Лекция №7.

Тема: **Описание действий над данными в языке Python.**

Классификация операторов. Управляющие операторы. Операторы для реализации базовых структур алгоритмов (линейные, разветвляющиеся, циклические).

Ключевые слова: простой оператор, составной оператор, оператор присваивания, составной оператор присваивания, пустой оператор, оператор проверки условия, оператор цикла с условием, оператор цикла с параметром.

Keywords: simple statement, compound statement; assignment statements, augmented assignment statements, pass statement, assert statement, while statement, for statement.

1 Классификация операторов. Управляющие операторы.

Операторы языка Python описывают некоторые алгоритмические действия, которые необходимо выполнить для решения задачи. Каждый оператор выражает некоторое действие над данными. Тело программы представлено блоком операторов. Для разделения операторов используется символ возврата каретки (нотация языка Python) или символ «;» (нотация языка С).

По критерию использования в базовых структурах алгоритмов все операторы языка Python разделяются на такие виды:

- операторы, реализующие линейную базовую структуру;
- операторы, реализующие разветвляющуюся базовую структуру;
- операторы, реализующие циклическую базовую структуру.

По форме все операторы языка Python разделяются на простые и составные операторы.

Простые операторы не содержат в себе других операторов. К простым операторам языка Python относят (см. https://docs.python.org/3.5/reference/simple_stmts.html):

- выражения;
- оператор присваивания;
- составной оператор присваивания;
- пустой оператор «pass»;
- оператор проверки условия «assert»;
- оператор удаления «del» (будет рассмотрен в другой лекции);
- оператор выхода из функции «return» (будет рассмотрен в другой лекции);
- оператор, возвращающий из функции объект генератора «yield» (будет рассмотрен в другой лекции);
- оператор генерации исключения «raise» (будет рассмотрен в другой лекции);
- операторы, расширяющие возможности передачи управления «break» и «continue»;
- оператор импорта модуля «import» (будет рассмотрен в другой лекции);
- операторы управления областью видимости переменных «global» и «nonlocal» (будут рассмотрены в другой лекции).

Формальное определение простого оператора:

```
simple_stmt =  expression_stmt | assert_stmt | assignment_stmt
               | augmented_assignment_stmt | pass_stmt
               | del_stmt | return_stmt | yield_stmt
               | raise_stmt | break_stmt | continue_stmt
               | import_stmt | global_stmt | nonlocal_stmt
```

Выражения позволяют вычислить значение некоторого типа данных. Выражение может включать в себя вызов стандартной или пользовательской функции. Пример

```
>>> x = pow(2, 4, 5) + pow(2, 4) + 2**4
>>> x
33
```

С помощью оператора присваивания переменная получает значение выражения, которое заменяет любое предыдущее значение переменной. Слева от знака присваивания записывается идентификатор, справа – выражение, значение которого вычисляется перед присваиванием. Допустимо присваивание значений переменным любого типа. Явно указывать тип переменной не нужно. Python самостоятельно отслеживает типы переменных на основе присваиваемых им значений. Пример

```
>>> import math
>>> # Переменной x присваиваем значение выражения.
>>> x = 5 + 10
>>> # Тип переменной x - целочисленный тип данных.
>>> type(x)
<class 'int'>
>>> x
15
>>> # Переменной x присваиваем значение выражения.
>>> x = 4 * math.sin(math.radians(30))
>>> # Тип переменной x - вещественный тип данных.
>>> type(x)
<class 'float'>
>>> x
1.9999999999999998
```

Составной оператор присваивания в языке Python объединяет простое присваивание с одной из следующих бинарных операций: "+"; "-"; "*"; "/"; "//"; "%"; "**"; ">>"; "<<"; "&"; "^"; "|". Пример

```
>>> a, b, c, d = 4, 6, 10, 12
>>> a += b
>>> b %= c
>>> c //=d
>>> print(a, b, c, sep='; ')
10; 6; 0
```

Пустой оператор не выполняет никакого действия и отображается в программе с помощью ключевого слова pass. Оператор pass может использоваться в качестве заглушки для составного оператора, функции или класса. Пример

```
>>> # Использование pass в качестве заглушки для функции
>>> def MyFunction():
      pass
```

