

ЛАБОРАТОРНЫЙ ПРАКТИКУМ ПО ПРОГРАММИРОВАНИЮ НА ЯЗЫКЕ PYTHON (Python Beginning)

*«Теперь он и тебя сосчитал» - сказал телёнок своей маме.
(Из сказки Альфа Трёйсена «Козлёнок, который умел считать до десяти»)*

Урок 4. (Lesson 4) Основы программирования

УПРАВЛЯЮЩИЕ КОНСТРУКЦИИ. ОПЕРАТОРЫ

Условный оператор

Прежде чем говорить об условном операторе в *Python*, необходимо вспомнить особенности условного оператора – оператора выбора в *C++*: управляющее выражение в данном операторе приводится к логическому типу – *bool*. Данный факт требует от нас учитывать правила вычисления типа результата выражения, основанные на выборе «*старшего*» типа операнда, участвующего в выражении (требования языков *C/C++* и *Python*).

```
>>> True; type(True)
True
<class 'bool'>
>>> True + 2; type(True + 2)
3
<class 'int'>
>>> False; type(False)
False
<class 'bool'>
>>> False + 3; type(False + 3)
3
<class 'int'>
>>> val_1 = 10; val_2 = 12; val_1 > val_2; type(val_1 > val_2)
False
<class 'bool'>
>>>
```

Но в механизмах, реализованных в *Python*, есть свои особенности, связанные с приведением к логическому типу вещественных значений.

```
>>> bool(1); bool(-1); bool(0.5); bool(1.5)
True
True
True
True
>>> bool(0.0001); bool(0.0)
True
False
>>>
```

Исходя из знаний, полученных при изучении C/C++, можно было бы предположить, что результат вычисления выражения `bool(0.0001)` будет равен `False`, но мы видим обратное – результат равен `True` (истина). То есть в *Python* не формируется цепочка преобразований: $any_type \rightarrow int \rightarrow bool$, а сразу выполняется преобразование $any_type \rightarrow bool$. По факту, выполняется сравнение – анализируемое значение отлично от нуля. И не важно, к какому типу оно относится, к вещественному или целочисленному – значение, эквивалентное `True`, должно быть отлично от нуля (для скалярных типов данных), либо, если оно относится к ссылочному типу, быть не пустым.

Вернемся к оператору выбора `if...else`. Сразу замечу, что разработчики *Python* объединили оператор выбора и оператор множественного выбора, отказавшись от реализации отдельного оператора `switch...case`. Рассмотрим синтаксис оператора выбора `if...else`.

```
if <Управляющее_выражение>:
    <Блок_выражений, выполняемых_при_выполнении_условия>
[elif <Управляющее_выражение>:
    <Блок_выражений, выполняемых_при_выполнении_условия>
]
[else:
    <Блок_выражений, выполняемых,_если_условие_ложно>]
```

Как уже было сказано ранее, значение операнда `<Управляющее_выражение>` анализируется после приведения его к логическому типу. Формально ограничений на количество «альтернативных» ветвей у оператора выбора нет. Рассмотрим следующие примеры:

```
# формальная процедура ввода PIN-кода
pin_code = 98765 # значение, сохраненное в системе
choice = int(input("Введите PIN-код: "))
if choice == pin_code:
    print("Отлично/OK")
    # блок процедур, связанных с успешной авторизацией
else:
    print("Ошибка/Access denied")
    # блок процедур, связанных с обработкой ошибки авторизации

# Определение типа корней квадратного уравнения
print("Введите значения коэффициентов квадратного уравнения")
A = float(input("A: "))
B = float(input("B: "))
C = float(input("C: "))
D = B * B - 4 * A * C # Вычисление значения дискриминанта
if D > 0:
    print("Уравнение имеет действительные корни")
elif D == 0:
    print("Уравнение имеет единственный действительный корень")
else:
```

```
print("Уравнение имеет комплексно сопряженные корни")

# Проверка на четное/нечетное
numb = int(input("Введите целочисленное значение: "))
if numb % 2 == 0:
    print("Введено четное значение")
else:
    print("Введено нечетное значение")
```

При составлении инструкций оператора выбора необходимо учитывать, какие значения и каких типов приводятся к значению *True*, а какие – к *False*.

```
>>> print(bool(""), bool(None), bool(()))
False False False
>>> arr1 = []; arr2 = [1,2]; str1 = "Hello"
>>> print(bool(arr1), bool(arr2), bool(str1))
False True True
>>>
```

Рассмотрим еще несколько модифицированных примеров из учебника Н.А. Прохоренок [3] (*неплохой учебник для быстрого ознакомления с языком*).

```
# -*- coding: cp1251 -*-
"""Скрипт для проведения опроса о предрасположенности пользователя"""
print("""Какой операционной системой вы пользуетесь?
1 - Windows 7
2 - Windows 8.x
3 - Windows 10
4 - Ubuntu Linux
5 - ZentOS Linux
6 - Android
7 - Другая UNIX/Linux - платформа""")
os = input("Введите число, соответствующее вашему ответу: ")
print("-----")
if os == "1":
    print("Вы выбрали - Windows 7. Стабильная, но не поддерживаемая ОС")
elif os == "2":
    print("Вы выбрали - Windows 8.x. А когда на 'десятку' перейдете?")
elif os == "3":
    print("Вы выбрали - Windows 10. Скоро апгрейд...")
elif os == "4":
    print("Вы выбрали - Ubuntu Linux. Выбор, достойный тинейджера!")
elif os == "5":
    print("Вы выбрали - ZentOS Linux. Похвально.")
elif os == "6":
    print("Вы выбрали Android. Это не шутка, я работаю под Android?")
elif os == "7":
    print("Вы выбрали UNIX/Linux - платформу. Bravo!")
```

```
else:  
    print("Вы даже не знаете, какая у Вас установлена ОС...")
```

