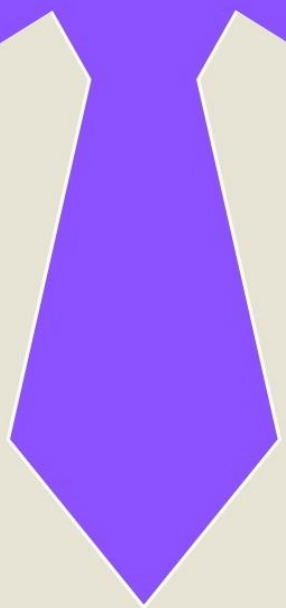
Телюк Максим

Введение в **Python:**

от переменных до функций



Учись. Пиши. Запускай.

Открой мир кода с Python

Телюк Максим

Введение в Python: от переменных до функций

Посвящение

Дорогой читатель!

Вы держите в руках пособие, которое станет вашим проводником в мир программирования. Python — это не просто язык кода, а инструмент, открывающий двери в увлекательную вселенную технологий.

Это пособие создано для тех, кто делает первые шаги в программировании. Здесь не будет сложных терминов и непонятных правил — только ясные объяснения, практические примеры и задания, которые помогут тебе освоить Python с нуля.

Введение

Приветствую вас в увлекательном путешествии по изучению Python — языка программирования, который открывает двери в мир технологий! Это пособие станет вашим надежным проводником: мы начнем с самых азов и постепенно освоим все необходимые навыки для уверенной работы с Python. Даже если вы никогда не программировали раньше — не переживай! Python создан специально для того, чтобы обучение было простым и увлекательным.

Краткая история

Python появился в конце 1980-х годов благодаря голландскому программисту Гвидо ван Россуму. В то время он работал в Нидерландах над одним проектом и хотел создать простой и понятный язык, который бы экономил время разработчиков. Первая версия Python 0.9.0 вышла в 1991 году — это был настоящий прорыв, потому что язык сочетал мощь и простоту.

Интересно, что название Python никак не связано со змеёй. Гвидо был фанатом британского комедийного шоу «Monty Python's Flying Circus» и назвал язык в его честь. Именно поэтому в официальной документации и сообществе часто встречаются отсылки к этому шоу — например, стандартные примеры используют фразу «spam and eggs» из скетча.

Python разрабатывался как язык общего назначения, но особое внимание уделялось читаемости кода. Гвидо придумал правило «There should be one — and preferably only one — obvious way to do it» («Должен быть один — и желательно только один — очевидный способ сделать это»). Благодаря этому код на Python выглядит аккуратно и понятно даже для новичков.

Сегодня Python — один из самых популярных языков в мире. Его используют в Google, NASA, YouTube и даже в искусственном интеллекте! Каждый год выходят новые версии, но принципы, заложенные Гвидо, остаются неизменными: простота, удобство и открытость для всех.

Почему стоит учить Python?

1. Простота и понятность

Код на Python выглядит почти как обычный английский, с минимумом сложных символов. Например, цикл выглядит так:

```
for i in range(5):
    print("Привет, мир!")
```

---

2. Универсальность

Python используется в самых разных сферах:

- Веб-разработка (Django, Flask)
- Наука и анализ данных (Pandas, NumPy)
- Искусственный интеллект и машинное обучение (TensorFlow, PyTorch)
- Автоматизация и скрипты (например, для обработки файлов)
- Разработка игр (Pygame)
- Микроконтроллеры и IoT (MicroPython)

3. Большое сообщество и обилие ресурсов

Одно из главных преимуществ Python — его огромное и дружелюбное сообщество. Миллионы разработчиков по всему миру используют этот язык, создают проекты и помогают новичкам.

Что ждет вас в этом пособии?

Этот курс построен по принципу «от простого к сложному». Вы начнёте с основ синтаксиса и постепенно перейдёте к более сложным заданиям. В каждой главе вас ждет:

- Теория с понятными примерами
- Практические задания
- Чек-листы

Работа с пособием

При работе с пособием мне бы хотелось дать несколько рекомендаций, которые помогут сделать ваше обучение более интересным и качественным.

1. Не бойся ошибиться при написании кода

Боязнь ошибиться присуща каждому в начале обучения, но вы должны понимать, что ошибки — это опыт. Ошибаются все программисты и даже профессионалы. Ошибки — это не провал, а часть пути. Чем чаще вы допускаете ошибки (и исправляете их), тем быстрее станете уверенным программистом.

