МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
БОРИСОГЛЕБСКИЙ ФИЛИАЛ
(БФ ФГБОУ ВО «ВГУ»)

МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО УЧЕБНОЙ ДИСЦИПЛИНЕ
Основы языка программирования Python

Методические указания для обучающихся по освоению дисциплины

Приступая к изучению учебной дисциплины, прежде всего обучающиеся должны ознакомиться с учебной программой дисциплины. Электронный вариант рабочей программы размещён на сайте БФ ВГУ.

Знание основных положений, отраженных в рабочей программе дисциплины, поможет обучающимся ориентироваться в изучаемом курсе, осознавать место и роль изучаемой дисциплины в подготовке будущего педагога, строить свою работу в соответствии с требованиями, заложенными в программе.

Основными формами аудиторных занятий по дисциплине являются лекции, и практические занятия, посещение которых обязательно для всех студентов (кроме студентов, обучающихся по индивидуальному плану).

В ходе лекционных занятий следует не только слушать излагаемый материал и кратко его конспектировать, но очень важно участвовать в анализе примеров, предлагаемых преподавателем, в рассмотрении и решении проблемных вопросов, выносимых на обсуждение. Необходимо критически осмысливать предлагаемый материал, задавать вопросы как уточняющего характера, помогающие уяснить отдельные излагаемые положения, так и вопросы продуктивного типа, направленные на расширение и углубление сведений по изучаемой теме, на выявление недостаточно освещенных вопросов, слабых мест в аргументации и т.п.

Не следует дословно записывать лекцию, лучше попытаться понять логику изложения и выделить наиболее важные положения лекции в виде опорного конспекта. Рекомендуется использовать различные формы выделения наиболее сложного, нового, непонятного материала, который требует дополнительной проработки: можно пометить его знаком вопроса (или записать на полях сам вопрос), цветом, размером букв и т.п. – это поможет быстро найти материал, вызвавший трудности, и в конце лекции (или сразу же, попутно) задать вопрос преподавателю (не следует оставлять непонятый материал без дополнительной проработки, без него иногда бывает невозможно понять последующие темы). Материал уже знакомый или понятный нуждается в меньшей детализации – это поможет сэкономить усилия во время конспектирования.

На практически занятиях рекомендуется активно участвовать в анализе решаемых задач, обсуждении алгоритма их решения, выборе способов реализации алгоритма на языке программирования. При возникновении затруднений в решении задач важно сразу выяснить все непонятные моменты, задав вопрос преподавателю.

При подготовке к промежуточной аттестации необходимо повторить пройденный материал в соответствии с учебной программой, примерным перечнем вопросов, выносящихся на зачет. Рекомендуется использовать конспекты лекций и источники, перечисленные в списке литературы в рабочей программе дисциплины, а также ресурсы электронно-библиотечных систем. Необходимо обратить особое внимание на темы учебных занятий, пропущенных по разным причинам. При необходимости можно обратиться за консультацией и методической помощью к преподавателю.

План лекционных занятий

Тема №1. Основы программирования на языке Python.

- Язык программирования Python. Структура программы.
- Типы данных: простые и структурированные.
- Условный оператор. Оператор выбора. Циклы.
- Структурированные типы данных.

Тема №2. Разработка прикладных программ на Python.

- Библиотеки Python. Стандартная библиотека.
- Сетевые возможности языка Python.
- Использование языка Python для математических расчётов.

Методические материалы по теме «Основы языка программирования Python. Знакомство с Python»

Краткая историческая справка

Язык программирования Python был создан к 1991 году голландцем Гвидо ван Россумом.

Свое имя – Пайтон (или Питон) – получил от названия телесериала, а не пресмыкающегося.

После того, как Россум разработал язык, он выложил его в Интернет, где сообщество программистов присоединилось к его улучшению.

Python активно развивается в настоящее время. Часто выходят новые версии. Существуют две поддерживаемые ветки: Python 2.x и Python 3.x. Здесь английской буквой "x" обозначается конкретный релиз. Между вторым и третьим Питоном есть небольшая разница. В данном курсе за основу берется Python 3.x.

Официальный сайт поддержки языка – http://python.org.

Основные особенности языка

Python – интерпретируемый язык программирования. Это значит, что исходный код частями преобразуется в машинный в процессе его чтения специальной программой – интерпретатором.

Python характеризуется ясным синтаксисом. Читать код на нем легче, чем на других языках программирования, т. к. в Питоне мало используются такие вспомогательные синтаксические элементы как скобки, точки с запятыми. С другой стороны, правила языка заставляют программистов делать отступы для обозначения вложенных конструкций. Понятно, что хорошо оформленный текст с малым количеством отвлекающих элементов читать и понимать легче.

Python – это полноценный во многом универсальный язык программирования, используемый в различных сферах. Основная, но не единственная, поддерживаемая им парадигма, – объектно-ориентированное программирование. Однако в данном курсе мы только упомянем об объектах, а будем изучать структурное программирование, так как оно является базой. Без знания основных типов данных, ветвлений, циклов, функций нет смысла изучать более сложные парадигмы, т. к. в них все это используется.

Интерпретаторы Python распространяется свободно на основании лицензии подобной GNU General Public License.

Дзен Питона

Если интерпретатору Питона дать команду **import** this ("импортируй это" здесь видимо следует понимать как "импортируй самого себя"), то выведется так называемый "Дзен Питона", иллюстрирующий идеологию и особенности данного языка. Понимание смысла этих постулатов в приложении к программированию придет тогда, когда вы освоите язык в полной мере и приобретете опыт практического программирования.

- Beautiful is better than ugly. Красивое лучше уродливого.
- Explicit is better than implicit. Явное лучше неявного.
- Simple is better than complex. Простое лучше сложного.
- Complex is better than complicated. Сложное лучше усложнённого.
- Flat is better than nested. Плоское лучше вложенного.
- Sparse is better than dense. Разрежённое лучше плотного.
- Readability counts. Удобочитаемость важна.
- Special cases aren't special enough to break the rules. Частные случаи не настолько существенны, чтобы нарушать правила.
- Although practicality beats purity. Однако практичность важнее чистоты.
- Errors should never pass silently. Ошибки никогда не должны замалчиваться.
- Unless explicitly silenced. За исключением замалчивания, которое задано явно.
- In the face of ambiguity, refuse the temptation to guess. Перед лицом неоднозначности сопротивляйтесь искушению угадать.