Несколько модифицированный вариант данного примера, использующий вложенный оператор выбора для обработки ситуации, когда пользователь ничего не ввел, а просто нажал клавишу *<Enter>* – ввел пустую строку.

```
# -*- coding: cp1251 -*-  
"""Скрипт для проведения опроса о предрасположенности прозователя"""  
print("""Какой операционной системой вы пользуетесь?  
1 - Windows 7  
2 - Windows 8.x  
3 - Windows 10  
4 - Ubuntu Linux  
5 - ZentOS Linux  
6 - Android  
7 - Другая UNIX/Linux - платформа""")  
os = input("Введите число, соответствующее вашему ответу: ")  
print("-----")  
if os != "":  
    if os == "1":  
        print("Вы выбрали - Windows 7. Стабильная, но не  
поддерживаемая ОС")  
    elif os == "2":  
        print("Вы выбрали - Windows 8.x. А когда на 'десятку'  
перейдете?")  
    elif os == "3":  
        print("Вы выбрали - Windows 10. Скоро апгрейд...")  
    elif os == "4":  
        print("Вы выбрали - Ubuntu Linux. Выбор, достойный  
тинеjdжера!")  
    elif os == "5":  
        print("Вы выбрали - ZentOS Linux. Похвально.")  
    elif os == "6":  
        print("Вы выбрали Android. Это не шутка, я работаю под  
Android?")  
    elif os == "7":  
        print("Вы выбрали UNIX/Linux - платформу. Bravo!")  
    else:  
        print("Вы даже не знаете, какая у Вас установлена ОС...")  
else:  
    print("Вы ничего не ввели. Однако...")
```

Оператор выбора поддерживает формат, формально соответствующий оператору выбора из языка *ANSI C*:

```
<Переменная> = <Если_истина> if <Условие> else <Если_ложь>
```

Задачи

Для закрепления материала решите несколько простых задач.

1. Составьте программу для нахождения корней квадратного уравнения. Программа должна находить не только действительные, но и комплексно сопряженные значения корней.
2. Составьте программу для определения существования треугольника по значениям длин его трех сторон.
3. Составьте программу для определения того, в какой четверти декартовой системы координат находится точка, координаты которой заданы в полярной системе координат.
4. Составьте программу для нахождения площади треугольника, заданного координатами своих вершин.
5. Составьте программу для определения принадлежности точки кругу. Данные координат и величина радиуса вводятся в декартовой системе координат.
6. Составить программу для определения количества дней в году.
7. Составить программу для определения количества дней в месяце по введенной пользователем дате. Количество дней в феврале должно формироваться с учетом високосного года.
8. Составить программу для определения количества разрядов во введенном пользователе положительном числе. Предполагается, что введенное значение может находиться в интервале $[-999; +999]$. Метод `len()` не использовать! Кроме того, программа должна сообщать о том, является число положительным или отрицательным.
9. Составить программу «Угадай число» с использованием инструкции `if...else`, реализующую *метод двоичного поиска*.
10. Составить программу для определения того, можно ли переместить контейнер, являющийся кубоидом (прямоугольным параллелепипедом), для которого заданы длины его граней – a_1, b_1, c_1 , в прямоугольный проём ангара, заданного длинами сторон – a_2, b_2 .

Оператор цикла

Язык программирования *Python* поддерживает только два вида циклов: `for` и `while`. Цикл с предусловием `do...while` в *Python* не реализован.

Цикл `for`

Цикл `for` в языке *Python* не является обычным пошаговым циклом, заимствованным из *ANSI C*. Он является измененной реализацией обобщенного алгоритма `std::for_each` из стандартной библиотеки языка *C++*. Изменения касаются внедрения в оператор блока обработки событий, связанных с использованием операторов `break` и `continue`. Синтаксис оператора `for`.

```
for <Текущий_элемент> in <Последовательность>:
    <Операторы_тела_цикла>
[else:
    <Блок,_выполняемые,_если_был_использован_оператор_continue>
]
```

Основные компоненты оператора *for*: *<Последовательность>* - объект, поддерживающий механизм итерации; *<Текущий_элемент>* - объект, доступный на каждой итерации цикла. Как и обобщенный алгоритм *std::for_each*, цикл *for* использует итераторы для доступа к элементам контейнера, определенного компонентой *<Последовательность>*. Но кроме контейнеров, таких, как сточки, списки, словари и кортежи, этот объект может быть определен при помощи генератора *range()*. Рассмотрим следующие примеры.

```
>>> for i in range(1,10,2):
    print(i, end=' ')

1 3 5 7 9                                # Результат выполнения цикла
>>>
>>> for i in "Hello World": print(i*3, end='')

HHHeeeelllllllooo   WWWooorrrrrlllddd   # Результат выполнения цикла
>>>
>>> sum_numb = 0 # начальное значение суммы
>>> for i in [1,2,3,4,5,6,7,8,9]:
    sum_numb += i # нахождение суммы элементов последовательности

>>> sum_numb # вывод результата
45
>>>
>>> arr = [1,2,3,4,-5,6,7,8,-9]
>>> for i in arr: print(i, end = ' ')

1 2 3 4 -5 6 7 8 -9 # Результат выполнения цикла
>>> for i in arr:
    if i < 0: break
    print(i, end = ' ')

1 2 3 4
>>> for i in arr:
    if i < 0: break
    print(i, end = ' ')
else:
    print("negative count")
```

```
1 2 3 4 # Результат выполнения цикла
>>> for i in arr:
    if i < 0: continue
    print(i, end = ' ')
else:
    print("Continue")
```

```
1 2 3 4 6 7 8 Continue # Результат выполнения цикла
>>>
```

Обратите внимание, в версии *Python* 3.5 блок *else* цикла *for* выполняется только в том случае, когда внутри цикла было прерывание итерации, а не самого цикла – был использован оператор *continue*. А информацию о том, что именно привело к исполнению оператора *continue*, – контекст события, должен формировать сам программист (в более ранних версиях языка данная ветвь цикла использовалась для обработки событий, связанных с досрочным прерыванием цикла по *break*)! Это видно на следующем примере.

```
>>> arr = [1,2,3,-4,5]
>>> for i in arr:
    if i < 0: break
    print(i, end = ' ')
else: print("Negative item")

1 2 3
>>> for i in arr:
    if i < 0: continue
    print(i, end = ' ')
else: print("Negative item", i)

1 2 3 5 Negative item 5
>>>
```

В последнем цикле в ветви *else* выведено значение *<Текущего_элемента>*, соответствующее последней успешной итерации цикла.

Задача. Составьте программу, предназначенную для обработки данных, получаемых от цифрового бытового медицинского термометра, предназначенного для получения данных температуры человека – пациента. Программа «читает» данные из буфера датчика температуры и «передает» их блоку отображения на экране. Диапазон температуры тела живого пациента находится в интервале [33.0;41.9]. В случае аппаратной ошибки датчик термометра может выдавать ошибочное значение. Для обработки ошибочных данных должна использоваться стратегия – «мусор на входе, на выходе – допустимое значение», заключающаяся в том, что вместо некорректных данных для отображения должны передаваться последние корректные данные.

