```
>>> for i in b:
        for j in i:
            c.append(j)


>>> c
[1, 10, 2, 20, 3, 30]
```

В конец генератора можно добавлять конструкцию **if**. Например, надо из строки извлечь все цифры:

```
>>> a = "lsj94ksd231 9"
>>> b = [int(i) for i in a if '0'<=i<='9']
>>> b
[9, 4, 2, 3, 1, 9]
```

Или заполнить список числами, кратными 30 или 31:

```
>>> a = [i for i in range(30,250) if i%30==0 or i%31==0]
>>> a
[30, 31, 60, 62, 90, 93, 120, 124, 150, 155, 180, 186, 210, 217, 240, 248]
```

Таким образом, генераторы позволяют создавать списки легче и быстрее. Однако заменить ими достаточно сложные конструкции не получится. Например, когда условие проверки должно включать ветку **else**.

Оборудование и материалы.

Персональный компьютер, среда разработки Python.

Указания по технике безопасности:

Соответствуют технике безопасности по работе с компьютерной техникой.

Задания

Итерирование по нескольким последовательностям с помощью функции zip().

Существует еще один полезный приём – параллельное итерирование по нескольким последовательностям с помощью функции zip().

```
>>> days = ['Понедельник', 'Вторник', 'Среда']
>>> fruits = ['банан', 'апельсин', 'персик']
>>> drinks = ['кофе', 'чай', 'пиво']
>>> desserts = ['тирамису', 'мороженое', 'пирог', 'пудинг']
>>> for day, fruit, drink, dessert in zip(days, fruits, drinks, desserts):
        print(day, ': пей ', drink, 'ешь ', fruit, 'дегустируй ', dessert)


Понедельник : пей  кофе ешь  банан дегустируй  тирамису
Вторник : пей  чай ешь  апельсин дегустируй  мороженое
Среда : пей  пиво ешь  персик дегустируй  пирог
```

Функция zip() прекращает свою работу, когда выполняется самая короткая последовательность. Один из списков (desserts) оказался длиннее остальных, поэтому никто не получит пудинг, пока мы не увеличим остальные списки.

При работе со словарями мы видели, как с помощью функции dict() можно создавать словари из последовательностей, содержащих два элемента, вроде кортежей, списков или строк. Вы можете использовать функцию zip(), чтобыпройти по нескольким последовательностям и создать кортежи из элементов с одинаковыми смещениями. Создадим два кортежа из соответствующих друг другу английских и французских слов:

```
>>> english = 'Monday', 'Tuesday', 'Wednesday'
>>> french = 'Lundi', 'Mardi', 'Mercredi'
```

Теперь используем функцию zip(), чтобы объединить эти кортежи в пару. Значение, возвращаемое функцией zip(), само по себе не является списком или кортежем, но его можно преобразовать в любую из этих последовательностей:

```
>>> list(zip(english,french))
[('Monday', 'Lundi'), ('Tuesday', 'Mardi'), ('Wednesday', 'Mercredi')]
```

Передайте результат работы функции zip() непосредственно функции dict() — и у нас готов небольшой англо-французский словарь!

```
>>> dict(zip(english, french))
{'Monday': 'Lundi', 'Tuesday': 'Mardi', 'Wednesday': 'Mercredi'}
```

Генерирование числовых последовательностей с помощью функции range()

Функция range() возвращает поток чисел в заданном диапазоне без необходимости создавать и сохранять крупную структуру данных вроде списка или кортежа.

Это позволяет вам создавать большие диапазоны, не использовав всю память компьютера и не обрушив программу. Вы можете применять функцию range() аналогично slice(): range(start, stop, step). Если опустите значение start, диапазон начнется с 0. Необходимым является лишь значение stop: как и в случае со slice(), оно определяет последнее значение, которое будет создано прямо перед остановкой функции. Значение по умолчанию step равно 1, но вы можете изменить его на -1.

