```
>>> for i in 'hello world':
        if i=='o':
                break
        print(i*2, end='')


hheellll
```

Команда else

Слово else, примененное в цикле for или while, проверяет, был ли произведен выход из цикла инструкцией break, или же "естественным" образом. Блок инструкций внутри else выполнится только в том случае, если выход из цикла произошел без помощи break.

```
>>> for i in 'hello world':
        if i=='a':
                break
else:
                print('Буквы a в строке нет')


Буквы a в строке нет
```

Оборудование и материалы.

Персональный компьютер, среда разработки Python.

Указания по технике безопасности:

Соответствуют технике безопасности по работе с компьютерной техникой.

Задания

Повторяем действия с помощью while

Проверки с помощью if, elif и else выполняются последовательно. Иногда нам нужно выполнить какие-то операции более чем один раз. Нам нужен цикл, и простейшим вариантом циклов в Python является while. Попробуйте запустить с помощью интерактивного интерпретатора следующий пример — это простейший цикл, который выводит на экран значения от 1 до 5:

```
>>> while count <= 5:
        print(count)
        count += 1


1
2
3
4
5
```

Сначала мы присваиваем значение 1 переменной count. Цикл while сравнивает значение переменной count с числом 5 и продолжает работу, если значение переменной count меньше или равно 5. Внутри цикла мы выводим значение переменной count, а затем увеличиваем его на 1 с помощью выражения count += 1. Python возвращается к верхушке цикла и снова сравнивает значение переменной count с числом 5. Значение переменной count теперь равно 2, поэтому содержимое цикла while выполняется снова и переменная count увеличивается до 3.

Это продолжается до тех пор, пока переменная count не будет увеличена с 5 до 6 в нижней части цикла. Во время очередного возврата наверх цикла проверка count<= 5 вернет значение False и цикл while закончится. Python перейдет к выполнению следующих строк.

Прерываем цикл с помощью break

Если вы хотите, чтобы цикл выполнялся до тех пор, пока что-то не произойдет, но вы не знаете точно, когда это событие случится, можете воспользоваться бесконечным циклом, содержащим оператор break. В этот раз мы считаем строку с клавиатуры с помощью функции input(), а затем выведем ее на экран, сделав первую букву прописной. Мы прервем цикл, когда будет введена строка, содержащая только букву «q»:

```
>>> while True:
        stuff = input("Строка с заглавной буквы [или q для выхода]: ")
        if stuff == "q":
                break
        print(stuff.capitalize())


Строка с заглавной буквы [или q для выхода]: тест
Тест
Строка с заглавной буквы [или q для выхода]: да, всё работает
Да, всё работает
Строка с заглавной буквы [или q для выхода]: q
```

Пропускаем итерации с помощью continue

Иногда вам нужно не прерывать весь цикл, а только пропустить по какой-то причине одну итерацию. Рассмотрим воображаемый пример: считаем целое число, выведем на экран его значение в квадрате, если оно нечетное, и пропустим его, если оно четное. И вновь для выхода из цикла используем строку "q":

```
>>> while True:
        value = input("Целое, пожалуйста [или q для выхода]: ")
        if value == 'q':
                break
        number = int(value)
        if number % 2 == 0:
                continue
        print(number, "квадрат = ", number*number)


Целое, пожалуйста [или q для выхода]: 2
Целое, пожалуйста [или q для выхода]: 5
5 квадрат =  25
Целое, пожалуйста [или q для выхода]: 3
3 квадрат =  9
Целое, пожалуйста [или q для выхода]: 80
Целое, пожалуйста [или q для выхода]: q
>>>
```

Проверяем, завершился ли цикл заранее, с помощью else

Если цикл while завершился нормально (без вызова break), управление передается в опциональный блок else. Вы можете использовать его в цикле, где выполняете некоторую проверку и прерываете цикл, как только проверка успешно выполняется. Блок else выполнится в том случае, если цикл while будет пройден полностью, но искомый объект не будет найден:

```
>>> numbers = [1, 3, 5]
>>> position = 0
>>> while position < len(numbers):
        number = numbers[position]
        if number % 2 == 0:
                print('Найдено чётное число: ', number)
                break
        position +=1
else: #Выхода не произошло
        print('Чётное число в списке отсутствует')


Чётное число в списке отсутствует
```

Выполняем итерации с помощью for

В Python итераторы часто используются по одной простой причине. Они позволяют вам проходить структуры данных, не зная, насколько эти структуры велики и как реализованы. Вы даже можете пройти по данным, которые были созданы во время работы программы, что позволяет обработать потоки данных, которые в противном случае не поместились бы в память компьютера.

Вполне возможно пройти по последовательности таким образом:

```
>>> hobbits = ['Фродо', 'Сэм', 'Мерри', 'Пин']
>>> current = 0
>>> while current < len(hobbits):
        print(hobbits[current])
        current +=1


Фродо
Сэм
Мерри
Пин
```

Однако существует более характерный для Python способ решения этой задачи:

```
>>> for hobbit in hobbits:
        print(hobbit)


Фродо
Сэм
Мерри
Пин
```

Списки вроде rabbits являются одними из итерабельных объектов в Python наряду со строками, кортежами, словарями и некоторыми другими элементами. Итерирование по кортежу или списку возвращает один элемент за раз. Итерирование по строке возвращает один символ за раз, как показано здесь:

```
>>> word = 'котэ'
>>> for letter in word:
        print(letter)


к
о
т
э
```

