

# Основы языка Python

Буткемп «Программирование»



# Оглавление

|   |    |
|---|----|
| Введение                                  | 3  |
| История появления языка Python            | 3  |
| Где используется Python                   | 4  |
| Установка Python                          | 5  |
| Среда разработки                          | 6  |
| Первая программа                          | 7  |
| Ввод и вывод данных. Переменные           | 7  |
| Команды print() и input()                 | 7  |
| Переменные в Python и их типы             | 8  |
| Применение типов данных в команде input() | 8  |
| Форматированный вывод                     | 8  |
| Списки и словари. Цикл for                | 9  |
| Списки                                    | 9  |
| Цикл for                                  | 9  |
| Словари                                   | 10 |
| Условные выражения                        | 10 |
| Операторы сравнения                       | 10 |
| Условные операторы                        | 11 |
| Дополнительное условие (elif)             | 11 |
| Логические операторы                      | 11 |
| Сложные условия                           | 12 |
| Практика решения задач                    | 13 |
| Задача «Список покупок»                   | 13 |
| Решение задачи «Список покупок»           | 14 |
| Задача «Стоимость покупок»                | 15 |
| Решение задачи «Стоимость покупок»        | 16 |
| Задача «Калькулятор»                      | 17 |
| Решение задачи «Калькулятор»              | 18 |
| Итоги                                     | 19 |
| Домашнее задание                          | 19 |

# Введение

Здравствуйтесь, уважаемые студенты! Рады приветствовать вас на модуле «Основы веб-разработки на Python»! Здесь вы получите базовые знания о языке Python, которые помогут вам начать свой путь в мир программирования.

Мы начнём с основных понятий. Познакомимся с основными типами данных, синтаксиса языка, и буквально через пару уроков перейдём к полноценным проектам. Научимся использовать библиотеки, узнаем, как работает API, и создадим чат-бота.

## История появления языка Python

Язык Python разработал голландский программист Гвидо ван Россум в 1991 году. Не подумайте, что язык назван в честь змеи питона: Гвидо был большим фанатом британского комедийного сериала «Летающий цирк Монти Пайтона» и именно оттуда пришло название языка.

Создание языка программирования Python у Гвидо ван Россума было мотивировано несколькими факторами и потребностями, которые он видел в существующих языках программирования:

- **Простота и читаемость:** он стремился к лёгкости синтаксиса, чтобы программисты могли выразить идеи более ясно и кратко.
- **Удовольствие от программирования:** Гвидо хотел, чтобы программирование приносило удовольствие, и поэтому создавал язык с минимальной сложностью.
- **Лёгкая интеграция:** Python должен был легко сочетаться с другими языками и системами для разнообразных задач.
- **Универсальность:** Гвидо стремился создать универсальный язык, подходящий для разных областей, от веб-разработки до научных исследований.
- **Открытость и совместное развитие:** сделав Python открытым исходным кодом, он привлёк разработчиков со всего мира, что способствовало его быстрому развитию и распространению.

В итоге Python стал одним из самых популярных языков программирования благодаря простоте, гибкости и активному сообществу разработчиков.

# Где используется Python

Сегодня Python активно применяется в различных сферах:

