Методические указания

Урок 4.2. Операторы break, continue.

Задачи урока:

- Цикл while;
- Оператор break;
- Оператор continue;
- Оператор else в циклах.

0. Подготовка к уроку

До начала урока преподавателю необходимо:

1) Просмотреть, как ученики справились с домашним заданием
2) Прочитать методичку

1. Цикл с предусловием while

Учитель: Продолжаем изучать циклы. Изучая цикл for, мы говорили, что в Python (да и в других языках программирования) циклы делятся на два типа:

1. счетные циклы, когда мы знаем сколько в точности раз нужно повторить некоторые действия (цикл for);
2. условные циклы, когда мы не знаем сколько в точности раз нужно повторять, но знаем условие и пока оно истинно мы будем повторять действия (цикл while).

Пришло время познакомиться со вторым типом циклов - циклом while.

Учитель: Давайте начнем со структуры цикла while:

```
while условие:
    блок кода
```

Двоеточие (:) в конце строки сообщает Python, что дальше находится блок кода, называемый телом цикла, именно он и будет повторяться, пока истинно условие.
Условие это любое логическое выражение, которое принимает два значения True/ False.
Учитель: Давайте рассмотрим программу, которая с помощью цикла while распечатает 10 раз слово “Привет”.

```
i = 0
while i < 10:
    print('Привет')
    i += 1
```

Мы создали переменную i с начальным значением 0, и поставили условие в цикле while: пока значение переменной i < 10. Тело цикла содержит две команды:

1. печать текста “Привет”
2. увеличение значения переменной на 1

Тем самым цикл выполнится РОВНО 10 раз, поскольку спустя 10 итераций, значение переменной i будет равно 10 и условие i < 10 становится ложным (ведь 10 = 10, а не меньше 10).

Учитель: *Вопрос: можно ли такой цикл записать с помощью for?*
Ответ: Да, поскольку мы заранее знаем сколько раз в точности надо распечатать слово Привет.

Учитель: При изучении цикла for, мы сказали, что можем считывать сколько угодно чисел, если заранее знаем их количество. Цикл while открывает нам еще одну возможность: считывать числа, до тех пор пока не будет считано какое-то **специфическое число** (заранее указанное в условии). То есть, мы не знаем точное количество считываний чисел, но знаем, когда надо остановиться!

Давайте рассмотрим следующую программу, считывающую числа, пока не будет введено число −1, и выводящую квадраты считанных чисел вместе с поясняющей надписью:

```
num = int(input())
while num != -1:
    print('Квадрат вашего числа равен:', num * num)
    num = int(input())
```

Мы поставили условие: пока считанное число не равно −1. И пока это условие истинно (равно True) выполняется тело цикла, которое в этом случае печатает квадрат считанного числа с поясняющей надписью и переходит к следующему числу (считывает следующее число).

Учитель: *Важны оба шага:*

1. *правильная начальная инициализация переменной num (в нашем случае мы считываем первое число);*
2. *изменение переменной num внутри цикла*.

Если пропустить один из этих шагов, можно получить неработающую программу или работающую бесконечно долго. Впрочем, об этом немного позже.

Учитель: Возможно вы заметили, что цикл while по структуре очень похож на условный оператор if. **Структура цикла while:**

```
while условие:
    блок кода
```

Структура условного оператора if:

```
if условие:
    блок кода
```

Отличие в том, что блок кода в случае истинности условия в условном операторе выполняется лишь один раз, а в цикле while много раз, пока истинно условие.

Учитель: Давайте проведем параллель между двумя циклами. Любой цикл for можно заменить циклом while. Однако, код цикла for получается короче и нагляднее. То есть, когда известно точное количество итераций цикла, выбираем цикл for, это общепринятая практика.

Любой цикл for можно заменить циклом while, но в обратную сторону это утверждение не работает. Существуют задачи, которые решаются только с помощью цикла while. Одна из них - обработка данных до считывания так называемого “стоп значения”, после которого необходимо остановиться.

Учитель: Например программа, считывает целые числа пока пользователь не введёт слово stop. Реализовать ее с помощью цикла for крайне неудобно. Кстати для остановки циклов for и while в нужном месте, мы можем использовать команду break. break полностью останавливает цикл.

```
while True:
    num = input()
    if num == ‘stop’:
        break
```

Учитель: **Бесконечный цикл** - цикл, не имеющий возможности завершиться.

