Знакомство с циклом while

На данном занятии мы познакомимся с оператором цикла while. Цикл позволяет повторять организовать многократное повторение одних и тех же действий. Кроме того, мы сделаем акцент вот на чем: в одной и той же строчке программы на разных итерациях цикла переменные могут иметь разное значение.

Цикл *while*

Он выполняет блок кода, **пока истинно** какое-то условие.

Напомним, условный оператор *if* проверяет условие и, в зависимости от того, истинно оно или ложно, выполняет либо не выполняет следующий записанный с отступом блок. После этого программа в любом случае выполняется дальше (там еще может быть *elif* или *else*, но сути это не меняет).

Оператор *while* («пока») тоже проверяет условие и тоже, в случае его истинности, выполняет следующий блок кода (**тело цикла**). Однако после выполнения этого блока кода выполняется не то, что идет после него, а снова проверяется условие, записанное после *while*.

Ведь при выполнении тела цикла значения каких-то переменных могли измениться — в результате условие цикла может уже не быть истинным. Если условие все еще истинно, тело цикла выполняется снова. Как только условие цикла перестало выполняться (в том числе если оно с самого начала не было выполнено), программа идёт дальше — выполняются команды, записанные после тела цикла.

Условие цикла записывается как и для *if*: с помощью операций отношения (>, >=, <, <=, != , ==). Сложные условия можно составлять с помощью логических операций *not*, *and*, *or*.

Действия, расположенные в теле цикла (блок кода), записываются со смещением вправо на четыре пробела относительно начала слова *while*. Переменные, входящие в условие, должны на момент проверки условия цикла иметь значения.

```
while условие:
    блок кода (тело цикла)
```

Важно!

Один шаг цикла (выполнение тела цикла) еще называют **итерацией**.

Используйте цикл *while* всегда, когда какая-то часть кода должна выполниться несколько раз, причем невозможно заранее сказать, сколько именно.

Давайте посмотрим программу, в которой цикл будет выполняться, пока не введут число меньше 0:

```
number =int(input())
while number >0:
print('Вы ввели положительное число! Вводите дальше.')
    number =int(input())
print('Так-так, что тут у нас...')
print('Вы ввели отрицательное число или ноль. Всё.')
```

Разберемся, как будет работать эта программа.

Сначала выполняется первая строчка: number = int(input()) — пользователь вводит целое число. (Мы предполагаем, что пользователь действительно ввел число, и программа не вылетела с ошибкой.) Предположим, он ввел число 10. Оно записано в переменной *number*.

Выполняется вторая строчка: while number > 0: — «пока number > 0» — здесь проверяется, выполнено ли условие number > 0. Поскольку мы предположили, что *number* в этот момент равно 10, тогда условие выполнено, поэтому дальше выполняется блок, записанный с отступом, — тело цикла.

Третья строчка программы выводит на экран строку, тут все понятно.

Четвертая строчка вновь считывает с клавиатуры число и сохраняет его в переменную *number*. Пусть пользователь ввел 2.

Когда выполнение программы доходит до конца тела цикла, происходит возврат к заголовку цикла (второй строчке программы) и повторная проверка условия. Поскольку 2 > 0, снова выполняется тело цикла.

Третья строчка снова выводит на экран сообщение, четвертая строчка снова считывает число (пусть это будет число 3), пятая строчка снова выводит на экран сообщение...

Закончив тело цикла, опять проверяем условие в заголовке. *number* равно 3, 3 > 0, поэтому продолжаем.

Третья строчка опять выводит на экран сообщение, четвертая строчка опять считывает число. Пусть теперь это будет −1. Обратите внимание: переменная *number* на каждой итерации цикла приобретает новое значение! Пятая строчка опять выводит на экран сообщение...

Вновь вернувшись на вторую строчку, получаем, что −1 > 0 — ложно. Поэтому цикл завершается, тело цикла больше не выполняется, прыгаем сразу на следующую после цикла строчку программы — шестую. Она выводит последнее сообщение.

Все.

Составной оператор присваивания

Напомним, что в операторе присваивания одно и то же имя переменной может стоять и справа (в составе какого-то выражения), и слева. В этом случае сначала вычисляется правая часть со старым значением переменной, после чего результат становится новым значением этой переменной. Ни в коем случае не воспринимайте такой оператор присваивания как уравнение!

```
number =int(input())# например, 5
number = number +1# тогда здесь number становится равным 6
print(number)
```

Важно!

Для конструкций вида number = number + 1 существует и сокращенная форма записи оператора присваивания: number += 1. Аналогично оператор x = x + y можно записать как x += y, оператор x = x * y — как x *= y, и так для любого из семи арифметических действий.

Сигнал остановки

Рассмотрим такую задачу: пользователь вводит числа. Пусть это будут цены на купленные в магазине товары, а наша программа — часть программного обеспечения кассового аппарата. Ввод −1 — сигнал остановки. Нужно сосчитать сумму всех введенных чисел (сумму чека).

Поскольку требуется повторить нечто (ввод очередной цены) неизвестное количество раз, потребуется цикл *while*. Нам понадобится как минимум две переменные: *price* для цены очередного товара и *total* — для общей суммы.

Если бы мы знали точно, что пользователю надо купить ровно три товара, цикл (и ввод −1 как условие его прерывания) был бы не нужен. Тогда программа могла бы выглядеть так:

```
total =0
price =float(input())
total = total + price
price =float(input())
total = total + price
price =float(input())
total = total + price
print('Суммавведённыхчиселравна', total)
```

Обратите внимание: мы назвали переменные осмысленно. Это очень облегчит жизнь программисту, который будет читать наш код позже, даже если это будете вы сами неделю спустя. Однако интерпретатор Python к этому факту совершенно равнодушен. Чтобы значения переменных соответствовали названиям и тому смыслу, который мы в них закладываем, нужно поддерживать переменные в актуальном состоянии. И только вы, программист, можете это сделать.

