PYTHON

Тема3: Области видимости

Область видимости – область в которой определяются переменные и выполняется их поиск.

Всегда, когда в программе используется какое то имя, интерпретатор создает, изменяет или отыскивает это имя в пространстве имен – в области, где находятся имена.

Когда мы говорим о поиске значения имени применительно к программному коду, под термином область видимости подразумевается пространство имен: то есть место в программном коде, где имени было присвоено значение, определяет область видимости этого имени для программного кода.

- Имена, определяемые внутри инструкции **def**, видны только программному коду внутри инструкции **def**.

К этим именам нельзя обратиться за пределами функции.

- Имена, определяемые внутри инструкции **def**, не вступают в конфликт с именами, находящимися за пределами инструкции **def**, даже если и там и там присутствуют одинаковые имена.

Имя **X**, которому присвоено значение за пределами данной инструкции **def** (например, в другой инструкции **def** или на верхнем уровне модуля), полностью отлично от имени **X**, которому присвоено значение внутри инструкции **def**.

PYTHON

Тема3: Области видимости

Три основные области видимости при присваивание переменной **:**

- *внутри инструкции def*, переменная является **локальной** для этой функции
- *в пределах объемлющей инструкции def*, переменная является **нелокальной** для этой функции
- *за пределами всех инструкций def*, она является **глобальной** для всего файла.

```
X = 99          # создает глобальную переменную с именем X (видима из любого места в файле)
def func():
    X = 88      # создает локальную переменную X(она видима только внутри инструкции def)
```

PYTHON

Тема3: Области видимости

Функции образуют локальную область видимости, а модули – глобальную.

Эти две области взаимосвязаны между собой следующим образом:

- **Объемлющий модуль** – это глобальная область видимости. Глобальные переменные для внешнего мира становятся атрибутами объекта модуля, но внутри модуля могут использоваться как простые переменные.
- **Глобальная область** видимости охватывает единственный файл. Не надо заблуждаться насчет слова «глобальный» – имена на верхнем уровне файла являются глобальными только для программного кода в этом файле. Имена всегда относятся к какому-нибудь модулю и всегда необходимо явно импортировать модуль, чтобы иметь возможность использовать имена, определяемые в нем. Когда вы слышите слово «глобальный», подразумевайте «модуль».
- Каждый вызов функции создает новую локальную область видимости. Всякий раз, когда вызывается функция, создается новая локальная область видимости – то есть пространство имен, в котором находятся имена, определяемые внутри функции. Каждую инструкцию def (и выражение lambda) можно представить себе, как определение новой локальной области видимости.
- Операция присваивания создает локальные имена, если они не были объявлены глобальными или нелокальными. По умолчанию все имена, которым присваиваются значения внутри функции, помещаются в локальную область видимости. Если необходимо присвоить значение имени верхнего уровня в модуле, который вмещает функцию, это имя необходимо объявить внутри функции глобальным с помощью инструкции global. Если необходимо присвоить значение имени, которое находится в объемлющей инструкции def, в Python 3.0 это имя необходимо объявить внутри функции с помощью инструкции nonlocal.
- Все остальные имена являются локальными в области видимости объемлющей функции, глобальными или встроенными.