Итерирование по словарю (или его функции keys()) возвращает ключи. В этом примере ключи являются номерами пальцев на руках:

```
>>> fingers = {'Первый' : 'Большой', 'Второй' : 'Указательный', 'Третий' :
'Средний', 'Четвёртый' : 'Безымянный', 'Пятый' : 'Мизинец'}
>>> for finger in fingers: # или for finger in fingers.keys():
        print(finger)


Первый
Второй
Третий
Четвёртый
Пятый
>>>
```

Чтобы итерировать по значениям, а не по ключам, следует использовать функцию values():

```
>>> for value in fingers.values():
        print(value)


Большой
Указательный
Средний
Безымянный
Мизинец
```

Чтобы вернуть как ключ, так и значение словаря, вы можете использовать функцию items():

```
>>> for item in fingers.items():
        print(item)


('Первый', 'Большой')
('Второй', 'Указательный')
('Третий', 'Средний')
('Четвёртый', 'Безымянный')
('Пятый', 'Мизинец')
>>>
```

Помните, что можете присвоить значение кортежу за один шаг. Для каждого кортежа, возвращенного функцией items(), присвойте первое значение (ключ) переменной card, а второе (значение) — переменной contents:

```
>>> for number, finger in fingers.items():
        print('Палец ', number, 'называется', finger)


Палец  Первый называется Большой
Палец  Второй называется Указательный
Палец  Третий называется Средний
Палец  Четвёртый называется Безымянный
Палец  Пятый называется Мизинец
```

Прерываем цикл с помощью break

Ключевое слово break в цикле for прерывает этот цикл точно так же, как и цикл while.

Пропускаем итерации с помощью continue

Добавление ключевого слова continue в цикл for позволяет перейти на следующую итерацию цикла, как и в случае с циклом while.

Проверяем, завершился ли цикл заранее, с помощью else

Как и в цикле while, в for имеется опциональный блок else, который проверяет, выполнился ли цикл for полностью. Если ключевое слово break не было вызвано, будет выполнен блок else.

Это полезно, если вам нужно убедиться в том, что предыдущий цикл выполнился полностью, вместо того чтобы рано прерваться. Цикл for в следующем примере выводит на экран название сыра и прерывается, если сыра в магазине не найдется:

```
>>> cheeses = []
>>> for cheese in cheeses:
        print('Этот магазин имеет отличный ', cheese)
        break
else: # Отсутствие прерывания означает, что сыр не найден
        print('Сыра в этом магазине нет, не так ли?')


Сыра в этом магазине нет, не так ли?
>>>
```

> Примечание:
>
> Как и в цикле while, в цикле for использование блока else может показаться нелогичным. Можно рассматривать цикл for как поиск чего-то, в таком случае else будет вызываться, если вы ничего не нашли. Чтобы получить тот же эффект без блока else, используйте переменную, которая будет показывать, нашелся ли искомый элемент в цикле for, как здесь:
>
> ```
> >>> cheeses = []
> >>> found_one = False
> >>> for cheese in cheeses:
>         found_one = True
>         print('В этом магазине есть отличный сыр ', cheese)
>         break
>
> >>> if not found_one:
>         print('Наверное, это не продуктовый магазин.')
>
>
> Наверное, это не продуктовый магазин.
> >>>
> ```

<u>Контрольные вопросы</u>

1. Для чего используются циклы?
2. Опишите особенности применения цикла while.
3. В чём разница между elifи else?
4. Для чего используется оператор break?
5. Для чего используется оператор continue?
6. Как можно проверить, завершился ли цикл досрочно?
7. В чём особенности синтаксиса цикла for?
8. Как выполнить итерирование по словарю? Как вывести ключи?
9. Как выполнить итерирование словаря и вывести его значения?
10. Как вывести и ключи и значения словаря?
11. Как используется прерывание в цикле for? Для чего?

Список литературы, рекомендуемый к использованию по данной теме:

1. Орлов, С. А. Программная инженерия. Технологии разработки программного обеспечения : учебник / С.А. Орлов. - 5-е изд., обновл. и доп. - СПб. : Питер, 2017. - 640 с.
2. Гагарина, Л. Г. Современные проблемы информатики и вычислительной техники : [учеб. пособие] / Л.Г. Гагарина, А.А. Петров. - М. : Форум, 2016. - 368 с.
3. Шкляр, М. Ф. Основы научных исследований : учеб. пособие / М.Ф. Шкляр. - 6-е изд. - М. : Дашков и Ко, 2016. - 208 с.
4. Михеева, Е. В. Информационные технологии в профессиональной деятельности : учеб. пособие / Е.В. Михеева. - 14-е изд., стер. - М. : Академия, 2016. - 384 с.
5. Любанович Билл. Простой Python. Современный стиль программирования. — СПб.: Питер, 2016. — 480 с.

Лабораторная работа 7. Генерирование числовых последовательностей

Цель работы:Научиться генерировать и итерировать числовые последовательности.Изучить итерирование по нескольким последовательностям. Генерирование числовых последовательностей с помощью функции range(). Включения списков и словарей, множества и генератора.

Компетенции:

Код	Формулировка:
ОК-5	быть готовым работать с информацией в различных формах, использовать для ее получения, обработки, передачи, хранения и защиты современные компьютерные технологии
ОПК-5	способностью использовать основные приемы обработки и представления экспериментальных данных
ОПК-6	способность осуществлять поиск, хранение, обработку и анализ информации из различных источников и баз данных, представлять ее в требуемом формате с использованием информационных, компьютерных и сетевых технологий