Питон
Курс лекций
Лекция третья

© Зайцев Вадим Валерьевич, 2002–2010,
spider.vz@gmail.com

Итак, узнав всё необходимое об обеспечении ЭВМ, о проектировании программ, об эволюции языков программирования и о том, как работать с интерпретатором питона, попробуем перейти к чему-нибудь более конкретному, а именно: разобрать простенькую программу на языке питон.

3 Типы данных и простейшие конструкции питона

3.1 Понятие переменной. Оператор присваивания

Вот типичный пример программы на питоне:

```
a = 1
b = 2
print "a + b = a+b
```

На первый взгляд программа проста и, хотя она на самом деле, возможно, ещё более проста, чем кажется, здесь есть о чём поговорить.

Первая строчка. У переменной по имени `a` появляется значение, равное единице.

Определение. Переменная есть имя, присвоенное одной или нескольким ячейкам памяти, содержащим некое значение.

У некоторых сразу могут возникнуть два вопроса:

1. Может ли одно и то же имя указывать на разные ячейки одновременно?

2. Может ли одна и та же ячейка памяти иметь одновременно несколько имён?

Для питона ответы соответственно: *нет* и *да*. Имя связано только с одним набором ячеек, а вот один и тот же набор ячеек может иметь сразу

несколько имён. Позже станет понятно, что это справедливо только для сложных переменных: последовательностей и объектов, а строки и числа имеют только по одному имени.

Множество допустимых значений переменной — это её тип. Так, например, 25 — это целое число, π — вещественное, а «мехмат» — это строка. В питоне переменные могут в процессе жизни легко менять свой тип, поэтому он называется нетипизированнным языком. Конечно же, эта бестиповость не означает, что в питоне нет данных различных типов. Вовсе нет! Просто программисту не обязательно об этом задумываться. Таким образом, данные имеют тип, а переменные — нет.

Теперь поговорим о знаке равенства, стоящем между именем переменной и её будущим значением. Так обозначается оператор присваивания, один из важнейших операторов в большинстве языков. В одной книге по языкам программирования автор утверждал, что <есть только один оператор, который фактически что-то делает, — оператор присваивания. Все другие операторы... существуют только для того, чтобы управлять последовательностью выполнения операторов присваивания>. Мнение это спорное, но правильное.

Определения оператора присваивания мы давать не будем, а ограничимся описанием того, что происходит при его выполнении:

1. Вычисление значения выражения в правой части оператора (справа от знака равенства до конца строки).
2. Вычисление выражения в левой части оператора (выражение это должно однозначно определить адрес ячеек памяти).
3. Копирование значения из шага 1 в ячейки из шага 2.

На практике чаще всего слева стоит имя переменной, хотя имён может быть и несколько. Например, первые две строки нашей программы можно было объединить в одну:

```
a,b=1,2
```

Строгую теоретическую базу под возможность использования оператора присваивания таким образом мы подведём позже.

Кроме того, оператор присваивания позволяет присваивать *одно и то же* значение сразу нескольким переменным:

```
c=d=e=0
```

Третья строка программы содержит оператор вывода на экран.

3.2 Вывод данных

В данном случае будут напечатаны две вещи: строка `a + b =`, не претерпевшая изменений, и вычисленное значение выражения a+b. После этого курсор будет переведён на новую строку. Таким образом, на экране мы увидим:

```
a + b = 3
```

Пробел между двумя выведенными объектами оператор вывода вставляет автоматически, а на новую строку переходит только после вывода всех значений. Если это необходимо сделать в другом месте, программист должен либо использовать несколько операторов вывода, либо явно указать переход в виде `"\n` — в этом месте и будет разорвана строка. Так,

```
print "a +\nb = 5
```

выдаст:

```
a␣+
b␣=␣5
```

Обратный слэш вместе с последующей буквой называется управляющей последовательностью. Некоторые буквы не порождают такой последовательности и выводятся как есть, но во избежание сюрпризов стоит каждый обратный слэш набирать как `"\\` — эта простейшая управляющая последовательность используется для обозначение слэша как такового. Использование обратного слэша для ввода обычных символов называется маскировкой. Подробнее об этом мы поговорим, когда дойдём до символьного типа данных.

Если переход на новую строку не нужен даже в конце оператора `print`, следует после списка всех значений поставить дополнительную запятую:

```
print "one
print "two
print "three"
one␣two␣three
```

Иногда бывает необходимым сделать так, чтобы вывод был более красивым: сделать выравнивание по какой-то стороне текста, добавить пробелов и т.д. В питоне это можно делать, не выходя за пределы оператора `print`, причём нужно задать только поле, которое должно занимать значение переменной, а нужное количество пробелов оператор вывода вставит автоматически. Делается это так:

```
print '%-5d = %5d' % (25, 34)
```

Первым параметром идёт заключённая в кавычки строка, содержимое которой и определяет выводимый формат. Затем следуют все выводимые переменные или значения, перечисленные в скобках. Все символы форматирующей строки, за исключением символа процента (и следующих за ним числа и буквы), будут выведены как обычно. Сам процент называется форматирующим оператором. На место каждого из форматирующих операторов будет вставлено соответствующее значение следующим образом: число определяет количество экранных знакомест (для текстового режима) или пробелов (для графического), отведённых для значения. Если длина выводимого значения больше этого числа, пробелы не добавляются. Если же меньше, дописываются пробелы справа (если число отрицательное) или слева (если положительное) так, чтобы длина выведенной строки была равна заданному числу. Буква после числа означает формат вывода и может иметь значение `d` для целых чисел, `f` для вещественных или `s` для

строк (или вывода чисел как строк). Таким образом, наше выражение будет напечатано так:

```
25␣␣␣␣=␣␣␣␣34
```

Каждое число отделяет от знака равенства не три, а четыре пробела — ещё один пробел мы сами вписали в форматирующую строку.

Для вещественных чисел имеется возможность задать нужное количество символов после запятой, округление будет произведено автоматически:

```
print 'Число пи примерно равно %5.3f' % 3.1415926535897931
Число␣пи␣примерно␣равно␣3.142
```

3.3 Ввод данных

Логично было бы предположить наряду с оператором вывода существование оператора ввода. И он действительно есть и называется `input`. Используется он следующим простейшим образом:

```
x=input()
```

При этом у пользователя спрашивается питоновское выражение, значение которого и заносится в переменную `x`. Это именно значение выражения, поэтому, например, если ввести `25+59`, в `x` будет передано `84`, а если попытаться ввести строку, питон выдаст ошибку — строки надо заключать в кавычки явным образом. Естественно, в этом выражении, как и в любом другом, можно использовать имена уже определённых переменных, на место которых будут подставлены их текущие значения.

Второй способ использования оператора ввода такой:

```
N=input('N=')
```

Строка, передаваемая оператору `input`, называется приглашением, она выдаётся на экран перед запросом выражения пользователя (который происходит совершенно точно так же).