С переменной *price* все относительно понятно: ее значение обновляется при считывании с клавиатуры на каждой итерации цикла, как это делалось во многих других задачах. *total* сначала равно нулю: до начала ввода цен их сумма, конечно, ноль. Однако значение переменной *total* устаревает каждый раз, когда пользователь вводит цену очередного товара.

Поэтому нам нужно прибавить к значению *total* только что введенную цену, чтобы эта переменная по-прежнему обозначала сумму цен всех купленных товаров.

Если бы мы хотели сократить запись, можно было бы организовать цикл, который выполнился бы ровно три раза. Для этого нам потребуется переменная-счетчик, которая внутри цикла будет считать каждую итерацию цикла. А условием выхода обозначим выполнение нужного количества итераций:

```
count =0
total =0
while count <3:
    price =float(input())
    total = total + price
    count = count +1
print('Сумма введённых чисел равна', total)
```

Обратите внимание: *total* и *count* должны обнуляться до цикла.

Однако у нас в задаче количество товаров неизвестно, поэтому понадобится цикл до ввода сигнала остановки (−1). С учетом сказанного выше программа будет выглядеть так:

```
total =0
print('Вводите цены; для остановки введите -1.')
price =float(input())
while price >0:
total = total + price  # можно заменить на аналогичную запись
# total += price
    price =float(input())
print('Общая стоимость равна', total)
```

Подсчет количества элементов, удовлетворяющих условию

Рассмотрим еще одну задачу.

Пользователь вводит целые числа. Ввод чисел прекращается, если введено число 0. Необходимо определить, сколько чисел среди введенных оканчивались на 2 и были кратны числу 4. Теперь нам надо проверять последовательность чисел.

Для каждого введенного числа надо делать проверку, соответствует ли оно условию. Если оно подходит под условие, увеличиваем счетчик таких чисел.

И уже после цикла, когда остановился ввод чисел, выводим результат — посчитанное количество нужных чисел.

```
count =0
number =int(input())
while number !=0:
if number %10==2 and number %4==0:
        count +=1
number =int(input())
print('Количество искомых чисел:', count)
```

Обратите внимание: до цикла необходимо задать начальное значение для переменной *count*. Ведь когда придет первое подходящее под условие число, у нас *count* будет увеличиваться на 1 относительно предыдущего значения. А значит, это значение должно быть задано.

Давайте посмотрим, как будет работать эта программа для последовательности чисел: 12, 3, 32, 14, 0.

Шаг	Действие	Пояснение	Цикл
1	`count = 0`	`count = 0`	
2	`number = int(input())`	`number = 12`	
3	`while number != 0:`	`12 != 0` (Истина)	Вход в цикл, 1-я итерация
4	`if number % 10 == 2 and number % 4 == 0:`	`12 % 10 == 2 and 12 % 4 == 0` (Истина)	Заходим в if
5	`count += 1`	`count = 1`	
6	`number = int(input())`	`number = 3`	
7	`while number != 0:`	`3 != 0` (Истина)	2-я итерация
8	`if number % 10 == 2 and number % 4 == 0:`	`3 % 10 == 2 and 3 % 4 == 0` (Ложь)	Пропускаем if
9	`number = int(input())`	`number = 32`	
10	`while number != 0:`	`32 != 0` (Истина)	3-я итерация
11	`if number % 10 == 2 and number % 4 == 0:`	`32 % 10 == 2 and 32 % 4 == 0` (Истина)	Заходим в if
12	`count += 1`	`count = 2`	
13	`number = int(input())`	`number = 14`	
14	`while number != 0:`	`14 != 0` (Истина)	4-я итерация
15	`if number % 10 == 2 and number % 4 == 0:`	`14 % 10 == 2 and 14 % 4 == 0` (Ложь)	Пропускаем if
16	`number = int(input())`	`number = 0`	
17	`while number != 0:`	`0 != 0` (Ложь)	Выход из цикла
18	`print('Количество искомых чисел:', count)`	Вывод вычисленного *count*	

Поиск максимума и минимума

Очень часто в задачах приходится использовать различные статистические алгоритмы: поиск максимума, минимума, среднего значения, медианы и моды чисел, главный из которых — определение максимального и минимального значений на множестве данных.

Рассмотрим алгоритм в общем виде.

1. Заведем отдельную переменную для хранения максимума и минимума. В качестве начального значения можно задать:

- Заведомо малое для анализируемых данных значения, для максимума это будет совсем маленькое число: например, если мы вычисляем максимальный балл за экзамен, можно взять maximum = 0, тогда гарантированно произойдет замена максимума. Минимуму же, наоборот, присваивается заведомо большое значение
- Первый элемент данных

2. В теле цикла каждый подходящий элемент данных обрабатывается операторами по принципу:

- Если текущий элемент больше максимума, меняем максимум
- Если текущий элемент меньше минимума, заменяем минимум

Пример 1. Витя анализировал список литературы и решил, что хочет начать с самой большой по объему книги. Напишем программу, которая поможет мальчику определить, сколько страниц ему предстоит прочитать. Витя последовательно вводит количество страниц каждой книги из списка, а окончанием ввода служит ввод любого отрицательного числа.

```
biggest_book =0
n =int(input())
while n >0:
if n > biggest_book:
      biggest_book = n
   n =int(input())
print(biggest_book)
```

Так как книга не может содержать в себе 0 страниц, для значения максимума мы можем взять 0.

