# **Структура программы**

Неважно, на каком языке написана программа и что она делает, структура практически любой программы сводится к трём составляющим:

* ввод данных в программу;
* обработка введённых данных в программе;
* вывод результата обработки данных.

Разберём пример 1.

program\_structure\_1.py

*#!/usr/bin/env python3*name = input() *# Ввод данных*print(name) *# Вывод данных*

Строка 3 содержит ввод данных в программу:

* функция [input()](https://docs.python.org/3/library/functions.html#input), запустившись, ожидает ввода данных от пользователя;
* полученные от пользователя данные внутри тела функции преобразуются в объект типа str (строка);
* функция input() возвращает в основную программу результат своей работы — объект типа str, содержащий данные, введённые пользователем с клавиатуры;
* интерпретатор Python создаёт переменную name и связывает её с только что созданным (инициализированным) объектом, возвращённым функцией input() (создаёт ссылку на объект).

На строке 4 уже знакомая нам функция print() принимает в качестве аргумента переменную name и выводит на экран строковое представление объекта, связанного с ней.

Запустив программу, мы увидим, что ничего не происходит. Программа ожидает ввода данных. Только после того, как мы введём что-нибудь с клавиатуры и нажмём Enter, программа продолжит свою работу.

А зачем передавать данные в программу и что делать с полученными данными?

Это зависит от конкретных задач, которые нужно решить. Задачи могут быть совершенно разные.

Например:

* нужно что-то посчитать;
* представить красивый график;
* вывести отчёт;
* сделать для пользователя удобный интерфейс для работы с данными;
* множество других разнообразных задач.

Способы ввода, обработки и вывода могут быть разные.

Варианты ввода данных. Данные могут быть:

* введены пользователем вручную;
* переданы другой программой;
* считаны из текстового или бинарного файла;
* быть получены из базы данных;
* получены локально или по сети.

Варианты обработки данных:

* программа может разбивать данные на части;
* какие-то части данных обрабатывать в теле программы последовательно или параллельно;
* вызывать другие программы и передавать им какие-то части данных;
* принимать возвращаемые этими программами данные и снова как-то обрабатывать;
* данные могу передаваться по цепочке от программы к программе и видоизменяться на каждом этапе.

Варианты вывода результатов обработки. Программа может:

* выводить данные на экран пользователя или другое устройство вывода (например, на принтер);
* выводить данные в текстовый или двоичный файл;
* записывать данные в базу данных;
* передавать данные локально или по сети другим программам.

Как на языке Python вводить данные в программу и выводить информацию о том, что данные обработаны?

Простейший способ ввода данных в программу: получить данные от пользователя через встроенную функцию input().

Простейший способ вывода обработанных данных: вывести на экран пользователя через встроенную функцию print().

ВНИМАНИЕ

Программа не обязательно должна выводить сами обработанные данные, она должна информировать пользователя или другие программы о том, что обработка данных завершена.

Как мы уже было сказано в секции [**Переменные и объекты**](https://pythonfs.ru/docs/python-language/variables-and-objects), функция — это именованный алгоритм, изолированный от основной программы. Функцию можно рассматривать как подпрограмму, вызываемую из основной программы.

Как и любая программа, функция имеет ввод и вывод.

Аргументы, переданные в функцию — это входные данные.

Возвращаемый результат работы функции — выходные данные.

Функция input() принимает входные данные с клавиатуры, обрабатывает их и возвращает в программу.

Функция print() принимает из основной программы ссылки на объекты, связанный с переменной name, выполняет форматирование (оформление) строковых представлений переданных объектов и выводит результат на экран.

Рассмотрим пример 2.

program\_structure\_2.py

*#!/usr/bin/env python3*name = input('Назовите ваше имя: ') *# Приглашение для ввода данных*print('Привет,', name) *# Выведет: Привет, <имя>*print('Привет,', name, '!') *# Выведет: Привет, <имя> !*print('Привет, ', name, '!', sep='') *# Выведет: Привет, <имя>! Отсутствие пробелов при передаче именованных аргументов*print('Привет, ', name, '!', sep = '') *# Выведет: Привет, <имя>! Плохая практика — пробелы*print('Привет, ', name, '!', sep='', end='') *# Выведет: Привет, <имя>! без \n (без символа перевода строки)*print('Привет, ', name, '!', end='', sep='') *# Выведет: Привет, <имя>! без \n Пример произвольного порядка именованных аргументов*print('Привет,', name, end='!\n') *# Выведет: Привет, <имя>! с символом перевода строки*

Чтобы пользователю было понятно, какую информацию нужно ввести в программу, функция input() может принимать в качестве арумента приглашение для ввода данных и выводить его на экран. Это мы видим это на строке 3.

На строке 4 мы вызываем функцию print() и передаём ей через запятую два параметра:

1. объект 'Привет,' типа str (строка) в одинарных кавычках;
2. строковое представление объекта, на который ссылается переменная name.

