Переменные, операторы на PYTHON

Урок 4

Актау, 2018

Переменные

python

Таблица 1.1. Ключевые слова Python

and	del	from	None	True
as	elif	global	nonlocal	try
assert	else	if	not	while
break	except	import	or	with
class	False	in	pass	yield
continue	finally	is	raise	
def	for	lambda	return	

Ключевые слова не могут быть модифицированы, и попытка использовать переменную с соответствующим именем приведет к ошибке.

Таблица 1.2. Встроенные идентификаторы Python

ArithmeticError	SyntaxError	float
AssertionError	SyntaxWarning	format
AttributeError	SystemError	frozenset
BaseException	SystemExit	getattr
BlockingIOError	TabError	globals
BrokenPipeError	TimeoutError	hasattr
BufferError	True	hash
BytesWarning	TypeError	help
ChildProcessError	UnboundLocalError	hex
ConnectionAbortedError	UnicodeDecodeError	id
ConnectionError	UnicodeEncodeError	input
ConnectionRefusedError	UnicodeError	int
ConnectionResetError	UnicodeTranslateError	isinstance
DeprecationWarning	UnicodeWarning	issubclass
EOFError	UserWarning	iter
Ellipsis	ValueError	len


```
EnvironmentError                Warning                 license
Exception                       WindowsError            list
False                           ZeroDivisionError       locals
FileExistsError                 __build_class__         map
FileNotFoundError               __debug__               max
FloatingPointError              __doc__                 memoryview
FutureWarning                   __import__              min
GeneratorExit                   __loader__              next
IOError                         __name__                object
ImportError                     __package__             oct
ImportWarning                   abs                     open
IndentationError                all                     ord
IndexError                      any                     pow
InterruptedError                ascii                   print
IsADirectoryError               bin                     property
KeyError                        bool                    quit
KeyboardInterrupt               bytearray               range
LookupError                     bytes                   repr
MemoryError                     callable                reversed
NameError                       chr                     round
None                            classmethod             set
NotADirectoryError              compile                 setattr
NotImplemented                  complex                 slice
NotImplementedError             copyright               sorted
OSError                         credits                 staticmethod
OverflowError                   delattr                 str
PendingDeprecationWarning       dict                    sum
PermissionError                 dir                     super
ProcessLookupError              divmod                  tuple
ReferenceError                  enumerate               type
ResourceWarning                 eval                    vars
RuntimeError                    exec                    zip
RuntimeWarning                  exit

StopIteration                   filter
```

Операторы

python

Обычно выделяют четыре группы операторов:

- арифметические;
- побитовые;
- операторы сравнения;
- логические операторы.

Арифметические операторы предназначены в первую очередь для выполнения арифметических вычислений. В таблице 1.3 перечислены и кратко описаны основные арифметические операторы языка Python.

Арифметические операторы

python

Таблица 1.3. Арифметические операторы

Оператор	Описание
+	Оператор сложения. Вычисляется сумма двух чисел
-	Оператор вычитания. Вычисляется разность двух чисел
*	Оператор умножения. Вычисляется произведение двух чисел
/	Оператор деления. Вычисляется отношение двух чисел
//	Оператор целочисленного деления. Вычисляется целая часть от деления одного числа на другое
%	Оператор вычисления остатка от целочисленного деления. Вычисляется остаток от деления одного числа на другое
**	Оператор возведения в степень. Результатом является число, получающееся возведением первого операнда в степень, определяемую вторым операндом

Арифметические операторы

python

Листинг 1.2. Арифметические операторы

```
a=(5+2)**2-3*2 # Результат 43
b=6-5/2        # Результат 3.5
c=10//4+10%3   # Результат 3
# Результаты вычислений выводим на экран
print("Результаты вычислений:")
print(a,b,c)
```

Результат выполнения этого программного кода представлен ниже:

Результат выполнения программы (из листинга 1.2)

```
Результаты вычислений:
43 3.5 3
```

Воспользуемся функцией eval ( ) , которая позволяет вычислять выражения, заданные в текстовом формате.

