Наука о данных. Python

Лекция 1. Intro

Давайте знакомиться!

Никита Честнов:

- Закончил магистратуру МФТИ,
  кафедра ABBYY, РИОТ
- Аспирант МФТИ,
  кафедра Интеллектуальные системы
- ex-сотрудник ABBYY Lab, Tinkoff
- Преподаватель МФТИ

Давайте знакомиться!

Расскажите о себе:

- Есть ли у вас какой-либо опыт программирования?
- Какими языками программирования вы пользовались?
- Писали ли вы уже на Python?
- Что вы бы хотели научиться в этом курсе?

План курса

Курс “Наука о данных”:

- **блок Python** - 10 недель, лекция + семинар

- блок R

План курса

01. Введение. Знакомство с Python. Способы запуска программ на Python. Основной синтаксис. Работа со стандартным вводом и выводом.
02. Условия и циклы. Контейнеры, итераторы. Модуль collection
03. Работа со строками. Чтение и запись в файлы.
04. Функции (часть 1). Базовый синтаксис. Работа с аргументами. Генераторы.
05. Функции (часть 2). Области видимости. Замыкания. Декораторы.
06. ООП. Основные принципы, определения и базовый синтаксис. Magic-методы.
07. Лучшие практики программирования. Модули и пакеты. Юнит-тестирование.
08. Оптимизация и ускорение кода. Библиотека NumPy.
09. Работа с табличными данными. Библиотека Pandas.
10. Инструменты визуализации. Библиотека Matplotlib.

План курса. Примечания

- может измениться в зависимости от **вашей подготовки** - насколько подробно нужно объяснять базовый материал

- может измениться в зависимости от **ваших пожеланий** - какие темы вы бы хотели изучить

TL;DR - обратная связь приветствуется!

Отчетность на курсе

Виды отчетности:

- тесты - на семинаре по материалам прошлого занятия
- контесты - практика программирования
- проект - создание качественного десктоп-приложения
- лабораторная работа - практика анализа данных

Отчетность на курсе

Что необходимо для работы:

- Яндекс-аккаунт - для контестов
- Google-аккаунт - для тестов и лабораторки (опционально)
- GitHub-аккаунт - для проектов и лабораторной работы
- Telegram - для коммуникации

Вопросы по организации?

Python. Историческая справка

Python разрабатывается с 1989 г. Автор - **Гвидо ван Россум**.

Основные (мажорные) версии:

- Python 1.0 — январь 1994
- Python 2.0 — 16.10.2000
  - Python 2.7.18 - 20.04.2020 (RIP)
- Python 3.0 — 03.12.2008
  - Python 3.10.0 - 04.10.2021

Основные цели:

- взять все лучшее из других языков (ABC, Lisp, Haskell, C, C++)
- нацеленность на ясный синтаксис
  ("Computer Programming for Everybody")

Python. Основные черты

- интерпретируемый (99.9% - CPython)
- объектно-ориентированный - “всё есть объект”
- высокоуровневый язык
- встроенные высокоуровневые структуры данных - меньше писать ручками
- динамическая типизация
- простой синтаксис - легко изучать и читать
- поддержка модулей и пакетов - много бесплатных библиотек
- “универсальный” - используется для многих задач
- интеграция с другими языками - C (Cython), C++, Java (Jython)

Чем Python интересен нам?

- “научный калькулятор”
- универсальность
- интерпретируемость
- читаемость и простота написания

Интерпретируемость

Интерпретируемость

Преимущества компилятора:

- код быстрее (более оптимизирован)
- защита от изменения :)

Недостатки компилятора:

- использует гораздо больше RAM
- защита от изменений :(
- компиляция может занимать много времени
- только под определенную платформу (x86, arm, ...)

Преимущества интерпретатора:

- проще работать с исходным кодом
- минимальный объем RAM

Недостатки интерпретатора:

- каждый запуск = интерпретация, медленнее
- для запуска необходим интерпретатор

Типизация языка

Типизация языка — это то, как язык распознает типы переменных.

Python обладает **сильной динамической неявной типизацией**.

Разберем по пунктам здесь написанное…

Типизация. Сильная / слабая

aka строгая / нестрогая типизации

Показывает можно ли совершать операции с разными типами данных.

В языках со слабой типизацией возможны операции типа:

`число + строка`

Обычно это достигается за счет определенных правил приведения типов, которые применяются неявно.

Примеры:

- Сильная - Perl, Ruby, **Python**
- Слабая - C, C++

Типизация. Статическая / динамическая

При **статической** типизации конечные типы переменных **устанавливаются до выполнения** программы (обычно на этапе компиляции).
Т.е. если вы задали переменной тип *string*, у неё будет только тип *string*.

В **динамически** типизированном языке у переменной могут быть разные типы в разных частях программы, а в статически типизированном,

Примеры:

- Статическая - C, Java, C#
- Динамическая - **Python**, JavaScript, Ruby, Objective-C

Типизация. Явная / неявная

Явно-типизированные языки отличаются тем, что **тип** новых переменных, функций и их аргументов нужно **задавать явно**.

Языки с **неявной** типизацией **перекладывают** эту задачу на компилятор / интерпретатор.

