Урок 4. Бесконечный цикл. Логический тип данных

В прошлом уроке мы решали задачу с вычислением суммы цифр у числа и по ходу решения столкнулись с таким интересным явлением, как бесконечный цикл. Давайте вспомним наш пример и рассмотрим это явление подробнее.

main.py

```
num = int(input('Введите число: ')) #134
summa = 0
while num > 0:
  print('Последняя цифра: ', num % 10)
  if num % 10 == 5:
    break
  summa += num % 10
print(summa)
```

Бесконечный цикл

Здесь мы не отбрасываем последнюю цифру числа, и получается, что число у нас не меняется и мы вечно находим последнюю цифру. Таким образом, программа зацикливается и ломается, что в нашем случае не есть хорошо. На самом деле, при работе с циклами такая ситуация возникает достаточно часто, если программа написана неверно. Вообще, может возникнуть ощущение, что бесконечный цикл — это плохо и всегда ошибка, однако пока не будем делать поспешных выводов.

Пример 1

main.py

```
number = 5
while number > 0:
    print(number)

```

Представим, что нам дали задание протестировать мощности серверов в нашей компании. Для этого нужно выводить одно и то же число бесконечно, пока компьютеры позволяют. В нашем решении это число 5. На самом деле, решение этой задачи можно сделать даже проще. У нас получается, что 5 всегда больше нуля, правильно? Давайте в условии так и напишем:

main.py

```
number = 5
while 5 > 0:
    print(number)

```

Тогда переменная number нам теперь не нужна ни в начале, ни внутри print, там можно оставить просто цифру 5:

main.py

```
while 5 > 0:
    print(5)

```

Не поверите, но можно сделать ещё проще. Условие «5 больше нуля» — это всегда правда. А слово «правда» на английском будет **True**.

main.py

```
while True:
    print(5)

```

Эта строчка так и читается: «пока правда» или «пока истина». Именно с помощью такой конструкции чаще всего программисты и задают бесконечные циклы. Понятно, что где есть истина, там есть и ложь, то есть неправда. И на английском это будет **False**.

main.py

```
while False:
    print(5)

```

Что произойдёт в такой программе? А ничего не произойдёт, потому что в тело цикла мы не зайдём ни разу. Смысла в этом сейчас мало, поэтому вернёмся к нашему бесконечному циклу:

main.py

```
while True:
    print(5)

```

Заметим, кстати, что оба слова (True и False) пишутся с большой буквы и выделяются синим цветом. Это намекает нам на то, что ими нельзя называть переменные и нужно внимательно следить за регистром букв.

Немного о бесконечных циклах и приложениях

Пока вы это читаете (или смотрите видео), на вашем компьютере запущена куча бесконечных циклов, которые заставляют работать всё ваше программное обеспечение. Например, операционная система. Внутри бесконечного цикла прописан код, который обязан выполняться до тех пор, пока мы просто не выключим компьютер, тем самым принудительно выйдя из цикла. Смысл этого цикла примерно такой: «Эй, компьютер, бесконечно заставляй запущенные программы работать».

То же самое относится и к некоторым приложениям, запущенным на компьютере. И состояния этих приложений как раз и обозначаются с помощью True и False. Давайте рассмотрим программу, которая описывает состояние приложения.

Пример 2

main.py

```
while True:
    active = int(input('Продолжаем работать? 1/0: '))
    if active == 0:
        print('Приложение закрывается...')
        break
print('Работа завершена')

```

Возьмём и запустим вот такую программу. Введём цифру 1 несколько раз, затем введём 0.

```
Продолжаем работать? 1/0: 1
Продолжаем работать? 1/0: 1
Продолжаем работать? 1/0: 1
Продолжаем работать? 1/0: 0
Приложение закрывается...
Работа завершена
>
```

Программа завершилась. Здесь пока ничего сложного: с помощью True мы сделали бесконечный цикл, внутри мы обновляем переменную active, и если ввели 0, то закрываем приложение и выходим из цикла. Немного дополним нашу программу такой маленькой задачей: при закрытии приложения у пользователя спрашивают, поставил ли он оценку приложению или нет. И если поставит, то после завершения работы выводится соответствующее сообщение.

