ЯЗЫК PYTHON

Лекция 3. Цикл while

Циклы

- Хочется делать повторящиеся действия
- Иногда до какого-то момента
- Иногда - просто всегда
- Есть специальная конструкция языка - цикл

Цикл while

- Ключевое слово while
- Логическое выражение
- Двоеточие
- Тело цикла

Логика работы

- Проверяем условие
- Если False, то выходим из цикла
- Если True, то выполняем тело цикла
- Когда выполнили, снова проверяем условие
- И так далее

Простейший пример

```
1 i = 0
2 while i < 10:
3     name = input() # вводим имя
4     print("Hello, " + name)
5     i += 1
6 # Такого рода циклы лучше делать по-другому
```

Еще пример

```
1 name = ''
2 while not name:
3     name = input() # вводим имя
4 
5 print("Hello, " + name)
```

Проанализируем

- Первый вариант - самый простой
- И самый естественный
- Тело цикла выполняется полностью
- Или не выполняется совсем
- И первая проверка условия не требует специальной обработки

Проанализируем

- Второй вариант - уже посложнее
- Тело цикла - тоже либо выполняеся полностью, либо не выполняется совсем
- Но для первой проверки условия мы делаем маленький трюк
- Мимикрируем под ситуацию ошибочного ввода

Еще усложним

- Хотим напечатать пояснение к вводу
- При первой попытке - "Введите имя"
- При повторных - "Повторите ввод"

Первый вариант

- Поместим print между while и input
- Тогда нужно отличать первое исполнение цикла
- От последующих
- Например, завести вспомогательную переменную

Реализация

```
 1 name = ''
 2 first = True
 3 while not name:
 4     if first:
 5         print("Введите имя")
 6     else:
 7         print("Еще раз попробуйте")
 8     name = input() # вводим имя
 9     first = False
10 
11 print("Hello, " + name)
```

Второй вариант

- Мы специально определили переменную name до цикла
- Чтобы обработать первую итерацию цикла
- Но теперь у нас есть еще специальный признак первой итерации
- Используем его
- И еще возможность ленивого вычисления

Реализация

```
 1 first = True
 2 while first or not name:
 3     if first:
 4         print("Введите имя")
 5     else:
 6        print("Еще раз попробуйте")
 7     name = input() # вводим имя
 8     first = False
 9
10 print("Hello, " + name)
```

Третий вариант

- Попробуем без переменной first
- С первоначальной печатью попроще
- Можно ее поместить перед циклом
- И это даже логичнее смотрится
- Потому что она должна отработать ровно 1 раз

Пока еще с переменной first

```
 1 first = True
 2 print("Введите имя")
 3
 4 while first or not name:
 5     if not first:
 6         print("Еще раз попробуйте")
 7     name = input() # вводим имя
 8     first = False
 9
10 print("Hello, " + name)
```

Избавимся от переменной first

```
print("Введите имя")
name = ''

while not name:
    name = input() # вводим имя
    if not name:
        print("Еще раз попробуйте")

print("Hello, " + name)
```

Проанализируем

- От first избавились
- Но вернултсь к инициализации name
- И теперь у нас дублируется условие
- В if и в while

Корень проблемы

- Тело цикла - чтение строки и обработка
- Ключевой момент для принятия решения - после чтения строки
- То есть в середине тела цикла
- Хотелось бы иметь возможность выйти из цикла в середине тела

break

- Такая возможность есть
- Ключевое слово break
- Прекращает цикл
- Обычно используется в сочетании с if

Применим break

```
print("Введите имя")

while True:
    name = input() # вводим имя
    if name:
        break
    else:
        print("Еще раз попробуйте")

print("Hello, " + name)
```

Или так

```
1 print("Введите имя")
2
3 while True:
4     name = input() # вводим имя
5     if name:
6         break
7     print("Еще раз попробуйте")
8
9 print("Hello, " + name)
```

Варианты использования break

- break не обязательно сочетается с True в условии цикла
- Например, если в данном примере прекращать ввод после 5 попыток
- То счетчик попыток можно проверять под while
- И оставить break там, где он был

Варианты использования break

- break без большой надобности нежелателен
- Особенно в длинном теле цикла
- Усложняет понимание
- Но есть типовые ситуации, где он удобен

Когда break уместен

Рассмотрим другой сценарий

- Читаем в цикле входные данные
- Построчно
- Каждую строчку как-то обрабатываем
- Например, превращаем ее в число и считаем квадратный корень
- Иногда могут попадаться отрицательные числа и мы хотим их пропустить

Вариант реализации

```
1 i = 0
2 while i < 10:
3     v = float(input())
4     if v >= 0:
5         print(v, v ** 0.5)
6         i += 1
```