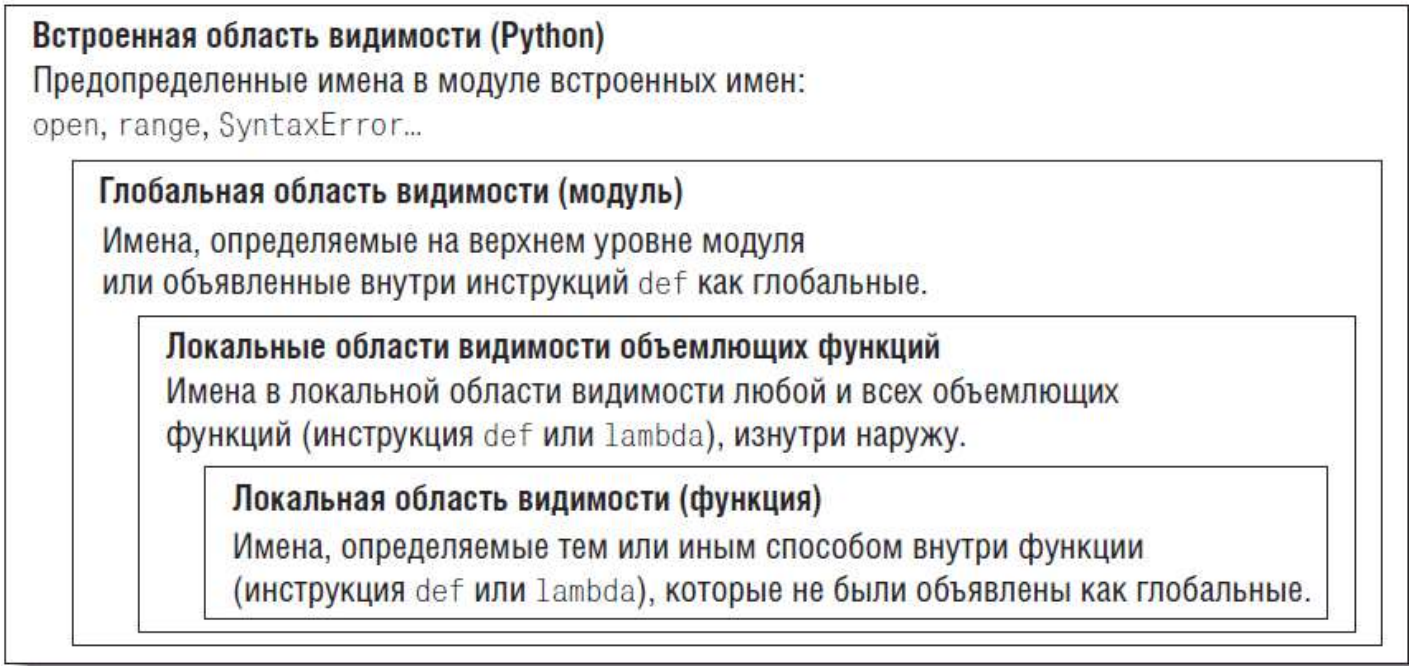
PYTHON

Тема3: Области видимости

Схема разрешения имен в языке Python называется правилом LEGB:

Когда внутри функции выполняется обращение к неизвестному имени, интерпретатор пытается отыскать его в четырех областях видимости:

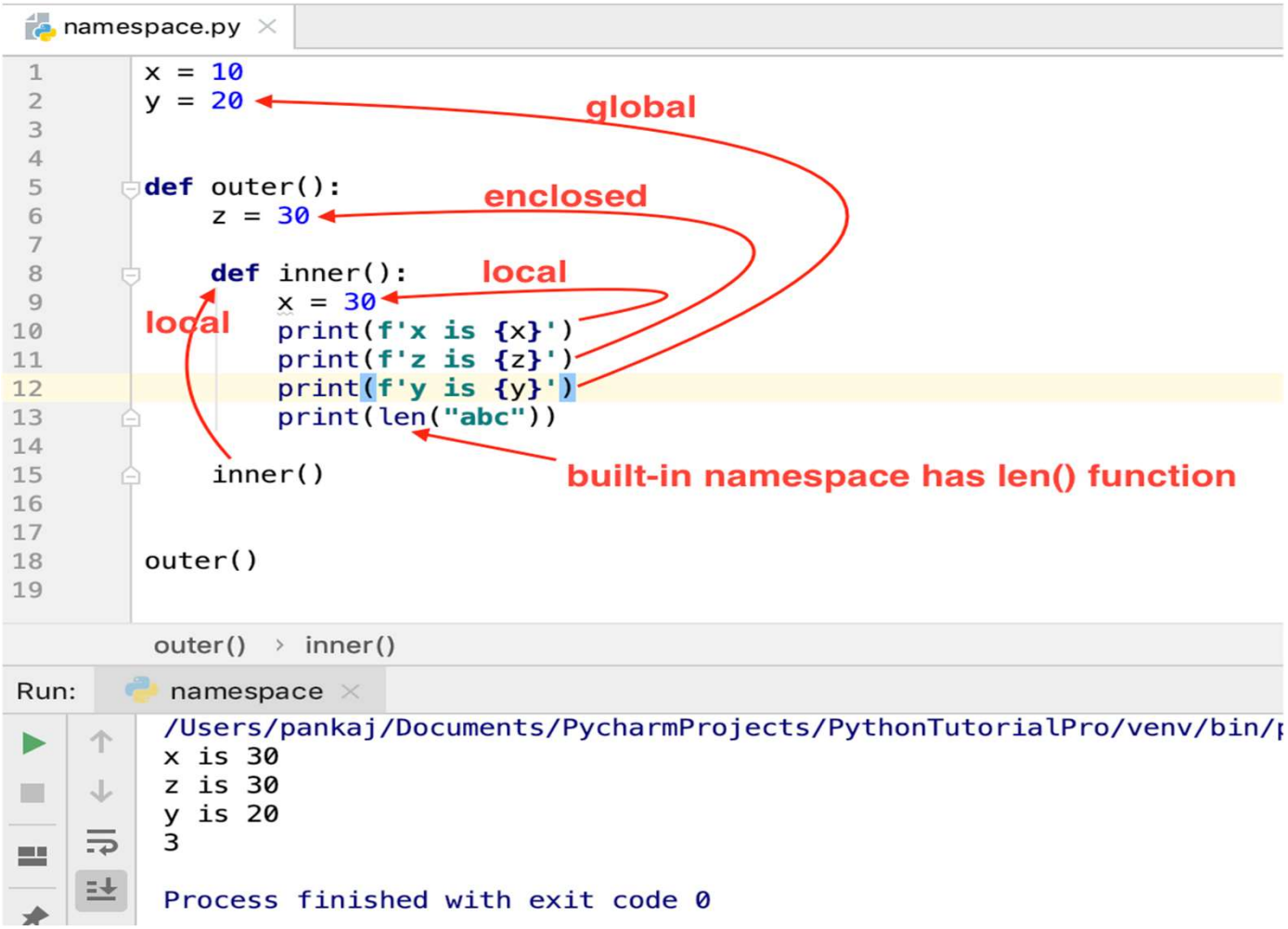
- в локальной (*local*, **L**),
- в локальной области любой объемлющей инструкции def (*enclosing*, **E**) или в выражении lambda,
- затем в глобальной (*global*, **G**)
- во встроенной (*built-in*, **B**). Поиск завершается, как только будет найдено первое подходящее имя.