Листинг 1.3. Использование функции eval ()

```
a="(5+2)**2-3*2" # Текстовое значение
b="6-5/2"        # Текстовое значение
c="10//4+10%3"   # Текстовое значение
# Результаты вычислений выводим на экран.
# Для "вычисления" текстовых выражений
# используем функцию eval()
print("Результатывычислений:")
print(a+" =",eval(a))
print(b+" =",eval(b))
print(c+" =",eval(c))
```

В результате выполнения этого программного кода

Результат выполнения программы (из листинга 1.3)

```
Результаты вычислений:
(5+2)**2-3*2 = 43
6-5/2 = 3.5
10//4+10%3 = 3
```

python

Побитовые операторы

Таблица 1.4. Побитовые операторы

Оператор	Описание
~	Побитовая инверсия (унарный оператор - у него один операнд). Результатом является число, получающееся заменой нулей на единицы и единиц на нули в побитовом представлении операнда (сам операнд при этом не меняется)
&	Побитовое И. При вычислении результата сравниваются побитовые представления операндов. Если на одной и той же позиции в операндах стоят единицы, то в числе-результате на этой же позиции будет единица. В противном случае (то есть если хотя бы в одной из двух позиций нуль) в числе-результате на соответствующей позиции будет нуль
|	Побитовое ИЛИ. Сравниваются побитовые представления операндов. Если на одной и той же позиции в операндах стоят нули, то в числе-результате на этой же позиции будет нуль. В противном случае (то есть если хотя бы в одной из двух позиций единица) в числе-результате на соответствующей позиции будет единица
^	Побитовое ИСКЮЧАЮЩЕЕ ИЛИ. Результат вычисляется сравнением побитовых представлений операндов. Если на одной и той же позиции в операндах стоят разные значения (у одного числа нуль, а у другого единица), то в числе-результате на этой же позиции будет единица. В противном случае (то есть если на соответствующих позициях в операндах стоят одинаковые числа) в числе-результате на этой позиции будет нуль
<<	Сдвиг влево. Результат вычисляется так: в побитовом представлении первого операнда выполняется сдвиг влево. Количество разрядов, на которые выполняется сдвиг, определяется вторым операндом. Младшие недостающие биты заполняются нулями
>>	Сдвиг вправо. Для вычисления результата в побитовом представлении первого операнда выполняется сдвиг вправо. Количество разрядов, на которые выполняется сдвиг, определяется вторым операндом. Биты слева заполняются значением самого старшего бита (для положительных чисел это нуль, а для отрицательных единица)

Побитовые операторы

python

Листинг 1.4. Побитовые операторы

```
a=70>>3
b=~a
c=a<<1
print(a,b,c)
print(7|3,7&3,7^3)
```

Результат выполнения этого программного кода такой:

Результат выполнения программы (из листинга 1.4)

```
8 -9 16
7 3 4
```

На всякий случай приведем пояснения относительно результатов выполнения программного кода. Результат выражения `70>>3` - это число, получаемое сдвигом битового представления числа 70 на 3 позиции вправо (с потерей младших битов). Двоичный код для числа 70 имеет вид 00...0001000110 (три бита, которые "пропадают" после сдвига вправо, выделены жирным шрифтом). После сдвига на 3 позиции вправо получаем 00...0000001000. Это код числа 8. Такой же результат можно было получить проще, если вспомнить, что сдвиг вправо на одну позицию эквивалентен целочисленному делению (делению без остатка) на число 2. Если 70 трижды поделить без остатка на 2, получим 8 (35 после первого деления, 17 после второго деления, и 8 после третьего деления).

Далее, если применить побитовое инвертирование к числу 00...0000001000 (значение переменной `a`), получим бинарный код 11...1111110111, который соответствует отрицательному числу -9. Желающие могут выполнить проверку самостоятельно, однако если вспомнить, что результатом операции `~a+1` является код для значения `-a` (в данном случае это -8), то несложно догадаться, что `~a` соответствует значению `-a-1` (то есть в данном случае это -9).

Значение выражения `a<<1` получаем сдвигом бинарного кода для значения переменной `a` на одну позицию вправо с заполнением младшего бита нулем (соответствует умножению значения переменной `a` на 2). Поскольку значение переменной `a` равно 8, то значение выражения `a<<1` равняется 16.

В следующих выражениях используются числа 7 (бинарный код 00...000111) и 3 (бинарный код 00..000011). Для побитовых операций `|` (или), `&` (и), `^` (исключающее или) получаем следующие результаты:

	00 ... 000111
|	00 ... 000011
7	00 ... 000111
	00 ... 000111
&	00 ... 000011
3	00 ... 000011
	00 ... 000111
^	00 ... 000011
4	00 ... 000100

В левом нижнем углу жирным шрифтом выделен результат соответствующей операции в десятичной системе счисления.

Логические операторы

python

С *логическими значениями* мы столкнемся при проверке условий в условной инструкции (операторе). Значений у логического типа всего два: `True` (*истина*) и `False` (*ложь*). Для работы со значениями логического типа предназначены специальные операторы, которые принято называть логическими. Используемые в Python *логические операторы* описаны в таблице 1.5.