Объекты-генераторы

- Функция `range()`

Объект-функция `range()` (встроенная функция – *built-in function of Python*) на каждой итерации цикла возвращает целочисленное значение (для этого внутри класса `range` определены методы `__next__()` и `__call__()`). Синтаксис функции `range()`:

```
range([<start=0>,] <end> [, <step=1>])
```

Примеры использования функции `range()`:

```
>>> range(10) # функция-генератор вне цикла, сама по себе...
range(0, 10) # выведена полная информация об объекте-функции
>>> # дополнена информацией о значении первого параметра
>>> type(range(10))
<class 'range'> # функция возвращает объект – экземпляр класса range
>>> # это именно функтор, функция-объект
>>> for i in range(-5, 25, 3):
    print(i, end = ' ')

-5 -2 1 4 7 10 13 16 19 22
>>>
```

Сразу замечу, что работа функции `range()` для версии Python 3.x и для 2.x существенно различается. Данный факт необходимо учитывать, так как во многих UNIX/Linux-системах по умолчанию установлен Python именно версии 2.x!

Как видно из приведенного примера, параметры `<start>` и `<end>` определяют границы интервала `[<start>; <end>)`, из которого с шагом, определенным параметром `<step>`, извлекаются целочисленные значения. Величина шага может быть только целочисленной! Функция-генератор `range()` возвращает только целочисленные значения, если вам нужно сгенерировать вещественные значения, то используйте дополнительные преобразования. Например:

```
>>> [i * 0.1 for i in range(1,11)]
[0.1, 0.2, 0.30000000000000004, 0.4, 0.5, 0.6000000000000001,
0.7000000000000001, 0.8, 0.9, 1.0]
>>>
>>> for i in range(1,11): print(i * 0.1, end = ' ')

0.1 0.2 0.30000000000000004 0.4 0.5 0.6000000000000001
0.7000000000000001 0.8 0.9 1.0
>>>
```



```
>>> range(1,11,0.1) # все-таки попробуем с вещественным шагом...
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    range(1,11,0.1)
TypeError: 'float' object cannot be interpreted as an integer
>>>
```

Если вам вдруг все-таки потребуется нечто, вроде функции-генератора, возвращающей значения с плавающей точкой, можете воспользоваться кодом следующего примера [7].

```
>>> def frange(start,stop,step): # объявление функции
    i = start # величина шага
    while i < stop: # организация внутреннего цикла
        yield i # механизм возврата значения из функции-генератора
        i += step

>>> for i in frange(0.0, 1.1, 0.1):
    print(i, end = ' ')
```

```
0.0 0.1 0.2 0.30000000000000004 0.4 0.5 0.6 0.7 0.7999999999999999
0.8999999999999999 0.9999999999999999 1.0999999999999999
>>> list(frange(0,5,0.25))
[0, 0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 1.75, 2.0, 2.25, 2.5, 2.75, 3.0,
3.25, 3.5, 3.75, 4.0, 4.25, 4.5, 4.75]
>>>
```

Пределы значений параметров генератора определяются параметрами константы `sys.maxsize` (вспомните библиотеку `<limits>` языка C++).

Преимущество данного объекта при организации циклов по сравнению с обычным списком или кортежем заключается в том, что он всегда занимает в фиксированный объем памяти вне зависимости от длины диапазона генерируемых значений [4]. Генераторы и выражения-генераторы являются той частью языка, которая определяет «синтаксический сахар». За более подробной информацией по данной теме обращайтесь к справочным руководствам [4, 5, 6].

Функция-объект `range()` может использоваться и для инициализации элементов создаваемого списка.

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(-5, 6, 2))
[-5, -3, -1, 1, 3, 5]
>>>
>>> arr1 = [1,2,34,17,-2,0.123,0.4,0]
>>> arr2 = list(range(len(arr1))); arr2
[0, 1, 2, 3, 4, 5, 6, 7]
```

```
>>> # прикольная ситуация со значением правой границы генератора...
>>> list(range(0))      # результат - пустой список
[]
>>> list(range(1,0))    # результат - пустой список
[]
>>> list(range(10,0))   # результат - пустой список
[]
>>>
```

Например, если при обработке последовательности данных измерений вам потребуются и значения индексов этих значений (номера отсчетов при измерении), но сделать это требуется наиболее эффективным с точки зрения производительности образом, вот тогда вы и используете выражение-генератор `list(range(len(<Последовательность>)))`.

Еще одним подтверждением того, что `range()` это объект является наличие у него двух методов: `index()` и `count()`.

Метод `index(<Значение>)` возвращает индекс элемента, имеющего указанное значение. Если указанное значение не входит в объект, возбуждается исключение класса `ValueError`.

```
>>> obj = range(-10, 11, 2)
>>> obj.index(-10), obj.index(2)
(0, 6)
>>> obj.index(1)
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    obj.index(1)
ValueError: 1 is not in range
>>>
>>> obj = range(-10,11,3)
>>> list(obj)
[-10, -7, -4, -1, 2, 5, 8]
>>> try:
    # критическая секция
    index_item_1 = obj.index(1)
except ValueError: # секция обработчика исключения
    print("Объект-генератор не содержит искомое значение")
```

```
Объект-генератор не содержит искомое значение # вывод программы
>>>
```

Метод `count(<Значение>)` возвращает количество элементов с указанным значением. Если в последовательности нет элементов, содержащих указанное значение, метод возвращает значение 0.

```
>>> obj = range(-100, 101)
>>> obj.count(-1), obj.count(100), obj.count(101)
(1, 1, 0)
```

```
>>>
```

- Функция `enumerate()`

Объект-функция `enumerate()` (встроенная функция – *built-in function of Python*) на каждой итерации цикла `for` возвращает кортеж – пару значений из индекса и значения текущего элемента.

```
>>> arr = [i for i in range(1,11)]
>>> for i, ai in enumerate(arr):
    if ai % 2 == 0:
        arr[i] *= 3
    else:
        ai += 1
```

```
>>> print(arr)
[1, 6, 3, 12, 5, 18, 7, 24, 9, 30]
>>>
```

Синтаксис данного объекта-функции:

```
enumerate(<Последовательность>[, <start=0>])
```

Необязательный параметр `<start>` позволяет задать **начальный номер** элемента последовательности, с которым будет «связан» **первый элемент** `<Последовательности>` (формируются кортежи, содержащие пары значений – номер и элемент `<Последовательности>`).

