Урок 1

Знакомство с Python

Где используется Python? Сильные стороны языка. Установка и запуск. Операции и инструкции. Переменные и типы данных. Динамическая типизация. Ввод-вывод. Логические операции. Операторы ветвления. Циклы.

Что такое Python

Python (лучше произносить «питон», хотя некоторые говорят «пайтон») — о предмете данного изучения лучше всего говорит создатель этого языка программирования — голландец Гвидо ван Россум: «Python — интерпретируемый, объектно-ориентированный высокоуровневый язык программирования с динамической семантикой. Встроенные высокоуровневые структуры данных в сочетании с динамической типизацией и связыванием делают язык привлекательным для быстрой разработки приложений (Rapid Application Development»).

Основные критерии, предъявляемые к современному языку программирования, таковы:

1. **Качество программного обеспечения.**

Для многих, в том числе и для меня, основные преимущества — это удобочитаемый синтаксис. Немного языков могут похвастаться им. Программный код на Python читается легче: многократное его использование и обслуживание выполняется гораздо проще, чем использование программного кода на других языках сценариев. Python содержит самые современные механизмы многократного использования программного кода, каким является ООП.

2. **Библиотеки поддержки.**

В составе Python поставляется большое число собранных и переносимых функциональных возможностей, известных как стандартная библиотека. Эта библиотека предоставляет массу возможностей, востребованных в прикладных программах, начиная от поиска текста по шаблону и заканчивая сетевыми функциями. Python допускает расширение как за счёт ваших собственных библиотек, так и за счёт библиотек, созданных другими разработчиками.

3. **Переносимость программ.**

Большая часть программ на языке Python выполняется без изменений на всех основных платформах. Перенос программного кода из Linux в Windows заключается в простом копировании файлов программ с одной машины на другую. Также Python предоставляет массу возможностей по созданию переносимых графических интерфейсов.

4. **Скорость разработки.**

По сравнению с компилирующими или строго типизированными языками, такими, как C, C++ или Java, Python во много раз повышает производительность труда разработчика. Объём программного кода на языке Python обычно составляет треть или даже пятую часть эквивалентного программного кода на языке C++ или Java, что означает меньший объём ввода с клавиатуры, меньшее количество времени на отладку и меньший объём трудозатрат на сопровождение. Кроме того, программы на языке Python запускаются сразу же, минуя длительные этапы компиляции и связывания, необходимые в некоторых других языках программирования, что еще больше увеличивает производительность труда программиста.

Некоторые термины

Интерпретируемый язык программирования — язык программирования, в котором исходный код программы не преобразуется в машинный код для непосредственного выполнения центральным процессором, а исполняется с помощью специальной программы-интерпретатора. Т.е. код программы выполняется «на лету», переводится в машинный язык строчка за стройкой во время выполнения программы.

Высокоуровневый язык программирования — язык программирования, разработанный для быстроты и удобства использования программистом. Основная черта высокоуровневых языков — это

абстракция, то есть введение смысловых конструкций, кратко описывающих структуры данных и операции над ними, описания которых на машинном коде (или другом низкоуровневом языке программирования) очень длинны и сложны для понимания.

Динамическая типизация — приём, широко используемый в языках программирования и языках спецификации, при котором переменная связывается с типом в момент присваивания значения, а не в момент объявления переменной. Таким образом, в различных участках программы одна и та же переменная может принимать значения разных типов.

Синтаксис — сторона языка программирования, которая описывает структуру программ как набор символов (обычно говорят — безотносительно к содержанию). Синтаксису языка противопоставляется его семантика. Синтаксис языка описывает «чистый» язык, в то время как семантика приписывает значения (действия) различным синтаксическим конструкциям.

Семантика — система правил определения поведения отдельных языковых конструкций. Семантика определяет смысловое значение предложений алгоритмического языка.

Модульность — принцип, согласно которому программа разделяется на отдельные именованные сущности, называемые модулями. Модульность часто является средством упрощения задачи проектирования программы и распределения процесса разработки между группами разработчиков. При разбиении программы на модули для каждого из них указывается реализуемая им функциональность, а также связи с другими модулями.

Задачи, решаемые с помощью Python

Будучи удачно спроектированным языком программирования, Python прекрасно подходит для решения широкого спектра задач. Он и как инструмент управления другими программными компонентами, и для реализации самостоятельных программ. Фактически круг ролей, которые может играть Python как многоцелевой язык программирования, практически не ограничен: он может использоваться для реализации чего угодно: от веб-сайтов и игровых программ до управления роботами и космическими кораблями.

Однако сферу использования Python в настоящее время можно разбить на несколько широких категорий. Следующие несколько разделов описывают наиболее типичные области применения Python.

1. **Системное программирование.**

Встроенные в Python интерфейсы доступа к службам операционных систем делают его идеальным инструментом для создания переносимых программ и утилит системного администрирования (иногда они называются инструментами командной оболочки).

2. **Графический интерфейс.**

Простота Python и высокая скорость разработки делают его отличным средством разработки графического интерфейса. В состав Python входит стандартный объектно-ориентированный интерфейс к Tk GUI API, который называется tkinter.

