IT CAREER HUB

Урок 7.

Цикл while

IT CAREER HUB

Цикл

Цикл — это конструкция в программировании, которая позволяет повторять выполнение блока кода (тела цикла) несколько раз, пока выполняется определённое условие или пока не пройдётся по всем элементам последовательности.

Циклы широко используются для автоматизации повторяющихся операций и работы с коллекциями данных.

Итерация

Итерация — это одно выполнение тела цикла. Итерации происходят до тех пор, пока выполняется условие цикла.

Цикл while

Цикл while — это цикл с предусловием, который выполняет блок кода до тех пор, пока условие остаётся истинным.

Его можно сравнить с оператором if. Только if проверяет условие один раз и выполняет код, если оно истинно, а цикл while повторяет выполнение кода многократно, пока условие остаётся истинным.

Синтаксис цикла while

```python
while условие: # условие цикла

    # блок кода или тело цикла
```

- **Условие:**
  Это выражение, которое проверяется перед каждой итерацией. Если оно истинно (True), выполняется блок кода. Как только условие становится ложным (False), цикл завершает свою работу.
- **Блок кода:**
  Код, который будет выполняться на каждой итерации, пока условие истинно. После выполнения блока кода интерпретатор всегда возвращается к проверке условия.

Пример

```python
i = 1

while i <= 5:

    print(i)

    i += 1
```

Объяснение:

- Переменная i инициализируется значением 1.
- Цикл продолжается до тех пор, пока значение переменной i меньше или равно 5.
- Внутри цикла переменная увеличивается на 1 с каждой итерацией с помощью оператора i += 1.
- Как только i становится больше 5, цикл завершается.

IT CAREER HUB

Задания для закрепления 1

1. Что произойдёт при выполнении следующего кода?

```python
i = 1
while i <= 3:
    print(i)
```

a. Код выведет числа от 1 до 3
b. Код выведет числа от 1 до 4
c. Код выведет 1 один раз
d. Код зациклится и будет выводить числа бесконечно

Посмотреть ответ

2. Что произойдёт при выполнении следующего кода?

```python
i = 1
while i <= 3:
    print(i)
    i += 1
```

a. Код выведет числа от 1 до 2
b. Код выведет числа от 1 до 3
c. Код выведет 1 один раз
d. Код зациклится и будет выводить числа бесконечно

Посмотреть ответ

3. Какой результат выведет следующий код?

```python
i = 10
while i > 7:
```

```
    i -= 1
print(i)
```

a. Число 8
b. Число 7
c. Числа от 10 до 8
d. Код выведет ошибку

Посмотреть ответ

4. Найди ошибку в коде

```python
i = 0
while i < 5:
print(i)
i += 1
```

Посмотреть ответ

5. Какое значение примет переменная i после выполнения следующего цикла?

```python
i = 2
while i <= 8:
    i += 2
```

a. 6
b. 8
c. 10
d. 12

Посмотреть ответ

Оператор break

Оператор break используется для принудительного завершения выполнения цикла. Когда интерпретатор встречает break внутри цикла, выполнение цикла немедленно прекращается, даже если условие цикла ещё истинно.

Пример

```python
i = 1

while i <= 10:

    print(i)

    if i == 5:

        break  # Прерывание цикла, когда i станет равно 5

    i += 1
```

Использование break с пользовательским вводом

Оператор break может быть полезен для выхода из цикла, когда программа ожидает ввода от пользователя и требуется завершить цикл при определённых условиях.

Пример

```python
while True:

    user_input = input("Введите 'exit', чтобы завершить цикл: ")

    if user_input == "exit":
```

```
        break

    print("Вы ввели:", user_input)
```

IT CAREER HUB

Оператор continue

Оператор continue в циклах используется для пропуска оставшейся части кода текущей итерации и перехода к следующей итерации цикла. Это значит, что после выполнения continue программа возвращается к проверке условия цикла и начинает новую итерацию, пропуская код, который находится ниже оператора continue.

Бесполезно ставить continue в последней строке тела цикла

Синтаксис:

```python
while условие:

    # код до continue

    if условие_для_continue:

        continue  # пропуск оставшегося кода в этой итерации

    # код, который будет пропущен, если сработает continue
```

Пример использования continue

Пример 1

```python
i = 0
while i < 5:
    i += 1
    if i % 2 == 0::
        continue  # Пропускаем итерацию, когда i четное
    print(i)
```

Оператор continue полезен, когда необходимо пропустить выполнение кода при выполнении определённого условия, но при этом не завершать весь цикл, а продолжать выполнение со следующей итерации.