```
>>> my_list = ['apple', 'banana', 'grapes', 'pear']
>>> counter_list_0 = list(enumerate(my_list)); print(counter_list_0)
[(0, 'apple'), (1, 'banana'), (2, 'grapes'), (3, 'pear')]
>>> counter_list_1 = list(enumerate(my_list, 1));
>>> print(counter_list_1)
[(1, 'apple'), (2, 'banana'), (3, 'grapes'), (4, 'pear')]
>>>
```

Следующий пример показывает, как именно в цикле получить доступ к элементу последовательности, а не работать с его копией.

```
>>> arr = [i for i in range(-10,11,3)]; arr
[-10, -7, -4, -1, 2, 5, 8]
>>> for i in arr:
    if i % 2 != 0:
        i += 10      # изменяется копия элемента списка

>>> arr              # содержимое списка не изменилось
[-10, -7, -4, -1, 2, 5, 8]
>>>
>>> for i, elem in enumerate(arr):
```

```
if elem % 2 != 0:
    arr[i] +=10 # изменяем непосредственно элемент списка

>>> arr # содержимое списка изменилось
[-10, 3, -4, 9, 2, 15, 8]
```

И ещё примеры.

```
>>> choices = ['pizza', 'pasta', 'salad', 'nachos']
>>> list(enumerate(choices))
[(0, 'pizza'), (1, 'pasta'), (2, 'salad'), (3, 'nachos')]
>>> for index, item in enumerate(choices, start = 1):
    print(index, item, end = '|')
```

```
1 pizza|2 pasta|3 salad|4 nachos|
>>> for index, item in enumerate(choices, 1):
    print(index, item, end = '|')
```

```
1 pizza|2 pasta|3 salad|4 nachos|
>>> for index, item in enumerate(choices):
    print(index, item, end = '|')
```

```
0 pizza|1 pasta|2 salad|3 nachos|
>>>
>>> arr = [i for i in enumerate('asm')]
>>> for i in arr: print(i, type(i), end = '|')
```

```
(0, 'a') <class 'tuple'>|(1, 's') <class 'tuple'>|(2, 'm') <class
'tuple'>|
>>>
```

Пусть это и немного опережая события, но связано с использованием `enumerate()`.

```
>>> arr = {i:j for i,j in enumerate("abc")}; arr; type(arr)
{0: 'a', 1: 'b', 2: 'c'}
<class 'dict'>
>>> arr = {(i,j) for i,j in enumerate("abc")}; arr; type(arr)
{(2, 'c'), (1, 'b'), (0, 'a')}
<class 'set'>
>>> arr = {i for i in enumerate("abc")}; arr; type(arr)
{(2, 'c'), (1, 'b'), (0, 'a')}
<class 'set'>
>>>
>>> arr = [(i,j) for (i,j) in enumerate("abc")]; arr; type(arr)
[(0, 'a'), (1, 'b'), (2, 'c')]
```

```
<class 'list'>
>>> arr = [(i,j) for i,j in enumerate("abc")]; arr; type(arr)
[(0, 'a'), (1, 'b'), (2, 'c')]
<class 'list'>
>>> arr = [i,j for (i,j) in enumerate("abc")]; arr; type(arr)
SyntaxError: invalid syntax
>>>
```

Ошибка в последнем примере вызвана тем, что параметры генератора списка i и j не заключены в круглые скобки. Предпоследний пример показывает, как именно нужно было записать данное выражение.

```
>>> sequence = {} # пустой словарь
>>> my_string = "John"
>>> for i, ch in enumerate(my_string):
    sequence[i] = ch

>>> print(sequence)
{0: 'J', 1: 'o', 2: 'h', 3: 'n'}
>>> for key in sequence:
    print(key, '=', sequence[key], end = ' ')

0 = J 1 = o 2 = h 3 = n
>>>
```

Задачи

Для закрепления материала решите несколько простых задач.

1. Составьте программу для вывода в стандартный поток значений кубов всех целых чисел от 5 до N (значение N пользователь вводит с клавиатуры). В цикле, реализованном в программе, должна быть проверка, что введенный параметр $N \geq 5$.
2. Составить программу для вывода на экран всех нечетных чисел из интервала $[5; 45]$. Рассмотреть реализации программы, использующей вложенную инструкцию *if*, так и без нее.
3. Вывести на экран все нечетные двузначные числа, у которых последняя цифра равна 1 или 9.
4. Составить программу для вывода на экран всех целых чисел, расположенных на интервале $[A; B]$, а также количество этих чисел N . Значения границ интервала A и B ($A < B$) вводятся пользователем. Рассмотреть варианты обработки ситуации не правильного ввода значения границ интервала, когда пользователь ввел значения $A > B$.
5. Составить программу для вывода на экран всех целых чисел, расположенных на интервале $(A; B)$, а также количество этих чисел N .

Значения границ интервала A и B ($A < B$) вводятся пользователем. Рассмотреть варианты обработки ситуации не правильного ввода значения границ интервала, когда пользователь ввел значения $A > B$.

6. Составить программу для нахождения суммы всех целых чисел, расположенных на интервале $[A; B]$ ($A < B$). Значения границ интервала вводятся пользователем в произвольном порядке. Например, вводимые пары значений 10 и 15, 25 и 10 считаются корректными значениями.
7. Составить программу для нахождения произведения всех целых чисел, расположенных на интервале $[A; B]$ ($A < B$). Значения границ интервала вводятся пользователем в произвольном порядке. Например, вводимые пары значений 10 и 15, 25 и 10 считаются корректными значениями.
8. Составить программу для вывода на экран всех целых чисел, лежащих в интервале $[A; B]$, кратных некоторому числу C .
9. Составить программу для вывода на экран всех двузначных чисел, сумма квадратов цифр которых делится на 13.
10. Составить программу для нахождения суммы ряда $1^2 + 2^2 + 3^2 + \dots + N^2$. Величина параметра N задается пользователем.

Цикл `while`

Цикл `while` в *Python* используется для организации цикла с предусловием. Напоминаю, что цикл с постусловием (`do...while`) в *Python* не реализован. Формат цикла `while`:

```
while <Условие>:
    <Инструкции_тела_цикла>
[ else:
    <Блок,_выполняемый_если_в_цикле_был_исполнен_оператор_continue>
]
```

Попробуйте код, навеянный старым, добрым вирусом «Бе».