- There should be one — and preferably only one — obvious way to do it. Должен существовать один — и, желательно, только один — очевидный способ сделать это.
- Although that way may not be obvious at first unless you're Dutch. Хотя он может быть с первого взгляда не очевиден, если ты не голландец.
- Now is better than never. Сейчас лучше, чем никогда.
- Although never is often better than *right* now. Однако, никогда чаще лучше, чем прямо сейчас.
- If the implementation is hard to explain, it's a bad idea. Если реализацию сложно объяснить — это плохая идея.
- If the implementation is easy to explain, it may be a good idea. Если реализацию легко объяснить — это может быть хорошая идея.
- Namespaces are one honking great idea — let's do more of those! Пространства имён — прекрасная идея, давайте делать их больше!

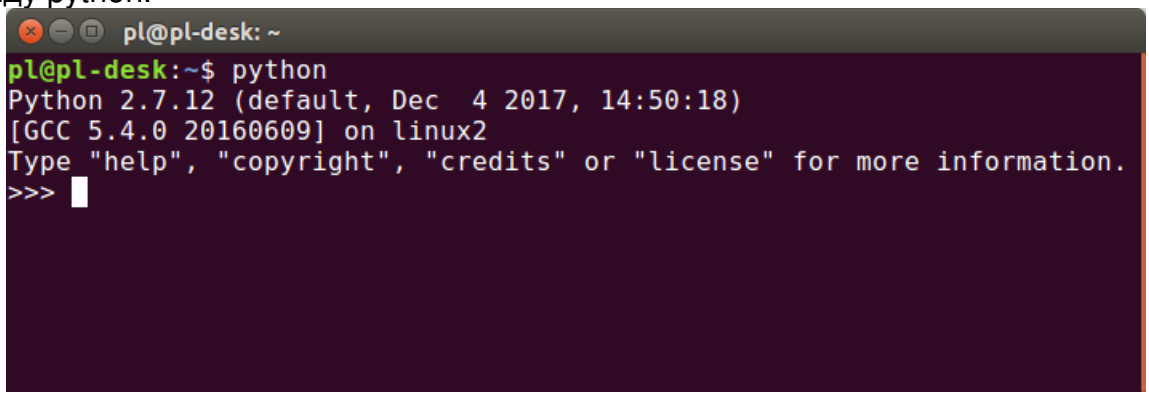
Как писать программы на Python

Интерактивный режим

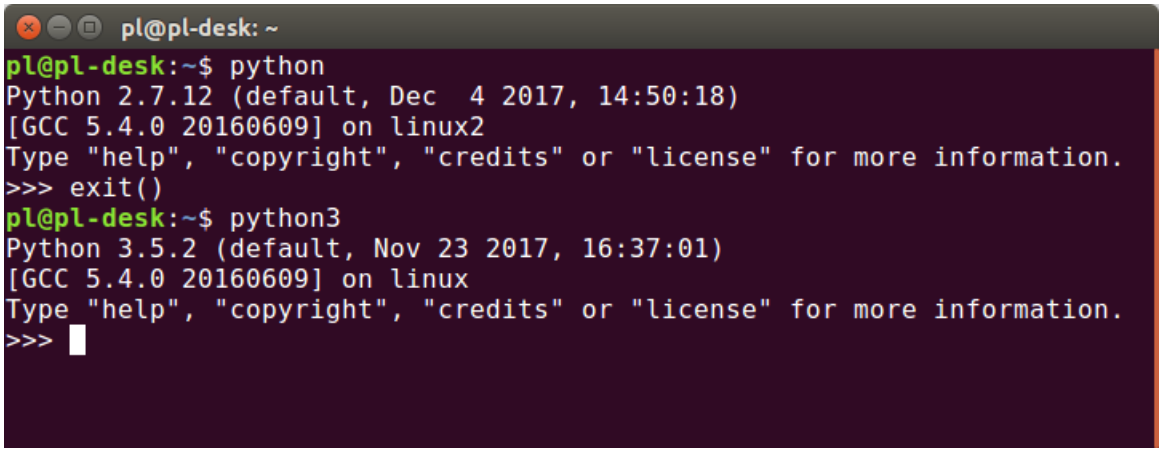
Грубо говоря, интерпретатор выполняет команды построчно. Пишешь строку, нажимаешь Enter, интерпретатор выполняет ее, наблюдаешь результат.

Это удобно, когда изучаешь особенности языка или тестирует какую-нибудь небольшую часть кода. Ведь если работать на компилируемом языке, то пришлось бы сначала создать файл с кодом на исходном языке программирования, затем передать его компилятору, получить от него исполняемый файл и только потом выполнить программу и оценить результат. К счастью, даже в случае с компилируемыми языками все эти действия выполняет среда разработки, что упрощает жизнь программиста.

В операционных системах на базе ядра Linux можно программировать на Python в интерактивном режиме с помощью приложения «Терминал», в котором работает командная оболочка Bash. Здесь, чтобы запустить интерпретатор, надо выполнить команду python.



