Основы программирования на Python. Элективный курс по информатике

Пояснительная записка

Элективный курс по информатике "Основы программирования на Python" представляет собой вводный курс по программированию, дающий представление о базовых понятиях структурного программирования (данных, переменных, ветвлениях, циклах и функциях). Python – это язык, обладающий рядом преимуществ перед другими языками для начинающих изучать программирование (ясность кода, быстрота реализации).

Место курса "Основы программирования на Python" в составе образовательной программы дополнительного образования детей "Лаборатория юного линуксоида" — первый год обучения.

Курс рассчитан примерно на 16-18 часов.

Цели и задачи курса

Основной целью данного элективного курса является формирование базовых понятий структурного программирования, развитие логики обучающихся. Курс не является полноценным учебником по Python.

Программа элективного курса

Представление об языках программирования и их историческом развитии. Способы трансляции программного кода. Типы данных (целые числа, числа с плавающей точкой, строки) и структуры данных (строки, списки, словари), переменные, выражения, ветвления (if, if-else, if-elif-else) и циклы (while, for) в программах. Ввод и вывод данных. Понятие о функции, локальных и глобальных переменных.

- История языков программирования. Компиляция и интерпретация. Урок 1
- Введение в язык программирования Python. Урок 2
- Типы данных в программировании. Определение переменной. Урок 3
- Логические выражения. Урок 4
- Условный оператор. Инструкция if. Урок 5
- Множественное ветвление. Урок 6
- Цикл While. Урок 7
- Ввод данных с клавиатуры. Урок 8
- Последовательности: строки. Урок 9
- Последовательности: списки. Урок 10
- Структуры данных: словари. Урок 11
- Цикл for в языке программирования Python. Урок 12

- Функции в языке программирования Python. Урок 13
- Параметры и аргументы функций. Локальные и глобальные переменные. Урок 14
- Задания по программированию на Python. Урок 15

История языков программирования. Компиляция и интерпретация. Урок 1

Конспект урока
Курс: Основы программирования на Python
Уровень: Программирование для начинающих

1. Программа. Язык программирования

Программу можно представить как набор последовательных команд (алгоритм) для определенного объекта (исполнителя), который должен их выполнить для достижения той или иной цели. Например, условно запрограммировать можно человека, если составить ему инструкцию "как собрать компьютер", и он примется ее исполнять. Очевидно, что инструкция будет на естественном языке (русском, английском или др.). Однако, обычно принято программировать не людей, а вычислительные машины. Трудность заключается в том, что такие машина не в состоянии понять наш язык. Для "инструктирования" вычислительных машин разработаны и разрабатываются специальные языки, называемые **языками программирования**. Языки программирования характеризуются однозначностью при формировании фразы (в выражении нельзя менять слова местами), а также ограниченным набором слов-команд. Другими словами, языки программирования являются **формальными**.

2. Основные этапы развития языков программирования

Машинный язык — это единственный способ взаимодействия с электронно-вычислительными машинами. Первые программы писались именно на нем, т. к. других средств «общения» человека и компьютера еще не было. Каждую команду машинного языка выполняет определенное электронное устройство. Данные и команды записываются в цифровом виде (например, в шестнадцатеричной или двоичной системах счисления). Понять программу на таком языке человеку очень сложно; к тому же даже небольшая программа будет состоять из множества строк кода. В довершении всего, у каждой вычислительной машины свой машинный язык; следовательно программа, написанная для одной ЭВМ, не будет работать на другой (придется писать снова).

Людям, в отличие от машин, более понятны слова, чем наборы цифр. Стремление человека оперировать словами и не цифрами привело к появлению **ассемблеров**. Это языки, в которых вместо численного обозначения команд и областей памяти используются буквенные.

Но тут сразу возникает проблема: машина не в состоянии понять набор букв. Необходим какой-нибудь «переводчик» на ее родной машинный язык. Так был придуман **транслятор** — специальная программа, преобразующая программный код с того или иного языка программирования в машинный код.

Ассемблеры и сегодня находят применение, т.к. системные программы (обслуживающие работу аппаратного обеспечения), написанные на ассемблере могут работать быстрее, чем аналогичные программы, написанные на других языках программирования.

После ассемблеров наступил рассвет языков так называемого **высокого уровня**. Для этих языков потребовалось разрабатывать более сложные трансляторы.

Программные коды, написанные на языках высокого уровня, обладают логичной структурой. Это облегчает разработку программы и ее отладку.

В отличие от ассемблеров, которые все еще остаются привязанными к своим типам машин, языки высоко уровня обладают **переносимостью**. Т.е., написав один раз программу, программист может выполнить ее на любой машине.

Следующим крупным этапом в эволюции программирования было появление **объектно-ориентированных языков** (ООП). Их отличие от языков высокого уровня заключается в возможности отстранения от алгоритма выполнения программы. С помощью таких языков разработчик как бы оперирует виртуальными объектами. На сегодняшний день, реализация больших и сложных проектов осуществляется в основном с помощью ООП.

3. Разнообразия языков программирования

Разнообразие языков весьма велико. Это может показаться странным: если написанную программу можно перенести на любую машину и выполнить там, то зачем придумывать такое множество языков?

Однако все немного сложнее. Можно сказать, что нет идеального языка, каждый чем-то хорош, а в чем-то уступает другому инструменту. Многие программисты старались и стараются придумать свой язык обладающий теми или иными преимуществами.

Можно лишь условно разделить языки по определенным критериям. Например, по типу решаемых задач (язык системного или прикладного назначения), по степени ориентации на решение узкого круга задач (проблемно-ориентированные или универсальные).

4. Трансляция

Ранее было сказано, что для перевода кода с одного языка программирования (например, высокого уровня) на другой (например, машинный язык) требуется специальная программа — **транслятор**.

Механизм этого перевода весьма сложен, однако выделяют два основных способа **трансляции** — это **компиляция** программы или ее **интерпретация**.

При компиляции исходный программный код сразу целиком переводится в машинный. Создается исполняемый файл, который уже никак не связан с исходным кодом. Выполнение исполняемого файла обеспечивается операционной системой самостоятельно.

При интерпретации выполнение кода происходит построчно. Интерпретатор, выполняя программу, напрямую взаимодействует с операционной системой.

Выполнение откомпилированной программы происходит быстрее, т.к. она представляет собой готовый машинный код. Однако на современных компьютерах снижение скорости выполнения при интерпретации обычно не заметна.

Введение в язык программирования Python. Урок 2

Конспект урока

Элективный курс: Основы программирования на Python

Уровень: Программирование для начинающих

Версия python: 2.-.-

История языка программирования

Python – это достаточно молодой язык программирования. Он был создан примерно в 1991 году голландцем Гвидо ван Россумом.

Свое название Пайтон (или Питон) получил от названия телесериала, а не пресмыкающегося. Однако по созвучию все-равно в качестве эмблемы был удостоен змеи.

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

Поскольку Python достаточно молодой язык программирования, то он активно совершенствуется и в настоящее время. Поэтому часто выходят новые версии. Официальный сайт http://python.org.

Особенности

Python – это интерпретируемый язык программирования. Т.е. программный код на языке Питон преобразуется в машинный код построчно специальной программой — интерпретатором.

Python имеет достаточно простой синтаксис. Читать код на этом языке программирования легко, т.к. в нем используется минимум вспомогательных элементов, а правила языка заставляют программистов делать отступы.

С другой стороны, Python – это полноценный, можно сказать универсальный, язык программирования. Это язык так называемого сверх высокого уровня: он поддерживает объектно-ориентированное программирование (на самом деле он и разрабатывался как объектно-ориентированный язык).

Также Python распространяется свободно под лицензией подобной GNU General Public License.

Дзэн питона

Если интерпретатору Питона дать команду import this ("импортировать это"), то выведется так называемый "Дзен Питона", иллюстрирующий идеологию и особенности данного языка.