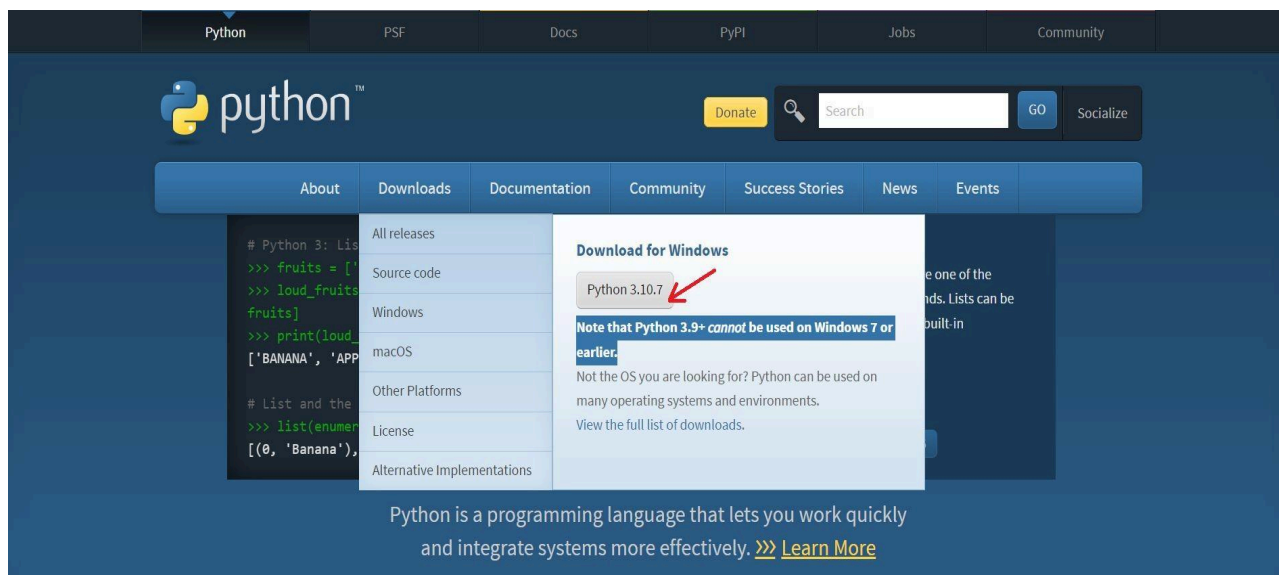
- **Веб-разработка.** Python имеет множество фреймворков для создания веб-приложений. Наверняка вы слышали о таких фреймворках, как Django, Flask и FastAPI. Фреймворки – это инструменты, которые помогают разработчикам создавать веб-приложения быстрее и эффективнее. Они предоставляют готовые шаблоны, структуры и компоненты, которые можно использовать при разработке веб-приложений, сокращая время, затрачиваемое на написание кода с нуля.
- **Научные исследования.** Python используется для обработки и анализа данных из различных источников, таких как эксперименты, опросы, сенсоры и большие объёмы данных. Библиотеки, такие как NumPy, pandas и SciPy, предоставляют инструменты для работы с данными, включая статистический анализ, визуализацию и машинное обучение.
- **Искусственный интеллект.** Python является одним из основных языков для разработки моделей машинного обучения и искусственного интеллекта. Библиотеки, такие как TensorFlow, PyTorch и scikit-learn, предоставляют инструменты для создания и обучения моделей.
- **Автоматизация задач.** Python идеально подходит для автоматизации задач, которые обычно выполняются людьми. Например, одновременное переименование большого количества файлов, преобразование типов файла из одного в другой, отправка сообщений электронной почты и др.
- **Игровая индустрия.** У Python есть своя библиотека pygame, которая используется для создания прототипов игр.
- **Написание софта.** На Python можно создавать настольные приложения с графическим интерфейсом. Наиболее популярными инструментами здесь являются PyQt, Tkinter. Например, такие популярные 3D-программы, как Blender и FreeCAD были созданы с использованием библиотеки PyQt.
- **Кибербезопасность.** Python-скрипты активно используются для обнаружения и анализа уязвимостей в сетях и приложениях.

Этот список далеко не полный, так как Python широко используется во многих сферах благодаря своей универсальности и обширному списку библиотек.

# Установка Python

1. Скачиваем интерпретатор Python с официального сайта <https://www.python.org/downloads/>.
2. На macOS и Linux необходимо обновить версию Python, так как он уже предустановлен:
  - **macOS:**
    - Откройте App Store на вашем компьютере.
    - Введите “Python” в поисковое поле.
    - Нажмите “Получить” (Get) рядом с приложением “Python”.
    - Введите свой пароль и нажмите “Установить” (Install).
    - Дождитесь завершения установки.
  - **Ubuntu:**
    - Откройте терминал (Ctrl+Alt+T).
    - Введите команду `sudo apt update`.
    - Введите пароль вашего пользователя.
    - Снова введите команду `sudo apt install python3`.
    - Нажмите Enter и дождитесь завершения установки.

