19. Циклы в языке Python.

Допустим, мы хотим вывести 5 раз на экран слово «привет». 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 100 или 10000 раз? В этом случае можно организовать цикл.

Цикл – это многократное выполнение одинаковых действий.

Выделяют **два вида циклов**:

- **цикл с параметром**, со счётчиком - цикл с известным числом повторений (сделать N раз)
- цикл **по условию** - с **неизвестным** числом повторений (делать, пока не надоест).

Вы знаете, что программа после запуска выполняется процессором автоматически. И при этом на каждом шаге нужно знать, сколько раз уже выполнен цикл и сколько ещё осталось выполнить. Для этого необходимо использовать ячейку памяти, в которой будет запоминаться количество выполненных шагов цикла – переменную. Такую переменную целого типа часто называют счётчиком шагов, переменной цикла, параметром цикла. Сначала можно записать в неё ноль (ни одного шага не сделано), а после каждого шага цикла увеличивать значение ячейки на единицу. Строка программы, содержащая проверку условия выполнения цикла, в том числе и переменную цикла, называется **заголовком цикла.** Все операторы, которые выполняются в цикле, называются **телом цикла.** При записи программы тело цикла сдвигается вправо на 4 позиции, так же как и в условном операторе. Этот приём позволяет показать, какие команды находятся внутри тела цикла и выполняются несколько раз, и обойтись без операторных скобок, ограничивающих тело цикла в других языках программирования. Поскольку цикл связан с повторением, циклические алгоритмы называют **итерационными** (от лат. iteratio – повторение). Каждое выполнение тела цикла называют **итерацией**.

Циклы с переменной (с параметром)

Если нужно организовать цикл, в котором тело цикла выполнится известное заданное число раз, можно применить цикл с параметром (со счётчиком, цикл с переменной). На языке Python он записывается так:

```
for <переменная цикла> in <значения>:      # заголовок цикла
. . . . <команды тела цикла>
```

for переводится как «для», in – «в». Всю конструкцию можно прочитать так: «для переменной цикла в указанном диапазоне значений делай команды тела цикла»

Внимание!

- Счётчик (параметр - переменная) принимает значения только из набора значений.
- Параметр - переменная не может изменяться в теле цикла.
- Тело цикла выполняется столько же раз, сколько значений в наборе.

Если в теле цикла только одна команда, то цикл можно записать в одну строку:

```
for <переменная цикла> in <значения>: <команда тела цикла>
```

Для *вывода на экран слова «привет» 10 раз* программа будет выглядеть так:

```
for i in range(10):  print( "привет" )
```

Здесь i – переменная цикла, которая последовательно принимает значения из диапазона, задающегося функцией **range().** В данном случае функция range() создает последовательность целых чисел от 0 до 10, не включая 10 (то есть от 0 до 9 включительно). Таким образом, цикл

выполняется для i = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 – ровно 10 раз. Переменная i – это счётчик выполненных итераций цикла.

Рассмотрим **варианты записи функции range().**

1) range(K) – создает числа от 0 до K-1 с шагом +1.
Например: range(5) [0, 1, 2, 3, 4]
2) range(N, K) - создает числа от N до K-1 с шагом +1.
Например: range(5, 11) [5, 6, 7, 8, 9, 10]
3) range(N, K, S) - создает числа от N до K-1 с шагом S.
Например: range(1, 15, 2) [1, 3, 5, 7, 9, 11, 13]
range(5, 1, -1) [5, 4, 3, 2]

Можно было записать эту программу с циклом for 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 функцию range().

Обратите внимание, что последовательность, которую создает 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.). Запишем *программу для вывода на экран всех степеней двойки от 2^1 до 2^{10}.*

```
for k in range(1,11):  print( 2 * * k )
```

Здесь переменная k последовательно принимает значения от 1 до 10 (ограничитель 11 не входит в последовательность) и при этом выводится значение выражения 2^k.

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 *выводит квадраты натуральных чисел от 10 до 1 в порядке убывания* (100, 81, 64, …, 4, 1):

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

В этом примере в качестве переменной цикла использовалась k, которая принимала значения 10, 9, 8, …, 1 (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 *вывести на экран все целые числа 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): print( 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).