PYTHON

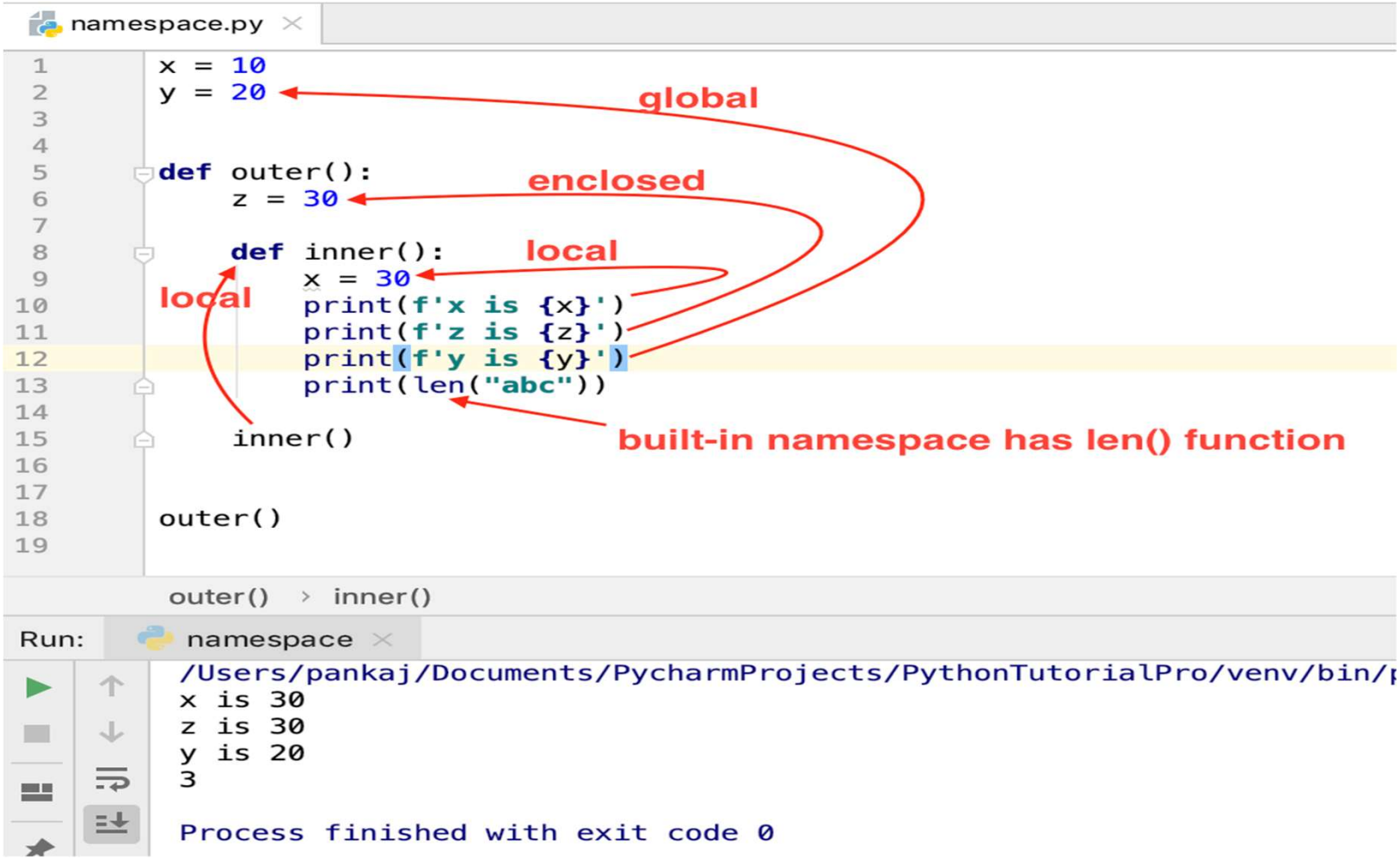
Тема3: Области видимости

```
x = 10
y = 20


def outer():
    z = 20

    def inner():
        x = 30
        print(f'x is {x}')
        print(f'y is {y}')
        print(f'z is {z}')
        print(len("abc"))

    inner()


outer()
```



PYTHON

Тема3: Области видимости



PYTHON

Тема3: Области видимости

Инструкция global:

- глобальные имена – это имена, которые определены на верхнем уровне вмещающего модуля.
- глобальные имена должны объявляться, только если им будут присваиваться значения внутри функций.
- обращаться к глобальным именам внутри функций можно и без объявления их глобальными.

Инструкция global позволяет изменять переменные, находящиеся на верхнем уровне модуля, внутри инструкции def.