ВНИМАНИЕ

Объект 'Привет,' не связан ни с одной переменной. Так тоже можно делать.

При выводе на экран объектов 'Привет,' и name функция print() добавит между ними пробел.

Если мы захотим поставить в конце знак восклицания, как показано на строке 5, и добавим объект ! третьим аргументом, функция print() поставить между именем пользователя и знаком ! пробел, что не соответствует правилам русского языка.

Функция print() спроектирована так, что может принимать дополнительный необязательный аргумент sep — separator — то есть разделитель. Аргумент sep необязательный, потому что для него у функции предусмотрено значение по умолчанию, равное ' ', то есть одному пробелу. Именно аргумент sep отвечает за то, как будут разделены между собой аргументы, которые нужно вывести на экран.

Аргумент sep — именованный (имеющий имя), в отличие от неименованных аргументов, передаваемых в функцию для вывода на экран.

Аргумент sep должен быть передан в функцию **после** остальных аргументов.

Аргумент sep имеет имя, чтобы функция print() могла отличить его от других аргументов и не приняла бы его за ещё один объект, который нужно вывести на экран.

Мы можем настроить поведение функции print(), передав в качестве аргумента sep пустую строку, используя выражение sep=''.

ВНИМАНИЕ

Если в выражениях мы использовали пробел между операндами (вспомним a = 1), то при передаче именованных аргументов в функцию между названием аргумента, оператором = и передаваемым объектом пробел принято **не ставить**. Это считается хорошей практикой, хотя интерпретатор может обрабатывать и форму записи с пробелами.

На строке 6 приведена хорошая практика оформления кода, а на строке 7 — плохая практика оформления кода.

Функция print() имеет ещё один именованный аргумент — end. Он отвечает за то, какой символ будет вставляться в конце строки вывода. По умолчанию это невидимый символ перевода строки, который в исходном тексте записывается как \n. Но мы можем настроить вывод, вставив в конец строки любые символы.

Если мы не хотим, чтобы в конце строки стоял символ перевода строки, передаём в функцию в качестве аргумента end пустую строку, используя выражение end='', как показано на строке 8.

Именованные аргументы можно записывать в произвольном порядке. Интерпретатор всё равно считает их правильно, потому что каждый из них имеет имя. На строке 9 аргументы sep и end переданы не в таком порядке, как на строке 8.

На строке 10 показано, как настроить вывод функции print(), чтобы в конце строки вывода ставился знак восклицания и символ перевода строки: end='!\n'.

Рассмотрим пример 3.

program\_structure\_3.py

*#!/usr/bin/env python3*name = input('Назовите ваше имя: ') *# Ввод*output = f'Привет, {name}!' *# Обработка*print(output) *# Вывод*

Python позволяет форматировать (оформлять) строки. Подробнее это будет рассмотрено в подсекции [**Строки**](https://pythonfs.ru/docs/python-language/built-in-data-types/sequences/strings), но один вид форматирования строк полезно рассмотреть уже сейчас.

На строке 4 мы, используя специальную форму записи, создаём строку и вставляем в неё другую строку — объект, на который ссылается переменная name.

Форма записи такая: f'строка{<переменная>}'.

Строка, созданная при помощи этой формы записи, называется **f-строка**.

Строка записывается в одинарных или двойных кавычках, перед строкой ставится символ f, а внутри строки вставляется одна или несколько переменных в фигурных скобках.

На строке 4 полученная таким способом строка связывается с переменной output.

program\_structure\_4.py

*#!/usr/bin/env python3*name = input('Назовите ваше имя: ')print(f'Привет, {name}!')

В примере 4 программа ведёт себя так же, как в примере 3, только без объявления переменной output: в функцию print() передаётся ссылка на объект, не связанный с ни с одной переменной и инициализированный (созданный) при помощи литерала с использованием f-строки.

program\_structure\_5.py

*#!/usr/bin/env python3*print(f"Привет, {input('Назовите ваше имя: ')}!") *# Вложенный вызов функции input()*

В примере 5 мы делаем то же, что в примере 4 и примере 3, но уже без использования переменной name.

В f-строку внутри фигурных скобок, где должна была быть переменная name, мы вставили вызов функции input() с приглашением ввода данных.

Интерпретатор Python может обрабатывать такую форму записи, и это позволяет нам использовать в программе объекты без связывания их с переменными.

ВНИМАНИЕ

В примере 5 мы заключили f-строку в двойные кавычки, потому что вызов функции input(), вставленный внутрь строки, содержит аргумент — строку, заключённую в одинарные кавычки.

Если бы аргумент функции input() был заключён в двойные кавычки, нам пришлось бы заключать f-строку в одинарные кавычки, иначе в процессе синтаксического анализа текста программы интерпретатор Python обнаружил бы синтаксическую ошибку и не смог выполнить программу.