После этого Витя начинает вводить количество страниц: например, он вводит 148. 148 > 0 — условие цикла выполняется, и мы переходим к операции сравнения. На данном шаге 148 > 0, значит, biggest_book = 148. Снова считываем число.

Предположим теперь введено 120. 120 > 0 — продолжаем работать в цикле. 120 > 148 — условие не выполняется, переходим к вводу новых данных, *biggest_book* все еще равен 148.

В этот раз мальчик ввел 486, мы заходим в цикл 486 > 148, производим замену biggest_book = 486. Продолжаем ввод. И так далее до тех пор пока не будет введено отрицательное число.

Пример 2. password123

Как известно, когда мы придумываем пароль от аккаунта ВКонтакте, электронной почты или Яндекс. Контеста, к этому паролю часто предъявляются определённые требования по сложности.

Напишите программу, которая имитирует проверку пароля, придуманного пользователем. Пользователь вводит пароль, потом ещё раз его же, для подтверждения.

- если пароль, который ввёл пользователь (в первый раз) короче 8 символов, программа выводит "Короткий!" и завершает свою работу
- если пароль достаточно длинный, но введённый во второй раз пароль не совпадает с первым, программа выводит "Различаются."
- если же и эта проверка пройдена успешно, программа выводит "OK" (латинскими буквами).

Формат ввода

Две строки — первый и второй пароль, введенные пользователем.

Формат вывода

Одна строка — результат проверки пароля.

Пример 1

Ввод	Вывод
пароль пароль	Короткий!

Пример 2

Ввод	Вывод
пароль123 пароль123	OK

```
p1 = input()
p2 = input()
if len(p1) < 8:
    print('Короткий!')
elif p1 != p2:
    print('Различаются.')
else:
    print('OK')
```

Пример 3. Скидки

В магазине акция: скидка 5% на товары, цена которых превышает 1000 рублей. Напишите программу, отчасти имитирующую работу кассового аппарата: вводятся цены покупаемых товаров, нужно вывести общую стоимость товаров с учётом скидки.

Формат ввода

Несколько действительных чисел — цены на товары. Каждое число записано в отдельной строке.
Последнее число — отрицательное — сигнал остановки.

Формат вывода

Одно действительное число — общая стоимость товаров с учётом скидки.

Пример

Ввод	Вывод
25 2000 370.35 -1	2295.35

```
x = float(input())
s = 0
while x > 0:
    if x > 1000:
        x = x - x * 0.05
    s = s + x
    x = float(input())

print(s)
```

Пример 4. Учитель

Когда Учитель достиг просветления, он понял, что должен раздать свои богатства, причём сделать это следующим образом: в первый день разделить все свои золотые монеты на 8 равных частей (счастливое число!), излишки (если таковые будут иметься) пожертвовать храму Будды, оставить себе одну восьмую часть, остальные раздать бедным.
Во второй день вновь разделить оставшиеся монеты на 8 частей и повторить вышеуказанные манипуляции. И продолжать так до тех пор, пока у него не останется так мало монет, что при делении их на 8 равных частей они все окажутся излишком.

Оставшиеся монеты можно оставить себе. Кроме того, Учитель не тратит свои деньги (во всяком случае, в дни после просветления): его кормят ученики, а в быту он аскетичен.

Хотя Учитель знает, конечно, сколько у него золотых монет изначально, но он не может сообразить, сколько монет окажется в конце — всё-таки он Учитель духовных практик, а не математики или программирования. Помогите ему.

Иными словами, найдите первую цифру введённого числа при записи его в системе счисления с основанием 8.

Формат ввода

Одно целое число — изначальное количество монет у Учителя.

Формат вывода

Одно целое число — количество монет, которое останется у Учителя в конце.

Пример

Ввод	Вывод
129	2

```
x = int(input())
while x >= 8:
    x = x // 8
print(x)
```

Пример 5. Таких берут в космонавты

После полета Юрия Гагарина в 1961 практически каждый мальчик СССР хотел стать космонавтом. Прошло уже более полувека, но профессия космонавт все так же престижна. К сожалению, не каждый желающий может пройти отбор, существуют высокие требования к уровню подготовки будущих космонавтов, а также ограничения по антропометрическим показателям. Например, рост космонавта не может быть больше 190 см и меньше 150 см.

Напишите программу, которая считывает рост претендентов в отряд космонавтов до тех пор, пока не будет введен «!». А затем выводит на первой строчке количество подходящих кандидатур, а на второй строке – минимальный и максимальный рост участников, отобранных в новый отряд космонавтов.

Гарантируется, что в отряд отберутся как минимум два летчика-космонавта.

Формат ввода

Несколько строк с ростом космонавтов и последняя строка «!».

Формат вывода

Две строки: количество кандидатур на первой, и минимальный и максимальный рост через пробел – на второй.

Пример

Ввод	Вывод
192 189 145 162 172 !	3 162 189

```
x = input()
min = 190
max = 150
k = 0
while x != ("!"):
    if int(x) <= 190 and int(x) >= 150:
        k = k + 1
        if int(x) < min:
            min = int(x)
        if int(x) > max:
            max = int(x)
    x = input()
print(k)
print(min, max)
```

Пример 6. 1024 и все-все-все

Во многих задачах, связанных с компьютерами, особенно близких к аппаратной части, важную роль играют числа, являющиеся степенями двойки: 1, 2, 4, 8 и так далее. Напишите программу, которая проверяет, является ли введённое натуральное число степенью двойки. Если да, то выводится сама эта степень; если нет, выводится «НЕТ».