2. Возьми карандаш в руки

Я советую вам взять карандаш и помечать различные структуры, которые вам не понятны или пока, что не запоминаются. С течением времени осознание прейдет в любом случае.

3. Попробуйте сломать код или дополнить

В исходном коде можно по желанию менять значения и анализировать исходный результат. Эта практика даст понимание структуры, а также поможет закрепить полученные знания.

Глава 1: Первые шаги

Урок 1.1: Установка Python

Урок 1.1 посвящен установке Python. Для того, чтобы облегчить этот этап я записал видео-материал, где показан путь установки Python. Чтобы перейти на видео достаточно отсканировать QR-code на странице.

1. Открыть официальный сайт python.org
2. Нажать на вкладку Downloads
3. Нажать желтую кнопку «Скачать»
4. В открывшемся окне начать установку
5. При установке отметить галочку «Add to PATH» (это позволит запустить Python из командной строки)
6. После завершения установки открыть командную строку
7. Вписать «python --version» (для проверки)
8. Установка завершена!

Если у вас получилось установить Python, то вы большой молодец!

Урок 1.2: Установка профессиональной среды разработки PyCharm

Среда разработки (англ. IDE — Integrated Development Environment) — это специальная программа, которая помогает писать код быстрее и удобнее. Представьте, что это как умный блокнот для программиста, но с кучей полезных функций.

Среды разработки бывают разными, но остановимся на самых лучших:

1. PyCharm

- Самый популярный выбор.
- Есть бесплатная версия (Community).
- Выглядит сложно, но очень умная.

2. Visual Studio Code (VS Code)

- Лёгкая и бесплатная.
- Нужно ставить дополнения для Python (но это просто).

3. IDLE

- Простейшая IDE, которая идёт в комплекте с Python.
- Подходит для первых шагов, но без удобных фишек.

В силу узко-направленности и более подходящей к языку программирования Python, я советую установить PyCharm. Он имеет больше полезных особенностей на дистанции.

Для успешной установки PyCharm можно перейти по QR-code ниже, и в формате видео ознакомится с установкой.

Обратите внимание, что во время установки желательно поставить галочку на пункте «Create associations .py» — это позволит создавать файлы в формате «py» для отличия Python файлов от обычных файлов.

После всех этапов установки, можно открыть PyCharm, создать новый проект, открыть Python файл и приступить к написанию первой команды!

Первая команда — «Привет мир!»

Для написания первой команды, требуется всего лишь написать в редакторе кода — **Print(“Hello world”)**. Команда — **Print()**, функция которая выводит на экран определенное количество значений, прописанных через запятую. После этого в **компиляторе** появится фраза — Hello world!. Просто, не так ведь? Но я пообещал, что в пособие не будут упоминаться сложные термины. Поэтому объясню, что такое компилятор.

Компилятор — переводчик для компьютера. Представьте, что вы пишите письмо другу на русском языке, но друг понимает только китайский. Чтобы он прочитал ваше сообщение, вам нужен переводчик. Компилятор переводит написанный вами код в машинный язык — набор нулей и единиц. После, компьютер принимает информацию, и только после этого выдает некий результат, будь то ошибка или правильно воспроизведенный код.

Перед тем как идти дальше, мне необходимо сообщить вам о том, что при переходе по QR-code ниже, можно попасть на веб-сервис GitHub. Оттуда, можно скачать весь исходный код из пособия.

GitHub — это социальная сеть для IT-специалистов, в которой можно найти проекты с открытым кодом от других разработчиков. Также, вы имеете возможность практиковаться в написании кода и хранить своё портфолио.

Урок 1.3: Переменные и типы данных

Переменные

Переменные в языке программирования Python — являются одновременно одной из самых легких тем, а также самых важных. Что же такое переменная?

Переменная — это некая коробка или ячейка памяти, в которую вы можете поместить, абсолютно все, что угодно, от какой-либо строки или цифры, до целого списка значений. В любой момент времени вы с легкостью можете:

- Узнать, что лежит в составе переменной
- Заменить содержимое переменной
- Использовать ее значение в вычислениях

---

age = 15 # Положили в коробку "age" число 15

name = "Анна" # А в коробку "name" — строку "Анна"

print(name, "тебе", age, "лет") # Анна тебе 15 лет

Для создания переменной необходимо после названия поставить =, а далее ввести то, чтобы вы хотели хранить в ней.

Правила именования переменных

Очень важно правильно прописывать переменные, а для этого и существуют всеобщие правила.

Разрешено при создании переменных:

- Писать английские буквы и цифры
- Писать нижнее подчеркивание _

Учтите, что регистр (режим, в котором буквы становятся заглавными или строчными) имеет значение!