1. Beautiful is better than ugly. Красивое лучше уродливого.
2. Explicit is better than implicit. Явное лучше неявного.
3. Simple is better than complex. Простое лучше сложного.
4. Complex is better than complicated. Сложное лучше усложнённого.
5. Flat is better than nested. Плоское лучше вложенного.
6. Sparse is better than dense. Разрежённое лучше плотного.
7. Readability counts. Удобочитаемость важна.
8. Special cases aren't special enough to break the rules. Частные случаи не настолько существенны, чтобы нарушать правила.
9. Although practicality beats purity. Однако практичность важнее чистоты.
10. Errors should never pass silently. Ошибки никогда не должны замалчиваться.
11. Unless explicitly silenced. За исключением замалчивания, которое задано явно.
12. In the face of ambiguity, refuse the temptation to guess. В случае неоднозначности сопротивляйтесь искушению угадать.
13. There should be one — and preferably only one — obvious way to do it. Должен существовать один — и, желательно, только один — очевидный способ сделать это.
14. 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.
15. Now is better than never. Сейчас лучше, чем никогда.
16. Although never is often better than *right* now. Однако, никогда чаще лучше, чем прямо сейчас.
17. If the implementation is hard to explain, it's a bad idea. Если реализацию сложно объяснить — это плохая идея.
18. If the implementation is easy to explain, it may be a good idea. Если реализацию легко объяснить — это может быть хорошая идея.

19. Namespaces are one honking great idea — let's do more of those! Пространства имён — прекрасная идея, давайте делать их больше!

Как писать программы

Интерактивный режим

Как уже было сказано интерпретатор выполняет команды построчно, т.е пишешь строку -> интерпретатор выполняет ее -> наблюдаешь результат и тихо радуешься.

Это очень удобно, когда человек только изучает программирование или тестирует какую-нибудь небольшую часть кода. Ведь если работать на компилируемом языке, то пришлось бы сначала написать код на исходном языке программирования, затем скомпилировать и уж потом запустить получившийся файл (с машинным кодом) на исполнение. Если окажется, что где-то в исходном коде была допущена ошибка, то придется перекомпилировать всю программу. В интерпретируемых языках такой проблемы нет.

Работать в интерактивном режиме в ОС Linux можно в консоли. Для этого следует выполнить команду **python**. Запустится интерпретатор, где сначала выведется информация о его версии и иная информация. Далее, приглашение к вводу (>>>).

Задание.*Запустите интерпретатор Питона.*

Поскольку никаких команд мы пока не знаем, то будем использовать Питон как калькулятор (возможности языка это позволяют).

```
1.   2+5
2.   3*(5-8)
3.   2.4 + 3.0/2
```

Наберите подобные примеры в интерактивном режиме (в конце каждого нажимайте Enter).

Ответ выдается сразу после нажатия Enter (завершения ввода команды).

Бывает, что в процессе ввода была допущена ошибка или требуется повторить ранее используемую команду. Чтобы не писать строку сначала, в консоли можно прокручивать список команд, используя для этого стрелки на клавиатуре.

Другой вариант работы в интерактивном режиме — это работа в среде разработки IDLE, у которой есть интерактивный режим работы. В отличие от консольного варианта здесь можно наблюдать подсветку синтаксиса (команды выделяются цветом). Прокручивать список команд можно с помощью комбинаций Alt+N, Alt+P.

Задание. *Запустите IDLE. Попробуйте решать математические примеры здесь.*

Создание скриптов

Несмотря на удобства интерактивного режима работы с Питоном, часто требуется сохранить исходный программный код для последующего использования. В таком случае подготавливаются файлы, которые передаются затем интерпретатору на исполнение. По отношению к интерпретируемым языкам программирования часто исходный код называют скриптом. Файлы с кодом на Python обычно имеют расширение ***.py**.

Подготовить скрипты можно в той же среде IDLE. Для этого, после запуска среды в меню нужно выбрать команду **File → New Window (Crtl + N)**, откроется новое окно. Затем желательно сразу сохранить файл (не забываем про расширение ***.py**). (Примечание: если набирать код, не сохранив файл в начале, то подсветка синтаксиса будет отсутствовать.) После того как код будет подготовлен, снова сохраните файл (чтобы обновить сохранение). Ну и наконец, можно запустить скрипт выполнив команду меню **Run → Run Module (F5)**. После этого в первом окне (где "работает" интерпретатор) появится результат выполнения кода.

Задание. *Подготовьте скрипт (с примерами). Запустите его на выполнение.*

На самом деле скрипты можно писать в любом текстовом редакторе (желательно, чтобы он поддерживал подсветку синтаксиса языка Python). Также существуют специальные программы для разработки, предоставляющие дополнительные возможности и удобства.

Запускать подготовленные файлы можно не только в IDLE, но и в консоли с помощью команды **python адрес/имя_файла**.

Задание. *В консоли передайте интерпретатору Питона ранее подготовленный файл на выполнение.*

Кроме того, существует возможность настроить выполнение скриптов с помощью двойного клика по файлу (в Windows данная возможность присутствует изначально).

Типы данных в программировании. Определение переменной. Урок 3

Методическая разработка урока
Элективный курс: Основы программирования на Python
Уровень: Программирование для начинающих

1. Данные, их типы

Можно заметить, что все, что мы делаем, мы делаем над чем-то — какими-то предметами или объектами. Мы изменяем объекты, их состояния и т.д. Можно провести аналогию с компьютером и сказать, что машина тоже манипулирует какими-то объектами (назовем их пока **данными**).

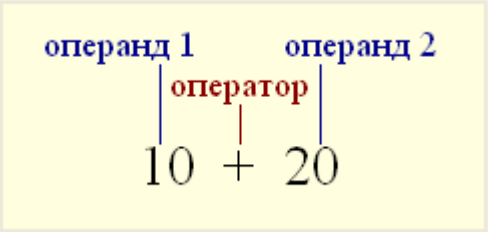
Понятно, что данные бывают разными. Очень часто компьютерной программе приходится работать с такими типами как число или строка (хотя это далеко не единственные типы данных). Например, на прошлом занятии мы «манипулировали» числами, выполняя над ними математические операции. Так, можно сказать, что при операции сложения мы выполняли изменение первого числа на величину второго, или при умножении — увеличивали одно число в количество раз, соответствующее второму.

Числа в свою очередь также бывают разными: целыми, дробными или могут иметь очень большое значение. В основном, при знакомстве с языком программирования Python нас буду интересовать три типа данных:

- **целые числа (integer)** – положительные и отрицательные целые числа, а также 0 (например, 4, 687, -45, 0);
- **числа с плавающей точкой (float point)** – дробные числа (например, 1.45, -3.789654, 0.00453). Примечание: разделителем целой и дробной части служит точка, а не запятая;
- **строки (string)** — набор символов, заключенных в кавычки (например, “ball”, “What is your name?”, 'dkfjUUv', '6589'). Примечание: кавычки могут быть одинарными или двойными.

2. Операции

Можно сказать, что **операция** — это выполнение каких-нибудь действия над данными (**операндами**). Для выполнения конкретных действий требуются специальные инструменты — **операторы**.



Например, запрограммировано, что символ «+» по отношению к числам выполняет операцию сложения, а по отношению к строкам — конкатенацию (соединение). Два знака «*» возводят первое число в степень второго.

Выражение	Результат выполнения
34.907 + 320.65	355.55699999999996
“Hi, ” + “world :)”	'Hi, world :)'
“Hi, ” * 10	'Hi, Hi, Hi, Hi, Hi, Hi, Hi, Hi, Hi, Hi, '

3. Изменение типа данных

А что будет, если мы попытаемся выполнить в одном выражении операцию с разными типами данным? Например, сложить целое и дробное число, число и строку. Однозначный ответ дать нельзя: так, при складывании целого числа и числа с плавающей точкой, получается число с плавающей точкой, а если попытаться сложить любое число и строку, то будет получена ошибка.