В этом примере вывод чисел будет осуществляться в столбик. А если нужно в строчку через пробел? Тогда в функцию вывода добавим **параметр end=" "** . Если в кавычках поставить пробел, то вывод будет осуществляться через пробел. Если другой символ или их сочетание, то элементы вывода будут разделяться этими символами. Если в кавычках пусто, то элементы выведутся слитно.

```
for i in range(0, 101, 5): print( i, end=",    " )
```

В данном примере выведутся числа значения переменной i в строчку через запятую с пробелом; 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100,

Рассмотрим ещё одну задачу – *найдём сумму всех натуральных чисел от 1 до 10*.

```
s = 0
for i in range(1,11):
   s = s + i
print(‘ сумма =  ‘ , s)
```

Для накопления суммы используется переменная s. До начала цикла ей задаётся начальное значение 0. (изначально коробка для чисел пуста) Переменная цикла i перебирает последовательно все числа из заданного диапазона и изменяется от 1 до 10, и на каждом шаге в теле цикла к сумме (в коробку к тому, что там уже лежало), добавляется очередное значение i. Затем выводится накопленное значение переменной s.

Изменим эту программу так, чтобы она считала сумму целых чисел из произвольного диапазона от A до B, где A<B. Для этого прежде, чем считать, введём значения для A и B. Для ввода значения расположены в одной строке и разделены пробелом.

```
A, B = map(int, input('введите два числа, a<b   ').split())
s = 0
for i in range(A, B+1): s = s + i
print(' сумма =  ' , s)
```

В математике хорошо известная задача – вычисление n! .
n! читается как «**факториал числа n**». Факториал – это произведение целых чисел от 1 до n. То есть n!=1*2*3*4*…*n. Например, при n=5, 5!=1*2*3*4*5=120.
Программа для вычисления произведения целых чисел очень похожа на программу вычисления суммы чисел. Следует не забыть, что значение переменной n должно быть известно до входа в цикл, поэтому оно в самом начале программы задаётся присваиванием или вводится с клавиатуры командой n = int(input(‘введите значение n’)). Ещё надо учесть, что при умножении на ноль получается в ответе ноль, поэтому начальное значение переменной s для накопления произведения должно быть равно 1. (В коробочке изначально должна лежать 1, а не пустота). В главной формуле для вычисления знак + заменяется на *. Получается s = s * i . (Запишите программу самостоятельно)

Циклы по условию

Циклы по условию делятся на циклы с предусловием и постусловием.

Цикл, в котором проверка условия выполняется при входе (перед выполнением очередного шага) называется **циклом с предусловием**, то есть циклом с предварительной проверкой условия. Это можно сравнить с такой ситуацией: перед тем, как прыгнуть в бассейн, нужно проверить, есть ли в нём вода.

У циклов с предусловием есть два важных *свойства*:

- цикл не выполнится ни разу, если условие в самом начале ложно;
- пока условие в заголовке цикла истинно, тело цикла выполняется, но как только условие станет ложным, его работа заканчивается.

Цикл с предусловием на языке Python записывается так:

пока

```
while  <условие истинно>:
. . . .  <делай команды>
. . . .  <измени параметр условия>
```

Вернёмся к задаче, которую мы решали в начале параграфа – *вывести на экран 10 раз слово «привет»*. Используем при этом цикл с условием, который выполняется до тех пор, пока некоторое условие истинно.

счётчик = 0 пока счётчик < 10: …. выводи("привет") …. увеличь счётчик на 1	i = 0 while i < 10: # заголовок цикла …. print("привет") # тело цикла …. i = i + 1 # тело цикла

После каждой итерации цикла переменная i увеличивается на 1 – цикл выполнен ещё один раз. Если программист забудет написать этот оператор, произойдёт зацикливание: программа никогда не остановится, потому что условие i < 10 никогда не станет ложным.