Формат ввода

Одно целое число.

Формат вывода

Одно целое число (показатель степени) или строка «НЕТ».

Пример

Ввод	Вывод
302231454903657293676544	78

```
n = int(input())
i = 1
k = 0
while i < n:
    i = i * 2
    k = k + 1
if i == n:
    print(k)
else:
    print('НЕТ')

```

Пример 7. Ищем клад — 1

Мы находимся на острове, на котором закопан клад. Мы находимся в точке с координатами (0, 0) и смотрим на север. Нам известно, где закопан клад, но этого мало: остров полон опасностей, и нужно перемещаться строго по указаниям карты, которая, к счастью, тоже имеется в нашем распоряжении. Мы хотим найти клад как можно скорее.

Известны координаты клада и указания, которым нужно следовать, чтобы его найти. Каждое указание карты состоит из одного слова и, возможно, одного натурального числа. Слово — одно из набора: «вперёд», «налево», «направо», «разворот» или «стоп».

После слова «вперёд» следует количество шагов, которое следует пройти в том направлении, куда мы в данный момент смотрим. Слова «налево» или «направо» означают, что нужно изменить направление взгляда под прямым углом, «разворот» — что прямо на обратное. Команда «стоп» означает остановку.

Найдите минимальное количество указаний карты, которое нужно выполнить, чтобы прийти к кладу.

Формат ввода

Сначала вводятся два числа на отдельных строчках: координаты клада по оси икс (запад-восток) и игрек (юг-север).

Затем следует некоторое количество указаний карты. Каждое указание карты состоит из одного слова и, возможно, одного натурального числа на отдельной строке. Слово — одно из набора: «вперёд», «налево», «направо», «разворот» или «стоп».

Формат вывода

Программа выводит на отдельных строках минимальное количество указаний карты, которое нужно выполнить, чтобы прийти к кладу, и направление взгляда в этот момент (одно из: «север», «юг», «запад», «восток»). Гарантируется, что карта приводит к кладу.

Мы находимся на острове, на котором закопан клад. Мы находимся в точке с координатами (0, 0) и смотрим на север. Нам известно, где закопан клад, но этого мало: остров полон опасностей, и нужно перемещаться строго по указаниям карты, которая, к счастью, тоже имеется в нашем распоряжении. Мы хотим найти клад как можно скорее.

Известны координаты клада и указания, которым нужно следовать, чтобы его найти. Каждое указание карты состоит из одного слова и, возможно, одного натурального числа. Слово — одно из набора: «вперёд», «налево», «направо», «разворот» или «стоп».

После слова «вперёд» следует количество шагов, которое следует пройти в том направлении, куда мы в данный момент смотрим. Слова «налево» или «направо» означают, что нужно изменить направление взгляда под прямым углом, «разворот» — что прямо на обратное. Команда «стоп» означает остановку.

Найдите минимальное количество указаний карты, которое нужно выполнить, чтобы прийти к кладу.

Формат ввода

Сначала вводятся два числа на отдельных строчках: координаты клада по оси икс (запад-восток) и игрек (юг-север).

Затем следует некоторое количество указаний карты. Каждое указание карты состоит из одного слова и, возможно, одного натурального числа на отдельной строке. Слово — одно из набора: «вперёд», «налево», «направо», «разворот» или «стоп».

Формат вывода

Программа выводит на отдельных строках минимальное количество указаний карты, которое нужно выполнить, чтобы прийти к кладу, и направление взгляда в этот момент (одно из: «север», «юг», «запад», «восток»). Гарантируется, что карта приводит к кладу.

Пример 1

Ввод	Вывод
-2 9 вперёд 9 налево вперёд 2 разворот вперёд 17 стоп	3 запад

Пример 2

Ввод	Вывод
0 1 вперёд 2 разворот вперёд 1	3 юг

```
try:
  the_minimum_number_of_instructions =0
  x =int(input())
  y =int(input())
```

```
x1 =0
y1 =0
move ='север'
direction_of_movement =input()
            while direction_of_movement !='стоп'or
direction_of_movement !=' ':
    ifint(x)== x1 andint(y)== y1:
        print(the_minimum_number_of_instructions)
        print(move)
        break
    else:
        the_minimum_number_of_instructions +=1
        if direction_of_movement =='вперёд':
            steps =int(input())
            if move =='север':
                y1 += steps
            elif move =='запад':
                x1 -= steps
            elif move =='юг':
                y1 -= steps
            elif move =='восток':
                x1 += steps
        elif direction_of_movement =='направо':
            if move =='север':
                move ='восток'
            elif move =='восток':
                move ='юг'
            elif move =='юг':
                move ='запад'
            elif move =='запад':
                move ='север'
        elif direction_of_movement =='налево':
            if move =='север':
                move ='запад'
            elif move =='запад':
                move ='юг'
            elif move =='юг':
                move ='восток'
            elif move =='восток':
                move ='север'
        elif direction_of_movement =='разворот':
            if move =='север':
                move ='юг'
            elif move =='юг':
                move ='север'
            elif move =='запад':
                move ='восток'
            elif move =='восток':
                move ='запад'
        direction_of_movement =input()
else:
    if x ==0and y ==0:
        print(0)
        print('север')
```

```
except EOFError:
    print(the_minimum_number_of_instructions)
    print(move)
```