После установки проверьте версию Python командой `python --version`.
3. Для Windows с версией 8 или выше можно смело устанавливать последнюю версию интерпретатора. Если версия Windows – 7 или ниже, то необходимо установить версию 3.8 или ниже.



После установки откройте командную строку(cmd), введите слово “python”, если всё успешно установилось, у вас выведется сообщение: “Python 3.10.7...”, какая именно версия была установлена.

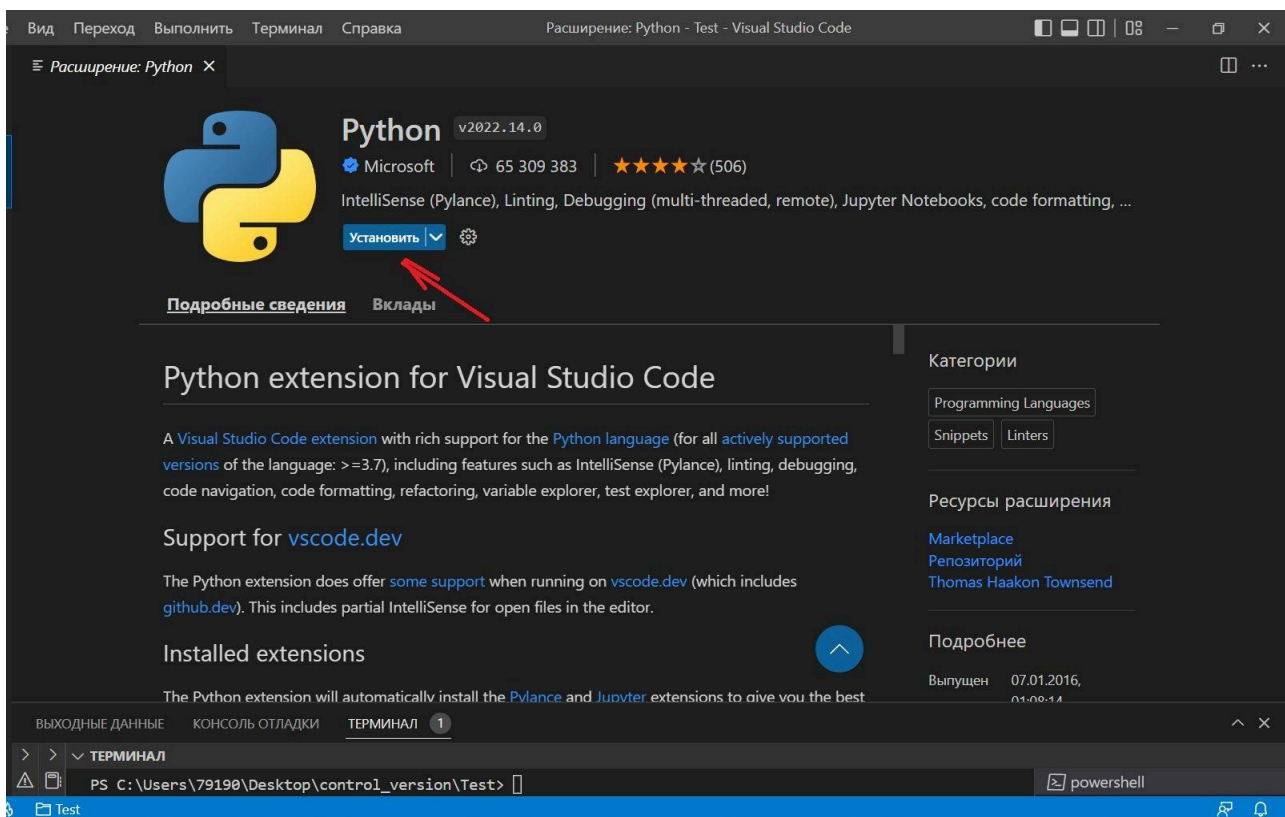
```
Командная строка - python
Microsoft Windows [Version 10.0.19044.2006]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\79190>python
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.
>>>
```

