Функции

Операторы и выражения, связанные с функциями

Оператор или выражение	Примеры
Выражения вызовов	`myfunc('spam', 'eggs', meat=ham, *rest)`
`def`	`def printer(message):` `  print('Hello ' + message)`
`return`	`def adder(a, b=1, *c):` `  return a + b + c[0]`

`global`	`x = 'old'`
	`def changer():`
	`  global x; x = 'new'`

`nonlocal` (Python 3.X)	`def outer():`
	`  x = 'old'`
	`  def changer():`
	`  nonlocal x; x = 'new'`

`yield`	`def squares(x):` `  for i in range(x): yield i ** 2`
`lambda`	`funcs = [lambda x: x**2, lambda x: x**3]`

Для чего используются функции?

- Доведение до максимума многократного использования кода и сведение к минимуму избыточности
- Процедурная декомпозиция

Написание кода функций

- def является исполняемым кодом.
- def создает объект и присваивает его имени.
- lambda создает объект, но возвращает его в качестве результата.
- return отправляет результирующий объект вызывающему коду.

Написание кода функций

- yield отправляет результирующий объект вызывающему коду, но запоминает место, где он остановился
- global объявляет переменные уровня модуля, предназначенные для присваивания.
- nonlocal объявляет переменные объемлющей функции, предназначенные для присваивания.

Написание кода функций

- Аргументы передаются по присваиванию (по ссылкам на объекты).
- Аргументы передаются по позиции, если только не указано иначе
- Аргументы, возвращаемые значения и переменные не объявляются.

Операторы def

- `def имя(аргумент1, аргумент2,... аргументN):`
  `операторы`

```
def имя(аргумент1, аргумент2,... аргументN):
    ...
    return значение
```

Оператор def исполняется во время выполнения

```
if test:
    def func():    # Одно определение func
        ...
else:
    def func():    # Иначе другое определение func
        ...
...
func()             # Вызов выбранной и созданной версии

othername = func   # Присваивание объекта функции
othername()        # Снова вызывает func


def func(): ...    # Создание объекта функции
func()             # Вызов объекта функции
func.attr = value  # Присоединение атрибутов
```

Определения и вызовы

```
>>> def times(x, y):          # Создание и присваивание функции
...     return x * y           # Тело выполняется при вызове

>>> times(2, 4)               # Аргументы в круглых скобках
8

>>> x = times(3.14, 4)        # Сохранение результирующего объекта
>>> x
12.56

>>> times('Ni', 4)            # Функции "лишены типов"
'NiNiNiNi'
```

Пример: пересечение последовательностей

```
def intersect(seq1, seq2):
    res = []                    # Начать с пустого результата
    for x in seq1:              # Просмотр seq1
        if x in seq2:           # Общий элемент?
            res.append(x)       # Добавить в конец результата
    return res
```

```
def intersect(seq1, seq2):
    res = []                    # Начать с пустого результата
    for x in seq1:              # Просмотр seq1
        if x in seq2:           # Общий элемент?
            res.append(x)       # Добавить в конец результата
    return res
```

```
>>> x = intersect([1, 2, 3], (1, 4))     # Разнородные типы
>>> x                                    # Сохраненный результирующий объект
[1]
```

Локальные переменные

- переменной res производится очевидное присваивание, так что она локальная;
- аргументы передаются по присваиванию, поэтому seq1 и seq2 тоже локальные;
- цикл for присваивает элементы переменной, а потому имя x — также локальная переменная.

Области видимости

- Почти все, что связано с именами, включая классификацию областей видимости, в Python происходит во время присваивания. Как мы уже видели, имена в Python начинают свое существование, когда им впервые присваиваются значения, и чтобы имена можно было использовать, им должны быть присвоены значения. Поскольку имена заранее не объявляются, Python применяет местоположение присваивания имени для ассоциирования (т.е. связывания) со специфическим пространством имен.
- Помимо упаковки кода для многократного использования функции добавляют к программам дополнительный уровень, чтобы свести к минимуму возможность возникновения конфликтов между переменными с одним и тем же именем — по умолчанию все имена, которым выполнено присваивание внутри функции, ассоциируются с пространством имен данной функции и никаким другим,.