```
while True:
    msg = input("Скажи 'БЕ':")
    if msg == 'БЕ' or msg == 'бе':
        print('Фиг тебе-бе-бе...'); break
```

Как видно из листинга, программа «*вежливо просит*» пользователя ввести фразу «БЕ». Проверка выполняется на соответствие введенного текста со строками «БЕ» и «бе». Во времена *MS-DOS* у программ была возможность блокировать прерывания типа *Ctrl-Alt-Del*, *Ctrl-Z* и несколько «*помотать нервы*» пользователю...

Также как и в цикле `for`, в цикле `while` реализован механизм обработки событий использования оператора `continue`.

```
>>> count = 0      # начальное значение счетчика цикла
>>> while count < 10:
```

```
print(count, end = ' ')\n    count += 1 # приращение счетчика цикла
```

0 1 2 3 4 5 6 7 8 9

```
>>> count = 0 # начальное значение счетчика цикла\n>>> while count < 10:\n    if count == 5: break\n    print(count, end = ' ')\n    count += 1 # приращение счетчика цикла
```

0 1 2 3 4

```
>>>\n>>> count = 0\n>>> while count < 10:\n    count += 1\n    if count % 2 == 0:\n        continue\n    print(count, end = ' ')\nelse:\n    print("Обработка 'альтернативной' ветви цикла")
```

1 3 5 7 9 Обработка 'альтернативной' ветви цикла\n>>>

Увы, на *MS Windows* '`\b`' не работает, как *backspace* («**забой символа**») и следующий код не так «**берет за душу**»...

```
import time\nimport sys\ntime_of_life = 9\nprint("И жизнь тебе осталось (секунд): ", end = ' ')\nwhile time_of_life > 0:\n    sys.stdout.write(str(time_of_life))\n    sys.stdout.flush()\n    time.sleep(1)\n    time_of_life -= 1\n    sys.stdout.write('\b')\n    sys.stdout.flush()\nprint(" СТРАШНО?...")
```

Зато вы вполне можете выполнить данную задачу, используя средства библиотеки *Tkinter*.

Задачи

Для закрепления материала решите несколько простых задач.

1. Составить программу для нахождения значения наибольшего общего делителя (НОД) заданных чисел, реализующую алгоритм Евклида: $\text{НОД}(A, B) = \text{НОД}(B, A \bmod B)$, если $B \neq 0$; $\text{НОД}(A, 0) = A$.
2. Составить программу для нахождения значения N -го члена ряда Фибоначчи. При условии, что $F_1 = 1$, $F_2 = 1$, $F_3 = 2$, $F_K = F_{K-1} + F_{K-2}$, $K = 3, 4, \dots$
3. Составить программу для нахождения номера первого члена ряда Фибоначчи, превышающего указанное целое значение N .
4. Составить программу, которая для введенного целого числа, являющегося членом ряда Фибоначчи – F_K , находит значения последующего и предыдущего членов ряда Фибоначчи, и номер данного числа K .
5. Составить программу для вывода на экран всех нечетных натуральных чисел из заданного интервала $[A; B]$. Значения границ интервала вводятся пользователем. Цикл *for* не использовать.
6. Составьте программу, имитирующую процесс ввода пользователем значений «логин» и «пароль». Количество ввода неверных значений пар «логин-пароль» должно ограничиваться тремя. По исчерпанию попыток ввода данных авторизации программы должна выводить на экран соответствующее сообщение – «Превышен лимит попыток авторизации». *Это только в кино «хакер» запускает программу перебора паролей, а «тупой сервер» бесчисленное количество раз допускает попытки ввода неверных данных и не блокирует данный процесс... И никакой службы аудита, никакого блокирования внешнего соединения. Кино =).*
7. Составить программу, предназначенную для вычисления значения суммы целых чисел, вводимых пользователем с клавиатуры. Признак конца ввода – введенное нулевое значение.
8. Дано целое число N ($N > 0$). Используя операции деления нацело и взятия остатка от деления, найти количество и сумму его цифр.
9. Дано целое число N . Определить, равна ли сумма составляющих его пар цифр числу 111. *Представляете, подобный механизм использовала компания "Micro Soft" для проверки на правильность значения ключа регистрации своих продуктов? В программе хранился не сам ключ, а алгоритм, позволяющий сравнивать определенный набор ключей.*
10. Составить программу для выполнения проверки, является ли введенное пользователем целое число N ($N > 1$) простым.

ИСПОЛЬЗОВАНИЕ МОДУЛЯ «ЧЕРЕПАХА»

Для повышения мотивации программиста при изучении языка программирования языка *Python* разработчики ввели в состав «стандартной поставки» языка модуль «*turtle*» («черепашка»). Данный модуль предназначен для управления процессом перемещения по графическому экрану приложения **исполнителя** «световое перо». Посредством вводимых команд можно управлять

процессом перемещения «пера» по экрану, рисовать линии, задавать их толщину и цвет.

Модуль *turtle* реализован средствами модуля *Tkinter*. Если вы работаете на *Ubuntu Linux*, то у вас, возможно, не установлен модуль *Tkinter* и, как следствие, модуль *turtle* будет недоступен.

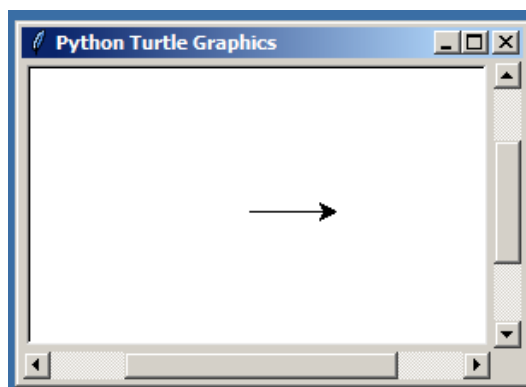


Рисунок 1. Простое окно «turtle» приложения

Код, соответствующий приложению (рис.1):