Выражение	Результат выполнения
1 + 0.65	1.6499999999999999
“Hi, ” + 15	О ш и б к а

Однако, бывают случаи, когда программа получает данные в виде чисел, а в результате должна выдавать строки (или наоборот). В таком случае можно использовать, так называемые, встроенные функции, позволяющие преобразовать один тип данных в другой. Так функция **int()** преобразует переданную ей строку (или число с плавающей точкой) в целое, функция **str()** преобразует переданный ей аргумент в строку, **float()** - в дробное число.

Выражение	Результат выполнения
int (“56”)	56
int (4.03)	4
int (“comp 486”)	О ш и б к а
str (56)	'56'
str (4.03)	'4.03'
float (56)	56.0
float (“56”)	56.0

4. Переменные

Все данные хранятся в ячейках памяти компьютера. Когда мы вводим число оно, помещается в память. Но мы же не знаем куда именно и просто теряем его. Вопрос: как обращаться к этим данными? Раньше, при написании программ на машинном языке, обращение к ячейкам памяти осуществляли с помощью указания регистров. Но уже с появлением ассемблеров, при обращении к данным стали использовать так называемые **переменные**. Механизм этой связи может различаться в разных языках программирования. На этом этапе достаточно запомнить, что данные связываются с каким-либо именем и в дальнейшем обращение к ним возможно по этому имени.

Связывание данных и имени (переменной) в языке программирования Python происходит с помощью знака "=" (операция **присваивания**). Например, запись sq = 4 означает, что объект (данные) в определенной области памяти связаны с именем sq и обращаться к ним теперь следует по этому имени.

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

1. Желательно давать переменным имена, несущие в себе смысловое значение.
2. Имя переменной не должно совпадать с командами языка (зарезервированными ключевыми словами).
3. Имя переменной должно начинаться с буквы или символа подчеркивания (_).

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

Пример работы с переменными:

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

5. Практическая работа

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

6. Примерный результат выполнения заданий

```
>>> tseloe = 10
>>> drobnoe = 8.4
>>> stroka = "No"
>>> big_netseloe = tseloe * 3.5
>>> drobnoe = drobnoe - 1
>>> tseloe / drobnoe
1.3513513513513513
>>> big_netseloe / drobnoe
4.7297297297297298
>>> stroka = stroka * 2 + "_" + "Yes" * 3
>>> tseloe
10
>>> drobnoe
7.4000000000000004
>>> big_netseloe
35.0
>>> stroka
'NoNo_YesYesYes'
>>>
```

7. Вопросы

1. Какие типы данных вы знаете? Дайте им краткую характеристику.
2. Можно ли преобразовать дробное число в целое? целое в дробное? В каких случаях можно строку преобразовать в число?
3. Приведите примеры операций. В каких случаях используется операция присвоения?
4. Для чего предназначены переменные? Как правильно дать переменной имя?

Логические выражения. Урок 4

Конспект урока
Элективный курс: Основы программирования на Python
Уровень: Программирование для начинающих

1. Логического выражения и логический тип данных

Часто в реальной жизни мы соглашаемся или отрицаем то или иное событие, факт и т.д. Например, "Сумма чисел 3 и 5 больше 7" является правдой или истинным выражением, "Сумма чисел 3 и 5 меньше 7" - ложным. Можно заметить, что подобные фразы предполагают только два результата: "Да" (правда) и "Нет" (ложь). Подобное есть и в программировании: если результатом вычисления выражения является "истина" или "ложь", то такое выражение называется логическим.

На прошлом уроке были отмечены три типа данных: целые и дробные числа, а также строки. Также выделяют логический тип данных. У этого типа всего два возможных значения: True (правда) — 1 и False (ложь) — 0. Эти значения и являются результатом логических выражений.

2. Логические операторы

Говоря на естественном языке (например, русском) мы обозначаем сравнение словами "равно", "больше", "меньше" и им подобными. В языках программирования обычно используются специальные знаки, подобные тем, которые используются в математических выражениях: > (больше), < (меньше), >= (больше или равно), <= (меньше или равно).

Новыми для восприятия могут оказаться обозначение равенства: == (два знака "равно"); а также неравенства !=. Часто начинающие программисты ошибаются и вместо двух знаков "=" пишут один. Однако, как вы должны помнить из предыдущего урока, один знак используется в операциях присвоения.

Примеры работы с логическими выражениями на языке программирования Python (после # написаны комментарии-пояснения):

```
1.   x = 12 - 5 # это не логическая операция, а операция присвоения переменной x результата выражения 12 — 5
2.   x == 4 # x равен 4
3.   x == 7 # x равен 7
4.   x != 7 # x не равен 7
5.   x != 4 # x не равен 4
```

```
6.   x > 5 # x больше 5
7.   x < 5 # x меньше 5
8.   x <= 6 # x больше или равен 6
9.   x >= 6 # x меньше или равен 6
```

Определите устно результаты выполнения операций, приведенных в примере выше. Проверьте правильность ваших предположений, выполнив данные выражения с помощью интерпретатора языка Python.

3. Сложные логические выражения

Выражения типа "skokomnogo >= 1023" является простым. Однако, на практике не редко используются более сложные логические выражения. Может понадобиться получить логический тип ответа ("Да" или "Нет") в зависимости от результата выполнения двух простых выражений. Например, "на улице идет снег или дождь", "переменная new больше 12 и меньше 20" и т.п.

В таких случаях требуются специальные операторы, объединяющие два и более простых логических выражения. Мы рассмотрим два способа объединения: через, так называемые, логические **И (and)** и **ИЛИ (or)**.

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

Чтобы получить истину (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 все составляющие его простые выражения.

Примеры работы со сложными логическими выражениями на языке программирования Python (после # написаны комментарии-пояснения):

```
1.   x = 8
2.   y = 13
3.   x == 8 and y < 15 # x равен 8 и y меньше 15
4.   x > 8 and y < 15 # x больше 8 и y меньше 15
5.   x != 0 or y >15 # x не равен 0 или y меньше 15
6.   x < 0 or y >15 # x меньше 0 или y меньше 15
```

Определите устно результаты выполнения операций, приведенных в примере выше. Проверьте правильность ваших предположений, выполнив данные выражения с помощью интерпретатора языка Python.

4. Практическая работа

1. Присвойте двум переменным любые числовые значения

2. Составьте четыре сложных логических выражения с помощью оператора and, два из которых должны давать истину, а два других - ложь.
3. Аналогично выполните п. 2, но уже используя оператор or.
4. Попробуйте использовать в сложных логических выражениях работу с переменными строкового типа.

