

Операторы цикла

01

Цикл в программировании

Цикл

блок повторяющегося кода при определенных условиях

Циклы выполняются до тех пор, пока определенное условие истинно или пока не будет достигнута определенная граница. Когда условие не выполняется или граница достигнута, выполнение цикла прекращается и управление передается следующим строкам кода после цикла

Итерация

процесс выполнения блока кода

Рассмотрим два способа решения алгоритмической задачи, нам необходимо 5 раз вывести слово “Hello”

Пример

ооо	
	вывести “Hello”

Пример

ооо	
	Пока выведено “Hello” < 5: вывести “Hello”

Решение данной задачи предполагает использование цикла. Видим, что второе решение более лаконичное, так как оно содержит в себе цикл, в данном случае цикл выполнил 5 итераций. Если бы нам нужно было вывести большее количество раз слово “Hello”, например, 100 раз, то в таком случае первый способ решения стал совсем затратным и неэффективным

02

Цикл while

Цикл while в языке программирования Python выполняет один и тот же блок кода до тех пор, пока условие остается истинным. Это позволяет повторять выполнение кода, пока не будет достигнуто заданное условие или до тех пор, пока условие истинно

Синтаксис цикла while

...	
	while условие: блок кода # блок кода, который будет выполняться, пока условие истинно

Структура while состоит из ключевого слова `while`, за которым следует условие, и блока кода, который должен выполняться, пока условие истинно

Здесь условие - это выражение, которое возвращает значение типа `bool` (`True` или `False`). Если условие истинно, блок кода внутри цикла `while` выполняется. После завершения блока кода выполнение вернется к проверке условия. Если условие остается истинным, блок кода будет выполнен снова. Цикл продолжает выполняться, пока условие остается истинным

Пример

...	
	<pre>x = 1 while x <= 5: print(x) x += 1</pre>

Вывод

...	
	<pre>1 2 3 4 5</pre>

В этом примере цикл `while` будет выполняться, пока значение переменной `x` не превысит 5. На каждой итерации цикла будет выводиться значение переменной `x`, а затем значение переменной `x` будет увеличиваться на 1. Цикл продолжает работать до тех пор, пока `x` не станет больше 5

Оператор `while` позволяет повторять выполнение блока кода на основе проверки условия, что делает его мощным инструментом для автоматизации повторяющихся задач

Операторы `else`, `break` и `continue`

В цикле `while` в Python можно использовать операторы `else`, `break` и `continue` для управления выполнением цикла.

01 Оператор `else`

Блок кода, определенный внутри оператора `else`, будет выполнен после окончания работы цикла `while`, если условие становится ложным, и цикл завершается естественным образом без использования оператора `break`

Пример

...	
	<pre>count = 0 while count < 5: print(count) count += 1 else: print("Цикл завершен")</pre>

Вывод

...	
	<pre>0 1 2 3 4 Цикл завершен</pre>

В этом примере цикл while будет продолжаться, пока значение переменной count меньше 5. Когда count становится равным 5, условие становится ложным и цикл завершается. Затем блок кода в разделе else будет выполнен, и выведется сообщение "Цикл завершен"

02 Оператор break

Когда внутри цикла while встречается оператор break, выполнение цикла прерывается, и управление передается непосредственно за пределы цикла

Пример

```
... |  
count = 0  
while count < 5:  
    print(count)  
    count += 1  
    if count == 3:  
        break
```

Вывод

```
... |  
0  
1  
2
```

В этом примере цикл while будет повторяться, пока значение переменной count меньше 5. С помощью оператора if проверяется условие count == 3. Когда значение count становится равным 3, выполнение цикла прерывается с помощью оператора break

03 Оператор continue

Когда внутри цикла while встречается оператор continue, выполнение текущей итерации прекращается, и управление возвращается к началу цикла для следующей итерации

Пример

```
... |  
count = 0  
while count < 5:  
    count += 1  
    if count == 3:  
        continue  
    print(count)
```

Вывод

```
... |  
1  
2  
4  
5
```

В этом примере цикл while будет повторяться, пока значение переменной count меньше 5. Внутри цикла с помощью оператора if проверяется условие count == 3. Когда значение count становится равным 3, оператор continue пропускает оставшуюся часть цикла и переходит к следующей итерации

Это означает, что при выводе числа 3 выполнение цикла пропустит дальнейший код и перейдет к следующей итерации. В результате будут выведены числа от 1 до 5, за исключением числа 3

Операторы else, break и continue предоставляют дополнительные возможности для контроля выполнения цикла while и позволяют более гибко управлять кодом внутри цикла в соответствии с определенными условиями и требованиями

Вложенные циклы

Вложенные циклы while представляют собой циклы, которые находятся внутри других циклов while. Они позволяют повторять определенные действия внутри внешнего цикла до тех пор, пока выполняется определенное условие

Пример

```
...  
x = 1  
while x <= 3:  
    y = 1  
    while y <= 5:  
        print(x, y)  
        y += 1  
    x += 1
```

Вывод

```
...  
1 1  
1 2  
1 3  
1 4  
1 5  
2 1  
2 2  
2 3  
2 4  
2 5  
3 1  
3 2  
3 3  
3 4  
3 5
```

В этом примере внешний цикл while выполняется до тех пор, пока значение переменной x не превысит 3. Внутри внешнего цикла находится вложенный цикл while, который выполняется до тех пор, пока значение переменной y не станет больше 5. На каждой итерации вложенного цикла while происходит вывод значений переменных x и y. Когда вложенный цикл завершается, внешний цикл продолжает свое выполнение.