Пример 2: пропуск ввода при условии

Python

```python
result = 1

while True:

    user_input = input("Введите число для перемножения: ")

    if user_input == "0":

        print("Пропуск итерации")

        continue  # Пропускаем оставшуюся часть текущей итерации

    if user_input == "exit":

        print("Выход из программы")

        break  # Прерывание цикла

    result *= int(user_input)

    print("Результат перемножения:", result)
```

Бесконечный цикл

Бесконечный цикл — это цикл, который никогда не завершает своё выполнение, потому что его условие всегда остаётся истинным.

Пример

```python
while True:

    print("Этот цикл будет выполняться бесконечно")
```

Бесконечные циклы часто используются в программах, где требуется постоянная работа, но их выполнение обычно прерывается при наступлении определённых условий. Для этого используют оператор break, который завершает цикл.

Пример

```python
while True:

    user_input = input("Введите 'stop', чтобы завершить цикл: ")

    if user_input == "stop":

        print("Цикл завершён.")

        break  # Прерывание цикла, если пользователь ввёл 'stop'
```

Бесконечные циклы также могут возникнуть случайно, если неправильно прописано условие завершения.

IT CAREER HUB

Примеры

Python

```python
i = 0

while i < 10:

    print(i)

    # Пропущено увеличение i, поэтому условие всегда истинно, и цикл бесконечен
```

Python

```python
i = 0

while i < 10:

    print(i)

    i -= 1

    # i меняется в другую сторону, не приближаясь к условию выхода
```

IT CAREER HUB

Задания для закрепления 2

1. Что будет выведено при выполнении следующего кода?

```python
i = 1
while i <= 5:
    if i == 3:
        break
    print(i)
    i += 1
```

a. 1, 2
b. 1, 2, 3
c. 1, 2, 3, 4, 5
d. Код зациклится

Посмотреть ответ

2. Какой результат будет выведен?

```python
i = 0
while i < 5:
    i += 1
    if i == 3:
        continue
    print(i)
```

a. 1, 2, 4, 5
b. 1, 2, 3, 4, 5
c. 0, 1, 2, 4, 5
d. Код зациклится

Посмотреть ответ

IT CAREER HUB

3. Какой результат будет выведен?

```python
i = 1
while i <= 5:
    if i % 2 == 0:
        i += 1
        continue
    print(i)
    i += 1
```

a. 1, 2, 3, 4
b. 1, 3, 5
c. 1, 3
d. 2, 4

Посмотреть ответ

IT CAREER HUB

Конструкция while/else

Конструкция while/else позволяет выполнить блок кода else после завершения цикла while, при нормальном завершении цикла, то есть без использования оператора break. Если в цикле срабатывает break, блок else не будет выполнен.

Синтаксис:

```python
while условие:

    # блок кода while

else:

    # блок кода else
```

Пример 1

```python
i = 1

while i <= 5:

    print(i)

    i += 1

else:

    print("Цикл завершён без прерываний.")
```

Пример 2

```python
result = 1
num = 1
while num < 6:
    user_input = input("Введите " + str(num) + "-е число для перемножения: ")
    num += 1
    if user_input == "0":
        print("Некорректные данные. Выход из программы")
        break  # Прерывание цикла
    result *= int(user_input)
else:
    print("Цикл завершён без прерываний.")
    print("Результат перемножения:", result)
```

continue в while/else

Оператор continue не прерывает цикл, а только пропускает часть итерации, поэтому блок else всё равно выполнится.