5. Примерный результат выполнения задания

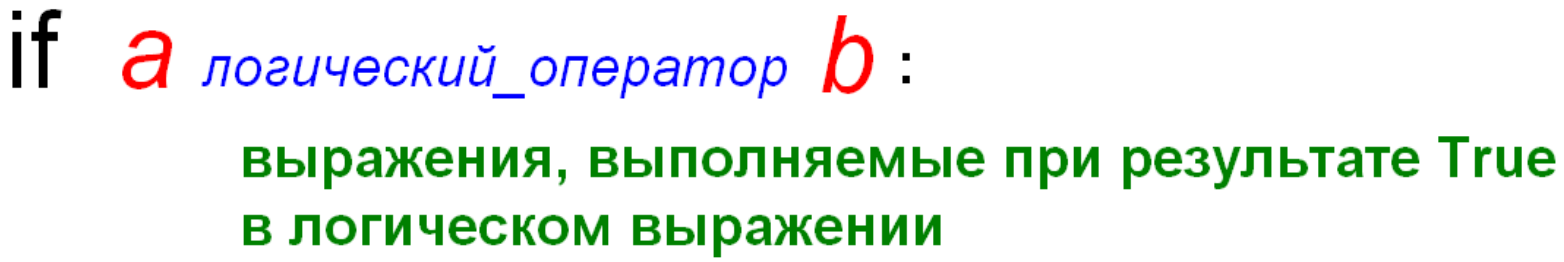
```
>>> num1 = 34
>>> num2 = 8.5
>>> num1 > 12 and num2 != 12
True
>>> num1 == 34 and num2 >= 8
True
>>> num1 <= 12 and num2 == 8.5
False
>>> num1 != 34 and num2 != 12
False
>>> num1 != 34 or num2 != 12
True
>>> num1 < 1 or num2 > 9.6
False
>>> str1 = "a"
>>> str2 = "b"
>>> str1 > 0 and str2 < "c"
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    str1 > 0 and str2 < "c"
TypeError: unorderable types: str() > int()
>>> str1 < "c" and str2 < "c"
True
>>> |
```

Условный оператор. Инструкция if. Урок 5

Методическая разработка урока
Элективный курс: Основы программирования на Python
Уровень: Программирование для начинающих

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 предусмотрены и в компьютерной программе. Например, часть кода должна выполняться лишь при определенном значении конкретной переменной. Обычно в языках программирования высокого уровня используется приблизительно такая конструкция:



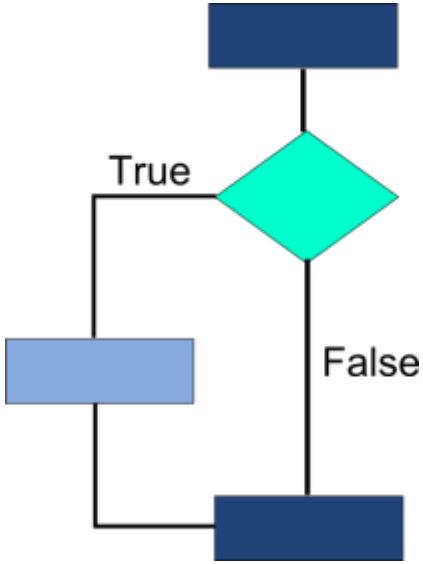
Пример на языке программирования Python (если значение numbig меньше 100, то возвести значение a в степень b, результат присвоить c):

```
1.   if numbig < 100:
2.         c = a**b
```

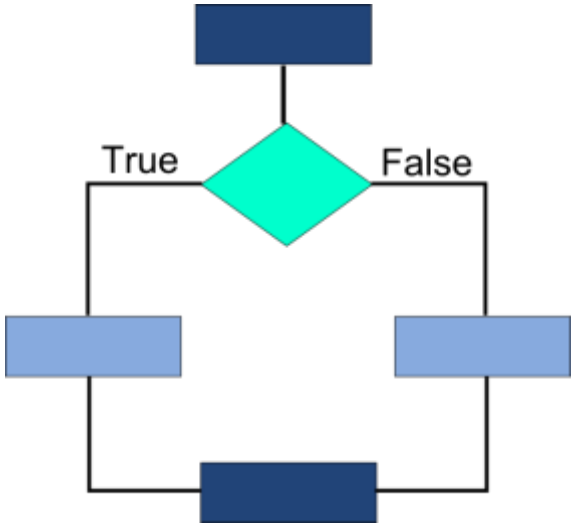
Первая строка «блока if» — это заголовок, в котором проверяется условие выполнения строк кода после двоеточия. В примере выше всего лишь одно выражение, однако в реалии их бывает куда больше.

Про Python говорят, что это язык программирования с достаточно ясным и легко читаемым кодом. Это связано с тем, что в Python 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 само 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 условия: 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:

```
1.  print ("Привет")
2.  tovar1 = 50
3.  tovar2 = 32
4.  if tovar1+ tovar2 > 99 :
5.      print ("Сумма не достаточна")
6.  else:
7.      print ("Чек оплачен")
8.  print ("Пока")
```

Практическая работа

1. Напишите программный код, в котором в случае, если значение некой переменной больше 0, выводилось бы специальное сообщение (используйте команду print). Один раз выполните программу при значении переменной больше 0, второй раз — меньше 0.
2. Усовершенствуйте предыдущий код с помощью альтернативной 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 3 выражений.

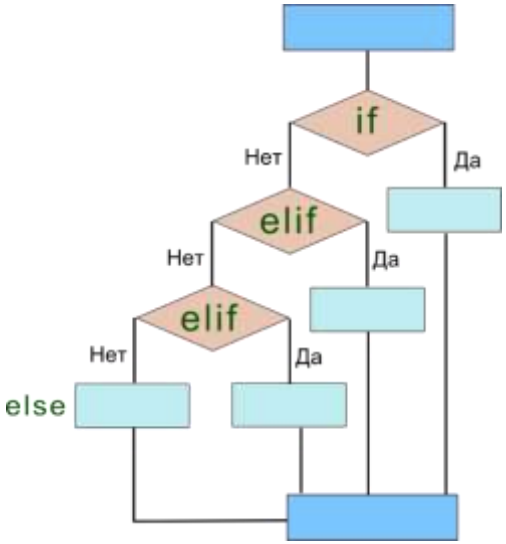
Множественное ветвление. Урок 6

Конспект урока
Элективный курс: Основы программирования на Python
Уровень: Программирование для начинающих

Логика выполняющейся программы может быть сложнее, чем выбор одной из двух ветвей. Например, в зависимости от значения той или иной переменной, может выполняться одна из трех (или более) ветвей программы.

Как организовать такое множественное ветвление? Наверное, можно использовать несколько инструкций **if**: сначала проверяется условие в первой инструкции **if** (если оно положительно, то будет выполняться вложенный в нее блок кода), затем — во второй инструкции **if** и т.д. Однако при таком подходе проверка последующих инструкций будет продолжаться даже тогда, когда, допустим, первое условие было удовлетворено и блок кода при данной ветке выполнен. Проверка последующих условий часто бывает бессмысленной.

Чтобы решить проблему множественного ветвления, в структурных языках программирования предусмотрена специальное расширение инструкции **if**, позволяющее направить поток выполнения программы лишь по одной ветви, минуя остальные. Данная расширенная инструкция, помимо необязательной части **else**, включает части **elif** (сокращение от "else if" - "еще если") и выглядит примерно так, как показано на блок-схеме. Частей **elif** может быть сколь угодно много (в пределах разумного :).



В отличии от использования множества инструкций **if**, инструкция **if-elif-else** прекращает проверку условий, как только найдет первое совпадение. Например, если условие при **if** (первая ветка) будет удовлетворено, то после выполнения вложенного блока выражений, программа вернется в основную ветку.

Примеры скриптов с использованием инструкции **if-elif-else** на языке программирования Python:

```
1.   x = -10
2.
3.   if x > 0:
4.        print (1)
5.   elif x < 0:
6.        print (-1)
7.   else:
8.        print (0)
```

```
1.   result = "no result"
2.   num1 = 3
3.
4.   if num1 == 0:
5.        result = 0
6.   elif num1==1:
7.        result = 1
8.   elif num1==2:
9.        result = 2
10.  elif num1==3:
11.       result = 3
12.  elif num1==4:
13.       result = 4
14.  elif num1==5:
15.       result = 5
16.  else:
17.       print ("Error")
18.
19.  print (result)
```

*Подумайте в какой момент прекратиться выполнение инструкции **if-elif-else** в примерах выше. При каком значении переменной могла «сработать» ветка **else**?*

Практическая работа

А. Напишите программу по следующему описанию:

1. двум переменным присваиваются числовые значения;
2. если значение первой переменной больше второй, то найти разницу значений переменных (вычесть из первой вторую), результат связать с третьей переменной;
3. если первая переменная имеет меньшее значение, чем вторая, то третью переменную связать с результатом суммы значений двух первых переменных;
4. во всех остальных случаях, присвоить третьей переменной значение первой переменной;
5. вывести значение третьей переменной на экран.

