Лабораторная работа №3

Циклы

1. Цикл for

Цикл for, также называемый циклом с параметром, в языке Питон богат возможностями. В цикле for указывается переменная и множество значений, по которому будет пробегать переменная. Множество значений может быть задано списком, кортежем, строкой или диапазоном.

Вот простейший пример использования цикла, где в качестве множества значений используется кортеж:

```
i = 1
for color in 'red', 'orange', 'yellow', 'green', 'cyan', 'blue', 'violet':
    print('#', i, ' color of rainbow is ', color, sep = '')
    i += 1
```

В этом примере переменная `color` последовательно принимает значения `'red'`, `'orange'` и т.д. В теле цикла выводится сообщение, которое содержит название цвета, то есть значение переменной `color`, а также номер итерации цикла число, которое сначала равно 1, а потом увеличивается на один (инструкцией i += 1 с каждым проходом цикла.

Инструкция i += 1 эквивалентна конструкции i = i + 1 (это просто сокращенная запись). Такую сокращенную запись можно использовать для всех арифметических операций: *=, -=, /=, %=...

В списке значений могут быть выражения различных типов, например:

```
for i in 1, 2, 3, 'one', 'two', 'three':
    print(i)
```

При первых трех итерациях цикла переменная i будет принимать значение типа `int`, при последующих трех — типа `str`.

1.2. Функция range

Как правило, циклы `for` используются либо для повторения какой-либо последовательности действий заданное число раз, либо для изменения значения переменной в цикле от некоторого начального значения до некоторого конечного.

Для повторения цикла некоторое заданное число раз n можно использовать цикл `for` вместе с функцией `range`:

```
for i in range(4):  # равносильно инструкции for i in 0, 1, 2, 3:
    # здесь можно выполнять циклические действия
    print(i)
    print(i ** 2)
# цикл закончился, поскольку закончился блок с отступом
print('Конец цикла')
```

В качестве n может использоваться числовая константа, переменная или произвольное арифметическое выражение (например, 2 ** 10). Если значение n равно нулю или отрицательное, то тело цикла не выполнится ни разу.

Функция `range` может также принимать не один, а два параметра. Вызов **`range`**(a, b) означает, что индексная переменная будеть принимать значения от a до b - 1, то есть первый параметр функции `range`, вызываемой с двумя параметрами, задает начальное значение индексной переменной, а второй параметр — первое значение, которое индексная переменная принимать **не будет**. Если же $a \geq b$, то цикл не будет выполнен ни разу.

Например, для того чтобы просуммировать значения чисел от 1 до n можно воспользоваться следующей программой:

```
sum = 0
n = 5
for i in range(1, n + 1):
    sum += i
print(sum)
```

В этом примере переменная i принимает значения 1, 2, ..., n, и значение переменной `sum` последовательно увеличивается на указанные значения.

Наконец, чтобы организовать цикл, в котором индексная переменная будет уменьшаться, необходимо использовать функцию `range` с тремя параметрами. Первый параметр задает начальное значение индексной переменной, второй параметр — значение, до которого будет изменяться индексная переменная (не включая его!), а третий параметр — величину изменения индексной переменной. Например, сделать цикл по всем нечетным числам от 1 до 99 можно при помощи функции `range`(1, 100, 2), а сделать цикл по всем числам от 100 до 1 можно при помощи `range`(100, 0, -1).

Более формально, цикл `for i in range(a, b, d)` при `d` > 0 задает значения индексной переменной `i = a, i = a + d, i = a + 2 * d` и так для всех значений, для которых i < b. Если же d < 0, то переменная цикла принимает все значения i > b.

1.3. Настройка функции print()

По умолчанию функция `print`() принимает несколько аргументов, выводит их через пробел, после чего ставит перевод строки. Это поведение можно изменить, используя именованные параметры `sep` (разделитель) и `end` (окончание).

```
print(1, 2, 3)
print(4, 5, 6)
print(1, 2, 3, sep=', ', end='. ')
print(4, 5, 6, sep=', ', end='. ')
print()
print(1, 2, 3, sep='', end=' -- ')
print(4, 5, 6, sep=' * ', end='.')
```

2. Цикл while

Цикл `while` (“пока”) позволяет выполнить одну и ту же последовательность действий, пока проверяемое условие истинно. Условие записывается до тела цикла и проверяется до выполнения тела цикла. Как правило, цикл `while` используется, когда невозможно определить точное значение количества проходов исполнения цикла.