Другой вариант:

```
x = int(input())
y = int(input())
z = input()
x1 = 0
y1 = 0
napr = 'север'
k = 0
while z != 'стоп':
    if x != x1 or y != y1:
        if z == "вперед":
            h = int(input())
            if napr == 'север':
                y1 = y1 + h
            if napr == 'юг':
                y1 = y1 - h
            if napr == 'восток':
                x1 = x1 + h
            if napr == 'запад':
                x1 = x1 - h
        if z == "разворот":
            if napr == 'север':
                napr = 'юг'
            elif napr == 'юг':
                napr = 'север'
            elif napr == 'восток':
                napr = 'запад'
            elif napr == 'запад':
                napr = 'восток'
        if z == "налево":
            if napr == 'север':
                napr = 'запад'
            elif napr == 'юг':
                napr = 'восток'
            elif napr == 'восток':
                napr = 'север'
            elif napr == 'запад':
                napr = 'юг'
        if z == "направо":
            if napr == 'север':
                napr = 'восток'
            elif napr == 'юг':
                napr = 'запад'
            elif napr == 'восток':
                napr = 'юг'
            elif napr == 'запад':
                napr = 'север'
        k = k + 1
    z = input()
print(k)
print(napr)
```

Задание на практику:

1. Дано положительное число N. Вывести все числа от 0 до N с помощью цикла while.
2. Дано положительное число N. Вывести все числа от N до 0 с помощью цикла while.
3. Вывести все нечетные числа от 1 до 30.
4. Пользователь вводит числа, пока он не введет ноль.

True и false, break и continue

Данное занятие посвящено условиям выхода из циклов. Рассматривается булев тип, даются задачи на использование флагов. Затем рассматриваются операторы break и continue, позволяющие в некоторых случаях избавиться от флагов.

Логический тип данных

Если *a* и *b* — числа (допустим, действительные), то у выражения a + b есть какое-то значение (зависящее от значений *a* и *b*) и тип — тоже действительное число. Как вы думаете, можно ли сказать, что у выражения a == b есть значение и тип? Или это просто конструкция, которая всегда должна стоять в условии *if* или *while*?

Логический тип

На самом деле такое выражение имеет и тип под названием **bool**, и значение: *True* (истина) или *False* (ложь). По-русски bool — это булев тип, или булево значение (в честь математика Джорджа Буля), иногда его еще называют логический тип.

Логический тип может иметь только два значения, а над переменными логического типа можно выполнять логические операции *not*, *and*, *or*.

Также для приведения к логическому типу можно использовать функцию *bool*, которая для ненулевого значения вернет истину.

```
k =True
print(k)# выведет True
print(not k)# выведет False
k =5>2
print(k)# выведет True
k =bool(0)
print(k)# выведет False
k =bool("")
print(k)# выведет False
k =bool(13)
print(k)# выведет True, т.к. число не 0
k =bool("q")
print(k)# выведет True, т.к. строка не пустая
k =bool("False")
print(k)# выведет True, т.к. строка не пустая
```

Или вот еще пример:

```
ifTrue:
print('Эта строка будет выведена на экран.')
else:
print('Эта строка никогда не будет выведена на экран.')
print(2*2==4)# выведет True
a =input()
b =input()
# Теперь переменная equal равна True, если строки a и b равны,
# и False в противном случае
equal=(a == b)
if equal andlen(a)<6:
print('Вы ввели два коротких одинаковых слова.')
```

Использование флагов

Обычно переменные с булевым значением используются в качестве флагов.

Флаг

Изначально флаг устанавливается в False, потом программа как-то работает, а при наступлении определенного события флаг устанавливается в True. После идет проверка, поднят ли флаг. В зависимости от ее результата выполняется то или иное действие. Иными словами, флаг — это переменная с булевым значением, которая показывает, наступило ли некое событие.

В примере ниже (эта программа — терапевтический тренажер для избавления физиков-экспериментаторов от синхрофазотронозависимости) имеется флаг *said_forbidden_word*, который означает «сказал ли пользователь запретное слово „синхрофазотрон“». Флаг равен True, если сказал, и False, если нет.

В самом начале пользователь еще ничего не успел сказать, поэтому флаг установлен в False. Далее на каждой итерации цикла, если пользователь сказал запретное слово, флаг устанавливается в True и остается в таком состоянии (при необходимости флаг можно и «опустить»). Как только флаг оказывается равен True, поведение программы меняется: перед каждым вводом выдается предупреждение, а в конце выдается другое сообщение.

Важно!Переменным-флагам особенно важно давать осмысленные имена (обычно — утверждения вроде *said_forbidden_word*, *found_value*, *mission_accomplished*, *mission_failed*), ведь флагов в программе бывает много.

```
forbidden_word='синхрофазотрон'
# можно было использовать и sep='', чтобы кавычки не отклеились от слова
print('Введите десять слов, но постарайтесь случайно не ввести слово "'+
forbidden_word+'"!')
said_forbidden_word=False
foriinrange(10):
ifsaid_forbidden_word:
print('Напоминаем, будьте осторожнее, не введите снова слово "'+
forbidden_word+'"!')
    word =input()
if word ==forbidden_word:
said_forbidden_word=True
# вместо предыдущих двух строк также можно написать:
# said_forbidden_word = (said_forbidden_word or word == forbidden_word)
ifsaid_forbidden_word:
print('Вынарушилиинструкции.')
else:
print('Спасибо, что ни разу не упомянули',forbidden_word)
```

Операторы break и continue. Бесконечные циклы

Если нужно прекратить работу цикла, как только случится некое событие, то, кроме флага, есть и другой способ — оператор разрыва цикла **break** (он работает и для цикла *for*). Это не функция и не заголовок блока, а оператор, который состоит из одного слова. Он немедленно прерывает выполнение цикла *for* или *while*.

```
for i inrange(10):
print('Итерация номер', i,'начинается...')
if i ==3:
print('Ха! Внезапный выход из цикла!')
break
print('Итерация номер', i,'успешно завершена.')
print('Цикл завершён.')
```

В частности, нередко встречается такая конструкция: цикл, выход из которого происходит не по записанному в заголовке цикла условию (это условие делается всегда истинным — как правило, просто True), а по оператору *break*, который уже заключен в какой-то условный оператор:

```
whileTrue:
    word =input()
if word =='стоп':
break
print('Выввели:', word)
print('Конец.')
```

Важно!Впрочем злоупотреблять этой конструкцией и вообще оператором *break* не стоит. Когда программист читает ваш код, он обычно предполагает, что после окончания цикла *while* условие в заголовке этого цикла ложно. Если же из цикла можно выйти по команде *break*, то это уже не так. Логика кода становится менее ясной.