Логические операторы

python

Таблица 1.5. Логические операторы

Оператор	Описание
`or`	Бинарный оператор (у оператора два операнда). Логическое *ИЛИ*. В общем случае результатом выражения `x or y` является `True`, если значение хотя бы одного из операндов `x` или `y` равно `True`. Если значения обеих операндов `x` и `y` равны `False`, результатом выражения `x or y` будет `False`. В Python выражения на основе оператора `or` вычисляются по упрощенной схеме: если первый операнд `x` интерпретируется как `True`, то `x` возвращается в качестве результата (второй операнд при этом не вычисляется). Если первый операнд `x` интерпретируется как `False`, то в качестве результата возвращается второй операнд `y`
`and`	Бинарный оператор (у оператора два операнда). Логическое *И*. В общем случае результатом выражения `x and y` является значение `True`, если значения обеих операндов `x` и `y` равны `True`. Если значение хотя бы одного из операндов `x` или `y` равно `False`, результатом выражения `x and y` будет `False`. В Python выражения на основе оператора `and` вычисляются по упрощенной схеме: если первый операнд `x` интерпретируется как `False`, то `x` возвращается в качестве результата (второй операнд при этом не вычисляется). Если первый операнд `x` интерпретируется как `True`, то в качестве результата возвращается второй операнд `y`
`not`	Логическое отрицание. Унарный оператор (у оператора один операнд). Результатом выражения `not x` будет значение `True`, если у операнда `x` значение `False`. Результатом выражения `not x` будет значение `False`, если у операнда `x` значение `True`

python

Листинг 1.5. Логические операторы

```
a=True
b=nota
print(a,b)
c=a and b
d=aorb
print(c,d)
```

Результат выполнения программного кода такой:

Результат выполнения программы (из листинга 1.5)

```
True False
False True
```

На заметку

В данном случае переменная `a` ссылается на логическое значение `True`. Значение переменной `b` - это значение выражения `not a`. Поскольку `a` - это `True`, то значение выражения `not a` равно `False`. Поэтому результатом выражения `a and b` является значение `False`, а значением выражения `a or b` является значение `True`.

Хотя по своей сути логические операторы должны возвращать логические значение, в Python это далеко не всегда так. Результатом выражений на основе операндов `or` и `and` является один из операндов соответствующего выражения. А операнды не обязательно должны быть логического типа - достаточно, чтобы они могли интерпретироваться как логические значения. Ситуацию иллюстрирует следующий пример (программный код в листинге 1.6).

Листинг 1.6. Снова логические операторы

```
x=10       # Числовая переменная
y=20       # Числовая переменная
z=x and y  # Логическое И
print(z)   # Результат логического И
z=x or y   # Логическое ИЛИ
print(z)   # Результат логического ИЛИ
# Логическое отрицание
print(not x)
```

В данном случае логические операторы используются с числовыми операндами. При выполнении программного кода получаем такой результат:

Результат выполнения программы (из листинга 1.6)

```
20
10
False
```

Операторы сравнения

python

Таблица 1.6. Операторы сравнения

Оператор	Описание
<	Строго меньше. Результатом является `True`, если значение операнда слева от оператора *меньше* значения операнда справа от оператора. Иначе возвращается значение `False`
>	Строго больше. Результатом является `True`, если значение операнда слева от оператора *больше* значения операнда справа от оператора. Иначе возвращается значение `False`
<=	Меньше или равно. Результатом является `True`, если значение операнда слева от оператора *не больше* значения операнда справа от оператора. Иначе возвращается значение `False`

Операторы сравнения

python

Оператор	Описание
`>=`	Больше или равно. Результатом является `True`, если значение операнда слева от оператора *не меньше* значения операнда справа от оператора. Иначе возвращается значение `False`
`==`	Равно. Результатом является `True`, если значение операнда слева от оператора *равно* значению операнда справа от оператора. Иначе возвращается значение `False`
`!=`	Не равно. Результатом является `True`, если значение операнда слева от оператора *не равно* значению операнда справа от оператора. Иначе возвращается значение `False`
`is`	Оператор проверки идентичности объектов. В качестве результата возвращается значение `True`, если оба операнда ссылаются на один и тот же объект. В противном случае (то есть если операнды ссылаются на разные объекты) возвращается значение `False`
`is not`	Оператор проверки неидентичности объектов. В качестве результата возвращается значение `True`, если операнды ссылаются на разные объекты. В противном случае (то есть если операнды ссылаются на один и тот же объект) возвращается значение `False`

Листинг 1.7. Операторы сравнения

```
a=100
b=200
print(a<b,a>=b,a==100,b!=199)
```

