

Семинар 4

Введение в программирование на Python

Папулин С.Ю.
papulin_hse@mail.ru

2015

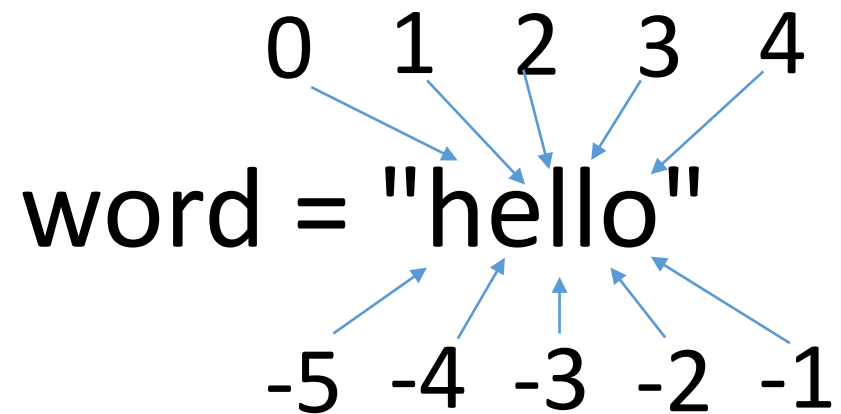
1. Циклы в Python
2. Функция `range()`
3. Другие полезные возможности
4. Примеры задач
5. Самостоятельная работа

1. Строки
2. Списки
3. Функции
4. Модули
5. Пакеты

Строки

Строка (string). Строка состоит из множества символов в определенной последовательности с произвольным доступом. Элементы строки неизменяемы (immutable).

К элементам строки обращаются с использованием индекса начиная с 0.



word[0]->"h"

word[2]->"l"

word[4]->"o"

word[-5]->"h"

word[-3]->"l"

word[-1]->"o"

Строки

Обозначение	Операция	Пример
$x + y$	Объединение	"text1"+"text2"="text1text2"
$x * y$	Повторение y раз	"For"*3="ForForFor"

len(string)	Количество символов в строке string	len("hello")=5
-------------	-------------------------------------	----------------

Ключевое слово IN и NOT IN

```
myString="Hello, World!"
```

```
"el" in myString
```

True

```
"el" not in myString
```

False

Срезы (slice)

Символы от start до stop – [start, stop)

word = "hello"

word[*start:stop*]

word[0:2]->"he"

word[:2]->"he"

word[2:5]->"llo"

word[2:]->"llo"

word[-5:5]->"hello"

word[:]->"hello"

Срезы (slice)

Символы от start до stop – [start, stop) – шагом step

word = "hello"

word[*start:stop:step*]

word[:5:1]->"hello"

word[4::-1]->"olleh"

word[0:5:1]->"hello"

word[4:0:-1]->"olle"

word[::2]->"hlo"

Метод find/rfind

`string.find(substr)` – ищет подстроку *substr* в строке `string` и возвращает индекс начала первой найденной подстроки или -1, если подстроки нет.

```
myString="Hello, World!"
```

```
myString.find("l") ->2
```

```
myString.find("World") ->7
```

`string.rfind(substr)`

```
myString.rfind("l") ->10
```

```
myString.find("World") ->7
```

`string.replace(substr1, substr2)` – заменяет подстроку *substr1* на подстроку *substr2*

```
myString.replace("o", "ooo")
```

```
'Hellooo, Woorld!'
```

Методы split и partition

`string.split(sep)` – разбивает строку на подстроки, используя *sep* как разделитель, и возвращает список подстрок.

```
myString.split(',')  
['Hello', ' World!']
```

`string.partition(sep)` – возвращает кортеж из трех элементов: 1 – подстрока до первого *sep*; 2 – *sep*; 3 – подстрока после *sep*.

```
myString.partition("o")  
('Hell', 'o', ', World!')
```

Методы join и strip

sep.join(strings) – объединяет строки через *sep*.

```
myStrings=["string1","string2","string3"]  
"::".join(myStrings)  
'string1::string2::string3'
```

string.strip(*chars*)

```
'    spacious    '.strip()  
'spacious'  
  
'www.example.com'.strip('cmowz. ')  
'example'
```