## Среда разработки

Мы будем работать в Visual Studio Code

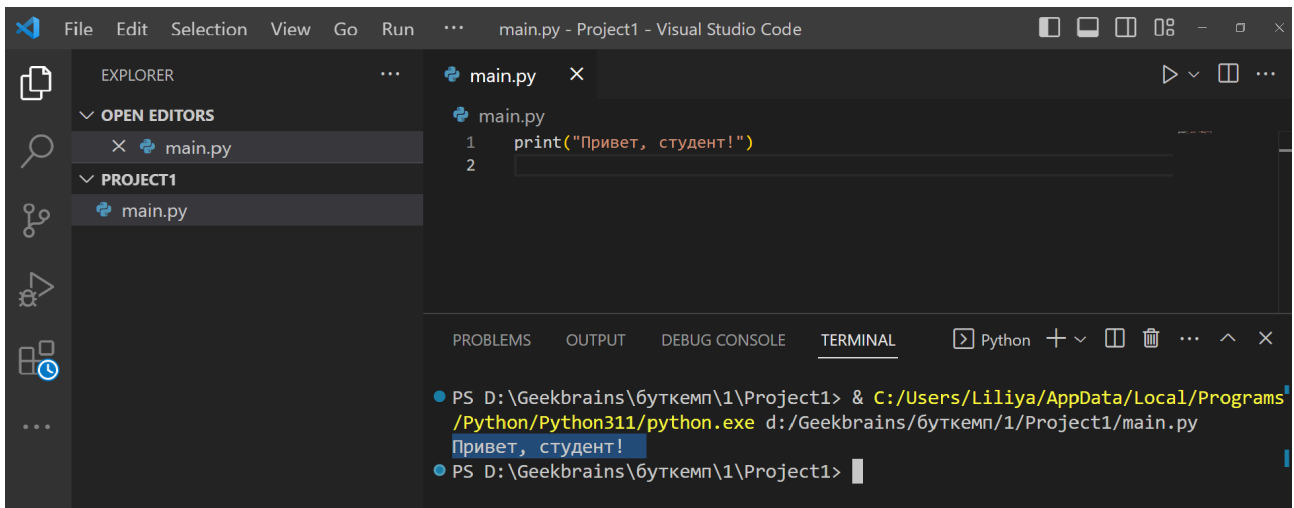
- Для комфортной работы установите расширение, которое будет подсвечивать синтаксис Python.
- Расширения (Extension) → введите в поиске «python» → Установить (install).



# Первая программа

1. В рабочем пространстве (Explorer) создайте файл для будущей программы с расширением `.py` (указываем, что это файл с расширением Python) и введите:  

```
>> print("Привет, студент!").
```
2. Запустите скрипт командой Run Python File (треугольник в правом верхнем углу).
3. Результат работы скрипта появится в консоли.



## Ввод и вывод данных. Переменные

### Команды print() и input()

Команда `print()` используется для вывода информации на экран.

Внутри круглых скобок через запятую мы указываем то, что необходимо вывести на экран. Если это какой-то текст, указываем его внутри кавычек (можно использовать как одинарные, так и двойные кавычки).

```
print('Привет!') # Привет
print(2 + 2) # 4
print('Сегодня', 1, 'сентября!') # Сегодня 1 сентября!
```

Команда `input()` ждёт ввода текста от пользователя.

```
print('Как тебя зовут?')
name = input()
print('Привет,', name)
```

## Переменные в Python и их типы

Благодаря динамической типизации, интерпретатор Python может сам определить тип.

```
age = 25 # int - целочисленный тип данных
name = 'Иван' # str - строка
pi = 3.14 # float - вещественный тип (дробное число)
is_active = True # bool - логический тип
```

## Применение типов данных в команде input()

Если мы ожидаем от пользователя определённый тип данных, можем обернуть команду input() соответствующей функцией.

```
print('Как вас зовут?')
name = input()
print("Сколько вам лет? ")
age = int(input())
print("Какой у вас рост (в метрах)? ")
height = float(input())
```

## Форматированный вывод

Улучшим читаемость вывода, используя f-строки для вставки значений переменных name, age и height в строки вывода:

```
print('Как вас зовут?')
name = input()
print("Сколько вам лет? ")
age = int(input())
print("Какой у вас рост (в метрах)? ")
height = float(input())

print(f"Привет, {name}!")
print(f"Вы ввели свой возраст: {age} лет.")
print(f"Ваш рост составляет: {height} метров.")
```



# Списки и словари. Цикл for

Мы разобрались с использованием переменных для хранения отдельных значений. Что, если нам нужно хранить множество значений?