Области видимости

- Имена, присвоенные внутри def, могут быть видны только в коде внутри этого оператора def. Ссылаться на такие имена извне функции нельзя.
- Имена, присвоенные внутри def, не конфликтуют с переменными за пределами def, даже если те же самые имена применяются где-то в другом месте. Имя X, присвоенное вне заданного оператора def (т.е. в другом def или на верхнем уровне файла модуля), представляет собой переменную, совершенно отличающуюся от имени X, присвоенную внутри этого def.

Области видимости

- если переменная присваивается внутри def, то она будет локальной в этой функции;
- если переменная присваивается в объемлющем def, тогда она будет нелокаль ной в отношении вложенных функций;
- если переменная присваивается за пределами всех def, то она будет глобальной в целом файле.

```
X = 99       # X с глобальной областью видимости (модуль)
def func():
    X = 88   # X с локальной областью видимости (функция): другая переменная
```

Детали, касающиеся областей видимости

- Функции предоставляют вложенные пространства имен (области видимости), которые локализуют применяемые в них имена, так что имена внутри функции не конфликтуют с именами за ее пределами (в модуле или в другой функции).

Детали, касающиеся областей видимости

- Включающий модуль является глобальной областью видимости. Каждый модуль представляет собой глобальную область видимости, т.е. пространство имен, в котором существуют переменные, создаваемые (присваиваемые) на верхнем уровне файла модуля. Для внешнего мира глобальные переменные становятся атрибутами объекта модуля после его импортирования, но могут также использоваться в качестве простых переменных внутри самого файла модуля.

Детали, касающиеся областей видимости

- Глобальная область видимости охватывает только одиночный файл. Не обманывайтесь насчет слова “глобальная” — имена на верхнем уровне файла глобальны только в коде внутри этого одного файла. Вообще говоря, в Python нет понятия единственной всеобъемлющей глобальной области видимости, основанной на файлах. Взамен имена распределяются по модулям, и модуль должен всегда явным образом импортироваться, чтобы появилась возможность работы с име нами, определенными в его файле. Когда вы слышите “глобальный” в Python, подразумевайте “модуль”.

Детали, касающиеся областей видимости

- Присвоенные имена являются локальными, если только не объявлены глобальными или нелокальными. По умолчанию все имена, присвоенные внутри определения функции, помещаются в локальную область видимости (пространство имен, ассоциированное с вызовом функции). Если необходимо присвоить имя, которое существует на верхнем уровне модуля, включающего в себя функцию, тогда такое имя можно объявить как глобальное в операторе global внутри функции. Если нужно присвоить имя, находящееся в объемлющем операторе def, то начиная с Python 3.X, имя можно объявить как нелокальное в операторе nonlocal.

Детали, касающиеся областей видимости

- Каждый вызов функции создает новую локальную область видимости. Всякий раз, когда вы вызываете функцию, создается новая локальная область видимости, т.е. пространство имен, в котором обычно будут существовать имена, созданные внутри функции. Вы можете думать о каждом операторе def (и выражении lambda) как об определении новой локальной области видимости, но локальная область видимости в действительности соответствует вызову функции. Поскольку Python разрешает функциям вызывать самих себя в цикле, каждый активный вызов получает собственную копию локальных переменных функции.

Распознавание имен.

Внутри оператора def:

присваивания имен по умолчанию создают либо изменяют локальные имена;

ссылки на имена производят поиск самое большее в четырех областях видимости — в локальной области видимости, затем в областях видимости объемлющих функций (если есть), далее в глобальной области видимости и, наконец, во встроенной области видимости;

имена, объявленные в операторах global и nonlocal, отображают присвоенные имена на области видимости включающего модуля и функции соответственно.

Правило LEGB

- Когда внутри функции указывается неуточненное имя, Python ищет его максимум в четырех местах — в локальной (L (local)) области видимости, затем в локальных областях видимости любых объемлющих (E (enclosing)) операторов def и lambda, далее в глобальной (G (global)) области видимости и, наконец, во встроенной (B (built-in)) области видимости — и останавливает поиск на первом же месте, где обнаруживается имя. Если в результате такого поиска имя найти не удалось, тогда Python сообщит об ошибке
- Когда внутри функции выполняется присваивание имени (вместо просто ссыл ки на него в выражении), Python всегда создает либо изменяет имя в локальной области видимости, если только оно не было объявлено в этой функции как глобальное или нелокальное.
- Когда внутри функции выполняется присваивание имени (вместо просто ссылки на него в выражении), Python всегда создает либо изменяет имя в локальной области видимости, если только оно не было объявлено в этой функции как глобальное или нелокальное.

Правило LEGB