Итак, для начала нам понадобится переменная, которая будет отвечать за то, поставит ли пользователь оценку приложению или нет. Понравилось или не понравилось. Звучит как раз как True и False. Будем считать, что изначально никакую оценку он ставить не хочет.

main.py ×

```
1   rate_check = False
2   while True:
```

Здесь мы присваиваем переменной rate_check значение False. Такие переменные ещё называют **флагом** (потому что он либо поднят, либо опущен). До этого мы напрямую работали только с текстом и числами, а теперь мы познакомились с логическим типом данных «Истина-Ложь». Теперь, после того как мы проверили active на ноль, необходимо спросить у пользователя про оценку.

```
4       if active == 0:
5           print('Приложение закрывается...')
6           break
7       rate = int(input('Поставите оценку приложению? 1/0: '))
```

И если ответ — «да», то поменять состояние переменной rate_check.

```
7       rate = int(input('Поставите оценку приложению? 1/0: '))
8
9       if rate == 1:
10          rate_check = True
```

Всё, что осталось сделать, — это сделать проверку после работы программы.

```
12  print('Работа завершена')
13  if rate_check == True:
14      print('Пользователь поставит оценку')
```

Как мы видим, переменные со значением True-False выполняют роль такого переключателя, как единица и нолик, включить и выключить. И кстати, нам совсем не обязательно делать здесь сравнение с True. Если это переменная логического типа, то Python достаточно оставить вот такую проверку условия:

```
12    print('Работа завершена')
13  if rate_check:
14      print('Пользователь поставит оценку')
15
```

Прямо как while true или while false. Не зря if и while так похожи!

Логический тип данных

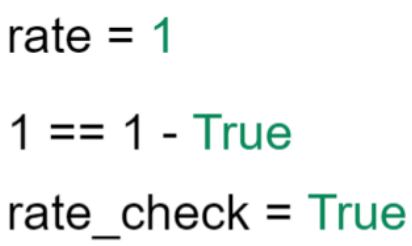
Теперь давайте обратим внимание вот на этот код:

```
if rate == 1:
    rate_check = True
```

Мы только что узнали, что условие внутри условного оператора вычисляется как логическое значение — правда или ложь. Получается, что если rate == 1 — это True, то в rate_check мы запишем True. Кажется, это можно оптимизировать.

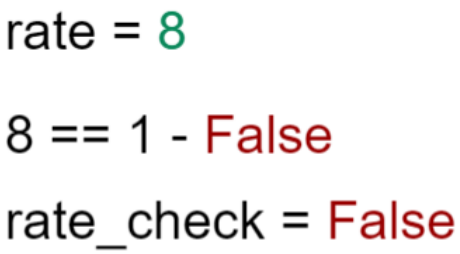
```
rate_check = (rate == 1)
```

Посмотрим на такую запись и разберём, почему эта запись работает так же, как условный оператор выше. Здесь возможны два случая.



Случай 1

Случай 1: пусть в переменной rate лежит единица. Тогда выражение «rate двойное равно 1» вычислится как True, и это самое True попадёт в наш rate_check. То есть поведение точно такое же, как в нашем условном операторе. Но, можете резонно заметить вы, в таком случае мы рискуем записать в наш rate_check что-то не то, если в rate была не единица. Что ж, давайте рассмотрим этот случай.



Случай 2

Случай 2: пусть в rate лежит что угодно, но не единица. Например, 8. Тогда выражение вычислится как False, и это самое False попадёт в наш rate_check. Но ведь в нашей программе там и так False изначально, то есть ничего не изменится и не сломается!

Вот так, зная про логический тип, мы можем сократить код нашей программы и ускорить её работу: ведь в новом варианте мы всё так же вычисляем условие, но не заставляем работать условный оператор.

Итоги урока

Подведём итоги урока:

- Бесконечные циклы могут быть результатом как ошибки в программе, так и намеренного зацикливания, например для работы веб-приложений или операционной системы.
- Для задания бесконечного цикла можно использовать конструкцию while True, что означает «пока истина».
- True и False, то есть истина и ложь, это логический тип данных, его также можно присваивать переменным. Такие переменные называются флагами, их можно использовать для разного рода переключений и проверок.
- Кстати, True и False пишутся с большой буквы и выделяются синим цветом. Поэтому всегда нужно следить за регистром букв.