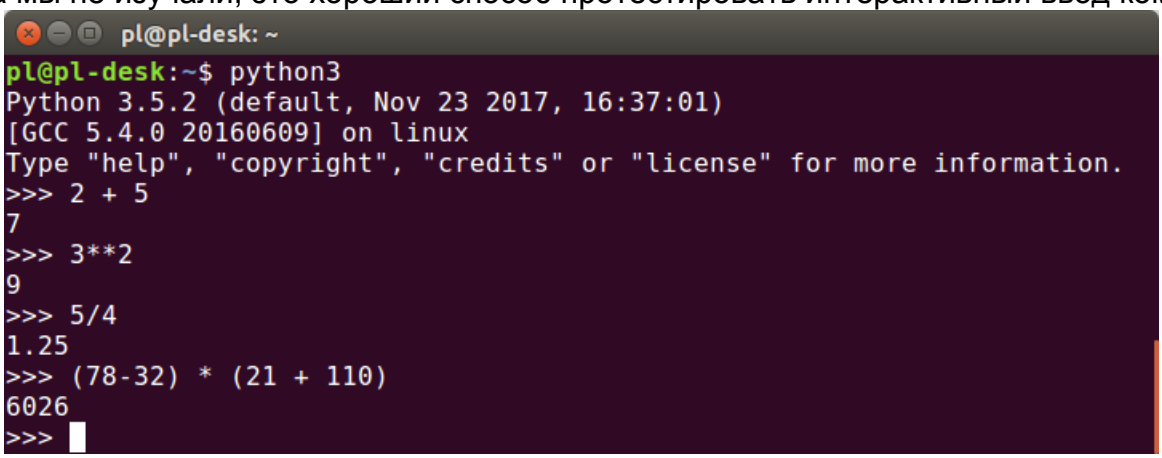
Скорее всего запустится интерпретатор второй ветки Питона, что можно увидеть в первой информационной строке. В данном случае запустилась версия 2.7.12. Первое число «2» указывает на то, что это интерпретатор для языка программирования Python 2. Последняя строка с тремя угловыми скобками (>>>) – это приглашение для ввода команд. Поскольку в данном курсе будет использоваться язык Python 3, выйдем из интерпретатора с помощью команды exit() (exit – выход). После чего выполним в терминале команду python3.



Есть вероятность, что пакет python3 может быть не установлен. Вам придется самостоятельно установить его.

Для операционных систем семейства Windows надо скачать интерпретатор с официального сайта языка (https://www.python.org/downloads/windows/). После установки он будет запускаться по ярлыку. Использовать командную оболочку здесь не требуется.

Возможности Python позволяют использовать его как калькулятор. Поскольку команды языка мы не изучали, это хороший способ протестировать интерактивный ввод команд.

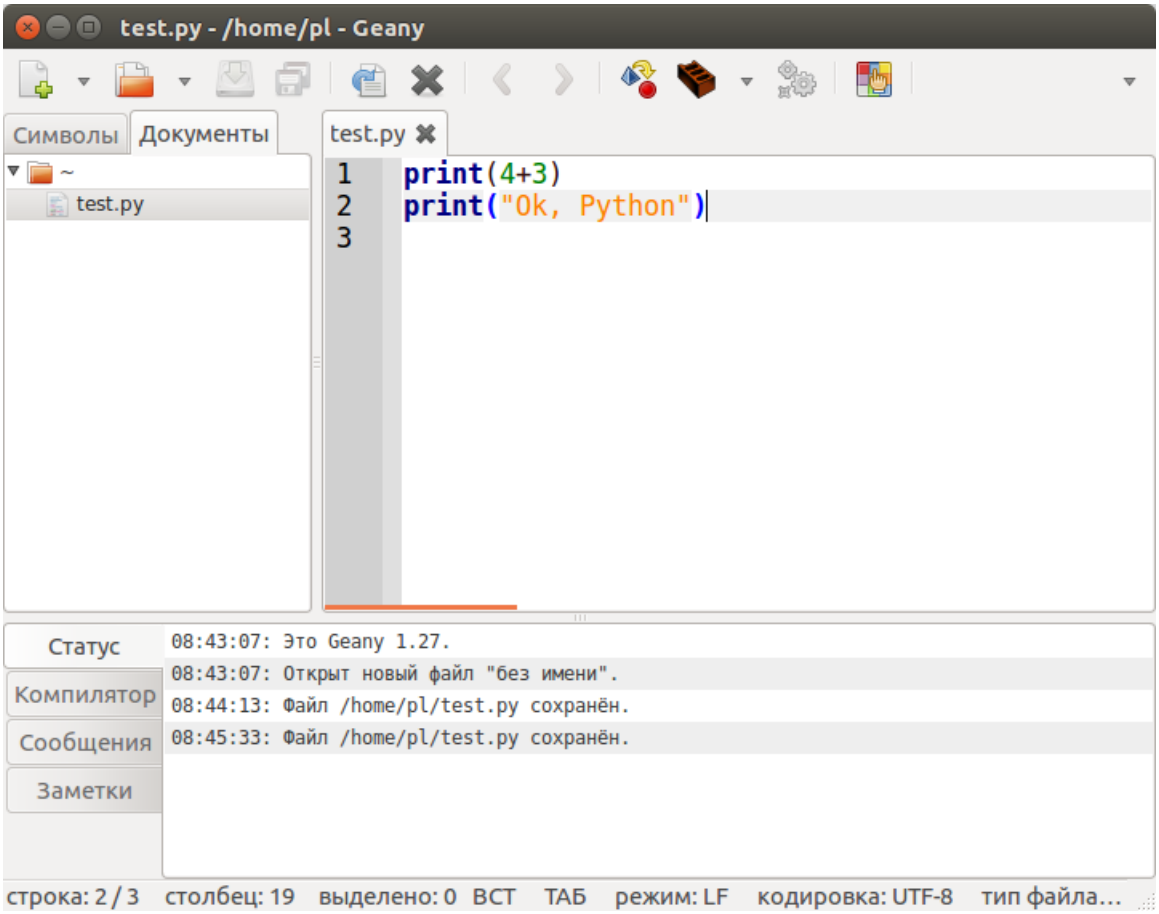


Бывает, что в процессе ввода была допущена ошибка или требуется повторить ранее используемую команду. Чтобы заново не вводить строку, в консоли можно прокручивать историю команд, используя для этого стрелки вверх и вних на клавиатуре. В среде IDLE (в Windows) для этого используются сочетания клавиш (скорее всего Alt+N и Alt+P).

