

# Оглавление

Предисловие.....	14
Кому стоит прочитать эту книгу.....	14
Структура издания.....	15
Примеры кода .....	17
Условные обозначения .....	18
Благодарности.....	19
Об авторе.....	23

## Часть I. Введение в прагматичный ИИ

<b>Глава 1.</b> Что такое ИИ .....	26
Функциональное введение в Python.....	27
Процедурные операторы .....	28
Вывод результатов.....	28
Создание и использование переменных .....	28
Множественные процедурные операторы.....	29
Сложение чисел .....	29
Склеивание строк.....	29
Сложные операторы.....	29
Строки и форматирование строк .....	30
Сложение и вычитание чисел.....	33
Умножение десятичных чисел .....	33
Использование показательных функций.....	33

Преобразование между различными числовыми типами данных .....	34
Округление .....	34
Структуры данных .....	34
Словари .....	35
Списки .....	36
Функции .....	36
Использование управляющих конструкций .....	45
Циклы for .....	45
Циклы while .....	46
Операторы if/else .....	47
Промежуточные вопросы .....	49
Резюме .....	52
<b>Глава 2.</b> ИИ и инструменты машинного обучения .....	55
Экосистема исследования данных языка Python: IPython, Pandas, NumPy, блокнот Jupyter, Sklearn .....	56
Язык R, RStudio, Shiny и ggplot .....	57
Электронные таблицы: Excel и Google Sheets .....	58
Разработка облачных приложений ИИ с помощью веб-сервисов Amazon .....	58
Интеграция разработки и эксплуатации на AWS .....	59
Непрерывная поставка .....	59
Создание среды разработки ПО для AWS .....	60
Настройки проекта Python для AWS .....	63
Интеграция с блокнотом Jupyter .....	67
Интеграция утилит командной строки .....	70
Интеграция AWS CodePipeline .....	74
Основные настройки Docker для исследования данных .....	79
Другие сервисы сборки: Jenkins, CircleCI и Travis .....	80
Резюме .....	80
<b>Глава 3.</b> Спартанский жизненный цикл ИИ .....	82
Прагматическая петля обратной связи при промышленной эксплуатации .....	83
AWS SageMaker .....	87
Петля обратной связи AWS Glue .....	89
AWS Batch .....	93
Петли обратной связи на основе Docker .....	95
Резюме .....	97

## Часть II. ИИ в облаке

<b>Глава 4.</b> Разработка ИИ в облачной среде с помощью облачной платформы Google.....	100
Обзор GCP .....	101
Colaboratory .....	102
Datalab .....	105
Расширяем возможности Datalab с помощью Docker и реестра контейнеров Google.....	105
Запуск полнофункциональных машин с помощью Datalab .....	106
BigQuery .....	108
Облачные сервисы ИИ компании Google .....	111
Тензорные процессоры Google и TensorFlow .....	115
Резюме.....	118
<b>Глава 5.</b> Разработка ИИ в облачной среде с помощью веб-сервисов Amazon .....	119
Создание решений дополненной и виртуальной реальностей на основе AWS.....	122
Компьютерное зрение: создание конвейеров AR/VR с помощью EFS и Flask.....	122
Создание конвейера инженерии данных с помощью EFS, Flask и Pandas.....	125
Резюме.....	141

## Часть III. Создание реальных приложений ИИ с нуля

<b>Глава 6.</b> Прогноз популярности в соцсетях в НБА.....	144
Постановка задачи .....	145
Сбор данных .....	145
Получение данных из труднодоступных источников .....	168
Получение данных о просмотрах страниц «Википедии» спортсменов .....	168
Получение данных о вовлеченности в Twitter спортсменов.....	173
Изучаем данные об игроках НБА .....	176
Машинное обучение без учителя для данных об игроках НБА .....	181
Построение фасетного графика по игрокам НБА на языке R.....	182
Собираем все воедино: команды, игроков, социальный авторитет и рекламные отчисления.....	183
Дальнейшие прагматичные шаги и учебные материалы .....	187
Резюме.....	187