```
import turtle          # импортировать модуль turtle
bob = turtle.Turtle()  # создать объект "перо"
bob.forward(50)         # переместить «перо» направо на 50 пикселей
turtle.done()          # поднять «перо» - останов процесса рисования
```

Рассмотрим некоторые команды, доступные в модуле *turtle*.

Таблица 1. Команды модуля *turtle*

Команда	Назначение команды	Пример использования
<i>forward(n)</i> <i>fd(n)</i>	Передвижение вперед в направлении острия стрелки на <i>n</i> пикселей.	<i>turtle.forward(50)</i> <i>turtle.fd(25)</i>
<i>backward(n)</i>	Передвижение назад на <i>n</i> пикселей.	<i>turtle.backward(30)</i>
<i>begin_fill()</i> <i>end_fill()</i>	Задать начало и конец закрашенной области	<i>turtle.begin_fill()</i> <i>turtle.end_fill()</i>
<i>up()</i> <i>penup()</i> <i>up()</i>	"Поднять" перо, чтобы при его перемещении не оставался «след». Используется для организации перемещение без рисования.	<i>turtle.up()</i>
<i>down()</i> <i>pd()</i>	"Опустить" перо, чтобы при перемещении оставался след, то есть осуществлялся процесс рисования. По умолчанию перо опущено. « <i>pd</i> – <i>pen down</i> »	<i>turtle.down()</i> <i>turtle.pd()</i>

<code>dot()</code>	Нарисовать на холсте <i>turtle</i> -приложения точку в текущей позиции экрана	<code>turtle.dot()</code>
<code>circle(r)</code>	Начертить окружность радиуса r , с центром слева от позиции курсора (позиция курсора рассматривается как вектор в $2D$ -пространстве <i>turtle</i> -приложения), если $r > 0$; если $r < 0$, то окружность чертится справа. Отображаемая окружность примыкает к позиции «курсора».	<code>turtle.circle(50)</code> <code>turtle.circle(-50)</code>
<code>circle(r, n)</code>	Начертить дугу радиуса r и градусной мерой n против часовой стрелки, если $r > 0$, или по часовой, если $r < 0$. При отображении дуги курсор перемещается – рисование осуществляется под позицией курсора-пера.	<code>turtle.circle(-50, 90)</code>
<code>clear()</code>	Очистить область рисования.	<code>clear()</code>
<code>color(c1, c2)</code>	Установить значение цвета $c1$ для линии рисования и значения $c2$ для цвета заполнения замкнутой области	<code>turtle.color("red")</code> <code>turtle.color("red", "green")</code> <code>turtle.color("#0000aa")</code>
<code>exitonclick()</code>	Задать обработчик события окна рисования – "клик" курсором мыши по полу формы приводит к завершению <i>turtle</i> -сеанса.	<code>turtle.exitonclick()</code>
<code>right(n)</code> <code>rt(n)</code>	Поворот направления движения пера направо на n градусов. По умолчанию поворот осуществляется по часовой стрелке.	<code>turtle.right(90)</code> <code>turtle.tr(90)</code>
<code>left(n)</code> <code>lt(n)</code>	Поворот направления движения пера налево на n градусов.	<code>turtle.left(72)</code> <code>turtle.lt(72)</code>
<code>goto(x, y)</code>	Перемещение пера в точку с координатами (x, y) в системе координат окна рисования. При этом ориентация пера не изменяется.	<code>turtle.goto(120, 240)</code>
<code>mainloop()</code>	Запустить главный цикл обработки событий окна <i>turtle</i> -приложения (<i>Tkinter</i> -приложения)	<code>turtle.mainloop()</code>

<code>x, y = pos()</code>	Прочитать значение текущей позиции пера. Метод <code>pos()</code> возвращает кортеж значений координат <code>x</code> и <code>y</code> .	<code>x, y = turtle.pos()</code>
<code>home()</code>	Переместить перо в точку <code>(0, 0)</code>	<code>turtle.home()</code>
<code>reset()</code>	«Сбросить» состояние <code>turtle</code> -приложения. Приводит к очистке внутренних буферов, уничтожению ранее созданных объектов и очистке экрана.	<code>turtle.reset()</code>
<code>radians()</code>	Установить механизм измерения угловых мер в радианах, по умолчанию для измерения угловых мер используются градусы.	<code>turtle.radians()</code>
<code>setx(x)</code> <code>sety(y)</code>	Установить новое значение координаты позиции курсора – приводит к изменению пользовательской системы координат без изменения положения курсора.	<code>turtle.position()</code> <code>(0.00, 240.00)</code> <code>turtle.setx(10)</code> <code>turtle.position()</code> <code>(10.00, 240.00)</code>
<code>setup()</code>	Установить размеры экрана <code>turtle</code> -приложения в пикселях	<code>turtle.setup(240,180)</code>
<code>shape(st)</code>	Задать вид пера	<code>turtle.shape("turtle")</code>
<code>speed(s)</code>	Установить значение скорости перемещения пера по холсту в условных единицах. <code>S = [1, 10]</code>	<code>turtle.speed(10)</code>
<code>tracer(flag)</code>	Включение (<code>flag=1</code>) и выключение (<code>flag=0</code>) режима отображения пера. По умолчанию перо включено (отображается в виде "знака"). При выключенном режиме отображения пера процесс рисования происходит значительно быстрее, чем при включенном.	<code>turtle.tracer(flag=0)</code>
<code>Turtle()</code>	Создать объект <code>turtle</code> -приложения	<code>turtle.Turtle()</code>
<code>undo()</code>	Отменить последнюю команду	<code>turtle.undo()</code>
<code>width(n)</code>	Задать толщину пера в пикселях	<code>turtle.width(10)</code>
<code>write(String)</code>	Вывести в поле окна <code>turtle</code> -приложения текст, определенный параметром <code>String</code> . Положение пера будет соответствовать нижнему левому углу области вывода текста	<code>turtle.write("Hello")</code>

Рассмотрим код, позволяющий нарисовать четыре точки на «холсте» и два отрезка, соединяющие их в пары (рис.2).

```
>>> import turtle
>>> turtle.tracer(0)      # отключить отображение пера
>>> # начало первого отрезка
>>> turtle.dot()          # нарисовать точку в текущей позиции пера
>>> turtle.forward(20)    # переместиться на 20 пикселей вперед
>>> turtle.dot()          # нарисовать точку в текущей позиции пера
>>> turtle.up()           # оторвать перо от поверхности холста
>>> # конец первого отрезка
>>> turtle.forward(20)
>>> # начало второго отрезка
>>> turtle.down()         # опустить перо на холст
>>> turtle.dot()          # нарисовать точку в текущей позиции пера
>>> turtle.forward(20)
>>> turtle.dot()
>>> # конец второго отрезка
```

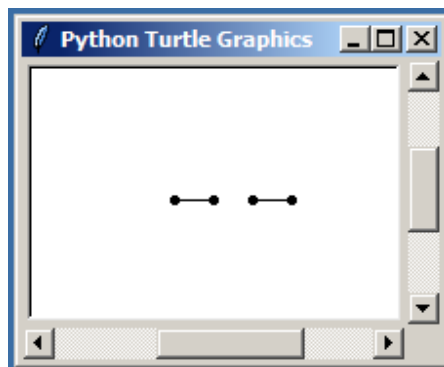


Рисунок 2. Рисование пунктиров – точек и соединяющих их отрезков

Рассмотрим задачу рисования квадрата со сторонами длиной 50 пикселей.