3. **Веб-сценарии.**

Интерпретатор Python поставляется вместе со стандартными интернет-модулями, которые позволяют программам выполнять разнообразные сетевые операции как в режиме клиента, так и в режиме сервера. Сценарии могут производить взаимодействия через сокеты, извлекать информацию из форм, отправленных серверным CGI-сценариям; передавать файлы по протоколу FTP; обрабатывать файлы XML; передавать, принимать, создавать и производить разбор писем электронной почты; загружать веб-страницы с указанных адресов URL и многое другое.

4. **Интеграция компонентов.**

Возможность Python расширяться и встраиваться в системы на языке и C++ делает его удобным и гибким языком для описания поведения других систем и компонентов. Например, интеграция с библиотекой на языке C позволяет Python проверять наличие библиотечных компонентов и запускать их, а встраивание Python в программные продукты позволяет производить настройку программных продуктов без необходимости пересобирать эти продукты.

5. **Приложения баз данных.**

В языке Python имеются интерфейсы доступа ко всем основным реляционным базам данных: Sybase, Oracle, Informix, ODBC, MySQL, PostgreSQL, SQLite и многим другим.

6. **Быстрое создание прототипов.**

В программах на языке Python компоненты, написанные на Python и на C, выглядят одинаково. Благодаря этому можно сначала создавать прототипы систем на языке Python, а затем переносить выбранные компоненты на компилируемые языки, что существенно экономит время разработки.

Проекты, в которых используется Python

- Компания Google использует Python в своей поисковой системе и оплачивает труд создателя Python — Гвидо ван Россума.
- Такие компании, как Intel, Cisco, Hewlett-Packard, Seagate, Qualcomm и IBM, используют Python для тестирования аппаратного обеспечения.
- Служба коллективного использования видеоматериалов YouTube в значительной степени реализована на Python.
- NSA использует Python для шифрования и анализа разведданных.
- Компании JPMorgan Chase, UBS, Getco и Citadel применяют Python для прогнозирования финансового рынка.
- Популярная программа BitTorrent для обмена файлами в пиринговых сетях написана на языке Python.
- Популярный веб-фреймворк App Engine от компании Google использует Python в качестве прикладного языка программирования.
- NASA, Los Alamos, JPL и Fermilab используют Python для научных вычислений.

Установка и запуск интерпретатора

Различия python 2.x и python 3.x

Существуют и параллельно развиваются две версии python — 2 и 3. Почему полностью не откажутся от 2-ой версии? Если коротко, то python3 не имеет полной обратной совместимости с предыдущей версией: на python2 написано очень много продуктов, у разработчиков часто нет возможности переписать всё на новую версию. Об отличиях в версиях можно почитать здесь.

Мы будем пользоваться только версией python3 и не будем говорить о python2.

Установка

Как уже отмечалось выше, Питон — интерпретируемый язык. Т.е. для того чтобы программы на Питоне выполнялись, на вашем ПК должна быть установлена программа-интерпретатор. Подробно и с картинками установка описана здесь.

Установка под Windows

Скачиваем установщик с официального сайта. Возьмите наиболее свежую версию (нам подойдет любая версия старше 3.2). Следуем указанию мастера установки. Рекомендую все параметры оставить по умолчанию (особенно путь!).

Установка под Linux

Здесь всё совсем просто: в любой Linux-системе python является изначально предустановленным, поскольку он является стандартным компонентом. Но будьте внимательны, сразу установлены две версии python2 и python3.

Запуск и выполнение

Программы на python — это обычные текстовые файлы, которые вы можете набирать в любом чистом текстовом редакторе. Чистым называется любой текстовый редактор, который не добавляет никаких символов, кроме набранных вами (например, MS Word точно не подойдёт).

Под Windows

При установке интерпретатора автоматически будет установлена простая графическая IDLE (среда разработки).
Для запуска: Пуск → Программы → Python 3.x → IDLE (Python GUI).

Чтобы запустить интерактивную оболочку интерпретатора, в командной строке наберите:

```
python3
```

Замечание. Если у вас интерпретатор не прописан в переменных среды, то вместо python3 необходимо указать полный путь к интерпретатору python, например:

```
c:/python35/python.exe
```

О том, как прописать переменную среды, здесь.

Под Linux

Для запуска интерактивной оболочки интерпретатора выполните в консоли:

```
python3
```

Оболочка Python — это место, где можно исследовать синтаксис Python, получать интерактивную справку по командам и отлаживать небольшие программы.

Сама по себе оболочка Python — замечательная интерактивная площадка для игр с языком. На протяжении всех уроков мы будем ей пользоваться, вы будете встречать примеры наподобие этого:

```
>>> 1 + 1
2
```

Первые три угловых скобки — >>> — обозначают приглашение оболочки Python. Его вводить не надо. Это только для того, чтобы показать, что этот пример должен выполняться в оболочке Python.

1 + 1 — это то, что вы вводите. В оболочке можете ввести любое корректное выражение или команду языка Python. Не стесняйтесь! Худшее, что может случиться, — это сообщение об ошибке. Команды выполняются сразу (как только вы нажмёте ↵ Enter), выражения вычисляются тоже немедленно, оболочка печатает результат.

