Области видимости

Области видимости

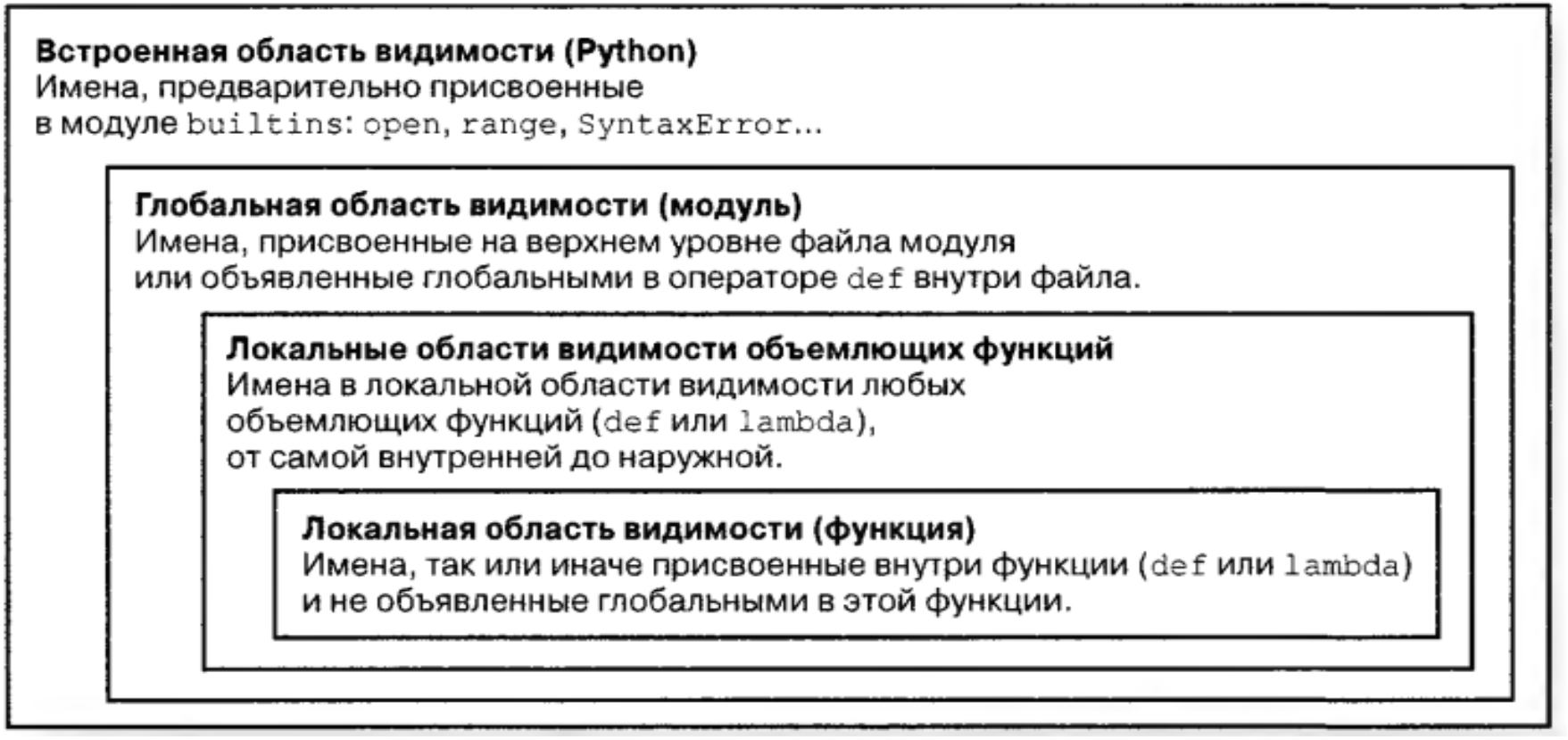
- если переменная присваивается внутри def, то она будет локальной в этой функции;
- если переменная присваивается в объемлющем def, тогда она будет нелокальной в отношении вложенных функций;
- если переменная присваивается за пределами всех def, то она будет глобальной в целом файле.

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

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.