Бесконечные циклы могут возникать именно при работе с циклами while, поскольку на плечи программиста ложится обязанность написать код, делающий условие ложным. Давайте рассмотрим следующий код:

```
i = 0
```

```
total = 0
while i < 10:
    total += i
```

Несложно предположить, что этот код **должен был** накапливать сумму чисел от 0 до 9. Однако переменная i, ответственная за условие цикла, не меняется внутри цикла и постоянно равна 0. Таким образом условие истинно постоянно и цикл превращается в бесконечный.

Важно: Бесконечный цикл повторяется, пока программа не будет прервана.

Учитель: Ну и несколько важных примечаний:

1. цикл while (пока) получил имя из-за характера своей работы: он выполняет некую задачу до тех пор, пока условие остается истинным
2. цикл while называют циклом с **предусловием**, поскольку выполнению тела цикла предшествует проверка условия (как и в условном операторе)
3. цикл while может не выполниться ни разу, если условие ложно с самого начала. Ведь это цикл с предусловием!

1. Оператор прерывания break

Учитель: Познакомимся с операторами, применяемыми в связке с циклами.

Оператор break прерывает **ближайший** цикл for или while. Рассмотрим код:

```
result = 0
for i in range(10)
    num = int(input())
    if num < 0:
        break
    result += num
print(result)
```

Программа считывает 10 чисел и суммирует их, пока не обнаружит отрицательное число. В этом случае выполнение цикла прерывается командой break.

Изучая частые сценарии, в которых используются циклы, мы писали программу для проверки простоты числа.

В ней использовалась сигнальная метка. Сможете вспомнить ее логику?

Учитель: Давайте рассмотрим еще один пример. Программа определяет, содержит ли число цифру 7:

```
n = int(input())
flag = False
while n != 0:
    last_digit = n % 10:
    if last_digit == 7:
        flag = True
        break
    n //= 10
if flag == True:
    print('Число', n, 'содержит цифру 7')
else:
    print('Число', n, 'не содержит цифру 7')
```

Как только встретили цифру 7, нет смысла сканировать цифры дальше.

Учитель: Оператор break открывает еще и возможность работы с бесконечными циклами. Бесконечным называется цикл while, не имеющий возможности завершиться — его условие всегда истинно. В Python можно создать бесконечный цикл следующим образом:

```
while True:
    print('Python!')
```

Такая программа будет печатать текст Python бесконечное количество раз (пока не будет прервана).

Так вот оператор break позволяет прерывать бесконечные циклы.

Учитель: Зачем это нужно? С помощью бесконечного цикла и оператора break **иногда** можно сделать код более читабельным. Например, если цикл while содержит сложное условие, то более простым может быть завершение цикла на основе условий внутри тела цикла, а не на основе условий в его заголовке. Мы можем писать такой код:

```
while True:
    if условие 1:  # условие для остановки цикла
        break
    ...
    if условие 2:  # еще одно условие для остановки цикла
        break
    ...
```

Такой код может быть немного читабельнее.

2. Оператор пропуска итераций continue

Учитель: Следующий оператор называется оператором пропуска отдельных итерации continue. Оператор continue позволяет перейти к следующей итерации цикла for или while до завершения всех команд в теле цикла.

Учитель демонстрирует ученикам код

```
for i in range(1, 101):
    if i == 7 or i == 17 or i == 29 or i == 78:
        continue  # переходим на следующую итерацию
    print(i)
```

и задает вопрос: *что делает данная программа?*

Ответ: программа выводит все числа от 1 до 100, кроме чисел 7, 17, 29 и 78.

Учитель: А можно ли в данном случае обойтись без оператора continue?

Ответ: Да, достаточно поменять тело цикла на:

```
for i in range(1, 101):
    if i != 7 and i != 17 and i != 29 and i != 78:
        print(i)
```

Учитель: А можно ли вообще обойтись без оператора continue?

Ответ: Да, всегда можно переписать программу без него. Более того, в современном программировании считается, что использовать оператор continue не очень хорошо. Кстати у циклов также можно использовать оператор else. Он выполняется когда условие для выполнения цикла становится ложным

```
a = 1
while a < 5:
   print('условие верно')
   a = a + 1
else:
   print('условие неверно')
```

3. Вложенные циклы

Учитель: Циклы — одна из основных конструкций любого языка программирования, они позволяют сокращать программный код.