Таким образом, вложенные циклы while позволяют повторять заданные действия внутри другого цикла до достижения определенного условия

03

Цикл for

Цикл for используется для итерации (перебора) элементов в итерируемом объекте, таком как последовательность, список, строка, кортеж, множество или словарь

Синтаксис цикла for

```
...  
for элемент in итерируемый_объект:  
    блок_кода # выполнить какое-то действие
```

На каждой итерации цикла, переменная элемент принимает следующее значение из итерируемого объекта, и затем выполняется заданное действие внутри цикла

Функция range

это функция, которая возвращает арифметическую прогрессию в виде итерируемого объекта



Функция `range()` возвращает объект диапазона, который можно использовать для итерации или преобразования в другой тип данных, такой как список

Примеры использования функции `range()`

01 Итерация по диапазону чисел

Пример

```
... |  
for num in range(1, 6):  
    print(num)
```

Вывод

...	
1	
2	
3	
4	
5	

02 Использование шага в диапазоне чисел

Пример

```
... |  
for num in range(1, 10, 2):  
    print(num)
```

Вывод

...	
1	
3	
5	
7	
9	

03 Обратный порядок диапазона чисел

Пример

```
... |  
for num in range(10, 0, -1):  
    print(num)
```

Вывод

...	
10	
9	
8	
7	
6	
5	
4	
3	
2	
1	

Таким образом, функция range() предоставляет удобный способ создания последовательности чисел

Операторы else, break и continue

В цикле for можно использовать операторы else, break и continue для управления выполнением цикла

01 Оператор break

Оператор break прерывает выполнение цикла, даже если все итерации не были выполнены

Пример

```
...  
for i in range(5):  
    if i == 3:  
        break  
    print(i)
```

Вывод

...	
0	
1	
2	

В этом примере, после выполнения 3 итераций, цикл будет прерван при использовании оператора break, и оставшиеся итерации не будут выполнены

02 Оператор else

Блок кода в операторе else будет выполнен, когда цикл завершит свою работу без использования оператора break

Пример

```
...  
for i in range(5):  
    print(i)  
    if i == 3:  
        break  
else:  
    print("Цикл завершен без использования break")
```

Вывод

...	
0	
1	
2	
3	

В этом примере, после выполнения 4 итераций, цикл будет прерван при использовании оператора break, и блок кода в операторе else не будет выполняться

03 Оператор continue

Оператор continue пропускает текущую итерацию цикла и переходит к следующей итерации

Пример

```
...  
for i in range(5):  
    if i == 2:  
        continue  
    print(i)
```

Вывод

...	
0	
1	
2	
3	
4	

В этом примере, при использовании оператора continue в итерации со значением 2, текущая итерация будет пропущена, и цикл продолжит выполнение со следующей итерации

Таким образом, операторы `else`, `break` и `continue` позволяют контролировать выполнение цикла `for` в Python. Оператор `else` может быть использован для выполнения блока кода после завершения цикла, `break` прерывает выполнение цикла и `continue` пропускает текущую итерацию и переходит к следующей

Вложенные циклы

Вложенные циклы `for` представляют собой циклы `for`, которые находятся внутри других циклов `for`. Они позволяют повторять определенные действия внутри внешнего цикла до тех пор, пока выполняется определенное условие

Пример

```
...  
for i in range(1, 4): # Внешний цикл  
    for j in range(1, 6): # Вложенный цикл  
        print(i, j)
```

Вывод

```
...  
1 1  
1 2  
1 3  
1 4  
1 5  
2 1  
2 2  
2 3  
2 4  
2 5  
3 1  
3 2  
3 3  
3 4  
3 5
```

В этом примере внешний цикл `for` выполняется для значений `i` от 1 до 3. Внутри внешнего цикла находится вложенный цикл `for`, который выполняется для значений `j` от 1 до 5. На каждой итерации вложенного цикла `for` выводятся значения `i` и `j`. Когда вложенный цикл завершается, внешний цикл продолжает свое выполнение

Таким образом, вложенные циклы `for` позволяют повторять заданные действия внутри другого цикла до достижения определенного условия

04

Когда использовать `for`, а когда `while`?

Выбор между циклами `for` и `while` зависит от конкретной задачи, требований и структуры данных. Вот несколько рекомендаций, когда использовать каждый из них

Использование цикла `for`

- Если вы хотите выполнить операцию на каждом элементе в итерируемом объекте (например, в списке, строке или кортеже), лучше использовать цикл `for`, так как он предоставляет удобный способ итерации по элементам

- Когда количество итераций заранее известно или фиксировано (например, с помощью функции `range()`), цикл `for` может быть более удобным в использовании и помогает избежать ошибок
 - При работе с коллекциями или последовательностями, где важна последовательность элементов, цикл `for` может быть предпочтительным, так как он гарантирует, что элементы будут перебираться в правильном порядке
-

Использование цикла `while`

- Когда условие выполнения цикла зависит от переменных или условий, изменяющихся в процессе выполнения цикла, цикл `while` может быть более подходящим выбором
- Если количество итераций неизвестно заранее и зависит от внешних факторов, условия или контекста, то цикл `while` может быть предпочтительным
- Если необходимо выполнить действие до тех пор, пока выполняется определенное условие или пока переменная не превысит определенное значение, цикл `while` может быть наиболее эффективным способом

Конечный выбор между циклами `for` и `while` также зависит от вашего собственного стиля программирования и предпочтений. В некоторых ситуациях возможно использование как цикла `for`, так и `while`

Итог

В этой теме мы разобрали понятие цикла, изучили типы циклов, поговорили о вложенных циклах, рассмотрели применение операторов `else`, `break`, `continue`, а также выделили критерии использования цикла `for` и `while`