Метод count

`string.count(substr)` – количество вхождений подстроки *substr* в строке `string`

```
words = "Hello, World!"  
c = words.count("l")  
print(c)
```



3

string.upper() и **string.lower()**

```
myString.upper()  
'HELLO, WORLD!'
```

```
myString.lower()  
'hello, world!'
```

string.isdigit() и **string.isalpha()**

```
myStrNum = "3542"  
myStrNum.isdigit()  
  
True
```

```
myStrS = "Hello"  
myStrS.isalpha()  
  
True
```

reversed для строки

reversed(*string*) возвращает итеративный объект с инвертированными символами строки *string*.

```
s1 = "my string"  
s2 = reversed(s1)
```

```
print(list(s1))  ➡ ['m', 'y', ' ', 's', 't', 'r', 'i', 'n', 'g']  
print(list(s2))  ➡ ['g', 'n', 'i', 'r', 't', 's', ' ', 'y', 'm']
```

Списки

Список (list). Множество элементов, каждый из которых имеет значение и позицию в списке.

```
l = [1, 3, 'string', [1, 2, 3]]
```

```
l[0] -> 1
```

```
l[2] -> 'string'
```

```
l[3] -> [1, 2, 3]
```

Строки

Обозначение	Операция	Пример
$x + y$	Объединение	$[1,2,3]+[4,5,6]=[1,2,3,4,5,6]$
$x * y$	Повторение y раз	$[1,'a']*3=[1,'a',1,'a',1,'a']$

<code>len(list)</code>	Количество элементов в списке <code>list</code>	<code>len([1,2,3])=3</code>
------------------------	---	-----------------------------

Ключевое слово IN и NOT IN

```
myList=[1, 2, 3, 5]
```

```
3 in myList
```

True

```
3 not in myList
```

False

numbers[*start:stop*]

numbers[*start:stop:step*]

numbers =[4,2,6,4,7]

numbers[0:2]->[4,2]

numbers[:2] ->[4,2]

numbers::-1]->[7, 4, 6, 2, 4]

numbers:::2]->[4, 6, 7]

Методы append и extend


list.append(*el*) – добавляет элемент *el* в конец списка list

```
numbers = [4, 2, 6, 4, 7]
```

```
numbers.append(8)            [4, 2, 6, 4, 7, 8]
```

list.extend(*list1*) – добавляет в конец списка list элементы списка *list1*


```
numbers = [4, 2, 6, 4, 7]
```

```
numbers.extend([8, 9, 10])            [4, 2, 6, 4, 7, 8, 9, 10]
```

Методы insert и remove

`list.insert(indx, el)` – добавляет элемент *el* в список list в позицию *indx*

```
numbers = [4, 2, 6, 4, 7]
```

```
numbers.insert(2, 33)            [4, 2, 33, 6, 4, 7]
```

`list.remove(el)` – удаляет первый встретившийся элемент со значением *el*

```
numbers = [4, 5, 7, 7, 5, 4]
```

```
numbers.remove(7)            [4, 5, 7, 5, 4]
```

Метод index

`list.index(el)` – возвращает индекс первого элемента *el* (слева) в списке list

`list.index(el, start)` – возвращает индекс первого элемента *el* (слева) в списке list начиная с индекса *start*

`list.index(el, start, stop)` – возвращает индекс первого элемента *el* (слева) в списке list начиная с индекса *start*

```
numbers = [4, 5, 7, 7, 5, 4]
```

```
numbers.index(5) → 1
```

```
numbers.index(5, 2) → 4
```

```
numbers.index(5, 1, 5) → 1
```

Метод count, sort

`list.count(el)` – количество вхождений элемента *el* в списке list

```
numbers = [4, 5, 7, 7, 5, 4]
```