Проанализируем

- Вполне работающее решение
- Но представим себе, что обработка неотрицательного v подлиннее
- И тогда большая часть тела цикла оказывается под if
- И у нас фактически увеличивается смещение тела цикла
- Доминирующей его части

continue

- На такой случай есть отдельная конструкция
- Ключевое слово continue
- Встретив этой ключевое слово в цикле
- Python прекращает текущую итерацию
- И переходит к очередной проверке условия цикла

Пример с continue

```
1 i = 0
2 while i < 10:
3     v = float(input())
4
5     if v < 0:
6         continue
7
8     print(v, v ** 0.5)
9     i += 1
```

Еще пример с continue

```
while True:
    name = input()

    if not name:
        continue

    print('Hello, ' + name)
```

Как отличить выход по break и по False в условии

- Иногда в точке выхода из цикла хочется понять, как мы из него вышли
  - Потому что условие превратилось в False
  - Или потому что сработал break
- Например, закончили перебор вариантов или нашли нужный

Как отличить выход по break и по False в условии

- Можно завести логическую переменную
- Перед циклом поставить ее в True перед циклом
- Поменять ее на False перед break
- Или наоборот

Пример с логической переменной

```
 1 n_total = 0
 2 n_success = 0
 3 found_break = False
 4
 5 while n_total < 10:
 6     name = input()
 7     n_total += 1
 8
 9     if not name:
10         continue
11
12     n_success += 1
13     print('hello, ' + name)
14
15     if n_success == 3:
```

else с while

- else можно использовать с while
- Если очередная проверка условия дала False
  - Выполняется код из else
- Если выходим по break - else-блок не выполняется

Пример с else

```
 1 n_total = 0
 2 n_success = 0
 3
 4 while n_total < 10:
 5     name = input()
 6     n_total += 1
 7
 8     if not name:
 9         continue
10
11     n_success += 1
12     print('hello, ' + name)
13
14     if n_success == 3:
15         break
```

Вложенные циклы

- Иногда нужно итерироваться в двух измерениях
- Например, печатаем таблицу умножения
- Нам нужно напечатать какое-то количество строчек
- Содержимое которых в целом следует определенному шаблону
- Получаем внешний цикл

Вложенные циклы

- А каждая строчка - состоит из фрагментов
- И каждый фрагмент строится по повторяющемуся шаблону
- Нам нужно напечатать какое-то количество строчек
- Получаем внутренний цикл

Вложенные циклы

- Ничего специального не требуется
- В теле цикла могут быть любые предложения
- В том числе - другие циклы
- break и continue применяются к самому вложенному циклу

pass

- Есть специальное предложение - pass
- Оно означает отсутствие действия
- Может применяться везде, где нужно какое-то действие
- По правилам синтаксиса
- Но ничего конкретного делать не хочется

pass

- В цикле есть вариант полезного использования pass
- Хотим остановить программу
- Пока пользователь не прервет ее выполнение извне
- Нужен бесконечный цикл
- Синтаксис требует хоть какого-то предложения в блоке

С бесконечным циклом

```
1 print('hello')
2
3 while True:
4     pass
```

Запись в файл

- Есть встроенная функция open
- Открывает файл
- Принимает два параметра
  - Имя файла
  - Режим открытия

Имя файла

- Строка
- Есть два варианта
  - Абсолютный путь: 'c:\\users\\vasya\\data.txt', '/tmp/12345'
  - Относительный путь: 'data.txt', '12345', 'data/4321.txt'

Путь

- Относительный путь смотрится, начиная от текущего каталога
- Текущий каталог - свойство программы
- При вызове из командной строки определяется тем каталогом, откуда вызвали python
- Если в Windows вызываем с помощью ярлыка - можно задать в свойствах приложения

Режим открытия

- Строка специального содержания
- Выбираем между бинарным файлом и текстовым
- Бинарный файл - буква 'b'
- Текстовый файл - буква 't'

Режим открытия

- Выбираем между чтением и записью
- Теоретически, можно читать и писать
- Но (обычно) удобнее и надежнее - что-то одно
- Чтение - 'r'
- Запись - 'w' (старое содержимое удаляется)

Работаем с файлом

- Пример открытия: `open('data.txt', 'wt')`
- Результат сохраняем в переменную
- Используем метод write() для записи данных
- Закрываем файл
  - Метод close
  - Или специальная конструкция with ... as (неявный вызов close)

Пример записи в файл

```
1 with open('data.txt', 'wt') as f:
2     f.write('hello\n')
3
4 # в этой точке файл уже закрыт
```

Пример упрощенного чтения из файла

```
1 with open('in.txt') as fin:
2     with open('out.txt', 'wt') as fout:
3         while True:
4             s = fin.readline()
5             if not s:
6                 break
7             fout.write(str(int(s) ** 0.5) + '\n')
```