со строками можно прочесть в [5].

```
In [1]: str1 = "This is string"
        print(str1)
        str2 = "This is quoted \"text\"."
        print(str2)
        str3 = 'This is quoted "text".'
        print(str3)
```

```
This is string
This is quoted "text".
This is quoted "text".
```

# Спецсимволы и «многострочные строки»

При использовании обычных одинарных или двойных кавычек в строках можно использовать последовательности спецсимволов, начинающиеся с обратного слэша, например, символ переноса строки `\n`, символ табуляции `\t` и т.д.

```
► In [3]: str1 = "First paragraph.\nSecond paragraph."  
print(str1)  
str2 = 'First\tSecond\nThird\tFourth'  
print(str2)
```

```
First paragraph.  
Second paragraph.  
First    Second  
Third    Fourth
```

# Спецсимволы и «многострочные строки»

```
In [4]: str1 = "First paragraph.  
Second paragraph."  
  
File "<ipython-input-4-426cf4dad83>", line 1  
      str1 = "First paragraph.  
              ^  
SyntaxError: EOL while scanning string literal
```

```
In [11]: str2 = """First paragraph.  
Second paragraph."""  
print(str2)  
str3 = "First\tSecond"  
print(str3)
```

```
First paragraph.  
Second paragraph.  
First    Second
```

Внутри обычных одинарных или двойных кавычек можно выполнять переход на новую строку! Для этого необходимо использовать синтаксис с тремя двойными кавычками.

# Склеивание строк

---

Python автоматически склеивает подряд идущие записи строк-констант. В других случаях необходимо использовать оператор +.

```
► In [18]: str1 = "This "is "test!"  
           print(str1)  
           str2 = "This " + "is " + "test!"  
           print(str2)  
           str3 = " And something else..."  
           print(str2 + str3)  
  
           This is test!  
           This is test!  
           This is test! And something else...
```

# Форматированные строки

```
► In [24]: name = input()
str = f"Hello dear {name}"
print(str)
n = 1/3
str2 = f"1/3 - {n:.3f}"
print(str2)
str3 = "1/3 - {:.3f}".format(n)
print(str3)
str4 = f"1/3 - %.3f" % (n)
print(str4)
```

```
Vasya
Hello dear Vasya
1/3 - 0.333
1/3 - 0.333
1/3 - 0.333
```

Внутри строк можно вставлять значения. Причём, при вставке, можно выполнять форматное преобразование чисел. Для этого необходимо использовать один из нескольких доступных вариантов и указать формат преобразования.

Прочсть о форматах можно в [6], [7] и [8].

# Простейшие управляющие конструкции

---

Кратко разберём простейшие управляющие конструкции. Прежде всего познакомимся с условным оператором if.

```
In [26]: n = int(input())  
         if n > 0:  
             print('Positive')  
         elif n < 0:  
             print('Negative')  
         else:  
             print('Zero')
```

```
-12  
Negative
```

# Простейшие управляющие конструкции

```
► In [28]: i = 0  
p = 1  
while i < 10:  
    print(p)  
    p *= 2  
    i += 1
```

```
1  
2  
4  
8  
16  
32  
64  
128  
256  
512
```

Простейший оператор `while` похож на аналоги из других языков.

Более подробно управляющие конструкции разберём на следующем занятии.



# Часть 3

---

ЯЗЫК PYTHON. КОЛЛЕКЦИОННЫЕ ТИПЫ ДАННЫХ

# Последовательные типы данных

---

В языке Python есть целый ряд последовательных типов данных (последовательностей), имеющих структуру, схожую с массивом (занумерованной последовательностью элементов). Их можно поделить на изменяемые (mutable) и неизменяемые (immutable).

Прежде всего, это:

- диапазоны – range (immutable);
- списки – list (mutable);
- кортежи – tuple (immutable).

Строки str в языке Python – это тоже последовательный тип данных. С ними можно работать также, как и с другими immutable конструкциями.