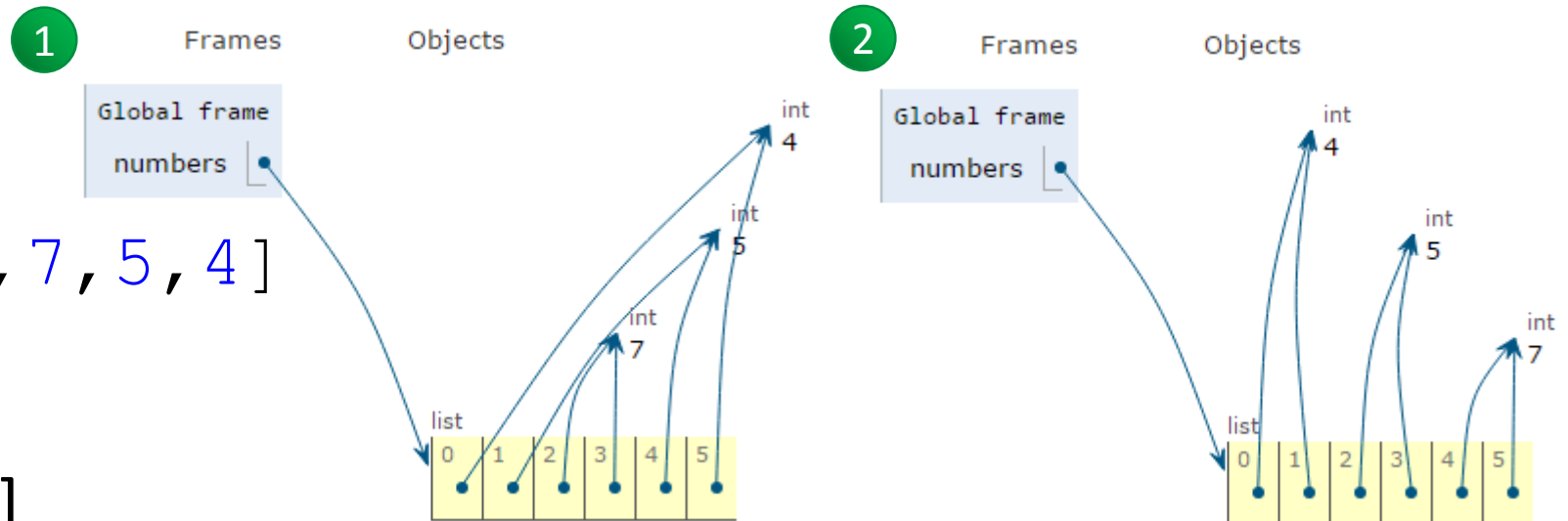
```
numbers.count(7) ➔ 2
```

`list.sort()` – сортирует список по возрастанию

1 `numbers = [4, 5, 7, 7, 5, 4]`

2 `numbers.sort()`

[4, 4, 5, 5, 7, 7]



Метод sort, reverse и clear

list.sort(key, reverse) – сортирует список list

reverse = True – по убыванию; False – по возрастанию

```
numbers = [4, 5, 7, 7, 5, 4]  
numbers.sort(reverse = True)
```

[7, 7, 5, 5, 4, 4]

list.reverse() – инвертирует список list

```
numbers = [4, 2, 6, 4, 7]  
numbers.reverse() → [7, 4, 6, 2, 4]
```


list.clear() – очищает список list

```
numbers.clear() → []
```


reversed для списка

reversed(*list*) возвращает итеративный объект с инвертированной последовательностью элементов списка *list*.