Оператор assert возбуждает исключение, если логическое выражение возвращает значение False. Оператор assert имеет следующий формат:

```
assert_stmt = "assert" expression ["," expression]
```

Пример использования оператора assert

```
# Вычисление квадратного корня из целого числа.
import math
x = int(input())
assert (x >= 0.0), "Ошибка ввода. Переменная x не может принимать отрицательное значение"
print("sqrt(x) = ", math.sqrt(x))
```

Результат работы программы:

```
>>>
16
sqrt(x) =  4.0
```

```
>>>
-8
Traceback (most recent call last):
  File "example.py", line 3, in <module>
    assert (x >= 0.0), "Ошибка ввода. Переменная x не может принимать отрицательное значение"
AssertionError: Ошибка ввода. Переменная x не может принимать отрицательное значение
```

Составные операторы состоят из других операторов. К составным операторам языка Python относят:

- оператор ветвления «if»;
- оператор цикла с условием «while»;
- оператор цикла с параметром «for»;
- оператор обработки исключений «try» (будет рассмотрен в другой лекции).
- оператор для работы с объектами контекстных менеджеров «with» (будет рассмотрен в другой лекции);
- оператор определения функции «funcdef» (будет рассмотрен в другой лекции);

- оператор определения класса «classdef» (будет рассмотрен в другой лекции);
- оператор декоратора «decorated» (будет рассмотрен в другой лекции).

Формальное определение составного оператора:

```
compound_stmt ::=  if_stmt | while_stmt | for_stmt | try_stmt
                   | with_stmt | funcdef | classdef | decorated
```

Составной оператор объединяет несколько последовательно выполняемых операторов, которые оформляются с одинаковыми отступами от левого края. Составной оператор используется в тех случаях, когда по правилам построения синтаксической конструкции можно использовать один оператор, но необходимо выполнить несколько операторов. Операторы внутри составного оператора разделяются символом возврата каретки или символом «;». Блок операторов в самом теле программы можно рассматривать как составной оператор, который объединяет действия над данными (операторы) в соответствии с некоторым алгоритмом.

Особым классом операторов являются управляющие операторы. Управляющие операторы изменяют последовательный порядок выполнения действий над данными. К управляющим операторам языка Python относятся:

- оператор ветвления «if»;
- оператор цикла с условием «while»;
- оператор цикла с параметром «for».

2 Операторы для реализации базовых структур алгоритмов (линейные, разветвляющиеся, циклические)

Простые операторы реализуют линейную базовую структуру.

Оператор ветвления «if» реализует разветвляющуюся базовую структуру. Синтаксис и примеры использования данного оператора были подробно рассмотрены в предыдущей лекции.

Оператор цикла с условием «while», оператор цикла с параметром «for» реализуют циклическую базовую структуру. Виды циклов, которые возможно организовать в языке Python, представлены на рис. 1.

Простые циклы содержат один циклический оператор, сложные циклы допускают вложенность циклических операторов. В языке Python существует две возможности управления циклом: использование логического выражения и итератора. При использовании логического выражения количество повторений цикла неизвестно, а механизм управления циклом находится в начале цикла (предусловие) или в конце цикла (постусловие). При использовании итератора количество повторений цикла устанавливается с помощью параметра - последовательности. Рассмотрим подробнее синтаксис и примеры использования операторов цикла.