Результат выполнения кода такой:

Результат выполнения программы (из листинга 1.7)

```
True False True True
```

Таблица 1.7. Некоторые математические функции

Функция	Описание
`abs()`	Вычисление модуля числа. Число, для которого вычисляется модуль, указывается аргументом функции. Может использоваться с комплексными числами. Например, результатом каждого из выражений `abs(5.0)`, `abs(-5.0)` и `abs(3+4j)` является значение `5.0`
`bin()`	Функция предназначена для преобразования числа из десятичной системы счисления в двоичную. Исходное число указывается аргументом функции, а результатом является текстовое представление для двоичного кода числа. Например, результатом выражения `bin(9)` будет текст `"0b1001"`
`complex()`	Функция используется для создания комплексных чисел на основе (переданных аргументами функции) действительной и мнимой частей числа, или на основе текстового представления комплексного числа. Например, результатом выражений `complex(3,4)` и `complex("3+4j")` будет комплексное число `3+4j`
`float()`	Функция используется для преобразования числовых значений и текстовых представлений для действительных чисел в числовые значения типа `float`. Например, результатом каждого из выражений `float(5)`, `float("5")`, `float("5.")` и `float("5.0")` является значение `5.0`
`hex()`	Функция предназначена для преобразования числа из десятичной системы счисления в шестнадцатеричную. Аргумент функции - число в десятичной системе счисления. Результат функции - текстовое представление этого числа в шестнадцатеричной системе. Например, результатом выражения `hex(123)` будет текст `"0x7b"`

python

python

`int()`	Функция для преобразования объекта (например, текста) в целое число. Если аргументом функции передано действительное число (тип `float`), то результат вычисляется отбрасыванием дробной части в действительном числе. Если аргументом указать текстовое представление целого числа, результатом будет само это число. Причем число (в текстовом представлении) может быть не только в десятичной системе, но и в двоичной, восьмеричной или шестнадцатеричной. В этом случае вторым аргументом указывается целое число, определяющее исходную систему счисления (соответственно, 2, 8 или 16). Например, результатом каждого из выражений `int(123.4)`, `int("123")`, `int("0b1111011",2)`, `int("0o173",8)` и `int("0x7b",16)` является значение 123
`max()`	Функция для вычисления максимального значения из набора чисел. Например, результатом выражения `max(-2,4,9,-1)` является значение 9
`.min()`	Функция для вычисления минимального значения из набора чисел. Например, результатом выражения `min(-2,4,9,-1)` является значение -2
`oct()`	Функция предназначена для преобразования числа из десятичной системы счисления в восьмеричную. Аргументом указывается число в десятичной системе счисления, а результатом является текстовое представление этого числа в восьмеричной системе. Например, результатом выражения `oct(9)` будет текст `"0o11"`
`pow()`	Функция для возведения в степень числа. Если у функции два аргумента, то результатом является первое число в степени, определяемой вторым числом. Другими словами, результатом выражения `pow(x,y)` является значение `x**y`. Если у функции три аргумента, то после возведения в степень вычисляется остаток от деления на третий аргумент: то есть результатом выражения `pow(x,y,z)` является значение `(x**y)%z`. Например, результатом выражения `pow(2,3)` будет значение 8, а результатом выражения `pow(2,3,5)` будет значение 3

Функция	Описание
round()	Функция предназначена для округления действительных значений до целочисленных. Аргументом указывается округляемое значение. Округление выполняется по таким правилам: • если дробная часть округляемого значения *меньше* 0.5, округление выполняется до ближайшего меньшего целого числа; • если дробная часть округляемого значения *больше* 0.5, округление выполняется до ближайшего большего целого числа; • если дробная часть округляемого значения *равняется* 0.5, округление выполняется до ближайшего четного целого числа. Если функции передать второй аргумент, то он будет определять количество знаков в дробной части, до которых будет выполняться округление (то есть в этом случае округление выполняется не до целого числа, а до действительного с количеством разрядов после запятой, определяемым вторым аргументом функции). Например, результатом выражения `round(4.6)` (дробная часть `0.6`) есть значение `5`, у выражения `round(-4.6)` (дробная часть `0.4`) значение `-5`, у выражения `round(4.4)` (дробная часть `0.4`) значение `4`, у выражения `round(-4.4)` (дробная часть `0.6`) значение `-4`, у выражения `round(4.5)` (дробная часть `0.5`) значение `4`, а у выражения `round(-4.5)` (дробная часть `0.5`) - соответственно, `-4`. Для сравнения: результат выражения `round(1.23456,3)` - число `1.235`