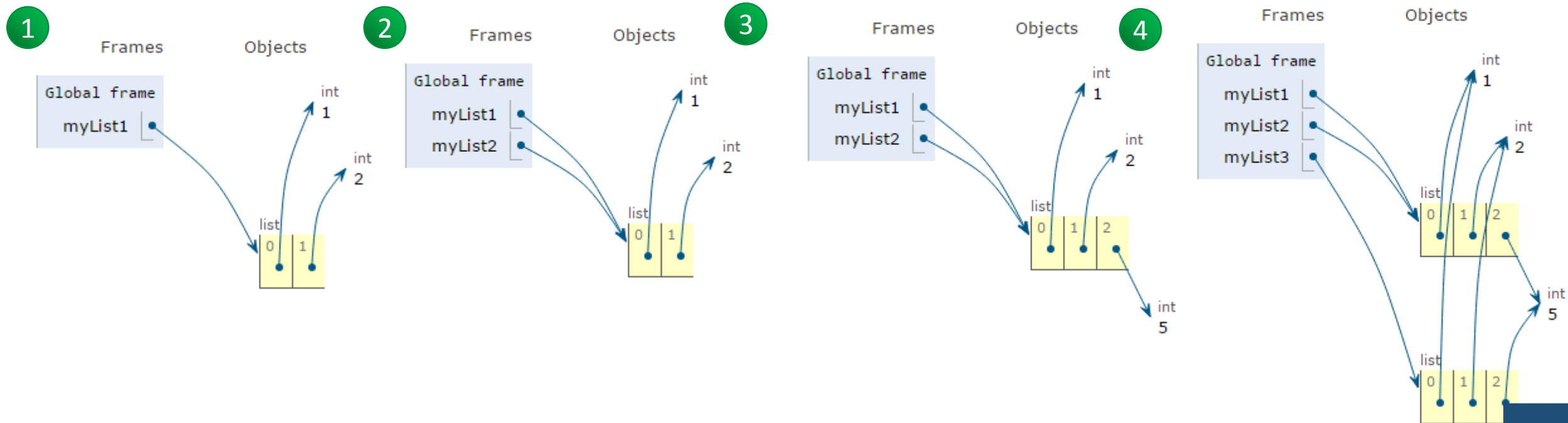
```
numbers1 = [1, 2, 3, 4, 5]  
numbers2 = list(reversed(numbers1))
```

<code>print(numbers1)</code>		<code>[1, 2, 3, 4, 5]</code>
<code>print(numbers2)</code>		<code>[5, 4, 3, 2, 1]</code>

Метод copy


list.copy() – копирует список

- 1 `myList1 = [1, 2]`
- 2 `myList2 = myList1`
- 3 `myList2.append(5)`
- 4 `myList3 = myList1.copy()`




1 `l = [1, 2, 3, 4]`

`lNew = [x**2 for x in l]`  `[1, 4, 9, 16]`

`lNew = [x**2 for x in l if x > 2]`  `[9, 16]`

2 `a = [4, 5, 3, 8, 2]`

`lNew = [(indx, el) for indx, el in enumerate(a)]`



`[(0, 4), (1, 5), (2, 3), (3, 8), (4, 2)]`

3 `[[i, j] for i in range(2) for j in range(3)]`



`[[0, 0], [0, 1], [0, 2], [1, 0], [1, 1], [1, 2]]`

Функции sum, zip и del

1 `sum([4, 5, 3, 8, 2])`  22

2 `a = [4, 5, 3, 8, 2]`
`b = [3, 4, 7, 2]`
`c = zip(a, b)`
`list(c)`


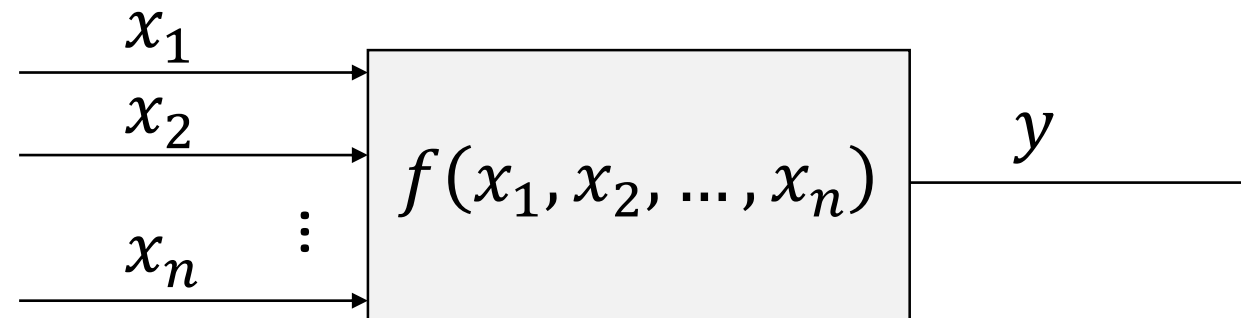
`[(4, 3), (5, 4), (3, 7), (8, 2)]`

3 `f = [1, 2, 3, 4, 5]`
`del` `f[0:2]`  `[3, 4, 5]`

Функции

Функция

Функция в Python – организованный блок программы, предназначенный для выполнения действий (указанных в теле функции) с возможностью повторного использования. Функция может принимать аргументы и возвращать результат выполнения.