2 — результат вычисления этого выражения. Как ожидалось, 1 + 1 является корректным выражением на Python. Результат, конечно же, 2.

Теперь попробуем другой пример:

```
>>> print('Hello world!')
Hello world!
```

Как правило, программы состоят более чем из одной строки, для ввода полноценной программы нужно воспользоваться любым текстовым редактором, например, Notepad++. Все скрипты (программы) python должны иметь расширение **.py** .

Для запуска python-скрипта:

```
python3 <путь к скрипту>/<имя_скрипта>.py
```

Пример (для Windows):

```
python3 c:/scripts/hello.py
```

IDE

Набирать программы в текстовом редакторе, а потом смотреть результат в консоли не очень удобно и занимает много времени. Поэтому рекомендуем пользоваться IDE (можете использовать любую привычную вам IDE). Хорошая IDE — PyCharm.

IDE (интегри́рованная среда́ разрабо́тки. англ. Integrated development environment) — комплекс программных средств, используемый программистами для разработки программного обеспечения.

PyCharm можно скачать с официального сайта для различных ОС. Community версия является бесплатной, её функционала на 100% хватит для изучения python.

Итак, интерпретатор установлен, текстовый редактор готов к приёму ваших первых программ. И как говорится, лучший способ познакомиться с языком программирования — это начать на нём писать.

Первая программа

Традиционно первой программой будет **hello_world.py**.

```
print("Hello world")
```

print() — функция, которая выводит данные в консоль.

«Hello world» — строка текста. Любой текст в python должен быть заключен в одинарные или двойные кавычки.

На протяжении всего текущего курса все наши программы будут консольными. Т.е. ввод и вывод данных будет осуществляться только через текстовую консоль.

Да, всё так просто! Мы набираем команды, а интерпретатор их выполняет. Прежде чем перейти к более сложным программам, немного теории.

Введение в программирование

Выражения и инструкции

Суть любой программы — получение, обработка и вывод данных. Данные в python представлены объектами.

С определённой точки зрения программы на языке Python можно разложить на такие составляющие, как модули, инструкции, выражения и объекты.

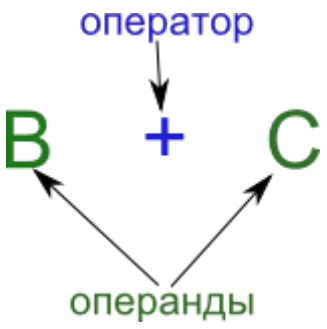
При этом:

1. Программы делятся на модули.
2. Модули содержат инструкции.
3. Инструкции состоят из выражений.
4. Выражения создают и обрабатывают объекты.

Понятия (выражения, операции, инструкции) довольно условны, но для эффективного понимания языка определимся с терминами, которыми мы будем оперировать.

Операция (англ. statement) — наименьшая автономная часть языка программирования; команда.

Пример операции:



Опера́нд (англ. operand) — данные, которые обрабатываются оператором.

Если в оболочке Python мы введём:

```
>>> 2 + 4
```

Получим результат — 6.

- 2+4 — операция;
- 2 и 4 — операнды;

- \+ — оператор;
- 6 — результат операции.

Операции, которые возвращают результат, будем называть выражениями. Операции, которые не возвращают результат, а указывают интерпретатору, что делать, — это инструкции.

Выражение — это операция, которая возвращает значение.

Инструкция — операция, которая не возвращает значение.

Запомните! В любом месте программы, где Питон ожидает получить значение, может быть использовано выражение, возвращающее это значение.

Пример:

```
2 + (6 - 10)
```

В качестве правого операнда для оператора +, используется выражение (6-10).

Это понятия довольно просты и интуитивны в понимании, мы привыкли к ним ещё на уроках математики, термины «операторы» и «операнды» также взяты из математики.

Арифметические операторы в Python

Оператор	Описание
+	Сложение
-	Вычитание
*	Умножение
/	Деление
//	Целочисленное деление
%	Остаток от деления
**	Возведение в степень

Переменные

Чтобы сохранить некоторые значение (данные) и воспользоваться ими далее в программе, используются переменные.

Рассмотрим такой пример:

```
a = 10
b = a + 5
print("10+5 =",b)
```

Разберём каждую строчку нашей программы:

1) a = 10 — создаём переменную a и присваиваем этой переменной значение 10, т.е. теперь в переменной a у нас хранится значение 10.
2) b = a + 5 — создаём новую переменную b, затем этой переменной присваиваем выражение (a+5), т.к. в переменной a у нас хранится значение 10, то вместо a подставляется её значение 10 и получаем b=(10+5) или после сложения b=15.
3) print(“10+5 =”,b) — команда (функция) print() выводит на экран значение своих аргументов или, проще говоря, те данные, которые вы указали в скобках. print() может выводить сразу несколько значений, для этого аргументы указываются через запятую. Наша функция имеет два аргумента: “10+5=” и b. Первый аргумент — это строка текста (строку текста легко можно отличить по верхним кавычкам “”). Вторым аргументом является переменная b, но на экран выводится не имя переменной, а её значение.

