


[← Урок Знакомство](#)

Знакомство со средой

- 1 Введение
- 2 Знакомство с IDE
- 3 Команда `print()`
- 4 Тестирующая система
- 5 Команда `input()`
- 6 Переменные
- 7 Трассировка

Аннотация

Первый урок посвящен введению в язык Python и знакомству с теми техническими средствами, которые понадобятся нам для обучения.

Сегодня мы научимся писать программы, которые умеют что-то выводить на экран и считывать информацию с клавиатуры. И познакомимся с переменными.

1. Введение

Программа — список команд, то есть инструкций для компьютера:

- отобразить что-нибудь на экране,
- вычислить что-нибудь

и т. д.

В вашем компьютере много разных программ. Программа-браузер показывает вам страницы в Интернете, программа-плеер проигрывает музыку, Word позволяет редактировать тексты, игры развлекают вас, вирусы мешают — и все это программы. Чтобы создать страницу на сайте в Интернете, которую получит и покажет ваш браузер (например, вашу страничку «ВКонтакте» или результаты поиска в Яндексе), компьютер, на котором работает этот сайт, часто тоже выполняет

какую-то программу.

Есть разные способы записать программу. Однако наиболее удобная для компьютера форма записи в виде машинного кода неудобна для человека. Поэтому для составления программ придумали языки программирования.

Язык программирования

Язык программирования — набор строгих правил, согласно которым компьютер может понимать команды и выполнять их. Текст программы, написанной на любом языке программирования, называется **программным кодом** (или просто кодом).

Языки программирования бывают двух основных типов: компилируемые и интерпретируемые. Если программа написана на компилируемом языке (например, Pascal или C++), перед ее выполнением нужно сначала полностью проверить на некоторые ошибки и перевести в более понятную для компьютера форму. Это делает специальная программа — компилятор.

Если программа написана на интерпретируемом языке (например, JavaScript или Python), она не переводится целиком в машинный код, а специальная программа — интерпретатор — идет по коду, анализирует и выполняет каждую отдельную команду. Такой подход придает языку особую гибкость и простоту в написании программ.

Язык Python, который мы сейчас начнем изучать, разработал голландский программист Гвидо Ван Россум (Guido van Rossum) в 1991 году. Не подумайте, что язык назван в честь змеи-питона: Гвидо был большим фанатом британского комедийного сериала «Летающий цирк Монти Пайтона» (англ. Monty Python's Flying Circus), и именно оттуда пришло название языка. В настоящее время в русском языке для обозначения используют два варианта — «Питон» и «Пайтон».

Python относится к интерпретируемым языкам программирования: чтобы запустить написанную на Python программу, нужен интерпретатор Python (его можно скачать с сайта python.org). Подробнее об установке языка Python смотрите в видеоинструкции к уроку.

2. Знакомство с IDE

Команды для интерпретатора можно писать в обычном текстовом редакторе (например, в «Блокноте»). Но чаще для этого пользуются специальной программой, которая называется **средой разработки** (англ. IDE, Integrated Development Environment). Среда разработки — тоже текстовый редактор, но с дополнительными возможностями. Например, она умеет сама находить на компьютере программу-интерпретатор и запускать одной кнопкой. Среда разработки, кроме того, форматирует написанный вами код, чтобы его удобно было читать, а иногда даже подсказывает, где вы допустили ошибку.

На первом этапе мы будем использовать среду разработки Wing IDE. Ее можно скачать с [сайта разработчика](#), фирмы Wingware. В будущем вам может потребоваться более сложная и более богатая возможностями среда разработки. В этом случае мы рекомендуем использовать [PyCharm](#) — продукт российской фирмы JetBrains.

Ну что же, пора приступить к разработке программ. Для начала запустите среду разработки WingIDE.

Давайте проверим настройки кодировок файлов. Эти настройки потребуются для корректной сдачи программ в тестирующую систему.