# Пример

---

```
▶ In [1]: 1 x = range(1, 22, 3)
          2 print(x)
          3 y = [1, 2, 3, 4]
          4 print(y)
          5 z = (1, 2, 3)
          6 print(z)
```

```
range(1, 22, 3)
[1, 2, 3, 4]
(1, 2, 3)
```

# Immutable против mutable

---

Чем mutable отличается от immutable? Объекты, относящиеся к mutable типам данных, позволяют изменять своё содержимое (поэлементно), а также изменять число элементов (увеличивать или уменьшать), не создавая нового объекта.

Например:

```
In [2]: 1 x = [1, 2, 3, 4]
        2 x[1] = 100
        3 x.append(-300)
        4 print(x)

[1, 100, 3, 4, -300]
```

Immutable объекты данных действий не позволяют.

# Immutable действия

---

Для всех immutable последовательностей есть набор базовых доступных операций. Данные операции доступны и для mutable последовательностей. Набор включает в себя:

|   |                                   |                         |
|---|-----------------------------------|-------------------------|
| <code>x in s</code>                       | <code>s[i]</code>                 | <code>len(s)</code>     |
| <code>x not in s</code>                   | <code>s[i:j]</code>               | <code>min(s)</code>     |
| <code>s + t</code>                        | <code>s[i:j:k]</code>             | <code>max(s)</code>     |
| <code>s * n</code> или <code>n * s</code> | <code>s.index(x[, i[, j]])</code> | <code>s.count(x)</code> |

Подробнее про данные операции можно прочесть в [9].

```

example1.py - E:\Works\Victor\Student...
File Edit Format Run Options Window Help
s = (1, 2, 3, 4, 5)
t = (6, 7)
n = 3
x = 2

print(x in s)
print(x not in s)
print(s + t)
print(s * n)
print(n * s)
print(s[2])
print(s[2:4])
print(s[1:5:2])
print(s.index(x))
print(s.index(x, 1))
print(s.index(x, 1, 3))
print(len(s))
print(min(s))
print(max(s))
print(s.count(x))

```

Ln: 20 Col: 17

IDLE Shell 3.10.7

File Edit Shell Debug Options Window Help

Python 3.10.7 (tags/v3.10.7:6cc6b13, Sep 5 2022, 14:08:36) [MSC v.1933 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license()" for more information.

>>>

===== RESTART: E:\Works\Victor\Students\STDB\Lection2\example1.py =====

True

False

(1, 2, 3, 4, 5, 6, 7)

(1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5)

(1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5)

3

(3, 4)

(2, 4)

1

1

1

5

1

5

1

>>>

Ln: 20 Col: 0

# Mutable действия

---

Для всех mutable последовательностей есть набор дополнительных (по отношению к immutable) базовых доступных операций. Он включает в себя:

|                           |  |                          |
|---------------------------|--|--------------------------|
| <code>s[i] = x</code>     | <code>s.append(x)</code>                         | <code>s.pop(i)</code>    |
| <code>s[i:j] = t</code>   | <code>s.clear()</code>                           | <code>s.remove(x)</code> |
| <code>s[i:j:k] = t</code> | <code>s.extend(t)</code> или <code>s += t</code> | <code>s.reverse()</code> |
| <code>del s[i:j]</code>   | <code>s *= n</code>                              | <code>s.copy()</code>    |
| <code>del s[i:j:k]</code> | <code>s.insert(i, x)</code>                      |                          |

Подробнее про данные операции можно прочесть в [9].

example1.py - E:\Works\Victo...  
File Edit Format Run Options Window Help

```
s = [1, 2, 3, 4, 5]
s[3] = 8
print(s)
s = [1, 2, 3, 4, 5]
s[2:3] = [9, 9]
print(s)
s = [1, 2, 3, 4, 5]
s[1:5:2] = [9, 9]
print(s)
s = [1, 2, 3, 4, 5]
del s[2:4]
print(s)
s = [1, 2, 3, 4, 5]
del s[1:5:2]
print(s)
s = [1, 2, 3, 4, 5]
s.append(6)
print(s)
s = [1, 2, 3, 4, 5]
s.clear()
print(s)
```