Учитель: Вложенный цикл расположен в другом цикле. Часы - хороший пример работы вложенного цикла. Три стрелки вращаются вокруг циферблата. Часовая смещается всего на 1 шаг за каждые 60 шагов минутной стрелки. И секундная должна сделать 60 шагов для 1 шага минутной стрелки. Значит за каждый полный оборот часовой стрелки (12 шагов), минутная стрелка делает 720 шагов.
Рассмотрим цикл, частично моделирующий электронные часы. Он показывает секунды от 0 до 59:

```
for seconds in range(60):
    print(seconds)
```

Добавим переменную **minutes** и вложим цикл написанный на предыдущем шаге внутрь еще одного цикла, который повторяется 60 раз:

```
for minutes in range(60):
    for seconds in range(60):
        print(minutes, ':', seconds)
```

Чтобы сделать модель законченной, добавим переменную **hours** для подсчета часов:

```
for hours in range(24):
    for minutes in range(60):
        for seconds in range(60):
            print(hours, ':', minutes, ':', seconds)
```

Результатом работы такого кода будет:
0 : 0 : 0
0 : 0 : 1
0 : 0 : 2
...
23 : 59 : 58

23 : 59 : 59

Учитель: Пример показывает:

- вложенный цикл выполняет все свои итерации для каждой отдельной итерации внешнего цикла;
- вложенные циклы завершают свои итерации быстрее, чем внешние;
- чтобы получить общее количество итераций вложенного цикла, надо **перемножить количество итераций всех циклов**.

Можно вкладывать друг в друга циклы for и while в произвольном порядке.

О работе операторов **break** и **continue** в связке с вложенными циклами:

Учитель: Оператор **break** прерывает тот цикл, где расположен.
Оператор **continue** осуществляет переход на следующую итерацию **ближайшего цикла**.
Например, представим в виде программы процесс решения задач Это цикл while - пока у нас есть задачи, а внутри него вложенный цикл обработки каждой задачи - допустим, `for i in range (12)`. Если вдруг попалась уже решенная задача, нужно сразу перейти к следующей, а не тратить время на повторное решение.

Давайте рассмотрим следующий вложенный цикл:

```
for i in range(3):
    for j in range(4):
        print(i, j)
```

Результатом его выполнения будут строки:

```
0 0
0 1
0 2
0 3
1 0
1 1
1 2
1 3
2 0
2 1
2 2
2 3
```

Учитель: Изменим код, добавив во вложенный цикл условный оператор с оператором break:

```
for i in range(3):
    for j in range(4):
        if i == j:
            break
        print(i, j)
```

Теперь, как только значение переменной j окажется равным значению переменной i, сработает условный оператор и прервется **внутренний цикл (!)**, при этом прерывании произойдет переход к следующей итерации внешнего цикла (поскольку после вложенного цикла for больше нет программных инструкций). Результатом выполнения кода с **break** будет:

```
1 0
2 0
2 1
```

Учитель: Заменим оператор **break** на оператор **continue**:

```
for i in range(3):
    for j in range(4):
        if i == j:
            continue
        print(i, j)
```

Теперь как только значение переменной j окажется равным значению переменной i, сработает условный оператор и произойдет переход **на следующую итерацию внутреннего цикла (!)**. Результатом выполнения кода с **continue** будет:

```
0 1
0 2
0 3
1 0
1 2
1 3
2 0
2 1
2 3
```

Учитель: Бывает необходимо прервать внешний цикл из-за выполнения некоторого условия во внутреннем. И, поскольку оператор **break** прерывает именно ближайший цикл, для такой реализации потребуется использование **флага:**:

```
flag = False
for i in range(3):
    for j in range(3):
        if (j==1) and (i==2):
            flag = True
            break
        print(i, j)
    if flag == True:
        break
```

3. Решение задач

Задача 1

Напишите программу, которая выводит чётные числа из заданного списка и останавливается, если встречает число 237.

Решение

```
numbers = [ 386, 462, 47, 418, 907, 344, 236, 375, 823, 566, 597, 978, 328,
615, 953, 345, 399, 162, 758, 219, 918, 237, 412, 566, 826, 248, 866, 950,
626, 949, 687, 217, ]

for x in numbers:

    if x == 237:

        break

    elif x % 2 == 0:

        print(x)
```

Дополнительно

Если на уроке остается время, то ученикам можно предложить начать прорешивать домашнее задание.

Домашняя работа

Задача 1

Напишите программу, которая спрашивает у пользователя ввести число и выводит факториал данного числа. Пример факториал числа 5! = 1 x 2 x 3 x 4 x 5

Решение

```
number = int(input())
factorial = 1
for i in range(1, number + 1):
    factorial *= i
print(factorial)
```