Области видимости



Области видимости

- Когда производится ссылка на переменную, Python ищет ее в следующем, порядке: в локальной области видимости, в локальных областях видимости объемлющих функций, в глобальной области видимости и во встроенной области видимости. Поиск прекращается при первом нахождении. Обычно областьвидимости переменной определяется местом в коде, где она присваивается. Объявления nonlocal в Python 3.X также способны принудительно отображать имена на области видимости объемлющих функций независимо от того, были имена присвоены или нет

Другие области видимости

- формально в Python есть еще три области видимости — временные переменные циклов в ряде включений, переменные ссылок на исключения в некоторых обработчиках try и локальные области видимости в операторах class. Первые две являются особыми случаями, редко влияющими на реальный код, а третья подпадает под действие правила LEGB.

Области видимости

```
# Глобальная область видимости
X = 99                  # Имена X и func присваиваются в модуле: глобальные

def func(Y):            # Имена Y и Z присваиваются в функции: локальные
    # Локальная область видимости
    Z = X + Y           # Имя X является глобальным
    return Z

func(1)                 # Имя func в модуле: result=100
```

Глобальные имена: X, func

Имя X является глобальным, потому что оно присваивается на верхнем уровне файла модуля; на него можно ссылаться внутри функции как на простую неуточненную переменную, не объявляя глобальным. Имя func глобально по той же причине; оператор def присваивает объект функции имени func на верхнем уровне модуля.

Локальные имена: Y, Z

Имена Y и Z являются локальными в функции (и существуют только в период ее выполнения), т.к. им обоим присваиваются значения в определении функции: Z посредством оператора =, а Y из-за того, что аргументы всегда передаются по присваиванию.

Встроенная область видимости

- встроенная область видимости представляет собой всего лишь встроенный модуль под названием builtins, но для запрашивания встроенных имен модуль builtins придется импортировать, потому что имя builtins само по себе встроенным не является...

```
>>> import builtins
>>> dir(builtins)
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException',
'BlockingIOError',
'BrokenPipeError', 'BufferError', 'BytesWarning', 'ChildProcessError',
'ConnectionAbortedError',
...многие другие имена не показаны...
'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property',
'quit', 'range', 'repr',
'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod',
'str', 'sum', 'super',
'tuple', 'type', 'vars', 'zip']
```

Встроенная область видимости

```
>>> zip                          # Обычный способ
<class 'zip'>
>>> import builtins              # Трудный путь: для настройки
>>> builtins.zip
<class 'zip'>
>>> zip is builtins.zip          # Тот же самый объект, другой поиск
True
```

```
def hider():
    open = 'spam'      # Локальная переменная, скрывает здесь встроенное имя
    ...
    open('data.txt') # Ошибка: open в этой области видимости
                     # больше не открывает файл!
```

Оператор global

- Глобальные имена — это переменные, присвоенные на верхнем уровне включающего их файла модуля.
- Глобальные имена должны объявляться, только если им выполняются присваивания внутри функции.
- На глобальные имена можно ссылаться внутри функции без их объявления.

Оператор global

```
X = 88              # Глобальная переменная X

def func():
    global X
    X = 99          # Глобальная переменная X: снаружи def

func()
print(X)            # Выводит 99
```

```
y, z = 1, 2       # Глобальные переменные в модуле
def all_global():
    global x      # Объявление присваиваемой глобальной переменной
    x = y + z     # Объявлять y, z не нужно: правило LEGB
```

Проектирование программы:

- минимизируйте количество глобальных переменных

```
X = 99
def func1():
    global X
    X = 88

def func2():
    global X
    X = 77
```

- минимизируйте количество межфайловых изменений

```
# first.py
X = 99          # Этому коду ничего не известно о second.py

# second.py
import first
print(first.X)    # Нормально: ссылка на имя из другого файла
first.X = 88      # Но его изменение может быть слишком тонким и неявным
```

Проектирование программы:

- наилучший способ взаимодействия между границами файлов предусматривает вызов функций с передачей им аргументов и получением обратно возвращаемых значений.

```
# first.py
X = 99

def setX(new):    # Функция доступа делает внешние изменения явными
    global X      # И она позволяет управлять доступом в одном месте
    X = new

# second.py
import first
first.setX(88)    # Вызвать функцию вместо изменения напрямую
```