Цикл можно построить и по-другому: сразу записать в счётчик нужное количество итераций, и после каждой итерации цикла уменьшать счётчик на 1. Тогда цикл должен закончиться при нулевом значении счётчика. Этот вариант несколько лучше, чем предыдущий, поскольку счётчик сравнивается с нулём, а такое сравнение выполняется в процессоре автоматически.

```
i = 10
while i != 0:
….  print( "привет" )
….  i = i - 1
```

Рассмотрим ещё один пример. *Выведем в столбик на экран все степени двойки от 2^1 до 2^n*, используя в программе цикл с предусловием while:

```
n = int(input('введите целое число     ' ))
k = 1
while  k <= n :
…. print( 2 * * k )
…. k += 1
```

А теперь *выведем все чётные числа от 0 до N*

```
N = int(input('введите целое число >0    ' ))
i = 1
while i <= N:
…. print( i, end=" ")
…. i = i + 2
```

Практически любой цикл с переменной можно заменить на равносильный ему цикл с условием: вместо вызова функции range() придётся задать отдельно начальное значение переменной цикла, условие продолжения цикла и правило изменения переменной цикла. Однако не любой цикл с условием может быть переписан как цикл с переменной. Если количество повторений цикла неизвестно и не может быть найдено заранее, цикл по переменной использовать не удаётся.

Рассмотрим ещё одну задачу, которая решается с помощью цикла с условием. Требуется *ввести с клавиатуры натуральное число и найти сумму и количество цифр его десятичной записи*. Например, если ввели число 123, программа должна вывести сумму 1+2+3 = 6 и количество = 3.

Сначала составим алгоритм решения это задачи.

Предположим, что число записано в переменной N. Нам нужно как-то разбить число на отдельные цифры. Вспомним, что остаток от деления числа на 10 равен последней цифре его десятичной записи. Запишем эту цифру в переменную d: d = N % 10

Сумму цифр будем хранить в целой переменной s. В самом начале, пока ни одну цифру ещё не обработали, значение этой переменной равно нулю: s = 0

Для того чтобы добавить к предыдущей сумме новую цифру, нужно заменить значение переменной s на s + d, то есть выполнить присваивание s = s + d

Количество цифр будем хранить в целой переменной k. В самом начале, пока ни одну цифру ещё не обработали, значение этой переменной равно нулю: k = 0.

Для того чтобы учесть новую цифру, нужно увеличить значение переменной k на 1, то есть выполнить присваивание k = k + 1 .

Для того чтобы затем отсечь последнюю цифру числа N, разделим N на 10 (основание системы счисления): N = N // 10

Эти три операции – выделение последней цифры числа, увеличение суммы и отсечение последней цифры – нужно выполнять несколько раз, пока все цифры не будут обработаны (и отсечены!) и в переменной N не останется ноль.

Изменение переменной n и счётчика для начального значения 123 можно записать в виде таблицы

n	Цифра d	Сумма s	Количество k
123	-	0	0
123	3	0+3	1
12	2	0+3+2	2
1	1	0+3+2+1	3
0		6	3

Ввести значение N счётчики = 0 пока n ≠ 0 …. записать последнюю цифру в d …. добавить цифру из d к сумме …. отсечь последнюю цифру от N …. увеличить счётчик цифр на 1 вывести сумму и количество	N = int(input("Введите целое число: ")) s = k = 0 while N != 0: …. d = N % 10 …. s = s + d …. N = N // 10 …. k = k+1 print(" В числе ", k, "цифр, их сумма =" , s)

В отличие от предыдущих примеров, здесь количество шагов цикла заранее неизвестно, оно определяется количеством цифр введённого числа.

Докажем, что эта программа не зациклится, то есть не будет работать бесконечно. Цикл завершается, когда переменная N становится равна нулю, поэтому нужно доказать, что это обязательно случится. По условию заданное число – натуральное, на каждом шаге цикла оно делится на 10 и остаток отбрасывается, поэтому значение переменной N каждый раз уменьшается в 10 раз. В результате после очередного уменьшения оно обязательно станет равно нулю.

Удалив строку программы с изменением значения переменной цикла, мы получим **бесконечный цикл или зацикливание**, то есть цикл, который никогда не заканчивается, а это плохо. Бесконечные циклы называют логическими ошибками, которых лучше избегать. Но это не значит, что в таком случае нельзя будет «вырваться» из цикла. Можно, если предусмотрительно использовать, например, **break**:

```
i = 1
while i > 0:
   print( i )
   if  i == 5: break
   i += 1
```

Оператора **break** (в переводе с англ. – «прервать») здесь организует остановку работы цикла, досрочный выход из цикла.

В данной программе мы использовали конструкцию ветвления с оператором if внутри цикла (в теле цикла) по условию while.

