Циклы

Циклы позволяют выполнять некоторое действие в зависимости от соблюдения некоторого условия. В языке Python есть следующие типы циклов:

- **while**
- **for**

Цикл while

Цикл **while** проверяет истинность некоторого условия, и если условие истинно, то выполняет инструкции цикла. Он имеет следующее формальное определение:

```
while условное_выражение:
    инструкции
```

осле ключевого слова **while** указывается условное выражение, и пока это выражение возвращает значение True, будет выполняться блок инструкций, который идет далее.

Все инструкции, которые относятся к циклу while, располагаются на последующих строках и должны иметь отступ от начала ключевого слова while.

```
number = 1

while number < 5:
    print(f"number = {number}")
    number += 1
print("Работа программы завершена")
```

В данном случае цикл while будет выполняться, пока переменная number меньше 5.

Сам блок цикла состоит из двух инструкций:

```
print(f"number = {number}")
number += 1
```

Обратите внимание, что они имеют отступы от начала оператора while - в данном случае от начала строки. Благодаря этому Python может определить, что они принадлежат циклу. В самом цикле сначала выводится значение переменной number, а потом ей присваивается новое значение. .

Также обратите внимание, что последняя инструкция print("Работа программы завершена") не имеет отступов от начала строки, поэтому она не входит в цикл while.

Весь процесс цикла можно представить следующим образом:

1. Сначала проверяется значение переменной number - больше ли оно 5. И поскольку вначале переменная равна 1, то это условие возвращает True, и поэтому выполняются инструкции цикла
   Инструкции цикла выводят на консоль строку number = 1. И далее значение переменной number увеличивается на единицу - теперь она равна 2. Однократное выполнение блока инструкций цикла называется **итерацией**. То есть таким образом, в цикле выполняется первая итерация.
2. Снова проверяется условие number < 5. Оно по прежнему равно True, так как number = 2, поэтому выполняются инструкции цикла
   Инструкции цикла выводят на консоль строку number = 2. И далее значение переменной number опять увеличивается на единицу - теперь она равна 3. Таким образом, выполняется вторая итерация.
3. Опять проверяется условие number < 5. Оно по прежнему равно True, так как number = 3, поэтому выполняются инструкции цикла
   Инструкции цикла выводят на консоль строку number = 3. И далее значение переменной number опять увеличивается на единицу - теперь она равна 4. То есть выполняется третья итерация.
4. Снова проверяется условие number < 5. Оно по прежнему равно True, так как number = 4, поэтому выполняются инструкции цикла
   Инструкции цикла выводят на консоль строку number = 4. И далее значение переменной number опять увеличивается на единицу - теперь она равна 5. То есть выполняется четвертая итерация.
5. И вновь проверяется условие number < 5. Но теперь оно равно **False**, так как number = 5, поэтому выполняются выход из цикла. Все цикл - завершился. Дальше уже выполняются действия, которые определены

после цикла. Таким образом, данный цикл произведет четыре прохода или четыре итерации

Цикл for

Другой тип циклов представляет конструкция **for**. Этот цикл пробегается по набору значений, помещает каждое значение в переменную, и затем в цикле мы можем с этой переменной производить различные действия. Формальное определение цикла for:

```
for переменная in набор_значений:
    инструкции
```

После ключевого слова **for** идет название переменной, в которую будут помещаться значения. Затем после оператора **in** указывается набор значений и двоеточие.

А со следующей строки располагается блок инструкций цикла, которые также должны иметь отступы от начала цикла.

При выполнении цикла Python последовательно получает все значения из набора и передает их переменную. Когда все значения из набора будут перебраны, цикл завершает свою работу.

В качестве набора значений, например, можно рассматривать строку, которая по сути представляет набор символов. Посмотрим на примере:

```
message = "Hello"

for c in message:
    print(c)
```

В цикле определяется переменную c, после оператора in в качестве перебираемого набора указана переменная message, которая хранит строку "Hello". В итоге цикл for будет перебираеть последовательно все символы из строки message и помещать их в переменную c. Блок самого цикла состоит из одной инструкции, которая выводит значение переменной c на консоль.

Вложенные циклы

Одни циклы внутри себя могут содержать другие циклы. Рассмотрим на примере вывода таблицы умножения:

```
i = 1
j = 1
while i < 10:
    while j < 10:
        print(i * j, end="\t")
        j += 1
    print("\n")
    j = 1
    i += 1
```

Внешний цикл while i < 10: срабатывает 9 раз пока переменная i не станет равна 10. Внутри этого цикла срабатывает внутренний цикл while j < 10:. Внутренний цикл также срабатывает 9 раз пока переменная j не станет равна 10. Причем все 9 итераций внутреннего цикла срабатывают в рамках одной итерации внешнего цикла.

