Введение в Python и Eric

Иван Хахаев, 2009

Ветвления и математические функции

В решениях задач по алгоритмизации одним из важнейших элементов является так называемое «ветвление», которое хорошо описывается в народных сказках - «Направо пойдёшь — голову потеряешь, прямо пойдёшь — коня потеряешь...», а проще говоря, ситуация «если ..., то ..., иначе ...». В языках программирования такой ситуации соответствует конструкция `if <условие> then <какие-то действия> else <другие действия> end if`. Отличие разных языков проявляется в особенностях записи условий и расположении остальных частей конструкции (её кратко называют «оператор `IF`»), а также в способе завершения конструкции (надо или нет писать `end if` и как именно).

Рассмотрим (прямом смысле) текст программы (рис. 1, `script-04.py`), записанный в IDE Eric.

```
1   # -*- coding: utf-8 -*-
2   # Задача: Дано целое число A,
3   # Определить чётное число или нечётное.
4   a=input("Ваше число? ")
5   if a%2<>0:
6       s="Нечётное"
7   else:
8       s="Чётное"
9   print s
10
```

Рисунок 1. Текст программы с ветвлением

В первой строке программы вводится число (естественно, определять чётное число или нечётное имеет смысл только для целых чисел), во второй строке начинается оператор `IF`. Условием является сравнение остатка от деления на 2 (операция вычисления остатка от деления обозначается символом «`%`»), и если этот остаток не равен нулю («не равно» обозначается как «`<>`», однако можно использовать и вариант «`!=`»), строковая переменная `s` получает значение «Нечётное», иначе эта же переменная получает значение «Чётное», поскольку чётные числа по определению делятся на 2 без остатка. Последним действием выводится получившееся значение строки `s`.

Как мы видели в задаче о вычислении налога, деление целого числа на целое с помощью оператора «/» даёт тоже целое число. Однако в Python существует специальный оператор целочисленного деления - «//». Если нужно специально получать результат целочисленного деления, лучше пользоваться вариантом «//». Все остальные арифметические операции записываются обычным образом, а вот оператор возведения в степень записывается как «**» (т.е. a^x будет записано как `a**x`)

На рис. 1 видно, что начало каждой «ветви» программы обозначается символом «:». Условие в операторе `IF` записывается без скобок. IDE Eric обеспечивает автоматические отступы в строках внутри составных операторов (одним из которых является оператор `IF`). Как таковое окончание оператора IF отсутствует. Python считает, что следующий оператор начинается в строке без отступа.

Таким образом, в Python отступы играют важную роль, а IDE Eric их делает автоматически там где надо (размер отступа устанавливается в настройках, см. выше). Кроме того, IDE Eric указывает на начало «ветвей» программы. рисуя чёрточки в сером поле слева.

Нужно заметить, что оператор `IF` в Python может иметь и более сложный вид. Например, требуется определить, является ли число положительным, отрицательным или нулём. Тогда текст программы будет выглядеть примерно так (`script-04-a.py`):

```
#  -*- coding: utf-8 -*-
# Определяем, является число
# положительным, отрицательным или нулём
a=input("Ваше число? ")
if a<0:
    s="Отрицательное"
elif a==0:
    s="Ноль!"
else:
   s="Положительное"
print s
```

Ключевое слово `elif` означает `else if`. Такая конструкция позволяет осуществить три варианта выбора. Каких -либо специальных средств, позволяющих осуществить выбор большего количества вариантов, в Python нет.

Пусть теперь нужно определить, попадает ли число в «первую десятку», т.е. интервал от 0 до 10. Будем считать, что 0 входит в этот интервал. Текст программы приведён ниже (`script-04-b.py`).

```
#  -*- coding: utf-8 -*-
# Задача: Дано целое число A,
# определить, попадает ли оно в первые 10
# (считаем с нуля!)
a=input("Ваше число? ")
if 0<=a<10:
    s="Попадает в десятку!"
else:
    s="Не попадает..."
print s
```

Здесь мы видим «двойной» оператор сравнения. Это ещё одна специфическая и очень полезная особенность Python.

Для вычисления математических функций (синус, косинус, квадратный корень, логарифмы и пр.) в Python нужно подключать специальную библиотеку `math` (а, например, для рисования нужно подключать графическую библиотеку). Такая организация позволяет не «загружать» систему ненужными в какой-то задаче возможностями и увеличивать скорость выполнения программ за счёт

экономии ресурсов.

Рассмотрим следующий пример (`script-05.py`).

```
#  -*- coding: utf-8 -*-
# Задача: Определить, можно ли из бревна, имеющего диаметр
# поперечного сечения D,
# выпилить квадратный брус со стороной A?
#
import math
d=input("Диаметр бревна: ")
a=input("Ширина бруса: ")
if a*math.sqrt(2)<=d:
    s="Можно!"
else:
    s="Нельзя..."
print s
```

Чтобы получить возможность использования функции вычисления квадратного корня (`sqrt()`) мы подключили библиотеку математических функций (`import math`). Конкретная функция из библиотеки в результате имеет «составное» имя — `имя_библиотеки.имя_функции` (в нашем примере — `math.sqrt()`).

Однако можно упростить использование функций из библиотеки, если изменить команду подключения библиотеки:

```
from math import *
d=input("Диаметр бревна: ")
a=input("Ширина бруса: ")
if a*sqrt(2)<=d:
    s="Можно!"
else:
    s="Нельзя..."
print s
```

В таком варианте вместо подключения целой библиотеки мы подключаем только функции из этой библиотеки (в данном случае мы подключили все функции инструкцией `from math import *`). Теперь функции можно использовать без указания имени библиотеки, однако нужно следить за тем, чтобы не было случайных совпадений «кратких» имён функций.

Полный список имеющихся математических функций с их краткими описаниями можно получить, набрав в окне выполнения (панель «Оболочка») команду

```
help('math')
```

Помимо стандартного набора тригонометрических функций и логарифмов интересно отметить такие функции как `degrees()` и `radians()`, устанавливающие единицы измерения углов в градусах и радианах соответственно, функцию `hypot(x,y)`, вычисляющую квадратный корень из суммы квадратов аргументов, а также наличие предопределённых констант `pi` и `e`.

Например, чтобы вычислить синус 30 градусов, нужно выполнить следующее:

```
from math import *
sin(radians(30))
```

Если нужен только список имён функций модуля (библиотеки) `math`, то его теперь (после подключения библиотеки) можно получить, набрав в окне выполнения

```
dir(math)
```

Вообще говоря, подключение всех функций библиотеки считается в Python плохой практикой, и более правильно было бы текст рассматриваемой программы написать так:

```
from math import sqrt
d=input("Диаметр бревна: ")
a=input("Ширина бруса: ")
if a*sqrt(2)<=d:
    s="Можно!"
else:
    s="Нельзя..."
print s
```