Во многих языках программирования существует **цикл с постусловием (цикл с известным условием окончания работы)**, в котором условие проверяется после завершения очередного шага цикла. Это полезно в том случае, когда нужно обязательно выполнить цикл хотя бы один раз.

У циклов с постусловием есть важное *свойство*: даже если условие изначально ложно, тело цикла выполнится один раз.

В языке Python нет цикла с постусловием, но его можно организовать с помощью цикла while.

```
while  True:
 ….  <операторы>
 ….  if <условие>:  break
```

Например, пользователь должен ввести с клавиатуры положительное число, и программист защищает программу от неверных входных данных.

```
while True:
   print ( "Введите положительное число:  " )
   n = int ( input() )
   if n > 0: break
```

Цикл, который начинается с заголовка **while True** будет выполняться бесконечно, потому что условие True всегда истинно. Выйти из такого цикла можно только с помощью оператора break. В данном случае он сработает тогда, когда станет истинным условие n > 0, то есть тогда, когда пользователь введет допустимое значение. В данной программе мы также использовали конструкцию краткого ветвления с оператором if внутри цикла (в теле цикла) по условию while.

Напишем программу, в которой осуществляется *ввод целых чисел до тех пор, пока не будет введён ноль, и подсчёт количества введённых положительных и отрицательных чисел*. Так как здесь в явном виде задано условие окончания работы, то воспользуемся конструкцией while True:

```
k1  =  k2  =  0
while  True:
   n  =  int( input( ' Введите целое число:   '))
   if n  >  0 :     k1  +=  1
   if n  <  0 :     k2  +=  1
   if n  ==  0 :   break
print  ( ' Введено :  ' )
print  ( ' положительных чисел - ', k1 )
print  ( ' отрицательных чисел - ', k2 )
```

Имеющееся условие окончания работы можно преобразовать в условие продолжения работы - работа продолжается, пока n не равно 0, значит, мы можем воспользоваться оператором while с таким условием:

```
k1  =  k2  =  0
n  =  int( input( ' Введите целое число:   '))
while  n !=  0:
   if n  >  0 : k1 += 1
   if n  <  0 : k2 += 1
   n  =  int( input( ' Введите целое число:   '))
print  ( ' Введено :  ' )
print  ( ' положительных чисел - ', k1 )
print  ( ' отрицательных чисел - ', k2 )
```

Вложенные циклы

В более сложных задачах часто бывает так, что на каждом шаге цикла нужно выполнять обработку данных, которая также представляет собой циклический алгоритм. В этом случае получается конструкция «цикл в цикле» или «вложенный цикл».

В любом вложенном цикле переменная внутреннего цикла изменяется быстрее, чем переменная внешнего цикла. Рассмотрим, например, такой вложенный цикл:

```
for i in range(1,5):
    for k in range(1,5):
        print (i, k, end=", ")
```

i	1	2	3	4
k	1,2,3,4	1,2,3,4	1,2,3,4	1,2,3,4

1 1, 1 2, 1 3, 1 4, 2 1, 2 2, 2 3, 2 4, 3 1, 3 2, 3 3, 3 4, 4 1, 4 2, 4 3, 4 4

При i = 1, переменная k пробегает все свои значения от 1 до 4.

При i = 2, переменная снова пробегает все свои значения от 1 до 4.

И так далее.

Таким образом, команда тела цикла print (i, k, end=", ") сработает 4*4=16 раз и будет выведена в строку последовательность пар чисел 1 1, 1 2, 1 3, 1 4, 2 1, 2 2, 2 3, 2 4, 3 1, … .

Рассмотрим, ещё такой вложенный цикл:

```
for i in range(1,5):
    for k in range(1,i+1):
        print ( i, k, end=", ")
```

На первом шаге (при i=1) переменная k принимает единственное значение 1.

Далее, при i=2 переменная k принимает последовательно значения 1 и 2.

На следующем шаге при i=3 переменная k проходит значения 1, 2 и 3, и т.д.

Команда тела цикла print (i, k, end=", ") выведет строку пар чисел

1 1, 2 1, 2 2, 3 1, 3 2, 3 3, 4 1, 4 2, 4 3, 4 4,

Предположим, что нужно *найти все простые числа в интервале от 2 до 1000*.