Про аргументы будет сказано подробнее в лекции «Функции», пока просто запомните, что это те данные, которые вы указываете в скобках через запятую, после каждой запятой идёт новый аргумент.

Обратите внимание! Если в качестве операнда в выражении используется имя переменной, то вместо имени подставляется её значение. Прежде чем использовать переменную, её нужно объявить.

Переменная — в традиционных языках программирования поименованная область памяти, имя или адрес которой можно использовать для осуществления доступа к данным, находящимся в переменной (по данному адресу).

Присваивание переменной — передача в переменную нового значения.

Значение переменной — информация, хранящаяся в переменной. В переменной может храниться текст, целое число, число с десятичной точкой и т.д.

Знак **=** является операцией присваивания, а также инструкцией, т.е. эта операция не возвращает результата.

Требования к имени переменной

1. В имени переменной используйте только латинские буквы a-zA-Z[1], цифры и символ нижнего подчеркивания _.
2. Имя переменной не может начинаться с цифры.

Помните! Python — регистрочувствительный язык, переменная name и Name — две совершенно разные переменные.

Типы данных (динамическая типизация)

Как было сказано ранее, практически всё, с чем мы имеем дело, программируя на python, — это объекты. Типы объектов могут быть либо встроенными, либо описанными программистом с помощью классов.

Python поддерживает динамическую типизацию, то есть тип переменной определяется автоматически во время исполнения. Поэтому вместо «присваивания значения переменной» лучше говорить о «связывании значения с некоторым именем».

```
>>> a = 8
```

Рассмотрим, как python обработает данное выражение:
В памяти будет создан объект целого типа(int), переменная a получит ссылку на этот объект.

Чтобы лучше понять суть динамической типизации, рассмотрим следующий пример:

```
>>> a = 4
>>> a = a + 1
>>> a = "text"
```

1) В памяти создается объект типа Int (целое), переменная a получает на него ссылку.
2) В правой части оператора **=** стоит выражение, сначала будет вычислен результат выражения. После вычисления результата, будет создан новый объект типа int (со значением 5), переменная a получит ссылку на новый объект в памяти, на старый объект Int (со значением 4) она больше не будет ссылаться.
3) Будет создан новый объект типа str (строка), переменная a снова изменит ссылку.

В отличии от языков со статической типизацией, таких как C++ или Pascal, переменная в python не имеет типа! Правильно говорить: «Переменная указывает на объект такого-то типа». Т.е. именно объект в памяти имеет тип, а переменная — просто указатель на конкретный объект.

Python предоставляет мощную коллекцию объектных типов, встроенных непосредственно в язык, поэтому обычно нет никакой необходимости создавать собственные реализации объектов, предназначенных для решения поставленных задач. Фактически, если у вас нет потребности в специальных видах обработки, которые не обеспечиваются встроенными типами объектов, вам всегда лучше использовать встроенные объекты вместо реализации своих собственных.

[1] На самом деле python3 в имени переменной позволяет использовать кириллицу и даже китайские иероглифы, но это очень плохая практика.

Встроенные типы данных (часть).

Название типа	Описание
int	Целые числа. Пример: 2, 4, 8, -10, -2
float	Числа с десятичной точкой. Пример: 2.6, -5.2
str	Строки(заключены в кавычки). “Hello”, ‘Вася’
bool	Логический тип. Принимает два значения: True, False
list	Список. Пример: [2, 2.4, “Hello”]
tuple	Кортеж. Пример: (2, 2.4, “Hello”)
dict	Словарь. {“name”: “Вася”, “age”: 10}

Явное и неявное преобразование типов

Начнем с примера:

```
>>> name = 'Вася'
>>> age = 10
>>> res = name + age
```

При попытке выполнить третью инструкцию, вы получите сообщение об ошибке:

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'...
```

А что вы ожидали получить, сложив «Васю» с цифрой 10?!

Ещё одной сильной стороной Python являются информативные ошибки (подробнее о них позже).

Выполнив такой код:

```
>>> a = 5
>>> b = -2.4
>>> c = a + b
>>> print(c)
```

Вы получите ожидаемый результат. На самом деле интерпретатор не может выполнить операции с разными типами. Чтобы избавить программиста от ненужной рутины, python в некоторых случаях проводит преобразование типов автоматически (неявно).

Почему же в первом примере, цифра 10 автоматически не была преобразована к строке “10” (так, например, поступает javascript)? Потому что подобные неявные преобразования порождают целый пласт сложно отлаживаемых ошибок. Философия Python говорит: «Явное лучше неявного».

Поэтому, если вы всё же хотите сложить строку “10” и “Вася”, то нужно явно преобразовать цифру к строке. Забегая вперёд: при сложении строк они склеиваются: “Вася” + “10” → “Вася10”

Для явного преобразования используются одноименные функции.

Для преобразования к типу int → используйте функцию Int(). Для преобразование к строке — str() и т.д.

Пример:

```
>>> int(2.5) + 6
```

Получим: 8

Преобразование дробного числа к int отбрасывает дробную часть.

Ввод/Вывод

Программа обрабатывает данные и выводит результаты. Где же программа должна брать данные для обработки? Где угодно! Например:

- Прочитать из файла.
- Получить с любого устройства ввода (сканер, клавиатура и т.п.).
- Получить по сети.

В рамках представленного курса данные в программу будем передавать одним из трех способов:

1. Ввод с клавиатуры.
2. Явно задать в программе в качестве значений переменных (захардкодить).
3. Прочитать из файла.

Чтобы запросить данные у пользователя с клавиатуры, воспользуемся функцией input().

Функция input() может получать необязательный аргумент — строку, которая будет выведена в качестве приглашение/уточнения, а в качестве результата вернёт введённые пользователем данные.