Как и zip(), функция range() возвращает итерабельный объект, поэтому вам нужно пройти по значениям с помощью конструкции for … in или преобразовать объект в последовательность вроде списка. Создадим диапазон 0, 1, 2:

```
>>> for x in range(0,3):
        print(x)


0
1
2
>>> list(range(0,3))
[0, 1, 2]
>>>
```

Вот так можно создать диапазон от 2 до 0:

```
>>> for x in range(2, -1, -1):
        print(x)


2
1
0
>>> list(range(2, -1, -1))
[2, 1, 0]
```

В следующем фрагменте кода используется шаг 2, чтобы получить все четные числа от 0 до 10

```
>>> list(range(0,11,2))
[0, 2, 4, 6, 8, 10]
```

Включения

Включение — это компактный способ создать структуру данных из одного или более итераторов. Включения позволяют вам объединять циклы и условные проверки, не

используя при этом громоздкий синтаксис. Если вы применяете включение, то можно сказать, что уже неплохо знаете Python. Иными словами, это одна из характерных особенностей данного языка.

Включение списков

Вы можете создать список целых чисел от 1 до 5, добавляя их туда по одному за раз, например, так:

```
>>> number_list = []
>>> number_list.append(1)
>>> number_list.append(2)
>>> number_list.append(3)
>>> number_list.append(4)
>>> number_list.append(5)
>>> number_list
[1, 2, 3, 4, 5]
```

Или же вы могли бы использовать итератор и функцию range():

```
>>> for number in range(1,6):
        number_list.append(number)


>>> number_list
[1, 2, 3, 4, 5]
```

Или же преобразовать в список сам результат работы функции range():

```
>>> number_list = list(range(1,6))
>>> number_list
[1, 2, 3, 4, 5]
```

Все эти подходы абсолютно корректны с точки зрения Python и сгенерируют одинаковый результат. Однако более характерным для Python является создание списка с помощью включения списка. Простейшая форма такого включения выглядит так:

[выражение for элемент in итерабельный объект]

Вот так выглядит включение списка целых чисел

```
>>> number_list = [number for number in range(1,6)]
>>> number_list
[1, 2, 3, 4, 5]
```

В первой строке вам нужно, чтобы первая переменная *number* сформировала значения для списка: следует разместить результат работы цикла в переменной number_list. Вторая переменная number является частью цикла for. Чтобы показать, что первая переменная number является выражением, попробуем такой вариант:

```
>>> number_list = [number-1 for number in range(1,6)]
>>> number_list
[0, 1, 2, 3, 4]
```

Включение списка перемещает цикл в квадратные скобки. Этот пример включения ненамного проще предыдущего, но это еще не все. Включение списка может содержать условное выражение, которое выглядит примерно так:

[выражение for элемент in итерабельныйобъект if условие]

Создадим новое включение, которое создает список, состоящий только из нечетных чисел, расположенных в диапазоне от 1 до 5:

```
>>> a_list = [number for number in range(1,6) if number % 2 == 1]
>>> a_list
[1, 3, 5]
```

Теперь включение выглядит чуть более компактно, чем его традиционный аналог:

```
>>> a_list = []
>>> for number in range(1,6):
        if number % 2 == 1:
                a_list.append(number)


>>> a_list
[1, 3, 5]
```

Наконец, точно так же, как и в случае вложенных циклов, можно написать более чем один набор операторов for ... в соответствующем выделении. Чтобы продемонстрировать это, сначала создадим старый добрый вложенный цикл и выведем на экран результат:

```
>>> rows = range(1,4)
>>> cols = range(1,3)
>>> for row in rows:
        for col in cols:
                print(row, col)


1 1
1 2
2 1
2 2
3 1
3 2
```

Теперь воспользуемся включением и присвоим его переменной cells, создавая тем самым список кортежей (row, col):

```
>>> rows = range(1,4)
>>> cols = range(1,3)
>>> cells = [(row, col) for row in rows for col in cols]
>>> for cell in cells:
        print(cell)


(1, 1)
(1, 2)
(2, 1)
(2, 2)
(3, 1)
(3, 2)
```

Кстати, вы можете воспользоваться распаковкой кортежа, чтобы выдернуть значения row и col из каждого кортежа по мере итерирования по списку cells:

```
>>> for row, col in cells:
        print(row, col)


1 1
1 2
2 1
2 2
3 1
3 2
```

Фрагменты for row ... и for col ... во включении также могут иметь свои проверки if.