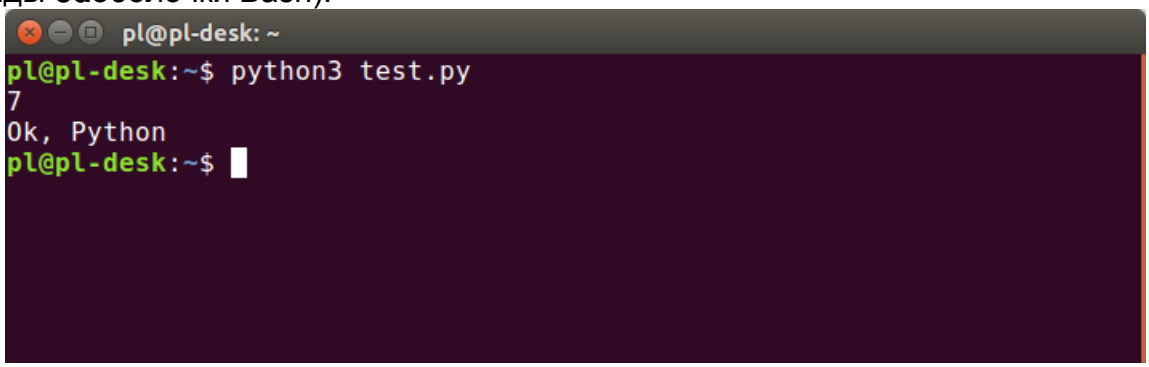
Создание скриптов

Несмотря на удобства интерактивного режима, чаще всего необходимо сохранить исходный программный код для последующего выполнения и использования. В таком случае подготавливаются файлы, которые передаются затем интерпретатору на исполнение. Файлы с кодом на Python обычно имеют расширение .py.

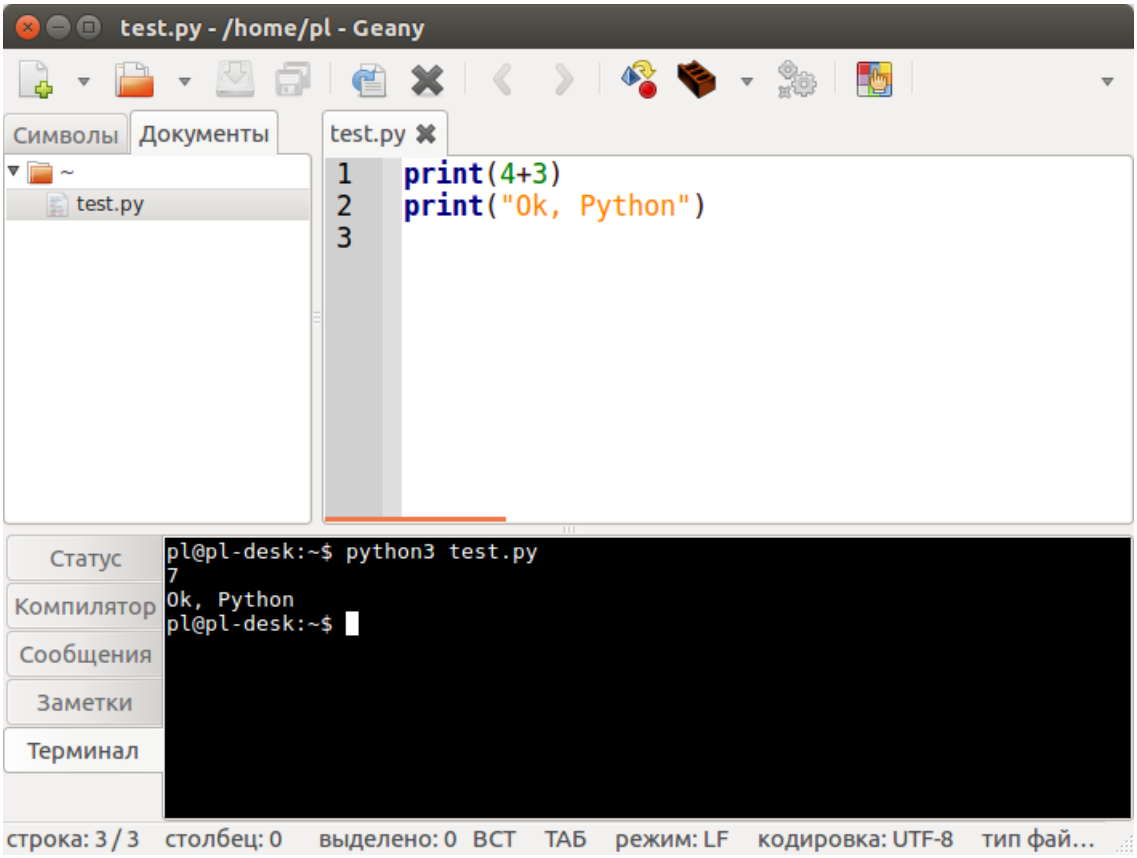
Существует целый ряд сред разработки для Python, например, PyCharm. Однако на первое время подойдет текстовый редактор с подсветкой синтаксиса, например, Geany.



Здесь создается и сохраняется файл кодом. Далее его можно запустить на выполнение через терминал. При этом сначала указывается интерпретатор (в данном случае python3), потом имя файла (если файл находится в другом каталоге, то указывается с адресом, или надо перейти в этот каталог с помощью команды **cd**оболочки Bash).



Однако в Geany можно дополнительно установить встроенный терминал (**sudo apt-get install** libvte9), что упростит работу.



Наконец, в редакторе можно просто нажать F5, что отправит файл на исполнение (терминал откроется сам, после выполнения программы и нажатия Enter закроется).

В Windows подготовить файлы можно в той же среде IDLE. Для этого в меню следует выбрать команду **File → New Window** (Crtl + N), откроется чистое (без приглашения >>>) новое окно. Желательно сразу сохранить файл с расширением .py, чтобы появилась подсветка синтаксиса. После того как код будет подготовлен, снова сохраните файл. Запуск скрипта выполняется командой **Run → Run Module** (F5). После этого в окне интерактивного режима появится результат выполнения кода.

Практическая работа

1. Запустите интерпретатор Питона в интерактивном режиме. Выполните несколько команд, например, арифметические примеры.
2. Подготовьте файл с кодом и передайте его на исполнение интерпретатору. Обратите внимание, что если просто записать арифметику, то никакого вывода не последует. Вы увидите пустоту. Это отличается от интерактивного режима. Чтобы увидеть решение, надо «обернуть» пример в функцию print().

Методические материалы по теме «Основы языка программирования Python. Типы данных»

Данные и их типы

В реальной жизни мы совершаем различные действия над окружающими нас предметами, или объектами. Мы меняем их свойства, наделяем новыми функциями. По аналогии с этим компьютерные программы также манипулируют объектами, только виртуальными, цифровыми. Пока не дойдем до уровня объектно-ориентированного программирования, будем называть такие объекты **данными**.