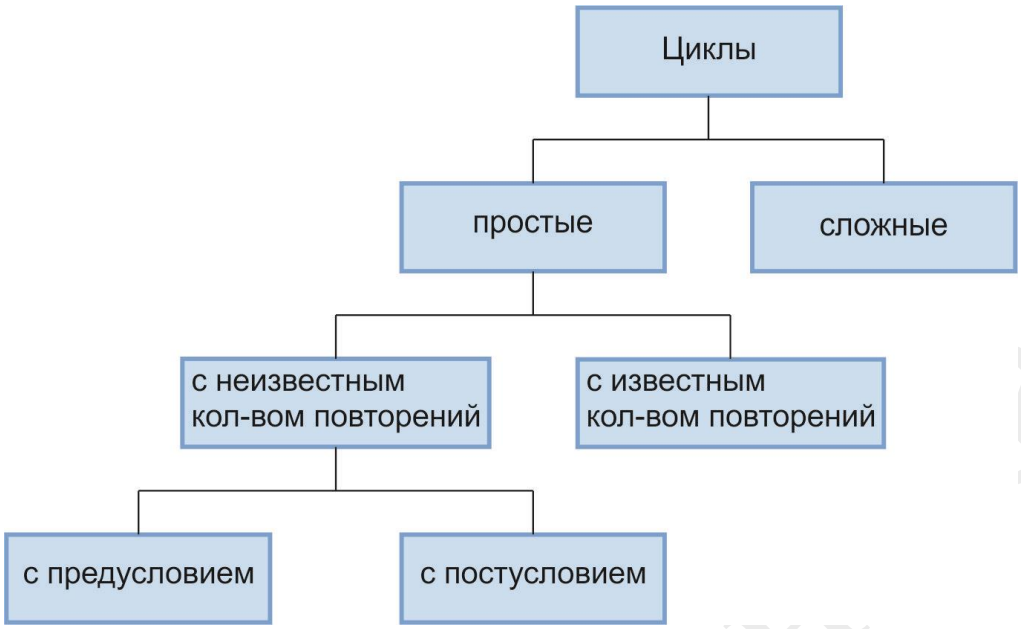

Рис. 1. Классификация циклов в языке Python

Оператор цикла «for» выполняет перебор элементов заданной последовательности (например, строка, список или кортеж) или другого итерируемого объекта. Оператор цикла с параметром «for» имеет следующий синтаксис

```
for_stmt = "for" target_list "in" expression_list ":"
                suite
           ["else" ":" suite]
```

В этом операторе "target_list" является параметром, которому присваивается значение элемента последовательности, "expression_list" – последовательность, обход которой будет выполняться, "suite" – блок операторов (тело цикла). Итерации цикла будут выполняться до тех пор, пока не закончиться обход всех элементов заданной последовательности. Выполнение очередной итерации включает в себя сначала выполнение тела цикла, а затем присваивание параметру цикла следующего элемента последовательности.

Оператор «for» в языке Python поддерживает необязательную часть «else», которая срабатывает после завершения обхода элементов последовательности (завершения цикла).

Часто параметр в операторе «for» должен принимать значения, являющиеся элементами арифметической прогрессии (например, 0, 2, 3, …, 9). Для создания последовательности целых чисел в заданном диапазоне (start÷stop) и с заданным шагом (step) используется функция range( ):

```
range(stop) -> range object
range(start, stop[, step]) -> range object
```

Пример работы функции range( ):

```
>>> list(range(5))
[0, 1, 2, 3, 4]
>>> list(range(5, 10))
[5, 6, 7, 8, 9]
>>> list(range(5, 15, 3))
[5, 8, 11, 14]
```

Рассмотрим простые примеры работы с оператором «for».

```
>>> # Оператор выводит значения температур по Цельсию
>>> # в диапазоне [-270C;200C] через каждые 10C и
>>> # соответствующие значения температур по Кельвину.
>>> for i in range(-27, 21):
        tc = 10 * i
        tk = tc + 273
        print(tc, ' ', tk)
```

```
>>> # Оператор выводит через пробел строчные латинские буквы
>>> # в алфавитном порядке.
>>> for i in range(ord('a'), ord('z') + 1):
        print(chr(i))
```

```
>>> # Оператор выводит через пробел строчные латинские буквы
>>> # в обратном порядке.
>>> for i in range(ord('z'), ord('a') - 1, -1):
        print(chr(i))
```

Оператор цикла с предусловием «while» проверяет условие и при его истинности выполняет тело цикла. Оператор цикла с предусловием «while» имеет следующий синтаксис:

```
while_stmt = "while" expression ":"
                    suite
               ["else" ":" suite]
```

В этом операторе "expression" является логическим выражением, истинность которого проверяется в начале каждой итерации. Логическое выражение может состоять из нескольких частей, соединённых логическими операциями and/or/not. Оператор "suite", выполняемый в цикле, может быть как простым, так и составным.

Сначала вычисляется истинность логического выражения (условия). Пока условие цикла истинно, выполняется очередная итерация цикла, иначе происходит выход из цикла. Выполнение очередной итерации включает в себя выполнение оператора/операторов. Так как истинность условия проверяется в начале каждой итерации, то тело цикла может не выполниться ни разу.