Б. Придумайте программу, в которой бы использовалась инструкция **if-elif-else**. Количество ветвей должно быть как минимум 4.

Цикл While. Урок 7

Методическая разработка урока
Элективный курс: Основы программирования на Python
Уровень: Программирование для начинающих

Циклы — это инструкции, выполняющие одну и туже последовательность действий, пока действует определенное 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 некая 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, то тогда 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) является инструкция **while**. Слово "while" с английского языка переводится как "пока" (вроде как "пока действует что-то, делать то-то"). Конструкцию инструкции **while** на языке Python можно описать следующей схемой:

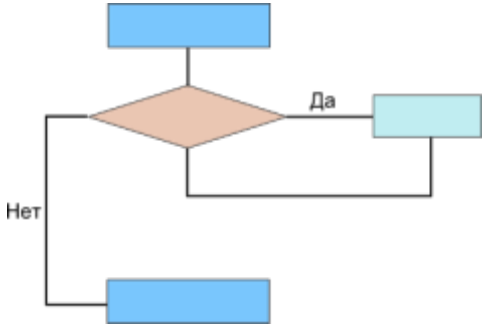
while *a* *логический_оператор* *b* :

выражения, выполняемые при результате True в логическом выражении

изменение *a*

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 организовать момент выхода из цикла — ложность выражения в заголовке. Так, например, изменяя значение какой-нибудь переменной в теле цикла, можно довести логическое выражение до ложности.



Эту изменяемую переменную, доводящую условие до ложности, обычно называют счетчиком. Как и всякой переменной ей можно давать произвольные имена, однако очень часто используют букву **i**. Простейший цикл на языке программирования Python может выглядеть так:

```
1.   str1 = "+"
2.   i = 0
3.   while i < 10:
4.        print (str1)
5.        i = i + 1
```

Примечание: в Python 3.0 аргумент команды **print** заключается в скобки. Пишется так: **print (str1)**

В последней строчке данного скрипта происходит увеличение значения переменной i на единицу, поэтому с каждым оборотом цикла она на нее и увеличивается. Когда будет достигнуто значение 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 будет 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:

```
1.   fib1 = 0
2.   fib2 = 1
3.   print (fib1)
4.   print (fib2)
5.   n = 10
6.   i = 0
7.   while i < n:
8.        fib_sum = fib1 + fib2
9.        print (fib_sum)
10.       fib1 = fib2
11.       fib2 = fib_sum
12.       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.

Подробности. Вводятся две переменные (fib1 и fib2), которым присваиваются начальные значения. Вводится переменная (n) и счетчик (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 обновляются значения fib1 и fib2 (для следующего витка цикла), а также увеличивается значение счетчика.

Практическая работа

1. Создайте 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 скрипт на языке программирования Python, в котором бы использовался цикл **while**.

Ввод данных с клавиатуры. Урок 8

Конспект урока по Python
Элективный курс: Основы программирования на Python
Уровень: Программирование для начинающих

Обычно компьютерная программа не является замкнутой сама на себе. Она обменивается данными с «внешней» для нее средой: может выводить и получать данные. С выводом данных мы уже косвенно знакомы, т.к за вывод данных в языке программирования Python отвечает функция **print ()**, которая выводит их на экран.

Ввод данных с клавиатуры в программу на языке программирования Python (начиная с версии 3.0) осуществляется с помощью функции **input ()**. Когда выполняется выражение, где присутствует данная функция, то поток выполнения программы останавливается в ожидании данных, которые пользователь должен ввести. После ввода данных и нажатия Enter, функция **input ()** завершается.

```
>>> input()
6784
'6784'
>>> input()
1 hi hello
'1 hi hello'
>>>
```

Когда выполняющаяся программа предлагает пользователю что-либо ввести, то пользователь может и не знать, что от него хотят. Надо как-то сообщить, ввод каких данных ожидает программа. С этой целью функции **input ()** можно передать параметр-приглашение (в кавычках); при выполнении функции сообщение будет появляться на экране вывода и информировать человека о запрашиваемых данных.

```
>>> input("Введите номер карты: ")
Введите номер карты: 7562023-131
'7562023-131'
>>> input("Input your name: ")
Input your name: Sasha
'Sasha'
>>>
```

Из примеров видно, что данные возвращаются в виде строки, даже если было введено число. (В более ранних версиях Python были две встроенные функции, позволяющие получать данные с клавиатуры: **raw_input ()**, возвращающая в программу строку и **input ()** - возвращающая число.) В версии 3.0, если требуется получить число, то результат выполнения функции **input()** можно изменить с помощью функций **int()** и **float()**.

```
>>> input ("Введите число: ")
Введите число: 10
'10'
>>> int (input ("Введите число: "))
Введите число: 10
10
>>> float (input ("Введите число: "))
Введите число: 10
10.0
>>> |
```

Результат, возвращаемый функцией **input()**, может быть присвоен переменной для дальнейшего использования.

```
>>> yourName = input ("What is your name? ")
What is your name? Masha
>>> yourName
'Masha'
>>>
```

Практическая работа

А. Создайте скрипт (файл data.py), который бы запрашивал у пользователя

1. его имя: "What is your name?"
2. возраст: "How old are you?"
3. место жительства: "Where are you live?"

, а затем выводил три строки

1. "This is **введенное_имя**"
2. "It is **введенный_возраст**"
3. "He live in **введенное_место_жительства**"

Б. Напишите программу (файл example.py), которая предлагала бы пользователю решить пример 4*100-54. Если пользователь пишет правильный ответ, то получает поздравления, иначе – сообщение об ошибке.

Подсказка: следует использовать конструкцию if ... else

В. Перепишите предыдущую программу так, чтобы пользователю предлагалось решать пример до тех пор, пока он не напишет правильный ответ.
Подсказка: следует использовать цикл while

Примерные ответы к практической работе

1.

```
nic = input ("What is your name? ")
old = input ("How old are you? ")
place = input ("Where are you live? ")

print ("This is " + nic)
print ("It is " + old)
print ("He live in " + place)
```

2.

```
examp = input ("Сколько будет 4*100-54? ")
if examp == "346" :
    print ("Правильно!")
else:
    print ("Ошибка!")
```

3.

```
examp = input ("Сколько будет 4*100-54? ")
while examp != "346":
    print ("Ошибка!")
    examp = input ("Сколько будет 4*100-54? ")
else:
    print ("Правильно!")
```

Последовательности: строки. Урок 9

Конспект урока по Python

Элективный курс: Основы программирования на Python

Уровень: Программирование для начинающих

Строки уже были упомянуты в уроке о типах данных; здесь рассмотрим их более подробно.

Строка — это сложный тип данных, представляющий собой последовательность, т.е. упорядоченный набор других объектов (букв или иных символов).

Строки на языке программирования Python могут заключаться как в одиночные, так и двойные кавычки. Однако, начало и конец строки должны обрамляться одинаковым типом кавычек.

Существует специальная функция **len ()**, позволяющая измерить длину строки. Результатом выполнения данной функции является число, показывающее количество символов в строке. Также для строк существуют операции конкатенации (+) и дублирования (*).

```
>>> len ("object no big")
13
>>> "!!!" + "stroka" + "!!!"
'!!!stroka!!!'
>>> "#" * 20
'####################'
>>>
```

Поскольку в последовательностях важен порядок символов, т.е у каждого есть свое положение, то можно обращаться к конкретному символу в строке и извлекать его с помощью оператора индексирования.

