**Программирование циклических алгоритмов**

**Как организовать цикл?**

Допустим, мы хотим вывести 5 раз на экран слово «при-вет». Можно, конечно, записать 5 одинаковых команд:

**print( "привет" )**

**print( "привет" )**

**print( "привет" )**

**print( "привет" )**

**print( "привет" )**

Но что если нужно будет сделать какие-то действия 1000 или 1000000 раз? В этом случае можно организовать цикл. Про-стейший цикл, нужный нам в этой задаче, мы хотели бы запи-сать так:

**# сделай 5 раз:**

**print( "привет" )**

К сожалению, в языке Python (как и во многих других язы-ках программирования) такого цикла нет3. Однако можно лег-ко запрограммировать те же действия немного по-другому. Да-вайте разберёмся, как можно организовать цикл в языке Python.

Вы знаете, что программа выполняется автоматически. И при этом в любой момент нужно знать, сколько раз уже выпол-нен цикл и сколько ещё осталось выполнить. Для этого необхо-димо использовать ячейку памяти (переменную). В ней можно, например, запоминать, сколько раз цикл уже был выполнен. Такую переменную целого типа часто называется *счётчиком*.

Сначала в переменную-счётчик записывают ноль (цикл ещё не выполнен ни разу), а после каждого повторения цикла уве-личивают значение счётчика на единицу:

**count = 0**

**while count < 5: # заголовок цикла**

**print( "привет" )**

**count += 1 # увеличение счётчика**

В этой программе используется новое служебное слово *while*, по-сле которого записано условие.

Поскольку цикл связан с повторением, циклические алго-ритмы называют *итерационными* (от лат. *iteratio* – повторе-ние). Каждое выполнение тела цикла называют *итерацией*.

Все операторы, которые выполняются в цикле (они называ-ются *телом цикла*) сдвигаются вправо на одинаковое число по-зиций, так же как и в условном операторе. Этот приём позволя-ет обойтись без операторных скобок, ограничивающих тело цик-ла в других языках программирования.

Нам нужно выполнять цикл 5 раз, то есть пока счётчик не станет равен 5. Об этом говорит *заголовок цикла*

**while count < 5:**

Его можно прочитать как «делай, пока *count* < 5».

После каждой итерации цикла переменная *count* увеличи-вается на 1 – цикл выполнен ещё один раз. Если программист забудет написать этот оператор, произойдёт *зацикливание*: про-грамма никогда не остановится, потому что условие *count* < 5 никогда не станет ложным.

Цикл можно построить и по-другому: сразу записать в счёт-чик нужное количество итераций, и после каждой итерации цикла *уменьшать* счётчик на 1. Тогда цикл должен закончить-ся при нулевом значении счётчика.

Этот вариант несколько лучше, чем предыдущий, посколь-ку счётчик сравнивается с нулём, а такое сравнение выполняет-ся в процессоре автоматически.

**Циклы по переменной**

Вернёмся снова к задаче, которую мы обсуждали в одном из параграфов – вывести на экран несколько раз слово «привет». Фактически нам нужно организовать цикл, в котором блок опе-раторов выполнится заданное число. Для этого можно приме-нить ещё один вид цикла – *цикл по переменной* (или *цикл с па-раметром*). На языке Python он записывается так:

**for i in range(10):**

**print( "привет" )**

Здесь слово *for* означает «для», переменная *i* (её называют *переменной цикла*) изменяется в диапазоне (**in range**) от 0 до 10, не включая 10 (то есть от 0 до 9 включительно). Таким обра-зом, цикл выполняется для *i* = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 – ровно 10 раз. Переменная *i* – это счётчик выполненных итераций цикла. Можно было записать этот цикл и по-другому:

**for i in [0,1,2,3,4,5,6,7,8,9]:**

**print( "привет" )**

В квадратных скобках через запятую перечислены все значения переменной, при которых выполняется цикл. Если их много, такой способ неудобен, лучше использовать встроенную функ-цию **range**.

Обратите внимание, что последовательность, которую стро-ит функция **range**, не бесконечна, то есть цикл с переменной всегда заканчивается, программа не может зациклиться.

Рассмотрим ещё один пример. В информатике важную роль играют степени числа 2 (2, 4, 8, 16 и т.д.). Давайте выведем на экран все степени двойки от 21 до 210. Для решения этой задачи мы можем написать программу, использующую цикл с услови-ем:

**power = 1**

**N = 2**

**while power <= 10 :**

**print( N )**

**N \*= 2**

**power += 1**

Вы наверняка заметили, что переменная **power** использу-ется трижды (см. блоки, выделенные фоном): в операторе при-сваивания начального значения, в условии выполнения цикла и в теле цикла (увеличение на 1).

Чтобы собрать все действия с переменной **power** в один опе-ратор, применим цикл по переменной. Нам нужно выполнить тело цикла при *power* = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. Чтобы получить такой набор значений, нужно вызвать функцию **range** с двумя аргументами: первый – это начальное значение (1), а второй – ограничитель, не входящий в последовательность (11):

**N = 2**

**for power in range(1,11):**

**print( N )**

**N \*= 2**

Запись цикла получилась проще, и поэтому меньше шан-сов сделать ошибку.

*!!!!!Запишите циклы, с помощью которых можно вывести на экран*

*целые числа от a до b (a b);*

*квадраты целых чисел от a до b (a b).*

Однако не любой цикл с условием может быть переписан как цикл с переменной. Если количество повторений цикла не-известно и не может быть найдено заранее (как в задаче с вы-числением суммы цифр числа), цикл по переменной использо-вать не удаётся.

С другой стороны, любой цикл по переменной можно заме-нить на равносильный цикл с условием: вместо вызова функции **range** придётся задать отдельно начальное значение перемен-ной цикла, условие продолжения цикла и правило изменения переменной цикла.

Рассмотрим ещё одну задачу – найдём сумму всех нату-ральных чисел от 1 до 1000. Для накопления суммы будем ис-пользовать переменную, которую назовём *summa*. В цикле дру-гая переменная (скажем, *i*) изменяется от 1 до 1000, и на каж-дом шаге этого цикла к сумме добавляется очередное значе-ние *i*:

**summa = 0**

**for i in range(1,1001):**

**summa += i**

*!!!Запишите циклы, с помощью которых можно вычислить*

*сумму целых чисел от a до b* (*a b*)*;*

*сумму квадратов целых чисел от a до b* (*a b*)*.*

**Шаг изменения переменной цикла**

По умолчанию функция **range** строит последовательность, в которой каждое следующее число на 1 больше предыдущего. Но это правило можно изменить, если при вызове функции **range** указать третий аргумент – шаг изменения переменной цикла. Следующая программа печатает квадраты натуральных чисел от 10 до 1 в порядке убывания:

**for k in range(10,0,-1):**

**print( k\*k )**

В этом примере шаг равен –1, то есть каждое следующее число на 1 меньше предыдущего. Заметим, что конечное значение 0 не входит в последовательность.

Пусть, например, нам нужно перебрать в цикле все значе-ния переменной *i* от 0 до 100, кратные пяти: 0, 5, 10, …, 100. Для этого нужно взять шаг изменения переменной 5:

**for i in range(0, 101, 5):**

**... # что-то делать с i**

Второй аргумент функции range равен 101 для того, чтобы последнее значение переменной *i* было равно 100. Значение-ограничитель должно быть больше, чем 100 (чтобы число 100 появилось в последовательности), но меньше, чем 106 (чтобы следующее число, 105, не появилось).