<b>Глава 7.</b> Создание интеллектуального бота Slack в AWS .....	188
Создание бота .....	188
Преобразование библиотеки в утилиту командной строки .....	189
Выводим бот на новый уровень с помощью сервиса AWS Step Functions .....	191
Настройка учетных данных IAM .....	193
Завершение создания пошаговой функции .....	203
Резюме .....	205
<b>Глава 8.</b> Извлечение полезной информации об управлении проектами из учетной записи GitHub-организации .....	206
Обзор проблем, возникающих при управлении программными проектами .....	206
Создание исходного каркаса проекта исследования данных .....	209
Сбор и преобразование данных .....	211
Обработка GitHub-организации в целом .....	213
Формирование предметно-ориентированной статистики .....	214
Подключение проекта по исследованию данных к интерфейсу командной строки .....	216
Исследование GitHub-организаций с помощью блокнота Jupiter .....	218
Изучаем метаданные файлов проекта CPython .....	221
Изучаем файлы, удаленные из проекта CPython .....	225
Развертывание проекта в каталоге пакетов Python .....	228
Резюме .....	231
<b>Глава 9.</b> Динамическая оптимизация виртуальных узлов EC2 в AWS .....	232
Выполнение заданий на платформе AWS .....	232
Спотовые виртуальные узлы .....	232
Теория спотовых виртуальных узлов и история цен на них .....	233
Создание утилиты и блокнота для сравнения цен на спотовые виртуальные узлы на основе машинного обучения .....	236
Написание модуля запуска спотового виртуального узла .....	243
Написание более сложного модуля запуска для спотового виртуального узла .....	250
Резюме .....	252
<b>Глава 10.</b> Недвижимость .....	254
Исследование цен на недвижимость в США .....	254
Интерактивная визуализация данных в Python .....	257
Кластеризация по порядку размера и цене .....	260
Резюме .....	269

<b>Глава 11.</b> Промышленная эксплуатация ИИ для пользовательского контента....	270
Получившее премию Netflix решение не было внедрено в промышленную эксплуатацию .....	271
Ключевые понятия рекомендательных систем .....	272
Использование фреймворка Surprise в языке Python.....	273
Облачные решения для создания рекомендательных систем .....	276
Проблемы, возникающие на практике при работе с рекомендациями .....	277
Облачный NLP и анализ тональности высказываний.....	282
NLP на платформе Azure.....	283
NLP на платформе GCP.....	286
Изучаем API сущностей .....	287
Бессерверный конвейер ИИ промышленного уровня для NLP на платформе AWS.....	290
Резюме .....	295

## Приложения

<b>Приложение А.</b> Аппаратные ускорители для ИИ.....	298
<b>Приложение Б.</b> Выбор размера кластера .....	300



# 1

## Что такое ИИ

Не путайте процесс с результатом.

*Джон Вуден  
(John Wooden)*

Если вы взяли в руки данную книгу, то, вероятно, вас интересует прагматичный ИИ. Книг, курсов и вебинаров по современным методам машинного и глубокого обучения вполне достаточно. Не хватает лишь информации о том, как довести проект до состояния, при котором появляется принципиальная возможность использования всех продвинутых методов. Именно для этого предназначена книга — чтобы сократить разрыв между теорией и практической реализацией связанных с искусственным интеллектом проектов.

Во многих случаях обучить свою модель невозможно по причине простой нехватки времени, ресурсов и навыков для реализации. Всегда есть лучшее решение, чем двигаться по заведомо провальному пути. А практикующие прагматичный ИИ всегда используют оптимальную для ситуации методику. В некоторых случаях это может означать вызов программного интерфейса приложения (application programming interface, API) с предварительно обученным методом. Еще одна прагматичная методика ИИ состоит в создании намеренно менее эффективной модели для упрощения понимания и развертывания в промышленной эксплуатации.

В 2009 г. Netflix объявил знаменитый конкурс, предложив приз \$1 млн команде разработчиков, которая сможет повысить точность рекомендаций компании на 10 %. Конкурс был очень увлекательным и опередил свое время в отношении науки о данных. Не многие знают, однако, что победивший алгоритм так и не был реализован вследствие потенциально больших затрат на разработку

и внедрение (<https://www.wired.com/2012/04/netflix-prize-costs/>). Вместо него были использованы некоторые алгоритмы команды, достигшей улучшения результатов «лишь» на 8,43 %. Это прекрасное подтверждение того, что подлинная цель многих задач ИИ — практичность, а вовсе не идеальный результат.

Данная книга уделяет основное внимание тому, как оказаться той самой командой со второго места, чье решение на самом деле попадет в промышленную эксплуатацию. Это связующая нить всей книги, чья цель — перевод кода в промышленную эксплуатацию, а не получение им формального титула «лучшее решение», которое никогда не станет программным продуктом.

## Функциональное введение в Python

Python — потрясающий язык программирования, способный на множество различных вещей. Есть мнение, что Python ни в чем не демонстрирует исключительных результатов, но достаточно хорош в абсолютном большинстве приложений. Подлинная сильная сторона этого языка — преднамеренное отсутствие сложности. Python допускает также программирование в множестве разных стилей. Его вполне можно использовать для выполнения операторов в процедурном стиле, построчно. Но можно также и в качестве сложного объектно-ориентированного языка программирования, обладающего такими продвинутыми возможностями, как метаклассы и множественное наследование.