```
>>> "afbhurjka; kdfj"[1]
'f'
>>> "afbhurjka; kdfj"[6]
'j'
>>>
```

В примере, выражение ”afbhurjka; kdfj”[1] привело к извлечению 2 символа. Дело в том, что индексация начинается не с единицы, а с нуля. Поэтому, когда требуется извлечь первый символ, то оператор индексирования должен выглядеть так: [0]. Также позволительно извлекать символы, начиная отсчет с конца. Тогда отсчет идет с -1 (последний символ).

```
>>> "где то так"[0]
'г'
>>> "где то так"[-1]
'к'
>>> "где то так"[-3]
'т'
>>>
```

Очевидно, что удобнее работать не с самими строками, а с переменными, которые на них ссылаются. Результат выполнения выражения индексирования можно присвоить другой переменной.

```
>>> a = "very big string"
>>> a[12]
'i'
>>> b = a[3]
>>> b
'y'
>>>
```

Можно извлекать из строки не один символ, а несколько, т.е. получать срез (подстроку). Оператор извлечения среза из строки выглядит так: [X:Y]. X – это индекс начала среза, а Y – его окончания; причем символ с номером Y в срез уже не входит. Если отсутствует первый индекс, то срез берется от начала до второго индекса; при отсутствии второго индекса, срез берется от первого индекса до конца строки.

```
>>> a = "very big string"
>>> a[0:4]
'very'
>>> a[5:8]
'big'
>>> a[9:]
'string'
>>>
```

Кроме того, можно извлекать символы не подряд, а через определенное количество символов. В таком случае оператор индексирования выглядит так: [X:Y:Z]; Z – это шаг, через который осуществляется выбор элементов.

```
>>> stroka = "Full Ball Fill Pack Ring"
>>> stroka [::5]
'FBFPR'
>>> stroka [0:10:2]
'Fl al'
>>>
```

Практическая работа

1. Свяжите переменную с любой строкой, состоящей не менее чем из 8 символов. Извлеките из строки первый символ, затем последний, третий с начала и третий с конца. Измерьте длину вашей строки.
2. Создайте строку длиной 10-15 символов (свяжите с переменной) и извлеките из нее следующие срезы:
   * Первые восемь символов;
   * Четыре символа из центра строки;
   * Символы с индексами кратными трем.

Последовательности: списки. Урок 10

Конспект урока по Python
Элективный курс: Основы программирования на Python
Уровень: Программирование для начинающих

Списки в языке программирования Python, как и строки, являются упорядоченными последовательностями. Однако, в отличии от строк, списки состоят не из символов, а из различных объектов (значений, данных), и заключаются не в кавычки, а в квадратные скобки [ ].

Примеры списков:

```
1.   [23, 656, -20, 67, -45] # список, состоящий из пяти объектов — целых чисел
2.   [4.15, 5.93, 6.45, 9.3, 10.0, 11.6] # список из дробных чисел
3.   ["Katy", "Sergei", "Oleg", "Dasha"] # список из строк
4.   ["Москва", "Титова", 12, 148] # смешанный список
5.   [[0, 0, 0], [0, 0, 1], [0, 1, 0]] # список, состоящий из списков
```

Списки могут состоять из различных объектов: чисел, строк и даже других списков. В последнем случае, списки называют вложенными.

Как и строки, списки можно соединять и повторять:

```
>>> [23, 656, -20, 67, -45] + [45, -3, 247]
[23, 656, -20, 67, -45, 45, -3, 247]
>>> [[0, 0], [0, 1], [1, 0]] * 2
[[0, 0], [0, 1], [1, 0], [0, 0], [0, 1], [1, 0]]
>>>
```

По аналогии с символами строк, можно получать доступ к объектам списка по их индексам, а также извлекать срезы:

```
>>> li = [1, 2, 3, 4, 5, 6, 7, 0]
>>> li[0]
1
>>> li[6]
7
>>> li[-1]
0
>>> li[0:3]
[1, 2, 3]
>>> li[3:7]
[4, 5, 6, 7]
>>>
```

В отличии от строк, **списки — это изменяемые последовательности**. Другими словами, изменить какой-нибудь символ в строке нельзя, а в списке можно изменять как отдельный элемент, так и целую группу (срез):

```
>>> stroka = "degurnay abrakadabra"
>>> stroka[2] = "j"
Traceback (most recent call last):
  File "<pyshell#15>", line 1, in <module>
    stroka[2] = "j"
TypeError: 'str' object does not support item assignment
>>> spisok = ["и", "на", "но", "из", "за"]
>>> spisok[3] = "в"
>>> spisok
['и', 'на', 'но', 'в', 'за']
>>> spisok[4:6] = ["перед", "у"]
>>> spisok
['и', 'на', 'но', 'в', 'перед', 'у']
>>>
```

В примере выше попытка изменить третью букву в строке привела к ошибке; попытка же изменить четвертый элемент списка была успешной (предлог "из" был заменен на "в"). Далее был изменен срез с пятого элемента по шестой.

Задания по программированию

1. Создайте два любых списка и свяжите их с переменными.
2. Извлеките из первого списка второй элемент.
3. Измените во втором списке последний объект. Выведите список на экран.
4. Соедините оба списка в один, присвоив результат новой переменной. Выведите получившийся список на экран.
5. "Снимите" срез из соединенного списка так, чтобы туда попали некоторые части обоих первоначальных списков. Срез свяжите с очередной новой переменной. Выведите значение этой переменной.
6. Добавьте в список-срез два новых элемента и снова выведите его.

Пример выполнения заданий

1
```
>>> spisok1 = [45, 2, 8, 97, 34]
>>> spisok2 = [65, 23, 10]
```

2
```
>>> spisok1 [1]
2
```

3
```
>>> spisok2 [-1] = 12
>>> spisok2
[65, 23, 12]
```

4
```
>>> big_spisok = spisok1 + spisok2
>>> big_spisok
[45, 2, 8, 97, 34, 65, 23, 12]
```

5
```
>>> small_spisok = big_spisok [3:7]
>>> small_spisok
[97, 34, 65, 23]
```

6
```
>>> small_spisok [4:6] = [56, 84]
>>> small_spisok
[97, 34, 65, 23, 56, 84]
```

Структуры данных: словари. Урок 11

Конспект урока по Python
Элективный курс: Основы программирования на Python
Уровень: Программирование для начинающих

Одним из сложных типов данных в языке программирования Python являются словари. **Словарь можно охарактеризовать как изменяемый неупорядоченный набор пар "ключ : значение".**

Чтобы представление о словаре стало более понятным, можно провести аналогию с обычным словарем, например, англо-русским. На каждое английское слово в таком словаре есть русское слово-перевод: cat – кошка, dog – собака, table – стол и т.д. Если англо-русский словарь описывать с помощью Python, то английские слова будут ключами, а русские — их значениями:

```
{'cat':'кошка', 'dog':'собака', 'bird':'птица'}
```

Общий синтаксис словаря на Питоне можно описать такой схемой:

Если набрать словарь в интерпретаторе Python, то после нажатия Enter можно заметить, что последовательность вывода пар "ключ : значение" может не совпадать с тем, как их ввели изначально.

```
>>> {'cat':'кошка', 'dog':'собака', 'bird':'птица', 'mouse':'мышь'}
{'bird': 'птица', 'mouse': 'мышь', 'dog': 'собака', 'cat': 'кошка'}
```

Дело в том, что в словаре абсолютно не важен порядок пар и интерпретатор выводит их в случайном порядке. Тогда как же получить доступ к определенному элементу, если индексация не возможна в принципе? Ответ: доступ к значения возможен по ключам.