Включение словаря

Для словарей также можно создать включение. Простейшая его форма выглядит привычно:

{ выражение_ключа: выражение_значения for выражение in итерабельный объект }

Как и в случае с включениями списка, выделения словарей также имеют проверки if и несколько операторов for:

```
>>> word = 'телеграмма'
>>> letter_counts = {letter : word.count(letter) for letter in word}
>>> letter_counts
{'т': 1, 'е': 2, 'л': 1, 'г': 1, 'р': 1, 'а': 2, 'м': 2}
```

Мы запускаем цикл, проходя по каждой из семи букв в строке letters, и считаем, сколько раз появляется эта буква. Два наших вызова word.count(letter) — это лишь пустая трата времени, поскольку нам нужно подсчитать буквы «е», «а» и «м» два раза. Но когда мы считаем буквы «е» во второй раз, то не причиняем вреда, поскольку лишь заменяем уже существующую запись в словаре; то же относится и к подсчету других повторяющихся букв. Следующий способ решения задачи более характерен для Python:

```
>>> word = 'телеграмма'
>>> letter_counts = {letter : word.count(letter) for letter in set(word)}
>>> letter_counts
{'а': 2, 'р': 1, 'м': 2, 'г': 1, 'е': 2, 'т': 1, 'л': 1}
```

Ключи словаря располагаются в ином, чем в предыдущем примере, порядке, поскольку итерирование по результату работы функции set(word) возвращает буквы в другом порядке, нежели итерирование по строке word.

Включение множества

Никто не хочет оказаться обиженным, поэтому даже у множеств есть включения.

Простейшая версия выглядит как включение списка или словаря, которые вы только что видели:

{ выражение for выражение in итерабельный объект}

Более длинные версии (проверки if, множественные операторы for) также доступны для множеств:

```
>>> a_set = {number for number in range(1,6) if number%3==1}
>>> a_set
{1, 4}
```

Включение генератора

Для кортежей не существует включений. Вы могли подумать, что замена квадратных скобок у выделения списка на круглые создаст включение кортежа. Может даже показаться, что это работает, поскольку исключение не будет сгенерировано, если вы напишете следующее:

```
>>> number_thing = (number for number in range(1,6))
```

В круглые скобки заключено включение генератора, оно возвращает объект генератора:

```
>>> type(number_thing)
<class 'generator'>
```

Сами генераторы мы рассмотрим более детально позже в данной лабораторной работе. Применение генераторов — это один из способов предоставить данные итератору.

Вы можете итерировать непосредственно по этому объекту генератора, как показано здесь:

```
>>> for number in number_thing:
        print(number)


1
2
3
4
5
```

Или же вы можете обернуть вызов list() вокруг включения генератора, чтобы заставить его работать как включение списка:

```
>>> number_thing = (number for number in range(1,6))
>>> number_list = list(number_thing)
>>> number_list
[1, 2, 3, 4, 5]
```

> Генератор может быть запущен лишь однажды. Списки, множества и словари существуют в памяти, но генератор создает свои значения во время работы программы и выдает их по одному за раз через итератор. Он не запоминает их, поэтому вы не можете перезапустить или создать резервную копию генератора.

Если вы попробуете проитерировать по генератору заново, то обнаружите, что он истощен:

```
>>> try_again = list(number_thing)
>>> try_again
[]
```

Вы можете создать генератор из включения генератора, как мы сделали это здесь, или из функции генератора. Сначала мы поговорим о функциях в целом, а затем рассмотрим частный случай — функции генератора.

Контрольные вопросы

1. Генерация и итерирование последовательностей.
2. Итерирование по нескольким последовательностям.
3. Генерирование числовых последовательностей с помощью функции range().
4. Включения списков и словарей, множества и генератора.
5. Приведите примеры применения генерации последовательностей.

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.