Ln: 59

example1.py - E:\Works\Victo...  
File Edit Format Run Options Window Help

```
print(s)
s = [1, 2, 3, 4, 5]
s.extend(t)
print(s)
s = [1, 2, 3, 4, 5]
s += t
print(s)
s = [1, 2, 3, 4, 5]
s *= n
print(s)
s = [1, 2, 3, 4, 5]
s.insert(2, 6)
print(s)
s = [1, 2, 3, 4, 5]
s.pop(2)
print(s)
s = [1, 2, 3, 4, 5]
s.remove(3)
print(s)
s = [1, 2, 3, 4, 5]
s.reverse()
print(s)
s = [1, 2, 3, 4, 5]
s.copy()
print(s)
```

Ln: 59 Col: 10

IDLE Shell 3.10.7  
dit Shell Debug Options Window Help

```
[1, 2, 3, 8, 5]
[1, 2, 9, 9, 4, 5]
[1, 9, 3, 9, 5]
[1, 2, 5]
[1, 3, 5]
[1, 2, 3, 4, 5, 6]
[]
[1, 2, 3, 4, 5, 6, 7]
[1, 2, 3, 4, 5, 6, 7]
[1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
[1, 2, 6, 3, 4, 5]
[1, 2, 4, 5]
[1, 2, 4, 5]
[5, 4, 3, 2, 1]
[1, 2, 3, 4, 5]
```

Ln: 186 Col: 0



# Списки – list

Python списки – это полноценный аналог массивов в языке Python. В них можно складывать элементы любых типов. Списки позволяют выполнять все immutable и mutable действия последовательностей, плюс метод sort.

```
► In [3]: 1 a = []  
          2 b = [1, 2, "abc", [3, 4]]  
          3 c = list()  
          4 d = list("abc")  
          5 print(a)  
          6 print(b)  
          7 print(c)  
          8 print(d)  
  
          []  
          [1, 2, 'abc', [3, 4]]  
          []  
          ['a', 'b', 'c']
```

# Оператор for

---

Оператор `for` позволяет построить цикл, поочерёдно перебирающий все элементы определенной последовательности или итерируемого объекта.

```
► In [4]: 1 a = [1, 2, 3, 4]
          2 for x in a:
          3     print(x)

          1
          2
          3
          4
```

# Ещё один способ задания списка

---

Список можно построить на основе итерируемого объекта при помощи конструкции [for].

```
In [8]: 1 a = [1, 2, 3, 4]
        2 sq = [x * x for x in a]
        3 print(sq)

[1, 4, 9, 16]
```

# Повторное использование for

---

```
In [12]: 1 a = [1, 2, 3, 4]
          2 b = [(x, y) for x in a for y in a]
          3 print(b[0:8])
          4 print(b[8:16])
```

[(1, 1), (1, 2), (1, 3), (1, 4), (2, 1), (2, 2), (2, 3), (2, 4)]  
[(3, 1), (3, 2), (3, 3), (3, 4), (4, 1), (4, 2), (4, 3), (4, 4)]

# Построение двумерного списка

---

```
In [9]: 1 a = [1, 2, 3, 4]
        2 sq = [[x * y for y in a] for x in a]
        3 print(sq)

[[1, 2, 3, 4], [2, 4, 6, 8], [3, 6, 9, 12], [4, 8, 12, 16]]
```

# Списки передаются по ссылкам!

---

```
▶ In [13]: 1 a = [1, 2, 3, 4]
           2 b = a
           3 b[0] = 100
           4 print(a)

           [100, 2, 3, 4]
```

# Для создания копии используйте copy()

---

```
In [14]: 1 a = [1, 2, 3, 4]
          2 b = a.copy()
          3 b[0] = 100
          4 print(a)
          5 print(b)
```

```
[1, 2, 3, 4]
```

```
[100, 2, 3, 4]
```

# [for] каждый раз создаёт новые сущности

```
In [21]: 1 a = 3 * [3 * [1]]  
2 print(a)  
3 a[0][0] = 5  
4 print(a)  
5 b = [3 * [x] for x in 3 * [1]]  
6 print(b)  
7 b[0][0] = 5  
8 print(b)
```

```
[[1, 1, 1], [1, 1, 1], [1, 1, 1]]  
[[5, 1, 1], [5, 1, 1], [5, 1, 1]]  
[[1, 1, 1], [1, 1, 1], [1, 1, 1]]  
[[5, 1, 1], [1, 1, 1], [1, 1, 1]]
```



# Диапазоны – range

---

Тип данных `range` предназначен для задания возрастающей или убывающей арифметической последовательности целых чисел (или эквивалентных им объектов других типов), ограниченной в рамках определённых значений. Допускает шаг последовательности, отличный от 1.