Пример 1

```python
result = 1

num = 1

while num < 6:

    user_input = input("Введите число №" + str(num) + " для перемножения: ")

    num += 1

    if user_input == "0":
```

```
            print("Некорректные данные. Выход из программы")
            break  # Прерывание цикла
        if int(user_input) > 1000:
            print("Слишком большое число. Пропуск итерации")
            continue  # Пропускаем оставшуюся часть текущей итерации
        result *= int(user_input)
    else:
        print("Цикл завершён без прерываний.")
        print("Результат перемножения:", result)
```

IT CAREER HUB

Задания для закрепления 3

1. Какой результат будет выведен?

```python
i = 1
while i <= 3:
    if i == 2:
        i += 1
        continue
    print(i)
    i += 1
else:
    print("Цикл завершён.")
```

a. 1, 3
b. 1, 3, "Цикл завершён."
c. "Цикл завершён."
d. 1, 2, 3

Посмотреть ответ

2. Какой результат будет выведен?

```python
i = 1
while i <= 4:
    if i == 3:
        break
    print(i)
    i += 1
else:
    print("Цикл завершён.")
```

a. 1, 2, "Цикл завершён."
b. 1, 2
c. 1, 2, 3
d. Цикл зациклится

Посмотреть ответ

Модуль random

Модуль random — это стандартный модуль Python, который предоставляет функции для генерации случайных чисел, а также для работы со случайным выбором элементов из коллекций.

Этот модуль полезен для задач, где необходима случайность данных, событий.

Примеры использования модуля random

Генерация случайного числа с плавающей запятой

```python
import random


# Генерируем случайное число от 0.0 до 1.0

print(random.random())
```

Генерация случайного целого числа

```python
import random


# Генерируем случайное целое число от 1 до 10 (включая обе границы)

print(random.randint(1, 10))
```

Генерация случайного целого числа с шагом

```python
import random

# Генерируем случайное целое число от 1 до 10 (не включая число 10)
print(random.randrange(1, 10))
# Генерируем случайное целое число от 1 до 10 (не включая число 10) и с шагом 2
(только по нечетным числам)
print(random.randrange(1, 10, 2))
```

Когда использовать random

Модуль random полезен в следующих случаях:

- **Симуляции:** Игры, моделирование случайных событий.
- **Генерация тестовых данных:** Создание случайных чисел или строк для тестирования алгоритмов.
- **Программы, требующие случайности:** Лотереи, случайное распределение задач.
- **Перемешивание данных:** Например, случайное перемешивание списка студентов для презентации или задания.

IT CAREER HUB

Задания для закрепления 4

Найдите ошибку в коде:

```python
import random


number = random.randint(1, 5)
if number == 6:
    print("Вы выиграли!")
```

Посмотреть ответ

IT CAREER HUB

Ответы на задания

Задания на закрепление 1	Вернуться к заданиям
1. Результат выполнения кода	Ответ: d
2. Результат выполнения кода	Ответ: b
3. Результат выполнения кода	Ответ: b
4. Ошибка в коде	Ответ: Внутри цикла блок кода должен быть с отступом
5. Значение i	Ответ: c
Задания на закрепление 2	Вернуться к заданиям
1. Результат выполнения кода	Ответ: a
2. Результат выполнения кода	Ответ: a
3. Результат выполнения кода	Ответ: b
Задания на закрепление 3	Вернуться к заданиям
1. Результат выполнения кода	Ответ: b
2. Результат выполнения кода	Ответ: b
Задания на закрепление 4	Вернуться к заданиям
Ошибка в коде	Ответ: Ошибка заключается в том, что `random.randint(1, 5)` генерирует число от 1 до 5 включительно, поэтому проверка на `number == 6` никогда не выполнится

IT CAREER HUB

Практические задания

1. Обратный отсчёт

Напишите программу, которая выводит обратный отсчёт от введённого числа до 1 и в конце "Отсчёт завершён!".

Пример ввода:

```
Python
Введите начальное число: 5
```

Пример вывода:

```
Python
5
4
3
2
1
Отсчёт завершён!
```

Решение:

```python
num = int(input("Введите начальное число: "))
while num > 0:
    print(num)
    num -= 1
print("Отсчёт завершён!")
```

IT CAREER HUB

2. Сумма положительных чисел

Напишите программу, которая запрашивает у пользователя числа и суммирует только положительные. Программа завершится и выведет общую сумму, когда пользователь введёт "0".

Пример вывода:

```
Python
Введите число: 5
Введите число: 10
Введите число: -1
Введите число: 3
Введите число: 0
Общая сумма положительных: 18
```

IT CAREER HUB

Решение:

```python
total_sum = 0

while True:
    num = int(input("Введите число: "))
    if num < 0:
        continue
    if num == 0:
        break
    total_sum += num

print("Общая сумма:", total_sum)
```