Определение функции

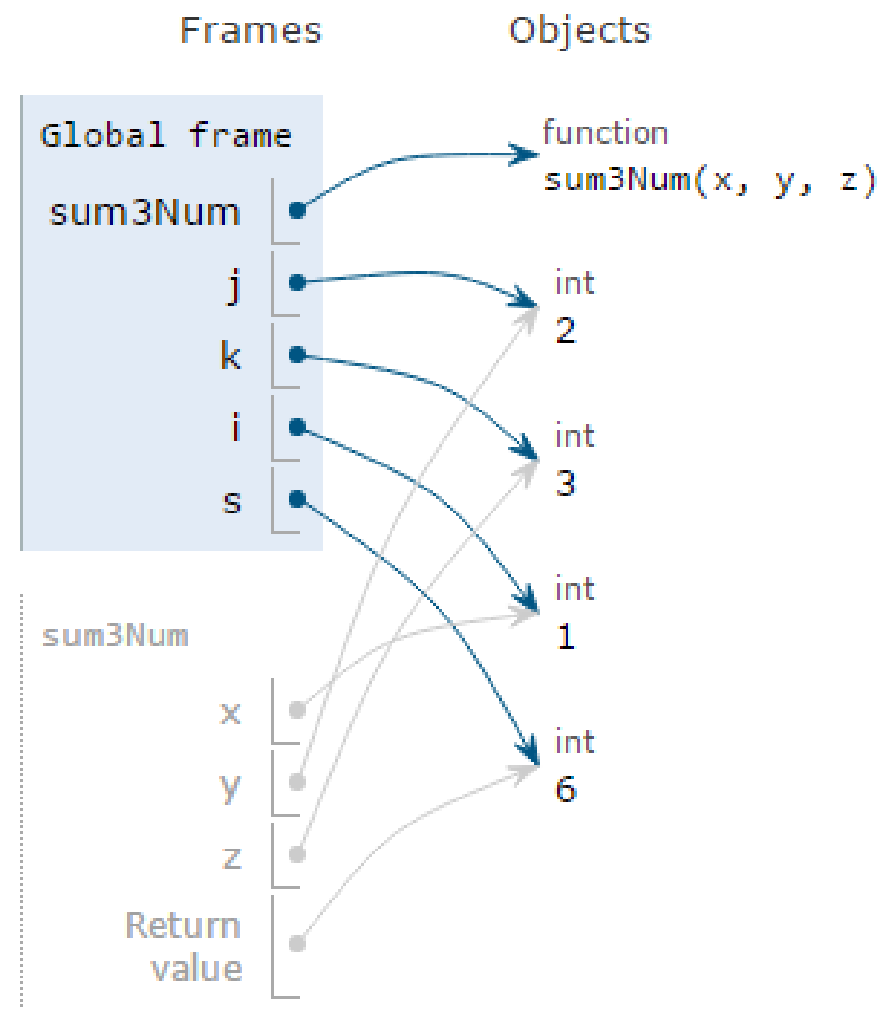
def имя_функции(аргументы):
 действия
 return выражение

```
def sum3Num(x, y, z):  
    return x + y + z
```

```
i = 1; j = 2; k = 3  
s = sum3Num(i, j, k)
```

```
print("Функция sum3Num: 1 + 2 + 3 = %i" % s)
```