## Списки

Списки в Python хранят упорядоченные коллекции элементов и могут содержать разные типы данных.

Создаём список:

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

Обратимся к элементам списка по индексу:

```
print(fruits[0]) # яблоко
print(fruits[1]) # банан
print(fruits[2]) # апельсин
```

Справа налево индексы отрицательные:

```
print(fruits[-1]) # апельсин
print(fruits[-2]) # банан
```

Добавим ещё один элемент в список:

```
fruits.append("манго")
print(fruits) # ['яблоко', 'банан', 'апельсин', 'манго']
```

## Цикл for

Цикл for используется для перебора элементов в последовательности (например, в списке) или для выполнения определённого числа раз.

```
# Перебор элементов списка
fruits = ["яблоко", "банан", "апельсин"]
for fruit in fruits:
    print(fruit)

# Цикл выполнится 5 раз, i будет принимать значения от 0 до 4
for i in range(5):
    print(i)
```

## Словари

Словари представляют коллекции пар ключ-значение, где ключи уникальны.

```
person = {"name": "Иван", "age": 30, "city": "Москва"}
print(person["name"])

# Иван

person["height"] = 1.65
print(person)

# {'name': 'Иван', 'age': 30, 'city': 'Москва', 'height': 1.65}
```

## Условные выражения

Условные выражения в Python, представленные конструкциями if, elif и else, позволяют вам создавать логические ветвления в коде. Это основной инструмент для принятия решений в программировании.

## Операторы сравнения

В Python операторы сравнения используются для сравнения двух значений или выражений и возвращают булевское значение (True или False) в зависимости от результата сравнения. Вот основные операторы сравнения в Python:

|    |                  |
|----|------------------|
| == | равно            |
| != | не равно         |
| >  | больше           |
| >= | больше или равно |
| <  | меньше           |
| <= | меньше или равно |

Такие операторы сравнения широко используются в условных выражениях (if-else), циклах и других местах, где требуется проверка условий.

## Условные операторы

Условные операторы (if, else) используются для выполнения разных блоков кода в зависимости от условий.

```
age = 25
if age >= 18:
    print("Вы совершеннолетний")
else:
    print("Вы несовершеннолетний")
```

## Дополнительное условие (elif)

Условный оператор elif обеспечивает более сложные ветвления.

```
x = 10
if x > 0:
    print("Число положительное")
elif x < 0:
    print("Число отрицательное")
else:
    print("Число равно нулю")
```

## Логические операторы

Логические операторы в Python используются для комбинирования и сравнения логических выражений. Вот три основных логических оператора:

|     |           |
|-----|-----------|
| and | и         |
| or  | или       |
| not | отрицание |

Логические операторы часто используются в условиях (if-else) для создания более сложных логических выражений.

## Сложные условия

Сложные условия используются, чтобы проверить одновременно несколько условий или выражений. Например:

```
age = 20
has_permission = True

if age >= 18 and has_permission:
    print("Вы совершеннолетний и имеете разрешение")
else:
    print("Вы не совершеннолетний или не имеете разрешение")
```

Это условие проверяет два различных условия для переменных `age` и `has_permission`, используя логический оператор "и" (`and`).

1. `age >= 18` проверяет, является ли пользователь совершеннолетним – значение переменной `age` должно быть больше или равно 18;
2. `has_permission` проверяет, есть ли у пользователя разрешение:
  - a. если значение переменной `has_permission` равно `True`, то разрешение есть;
  - b. если переменной равно `False` – у человека нет разрешения.

Общее условие `age >= 18 and has_permission` требует, чтобы оба условия выполнялись. Если оба условия истинны (человек совершеннолетний и имеет разрешение), то будет выведено сообщение "Вы совершеннолетний и имеете разрешение". В противном случае будет выполнен блок кода после ключевого слова `else`, и выведется сообщение "Вы не совершеннолетний или не имеете разрешение".