Очевидно, данные бывают разными. Часто компьютерной программе приходится работать с числами и строками. Так на прошлом уроке мы работали с числами, выполняя над ними арифметические операции. Операция сложения выполняла изменение первого числа на величину второго, а умножение увеличивало одно число в количество раз, соответствующее второму.

Числа в свою очередь также бывают разными: целыми, вещественными, могут иметь огромное значение или очень длинную дробную часть.

При знакомстве с языком программирования Python мы столкнемся с тремя типами данных:

- **целые числа** (тип int) – положительные и отрицательные целые числа, а также 0 (например, 4, 687, -45, 0).
- **числа с плавающей точкой** (тип float) – дробные, они же вещественные, числа (например, 1.45, -3.789654, 0.00453). Примечание: для разделения целой и дробной частей здесь используется точка, а не запятая.
- **строки** (тип str) — набор символов, заключенных в кавычки (например, "ball", "What is your name?", 'dkfjUUv', '6589'). Примечание: кавычки в Python могут быть одинарными или двойными; единственный символ в кавычках также является строкой, отдельного символьного типа в Питоне нет.

Операции в программировании

Операция – это выполнение каких-либо действий над данными, которые в данном случае именуют **операндами**. Само действие выполняет **оператор** – специальный инструмент. Если бы вы выполняли операцию постройки стола, то вашими операндами были бы доска и гвоздь, а оператором – молоток.

Так в математике и программировании символ плюса является оператором операции сложения по отношению к числам. В случае строк этот же оператор выполняет операцию *конкатенации*, т. е. соединения.

```
>>> 10.25 + 98.36
108.61
>>> 'Hello' + 'World'
'HelloWorld'
```

Здесь следует для себя отметить, что то, что делает оператор в операции, зависит не только от него, но и от типов данных, которыми он оперирует. Молоток в случае нападения на вас крокодила перестанет играть роль строительного инструмента. Однако в большинстве случаев операторы не универсальны. Например, знак плюса неприменим, если операндами являются, с одной стороны, число, а с другой – строка.

```
>>> 1 + 'a'
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Здесь в строке TypeError: unsupported operand type(s) **for** +: 'int' **and** 'str' интерпретатор сообщает, что произошла ошибка типа – неподдерживаемый операнд для типов int и str.

Изменение типов данных

Приведенную выше операцию все-таки можно выполнить, если превратить число 1 в строку "1". Для изменения одних типов данных в другие в языке Python предусмотрен ряд встроенных в него функций (что такое функция в принципе, вы узнаете в других уроках). Поскольку мы пока работаем только с тремя типами (int, float и str), то рассмотрим только соответствующие им функции: int(), float(), str().

```
>>> str(1) + 'a'
'1a'
>>> int('3') + 4
7
>>> float('3.2') + int('2')
5.2
```

```
>>> str(4) + str(1.2)
'41.2'
```
Эти функции преобразуют то, что помещается в их скобки соответственно в целое число, вещественное число или строку. Однако надо понимать, что преобразовать можно не все:
```
>>> int('hi')
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'hi'
```
Здесь возникла ошибка значения (ValueError), так как передан литерал (в данном случае строка с буквенными символами), который нельзя преобразовать к числу с основанием 10. Однако функция int() не такая простая:
```
>>> int('101', 2)
5
>>> int('F', 16)
15
```
Если вы знаете о различных системах счисления, то поймете, что здесь произошло.
Обратите внимание еще на одно. Данные могут называться **значениями**, а также **литералами**. Эти три понятия ("данные", "значение", "литерал") не обозначают одно и то же, но близки и нередко употребляются как синонимы. Чтобы понять различие между ними, места их употребления, надо изучить программирование глубже.

Переменные

Данные хранятся в ячейках памяти компьютера. Когда мы вводим число, оно помещается в какую-то ячейку памяти. Но как потом узнать, куда именно? Как впоследствии обращаться к этим данными? Нужно как-то запомнить, пометить соответствующую ячейку.
Раньше, при написании программ на машинном языке, обращение к ячейкам памяти осуществляли с помощью указания их регистров, т. е. конкретно сообщали, куда положить данные и откуда их взять. Однако с появлением ассемблеров при обращении к данным стали использовать словесные **переменные**, что куда удобней для человека.
Механизм связи между переменными и данными может различаться в зависимости от языка программирования и типов данных. Пока достаточно запомнить, что в программе данные связываются с каким-либо именем и в дальнейшем обращение к ним возможно по этому имени-переменной.
Слово "переменная" обозначает, что сущность может меняться, она непостоянна. Действительно, вы увидите это в дальнейшем, одна и та же переменная может быть связана сначала с одними данными, а потом – с другими. То есть ее значение может меняться, она переменчива.
В программе на языке Python, как и на большинстве других языков, связь между данными и переменными устанавливается с помощью знака **=**. Такая операция называется **присваивание** (также говорят "присвоение"). Например, выражение sq = 4 означает, что на объект, представляющий собой число 4, находящееся в определенной области памяти, теперь ссылается переменная sq, и обращаться к этому объекту следует по имени sq.

Имена переменных могут быть любыми. Однако есть несколько общих правил их написания:

1. Желательно давать переменным осмысленные имена, говорящие о назначении данных, на которые они ссылаются.
2. Имя переменной не должно совпадать с командами языка (зарезервированными ключевыми словами).
3. Имя переменной должно начинаться с буквы или символа подчеркивания (_), но не с цифры.
4. Имя переменной не должно содержать пробелы.

Чтобы узнать значение, на которое ссылается переменная, находясь в режиме интерпретатора, достаточно ее вызвать, т. е. написать имя и нажать Enter.