- Зайдите в меню **Edit** → **Preferences**. Перейдите к категории **Files**. Для опции **Default Encoding** выберите значение Unicode (UTF-8) utf-8.
- Зайдите в меню **Edit** → **Preferences**. Перейдите к категории **Debugger** → **I/O**. Для опции **Debug I/O Encoding** выберите значение Unicode (UTF-8) utf-8.

Примечание: на компьютерах с системой Mac OS вместо меню **Edit** необходимо открывать вкладку **Wing 101** или **Cmd + «,»**.

PEP 8

При оформлении программ мы будем пользоваться PEP 8 — Python Enhanced Proposal. Это документ, описывающий общепринятый (рекомендуемый) стиль написания программ на языке Python. Документ создан по рекомендациям Гвидо Ван Россума и Барри Уорсо, ознакомиться с материалами на русском языке можно, например, [тут](#). В наших материалах мы будем отмечать рекомендации PEP 8 таким блоком.

Среды разработки могут помочь с форматированием по PEP 8. Для включения такой возможности в Wing IDE зайдите в меню **Edit** → **Preferences**. Перейдите к категории **Editor** → **PEP 8**. Для опции **Auto-Reformat for PEP 8** выберите значение **Lines After Edit**.

3. Команда print()

А теперь изучим команду вывода на экран. Для вывода на экран используется команда `print()`.

Команда print()

Внутри круглых скобок через запятую мы указываем то, что необходимо вывести на экран. Если это какой-то текст, указываем его внутри кавычек. Кавычки могут быть как одинарными, так и двойными. Главное, чтобы текст начинался и заканчивался кавычками одного типа. Команда `print` записывается только строчными буквами, другое написание недопустимо, так как в Python строчные и заглавные буквы различны.

Создайте в среде новый файл и напишите в нем команду вывода на экран слова «Привет!» Команда будет выглядеть так:

```
print('Привет!')
```

Обратите внимание: слова выделены разными цветами. В среде Wing IDE используется цветовое выделение синтаксиса. Все стандартные команды и функции выделяются синим и голубым цветом, фиолетовым или зеленым выделяются текстовые данные — строки, заключенные в кавычки.

Могут использоваться другие цветовые настройки — это зависит от версии, IDE. А может задаваться программистом в настройках среды.

PEP 8


Избегайте использования пробелов сразу перед открывающей скобкой, после которой начинается список аргументов функции.

Правильно:

```
print('Привет!')
```

Неправильно:

```
print ('Привет!')
```

Настало время запустить первую программу. Однако кнопка запуска **Run**  пока недоступна. Перед запуском необходимо сохранить файл на диск.

Вы будете писать много программ, только за первый год обучения их будет более 500, поэтому для удобства создавайте для каждого урока свою папку, в которой будете хранить программы этого урока, а сами файлы с программами называйте так, чтобы было понятно, о какой задаче идет речь. Назовите данный файл `hello` и сохраните его. Теперь стала доступна кнопка запуска. Запустите программу.

4. Тестирующая система

Вы только что написали свою первую программу. А сейчас вы сдадите свою первую задачу в тестирующую систему.

Измените свою программу так, чтобы она решала **задачу «Приветствие»**. Запустите и проверьте ее. И, если она работает верно, сдавайте задание в тестирующую систему. Не забудьте сохранить новую программу в отдельном файле. Сейчас это кажется не очень важным, но потом, когда программ станет больше, а сами они длиннее и сложнее, привычка систематизировать материалы очень вам поможет. Чтобы не путаться, код какой программы в каком файле, именем может служить название задачи в тестирующей системе.

Написать или запустить неверную программу — не страшно. Что-то не понять — тоже не страшно. Даже опытные программисты иногда ошибаются и что-то не понимают. Поэтому в случае затруднений всегда смело задавайте вопросы и просите преподавателя вам помочь.

Ваша программа пока умеет только здороваться. Давайте сделаем так, чтобы она выводила две строки с текстом. В первой строке «Привет, Яндекс!», а на второй — «Приятно познакомиться.». Для этого нам понадобится еще одна команда `print`. Эту команду нам обязательно надо написать с новой строки. **Каждая новая команда пишется с новой строки!** Сохраните программу с новым именем `acquaintance`, запустите и проверьте. И, если все правильно, сдавайте задачу «Знакомство» в тестирующую систему.