```
>>> import turtle          # Подключить модуль
>>> turtle.reset()         # сбросить состояние сервиса turtle
>>> turtle.forward(50)      # переместить перо на 50 пх вперед
>>> turtle.right(90)        # повернуть направо
>>> turtle.forward(50); turtle.right(90)
>>> turtle.forward(50); turtle.right(90)
>>> turtle.forward(50); turtle.right(90)
>>> turtle.done()
```

Данная задача связаны с циклическим выполнением пары команд *turtler.forward(n)* и *turtle.right(k)*. Изменим код.

```
>>> import turtle
>>> smart = turtle.Turtle()
>>> for i in range(4):
>>>     smart.forward(50)
```

Не опубликованная версия, замечания и предложения направляйте на кафедру

```
smart.right(90)
```

```
>>> turtle.done() # shell уходит в даун...
```

Настоятельно не рекомендуется выполнять данные примеры в режиме пошагового исполнения (в окне shell IDLE), так как некоторые команды «ориентированы» на окно turtle-приложения и могут «подвешивать» shell.

Рассмотрим пример рисования пятиконечной звезды.

```
import turtle
star = turtle.Turtle()
for i in range(5):
    star.forward(50)
    star.right(144)
turtle.done()
```

Если изменить величину поворота, то получим другую звезду. Например, при величине поворота в 100 градусов, получается изображение, похожее на изображение Солнца. При величине поворота в 72 градуса получается правильный пятиугольник. Интересное изображение получается и при величине поворота в 170 градусов.

Задание. Поэкспериментируйте с данным кодом, изменяя число итераций цикла и величину поворота пера.

Ниже приведен код "спиральной" пятиконечной звезды.

```
import turtle
star = turtle.Turtle()
for i in range(20):
    star.forward(i * 10)
    star.right(144)
turtle.done()
```

Рассмотрим ещё несколько простых примеров.

```
# изменение формы "пера" turtle-приложения
import turtle
turtle.shape("turtle") # перо в форме черепашки =)
turtle.forward(50)
turtle.exitonclick() # завершить работу приложения одним кликом...

"""
Рисование изображения Солнца: линия красная, заполнение желтым цветом
"""
from turtle import *
color('red', 'yellow') # установить цвета рисования и заполнения
begin_fill() # определить заполнение замкнутых фигур
```

```
while True:
    forward(200)
    left(170)
    if abs(pos()) < 1:
        break
end_fill()
done()

"""
Нарисовать оси декартовой системы координат
"""

import turtle
turtle.reset()
turtle.tracer(1)          # включить прорисовку при перемещении пера
turtle.color('#0000FF')   # Задать цвет линии в системе RGB
"""

# При запуске приложения перо находится в начале пользовательской
системы координат (ПСК), в точке (0,0)
"""

turtle.write('0,0')        # Вывести надпись - значение начала ПСК
turtle.up()                # оторвать перо от поля рисования
# задать значения координат новой точки вывода на экран
x = -170
y = -120
# преобразовать координаты в строку
coords = str(x) + "," + str(y)
turtle.goto(x,y)           # переместить перо в указанную точку ПСК экрана
turtle.write(coords)       # вывести надпись - координаты точки
# задать значения координат новой точки вывода на экран
x = 130
y = 100
coords = str(x) + "," + str(y)
turtle.goto(x,y)
turtle.write(coords)
x = 0
y = -100
coords = str(x) + "," + str(y)
turtle.goto(x,y)
turtle.write(coords)
turtle.down()              # опустить перо на холст
x = 0
y = 100
coords = str(x) + "," + str(y)
turtle.goto(x,y)
turtle.write(coords)
turtle.up()                # оторвать перо от поля рисования
x = -150
y = 0
coords = str(x) + "," + str(y)
turtle.goto(x,y)
```

```
turtle.write(coords)
turtle.down()          # опустить перо на холст
x = 150
y = 0
coords = str(x) + "," + str(y)
turtle.goto(x,y)
turtle.write(coords)
# модуль turtle реализован средствами модуля tkinter, поэтому...
turtle.mainloop()     # нужно запустить цикл обработки событий окна
# END: все, конец файла модуля

#!/usr/bin/python3
# -*- coding: utf-8 -*-
"""
Данная программа выводит изображение смайлика.
Задавать кодовую таблицу нет особого смысла, так как в программе не
выводится текст в национальной кодировке (кириллица не используется).
"""
import turtle
turtle.reset()         # сбросить состояние turtle: очистить буферы и т.д.
turtle.tracer(0)       # отключить отображение пера при выводе рисунка
turtle.width(2)        # задать толщину рисуемой линии
turtle.up()            # поднять перо - оторвать его от поверхности холста
# задать координаты точки, в которую будет перенесено перо
x = 0
y = -100
turtle.goto(x,y)       # переместить перо в указанную точку окна в ПСК
turtle.begin_fill()    # задать начало закрашиваемой области
turtle.color('#ffaa00') # установить значение цвета
turtle.down()          # опустить перо
# нарисовать лицо
turtle.circle(100)     # нарисовать окружность - область закрашивания
turtle.end_fill()      # конец закрашиваемой области - линия замкнута...
turtle.color('black')  # установить новое значение цвета рисования
turtle.circle(100)     # нарисовать окружность - кривую
turtle.up()            # поднять перо
# задать координаты точки, в которую будет перенесено перо
x = -45
y = 20
turtle.goto(x,y)
turtle.down()
turtle.color('#0000aa')
turtle.begin_fill()
# рисуем глаза
turtle.circle(14)
turtle.up()
turtle.end_fill()
#
x = 45
```

