## Глава 2. Переменные, выражения и инструкции (Variables, expressions and statements)

### 2.1. Значения и типы

Значение (value) – одно из базовых понятий, с которым работают программы, наподобие букв или чисел. Значения могут выглядеть по-разному: 1, 2 или ‘Hello, World!’.

Эти значения принадлежат различным типам (types): 2 – это целочисленное значение (integer), ‘Hello, World!’ – строковое (string), т.к. содержит «строку» букв. Вы (и интерпретатор) можете распознать строки, т.к. они заключаются в кавычки.

Оператор *print* также работает с целыми числами:

```
>>> print 4

4
```

Если вы не уверены в типе значения, интерпретатор может предоставить вам подсказку:

```
>>> type('Hello,

World!') <type 'str'>

>>> type(17)

<type 'int'>
```

Не удивительно, что строки принадлежат к типу *str*, а целые значения - к типу *int*. Менее очевидно то, что числа с дробной частью принадлежат к типу *float*, такой формат называется с плавающей точкой (floating-point).

```
>>> type(3.2)

<type 'float'>
```

Как насчет ‘17’ и ‘3.2’? Они похожи на числа, но кавычки указывают на то, что это строки:

```
>>> type('17')
```

```
<type 'str'>

>>> type('3.2'

) <type 'str'>
```

Они являются строками.

Если вы встречаете большое число, то может возникнуть соблазн использовать разделители между группами цифр:

```
>>> print 1,000,000

1 0 0
```

Python интерпретировал 1,000,000 как последовательность целых чисел, разделенных запятой, и вывел на экран отдельные числа.

Мы столкнулись с примером семантической ошибки: код выполняется без сообщений об ошибке, но делает не то, что мы задумывали.

### 2.2. Переменные

Одна из наиболее мощных особенностей языка программирования – это манипуляция переменными (variables). Переменная – это имя, которое ссылается на значение.

Оператор присваивания (assignment statement) создает новые переменные и присваивает им значения:

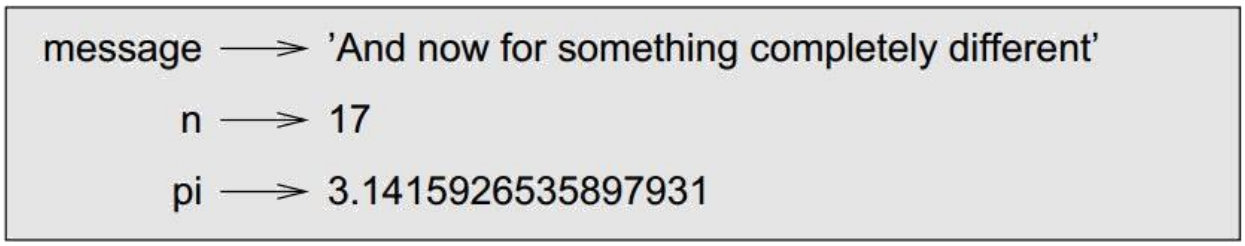
```
>>> message = 'And now for something completely different'

>>> n = 17

>>> pi = 3.1415926535897931
```

Этот пример демонстрирует три оператора присваивания. Сначала строка присваивается переменной с именем *message*; затем значение 17 – переменной *n*; третье присваивание – приближенное значение числа π присваивается переменной *pi*.

Общий способ представления переменных на бумаге заключается в написании имени со стрелкой, направленной на значение этой переменной. Подобный вид схемы называется диаграммой состояний (state diagram), потому что показывает состояние каждой из переменных. Следующая диаграмма показывает результат из предыдущего примера:




Для отображения на экране значения переменной можно воспользоваться инструкцией *print*:

```
>>> print n
17
>>> print pi
3.14159265359
```

Типы переменных – это типы значений, на которые они ссылаются.

```
>>> type(message)
<type 'str'>
>>> type(n)
<type 'int'>
>>> type(pi)
<type 'float'>
```

### 2.3. Имена переменных и ключевые слова

Программисты обычно используют для своих переменных понятные имена, которые раскрывают их назначение.

Имена переменных могут иметь произвольную длину. Они могут содержать буквы и цифры, но они должны начинаться с буквы. Можно использовать буквы в верхнем регистре, но лучше имена переменных начинать со строчной буквы (вы позже узнаете почему).

Символ подчеркивания ( **_** ) может встречаться в имени переменной. Он часто используется в именах с несколькими словами, такими как *my_name* или *airspeed_of_unladen_swallow*.

Если вы зададите для переменной некорректное имя, то получите синтаксическую ошибку:

```
>>> 76trombones = 'big parade'
```

```
SyntaxError: invalid syntax

>>> more@ = 1000000

SyntaxError: invalid syntax

>>> class = 'Advanced Theoretical Zymurgy'

SyntaxError: invalid syntax
```

*76trombones* – неправильное имя, т.к. оно не начинается с буквы. *more@* - неправильное имя, т.к. содержит некорректный символ @. Что не так с *class*?