Простейший (но не самый быстрый) алгоритм решения такой задачи на псевдокоде выглядит так:

```
for n in range(2,1001):
    if число n простое:
        print ( n )
```

Как же определить, что число простое? Как известно, простое число делится только на 1 и само на себя. Если число n не имеет делителей в диапазоне от 2 до n-1, то оно простое, а если хотя бы один делитель в этом интервале найден, то составное.

Чтобы проверить делимость числа n на некоторое число k, нужно взять остаток от деления n на k. Если этот остаток равен нулю, то n делится на k. Таким образом, программу можно записать так (здесь n, k и z – целочисленные переменные, z обозначает счётчик делителей):

```
for n in range(2,1001):
  z = 0
  for k in range(2, n):
      if n % k == 0: z += 1
  if z == 0: print ( n )
```

Циклы в других языках программирования

Циклы с параметром

Суммирование всех чисел от 1 до 10 на языках Паскаль и C++ выглядит так:

Паскаль	C++

var s, i : integer
s := 0;
for i:=1 to 10 do
  s := s + i;
	
	s = 0;
	for (i=1; i<=10; i++)
	  s += i;

В языке Паскаль и C++ переменная i изменяется в диапазоне от 1 до 10 включительно, каждое из этих значений добавляется к значению переменной s.

Циклы с условием

Фрагмент программы, реализующей Алгоритм Евклида на языках Паскаль и C++ записывается следующим образом:

Паскаль	C++

while (a <> 0) and (b <> 0) do
 if a > b then
  a := a mod b
 else
  b := b mod a;
	while (a != 0 && b != 0) {
	  if (a > b)
	  a = a % b;
	  else
	  b = b % a;
	 }

В языке Паскаль каждое условие в составе сложного условия нужно обязательно взять в скобки, это связано с другим порядком выполнения (приоритетом) логических операций.	В языке C++ тело цикла ограничивается фигурными скобками. Если в теле цикла записан всего один оператор, фигурные скобки можно не ставить.

Данный фрагмент программы содержит ветвление внутри цикла (ветвление является телом цикла).

Контрольные вопросы и задания

1. Что такое цикл? Какие виды циклов используются в Python?
2. Что называют заголовком цикла? Телом цикла? Итерацией?
3. Как в Python записывается цикл с переменной? Как переводятся служебные слова for , in?
4. В каком случае цикл с параметром можно записать в одну строку?
5. Какое действие выполняет функция range()?Что обозначено переменными A, B, C в записи range(A, B, C)?
6. Как можно перечислить значения переменной цикла в цикле for, не используя функцию range()? Приведите пример.
7. Напишите программу, которая выводит на экран кубы натуральных чисел от 10 до 1 в порядке убывания. *(2б)*
8. Напишите программу, которая выводит на экран в строчку через пробел все целые числа от 7 до 99, кратные семи. *(2б)*
9. Что такое факториал числа n? Запишите программу для вычисления факториала числа n, используя цикл с параметром. *(2б)*

10. Какой цикл называется циклом с предусловием? Как он записывается в Python? Как переводится слово while?
11. Каковы свойства цикла с предусловием?
12. Напишите программу с циклом while, которая выведет в столбик на экран все степени тройки от 3^1 до 3^n *(2б)*
13. Напишите программу с циклом while, которая выводит все нечётные числа от 0 до N. *(2б)*
14. Любой ли цикл с for можно заменить циклом с while? Любой ли цикл с while можно заменить циклом с for ?
15. Напишите программу с циклом while, которая для любого введённого натурального числа выводит количество и произведение его цифр. *(2б)*
16. Что такое зацикливание? Для чего используется оператор break и как переводится это слово?
17. Что такое цикл с постусловием? Каково его свойство?
18. Есть ли цикл с постусловием в языке Python? Как его организовать в Python?
19. Какова особенность цикла с заголовком while True?
20. Напишите программу, в которой осуществляется ввод целых чисел до тех пор, пока не будет введено отрицательное, и подсчёт количества введённых положительных чисел. *(2б)*
21. Дан фрагмент программы. Определите, сколько раз в нем выполнится команда print. Что будет выведено в результате? Запишите соответствующую строку.

```
for i in range(1,3):
    for k in range(1,3):
        print (i, k, end=", ")
```