```
y = 20
turtle.goto(x,y)
turtle.down()
turtle.color('#0000aa')
turtle.begin_fill()
turtle.circle(14)
turtle.up()
turtle.end_fill()
# прорисовка рта
x = -55
y = -50
turtle.goto(x,y)
turtle.right(45)
turtle.width(3)
turtle.down()
turtle.color('#aa0000')
turtle.circle(80,90)
turtle.up()
#
turtle.right(135)
# прорисовка линии носа
x = 0
y = 50
turtle.goto(x,y)
turtle.down()
turtle.width(5)
turtle.color('black')
turtle.forward(100)
#
turtle.mainloop()

# Заимствованный простой и интересный пример
#!/usr/bin/env python3
import sys
import turtle
def border(t, screen_x, screen_y):
    """(Turtle, int, int)
    Draws a border around the canvas in red.
    """
    # Lift the pen and move the turtle to the center.
    t.penup()
    t.home()
    # Move to lower left corner of the screen; leaves the turtle
    # facing west.
    t.forward(screen_x / 2)
    t.right(90)
    t.forward(screen_y / 2)
    t.setheading(180) # t.right(90) would also work.
    # Draw the border
```



```
t.pencolor('red')
t.pendown()
t.pensize(10)
for distance in (screen_x, screen_y, screen_x, screen_y):
    t.forward(distance)
    t.right(90)
# Raise the pen and move the turtle home again; it's a good idea
# to leave the turtle in a known state.
t.penup()
t.home()
def square(t, size, color):
    """(Turtle, int, str)
    Draw a square of the chosen colour and size.
    """
    t.pencolor(color)
    t.pendown()
    for i in range(4):
        t.forward(size)
        t.right(90)
def main():
    # Create screen and turtle.
    screen = turtle.Screen()
    screen.title('Square Demo')
    screen_x, screen_y = screen.screensize()
    t = turtle.Turtle()
    # Uncomment to draw the graphics as quickly as possible.
    ##t.speed(0)
    # Draw a border around the canvas
    border(t, screen_x, screen_y)
    # Draw a set of nested squares, varying the color.
    # The squares are 10%, 20%, etc. of half the size of the canvas.
    colors = ['red', 'orange', 'yellow', 'green', 'blue', 'violet']
    t.pensize(3)
    for i, color in enumerate(colors):
        square(t, (screen_y / 2) / 10 * (i+1), color)
    print('Hit any key to exit')
    dummy = input()
if __name__ == '__main__':
    main()
```

За большей информацией о возможностях рисования обращайтесь к документации (<https://docs.python.org/3/library/turtle.html>). Хотя не повредит ознакомиться и с встроенной документацией, доступной по команде:

```
>>> help(turtle)
```

Задачи

1. Напишите *turtle*-программу для рисования изображения “домика” - квадрат под треугольником без подъема пера при условии однократного перемещения по одной линии.
2. Напишите *turtle*-программу для вывода на экран изображения прямоугольника высотой 150 *px* и шириной 200 *px*. В данный прямоугольник по «принципу матрёшки» должны быть вписаны еще четыре подобных прямоугольника.
3. Напишите *turtle*-программу для рисования олимпийских колец: пять колец пяти цветов – сверху синее, черное и красное, снизу желтое и зеленое. Величина толщины линии должна быть отличной от единицы! На рисунке 3 представлены оригинальный и упрощенный варианты изображения.

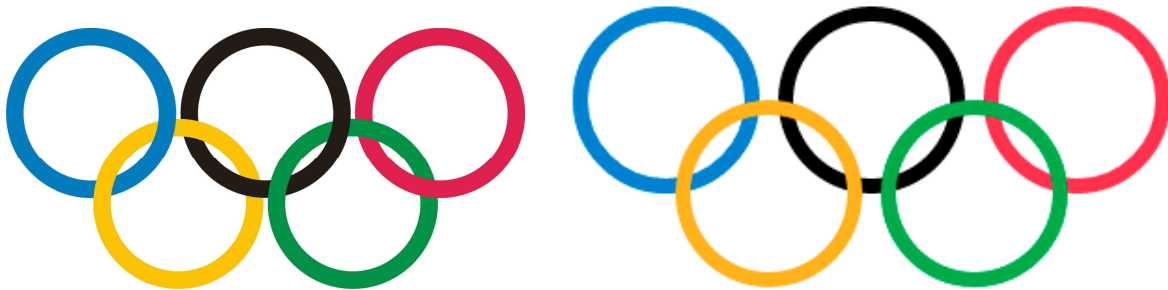


Рисунок 3. Оригинальное и упрощенное изображения олимпийских колец

4. Напишите *turtle*-программу для рисования простейшей новогодней открытки, включающей в себя изображения ёлки и снеговика. Цвет снега и фона заднего вида у открытки должны отличаться.
5. Напишите *turtle*-программу для рисования фрактальных кривых, например, треугольника Серпинского (рис.4).



Рисунок 4. Фракталы треугольник Серпинского с 1-го по 5-го порядков

6. Напишите *turtle*-программу для рисования трех квадратов, имеющих общую вершину, повернутых друг относительно друга на 22 градуса.

Литература и источники в Интернет

1. Numeric and Mathematical Modules [электронный ресурс]: <https://docs.python.org/3/library/numeric.html> (дата обращения – 07.08.2017).
2. Дистанционная подготовка по информатике [электронный ресурс]: <https://informatics.mscme.ru/> (дата обращения – 17.08.2017).
3. Прохоренок Н.А. Python 3 и PyQt 5. Разработка приложений [Текст] / Н.А. Прохоренок, В.А. Дронов. – СПб.: БХВ-Петербург, 2016. – 832.: ил.
4. Python: коллекции, часть 4/4: Все о выражениях-генераторах, генераторах списков, множеств и словарей [электронный ресурс]: <https://habrahabr.ru/post/320288/> (дата обращения – 2.09.2017).
5. Doug Hellmann. The Python 3 Standard Library by Example [Текст / Электронный ресурс] Addison-Wesley ISBN-13: 978-0-13-429105-5, ISBN-10: 0-13-429105-0 <https://www.amazon.com/Python-Standard-Library-Example-Developers/dp/0134291050> [Дата обращения - 2.09.2017].
6. Doug Hellmann. Python 3 Module of the Week [электронный ресурс]: <https://pymotw.com/3/> (дата обращения – 2.09.2017)
7. Jackson Cooper. Python's range() Function Explained [электронный ресурс]: <http://pythoncentral.io/pythons-range-function-explained/> Python Central (дата обращения – 4.09.2017).
8. Смоленский А., Как работает yield [электронный ресурс]: <https://habrahabr.ru/post/132554/> Хабрахабр (дата обращения – 4.09.2017).