**Управляющие конструкции - Башкаруу структуралары**

**1. Цикл for в Python**

В Python, цикл for используется для перебора элементов последовательности (список, кортеж, строка) или других итерируемых объектов. Синтаксис:

* for переменная in последовательность:
* # блок кода для выполнения

1. Проход по списку

fruits = ["яблоко", "банан", "вишня"]

for fruit in fruits:

print(fruit)

2. Проход по строке

word = "Python"

for letter in word:

print(letter)

3. Проход с использованием range()

Функция range() возвращает последовательность чисел, которую можно использовать в циклах for.

3.1

for i in range(5): # от 0 до 4

print(i)

3.2

for i in range(1, 6): # от 1 до 5

print(i)

3.3

for i in range(1, 10, 2): # от 1 до 9 с шагом 2

print(i)

4. Проход по кортежу

colors = ("красный", "зеленый", "синий")

for color in colors:

print(color)

5. Проход по словарю

5.1

person = {"имя": "Иван", "возраст": 25, "город": "Москва"}

for key in person:

print(key, "-", person[key])

5.2

for key, value in person.items():

print(key, "-", value)

6. Проход по множеству

numbers = {1, 2, 3, 4, 5}

for number in numbers:

print(number)

7. Вложенные циклы

Можно использовать цикл for внутри другого цикла for.

for i in range(3):

for j in range(3):

print(i, j)

8. Когда и где использовать цикл for?

1. Обработка списков данных: Если у вас есть список данных (например, имена пользователей, оценки, продукты и т.д.), и вы хотите выполнить определенное действие для каждого элемента, используйте цикл for.
2. Итерация по строкам: Когда необходимо обработать каждый символ в строке.
3. Работа с матрицами и двумерными данными: Вложенные циклы for могут быть полезны для работы с матрицами или двумерными списками.
4. Инициализация данных: Например, когда нужно заполнить список одинаковыми значениями или сгенерировать последовательность чисел.
5. Реализация алгоритмов: Многие алгоритмы, особенно в области обработки данных и математики, требуют последовательного прохода по данным.

**2. Цикл while в Python**

Цикл while позволяет выполнить блок кода до тех пор, пока выполняется определенное условие. Он может быть рассмотрен как итеративный цикл, который зависит от условия, в отличие от цикла for, который итерируется по элементам итерируемой последовательности.

1. Синтаксис:

while условие:

# блок кода для выполнения

2. Базовый пример

count = 0

while count < 5:

print(count)

count += 1

3. Использование else с while

Блок кода в else будет выполнен, когда условие while станет ложным.

count = 0

while count < 5:

print(count)

count += 1

else:

print("Цикл завершен")

4. Бесконечный цикл

Это цикл, который никогда не завершается. Он полезен в сценариях, где программа должна выполняться непрерывно, пока не будет прервана внешним вмешательством.

# while True:

# print("Бесконечный цикл")

# Для выхода из такого цикла вы можете использовать комбинацию клавиш Ctrl+C.

5. Прерывание цикла с помощью break

count = 0

while count < 10:

if count == 5:

break

print(count)

count += 1

6. Пропуск итерации с помощью continue

count = 0

while count < 10:

count += 1

if count % 2 == 0: # пропустить четные числа

continue

print(count)

7. Когда и где использовать цикл while?

1. Неопределенное количество итераций: Когда заранее неизвестно, сколько раз нужно выполнить блок кода, while является идеальным выбором. Например, чтение пользовательского ввода до тех пор, пока не будет получено корректное значение.
2. Симуляции и игры: Циклы while могут быть полезны в симуляциях и играх, где необходимо выполнение кода до тех пор, пока не достигнута какая-либо конечная точка или условие.
3. Мониторинг: Например, мониторинг изменения цен на веб-сайтах или ожидание определенного события в системе.
4. Бесконечные циклы: В некоторых приложениях, таких как серверы или встраиваемые системы, может потребоваться бесконечное выполнение цикла до тех пор, пока программа не будет остановлена внешним образом.

**3. Функция enumerate:**

3.1 Функция enumerate предоставляет удобный способ получить индекс и значение элемента при итерации по коллекции (чаще всего списку).

fruits = ["apple", "banana", "cherry", "date"]

for idx, fruit in enumerate(fruits):

print(idx, fruit)

3.2 Вы также можете задать стартовое значение для индекса с помощью второго аргумента в enumerate:

for idx, fruit in enumerate(fruits, start=1):

print(idx, fruit)

**4. Основы работы со строками:**

4.1 Конкатенация (склеивание):

s1 = 'hello'

s2 = "world"

combined = s1 + ' ' + s2

print(combined) # hello world

4.2 Индексация и срезы:

print(s1[0]) # h

print(s1[-1]) # o

print(s1[1:4]) # ell

4.3 Методы строк: upper() и lower() для изменения регистра:

print(s1.upper()) # HELLO

4.4 `split()` для разбиения строки по разделителю:

words = combined.split(' ')

print(words) # ['hello', 'world']

4.5 - `join()` для объединения списка в строку:

new\_string = '-'.join(words)

print(new\_string) # hello-world

4.6 - `replace()` для замены подстроки:

replaced = s1.replace('llo', 'y')

print(replaced) # hey

**5. Модуль random:**

Модуль random предоставляет функции для генерации случайных чисел, выбора случайных элементов из списков и другие операции.

5.1 Генерация случайного числа:

import random

# случайное число от 0 до 1

print(random.random())

# случайное целое число между a и b

print(random.randint(1, 10))

5.2 Выбор случайного элемента из списка:

choices = ["rock", "paper", "scissors"]

print(random.choice(choices))

5.3 Перемешивание списка:

numbers = [1, 2, 3, 4, 5]

random.shuffle(numbers)

print(numbers)

5.4 Выборка нескольких элементов из списка:

selected = random.sample(numbers, 3)

print(selected)