Другие способы доступа к глобальным переменным

```
var = 99                                # Глобальная переменная == атрибут модуля

def local():
    var = 0                             # Изменение локальной переменной

def glob1():
    global var                          # Объявление глобальной переменной
                                        #   (нормальное)
    var += 1                            # Изменение глобальной переменной

def glob2():
    var = 0                             # Изменение локальной переменной
    import thismod                      # Импортирование самого себя
    thismod.var += 1                    # Изменение глобальной переменной

def glob3():
    var = 0                             # Изменение локальной переменной
    import sys                          # Импортирование системной таблицы
    glob = sys.modules['thismod']       # Получение объекта модуля
                                        #   (либо использовать __name__)
    glob.var += 1                       # Изменение глобальной переменной

def test():
    print(var)
    local(); glob1(); glob2(); glob3()
    print(var)
```

Области видимости и вложенные функции

- Ссылка (X) ищет имя X сначала в текущей локальной области видимости (фун кция); затем в локальных областях видимости любых лексически объемлющих функций в исходном коде, от внутренней до наружной; далее в текущей глобаль ной области видимости (файл модуля); и, наконец, во встроенной области види мости (модуль builtins). Объявления global заставляют поиск взамен начинаться в глобальной области видимости (файл модуля).
- Присваивание (X = значение) по умолчанию создает либо изменяет имя X в текущей локальной области видимости. Если имя X объявлено глобальным внутри функции, тогда присваивание создает или изменяет имя X в области видимости включающего модуля. Если же имя X объявлено нелокальным внутри функции в Python 3.X (только), то присваивание изменяет имя X в локальной области видимости ближайшей объемлющей функции.

вложенные функции

```
X = 99              # Имя в глобальной области видимости: не используется
def f1():
    X = 88          # Локальное имя объемлющего def

    def f2():
        print(X)    # Ссылка во вложенном def
    f2()

f1()                # Выводит 88: локальное имя объемлющего def
```

Поиск в объемлющей области видимости работает, даже если уже произошел возврат из объемлющей функции. Скажем, в следующем коде определена функция, которая создает и возвращает объект другой функции, представляя более распространенный шаблон использования:

```
def f1():
    X = 88
    def f2():
        print(X)    # Помнит значение X из области видимости объемлющего def
    return f2       # Возвращает объект функции f2, но не вызывает функцию

action = f1()       # Создает и возвращает объект функции
action()            # Вызов функции: выводит 88
```

Фабричные функции: замыкания

```
>>> def maker(N):
        def action(X):          # Создание и возвращение функции action
            return X ** N       # action сохраняет N из объемлющей области видимости
        return action
    >>> f = maker(2)            # Передача 2 аргументу N
    >>> f
    <function maker.<locals>.action at 0x0000000002A4A158>
     >>> f(3)                  # Передача 3 аргументу X, в N запоминается 2: 3 ** 2
     9
     >>> f(4)                  # 4 ** 2
     16
    >>> g = maker(3)           # g запоминает 3, f запоминает 2
    >>> g(4)                   # 4 ** 3
    64
    >>> f(4)                   # 4 ** 2
    16

>>> def maker(N):
        return lambda X: X ** N     # Функции lambda тоже сохраняют состояние

>>> h = maker(3)
>>> h(4)                            # Снова 4 ** 3
64
```

Сохранение состояния с помощью стандартных значений

```
def f1():
    x = 88
    def f2(x=x):        # Запоминает X из объемлющей области видимости
                        # посредством стандартных значений
        print(x)
    f2()

f1()                    # Выводит 88
```

```
>>> def f1():
        x = 88          # Передача x вместо вложения
        f2(x)           # Опережающая ссылка допустима
>>> def f2(x):
        print(x)        # Плоский код все еще нередко лучше вложенного!
>>> f1()
88
```

Вложенные области видимости, стандартные значения и выражения lambda

```
def func():
    x = 4
    action = (lambda n: x ** n)          # x запоминается из объемлющего def
    return action
x = func()
print(x(2))                              # Выводит 16, 4 ** 2


def func():
    x = 4
    action = (lambda n, x=x: x ** n)     # Передача x вручную
    return action
```