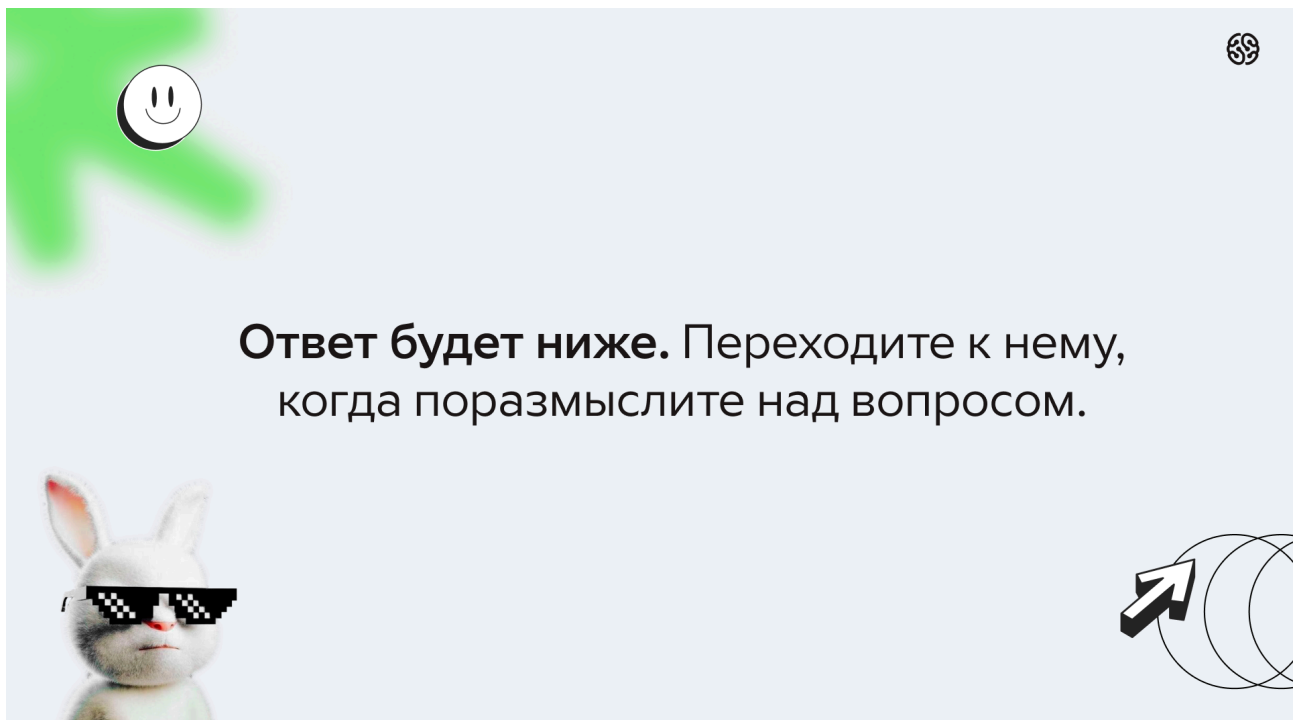
Вы можете создавать сложные условия, комбинируя логические операторы и использование скобок для определения порядка выполнения операций. Это позволяет вам создавать более точные и выразительные проверки.

# Практика решения задач

## Задача «Список покупок»

Создайте программу, которая запрашивает у пользователя список покупок, а затем выводит получившийся список на экран.

| Ввод   | Вывод  |
|--|--|
| <pre>Введите количество покупок: 5 Введите каждую покупку с новой строки: Футболка Джинсы Кроссовки Рюкзак Носки</pre> | <pre>Список покупок: [ 'Футболка', 'Джинсы',          'Кроссовки', 'Рюкзак', 'Носки' ]</pre> |



## Решение задачи «Список покупок»

```
1 purchases = []
2 print("Введите количество покупок:")
3 count = int(input())
4 print("Введите каждую покупку с новой строки:")
5 for i in range(count):
6     purchase = input()
7     purchases.append(purchase)
8 print("Список покупок:", purchases)
```

Разберём построчно данное решение:

**Строка 1.** Создаём список покупок.

**Строки 2-3.** Просим пользователя ввести количество покупок.

**Строки 4-6.** В цикле считываем название покупки, введённое пользователем

**Строка 7.** Добавляем покупку в список **purchases**.

**Строка 8.** Выводим список покупок на экран.

## Задача «Стоимость покупок»

Создайте программу учёта стоимости покупок в магазине.