Оператор **continue** немедленно завершает текущую итерацию цикла и переходит к следующей.

```
foriinrange(10):
print('Итерацияномер',i,'начинается...')
if i ==3:
print('...но её окончание таинственно пропадает.')
continue
print('Итерация номер', i,'успешно завершена.')
print('Цикл завершён.')
```

Рассмотрим еще один пример:

```
count =1
while count <100:
if count %5==0:
continue
print(count)
count+=1
```

Что будет напечатано в процессе выполнения программы?

Предполагается, что программа выведет все числа от 1 до 100, не кратные 5. Но на самом деле, если вы запустите программу в режиме трассировки, на экран выведется 1 2 3 4, а потом программа уйдет в бесконечный цикл. Почему это происходит?

Когда переменная *count* станет равна 5, записанное в операторе *if* условие станет истинным и выполнится оператор *continue*. Т. е. мы немедленно перейдем к следующей итерации цикла, пропуская вывод числа и увеличение счетчика *count*.

Переменная *count* так и не увеличится и по-прежнему останется со значением 5. Значит, условие в *if* будет все так же равно True, и цикл станет бесконечным.

Иными словами, часто использовать *break* и *continue* не рекомендуют, поскольку они приводят к произвольному перемещению точки выполнения программы по всему коду, что усложняет понимание и следование логике. Тем не менее разумное использование этих операторов может улучшить читабельность циклов в программе, уменьшив при этом количество вложенных блоков и необходимость в сложной логике выполнения цикла.

Например, рассмотрим следующую программу:

```
count=0
exitLoop=False
whilenotexitLoop:
print("Введите 'e' для выхода и любую другую клавишу для продолжения:")
sm=input()
ifsm=='e':
exitLoop=True
else:
```

```
        count +=1
print("Вызашливцикл ", count," раз(а)")
```

А теперь ту же самую программу напишем с использованием оператора *break*:

```
exitLoop=False
whilenotexitLoop:
print("Введите 'e' для выхода и любую другую клавишу для продолжения:")
sm=input()
ifsm=='e':
break
count+=1
print("Вы зашли в цикл ",count," раз(а)")
```

Чего нам удалось добиться? Во-первых, мы избежали использования как логической переменной, так и оператора *else*. Уменьшение количества используемых переменных и вложенных блоков улучшают читабельность и понимание кода больше, чем *break* или *continue* могут нанести вред.

Пример 1. 1984

Вы — сотрудник Министерства Правды в тоталитарной сверхдержаве Океании, которая то воюет, то заключает мир с двумя другими тоталитарными сверхдержавами, Евразией и Остазией. Ваша задача — информационное обеспечение войны в соответствии с указаниями правительства.

Изначально идёт война с Евразией, мир с Остазией.

Формат ввода

На первой строке указывается натуральное число N — количество команд от правительства.

Далее следует N команд. Каждая команда вводится на отдельной строке и представляет собой одну из трёх строк:

«С кем война?» означает, что нужно вывести название государства, с которым идёт война; «С кем мир?» — что нужно вывести название государства, с которым мир; «Меняем» означает, что с тем государством, с которым шла война, заключён мир, а с тем государством, с которым был мир, началась война; выводить по этой команде ничего не нужно.

Формат вывода

Несколько строк «Евразия» и «Остазия» в соответствии с поступившими командами.

Ввод	Вывод
6 С кем война? С кем мир? Меняем С кем война? Меняем С кем война?	Евразия Остазия Остазия Евразия

```
v = "Евразия"
m = "Остазия"
k = int(input())
for i in range(1, k + 1):
    y = input()
    if y == "С кем война?":
        print(v)
    if y == "С кем мир?":
        print(m)
    if y == "Меняем":
        v, m = m, v
```

Пример 2. Найди кота (break)

Напишите программу, которая находит кота. Пользователь вводит сначала количество строк, потом сами строки. Если хотя бы в одной введённой строке нашлось сочетание букв «Кот» или «кот», программа выводит «МЯУ», иначе программа выводит «НЕТ».

При обнаружении кота цикл должен немедленно прерываться командой break.

Формат ввода

В первой строке записано число n. Далее следует n строк.

Формат вывода

Напечатайте нужное сообщение в зависимости от того, есть во введенных строчках кот или нет.

Ввод	Вывод
4 Как устроен типичный фрукт: кожура; мякоть; косточки.	МЯУ

```
d = int(input())
k = True
for i in range(d):
    x = input()
    k1 = x.find('кот')
    k2 = x.find('Кот')
    if k1 >= 0 or k2 >= 0:
        k = False
        print('МЯУ')
        break
if k:
    print('НЕТ')
```

Пример 3. Найди кота — 2 (break)

Напишите программу, которая находит кота.