Синтаксис цикла `while` в простейшем случае выглядит так:

```
while условие:
    блок инструкций
```

При выполнении цикла `while` сначала проверяется условие. Если оно ложно, то выполнение цикла прекращается и управление передается на следующую инструкцию после тела цикла `while`. Если условие истинно, то выполняется инструкция, после чего условие проверяется снова и снова выполняется инструкция. Так продолжается до тех пор, пока условие будет истинно. Как только условие станет ложно, работа цикла завершится и управление передастся следующей инструкции после цикла.

Например, следующий фрагмент программы напечатает на экран квадраты всех целых чисел от 1 до 10. Видно, что цикл `while` может заменять цикл `for` ... `in` `range`(...):

```
i = 1
while i <= 10:
```

```
    print(i ** 2)
    i += 1
```

В этом примере переменная i внутри цикла изменяется от 1 до 10. Такая переменная, значение которой меняется с каждым новым проходом цикла, называется счетчиком. Заметим, что после выполнения этого фрагмента значение переменной i будет равно 11, поскольку именно при i == 11 условие i <= 10 впервые перестанет выполняться.

Вот еще один пример использования цикла `while` для определения количества цифр натурального числа n:

```
n = int(input())
length = 0
while n > 0:
    n //= 10  # это эквивалентно n = n // 10
    length += 1
print(length)
```

В этом цикле мы отбрасываем по одной цифре числа, начиная с конца, что эквивалентно целочисленному делению на `10 (n //= 10)`, при этом считаем в переменной `length`, сколько раз это было сделано.

В языке Питон есть и другой способ решения этой задачи: `length = len(str(i))`.

2.1. Инструкции управления циклом

После тела цикла можно написать слово `else`: и после него блок операций, который будет выполнен один раз после окончания цикла, когда проверяемое условие станет неверно::

```
i = 1
while i <= 10:
    print(i)
    i += 1
else:
    print('Цикл окончен, i =', i)
```

Казалось бы, никакого смысла в этом нет, ведь эту же инструкцию можно просто написать после окончания цикла. Смысл появляется только вместе с инструкцией `break`. Если во время выполнения Питон встречает инструкцию `break` внутри цикла, то он сразу же прекращает выполнение этого цикла и выходит из него. При этом ветка `else` исполняться не будет. Разумеется, инструкцию `break` осмысленно вызывать только внутри инструкции `if`, то есть она должна выполняться только при выполнении какого-то особенного условия.

Приведем пример программы, которая считывает числа до тех пор, пока не встретит отрицательное число. При появлении отрицательного числа программа завершается. В первом варианте последовательность чисел завершается числом 0 (при считывании которого надо остановиться).:

```
a = int(input())
while a != 0:
    if a < 0:
        print('Встретилось отрицательное число', a)
        break
    a = int(input())
else:
    print('Ни одного отрицательного числа не встретилось')
```

Во втором варианте программы сначала на вход подается количество элементов последовательности, а затем и сами элементы. В таком случае удобно воспользоваться

циклом `for`. Цикл `for` также может иметь ветку `else` и содержать инструкции `break` внутри себя:

```
n = int(input())
for i in range(n):
    a = int(input())
    if a < 0:
        print('Встретилось отрицательное число', a)
        break
else:
    print('Ни одного отрицательного числа не встретилось')
```

Другая инструкция управления циклом — `continue` (продолжение цикла). Если эта инструкция встречается где-то посередине цикла, то пропускаются все оставшиеся инструкции до конца цикла, и исполнение цикла продолжается со следующей итерации.