Функция sum3Num: 1 + 2 + 3 = 6

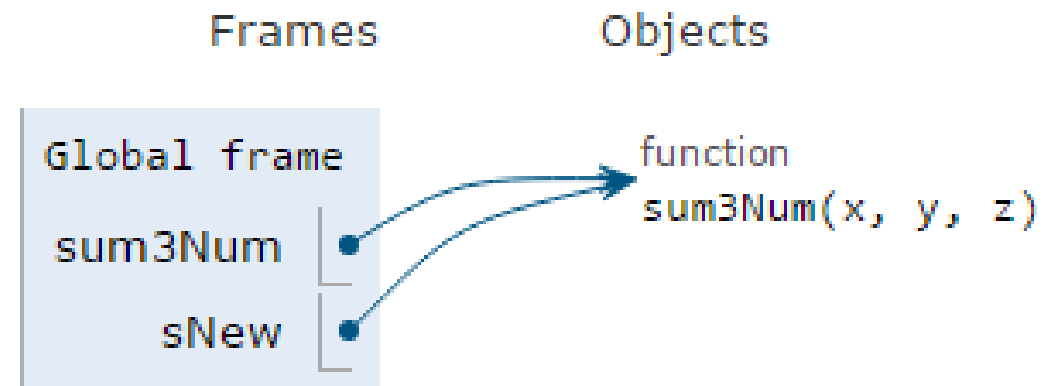


Присвоение функций

```
def sum3Num(x, y, z):  
    return x + y + z
```

```
sNew = sum3Num  
print("Функция sum3Num: 1 + 2 + 3 = %i" % sum3Num(1, 2, 3))  
print("Функция sNew: 1 + 2 + 3 = %i" % sNew(1, 2, 3))
```