```
>>> name = input("Введите ваше имя: ")
```

Введите ваше имя: <здесь программа остановится и будет ждать ввода с клавиатуры>.

Переменной name будет присвоена строка введённых символов.

Обратите внимание! input() всегда возвращает строку. Если вы хотите работать с цифрами, используйте функции преобразования типов int(), float().

```
>>> a = int(input("Введите целое число"))
```

Для вывода в консоль пользуемся функцией print().

Функция print() принимает неограниченное количество аргументов, которые будут выведены на экран.

```
>>> name = "Вася"
>>> print("Меня зовут", name)
       Меня зовут Вася
```

Логические операции

Часто в реальной жизни мы соглашаемся или отрицаем то или иное утверждение, событие, факт. Например, «Сумма чисел 3 и 5 больше 7» является правдивым утверждением, а «Сумма чисел 3 и 5 меньше 7» — ложным. Можно заметить, что, с точки зрения логики, подобные фразы предполагают только два результата: «правда» и «ложь». Если результатом вычисления выражения может быть лишь истина или ложь, то такое выражение называется логическим. Такие данные имеют тип bool.

С логическими операциями мы отлично знакомы с уроков математики.

Логические операторы

Оператор	Описание
>	Больше
<	Меньше
==	Равно
!=	Не равно
>=	Больше или равно
<=	Меньше или равно

```
>>> x = 5 > 10
```

Переменная будет ссылаться на объект типа bool со значением False (Ложь)

```
>>> x = (2 + 4) > 4
```

Результат: True (Истина).

Любое значение может быть преобразовано к логическому типу функцией bool().

Запомните! Функция bool() возвращает False от НУЛЯ и любой пустой последовательности, во всех остальных случаях будет True.

bool(0) → False

bool(“”) → False

bool(False) → False

bool(-1) → True

bool(“Hello”) → True

bool(“0”) → True

и т.д.

Логические выражения используются в различных инструкциях языков программирования, в частности в инструкциях ветвления

Инструкции ветвления

В теории программирования доказано, что программу для решения задачи любой сложности можно составить из трёх структур, называемых следованием, ветвлением и циклом.

Следованием называется конструкция, представляющая собой последовательное выполнение двух или более операторов (простых или составных).

Ветвление задаёт выполнение либо одного, либо другого оператора в зависимости от выполнения какого-либо условия.

Цикл задаёт многократное выполнение оператора.

Со следованием всё просто: все команды (инструкции) выполняются последовательно, пока программа не завершится.

Познакомимся с циклами.

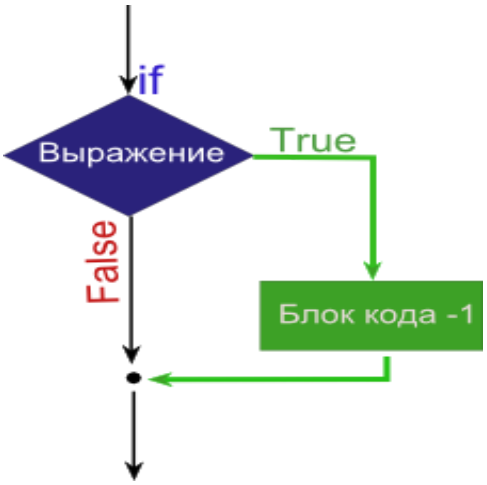


Рис. Схема ветвления if.

Описание схемы

Оператор if называют инструкцией. Помните, что такое инструкция? В качестве выражения может выступать любое выражение, которое будет автоматически преобразовано к логическому.

Цель if — выполнить определённый блок кода (подробнее см. ниже «Операторные скобки») при определённом условии.

Если выражение истинно (True), то выполняется «блок кода-1», если выражение ложно (False), то «блок кода-1» пропускается, программа выполняется дальше.

Пример 1.