В каждой итерации внутреннего цикла на консоль будет выводится произведение чисел i и j. Затем значение переменной j увеличивается на единицу. Когда внутренний цикл закончил работу, значений переменной j сбрасывается в 1, а значение переменной i увеличивается на единицу и происходит переход к следующей итерации внешнего цикла. И все повторяется, пока переменная i не станет равна 10. Соответственно внутренний цикл сработает всего 81 раз для всех итераций внешнего цикла.

Выход из цикла. break и continue

Для управления циклом мы можем использовать специальные операторы **break** и **continue**. Оператор **break** осуществляет выход из цикла. А оператор **continue** выполняет переход к следующей итерации цикла.

Оператор break может использоваться, если в цикле образуются условия, которые несовместимы с его дальнейшим выполнением. Рассмотрим следующий пример:

```
number = 0
while number < 5:
    number += 1
    if number == 3 :    # если number = 3, выходим из цикла
        break
    print(f"number = {number}")
```

Здесь цикл while проверяет условие number < 5. И пока number не равно 5, предполагается, что значение number будет выводиться на консоль. Однако внутри цикла также проверяется другое условие: if number == 3. То есть, если значение number равно 3, то с помощью оператора **break** выходим из цикла.

В отличие от оператора break оператор **continue** выполняет переход к следующей итерации цикла без его завершения. Например, в предыдущем примере заменим break на continue:

Словари

Словарь (dictionary) в языке Python хранит коллекцию элементов, где каждый элемент имеет уникальный ключ и ассоциированое с ним некоторое значение.

Определение словаря имеет следующий синтаксис:

```
dictionary = { ключ1:значение1, ключ2:значение2, ....}
```

В фигурных скобках через запятую определяется последовательность элементов, где для каждого элемента сначала указывается ключ и через двоеточие его значение.

Определим словарь:

```
users = {1: "Tom", 2: "Bob", 3: "Bill"}
```

В словаре users в качестве ключей используются числа, а в качестве значений - строки. То есть элемент с ключом 1 имеет значение "Tom", элемент с ключом 2 - значение "Bob" и т.д.

Другой пример:

```
emails = {"tom@gmail.com": "Tom", "bob@gmai.com": "Bob", "sam@gmail.com": "Sam"}
```

В словаре emails в качестве ключей используются строки - электронные адреса пользователей и в качестве значений тоже строки - имена пользователей.

Но необязательно ключи и строки должны быть однотипными. Они могу представлять разные типы:

```
objects = {1: "Tom", "2": True, 3: 100.6}
```

Мы можем также вообще определить пустой словарь без элементов:

```
objects = {}
```

Задача 1

Даны два словаря: `dictionary_1 = {'a': 300, 'b': 400}` и `dictionary_2 = {'c': 500, 'd': 600}`. Объедините их в один при помощи встроенных функций языка Python.

Посмотреть решение

Для объединения двух словарей создадим третий словарь в виде копии первого. Для этого используем встроенную функцию `copy()`. Далее к уже созданному словарю мы присоединяем второй словарь. Для этого мы используем встроенную функцию `update()`.

```
1. dictionary_1 = {'a': 100, 'b': 200}
2. dictionary_2 = {'x': 300, 'y': 200}
3. dictionary_3 = dictionary_1.copy()
4. dictionary_3.update(dictionary_2)
5. print(dictionary_3)
```

Задача 2

Дан словарь с числовыми значениями. Необходимо их все перемножить и вывести на экран.

Посмотреть решение

Для решения данной задачи создадим переменную `result`, в которой будем накапливать результаты умножения, а для начала инициируем ее значением `1`. Затем при помощи цикла `for` получим все значения словаря через его ключи. После этого результат умножения из переменной `result` выведем на экран.

```
1. my_dictionary = {'data1': 375, 'data2': 567, 'data3': -37, 'data4': 21}
2. result = 1
3. for key in my_dictionary:
4.     result = result * my_dictionary[key]
5. 
6. print(result)
```

3. Исправьте ошибки в коде, что бы получить требуемый вывод.

```
# данный код
d1 = {"a": 100. "b": 200. "c":300}
d2 = {a: 300, b: 200, d:400}
print(d1["b"] == d2["b"])
# требуемый вывод:
# True
```

4

```
Напишите функцию search_substr(subst, st), которая принимает 2 строки и определяет, име
В случае нахождения подстроки, возвращается фраза «Есть контакт!», а иначе «Мимо!».
Должно быть найдено совпадение независимо от регистра обеих строк.
```

Для решения задания необходимо воспользоваться строковыми методами **lower()** и **find()**. Стоит помнить, что **find()** возвращает -1 в случае ненахождения нужного элемента.

Решение – IDE

```
def search_substr(subst, st):
    if subst.lower() in st.lower():
                return 'Есть контакт!'
    else:
        return 'Мимо!'


# Тесты
print(search_substr('Кол', 'коЛокОл'))
print(search_substr('Колобок', 'колобоК'))
print(search_substr('Кол', 'Плов'))
```