Пользователь вводит строки до тех пор, пока он не введёт «СТОП». Программа выводит номер строки, на которой впервые был упомянут кот (наличие в строке сочетаний букв «Кот» или «кот»), или -1 (минус один), если кот не был упомянут.

При обнаружении кота цикл должен немедленно прерываться командой break.

Формат ввода

Несколько строк. Сигнал остановки — строка «СТОП».

Формат вывода

Одно число — номер первой строчки, в которой появился кот, или -1, если кота нет.

Пример

Ввод	Вывод
Как устроен типичный фрукт: кожура; мякоть; косточки. СТОП	3

Примечания

Обратите внимание, что программа должна прекратить свою работу после считывания строки «мякоть;», а остальные строчки, поданные на вход программы, считывать н нужно.

Данная задача дополнительно проверяется преподавателем.

```
k = True
x = input()
k3 = 0
while x != 'СТОП':
    k1 = x.find('кот')
    k2 = x.find('Кот')
    k3 = k3 + 1
    if k1 >= 0 or k2 >= 0:
        k = False
        break
    x = input()
if k:
    print('-1')
else:
    print(k3)
```

Пример 4. Найди кота — 3

Напишите программу, которая находит кота.

Пользователь вводит строки до тех пор, пока он не введёт «СТОП». Программа выводит, во-первых, общее количество строк, в которых были упомянуты коты, во-вторых, номер строки, на которой впервые был упомянут кот (в том же смысле, что и в предыдущих задачах), или -1 (минус один), если кот не был упомянут.

Формат ввода

Несколько строк. Сигнал остановки — строка «СТОП».

Формат вывода

Всегда два числа — общее количество строк с котом и номер первой такой строки (или -1, если такой строки нет). Числа должны быть разделены пробелом.

Пример 1

Ввод	Вывод
Как устроен типичный фрукт: кожура; мякоть; косточки. СТОП	1 3

Пример 2

Ввод	Вывод
Животное такое. С усами, хвостом. Мяукать умеет. Мышей ловит. (Если настроение подходящее.) Кто бы это мог быть? СТОП	0 -1

```
x = input()
n = 0
n1 = 0
k3 = False
k4 = 0
while x != "СТОП":
    k1 = x.find("Кот")
    k2 = x.find("кот")
    k4 = k4 + 1
    if k1 >= 0 or k2 >= 0:
        n = n + 1
        k3 = True
    if (k1 >= 0 or k2 >= 0) and n1 == 0:
        n1 = k4
    x = input()
if k3:
    print(n, n1)
else:
    print(0, -1)
```

Пример 5. Школа танцев

В танце очень важно чувствовать ритм музыки. Напишите программу, которая проверяет, правильно ли ученик отсчитывает: раз, два, три, четыре, раз, два, три, четыре... При этом считается, что у учителя есть некоторый ограниченный запас терпения, и после определённого числа ошибок он заканчивает занятие.

Формат ввода

На первой строке вводится натуральное число n — запас терпения учителя.

Далее следуют строки с отсчётами.

Формат вывода

Пока в вводе повторяются по очереди строки «раз», «два», «три», «четыре», программа не выводит ничего. Как только выводится что-то иное, чем ожидалось, программа выводит: «Правильных отсчётов было <количество правильных отсчётов>, но теперь вы ошиблись.» (Количество правильных отсчётов после этого считается заново, и сами отсчёты снова должны начинаться с «раз».) Если это произошло в n-й раз, выводится «На сегодня хватит.», и дальнейший ввод игнорируется.

Пример

Ввод	Вывод
2 раз два три четыре раз двыа раз два три три	Правильных отсчётов было 5, но теперь вы ошиблись. Правильных отсчётов было 3, но теперь вы ошиблись. На сегодня хватит.

```
y = int(input())
k = 0
n = 0
s = str("Правильных отсчётов было ")
while n < y:
    x = input()
    if x == "раз":
        y1 = input()
        k = k + 1
    else:
        n = n + 1
        print(s, k, ', но теперь вы ошиблись.', sep="")
        k = 0
        continue
    if y1 == "два":
        z = input()
        k = k + 1
    else:
        n = n + 1
        print(s, k, ', но теперь вы ошиблись.', sep="")
        k = 0
        continue
    if z == "три":
        k3 = True
        t = input()
        k = k + 1
    else:
        n = n + 1
        print(s, k, ', но теперь вы ошиблись.', sep="")
        k = 0
        continue
    if t == "четыре":
        k4 = True
        k = k + 1
    else:
        n = n + 1
        print(s, k, ', но теперь вы ошиблись.', sep="")
        k = 0
        continue
print("На сегодня хватит.")
```

Пример 6. Биржевой робот

Напишите робота для автоматической торговли акциями на бирже.

Вводится цена акций в первый, второй и т. д. дни, ноль — сигнал остановки. Возможно, сначала цена уменьшается. В какой-то момент цена начинает расти. Мы покупаем акции в первый день, когда их цена превышает цену в предыдущий день. После этого в какой-то момент цена акций начинает уменьшаться. Мы продаём акции в первый же день, как только их цена становится меньше цены в предыдущий день. Возможно, после этого цены как-то ещё меняются.

Гарантируется, что среди введенных цен точно будет день, когда цена начнет расти, а после день, когда цена начнет падать. После продажи акций робот больше не участвует в торгах на бирже.

Программа должна вывести цену акций, по которой мы их купили, цену, по которой продали, и выгоду с каждой акции (возможно, отрицательную).

Не следует пользоваться советами этого робота в реальной жизни.

Формат ввода

Несколько целых чисел — цены акций в последовательные дни. Число 0 — сигнал прекращения ввода цен.

Формат вывода

Три целых числа — цена покупки, цена продажи, выгода.