```
original_password = 'x777'                          # правильный пароль, хранится в программе
password = input('Введите пароль:') # просим пользователя ввести пароль
access = 0                                                  # переменная, хранит разрешение на доступ
if password == original_password:      # если введен правильный пароль
    print('Пароль принят, добро пожаловать в систему')
    access = 1
if password != original_password:       # если введен неправильный пароль
    print('Пароль неверен, вход запрещен')
```

Рассмотрим пример.

Цель программы — запросить у пользователя пароль, в случае верно введённого пароля дать доступ. Разрешение доступа контролируется переменной access (переводится — «доступ»).

В 4 строке сравниваем введённый пользователем пароль с паролем, хранящимся в программе, если они равны, то сообщаем пользователю, что его пароль принят и меняем значение переменной access на 1 (1 — доступ разрешен, 0 -запрещен).

В 7 строке проверяем, если пароль введён неверно, сообщаем об этом пользователю. Т.к. пароль неверный, значение переменной access оставляем в значении 0.

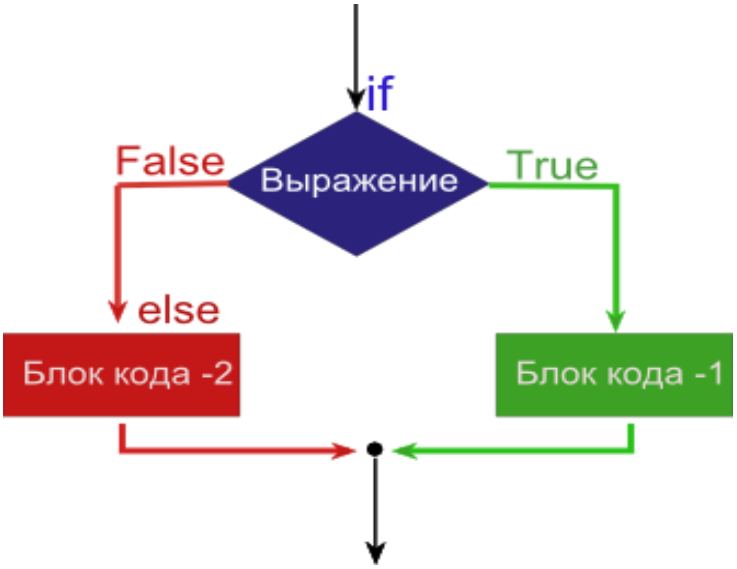


Рис. Схема ветвления if else

Описание схемы.

Если выражение истинно (True), то выполняется «Блок кода-1», если выражение ложно (False), то выполняется «Блок кода-2». Т.е. выполняется либо первый блок, либо второй.

Используя эти знания, предыдущий пример можно переписать.

Пример 2.

```
original_password = 'x777'            # правильный пароль, хранится в программе
password = input('Введите пароль: ') # просим пользователя ввести пароль
access = 0                            # переменная, хранит разрешение на доступ
# Если введен правильный пароль
if password == original_password:
   print('Пароль принят, добро пожаловать в систему')
   access = 1
# Иначе, т.е. Если неправильный пароль
else:
   print('Пароль неверен, вход запрещен')
```

Вложенные инструкции

Внутри блока условной инструкции могут находиться любые другие инструкции, в том числе и условная инструкция. Такие инструкции называются вложенными. Синтаксис вложенной условной инструкции:

```
if условие1:
...
      if условие2:
      ...
      else:
      ...
      …
else:
...
```

Вместо многоточий можно писать произвольные инструкции. Обратите внимание на размеры отступов перед инструкциями. Блок вложенной условной инструкции отделяется двойным отступом. Уровень вложенности условных инструкций может быть произвольным, то есть внутри одной условной инструкции может быть вторая, а внутри неё — еще одна и т.д.

Условие 2 проверяется, только если верно Условие 1.

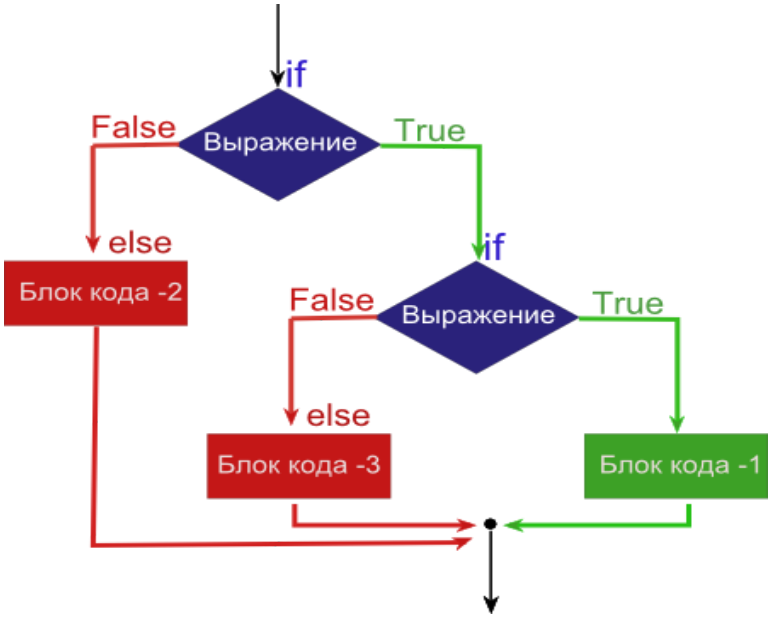


Рис. Схема вложенных инструкций ветвления

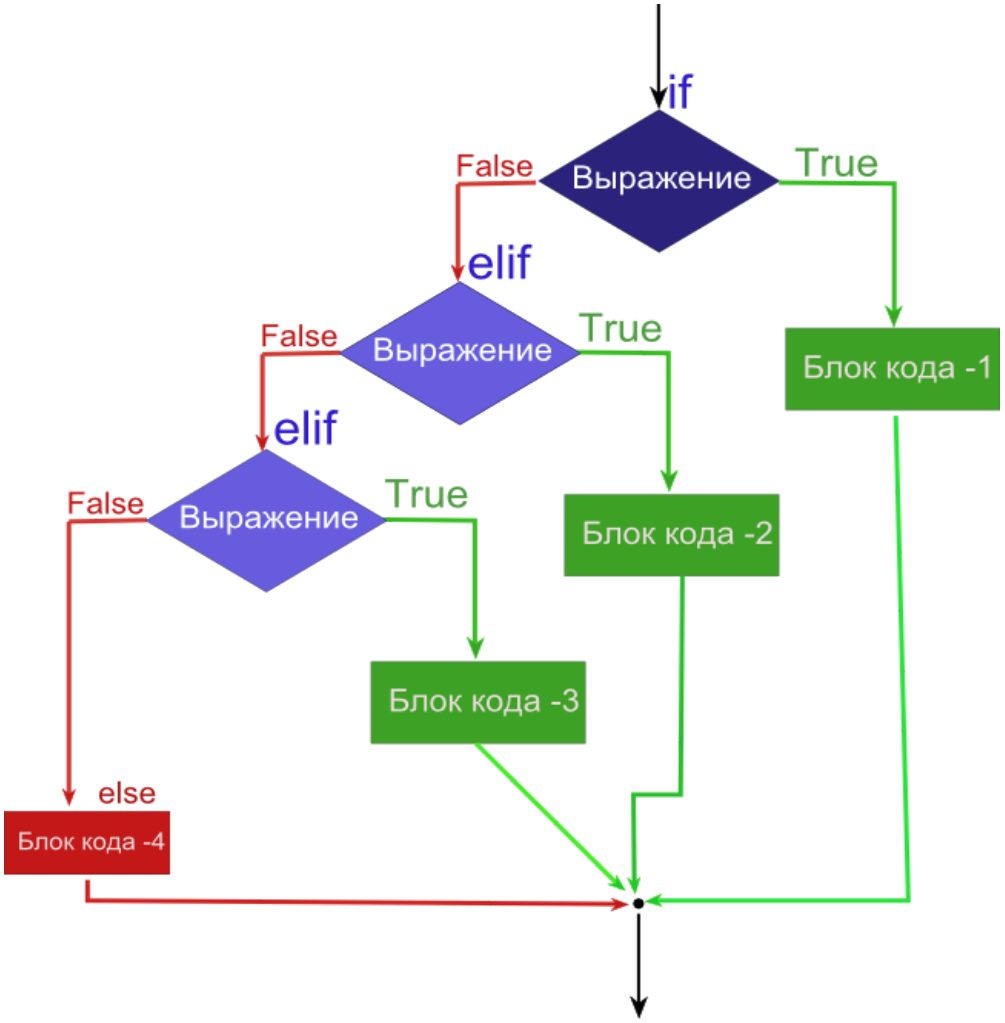


Рис. Полная схема инструкции ветвления

Описание схемы.

Оператор elif переводится как «иначе если». Логическое выражение, стоящее после оператора elif, проверяется, только если все вышестоящие условия ложные. Т.е. в данной схеме может выполниться только один блок кода (1, или 2-ой, или 3-ий, или 4-ый). Если одно из выражений истинно, то нижестоящие условия проверяться не будут.

Если нужно чтобы проверялись все условия, независимо от результата предыдущего, то следует использовать несколько независимых операторов if.

Пример 3.