```
>>> d = {'cat':'кошка', 'dog':'собака', 'bird':'птица', 'mouse':'мышь'}
>>> d['cat']
'кошка'
>>> d['bird']
'птица'
```

Словари, как и списки, можно изменять: изменять, добавлять и удалять элементы (пары "ключ : значение"). Изначально словарь можно создать пустым (например, **d = { }**) и лишь потом заполнить его элементами. Добавление и изменение имеет одинаковый синтаксис: **словарь [ключ] = значение**. Ключ может быть как старым (тогда происходит изменение значения), так и новым (происходит добавление пары). Удаление элемента словаря осуществляется с помощью инструкции **del**.

```
>>> d = {'cat':'кошка', 'dog':'собака', 'bird':'птица',
'mouse':'мышь'}
>>> d['elephant'] = 'КлоП'
>>> d['fox'] = 'лиса'
>>> d
{'fox': 'лиса', 'dog': 'собака', 'cat': 'кошка', 'elephant':
'КлоП', 'mouse': 'мышь', 'bird': 'птица'}
>>> d['elephant'] = 'слон'
>>> del d['bird']
>>> d
{'fox': 'лиса', 'dog': 'собака', 'cat': 'кошка', 'elephant':
'слон', 'mouse': 'мышь'}
```

Словари — это широко используемый тип данных языка Python. На самом деле возможности манипуляции со словарями намного превосходят описанные в данном уроке. Словари могут быть более сложными (содержать вложенные сложные типы данных, например, те же словари или списки).

Задания по программированию

1. Создайте словарь, связав его с переменной school, и наполните его данными, которые бы отражали количество учащихся в десяти разных классах (например, 1а, 1б, 2б, 6а, 7в и т.д.).
2. Узнайте сколько человек в каком-нибудь классе.
3. Представьте, что в школе произошли ниже следующие изменения. Внесите соответствующие изменения в словарь.
    1. В трех классах изменилось количество учащихся.
    2. В школе появилось два новых класса.
    3. В школе расформировали один из классов.
4. Выведите содержимое словаря на экран.

Пример выполнения задания

```
>>> school = {'5a':10, '5b':12, '6a':9, '6b':10, '6c':12,
'7a':12, '8a':7, '8b':9, '9a':5, '9b':6, '9c':5}
>>> school ['5a']
10
>>> school ['9a'] = 4
>>> school ['9b'] = 10
>>> school ['6a'] = 11
>>> school ['7b'] = 5
>>> school ['8c'] = 6
>>> del school ['9c']
>>> school
{'6a': 11, '6b': 10, '6c': 12, '7b': 5, '7a': 12, '8b': 9,
'8c': 6, '9b': 10, '8a': 7, '9a': 4, '5a': 10, '5b': 12}
>>>
```

Цикл for в языке программирования Python. Урок 12

Конспект урока по Python

Элективный курс: Основы программирования на Python

Уровень: Программирование для начинающих

В седьмом уроке был рассмотрен цикл **while**. Однако это не единственный способ организации повторения группы выражений на языке Python. В программах, написанных на Питоне, широко применяется цикл **for**, который по сути является циклом обхода какого-либо множества упорядоченных элементов (символов строки, объектов списка) и выполнения в своем теле различных операций над этими объектами. Например, если имеется список чисел и необходимо увеличить значение каждого элемента на две единицы, то можно перебрать список с помощью цикла **for**, выполнив над каждым его элементом соответствующее действие.

```
>>> spisok = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> i = 0
>>> for element in spisok:
        spisok[i]= element + 2
        i = i + 1


>>> spisok
[2, 12, 22, 32, 42, 52, 62, 72, 82, 92]
>>>
```

В примере имеется первоначальный список, вводится переменная-счетчик **i**. В заголовке цикла **for** происходит обращение к первому (или очередному) элементу списка. В теле цикла элементу с индексом **i** присваивается сумма значения текущего (обрабатываемого) элемента и двойки. Далее индекс увеличивается на единицу, а поток выполнения программы переходит снова в заголовок цикла **for**, где происходит обращение к следующему элементу списка. Когда все элементы обработаны цикл **for** заканчивает свою работу. Отсутствие очередного элемента является условием завершения работы цикла **for** (для сравнения: в цикле **while** условием завершения служит результат **false** логического выражения в заголовке). Еще один момент: если счетчик не увеличивать на единицу (выражение **i = i + 1**), то не смотря на то, что все элементы списка будут обработаны, результат все время будет присваиваться первому элементу списка (с индексом 0). Можете попробовать.

Перебирать можно и строки.

```
>>> stroka = "привет"
>>> for bukva in stroka:
        print (bukva)


п
р
и
в
е
т
>>> for bukva in stroka:
        print (bukva, end='!?!')


п!?!р!?!и!?!в!?!е!?!т!?!
>>>
```

На самом деле использование цикла **for** в языке программирования Python более разнообразно, чем в приведенных выше примерах. Также следует запомнить, что цикл **for** в Питоне особенный. Он не является аналогом циклов **for** во многих других языках программирования, где цикл **for** – это, так называемый, цикл со счетчиком.

Задания по программированию

Задание. Создайте список, состоящий из четырех строк. Затем, с помощью цикла for, выведите строки поочередно на экран.

Ответ:

```
>>> list1 = ["hi", "oh", 'no', 'yes']
>>> for i in list1:
        print (i)


hi
oh
no
yes
>>>
```

Задание. Измените предыдущую программу так, чтобы в конце каждой буквы строки добавлялось тире.
Подсказка: цикл for может быть вложен в другой цикл.

Ответ:

```
>>> for i in list1:
        for a in i:
            print (a, end='-')


h-i-o-h-n-o-y-e-s-
>>>
```

Задание. Создайте список, содержащий элементы целочисленного типа, затем с помощью цикла перебора измените тип данных элементов на числа с плавающей точкой.
Подсказка: используйте встроенную функцию float().

Ответ:

```
>>> list2 = [56, 78, 45, 23]
>>> i = 0
>>> for a in list2:
        list2[i] = float(a)
        i = i+1


>>> list2
[56.0, 78.0, 45.0, 23.0]
```

Функции в языке программирования Python. Урок 13

Конспект урока

Курс: Основы программирования на Python

Уровень: Программирование для начинающих

1. Нужны ли функции?

Функции в программировании можно представить как изолированный блок кода, обращение к которому в течение выполнения программы может быть многократным. Зачем нужны такие блоки инструкций? В первую очередь, чтобы сократить объем исходного кода: рациональней вынести часто повторяющиеся выражения в отдельный блок и, затем, по мере надобности, обращаться к нему.

Представим себе следующую ситуацию. Требуется написать скрипт, который при исполнении должен три раза запрашивать у пользователя разные данные, но выполнять с ними одни и те же действия.

```
a = input ("Введите первое число: ")
b = input ("Введите второе число: ")
a = float (a)
b = float (b)
if a > b:
    print (a - b)
else:
    print (b - a)

c = input ("Введите первое число: ")
d = input ("Введите второе число: ")
c = float (c)
d = float (d)
if c > d:
    print (c - d)
else:
    print (d - c)

e = input ("Введите первое число: ")
f = input ("Введите второе число: ")
e = float (e)
f = float (f)
if e > f:
    print (e - f)
else:
    print (f - e)
```

Данная программа находит модуль разницы двух чисел. Очевидно, что такая запись исходного кода не рациональна: три почти одинаковых блока. На самом деле можно было бы использовать цикл **while** для организации повторения.