Ввод	Вывод
32 30 31 34 38 37 39 0	31 37 6

```
x = int(input())
y = int(input())
s = False
while y != 0:
    if not(s):
        if x < y:
            pokypka = y
            s = True
    if s:
        if x > y:
            prod = y
            break
    x = y
    y = int(input())
print(pokypka, prod, prod - pokypka)
```

Пример 7. Проверка блокчейна

Блокчейн (blockchain) переводится как «цепочка блоков». Это способ хранения данных, защищённый от подделки, используемый, в частности, криптовалютойбиткоин.

Блокчейн действительно представляет собой последовательность блоков. Каждый блок представляет собой некоторую полезную информацию (в частности, в случае **биткоина** это список транзакций за определённый период времени — кто кому когда сколько денег передал), снабжённую случайным числом и некоторыми служебными данными, в том числе **хэшем** — числом, которое по определённой формуле зависит от остальной части блока и хэша предыдущего блока.

Хэш должен быть меньше определённого числа. При этом формула, по которой вычисляется хэш, устроена так, что невозможно получить достаточно маленький хэш иначе, чем перебирая различные значения случайного числа. Поэтому если злоумышленник решит подделать блокчейн (и, допустим, вставить в его середину блок с записью о том, что все люди передают ему все свои деньги), то ему придётся подобрать новое случайное число в новое поддельном блоке и всех последующих (ведь хэш каждого следующего блока зависит от хэша предыдущего), что потребует невозможно больших вычислительных мощностей.

Поэтому блокчейн в целом защищён от подобных атак.

Напишите программу, которая проводит проверку правильности хэшей в модельном блокчейне с простой хэш-функцией.

Блок b_n с номером n включает полезную информацию m_n, представленную натуральным числом, r_n — случайное число от 0 до 255 и h_n — хеш (целое число от 0 до 255). У каждого блока хэш вычисляется по формуле $h_n = 37\times(m_n+r_n+h_{n-1})$ (по модулю 256), при вычислении хэша начального блока h_0 вместо хэша предыдущего блока берётся ноль.

При этом каждый блок представлен одним числом $b_n = h_n + r_n\times256 + m_n\times256^2$. При этом требуется, чтобы хэш h_n был меньше 100.

Формат ввода

На первой строке вводится натуральное число N — количество блоков. Далее следуют N чисел b_n, каждое на отдельной строке.

Формат вывода

Следует вывести номер первого блока, у которого неправильный хэш (не меньше 100 или не совпадает с вычисленным по указанной в условии формуле), или -1, если все хэши в блокчейне правильные. Нумерация блоков идёт с нуля, т. е. они имеют номера от 0 до N-1.

Пример 1

Ввод	Вывод
5 6122802 14406496 15230209 2541121 1758741	-1

Пример 2

Ввод	Вывод
5 1865535 13479687 16689153 1839958 5214020	3

```
x = int(input())
y = True
for i in range(x):
    if i == 0:
        h1 = 0
    else:
        h1 = h
    b = int(input())
    h = b % 256
    b //= 256
    r = b % 256
    b //= 256
    m = b
    if ((h != 37 * (m + r + h1) % 256) or (h >= 100)) and y:
        print(i)
        y = False
if y:
    print('-1')
```

Задания на практику

Задача 1. Бегать — это полезно
Представим, что у нас далёкое будущее и мы можем заниматься спортом на планетах со странными перепадами температур. Спортсмен бегает по стадиону до тех пор, пока температура на улице больше 15 градусов. Он пробегает 20 метров, затем температура падает на два градуса, и, если уже в этот момент она стала меньше либо равна 15 градусам, спорт сразу заканчивается. Если же всё в порядке, то он проходит пешком ещё 10 метров. Затем всё это повторяется. Напишите программу, которая спрашивает у пользователя, сколько на улице градусов, и, исходя из погоды, считает количество пройденных по стадиону метров и выводит ответ на экран.

Задача 2. Расшифровка кода
Нам нужно расшифровать определённый код в виде одного большого числа. Для этого нужно посчитать сумму цифр справа налево. Если мы встречаем в числе цифру 5, то выводим сообщение «Обнаружен разрыв» и заканчиваем считать сумму. В конце программы на экран выводится сумма тех цифр, которые мы взяли.

```
number = int(input('Сколько число '))
summ = 0
while number != 0:
    last_number = number % 10
    print(last_number)
    if last_number == 5:
        print('Обнаружен разрыв')
        break
    summ += last_number
    number //= 10
print('Смумма цифр ', summ)
```

Задача 3. Ставки приняты, ставок больше нет
Костя опять за старое! Только теперь он играет в кубики более надёжно, то есть на какую-то фиксированную сумму. И при этом пока что постоянно выигрывает! Однако по правилам это не мешает ему проиграть сразу всё.
Напишите программу, которая запрашивает у пользователя начальное количество денег и, пока оно меньше 10 000, запрашивает число, которое выпало на кубике (от 1 до 6). Если на кубике выпало 3, то выводится сообщение «Вы проиграли всё!», и деньги обнуляются. Если выпало другое число, к сумме прибавляется 500.

Пример:
Введите стартовую сумму: 5000
Сколько выпало на кубике? 4
Выиграли 500 рублей!
Сколько выпало на кубике? 5
Выиграли 500 рублей!
Сколько выпало на кубике? 3

Вы проиграли всё!
Игра закончена.
Итого осталось: 0 рублей
Пример 2:
Введите стартовую сумму: 9700
Сколько выпало на кубике? 4
Выиграли 500 рублей!
Игра закончена.
Итого осталось: 10200 рублей