```
>>> sq = 4
>>> sq
4
```

Вот более сложный пример работы с переменными в интерактивном режиме:

```
>>> apples = 100
>>> eat_day = 5
>>> day = 7
>>> apples = apples - eat_day * day
>>> apples
65
```

Здесь фигурируют три переменные: apples, eat_day и day. Каждой из них присваивается свое значение. Выражение apples = apples - eat_day * day сложное. Сначала выполняется подвыражение, стоящее справа от знака равенства. После этого его результат присваивается переменной apples, в результате чего ее старое значение (100) теряется. В подвыражении apples - eat_day * day вместо имен переменных на самом деле используются их значения, т. е. числа 100, 5 и 7.

Практическая работа

1. Переменной var_int присвойте значение 10, var_float - значение 8.4, var_str - "No".
2. Измените значение, хранимое в переменной var_int, увеличив его в 3.5 раза, результат свяжите с переменной big_int.
3. Измените значение, хранимое в переменной var_float, уменьшив его на единицу, результат свяжите с той же переменной.
4. Разделите var_int на var_float, а затем big_int на var_float. Результат данных выражений не привязывайте ни к каким переменным.
5. Измените значение переменной var_str на "NoNoYesYesYes". При формировании нового значения используйте операции конкатенации (+) и повторения строки (*).
6. Выведите значения всех переменных.

Методические материалы по теме «Основы языка программирования Python. Логические выражения и операторы»

Логические выражения и логический тип данных

Часто в реальной жизни мы соглашаемся с каким-либо утверждением или отрицаем его. Например, если вам скажут, что сумма чисел 3 и 5 больше 7, вы согласитесь,

скажете: «Да, это правда». Если же кто-то будет утверждать, что сумма трех и пяти меньше семи, то вы расцените такое утверждение как ложное.
Подобные фразы предполагают только два возможных ответа – либо "да", когда выражение оценивается как правда, истина, либо "нет", когда утверждение оценивается как ошибочное, ложное. В программировании и математике **если результатом вычисления выражения может быть лишь истина или ложь, то такое выражение называется логическим**.
Например, выражение 4 > 5 является логическим, так как его результатом является либо правда, либо ложь. Выражение 4 + 5 не является логическим, так как результатом его выполнения является число.
На прошлом уроке мы познакомились с тремя типами данных – целыми и вещественными числами, а также строками. Сегодня введем четвертый – **логический тип данных** (тип bool). Его также называют булевым. У этого типа всего два возможных значения: **True** (правда) и **False** (ложь).

```
>>> a = True
>>> type(a)
<class 'bool'>
>>> b = False
>>> type(b)
<class 'bool'>
```

Здесь переменной a было присвоено значение True, после чего с помощью встроенной в Python функции type() проверен ее тип. Интерпретатор сообщил, что это переменная класса bool. Понятия "класс" и "тип данных" в данном случае одно и то же. Переменная b также связана с булевым значением.
В программировании False обычно приравнивают к нулю, а True – к единице. Чтобы в этом убедиться, можно преобразовать булево значение к целочисленному типу:

```
>>> int(True)
1
>>> int(False)
0
```

Возможно и обратное. Можно преобразовать какое-либо значение к булевому типу:

```
>>> bool(3.4)
True
>>> bool(-150)
True
>>> bool(0)
False
>>> bool(' ')
True
>>> bool('')
False
```

И здесь работает правило: всё, что не 0 и не пустота, является правдой.
Логические операторы
Говоря на естественном языке (например, русском) мы обозначаем сравнения словами "равно", "больше", "меньше". В языках программирования используются специальные знаки, подобные тем, которые используются в математике: > (больше), < (меньше), >= (больше или равно), <= (меньше или равно), == (равно), != (не равно).
Не путайте операцию присваивания значения переменной, обозначаемую в языке Python одиночным знаком "равно", и операцию сравнения (два знака "равно"). Присваивание и сравнение – разные операции.

```
>>> a = 10
>>> b = 5
>>> a + b > 14
True
```

```
>>> a < 14 - b
False
>>> a <= b + 5
True
>>> a != b
True
>>> a == b
False
>>> c = a == b
>>> a, b, c
(10, 5, False)
```

В данном примере выражение c = a== b состоит из двух подвыражений. Сначала происходит сравнение (==) переменных a и b. После этого результат логической операции присваивается переменной c. Выражение a,b, c просто выводит значения переменных на экран.

Сложные логические выражения

Логические выражения типа kByte >= 1023 являются простыми, так как в них выполняется только одна логическая операция. Однако, на практике нередко возникает необходимость в более сложных выражениях. Может понадобиться получить ответа "Да" или "Нет" в зависимости от результата выполнения двух простых выражений. Например, "на улице идет снег или дождь", "переменная news больше 12 и меньше 20". В таких случаях используются специальные операторы, объединяющие два и более простых логических выражения. Широко используются два оператора – так называемые логические И (**and**) и ИЛИ (**or**).

Чтобы получить True при использовании оператора **and**, необходимо, чтобы результаты обоих простых выражений, которые связывает данный оператор, были истинными. Если хотя бы в одном случае результатом будет False, то и все сложное выражение будет ложным.

Чтобы получить True при использовании оператора **or**, необходимо, чтобы результат хотя бы одного простого выражения, входящего в состав сложного, был истинным. В случае оператора **or** сложное выражение становится ложным лишь тогда, когда ложны оба составляющие его простые выражения.

Допустим, переменной x было присвоено значение 8 (x = 8), переменной y присвоили 13 (y = 13). Логическое выражение y < 15 and x > 8 будет выполняться следующим образом. Сначала выполнится выражение y < 15. Его результатом будет True. Затем выполнится выражение x > 8. Его результатом будет False. Далее выражение сведется к True and False, что вернет False.

```
>>> x = 8
>>> y = 13
>>> y < 15 and x > 8
False
```

Если бы мы записали выражение так: x > 8 and y < 15, то оно также вернуло бы False. Однако сравнение y < 15 не выполнялось бы интерпретатором, так как его незачем выполнять. Ведь первое простое логическое выражение (x > 8) уже вернуло ложь, которая, в случае оператора and, превращает все выражение в ложь.

В случае с оператором or второе простое выражение проверяется, если первое вернуло ложь, и не проверяется, если уже первое вернуло истину. Так как для истинности всего выражения достаточно единственного True, неважно по какую сторону от or оно стоит.

```
>>> y < 15 or x > 8
True
```

В языке Python есть еще унарный логический оператор **not**, т. е. отрицание. Он превращает правду в ложь, а ложь в правду. Унарный он потому, что применяется к

одному выражению, стоящему после него, а не справа и слева от него как в случае бинарных and и or.

```
>>> not y < 15
False
```

Здесь y < 15 возвращает True. Отрицая это, мы получаем False.