Слово *class* является зарезервированным (или ключевым) словом (keywords), оно используется Python для распознавания структуры программы и не может быть использовано в качестве имени переменной.

В Python зарезервировано 31 слово:

```
and del from not while as elif global or with assert else if
pass yield break except import print class exec in raise
continue finally is return def for lambda try
```

В новой версии Python 3.0, *exec* больше не является зарезервированным словом, а *nonlocal* – является.

### 2.4. Операторы

Инструкции (statement) – это единица измерения кода, которую интерпретатор Python может выполнять. Мы уже встречались с двумя видами инструкций: *print* (вывод на экран) и *присваивание*.

Когда вы вводите инструкцию в интерактивном режиме, интерпретатор выполняет ее и отображает результат.

Скрипт обычно содержит последовательность инструкций. Если инструкций несколько, то результат появляется одновременно с выполнением инструкции.

Например, выполнение скрипта:

```
print 1

x = 2

print x
```

приводит к результату

```
1
```

```
2
```

Инструкция присваивания ничего не выводит на экран.

### 2.5. Операторы и операнды

*Операторы* (operators) – специальные символы, которые представляют вычисления, наподобие сложения и умножения. Значения, к которым применяется оператор, называются *операндами* (operands).

Операторы +, -, *, / и ** выполняют соответственно сложение, вычитание, умножение и возведение в степень:

```
20+32

hour-1

hour*60+minute

minute/60

5**2

(5+9)*(15-7)
```

Операция деления может привести к неожиданным результатам:

```
>>> minute = 59

>>> minute/60

0
```

Значение переменной *minute* равно 59, операция деления 59 на 60 приведет к результату 0.98333, а не к нулю! Причиной подобного результата в Python является округление (floor division).

Подробнее об исторических предпосылках округления можно узнать в блоге автора Python:

http://python-history.blogspot.com/2010/08/why-pythons-integer-division-floors.html

Если оба операнда целочисленные, то результат тоже будет целочисленным. В нашем примере дробная часть отбрасывается, и в результате получаем нуль.

Если какой-либо из операндов является числом с плавающей точкой, то результат тоже будет типа *float*:

```
>>> minute/60.0
0.9833333333333333
```

В новой версии Python 3.0, результат деления целочисленных значений будет числом с плавающей точкой.

## 2.6. Выражения

*Выражение* (expression) – это комбинация значений, переменных и операторов. Значение (или переменная) само по себе является выражением, поэтому все следующие выражения являются корректными (предполагается, что переменной *x* было присвоено значение):

```
17
x
x + 17
```

Если выражения вводятся в интерактивном режиме, то интерпретатор вычисляет (evaluates) их и отображает результат:

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

В скрипте подобное выполняться не будет. Вывод результата без использования *print*, работает только в интерактивном режиме Python!

## 2.7. Порядок операций

Если в выражении встречается больше, чем один оператор, то порядок вычислений зависит от правил старшинства (rules of precedence). Для математических операций, Python следует математическим соглашениям. Аббревиатура **PEMDAS** является простым способом для запоминания правил:

- Скобки (**P**arentheses) имеют наивысший приоритет и могут использоваться для принудительного определения порядка вычислений в выражении. Таким образом, результат выражения 2*(3-1) будет равен 4, (1+1)**(5-2) будет равен 8. Вы также можете использовать скобки для упрощения чтения выражений, например, (minute*100) / 60, если это не повлияет на результат.
- Возведение в степень (**E**xponentiation) имеет наибольший приоритет, так 2**1+1 равно 3, а не 4, и 3*1**3 равно 3, а не 27.

- Умножение и деление (**M**ultiplication and **D**ivision) имеют одинаковый приоритет, который выше сложение и вычитания (**A**ddition and **S**ubtraction), которые также имеют одинаковый приоритет. Таким образом, 2*3-1 равно 5, а не 4, и 6+4/2 равно 8, а не 5.
- Операторы с одинаковым приоритетом вычисляются слева направо. Таким образом, 5-3-1 равно 1, а не 3.

Если вы сомневаетесь, то поставьте скобки.

### 2.8. Модульные операторы

Модульные операторы работают с целочисленными значениями и возвращают остаток от деления (yields the remainder) двух операндов. В Python модульный оператор представлен символом (%). Синтаксис у него следующий:

```
>>> quotient = 7 / 3

>>> print quotient

2

>>> remainder = 7 % 3

>>> print remainder

1
```

Модульный оператор может быть очень полезен, если вы хотите проверить делится ли *x* на *y* без остатка – если *x* % *y* равно 0.

Также, вы можете извлечь самую правую цифру или цифры из числа. Например, *x* % 10 вернет самую правую цифру числа *x* (по основанию 10).