---

name ≠ Name ≠ NAME # Это три разные переменные!

Запрещено при создании переменных (ошибка):

- Ставить специальные символы в начале переменной (&, $, ↑ и т.д.)
- Ставить цифры
- Ставить пробелы между словами
- Называть переменные как команды в Python

Обычно в Python, люди используют метод **snake_case**. При помощи этого метода, переменные в прочтении становятся наиболее читаемыми для людей. В его основу входит формат, при котором люди вместо пробела между словами ставят нижнее подчеркивание, названия пишут при помощи строчных букв.

---

first_place = "Яна"

first_name = "Макс"

print_value = 10

Советую привыкать с самого начала прописывать осмысленные переменные, то есть переменные в основу, которых входит какое-то смысловые значение. Например:

Плохо: x = 10

Хорошо: score = 10

Читаемость очень важна в написании проектов: другие программисты (или ты через месяц) поймут, что значит, «user_age», но не поймут «ua». Напомню, что переменная — это некая коробка с именем, куда можно положить данные. Называй её понятно и по правилам!

Типы данных

В Python у каждой переменной есть свой тип данных — это как "разновидность" информации, с которой можно работать. Ознакомимся с основными типами на начальном пути:

- Целые числа — int
- Дробные числа — float
- Строки — str
- Булевы — bool

```
age = 15                        # int
temperature = 36.6              # float
message = 'Привет, мир!'        # str
is_raining = False              # bool
passed_exam = True              # bool
```

Тип данных — целые числа, указывается в переменных без каких-либо отличительных свойств. Достаточно написать число.

Тип данных — дробные числа, указывается с точкой, не с запятой!

Тип данных — строчка, указывается в кавычка, в одинарных или двойных, без разницы (главное, чтобы кавычки в начале и конце совпадали.

Тип данных — булевый, это тип данных, который умеет хранить только два значения — «True» или «False» (прошу обратить внимание что «True» и «False» пишутся с заглавной буквы).

Рассмотрим операции с числами.

Всего существует 7 основных операций с числами (float и int).

1. Сложение — +
2. Вычитание — -
3. Умножение — *
4. Деление — /
5. Целочисленное деление — //
6. Нахождения остатка от деления — %
7. Возведение в степень — **

---

n_1 = 1+1 # Сложение, Ответ: 2

n_2 = 1-1 # Вычитания, Ответ: 0

n_3 = 5*2 # Умножения, Ответ: 10

n_4 = 4.5/2 # Деление, Ответ: 2.25

n_5 = 10//3 # Целочисленное деление, Ответ: 3

n_6 = 15%4 # остаточное деление, Ответ: 3

n_7 = 53** # Возведение в степень, Ответ: 125

Советую открыть PyCharm и попробовать самому провести взаимодействия с разными числами.

Функции для чисел

1. abs(n) — модуль числа n
2. min(x, y) — Поиск наименьшего числа среди x, y и т.д.
3. max(x, y) — Поиск наибольшего числа среди x, y т.д.
4. round(x/y) — округление при деление в большую сторону
5. pow(x, y) — возведение числа x в степень y

---

abs(-1) # Число в модуле, Ответ: 1

min(10,1) # Поиск наименьшего числа, Ответ: 1

max(10,1) # Поиск наибольшего числа, Ответ: 10

round(5/3) # округление при деление в большую сторону

Ответ: 2

pow(5, 3) # Возведение в степень, Ответ: 125

Рассмотрим операции с текстом

Всего есть две операции с текстом:

1. Конкатенация (склеивание строк)
2. Умножение (воспроизведение строки определенное количество раз)

---

string_1 = 'Hi ' + 'Maxim' #Конкатенация

string_2 = 'Hellow world!' * 3 #Умножение

Желательно также открыть PyCharm и осмыслить как это происходит.

Динамическая типизация

Слово динамическая типизация звучит страшно, но на самом деле, это легко. Представьте, что в определенный момент вам необходимо поменять один тип данных на другой. К примеру, число десять превратить в строчку. И при помощи функций вы с легкостью это можете сделать. Это и называется динамической типизацией, когда вы задаете или меняете тип данных элемента.

С помощью Динамической типизации можно быстро менять тип данных, но нужно быть внимательным, ведь, к примеру, **булевы** нельзя складывать с **строчками**.