PYTHON

Тема3: Области видимости

```
X = 88           # Глобальная переменная X

def func():
    global X
    X = 99       # Глобальная переменная X: за пределами инструкции def

func()
print(X)         # Выведет 99

# ------------------------------------------------------------------------
y, z = 1, 2          # Глобальные переменные в модуле

def all_global():
    global x         # Объявляется глобальной для присваивания
    x = y + z        # Объявлять y, z не требуется: применяется правило LEGB
```

PYTHON

Тема 4: обработка ошибок

Синтаксические ошибки

```
1. for k in range(10)
2.     print(k)
```

```
 File ".../try_except/syntax_error.py", line 1
    for k in range(10)
                      ^
SyntaxError: invalid syntax
```

PYTHON

Тема 4: обработка ошибок

Ошибки времени выполнения

```
# Арифметическая ошибка:
n = 1.0
for k in range(5):
    print(n/k)
```

```
 File "…/try_except/runtime_exeption.py", line 4, in <module>
    print(n/k)
ZeroDivisionError: float division by zero
```

PYTHON

Тема 4: обработка ошибок

Ошибки времени выполнения

```
# Ошибка при работе со словарём:
a = {'host': 'localhost', 'port': 80}
print(a ['IP'])
```

```
 File ".../try_except/runtime_exeption_dic.py", line 3, in <module>
    print(a ['IP'])
KeyError: 'IP'
```

PYTHON

Тема 4: обработка ошибок

Обработка ошибок

```
# Для самостоятельной обработки ошибок внутри программы, возникающих во время выполнения, используются ключевые слова try ... except:

try:
    a = 1.0
    s = 0
    for i in range(5):
        s = s + a / i
    print(s)
except:
   print('Произошло деление на ноль')

# Любая ошибка во время выполнения программы внутри блока try приведет к выполнению кода в блоке except.
```

PYTHON

Тема 4: обработка ошибок

Обработка ошибки определённого типа

```
# После ключевого слова except можно указать тип ошибки

try:
    f = open("datafile.txt", " r ")
    a = f.readline()
except IOError:
    print('Невозможно открыть или прочитать файл')

# > Блок except IOError: выполнится только если произойдёт ошибка, связанная с
вводом/выводом.
# > Ошибки других типов будут обрабатываться объемлющим кодом
```

Типы исключений при работе с файлами

Другие типы исключений для работы с файлами:

- `FileNotFoundError`
  Открываемый файл или каталог не существует

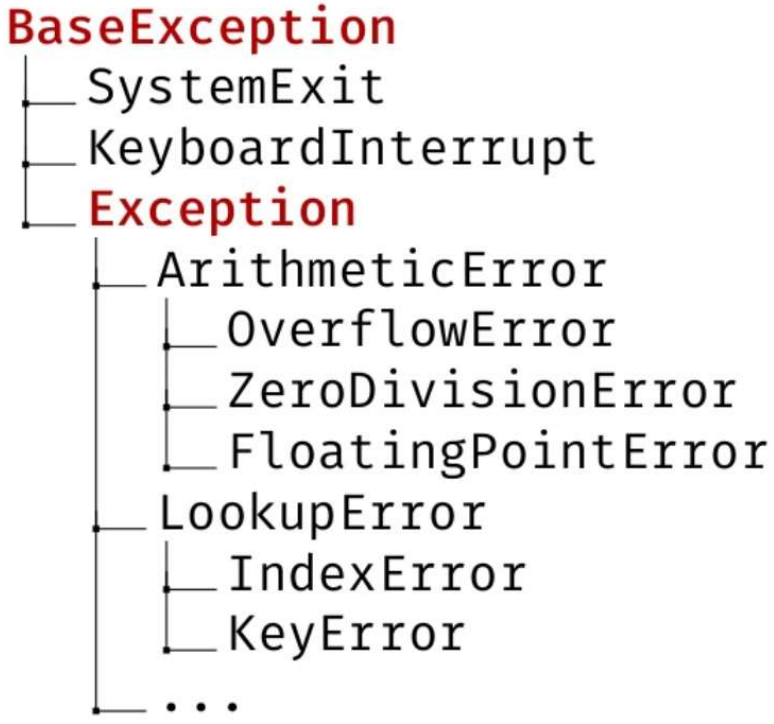
- `FileExistsError`
  Создаваемый файл или каталог уже существует

- `PermissionError`
  Доступ к файлу или каталогу при недостаточном уровне прав

PYTHON

Тема 4: Обработка ошибок

Иерархия исключений



Все исключения кроме `SystemExit` и `KeyboardInterrupt` являются потомками базового класса `Exception`.