Примеры:

- Явная - C++, C#, Rust
- Неявная - **Python**, PHP, JavaScript

The Zen of Python

Дзен Python — это набор из 19 «руководящих принципов» написания программ, влияющих на структуру языка программирования Python:

```
 1.   Beautiful is better than ugly.
 2.   Explicit is better than implicit.
 3.   Simple is better than complex.
 4.   Complex is better than complicated.
 5.   Flat is better than nested.
 6.   Sparse is better than dense.
 7.   Readability counts.
 8.   Special cases aren't special enough to break the rules.
 9.   Although practicality beats purity.
10.   Errors should never pass silently.
11.   Unless explicitly silenced.
12.   In the face of ambiguity, refuse the temptation to guess.
13.   There should be one-- and preferably only one --obvious way to do it.
14.   Although that way may not be obvious at first unless you're Dutch.
15.   Now is better than never.
16.   Although never is often better than *right* now.
17.   If the implementation is hard to explain, it's a bad idea.
18.   If the implementation is easy to explain, it may be a good idea.
19.   Namespaces are one honking great idea -- let's do more of those!
```

Основы Python

Типы данных в Python

- **базовые типы данных**
- контейнерные типы данных
- остальные классы (в Python тип переменной = класс объекта)

Еще одно разделение типов данных:

изменяемые (mutable) и неизменяемые (immutable)

Базовые типы данных

- int
- float
- complex
- bool
- str
- NoneType

Базовые типы данных. *int*

Целочисленный тип данных в Python (единственный!)

Доступны базовые арифметические операции:

`+`, `-`, `*`, `/`, `**` (степень), `//` (div), `%` (mod)

Feature для удобства - можно использовать разделитель для лучшей читаемости: `1_000_000` вместо `1000000`

Мега-feature - поддерживает длинную арифметику (из коробки)!

```
>>> 15246 ** 100
206819528356229730830562920314095366473179299938671956578169594991374059091885932442141628317549068171254704330255637310947737365625810787931470676597547798437988676066101812206657870709791404351313991521044629794414488948026235717051447318989556383001028872928540254824144122911057588519414395346090252936182900034587504602788354531948243267475080519854244347109529665995702775190459272895289167026214016689133857013760
```

Базовые типы данных. *float*

Тип данных с плавающей точкой - дробные числа (тоже единственный)

Так же поддерживает базовую арифметику.

Точность - double.

Разделитель - **точка** (запятая не сработает!)

Feature для удобства - экспоненциальная запись числа:

`1.5e-4` вместо `0.00015`

Базовые типы данных. *complex*

Да, здесь есть и такое…

Тип данных для комплексных чисел (по сути - 2 float).
Поддерживает базовую арифметику + операции над комплексными числами.

На практике используется очень редко - в научных вычислениях.

Замечание - в качестве символа мнимой единицы используется `j` (не `i`):

```
2 + 3.4j
```

Базовые типы данных. *bool*

Логический тип данных

Имеет 2 значения - `True` и `False` (именно с заглавной буквы!)

Доступны логические операции: `not`, `and`, `or`

Операции сравнения: `<, <=, >, >=, ==, !=`

Приятный бонус - доступны двойные неравенства:

`1 < x <= 4` И `1 < x > 4`

Базовые типы данных. *str*

Строковый тип данных

Может хранить в себе символьные строки произвольной длины. В качестве символов могут быть символы разных кодировок (не только ASCII!).

Выделяются одинарными или двойными кавычками:

```
“Hello World!”
```

или

```
‘Hello World’
```

Базовые типы данных. *NoneType*

Специальный тип данных для объекта None

`None` - специальное значение переменной, означающее "ничего" или "отсутствие значения". Является аналогом `null` из других языков программирования.

Существует много случаев, когда следует использовать None.

Пример - вы хотите выполнить действие, которое может работать либо завершиться неудачно. Используя `None`, вы можете проверить успех действия.

Создание переменной в Python

Оператор “присваивания” - =, но есть нюанс…

В Python по сути нет оператора присваивания в классическом понимании!

…, который всегда выделяет новую ячейку в памяти и записывает в нее значение.

Вместо этого оператор = является **оператором связывания** - связывает имя переменной с адресом объекта в памяти (похоже на ссылки в C++)

Запуск программ на Python

Для написания и запуска программ на Python существует огромное количество инструментов:

- “персональные”
- “облачные”

“Персональные” инструменты

Шаг №0 - установить интерпретатор Python на свой компьютер.

Вариант 1. Командная строка. Режим интерпретатора

Открываете терминал, выполняете `python`, у вас открывается интерпретатор, можете вводить команды

Вариант 2. Командная строка. Запуск написанных программ

Создаете файл с расширением .py, пишете в нем команды.

Для запуска - `python имя_файла.py`

“Персональные” инструменты

Вариант 3. “Продвинутые интерпретаторы”

Хорошие кандидаты:

- ipython
- Jupyter Notebook / Jupyter Lab

“Персональные” инструменты

Вариант 4. IDE на ваш выбор

Хорошие кандидаты:

- **PyCharm**
- VS Code
- Spyder
- vim (неиронично)

“Облачные” инструменты

Google Colab - по сути, Jupyter Notebook в облаке.

Преимущества:

- удобно делиться
- удобно разрабатывать вместе
- можно использовать мощности Google (с ограничениями)

Вопросы?