Дан список покупок:

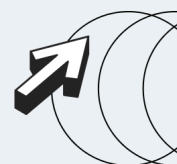
```
purchases = ['Футболка', 'Джинсы', 'Кроссовки', 'Рюкзак', 'Носки']
```

Программа перебирает товары из списка, просит пользователя ввести стоимость товара и сохраняет эти значения в словаре.

| Ввод   | Вывод   |
|--|---|
| Введите стоимость товара Футболка:<br>1000<br>Введите стоимость товара Джинсы:<br>1500<br>Введите                   стоимость                   товара<br>Кроссовки:<br>2000<br>Введите стоимость товара Рюкзак:<br>1800<br>Введите стоимость товара Носки:<br>200 | Стоимость товаров: {'Футболка':<br>1000.0,           'Джинсы':       1500.0,<br>'Кроссовки':     2000.0,     'Рюкзак':<br>1800.0, 'Носки': 200.0} |



Ответ будет ниже. Переходите к нему,  
когда поразмыслите над вопросом.



## Решение задачи «Стоимость покупок»

```
1 purchases = ['Футболка', 'Джинсы', 'Кроссовки', 'Рюкзак', 'Носки']
2 prices = {}
3 for purchase in purchases:
4     print(f"Введите стоимость товара {purchase}:")
5     price = float(input())
6     prices[purchase] = price
7 print("Стоимость товаров:", prices)
```

Разберём построчно данное решение:

**Строка 1.** Объявляем наш список **purchases**.

**Строка 2.** Создаём словарь **prices**, в котором будут храниться цены покупок.

**Строки 3-5.** В цикле перебираем элементы списка **purchases**, просим пользователя ввести стоимость покупки.

**Строка 6.** Добавляем в словарь стоимость покупки.

**Строка 7.** Выводим получившийся словарь на экран.



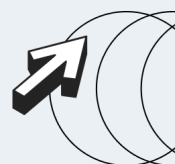
## **Задача «Калькулятор»**

Создайте программу «Калькулятор», которая запрашивает у пользователя два числа и знак операции затем выполняет соответствующую операцию и выводит результат в формате:

"Результат: число1 оператор число2 = результат".



Ответ будет ниже. Переходите к нему,  
когда поразмыслите над вопросом.



## Решение задачи «Калькулятор»

Если пользователь ввёл неверный знак операции, программа сообщает об ошибке.

```
1 print("Введите первое число:")
2 num1 = float(input())
3 print("Введите знак операции (+, -):")
4 operator = input()
5 print("Введите второе число:")
6 num2 = float(input())
7
8 result = None
9
10 if operator == '+':
11     result = num1 + num2
```

```
12 elif operator == '-':
13     result = num1 - num2
14 else:
15     print("Неверный знак операции")
16
17 if result:
18     print(f"Результат: {num1} {operator} {num2} = {result}")
```

Разберём построчно решение задачи:

**Строки 1-6.** Просим пользователя ввести числа и знак операции.

**Строка 8.** Заведём переменную **result** для хранения результата вычисления. Так как пока у нас нет результата, присвоим ей значение **None**.





**Строки 10-13.** Выполним арифметическую операцию в зависимости от знака.

**Строки 14-15.** Обработаем случай, если введён неверный знак операции.

**Строки 17-18.** Напечатаем результат, если он есть.

## Итоги

На этом уроке мы:

-  Познакомились с историей создания языка Python, его «философией».
-  Изучили операторы ввода и вывода данных.
-  Познакомились с использованием переменные, списков, словарей.
-  Изучили цикл for и условные операторы.

## Домашнее задание

1. Доработайте программу «Калькулятор», добавив операции умножения и деления, не забудьте добавить обработку ошибки деления на ноль.
2. Напишите программу, которая позволит пользователю ввести список покупок и их стоимость. Программа должна выводить общую стоимость всех покупок после ввода элементов.