При изучении языка Python, особенно в контексте науки о данных, на некоторых его частях можно не заострять внимания. Можно даже сказать, что многие его составные части, особенно некоторые объектно-ориентированные возможности, никогда не используются при написании блокнотов Jupiter. Вместо этого имеет смысл использовать альтернативный подход с упором на функции. Данный раздел вкратце познакомит вас с языком Python, который можно рассматривать как новый Microsoft Excel.

Один из недавних моих аспирантов сказал мне, что до моего курса машинного обучения его беспокоил сложный вид соответствующего кода. Но после нескольких месяцев работы с языком Python и блокнотами Jupiter он почувствовал уверенность при использовании Python для решения задач науки о данных. Я убежден, исходя из виденного мной во время преподавания, что любой пользователь Excel может научиться использовать блокноты Jupiter на Python.



Стоит отметить, что в развертывании кода в промышленной эксплуатации Jupyter может играть роль механизма доставки, а может и не играть. В последнее время стали очень популярны новые фреймворки, вроде Databricks, SageMaker и Datalab, создающие возможности введения в промышленную эксплуатацию с помощью Jupyter, но чаще всего Jupyter используется для выполнения различных экспериментов.

## Процедурные операторы

Для работы со следующими примерами у вас должен быть установлен Python версии не ниже 3.6. Скачать последнюю версию Python можно по адресу <https://www.python.org/downloads/>. Процедурные операторы фактически представляют собой операторы, допускающие построчное выполнение. Ниже перечислены типы выполняемых процедурных операторов.

- ❑ Блокнот Jupyter.
- ❑ Командная оболочка IPython.
- ❑ Интерпретатор Python.
- ❑ Сценарии Python.

## Вывод результатов

Вывод результатов в Python очень прост. Функция `print` получает входные данные и выводит их в консоль:

```
In [1]: print("Hello world")
...:
Hello world
```

## Создание и использование переменных

Переменные создаются путем присваивания. В следующем примере переменной присваивается значение, после чего она выводится в консоль посредством объединения двух операторов через точку с запятой. Стиль с подобным использованием точек с запятой можно часто встретить в блокнотах Jupyter, но в коде и библиотеках, предназначенных для промышленной эксплуатации, на него обычно смотрят с неодобрением.

```
In [2]: variable = "armbar"; print(variable)
armbar
```

## Множественные процедурные операторы

Полное решение задачи может представлять собой обычный процедурный код, показанный ниже. Такой стиль уместен в блокнотах Jupiter, но редко встречается в коде, предназначенном для промышленной эксплуатации.

```
In [3]: attack_one = "kimura"
...: attack_two = "arm triangle"
...: print("In Brazilian jiu-jitsu a common attack is a:", attack_one)
...: print("Another common attack is a:", attack_two)
...:
In Brazilian jiu-jitsu a common attack is a: kimura
Another common attack is a: arm triangle
```

## Сложение чисел

Python можно использовать и в качестве калькулятора. Прекрасный способ привыкнуть к языку — начать использовать его вместо Microsoft Excel или приложения-калькулятора.

```
In [4]: 1+1
...:
Out[4]: 2
```

## Склеивание строк

Можно складывать и строки:

```
In [6]: "arm" + "bar"
...:
Out[6]: 'armbar'
```

## Сложные операторы

Можно создавать и более сложные операторы — с использованием структур данных вроде переменной `belt`, представляющей собой список:

```
In [7]: belts = ["white", "blue", "purple", "brown", "black"]
...: for belt in belts:
...:     if "black" in belt:
...:         print("The belt I want to earn is:", belt)
...:     else:
...:         print("This is not the belt I want to end up with:", belt)
```

```
...:
This is not the belt I want to end up with: white
This is not the belt I want to end up with: blue
This is not the belt I want to end up with: purple
This is not the belt I want to end up with: brown
The belt I want to earn is: black
```

## Строки и форматирование строк

*Строки* (strings) — последовательности символов. Они часто форматируются программно. Практически все программы на языке Python применяют строки для отправки сообщений пользователям. Однако важно хорошо представлять себе несколько базовых понятий.