```
>>> a = 5
>>> b = 0
>>> not a
False
>>> not b
True
```

Число 5 трактуется как истина, отрицание истины дает ложь. Ноль приравнивается к False. Отрицание False дает True.

Практическая работа

1. Присвойте двум переменным любые числовые значения.
2. Составьте два сложных логических выражения с помощью оператора **and**, одно из которых должно давать истину, а другое – ложь.
3. Аналогично выполните п. 2, но уже используя оператор **or**.
4. Попробуйте использовать в логических выражениях переменные строкового типа. Объясните результат.

Методические материалы по теме «Основы языка программирования Python. Условный оператор»

Ход выполнения программы может быть линейным, т.е. таким, когда выражения выполняются, начиная с первого и заканчивая последним, по порядку, не пропуская ни одной строки кода. Но чаще бывает совсем не так. При выполнении программного кода некоторые его участки могут быть пропущены. Чтобы лучше понять почему, проведем аналогию с реальной жизнью. Допустим, человек живет по расписанию (можно сказать, расписание — это своеобразный "программный код", который следует выполнить). В его расписании в 18.00 стоит поход в бассейн. Однако человеку поступает информация, что воду слили, и бассейн не работает. Вполне логично отменить свое занятие по плаванию. Т.е. одним из условий посещения бассейна должно быть его функционирование, иначе должны выполняться другие действия.

Похожая нелинейность действий может быть предусмотрена и в компьютерной программе. Например, часть кода должна выполняться лишь при определенном значении конкретной переменной. Обычно в языках программирования используется приблизительно такая конструкция:

```
if a логический_оператор b :
    выражения, выполняемые при результате True
    в логическом выражении
```

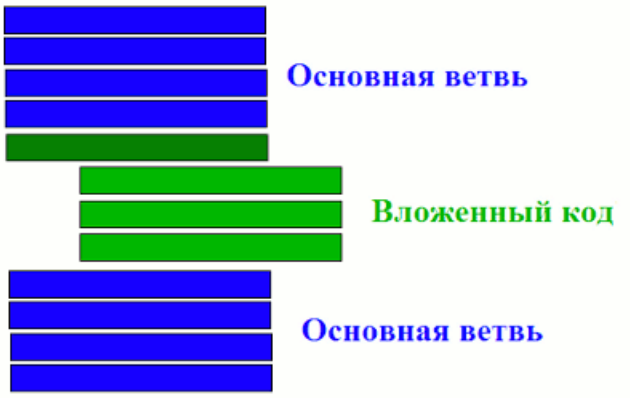
Пример ее реализации на языке программирования Python:

```
if numbig < 100: # если значение переменной numbig меньше 100, то
...
    c = a**b     # возвести значение переменной a в степень b,
                 # результат присвоить c.
```

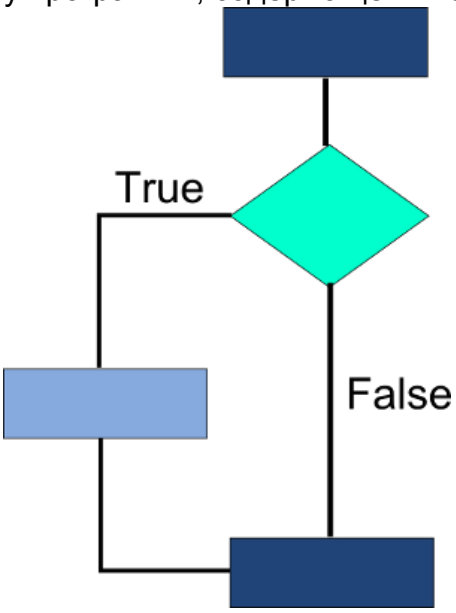
Первая строка конструкции **if** — это заголовок, в котором проверяется условие выполнения строк кода после двоеточия (тела конструкции). В примере выше тело содержит всего лишь одно выражение, однако чаще их бывает куда больше.

Про Python говорят, что это язык программирования с достаточно ясным и легко читаемым кодом. Это связано с тем, что в нем сведены к минимуму вспомогательные элементы (скобки, точка с запятой), а для разделения синтаксических конструкций используются отступы от начала строки. Учитывая это, в конструкции **if** код, который

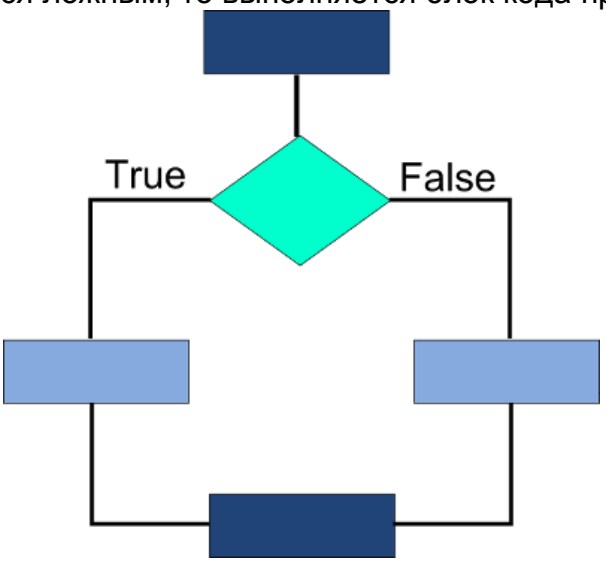
выполняется при соблюдении условия, должен обязательно иметь отступ вправо. Остальной код (основная программа) должен иметь тот же отступ, что и слово **if**. Обычно отступ делается с помощью клавиши Tab.



Можно изобразить блок-схему программы, содержащей инструкцию **if**, в таком виде:



Встречается и более сложная форма ветвления: **if–else**. Если условие при инструкции **if** оказывается ложным, то выполняется блок кода при инструкции **else**.



Пример кода с веткой **else** на языке программирования Python:

```
print "Привет"
tovar1 = 50
tovar2 = 32
if tovar1+ tovar2 > 99 :
```

```
    print "Сумма не достаточна"
else:
    print "Чек оплачен"
print "Пока"
```

Практическая работа

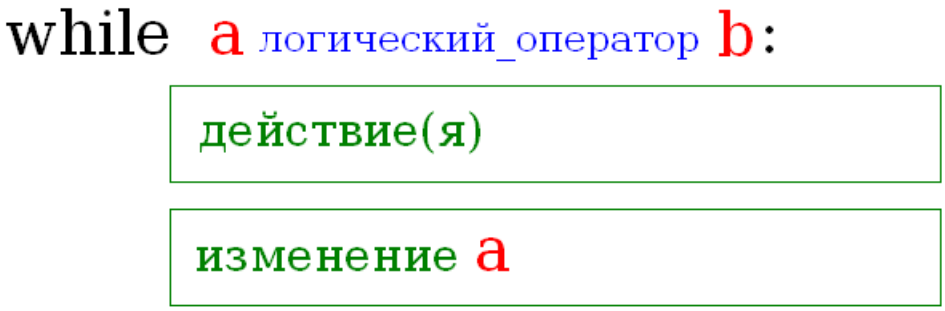
1. Напишите программный код, в котором в случае, если значение некой переменной больше 0, выводилось бы специальное сообщение (используйте функцию **print**). Один раз выполните программу при значении переменной больше 0, второй раз — меньше 0.
2. Усовершенствуйте предыдущий код с помощью ветки **else** так, чтобы в зависимости от значения переменной, выводилась либо 1, либо -1.
3. Самостоятельно придумайте программу, в которой бы использовалась инструкция **if**(желательно с веткой **else**). Вложенный код должен содержать не менее трех выражений.

Методические материалы по теме «Основы языка программирования Python. Цикл While»

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

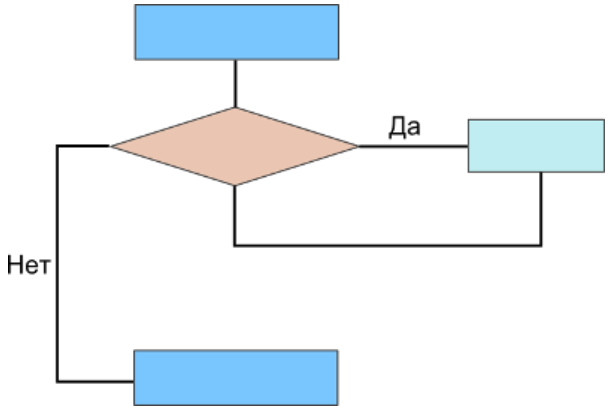
В реальной жизни мы довольно часто сталкиваемся с циклами. Например, ходьба человека — вполне циклическое явление: шаг левой, шаг правой, снова левой-правой и т.д., пока не будет достигнута определенная цель (например, школа или магазин). В компьютерных программах наряду с инструкциями ветвлениями (т.е. выбором пути действия) также существуют инструкции циклов (повторения действия). Если бы инструкций цикла не существовало, пришлось бы много раз вставлять в программу один и тот же код подряд столько раз, сколько нужно выполнить одинаковую последовательность действий.

Универсальным организатором цикла в языке программирования Python (как и во многих других языках) является конструкция **while**. Слово "while" с английского языка переводится как "пока" ("пока логическое выражение возвращает истину, выполнять определенные операции"). Конструкцию **while** на языке Python можно описать следующей схемой:



Эта схема приблизительна, т.к. логическое выражение в заголовке цикла **while** может быть более сложным, а изменяться может переменная (или выражение) b.

Может возникнуть вопрос: "Зачем изменять a или b?". Когда выполнение программного кода доходит до цикла **while**, выполняется логическое выражение в заголовке, и, если было получено **True** (истина), выполняются вложенные выражения. После поток выполнения программы снова возвращается в заголовок цикла **while**, и снова проверяется условие. Если условие никогда не будет ложным, то не будет причин остановки цикла и программа *зациклится*. Чтобы этого не произошло, необходимо предусмотреть возможность выхода из цикла — ложность выражения в заголовке. Таким образом, изменяя значение переменной в теле цикла, можно довести логическое выражение до ложности.



Эту изменяемую переменную, которая используется в заголовке цикла **while**, обычно называют счетчиком. Как и всякой переменной ей можно давать произвольные имена, однако очень часто используют буквы i и j. Простейший цикл на языке программирования Python может выглядеть так:

```
str1 = "+"
i = 0
while i < 10:
     print (str1)
     i = i + 1
```

В последней строчке кода происходит увеличение значения переменной i на единицу, поэтому с каждым оборотом цикла ее значение увеличивается. Когда будет достигнуто число 10, логическое выражение i < 10 даст ложный результат, выполнение тела цикла будет прекращено, а поток выполнения программы перейдет на команды следующие за всей конструкцией цикла. Результатом выполнения скрипта приведенного выше является вывод на экран десяти знаков + в столбик. Если увеличивать счетчик в теле цикла не на единицу, а на 2, то будет выведено только пять знаков, т.к цикл сделает лишь пять оборотов.

Более сложный пример с использованием цикла:

```
fib1 = 0
fib2 = 1
print (fib1)
print (fib2)
n = 10
i = 0
while i < n:
     fib_sum = fib1 + fib2
     print (fib_sum)
     fib1 = fib2
     fib2 = fib_sum
     i = i + 1
```

Этот пример выводит *числа Фибоначчи* — ряд чисел, в котором каждое последующее число равно сумме двух предыдущих: 0, 1, 1, 2, 3, 5, 8, 13 и т.д. Скрипт выводит двенадцать членов ряда: два (0 и 1) выводятся вне цикла и десять выводятся в результате выполнения цикла.

Как это происходит? Вводятся две переменные (fib1 и fib2), которым присваиваются начальные значения. Присваиваются значения переменной n и счетчику i, между которыми те или иные математические отношения формируют желаемое число витков цикла. Внутри цикла создается переменная fib_sum, которой присваивается сумма двух предыдущих членов ряда, и ее же значение выводится на экран. Далее изменяются значения fib1 и fib2 (первому присваивается второе, а второму - сумма), а также увеличивается значение счетчика.

Практическая работа

1. Напишите скрипт на языке программирования Python, выводящий ряд чисел Фибоначчи (см. пример выше). Запустите его на выполнение. Затем измените код так, чтобы выводился ряд чисел Фибоначчи, начиная с пятого члена ряда и заканчивая двадцатым.
2. Напишите цикл, выводящий ряд четных чисел от 0 до 20. Затем, каждое третье число в ряде от -1 до -21.
3. Самостоятельно придумайте программу на Python, в которой бы использовался цикл **while**.