```
color = 'red'
if color == 'blue':
    print('синий')
# elif сокращение от else if(иначе если)
elif color == 'red':
    print('красный')
elif color == 'green':
    print('зеленый')
# else выполняется, только если все предыдущие проверки вернули False
else:
    print('неизвестный цвет')
```

Операторные скобки

В любом языке программирования существует необходимость выделять блоки кода, для этого используются специальные синтаксические конструкции, показывающие начало и конец блока. В паскале это ключевые слова begin … end; в C++ фигурные скобки {...}. В python операторными скобками являются одинаковые отступы слева перед всеми инструкциями блока.

Подобный синтаксис языка хорош тем, что заставляет программиста правильно табулировать свой код, улучшая читабельность.

Если вспомнить, что символом конца строки в pascal является точка с запятой, то код любой код на паскале можно писать в одну строку, что сильно ухудшает читабельность.

Знакомство с циклами

Все программы, которые мы писали до сих пор, запускались, выполняли необходимые действия, выводили результат и завершали свою работу. Чтобы выполнить любую из наших программ с другим набором данных, необходимо запустить программу заново. Но как много реальных программ вы знаете, которые, выполнив некоторые действия, немедленно завершают свою работу?

Практически все программы работают непрерывно: выполнив одни действия, ожидают новых инструкций, и так до тех пор, пока пользователь не завершит работу программы. Работу большинства программ можно представить в таком виде: получение данных/инструкций --> обработка данных --> вывод результата, получение данных/инструкций --> обработка данных --> вывод результата … И так до тех пор, пока пользователь не завершит работу с программой. Это и есть работа программы в цикле.

Циклы — это инструкции, выполняющие одну и ту же последовательность действий, пока действует заданное условие.

В питоне существуют два типа циклов: while и for in. В данной лекции мы познакомимся только с первым циклом.

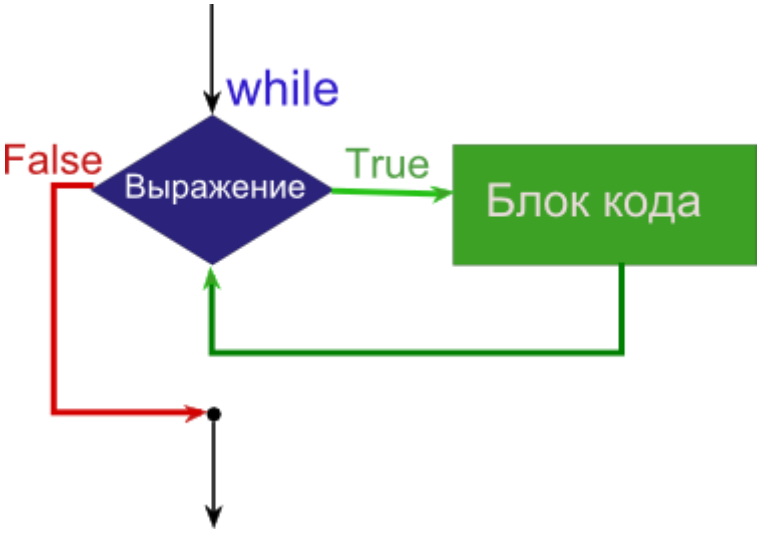


Рис. Схема цикла while

Описание схемы

Если выражение истинно (True), то выполняется «блок кода», программа снова возвращается к проверке логического выражения. Если выражение ложно (False), то программа продолжает свою

работу, не выполняя «блок кода». Т.е. «блок кода» выполняется до тех пор, пока логическое выражение, стоящее после оператора while, истинно.

Блок кода внутри цикла — называется **телом цикла.**

Один шаг цикла (однократное выполнение тела цикла) — называется **итерацией.**

Пример цикла:

```
number = int(input('Введите целое число от 0 до 9: '))
while number<10:
   print(number)
   number = number + 1
print('программа завершена успешно')
```

Рассмотрим пример.

Программа выводит на экран все числа — от введённого числа до 9-ти включительно с шагом 1. Например, если мы введём число 7, то программа выведет 7, 8 и 9.

Вторая строка — это оператор цикла while и number<10 — логическое выражение.

Третья и четвёртая строка — это тело цикла, которое будет выполняться до тех пор, пока логическое выражение number<10 будет истинно. Пятая строчка не относится к телу цикла, т.к. перед ней нет отступа.

Сколько раз выполнится тело цикла, заранее неизвестно — это зависит от заданного значения переменной number.

Обратите внимание на строчку четыре: при каждом выполнении этой строки в цикле её значение будет увеличиваться на единицу до тех пор, пока значение переменной number не станет больше либо равно 10 (при этом значении логическое выражение number<10 станет ложным), цикл завершится.

Зацикливание

Рассмотрим такой пример:

```
a = 5
while a>0:
   print("!")
   a = a+1
```

Запустив данный пример, увидите кучу восклицательных знаков … до бесконечности. Данный цикл при текущих условиях не завершится никогда, потому что a всегда будет больше нуля, условие a>0 всегда будет верным. В программах нужно избегать бесконечных циклов, операционная система считает зациклившуюся программу повисшей (нерабочей) и предлагает «снять с неё задачу».

Инструкции break, continue

В теле цикла можно использовать вспомогательные инструкции break и continue. Иногда использование этих инструкций позволяет упростить ваш код и сделать его более читабельным.

Оператор continue начинает следующий проход цикла, минуя оставшееся тело цикла.

Оператор break досрочно прерывает цикл.

```
i = 0
while True:
    i += 1
    if i >= 10:
# инструкция break при выполнении немедленно заканчивает выполнения цикла
        break
    if i % 2 == 0:
# переходим к проверке условия цикла, пропуская все операторы за инструкцией
        continue
    print(i)
    i += 1
```

Если вы начинающий программист, то пока не «забивайте» голову данными конструкциями, просто знайте о их существовании.

Домашнее задание

1. Смотреть https://github.com/GeekBrainsTutorial/Python_lessons_basic/tree/master/lesson01

Большинство заданий делятся на три категории easy, normal и hard:

- easy — простенькие задачи на понимание основ;
- normal — если вы делаете эти задачи, то вы хорошо усвоили урок;
- hard — наиболее хитрые задачи, часто с подвохами, для продвинутых слушателей.

Если вы не можете сделать normal задачи — это повод пересмотреть урок, перечитать методичку и обратиться к преподавателю за помощью.

Если не можете сделать hard — не переживайте, со временем научитесь.

Решение большинства задач будет разбираться в начале каждого вебинара.

Дополнительные материалы

Всё то, о чём сказано здесь, но подробнее:

1. Настройка Python path.
2. Список всех операторов.

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. Учим python качественно(habr).
2. Самоучитель по python.
3. Лутц М. Изучаем Python. — М.: Символ-Плюс, 2011 (4-е издание).