Модуль 3
Урок 2. Пространства имён и области видимости
Изучение области видимости и разберётесь в пространствах имён модулей.

Пространство имён (namespace) — это набор связей имён с объектами. В настоящий момент большинство пространств имён реализованы в виде словарей Python, но не стоит заострять на этом внимание (если только по поводу производительности): возможно, в будущем реализация изменится. Примеры пространств имён: набор встроенных имён (функции вроде abs() и имён встроенных исключений); глобальные имена в модуле; локальные имена при вызове функции. Важная вещь, которую необходимо знать о пространствах имён — это то, что нет абсолютно никакой связи между именами в разных пространствах имён: например, два разных модуля могут без проблем определять функцию «maximize», так как пользователи модулей будут использовать имена модулей в качестве префиксов.

Сегодня мы будем говорить о важных теоретических основах, которые необходимо понимать и помнить, чтобы писать грамотный, читаемый и красивый код. Мы будем вести речь об областях видимости переменных. Эта статья будет полезна не только новичкам, но и опытным программистам, которые пришли в Python из другого языка и хотят разобраться с его механиками работы.

Области видимости определяют, в какой части программы мы можем работать с той или иной переменной, а от каких переменная «скрыта». Крайне важно понимать, как использовать только те значения и переменные, которые нам нужны, и как интерпретатор языка себя при этом ведет. А еще мы посмотрим, как обходить ограничения, накладываемые областями видимости на действия с переменными. В Python существует целых 3 области видимости:

- Локальная
- Глобальная
- Нелокальная

Последняя, нелокальная область видимости, была добавлена в Python 3. Обычно, речь заходит про области видимости, когда происходит знакомство с функциями. На их примере мы и будем рассматривать работу областей видимости переменных.

Локальная область видимости

Рассмотрим функцию, которая выведет список some_list поэлементно:

```
def print_list(some_list):
    for element in some_list:
        print(element)
```

Здесь element и some_list – локальные переменные, которые видны только внутри функции, и которые не могут использоваться за ее пределами с теми значениями, которые были им присвоены внутри функции при ее работе. То есть, если мы в основном теле программы вызовем print(element), то получим ошибку:

NameError: name 'element' is not defined

Теперь мы поступим следующим образом:

```
def print_list(some_list):
    for element in some_list:
        print(element)

element = 'q'
print_list([1, 2, 3])
print(element)
```

Результат:
1
2
3
q

Здесь переменная element внутри функции и переменная с таким же именем вне ее – это две разные переменные, их значения не перекрещиваются и не взаимозаменяются. Они называются одинаково, но ссылаются на разные объекты в памяти. Более того, переменная с именем element внутри функции живет столько же, сколько выполняется функция и не больше. И не следует давать локальным и глобальным переменным одинаковые имена, сейчас покажу почему:

```
def print_list(some_list):
    for element in sudden_list:
        print(element)

sudden_list = [0, 0, 0]
print_list([1, 2, 3])
```

Результат:

```
0
0
0
```

Обратите внимание на то, что интерпретатор не указал нам на ошибки. А все потому что sudden_list находится в глобальной области видимости, то есть изнутри функции print_list мы можем к нему обращаться, поскольку изнутри видно то, что происходит снаружи. По причине таких механик работы старайтесь называть локальные переменные внутри функции не так, как называете переменные в глобальной области видимости.

Здесь важно поговорить о константах. Интерпретатору Python нет разницы как вы называете переменную, поэтому код выше будет лучше переписать в следующем виде:

```
SUDDEN_LIST = [0, 0, 0]

def print_list(some_list):
    for element in SUDDEN_LIST:
        print(element)

print_list([1, 2, 3])
```

Теперь все на своих местах. Дело в том, что в Python нельзя каким-то образом строго определить константу, как объект, который не должен быть изменен. Так что то, как вы используете значение переменной, имя которой записано заглавными буквами, остается лишь на вашей совести. Другой вопрос, что таким способом записанная переменная даст понять тому, кто будет читать ваш код, что переменная нигде изменяться не будет. Или по крайней мере не должна.

Глобальная область видимости

В Python есть ключевое слово global, которое позволяет изменять изнутри функции значение глобальной переменной. Оно записывается перед именем переменной, которая дальше внутри функции будет считаться глобальной. Как видно из примера, теперь значение переменной candy увеличивается, и обратите внимание на то, что мы не передаем ее в качестве аргумента функции get_candy().

```
candy = 5

def get_candy():
    global candy
    candy += 1
    print('У меня {} конфет.'.format(candy))

get_candy()
get_candy()
print(candy)
```

Результат:

```
У меня 6 конфет.
У меня 7 конфет.
7
```

Однако менять значение глобальной переменной изнутри функции – не лучшая практика и лучше так не делать, поскольку читаемости кода это не способствует. Чем меньше то, что происходит внутри функции будет зависеть от глобальной области видимости, тем лучше.

Нелокальная область видимости

Появилось это понятие в Python 3 вместе с ключевым словом nonlocal. Логика его написания примерно такая же, как и у global. Однако у nonlocal есть особенность. Nonlocal используется чаще всего во вложенных функциях, когда мы хотим дать интерпретатору понять, что для вложенной функции определенная переменная не является локальной, но она и не является глобальной в общем смысле.

```
    def get_candy():
    candy = 5
    def increment_candy():
        nonlocal candy
        candy += 1
        return candy
    return increment_candy

result = get_candy()()
print('Всего {} конфет.'.format(result))
```

Результат:

Всего 6 конфет.

Насколько это полезно вам предстоит решить самостоятельно.

В качестве вывода можно сформулировать несколько правил:

1. Изнутри функции видны переменные, которые были определены и внутри нее и снаружи. Переменные, определенные внутри – локальные, снаружи – глобальные.
2. Снаружи функций не видны никакие переменные, определенные внутри них.
3. Изнутри функции можно изменять значение переменных, которые определены в глобальной области видимости с помощью спецификатора **global**.
4. Изнутри *вложенной* функции с помощью спецификатора **nonlocal** можно изменять значения переменных, которые были определены во внешней функции, но не находятся в глобальной области видимости.

Задание: Используя функции, локальные и глобальные переменные, создайте программу, которая содержит:

1. Функцию, которая позволяет пользователю ввести свои Имя и Фамилию, а затем - произвольный список с клавиатуры, по результату работы – возвращает готовый список.
2. Функцию, которая записывает данный список в файл.
3. Функцию, которая выводит полученный Имя и Фамилию пользователя, а также полученный файл на печать.