Левая граница включается. Правая – нет.

Обычно диапазоны используют для построения циклов `for`.

# Пример

---

```
► In [22]: 1 a = range(0, 10)
           2 print(list(a))
           3 b = range(1, 22, 3)
           4 print(list(b))
           5 c = range(22, 1, -3)
           6 print(list(c))
           7 for i in c:
           8     print(i)

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 4, 7, 10, 13, 16, 19]
[22, 19, 16, 13, 10, 7, 4]
22
19
16
13
10
7
4
```

# Вырезки в Python

---

На слайде 6 мы увидели, что с immutable последовательностями возможна операция `s[i:j:k]`. Что это такое и как работает со списками? Данная конструкция называется вырезкой (slicing).

В целом для списков доступно несколько видов вырезок:

- `[start:end]` – элементы, начиная со `start` до (включительно) `end - 1`
- `[start:]` – элементы, начиная со `start` и до конца списка
- `[:end]` – элементы, начиная с начала списка до (включительно) `end-1`
- `[:]` – копия всего списка
- `[start:end:step]` – элементы, начиная от `start` до (не включая) `end`, с шагом `step`

# Разберем на примере

---

```
► In [1]: 1 a = [1, 2, 3, 4, 5, 6, 7]
          2 b = a[2:5]
          3 c = a[2:]
          4 d = a[:2]
          5 e = a[:]
          6 f = a[2:6:2]
          7 print(b)
          8 print(c)
          9 print(d)
         10 print(e)
         11 print(f)
```

[3, 4, 5]  
[3, 4, 5, 6, 7]  
[1, 2]  
[1, 2, 3, 4, 5, 6, 7]  
[3, 5]

# Отрицательные индексы

---

Отрицательные индексы в Python означают номер с конца. Причём нумерация с конца списка начинается не с нуля, а с -1.

В целом для списков доступно несколько видов вырезок:

- `[-n]` – n-й с конца элемент списка;
- `[-n:]` – n последних элемента списка;
- `[:-n]` – все элементы, за исключением n последних элементов списка;
- `[-m:-n]` – все элементы, начиная с m-го с конца и до n-го с конца (не включая n-й).

# Разберем на примере

---

```
► In [3]: 1 a = [1, 2, 3, 4, 5, 6, 7]
          2 b = a[-3]
          3 c = a[-2:]
          4 d = a[:-2]
          5 e = a[-5:-1]
          6 print(b)
          7 print(c)
          8 print(d)
          9 print(e)

5
[6, 7]
[1, 2, 3, 4, 5]
[3, 4, 5, 6]
```

# Отрицательный шаг

---

```
▶ In [7]: 1 a = [1, 2, 3, 4, 5, 6, 7]
          2 b = a[-3::-1]
          3 c = a[3::-1]
          4 d = a[:-3:-1]
          5 e = a[:3:-1]
          6 f = a[6:1:-2]
          7 print(b)
          8 print(c)
          9 print(d)
         10 print(e)
         11 print(f)
```

[5, 4, 3, 2, 1]  
[4, 3, 2, 1]  
[7, 6]  
[7, 6, 5]  
[7, 5, 3]

# Больше чем есть – не страшно!

---

```
In [11]: 1 a = [1, 2, 3, 4, 5, 6, 7]
          2 b = a[-10:]
          3 c = a[11:12]
          4 d = a[-12:-1]
          5 print(b)
          6 print(c)
          7 print(d)

[1, 2, 3, 4, 5, 6, 7]
[]
[1, 2, 3, 4, 5, 6]
```



# Но не с индексом!

---

```
► In [12]: 1 a = [1, 2, 3, 4, 5, 6, 7]
           2 b = a[-10]
           3 print(b)
```

```
-----
IndexError                                Traceback (most recent call last)
<ipython-input-12-f712ec197f69> in <module>()
      1 a = [1, 2, 3, 4, 5, 6, 7]
----> 2 b = a[-10]
      3 print(b)

IndexError: list index out of range
```

```
In [ ]: 1
```