---

type(1) #проверка типа данных

Ответ: int

str(10) #смена типа данных на str

Ответ: “10”

float(12) # смена типа данных на float

Ответ: 12.0

bool('1') # проверка на наличие чего либо в кавычках

Ответ: 1

bool('')

Ответ: 0

Функция Input()

При помощи функции input(), мы можем требовать от пользователя программы ввод неких данных. Для того, чтобы пользоваться данными, нужно чтобы они где-то хранились — в переменной. Внутри скобок можно прописать строку с какой-либо просьбой к пользователю. Исходный тип данных у функции input() — str. Это вся информация, которая нужна для выполнения простейших программ.

---

password = input('Назовите пароль: ')

Тип данных str

password = int(input('Назовите пароль: '))

Тип данных int

Форматирование строк

Чтобы постоянно не писать в функции print() какие-либо переменные через запятую, можно воспользоваться методом форматирования строк. Достаточно в функции print() перед кавычками поставить латинскую букву f, а внутри переменной проставить фигурные скобки, внутри них прописать переменные. К примеру:

---

print(f”Привет {name}”)

print(”Привет ”, name)

#Одно и тоже, но по структуре разное

Бонус (#комментарий)

Комментарий в Python служит в целях пояснения чего-либо. Для его написания необходимо поставить решетку в любой части программы и написать пояснение. В таком случае программа не обработает часть кода после решетки.

Задание:

Откройте исходный код по названию «Chapter_1, Task_1» на GitHub. Попробуйте видоизменить различные значения или добавить свои.

Check-list

Что я должен знать и уметь (приготовь карандаш):

1. Переменные

- [ ] Знаю, что переменная — это "коробка" для хранения данных.
- [ ] Умею создавать переменные (x = 5).
- [ ] Понимаю, что имя переменной не может начинаться с цифры и содержать пробелы.
- [ ] Знаю, что = (равно) это присваивание, а не равенство.

2. Типы данных

- [ ] Различаю основные типы
- [ ] Понимаю, что "5" (строка) и 5 (число) — это разные вещи.
- [ ] Умею преобразовывать типы

3. Ввод данных input()

- [ ] Умею запрашивать данные у пользователя
- [ ] Знаю, что input() всегда возвращает строку (str), даже если ввели число.
- [ ] Умею преобразовывать ввод в числа

4. Вывод данных print()

- [ ] Умею выводить текст и переменные
- [ ] Знаю, как использовать f-строки для удобного вывода

5. Операции с числами

- [ ] Знаю основные арифметические операции
- [ ] Понимаю разницу между / и //
- [ ] Умею комбинировать операции

6. Простые программы

- [ ] Могу написать программу, которая

  - Спрашивает имя и возраст, затем выводит приветствие.
  - Считает сумму, разность или произведение двух чисел.
  - Генерирует смешной текст из введённых слов (как в задании "Случайная история").

Глава 2: Управляющие конструкции

Урок 2.1: Условные операторы

Условные операторы — логика программы. Представьте, что программа — это дорога, а условный оператор — это светофор, который решает, по какой дороге ехать дальше. Самая простая идея: "Если выполняется условие — делаем одно, иначе — другое". Приведу пример из жизни: если на улице дождь, то нужно взять зонт, иначе оставить зонт дома. В Python это выглядит даже легче, чем читается! Давайте разберемся с этой темой.

Оператор if (если)

Самый простой вариант - выполнить код, если условие истинно:

```
if условие:
    # код, который выполнится, если условие True
    print("Условие истинно")
```

Обрати внимание, что мы используем ключевое слово «if», после прописываем условие, а далее ставим двоеточие. После всего, ниже с отступом в 4 пробела (одну табуляцию) прописываем код, который должен выполняться если заданное условие выполняется. Все, что идет после двоеточия называется — блок кода.

```
age = 16
if age >= 16:
    print("Вы можете получить права")
```

В данном случае мы проверяем возраст. Если переменная больше или равна 16, то «Вы можете получить права».

Условия задаются при помощи четырех знаков:

1. Равенство — == (два знака ровно)
2. Неравенствро — !=
3. Больше, меньше равно — >=, <=
4. Больше, меньше — >, <

Оператор if-else (если-иначе)

Добавляем действие, когда условие ложное:

```
if условие:
    # код при истинности
else:
    # код при ложности
```

В это случае после «else» нет необходимости, что-либо писать. Блок кода в «else» выполниться при условии, что изначальное условие будет ложным.

```
temperature = 25
if temperature > 20:
    print("На улице тепло")
else:
    print("На улице холодно")
```

Оператор if-elif-else (если-иначе если-иначе)