### 2.9. Строковые операции

Оператор + работает со строками, но это не сложение в математическом смысле. Вместо этого, выполняется операция конкатенации (concatenation), которая объединяет строки одна за другой. Например,

```
>>> first = 10

>>> second = 15

>>> print first+second

25

>>> first = '100'
```

```
>>> second = '150'

>>> print first + second

100150
```

### 2.10. Ввод входных данных

Иногда необходимо вводить значение переменной, используя клавиатуру. Python предоставляет встроенную функцию *raw_input*, которая получает входные значения с клавиатуры. В новой версии Python 3.0 эта функция носит название *input*. Когда вызывается эта функция, программа останавливается и ожидает действий пользователя. Когда пользователь нажимает *Return* или *Enter*, программа продолжает выполнение и *raw_input* возвращает введенную пользователем строку.

```
>>> input = raw_input()

Some silly stuff

>>> print input

Some silly stuff
```

Перед тем, как получить от пользователя некоторые данные, будет хорошим тоном – вывести на экран информацию о том, что пользователю необходимо ввести. Вы можете передать эту информацию в качестве входного параметра функции *raw_input*:

```
>>> name = raw_input('What is your name?\n')

What is your name?

Chuck

>>> print name

Chuck
```

Последовательность *\n* в конце приглашения обозначает новую строку (newline), поэтому пользователь начинает ввод со следующей после приглашения строки.

Если вы ожидаете от пользователя ввода целочисленного значения, то можете воспользоваться функцией *int* для преобразования входной строки в число:

```
>>>  prompt = 'What...is the airspeed velocity of an unladen
swallow?\n'
```

```
>>> speed = raw_input(prompt)
What...is the airspeed velocity of an unladen swallow?
17
>>> int(speed)
17
>>> int(speed) + 5
22
```

Не всегда преобразование типов проходит гладко:

```
>>> speed = raw_input(prompt)

What...is the airspeed velocity of an unladen swallow?

What do you mean, an African or a European swallow?

>>> int(speed)


Traceback (most recent call last):

  File "<pyshell#10>", line 1, in <module>

    int(speed)

ValueError: invalid literal for int() with base 10: 'What do you
mean, an African or a European swallow?'
```

Позже мы рассмотрим, как обрабатывать подобные ошибки.

### 2.11. Комментарии

Когда программы разрастаются то, их становится сложно читать. Чтобы упростить чтение отдельных частей программы в программу добавляются заметки, которые на человеческом языке рассказывают, что программа делает. Эти заметки называются комментариями и начинаются с символа #.

```
# compute the percentage of the hour that has elapsed
percentage = (minute * 100) / 60
```

Комментарии можно оставлять в конце выражения:

```
percentage = (minute * 100) / 60 # percentage of an hour
```

Весь текст, начиная от # и до конца строки, игнорируется Python.

Комментарии призваны ответить на вопрос, почему? Они обычно содержат полезную информацию, которой НЕТ в коде.

## 2.14. Словарь

*Присваивание (assignment)*: выражение, в котором некоторое значение присваивается некоторой переменной.

*Конкатенация (concatenate):* объединение двух операндов один за другим.

*Комментарий (comment):* информация в программе, которая предназначена для программистов (или тех, кто читает исходный код) и не влияет на выполнение программы.

*Выражение (expression):* комбинация переменных, операторов и значений, представленных единственным результирующим значением (single result value).

*Плавающая точка (floating-point):* тип представления чисел с дробной частью.

*Остаток от деления (floor division):* операция, которая делит два числа и отсекает дробную часть.

*Целое число (integer):* тип представления целых чисел.

*Ключевые слова (keyword):* зарезервированные слова, которые используются компилятором для анализа программы; эти слова нельзя использовать в качестве имен переменных.

*Модульный оператор (modulus operator):* оператор, представленный символом (%), который работает с целыми числами и возвращает остаток от деления двух чисел.

*Операнд (operand):* одно из значений, которым оперирует оператор.

*Оператор (operator):* специальный символ, который представляет простые вычисления, подобные сложению, умножению или конкатенации строк.

*Правила старшинства (rules of precedence):* набор правил, которые определяют порядок вычисления выражений, содержащих несколько операторов и операндов.

*Диаграмма состояний (state diagram):* графическое представление множества переменных и связанных с ними значений.

*Инструкция (statement):* фрагмент кода, представляющий команду или действие. Например, инструкция присваивания или вывода на экран.

*Строка (string):* тип представления последовательности символов.

*Тип (type):* категории значений. Мы уже встречали целые (int), с плавающей точкой (float) и строковый (str).

*Значение (value):* один из базовых элементов данных, подобно числам или строкам, которыми манипулирует программа.

*Переменная (variable):* имя, связанное со значением.

## 2.15. Упражнения

1. Укажите примеры значений и типов в языке Python.
2. В чем заключается семантическая ошибка?
3. Что такое переменная?
4. Для чего в Python используются ключевые слова?
5. Что такое инструкция?
6. Что такое операторы и операнды?
7. Что такое выражение?
8. Напишите программу, которая возвращает самую правую цифру числа.
9. Напишите программу, которая возвращает результат конкатенации двух строк.
10. Напишите программу, которая позволяет пользователю ввести с клавиатуры произвольное значение.