Оператор «while» в языке Python поддерживает необязательную часть «else», которая срабатывает после того, как "expression" вернет значение False.

Минимальный бесконечный цикл с предусловием может быть реализован таким образом

```
>>> while True:
        pass
```

Рассмотрим простые примеры работы с оператором «while».

```
>>> # Оператор выводит значения температур по Цельсию
>>> # в диапазоне [-270C;200C] через каждые 10C и
>>> # соответствующие значения температур по Кельвину.
>>> tc = -280
>>> while tc < 200:
        tc += 10
        tk = tc + 273
        print(tc, ' ', tk)
```

```
>>> # Оператор определяет число натуральных чисел,
>>> # сумма которых не превышает значение s
>>> s = int(input())
150
>>> k = 0
>>> summa = 0
>>> while summa <= s:
        k += 1
        summa += k
>>> k
17
```

Оператор цикла с постусловием выполняет тело цикла, а затем проверяет условие продолжение цикла. В языке Python для организации цикла с постусловием используется оператор «while» совместно с оператором ветвления «if». Формально цикл с постусловием можно определить следующим образом

```
do_while_stmt = "while" "True" ":"
                    suite
                    "if" expression ":" "break"
                ["else" ":" suite]
```

В этом операторе "expression" является логическим выражением, истинность которого проверяется в конце каждой итерации. Логическое выражение может состоять из нескольких частей, соединённых логическими операциями and/or/not. Оператор "suite", выполняемый в цикле, может быть как простым, так и составным. Пока условие цикла ложно, выполняется очередная итерация цикла, иначе происходит выход из цикла. Так как истинность условия проверяется в конце каждой итерации, то тело цикла выполнится хотя бы один раз.

Минимальный бесконечный цикл с постусловием может быть реализован таким образом

```
>>> while True:
        pass
        if False: break
```

Рассмотрим простые примеры организации цикла с постусловием с помощью оператора «while».

```
>>> # Оператор выводит значения температур по Цельсию
>>> # в диапазоне [-270C;200C] через каждые 10C и
>>> # соответствующие значения температур по Кельвину.
>>> tc = -280
>>> while True:
        tc += 10
        tk = tc + 273
        print(tc, ' ', tk)
        if (tc >= 200): break
```

```
>>> # Оператор определяет число натуральных чисел,
>>> # сумма которых не превышает значение s
>>> s = int(input())
150
>>> k = summa = 0
>>> while True:
     k += 1
     summa += k
     if summa > s : break


>>> k
17
```

Для преждевременного завершения выполнения тела цикла существуют стандартные операторы:

- break – прерывает выполнение оператора цикла и передает управление следующему оператору в программе. Часть «else» в операторе цикла (если она присутствует) не выполняется;
- continue – досрочное завершает итерацию цикла и переходит к следующей итерации.

Рассмотрим примеры использования данных операторов.

```
>>> # Оператор выводит числа от 1 до 100,
>>> # кроме чисел от 5 до 10 включительно
>>> for i in range(1, 101):
     if i > 4 and i < 11:
          continue
     print(i, end = ' ')
```

```
>>> # Оператор выводит числа от 1 до 100.
>>> i = 1
>>> while True:
     if i > 100: break
     print(i, end = ' ')
     i += 1
```

Рассмотрим примеры построения циклических алгоритмов.

Пример №1: Написать программу для вычисления выражения $\frac{m!+n!}{(m+n)!}$.

```
>>> m, n = int(input()), int(input())
4
5
>>> c = 1
>>> for i in range(1, m + n + 1):
     c *= i
     if i == m: a = c
     if i == n: b = c
>>> p = (a + b) / c
>>> p
```

Пример №2. Написать программу-экзаменатор для выставления оценки за знание таблицы умножения

```
import random
# Количество вопросов.
n = 10
e = 0
for q in range(n):
    a = random.randint(1, n)
    b = random.randint(1, n)
    print(q + 1, '; ', a, '*', b, ' = ')
    c = int(input())
    # Если ответ неверный, увеличивается счетчик ошибок.
    if (a * b) != c: e += 1
if e == 0: print('Отлично')
elif e == 1 or e == 2: print('Хорошо')
elif e in range(3, 6): print('Удовлетворительно')
else: print('Неудовлетворительно')
```