Данный оператор — «elif» был создан для дополнительной проверки.

```
if условие1:
    # код 1
elif условие2:
    # код 2
else:
    # код, если ни одно условие не сработало
```

```
grade = 85
if grade >= 90:
    print("Отлично!")
elif grade >= 70:
    print("Хорошо")
elif grade >= 50:
    print("Удовлетворительно")
else:
    print("Неудовлетворительно")
```

Особенности Python

1. Отступы - в Python блоки кода выделяются отступами (обычно 4 пробела)

2. Двоеточие - после условия всегда ставится двоеточие

3. Логические операторы:

- and (и)
- or (или)
- not (не)

Логические операторы это способ задавать дополнительные условия.

```
age = 17
has_ticket = True

if age >= 16 and has_ticket:
    print("Вы можете посетить концерт")
else:
    print("Вы не можете посетить концерт")
```

В данном условие прописано, что если «age» больше или равно 16 и при этом «has_ticket» будет равно True, то «Вы можете посетить концерт».

Условные операторы - основа программирования, поэтому важно хорошо их освоить!

Задание:

Попробуйте написать программу, которая:

1. Запрашивает у пользователя число
2. Проверяет, является ли число положительным, отрицательным или нулём
3. Выводит соответствующее сообщение

Урок 2.2: Циклы

Циклы в Python — как повторяющиеся действия. Они нужны для того, чтобы выполнять одно и то же действие много раз без повторения кода. Представьте, что тебе нужно:

- Присесть 10 раз
- Перебрать все элементы списка
- Повторять вопрос, пока пользователь не даст правильный ответ

Для разных целей всего существует два цикла: «for» и «while».

Цикл for

Цикл for используется в случае, когда мы знаем количество итераций (повторений блока кода). Такой цикл работает по принципу «в наборе данных сделай что-то»

---

for переменная in последовательность:

    # действия повторяются для каждого элемента

Примеры:

- С числами при помощи функции range:

Вывести числа от 1 до 5

for i in range(1, 6): # range(старт, стоп+1)

    print(i)

Range() — n количество итераций, от заданного значения x и до заданного y значения не включительно

- Со строками:

Перебрать все буквы в слове

word = "Привет"

for letter in word:

**   print(letter)**

Цикл while

Цикл while — цикл, при котором блок кода будет выводится до тех пор, пока условие истинно.

---

while условие:

    # выполняем код, пока условие True

Примеры:

- Счетчик

---

count = 0

while count < 5:

**   print(count)**

**   count += 1** # увеличиваем count на 1

- Проверка ввода:

```
password = ""
while password != "12345":
    password = input("Введите пароль: ")
print("Добро пожаловать!")
```

- Бесконечный цикл (Будьте бдительны!)

```
while True:
    print("Этот цикл будет работать вечно! Нажмите Ctrl+C для остановки")
```

Во избежание бесконечного цикла, можно использовать ключевое слово **break**

Полезные команды для циклов:

- break — немедленно выйти из цикла
- continue — перейти к следующей итерации, пропуская остаток кода
- else — выполняется, если цикл завершился нормально (без break)

```
for num in range(10):
    if num == 5:
        break  # выход при num=5
    print(num)  # выведет 0 1 2 3 4
```

Задание:

Напишите программу, которая выводит все чётные числа от 1 до 20

Check-list

Что я должен знать и уметь (приготовь карандаш):

1. Условные операторы (if, elif, else)

- [ ] Понимаю синтаксис:
- [ ] Умею сравнивать числа и строки
- [ ] ==, !=, >, <, >=, <=
- [ ] and, or, not
- [ ] Могу написать программу с несколькими условиями (например, определение знака числа, проверка возраста).
- [ ] Знаю, что elif и else необязательны.

2. Цикл while

- [ ] Понимаю, что while выполняется, пока условие True.
- [ ] Умею писать циклы с условием
- [ ] Знаю, как избежать бесконечного цикла.
- [ ] Могу использовать break для выхода из цикла и continue для перехода к следующей итерации.

3. Цикл for

- [ ] Знаю, что for используется для перебора последовательностей.
- [ ] Умею работать с range()
- [ ] Могу перебирать строки и списки

Глава 3: Работа с данными

Урок 3.1: Списки

Списки — это упорядоченные коллекции элементов (чисел, строк, других списков и т.д.). Они очень похожи на обычные списки в жизни, например, на список покупок.

Создание списка

Создать список можно так:

---

fruits = ["яблоко", "банан", "апельсин"] # Строки

numbers = [1, 2, 3, 4, 5] # Числа

mixed = [1, "текст", True, 3.14] # Разные типы

empty = [] # Пустой список

Индексация (доступ к элементам)

- Индексы начинаются с 0 (первый элемент — индекс 0).
- Используют отрицательные индексы (последний элемент — -1).