PYTHON **Тема 4:** Обработка ошибок

KeyError Код в стиле EAFP

```
# Код в стиле EAFP: Easier to ask for forgiveness than permission
def print_dict_val(d, my_key):
    try:
        print(d[my_key])
    except KeyError:
        print('Ключ не найден: ', my_key)


# или в стиле LBYL: Look before you leap
def print_dict_val(d, my_key):
    if my_key in d:
        print(d[my_key])
    else:
        print('Ключ не найден: ', my_key)
```

Дополнительная информация об ошибке

```
# В блоке except можно указать имя переменной, которая будет иметь тип ошибки и
содержать информацию об ошибке:

try:
    f = open("datafile.txt", "r")
    a = f.readline()
except IOError as err:
    print('Невозможно открыть или прочитать файл ')
    print('Имя файла : ', err.filename)

# Невозможно открыть или прочитать файл
# Имя datafile.txt
```

Несколько блоков except

```
# Блок try может вызывать ошибки различных типов. Для каждого типа ошибки можно создать свой блок except, указав тип ошибки:
try:
    f = open("datafile.txt", "r")
    str_value = f.readline()
    a = int(str_value)
except FileNotFoundError as err:
    print("Невозможно открыть или прочитать файл")
except ValueError as err:
    print("Ошибка преобразования")
except:
    print("Неизвестная ошибка")
```

PYTHON

Тема 4: Обработка ошибок

Блок except для нескольких исключений

```
try:
    f = open("datafile.txt", "r")
    str_value = f.readline()
    a = int(str_value)
except (FileNotFoundError, ValueError) as err:
    print("Ошибка загрузки данных из файла")
except:
    print("Неизвестная ошибка")
```

PYTHON

Тема 4: Обработка ошибок

Конструкция **try ... except ... else**

 # В “защищаемом” участке кода делается попытка открыть файл для чтения.
Если файл не существует, но генерируется исключение и управление передаётся блоку **except**, *иначе выполняется блок* **else**:

```
try:
    f = open("access.log", "r")
except FileNotFoundError as err:
    print('File don’t open or read.')
    print('Name of file: ', err.filename)
else:
    a = f.readline()

# Переменная f, объявленная в блоке try, доступна и в блоке else.
```

PYTHON

Тема 4: Обработка ошибок

Блок **finally**

```
# После блоков except и else может быть определён блок finally, который выполняется в любом случае:

f = open("datafile.txt", "r")
try:
    str_value = f.readline()
    a = int(str_value)
except ValueError as err:
    print("Conversion error")
finally:
    f.close()

# Файл закроется в любом случае.
```

PYTHON

Тема 4: Обработка ошибок

Ввод данных с клавиатуры

```
host_n = int(input('Input host number'))
port_n = int(input('Input port number'))
print('Socket host{:2d}.example.com:{:d}'.format(host_n, port_n))

# При вводе не числовых значений программа сообщит об ошибке и остановится:

>>> Input host number: ten
>>> . . . . . .
ValueError: could not convert string to int: 'ten'

# Это плохая реакция программы на ошибку: нет возможности исправить ошибку не
перезапуская программу
```

PYTHON

Тема 4: Обработка ошибок

Контроль ввода данных

```
def input_as(text, type_of_value):
    is_bad_input = True
    while is_bad_input:
        try:
            val = input(text)
            val = type_of_value(val)
            is_bad_input = False
        except ValueError as err:
            print("It doesn't " + type_of_value.__name__ + ", try again.")
    return val


val = input_as("Input integer: ", int)
print( "Input value", val)
```