Когда сдадите задачу, попробуйте внести в свою программу ошибки: добавьте опечатку в слово `print`, уберите открывающую или закрывающую скобку, придумайте еще что-нибудь необычное.

Обратите внимание: в консоли появляется сообщение, в котором есть слово **Error**. **SyntaxError: invalid syntax** часто означает именно забытую кавычку или скобку. Проведите небольшой эксперимент: если вы добавите опечатку в слово `print`, оно станет черным — редактор не узнает его и не понимает, что это команда.

Давайте теперь немного изменим нашу последнюю программу. Попробуйте заменить строчку `print('Привет, Яндекс!')` на строку `print('Привет', 'Яндекс!')`. Запустите и посмотрите, что будет выведено на экран. В этот раз мы внутри команды `print` указали две строки, заключенные в кавычки, но разделили эти строки между собой запятой. Как уже говорилось в начале урока, внутри круглых скобок через запятую указывается то, что нужно вывести на экран. В данном случае на экран выводятся две строки. Первая — «Привет», вторая — «Яндекс!» Эти строки пишутся в команде `print` через запятую, а появляются на экране через пробел. То есть запятая внутри команды заменяется на пробел при выводе на экран.

PEP 8

Обратите внимание: после запятой согласно стандарту PEP 8 обязательно нужно добавлять пробел.

Правильно:

```
print('Привет', 'Яндекс!')
```

Неправильно:

```
print('Привет','Яндекс!')
```

Выполнив программу, мы видим, что фраза «Привет Яндекс!» не соответствует требуемой. В ней не хватает запятой. Запятая должна быть выведена на экран, а это значит, что она должна оказаться внутри строки, заключенной в кавычки. Исправьте программу и добейтесь правильного вывода. Самостоятельно разбейте фразу «Приятно познакомиться.» на две строки, которые внутри команды `print` будут записываться через запятую.

5. Команда `input()`

Пока что все наши программы выводили на экран фразы, известные в момент написания кода. Но программы могут работать и с данными, которые станут известны только во время выполнения: например, их будет вводить пользователь с клавиатуры. Мы можем реализовать это так:

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

Запустите эту программу. Она выводит на экран строчку «Как тебя зовут?» и дальше ждет от пользователя ввода ответа на вопрос, ввода имени.

Введите имя. Запустите еще раз. Введите чужое имя.

Здесь используется команда `input()`.

Команда `input()`

Она всегда пишется с круглыми скобками. Команда работает так: когда программа доходит до места, где есть `input()`, она ждет, пока пользователь введет строку с клавиатуры (ввод завершается нажатием клавиши Enter). Введенная строка подставляется на место `input()`.

То есть, если вы ввели «Аня», программа дальше будет работать так, как будто на месте `input()` было написано «Аня».

Таким образом, `input()` получает от пользователя какие-то данные и в место вызова подставляет строковое значение, в нашем случае записывает его в качестве значения переменной `name`. Мы рассмотрим, что значит сохранить в значение переменной, а пока запомните:

если нужно, чтобы программа что-то печатала на экране и это увидел пользователь, — `print`. Если нужно, чтобы пользователь что-то напечатал с клавиатуры и чтобы программа могла использовать эти данные, — `input()`.

6. Переменные

Команду `name = input()` можно считать так: «Подожди, пока пользователь введет какую-то строку и помести введенную строку в переменную `name`».

Попробуем разобраться, что это значит. Переменные имеют имя и значение.

Имя переменной

Имя переменной должно отражать ее назначение и может состоять из латинских букв, цифр и символа подчеркивания.

Имя не может начинаться с цифры.

PEP 8

Для именования переменных принято использовать стиль `lower_case_with_underscores` (слова из маленьких букв с подчеркиваниями).