---

fruits = ["яблоко", "банан", "апельсин"]

print(fruits[0]) # "яблоко" (первый элемент)

print(fruits[-1]) # "апельсин" (последний элемент)

Срезы (slice)

Срезы (slice) — получение части списка

Синтаксис: **список[начало:конец:шаг]**

- начало — включительно (по умолчанию 0).
- конец — не включается (по умолчанию до конца)
- шаг — через сколько элементов брать (по умолчанию 1)

---

numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

print(numbers[2:5]) # [2, 3, 4] (элементы с 2 по 4)

print(numbers[:3]) # [0, 1, 2] (первые 3 элемента)

print(numbers[5:]) # [5, 6, 7, 8, 9] (с 5 до конца)

print(numbers[::2]) # [0, 2, 4, 6, 8] (каждый второй)

print(numbers[::-1]) # [9, 8, 7, 6, 5, 4, 3, 2, 1, 0] (разворот)

Основные методы списков

- append() — добавить элемент в конец

---

fruits = ["яблоко", "банан"]

fruits.append("киви")

print(fruits) # ["яблоко", "банан", "киви"]

- remove() — удалить элемент по значению

---

fruits = ["яблоко", "банан", "киви"]
fruits.remove("банан")

print(fruits) # ["яблоко", "киви"]

- sort() — сортировка списка

---

numbers = [3, 1, 4, 2]
numbers.sort()

print(numbers) # [1, 2, 3, 4] (по возрастанию)

fruits = ["яблоко", "банан", "Апельсин"]

fruits.sort()

print(fruits) # ['Апельсин', 'банан', 'яблоко'] (по алфавиту, учитывая регистр)

Урок 3.2: Кортежи (tuple) и множества (set)

Эти структуры данных очень похожи на списки, но имеют важные отличия. Давайте разберем их!

Кортежи

Кортеж — это неизменяемый (immutable) список.

- Создаётся через круглые скобки () (или без них).
- Элементы нельзя изменить после создания.

---

coordinates = (10, 20)

colors = "красный", "зелёный", "синий" # Скобки можно не ставить

empty_tuple = ()

Кортежи нужны для защиты данных от случайных изменений, также и можно использовать как ключи в **словарях** (урок 3.3). К слову, кортежи работают быстрее списков.

---

point = (3, 5)
print(point[0]) # 3 (индексация как в списках)

 # point[0] = 7 # Ошибка! Кортеж нельзя изменить

x, y = point # Распаковка кортежа

print(x, y) # 3 5

Методы кортежей
Их мало (т.к. кортежи неизменяемы):

- count(x) — сколько раз встречается x.
- index(x) — индекс первого вхождения x.

---

numbers = (1, 2, 2, 3)
print(numbers.count(2)) # 2

print(numbers.index(3)) # 3

Множества (set)

Множество — это неупорядоченная коллекция уникальных элементов.

- Создаётся через фигурные скобки {} (но для пустого set()).
- Нет индексов! Порядок элементов не сохраняется.

---

fruits = {"яблоко", "банан", "апельсин"}

empty_set = set() # Пустое множество (не `{}` — это словарь!)

Индексы нужны для вхождения элемента, а также для удаления дубликатов из списка.

Также над множествами можно совершать операции: объединение, пресечение и т.д.

---

a = {1, 2, 3}

b = {3, 4, 5}

print(a | b) # {1, 2, 3, 4, 5} (объединение)

print(a & b) # {3} (пересечение)

print(a - b) # {1, 2} (разность)

Множества имеют различные методы:

- add(x) — добавить элемент.
- remove(x) — удалить элемент (ошибка, если нет).
- discard(x) — удалить, если есть (без ошибки).
- pop() — удаляет и возвращает случайный элемент.

```
nums = {1, 2, 3}
nums.add(4)      # {1, 2, 3, 4}
nums.discard(2)  # {1, 3, 4}
```

Характеристика	Список	Кортеж	Множество
Изменяемость	✚	▬	✚
Упорядоченность	✚	✚	▬
Уникальность	▬	▬	✚
Скобки	[ ]	( )	{ }

Урок 3.3: Словари

Представь, что словарь — это обычная книга-словарь, где есть:

- Слово (ключ) — что мы ищем
- Значение (значение) — что это слово означает

```
my_dict = {
    "яблоко": "apple",
    "собака": "dog",
    "книга": "book"
}
```

Здесь:

- "яблоко", "собака", "книга" — ключи (как слова в словаре)
- "apple", "dog", "book" — значения (как перевод этих слов)

Словарь можно создать несколькими способами:

1. Вручную

---

grades = {"Анна": 5, "Иван": 4, "Мария": 5}

В данном случае необходимо открыть фигурные скобки (как в множествах), вписать ключ (“Анна”), поставить двоеточие, вписать значение для ключа (5).