- ❑ Строки можно создавать с помощью одинарных, двойных и тройных/двойных кавычек.
- ❑ Строки можно форматировать.
- ❑ Строки могут находиться в нескольких кодировках, включая Unicode.
- ❑ Существует множество методов работы со строками. В редакторе или командной оболочке IPython можно увидеть список этих методов, выделенных с помощью автозаполнения табуляцией.

```
In [8]: basic_string = ""
```

```
In [9]: basic_string.
        capitalize()  encode()          format()
isalpha()  islower()    istitle()      lower()
           caselfold()  endswith()    format_map()
isdecimal() isnumeric()  isupper()     lstrip()
           center()    expandtabs()  index()
isdigit()  isprintable() join()         maketrans() >
           count()    find()       isalnum()
isidentifier() isspace()   ljust()      partition()
```

### Простейший тип строки

Простейший тип строки — переменная, которой присвоено значение в виде фразы в кавычках. Кавычки могут быть тройными, двойными или одинарными.

```
In [10]: basic_string = "Brazilian jiu-jitsu"
```

## Разбиение строк

Строку можно превратить в список, разбив ее по пробелам или другим символам:

```
In [11]: #Разбиение по пробелам (по умолчанию)
...: basic_string.split()
Out[11]: ['Brazilian', 'jiu-jitsu']
```

```
In [12]: #Разбиение по дефисам
...: string_with_hyphen = "Brazilian-jiu-jitsu"
...: string_with_hyphen.split("-")
...:
Out[12]: ['Brazilian', 'jiu-jitsu']
```

## Все заглавные буквы

В Python есть множество удобных встроенных функций для преобразования строк. Вот так, например, можно преобразовать все символы в строке в верхний регистр:

```
In [13]: basic_string.upper()
Out[13]: 'BRAZILIAN JIU JITSU'
```

## Срезы строк

Строки можно разрезать на части, указывая на длину:

```
In [14]: #Получить первые два символа строки
...: basic_string[:2]
Out[14]: 'Br'
In [15]: #Получить длину строки
...: len(basic_string)
Out[15]: 19
```

## Склеивание строк

Строки можно склеивать путем конкатенации или присваивания строки переменной с последующим прибавлением к ней строк. Такой стиль приемлем и интуитивно понятен для блокнотов Jupiter, но из соображений производительности лучше использовать в коде для промышленной эксплуатации f-строки.

```
In [16]: basic_string + " is my favorite Martial Art"
Out[16]: 'Brazilian jiu-jitsu is my favorite Martial Art'
```

## Сложное форматирование строк

Один из лучших способов форматирования строк в современном Python 3 — использование f-строк:

```
¶ In [17]: f'I love practicing my favorite Martial Art,
           {basic_string}'
...:
Out[17]: 'I love practicing my favorite Martial Art,
         Brazilian jiu-jitsu'
```

## Для обертывания строк можно использовать тройные кавычки

Иногда может понадобиться взять фрагмент текста и присвоить его переменной. Проще всего сделать это в Python с помощью заключения фразы в тройные кавычки:

```
In [18]: f"""
...: This phrase is multiple sentences long.
...: The phrase can be formatted like simpler sentences,
...: for example, I can still talk about my favorite
...: Martial Art {basic_string}
...: """
Out[18]: '\nThis phrase is multiple sentences long.\nThe phrase
can be formatted like simpler sentences,\nfor example,
I can still talk about
my favorite Martial Art Brazilian jiu-jitsu\n'
```

## Разрывы строк можно удалить с помощью метода Replace

Вышеприведенная длинная строка содержала символы разрывов строки — символы `\n`, которые можно удалить с помощью метода `replace`:

```
In [19]: f"""
...: This phrase is multiple sentences long.
...: The phrase can be formatted like simpler sentences,
...: for example, I can still talk about my favorite
...: Martial Art {basic_string}
...: """.replace("\n", "")
Out[19]: 'This phrase is multiple sentences long. The phrase can be
formatted like simpler sentences, for example, I can still talk about
my favorite Martial Art Brazilian jiu-jitsu'
```

## Числа и арифметические операции

В Python есть встроенный калькулятор. Он способен выполнять множество простых и сложных арифметических операций без всяких дополнительных библиотек.

## Сложение и вычитание чисел

Язык Python также демонстрирует гибкость в форматировании фраз на основе f-строк:

```
In [20]: steps = (1+1)-1
...: print(f"Two Steps Forward: One Step Back = {steps}")
...:
Two Steps Forward: One Step Back = 1
```

## Умножение десятичных чисел

В язык Python также встроена поддержка десятичных чисел, что упрощает решение арифметических задач:

```
In [21]:
...: body_fat_percentage = 0.10
...: weight = 200
...: fat_total = body_fat_percentage * weight
...: print(f"I weight 200lbs, and {fat_total}lbs of that is fat")
...:
I weight 200lbs, and 20.0lbs of that is fat
```

## Использование показательных функций

С помощью библиотеки `math` можно легко вычислить, например, 2 в третьей степени:

```
In [22]: import math
...: math.pow(2,3)
```

```
Out[22]: 8.0
```

Другой способ:

```
>>> 2**3
```

```
8
```