# Кортежи – tuple

---

Кортежи похожи на списки, но не могут изменяться и для них нет сортировки. Делаются кортежи при помощи круглых скобок.

```
▶ In [15]: 1 a = (1, 2, 3, 4, 5, 6, 7)
           2 b = (-10,)
           3 c = ()
           4 print(a)
           5 print(b)
           6 print(c)
           7 print(type(c))

(1, 2, 3, 4, 5, 6, 7)
(-10,)
()
<class 'tuple'>
```

```
In [ ]:
```

```
1
```

# Множества – set

---

Во многих языках программирования с массивами возможна работа в терминах множеств. Например, вычисление нового массива, как разности между двумя существующими массивами.

В Python для этих целей есть тип set.

# Пример

```
▶ In [16]: 1 a = set([1, 2, 3, 4])
           2 b = set([3, 4, 5, 6])
           3 print(a | b) # Объединение
           4 print(a & b) # Пересечение
           5 print(a - b) # Разность
           6 print(a ^ b) # Симметрическая разность

           {1, 2, 3, 4, 5, 6}
           {3, 4}
           {1, 2}
           {1, 2, 5, 6}
```

In [ ]:

1

Подробнее о работе с множествами можно прочесть в [10].

# Словари – dict

---

В Python, так же, как и во многих других языках есть возможность построения ассоциативного массива / хэша / словаря. Для этого есть тип dict.

# Пример

```
▶ In [20]: 1 a = dict(one = 1, two = 2, three = 3)
           2 b = {'one': 1, 'two': 2, 'three': 3}
           3 c = dict([('two', 2), ('one', 1), ('three', 3)])
           4 d = dict({'three': 3, 'one': 1, 'two': 2})
           5 print(a)
           6 print(b.keys())
           7 print(c.values())
           8 print(len(d))

{'one': 1, 'two': 2, 'three': 3}
dict_keys(['one', 'two', 'three'])
dict_values([2, 1, 3])
3
```

```
In [ ]: 1
```

Подробнее о работе с множествами можно прочесть в [11].

# Определение функций

Простейшее определение функций в Python подразумевает использование конструкции def:

```
▶ In [2]: 1 def addings(a, b, c):  
          2     return a + b + c  
          3 print(addings(1, 2, 3))  
          4 prm1 = [1, 2, 3]  
          5 print(addings(*prm1))  
          6 prm2 = (1, 2, 3)  
          7 print(addings(*prm2))|  
  
          6  
          6  
          6
```

Конструкция return, возвращающая результат функции обязательна.

# Значение по умолчанию

---

```
▶ In [3]: 1 def addings(a, b = 2, c = 3):  
2         return a + b + c  
3 print(addings(1))  
4 prm1 = [1, 2, 3]  
5 print(addings(*prm1))  
6 prm2 = (1, 2)  
7 print(addings(*prm2))  
  
6  
6  
6
```



# Полезные ссылки и литература

---

1. <https://github.com/profjsb/python-seminar> – Python for Data Science (Seminar Course at UC Berkeley; AY 250).
2. <https://www.python.org/downloads/> – Раздел “Download” на официальном сайте языка Python.
3. <https://docs.python.org/3.12/> – документация на официальном сайте языка Python.
4. <https://www.python.org/dev/peps/pep-0008/> – описание PEP 8.
5. <https://docs.python.org/3.12/library/stdtypes.html#string-methods> – строковые методы в Python.
6. <https://docs.python.org/3.12/library/string.html#formatstrings> – раздел о новом формате строк.
7. [https://docs.python.org/3.12/reference/lexical\\_analysis.html#f-strings](https://docs.python.org/3.12/reference/lexical_analysis.html#f-strings) – раздел о литералах в новом формате.
8. <https://docs.python.org/3.12/library/stdtypes.html#old-string-formatting> – раздел о старом формате строк.
9. <https://docs.python.org/3.12/library/stdtypes.html#sequence-types-list-tuple-range> – документация Python о последовательных типах данных
10. <https://docs.python.org/3.12/library/stdtypes.html#set-types-set-frozenset> – документация Python о множествах
11. <https://docs.python.org/3.12/library/stdtypes.html#mapping-types-dict> – документация Python о словарях