```
i = 0
while i < 3:
    a = input ("Введите первое число: ")
    b = input ("Введите второе число: ")
    a = float (a)
    b = float (b)
    if a > b:
        print (a - b)
    else:
        print (b - a)
    i = i + 1
```

Однако, в этом случае есть небольшая проблема. Вводимые пользователем данные всегда связываются с переменными **a** и **b**. При каждом витке цикла старые данные теряются. Но что делать, если ссылку на вводимые пользователем данные (все шесть чисел) надо сохранить для дальнейшего использования этих самых данных в программном коде. В данном случае поможет использование функций.

2. Программирование функций на python

Предыдущий скрипт, но уже с использованием функций на языке программирования Python, может выглядеть так:

```python
def schet ():
    a = input ("Введите первое число: ")
    a = float (a)
    b = input ("Введите второе число: ")
    b = float (b)
    if a > b:
        print (a - b)
    else:
        print (b - a)
    return a, b

c, d = schet ()
e, f = schet ()
z, x = schet ()
```

Разберем представленный программный код. **def** – это инструкция (команда) языка программирования Python, позволяющая создавать функцию. **schet** – это имя функции, которое (так же как и имена переменных) может быть почти любым, но желательно смысловым. Далее идет двоеточие, обозначающее окончание заголовка функции (аналогично с условиями и циклами). После заголовка, с новой строки и с отступом, следуют выражения тела функции. В конце тела функции присутствует инструкция **return** (которой может и не быть), она передает то или иное значение в основную ветку программы. Если бы в функции не было инструкции **return**, то в основную программу ничего бы не возвращалось и переменным **c** и **d** (или **e** и **f**, а также **z** и **x**) ничего не присваивалось.

После функции идет, так называемая, основная ветка программы, в которой переменным попарно присваивается результат выполнения вызываемой функции. На самом деле вызов функции может ничему и не присваиваться.

Выражения тела функции выполняются лишь тогда, когда она вызывается в основной ветке программы. Так, например, если функция присутствует в исходном коде, но нигде не вызывается в нем, то содержащиеся в ней инструкции не будут выполнены ни разу.

3. Практическая работа

1. По аналогии с примером представленным в уроке напишите скрипт, в котором бы в теле функции производились вычисления со значениями трех переменных.
2. Придумайте программу, в которой бы присутствовали две функции, основная ветка программы содержала не менее 10 строк, и каждая функция вызывалась как минимум один раз.

Параметры и аргументы функций. Локальные и глобальные переменные. Урок 14

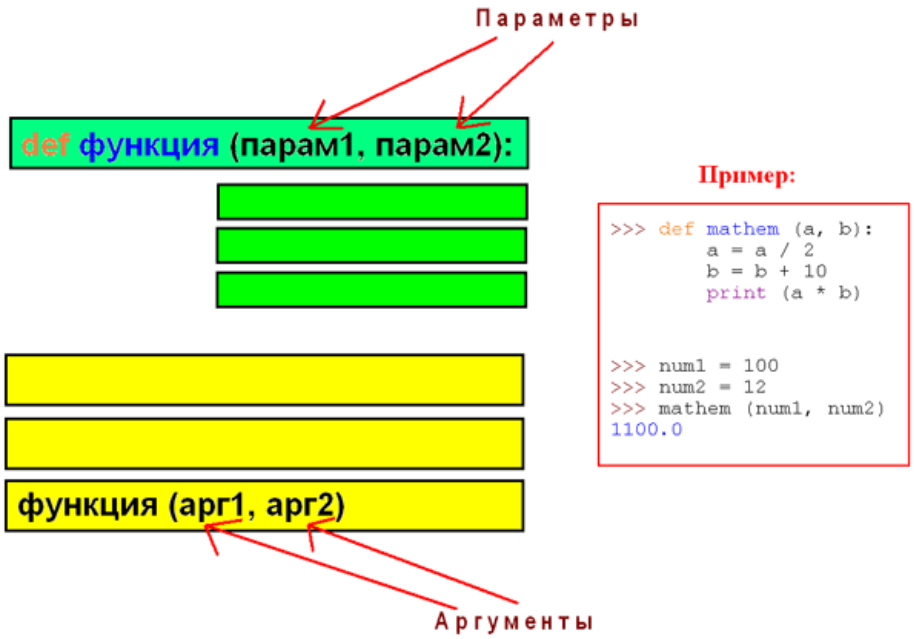
Конспект урока

Элективный курс: Основы программирования на Python

Уровень: Программирование для начинающих

1. параметры и аргументы функций

Часто функция используется для обработки данных, полученных из внешней для нее среды (основной ветки программы). Данные передаются функции при ее вызове в скобках и называются **аргументами**. Однако, чтобы функция могла «взять» передаваемые ей данные, необходимо при ее создании указать **параметры**-переменные (тоже в скобках).



Когда функция вызвана, значения конкретных аргументов «подставляются» вместо параметров. Количество аргументов и параметров должно совпадать (не всегда: можно запрограммировать переменное количество принимаемых аргументов). В качестве аргументов могут выступать как конкретные значения, так и переменные.

2. локальные и глобальные переменные

Если вы напишете в IDLE выражения, приведенные в примере выше, а затем попробуете вывести значения переменных, то увидите следующую картину:

```
>>> num1
100
>>> num2
12
>>> a
Traceback (most recent call last):
  File "<pyshell#17>", line 1, in <module>
    a
NameError: name 'a' is not defined
>>> b
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    b
NameError: name 'b' is not defined
>>>
```

Переменные **num1** и **num2** не изменили своих первоначальных значений. Дело в том, что в функцию передавались копии значений, а не оригиналы. А вот переменных a и b оказывается нет и в помине (надпись «name 'b' is not defined» переводится как «переменная b не определена»). Данные переменные существуют лишь в момент выполнения функции и называются локальными. В противовес им, переменные **num1** и **num2** видны не только во внешней ветке, но и внутри функции:

```
>>> def mathem ():
        print (num2 + num1)


>>> mathem()
112
```

Переменные, определенные за пределами функций, является **глобальными**.

3. задания по программированию

1. Создайте функцию:

```
1.   def func1(num):
2.        n = num * 5
3.        print (n)
```

2. Вызовите предыдущую функцию, передав ей в качестве аргумента значение какой-нибудь глобальной переменной, затем любое число и, наконец, любую строку.

3. Выполните с помощью интерпретатора Python скрипт, предварительно исправив код функции так, чтобы переменная **num** ссылалась на новое число:

```
1.   def func2 (a):
2.        b = 10
3.        if a < 3:
4.             a = a * 10
5.
6.   num = 1
7.   num = func2 (num)
8.   num
```

Задания по программированию на Python. Урок 15

Курс: Основы программирования на Python

Уровень: Программирование для начинающих

Задание 1

Напишите код по следующему словесному алгоритму:

1. Попросить пользователя ввести число от 1 до 9. Полученные данные связать с переменной x.

2. Если пользователь ввел число от 1 до 3, то ...
   * попросить пользователя ввести строку. Полученные данные связать с переменной s;
   * попросить пользователя ввести число повторов строки. Полученные данные связать с переменной n, предварительно преобразовав их в целочисленный тип;
   * выполнить цикл повторения строки n раз;
   * вывести результат работы цикла.
3. Если пользователь ввел число от 4 до 6, то ...
   * попросить пользователя ввести степень, в которую следует возвести число. Полученные данные связать с переменной m;

   * реализовать возведение числа x в степень m;

   * вывести полученный результат.

4. Если пользователь ввел число от 7 до 9, то выполнить увеличения числа n на единицу в цикле 10 раз, при этом на экран вывести все 10 чисел.
5. Во всех остальных случаях выводить надпись «Чайник».

Задание 2

Напишите программу, которая бы выполняла следующие задачи:

1. выводила название программы "Общество на рубеже XX-XXI веков";
2. запрашивала у пользователя его возраст;
3. если пользователь вводит числа от 0 до 7, то программа выводила надпись "Вам в детский сад";
4. от 7 до 18 - "Вам в школу";
5. от 18 до 25 - "Вам в профессиональное учебное заведение";
6. от 25 до 60 - "Вам на работу";
7. от 60 до 120 – "Вам предоставляется выбор";
8. меньше 0 и больше 120 – пятикратный вывод надписи "Ошибка! Это программа для людей!"
9.

В программе желательно присутствие всех «атрибутов» структурного программирования: функции, ветвления и цикла.