Словари можно продолжать бесконечно долго, элементы можно вписывать через запятую.

2. Через dict()

---

phone_book = dict(Анна="123-456", Иван="789-012"

При создание словаря, нужно прописать функцию dict(), внутри скобок через запятую прописать элементы, используя равно. На первом месте в элементе будет стоять ключ, а на втором значение.

Как работать со словарем?

1. Получить значение

---

print(grades["Анна"]) # Выведет: 5

---

2. Добавить или изменить элемент

---

grades["Петр"] = 4 # Добавит нового ученика

grades["Иван"] = 3 # Изменит оценку Ивана

---

3. Проверить, есть ли ключ

```
if "Мария" in grades:
    print("Мария есть в журнале!")
```

---

4. Удалить элемент

---

del grades["Иван"] # Удалит Ивана из журнала

Полезные методы:

Методы в Python — это та же функция, но доступная только определенным типам объектов.

1. .keys() — выведет все ключи

print(grades.keys()) # Выведет: ['Анна', 'Мария', 'Петр']

2..values() — выведет все значения

---

print(grades.values()) # Выведет: [5, 5, 4]

---

3. .items() — пары ключ-значение

```
for name, grade in grades.items():
    print(f"{name}: {grade}")
```

Чем словари лучше списков?

1. Быстрый поиск — находим значения сразу по ключу (не нужно перебирать весь список)

2. Читаемость — grades["Анна"] понятнее, чем grades[0]

3. Гибкость — ключами могут быть не только числа

Словари — это мощный инструмент, который часто используют в реальных проектах!

Задание:

1. Создай словарь с тремя любимыми играми и их жанрами:
2. Дополни свой словарь добавив новую игру, после удали одну игру
3. Напиши цикл, который выводит все игры и их жанры

Check-list

Что я должен знать и уметь (приготовь карандаш):

1. Работа со списками (lists)

- [ ] Создавать списки
- [ ] Добавлять элементы
- [ ] Удалять элементы
- [ ] Сортировать
- [ ] Использовать срезы

2. Работа с кортежами (tuples)

- [ ] Создавать кортежи (неизменяемые!)
- [ ] Обращаться по индексу
- [ ] Преобразовывать в список и обратно

3. Работа с множествами (sets)

- [ ] Создавать множества (уникальные элементы!)
- [ ] Операции с множествами
- [ ] Добавлять/удалять элементы

4. Работа со словарями (dicts)

- [ ] Создавать словари (ключ: значение)
- [ ] Получать/изменять значения
- [ ] Проверять наличие ключа
- [ ] Перебирать словарь

Глава 4: Функции и анонимные функции

Урок 4.1: Функции

Функции в Python — это как мини-программы внутри программы, которые выполняют определённую задачу. Их можно вызывать много раз, чтобы не писать один и тот же код заново.

Как создать функцию?

Используем ключевое слово def, даём имя и указываем аргументы (если нужно):

```
def приветствие(имя):
    print(f"Привет, {имя}!")
приветствие("Саша")  # Выведет: Привет, Саша!
```

Как работает?

1. def — говорит Python, что мы создаём функцию.
2. приветствие — имя функции.
3. (имя) — аргумент (переменная, которую можно передать в функцию).
4. Внутри функции пишем код, который выполняется при её вызове.

Функция с возвратом значения (return)

Функция возвращает результат (например, число, строку, список):

```
def сложить(a, b):
    return a + b
```

```
результат = сложить(3, 5)
print(результат)  # Выведет: 8
```

Урок 4.2: Анонимные функции (lambda)

Иногда функция нужна на один — два раза, и её неудобно объявлять через def. Тогда используют lambda.

Пример обычной функции:

```
def умножить_на_два(x):
    return x * 2
print(умножить_на_два(5))  # 10
```

То же самое, но через lambda:

```
умножить_на_два = lambda x: x * 2
print(умножить_на_два(5))  # 10
```

Или даже без сохранения в переменную:

```
print((lambda x: x * 2)(5))  # 10
```

Где используют lambda?