Если инструкции `break` и `continue` содержатся внутри нескольких вложенных циклов, то они влияют лишь на исполнение самого внутреннего цикла. Вот не самый интеллектуальный пример, который это демонстрирует:

```
for i in range(3):
    for j in range(5):
        if j > i:
            break
        print(i, j)
```

Увлечение инструкциями `break` и `continue` не поощряется, если можно обойтись без их использования. Вот типичный пример плохого использования инструкции `break` (данный код считает количество знаков в числе).

```
n = int(input())
length = 0
while True:
    length += 1
    n //= 10
    if n == 0:
        break
print('Длина числа равна', length)
```

Гораздо лучше переписать этот цикл так:

```
n = int(input())
length = 0
while n != 0:
    length += 1
    n //= 10
print('Длина числа равна', length)
```

Впрочем, на Питоне можно предложить и более изящное решение:

```
n = int(input())
print('Длина числа равна', len(str(n)))
```

2.2 Множественное присваивание

В Питоне можно за одну инструкцию присваивания изменять значение сразу нескольких переменных. Делается это так:

```
a, b = 0, 1
```

Отличие двух способов состоит в том, что множественное присваивание в первом способе меняет значение двух переменных одновременно.

Если слева от знака «=» в множественном присваивании должны стоять через запятую имена переменных, то справа могут стоять произвольные выражения, разделённые запятыми. Главное, чтобы слева и справа от знака присваивания было одинаковое число элементов.

Множественное присваивание удобно использовать, когда нужно обменять значения двух переменных. В обычных языках программирования без использования специальных функций это делается так:

```
a = 1
b = 2
tmp = a
a = b
b = tmp
print(a, b)
# 2 1
```

В Питоне то же действие записывается в одну строчку:

```
a = 1
b = 2
a, b = b, a
print(a, b)
# 2 1
```

3. Практические задания

К любой задаче по требованию преподавателя уметь строить блок-схемы.

1. Напечатать таблицу перевода 1, 2, ... 20 долларов США в рубли по текущему курсу (значение курса вводится с клавиатуры).
2. Напечатать таблицу умножения на число n (значение n вводится с клавиатуры; $1 \le n \le 9$).
3. Найти сумму квадратов всех целых чисел от a до b (значения a и b вводятся с клавиатуры; $b \ge a$).
4. Дано пятизначное число. Найти число, получаемое при прочтении его цифр справа налево.
5. Вычислить сумму 1! 2! 3! ... n!, k! 1 2 3 ... k (значение n вводится с клавиатуры; $1 < n \le 10$).
6. Дана последовательность из n вещественных чисел, начинающаяся с отрицательного числа. Определить, какое количество отрицательных чисел записано в начале последовательности. Условный оператор не использовать.
7. Последовательность Фибоначчи образуется так: первый и второй члены последовательности равны 1, каждый следующий равен сумме двух предыдущих (1, 1, 2, 3, 5, 8, 13, ...). Найти первое число в последовательности Фибоначчи, большее n (значение n вводится с клавиатуры; $n > 1$).
8. Дано натуральное число. Определить, сколько раз в нем встречается минимальная цифра (например, для числа для числа 102 200 ответ равен 3, для числа 40 330 — 2, для числа 10 345 — 1).
9. Выяснить, является ли заданное число n членом арифметической прогрессии, первый член которой равен f, а шаг — s (n, f, s вводятся с клавиатуры).
10. Дано натуральное число, в котором все цифры различны. Определить, какая

11. цифра расположена в нем левее: максимальная или минимальная.
12. Известен факториал числа. Найти это число (факториал числа n равен 1 2 ... n).
13. В некоторой стране используются денежные купюры достоинством в 1, 2, 4, 8, 16, 32 и 64. Дано натуральное число n. Как наименьшим количеством таких денежных купюр можно выплатить сумму n (указать количество каждой из используемых для выплаты купюр)? Предполагается, что имеется достаточно большое количество купюр всех достоинств.
14. Старинная задача. Имеется 100 рублей. Сколько быков, коров и телят можно купить на все эти деньги, если плата за быка — 10 рублей, за корову — 5 рублей, за теленка — полтинник (0,5 рубля) и надо купить 100 голов скота?
15. Дано натуральное число n. Вычислить $1^1 + 2^2 + ... + n^n$.