Избегайте использовать такие символы, которые могут неоднозначно трактоваться в различных шрифтах: это буква **O** (большая и маленькая) и цифра **0**, буква **I** (большая и маленькая) и цифра **1**. Нельзя использовать в качестве имени переменной и ключевые слова, которые существуют в языке.

В вышеописанном примере переменная содержит в себе имя пользователя, поэтому мы назвали ее `name` (имя). Обратите внимание: слово `name` не подсвечено никаким цветом — в Python это слово ничего не обозначает. Оно что-то значит только в этой программе и только потому, что мы употребили

оператор присваивания. При этом интерпретатору совершенно неважно, что значит слово `name` в английском языке, и мы с тем же успехом могли использовать любое другое имя: например, `user` («пользователь») или просто `n`, или даже `hello`. За имена переменных отвечает программист, то есть вы.

Соблюдайте правило: если в переменной хранится приветствие, пусть она так и называется, если имя — пусть она и называется соответственно.

Значение переменной

Значение переменной — то, что сохраняет в себе переменная.

Знак «`=`» обозначает команду под названием «оператор присваивания». Оператор присваивания присваивает значение, которое находится справа от знака равно, переменной, которая находится слева от знака равно.

В нашем случае это то, что поместил в нее пользователь командой `input()`. Это текстовое значение — строка. То есть переменная сохраняет в себе строковое значение. Говорят, что переменная строкового типа.

PEP 8

Всегда окружайте оператор присваивания одним пробелом с каждой стороны:

Правильно:

```
bird = "Тук-тук"
```

Неправильно:

```
bird="Тук-тук"
```

Еще пример:

```
print('Какая твоя любимая еда?')
meal = input()
print('Да.', meal, '- это вкусно.')
```

Обратите внимание: интерпретатор ждет, что пользователь что-то введет с клавиатуры ровно столько раз, сколько команд `input()` встречается в программе. Каждый `input()` завершается нажатием Enter на клавиатуре.

```
print('Как тебя зовут?')
name = input()
print('Привет,', name)
print('А какая твоя любимая еда?')
meal = input()
```

```
print('Да.', meal, '- это вкусно.')
```

Мы задали значение переменной. И что же, оно никогда не меняется? Конечно, в двух разных программах могут быть переменные с одинаковыми названиями, но разными значениями. Но могут ли в пределах одной программы под одним именем быть разные значения?

Да! Оператор присваивания сообщает переменной то или иное значение независимо от того, была ли эта переменная введена раньше. Вы можете менять значение переменной, записав еще один оператор присваивания. Если у нас имеется переменная, мы можем делать с ее значением все что угодно — например, присвоить другой переменной:

```
hello = 'Здравствуйте.'  
hello2 = hello  
print(hello2)
```

Итак, если вы хотите, чтобы у вас была **переменная с каким-то именем и каким-то значением**, нужно написать на отдельной строчке:

```
<имя переменной> = <значение переменной>
```

Как только эта команда выполнится, в программе появится указанная переменная с таким значением.

Помните: команды выполняются последовательно, в том же порядке, в котором они написаны.

А теперь **смоделируем небольшой взлом программы**.

Загрузите файл [guessing_game.py](#). Это тоже программа на Python. Для начала просто запустите ее (двойным кликом). Как видите, это игра-угадайка. Шанс угадать невелик, у кого получится — тот везучий. Как бы облегчить выигрыш?

Теперь откройте эту программу в редакторе. Это делается не двойным кликом, а кликом правой кнопкой и выбором пункта **Edit with Wing IDE** (или **Редактировать с помощью Wing IDE**). Смысл некоторых строчек этой программы вы уже знаете. О других скоро узнаете или сможете догадаться.

Сейчас мы научимся жульничать в игре путем изменения этой программы. Конечно, можно было бы заменить всю программу единственной строчкой — поздравлением с победой. Но мы будем считать, что менять можно только одну строчку — ту, которая сейчас пуста. Мы можем вписать туда любую команду.