Чаще всего lambda используют — в функциях типа map(), filter(), sorted(). В таких случаях сама функция передается в качестве аргумента.

люди = [("Анна", 25), ("Петя", 17), ("Мария", 30)]

Сортируем по возрасту (второй элемент в кортеже)

отсортировано = sorted(люди, key=lambda человек: человек[1])

print(отсортировано)

Вывод: [('Петя', 17), ('Анна', 25), ('Мария', 30)]

Разница между def и lambda

Обычная функция (def)	Анонимная функция (lambda)
Имеет имя	Без имени
Может содержать много строк	Только одну строку
Подходит для сложной логики	Для простых операций
Можно вызывать много раз	Часто выводится в функции (def)

Задание:

1. Создай функцию среднее_арифметическое(a, b, c), которая:
   - Принимает три числа.
   - Возвращает их среднее арифметическое
2. Перепиши функцию умножить_на_три(x) в влямбда-функциииии

Check-list

Что я должен знать и уметь (приготовь карандаш):

1. Основы функций (def)

- [ ] Понимаю, что такое функция — это блок кода, который выполняет определённую задачу и может вызываться многократно.
- [ ] Умею создавать функцию с помощью def:
- [ ] Знаю, как передавать аргументы в функцию:
- [ ] Понимаю, что такое return — возврат значения из функции:
- [ ] Могу отличить параметры (в определении) от аргументов (при вызове).

2. Lambda-функции

- [ ] Знаю, что lambda — это анонимная функция, которая записывается в одну строку
- [ ] Умею создавать lambda функцию
- [ ] Знаю, что lambda не может содержать сложную логику

3. Практические навыки

- [ ] Могу написать функцию, которая:
  - Принимает список чисел и возвращает их сумму.
  - Фильтрации чётных чисел:

Заключение

Поздравляем! Вы освоили ключевые основы программирования на Python:

- Переменные — научились хранить и изменять данные, давая им понятные имена.
- Типы данных — разобрались с числами, строками, списками и другими структурами, понимая, как их правильно использовать.
- Функции — узнали, как создавать собственные блоки кода для повторного использования и структурирования программ.
- Циклы — научились автоматизировать повторяющиеся действия с помощью for и while, экономя время и усилия.

Этот фундамент позволит вам двигаться дальше — к более сложным темам, таким как работа с файлами, обработка ошибок, объектно-ориентированное программирование (ООП) и создание реальных проектов.

Главное — продолжайте практиковаться: пишите код, экспериментируйте и не бойтесь ошибок. Они лучшие учителя! Удачи в изучении Python!

ОГЛАВЛЕНИЕ

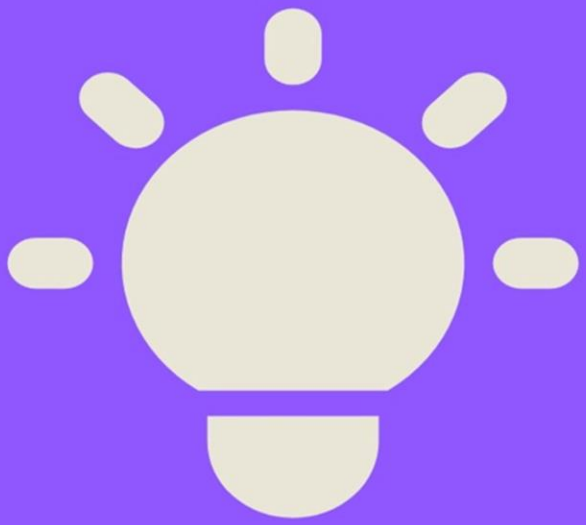
Введение 3
Глава 1. Первые шаги 6
1.1. Установка Python 6
1.2. Установка PyCharm 7
1.3. Переменные и типы данных 9

Глава 2. Управляющие конструкции 18
2.1. Условные операторы 18
2.2. Циклы 23

Глава 3. Работа с данными 27
3.1. Списки 27
3.2. Кортежи и множества 29
3.3. Словари 32

Глава 4. Работа с данными 37
4.1. Функции 37
4.2. Анонимные функции 38

Заключение 41



Python — это язык программирования. Он был создан в 1991 году. Назван в честь комедийного шоу «Монти Пайтон».

Python не требует сложных настроек. Он работает на Windows, macOS и Linux

Python популярен. Его используют в YouTube, NASA и других крупных компаниях.

Этот язык подходит для начинающих. Он проще, чем C++ или Java. При этом на нём можно создавать серьёзные проекты.

Для изучения Python не нужно специальное образование. Достаточно базовых знаний математики и логики.