Мы знаем, что к тому моменту, когда выполнение программы доходит до пустой строчки, в переменной под названием `planet` лежит название загаданной планеты. Сделайте так, чтобы в этот момент оно выводилось прямо на экран, — тогда игроку останется лишь повторить название для победы (это потребуется вам в задаче «Взлом планетной угайки»).

Эта задача проверяется преподавателем, поэтому для получения баллов необходимо, чтобы преподаватель проверил сданное решение.

7. Трассировка

Задача (для разбора). Предположим, у нас есть программа, которая входит в интерфейс сайта «Госуслуги» и служит для смены имени. Как будет работать эта программа, что она выведет при каком-либо пользовательском вводе?

```
print('Введите фамилию:')
surname = input()
print('Введите имя:')
name = input()
print(name, surname)
print('Введите новое имя:')
new_name = input()
print(name, surname)
print(new_name, name)
name = new_name
print(new_name, name)
print(name, surname)
```

Давайте немного изменим программу и посмотрим, что теперь получится.

```
print('Введите фамилию:')
surname = input()
print('Введите имя:')
name = input()
print(name, surname)
print('Введите новое имя:')
new_name = input()
old_name = name
name = new_name
print(new_name, old_name)
print(name, surname)
```

Если вы написали программу и не уверены в правильности ее написания, разберите, что она делает. Если не уверены, что все сделали правильно, — запустите и проверьте свои рассуждения.

Комментарии

Для удобства можно использовать комментарии, которые позволяют программисту делать для себя пометки в коде или делать часть кода не выполнимой, не видимой для интерпретатора.

Если вы начнете строку со знака решетки #, интерпретатор Python будет игнорировать всю эту строку. Программа будет выполняться так, как будто строки нет. Такая строка называется **комментарием**.

Комментарии нужны в двух случаях:

1. Когда нужно добавить в программу какую-то пометку для человека, который будет читать эту программу (например, см. третью строку `guessing_game`).
2. Когда нужно убрать какую-то строку кода, но удалять ее не хочется (например, потом ее,

возможно, понадобится вернуть). Это называется «закомментировать» строчку.

PEP 8

«Встрочные» комментарии находятся в той же строке, что и инструкция. Они должны **отделяться по крайней мере двумя пробелами** от инструкции и начинаться с символа **#** и одного пробела.

Комментарии в строке с кодом не нужны и только отвлекают от чтения, если они объясняют очевидное.

Правильно:

```
x = x + 1 # компенсация границы
```

Неправильно:

```
x = x + 1 # увеличение на единицу
```

Если нужно закомментировать сразу несколько строчек подряд, не делайте это вручную. Выделите эти строчки и выберите **Source → Toggle block comment** (там же можно убрать комментирование).

Заметьте, что в конце `guessing_game` на отдельной строчке стоит знакомая команда `input()`. Зачем она нужна?

Ответ: когда запускаешь программу двойным кликом, окно с программой закрывается сразу, как только программа заканчивает работу. Если программа что-то выводит на экран в конце работы, пользователь этого не увидит. А так программа ждет, пока пользователь нажмет клавишу Enter. Конечно, он может что-то ввести, но это неважно — мы все равно никак не используем этот ввод. Попробуйте закомментировать последнюю строчку `guessing_game` и запустить программу из проводника двойным кликом.

Заметьте: если запустить программу, у которой в конце `input()`, из редактора Wing IDE, в конце ее работы придется лишний раз нажать Enter, хотя в этом и нет необходимости.

Попробуйте запустить программу `hello` или любую другую из своих старых программ двойным кликом — сначала в исходном виде, потом с `input()` в конце.

Справка

Исключительное право на учебную программу и все сопутствующие ей учебные материалы, доступные в рамках сервиса, принадлежат АНО ДПО «Образовательные технологии Яндекса». Воспроизведение, копирование, распространение и иное использование программы и материалов допустимо только с предварительного письменного согласия АНО ДПО «Образовательные технологии Яндекса».

[Пользовательское соглашение.](#)

© 2018 – 2024 ООО «Яндекс»