Функция sum3Num: 1 + 2 + 3 = 6
Функция sNew: 1 + 2 + 3 = 6

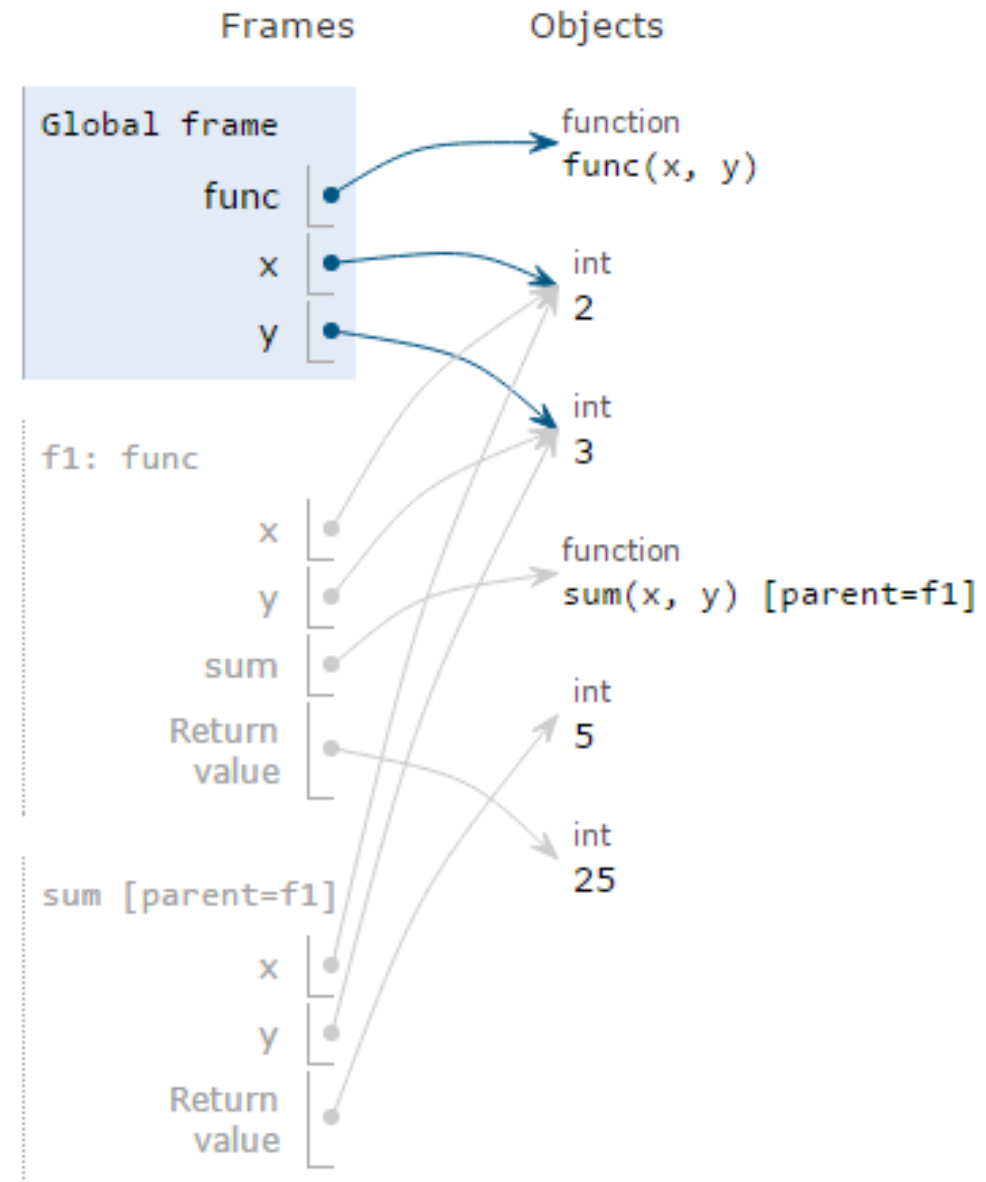


Функция в функции

```
def func(x, y):  
    def sum(x, y):  
        return x+y  
    return sum(x, y) ** 2
```

```
x = 2  
y = 3  
print("(%i + %i)^2 = %i" % (x, y, func(2, 3)))
```

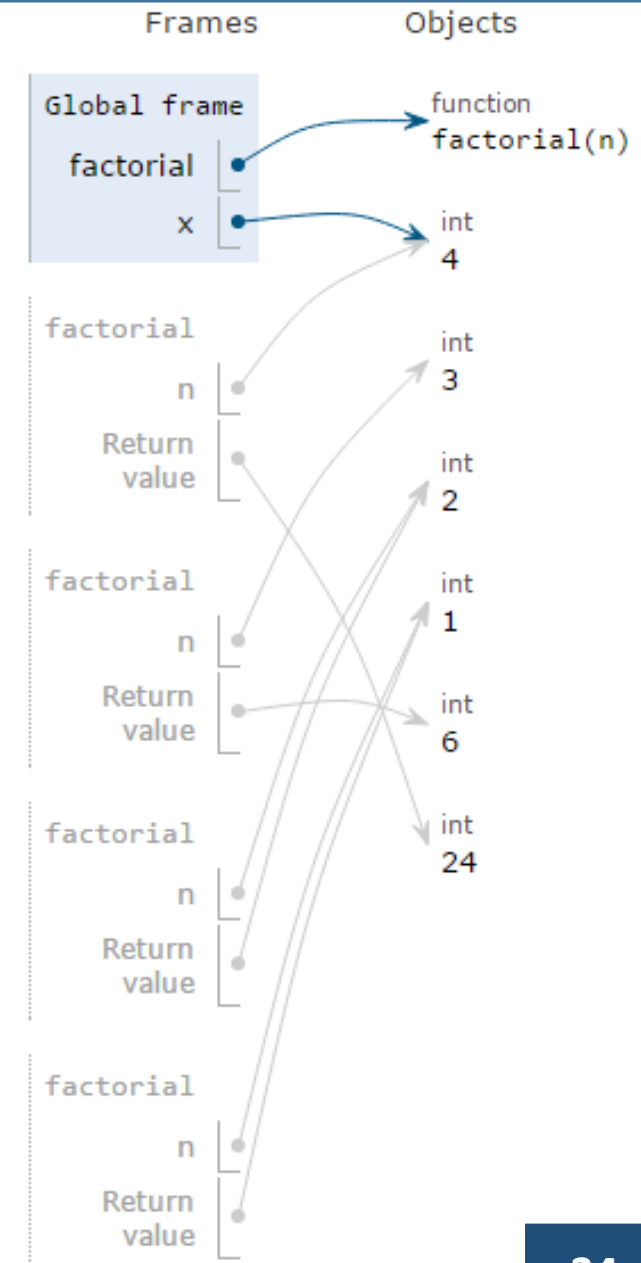
$(2 + 3)^2 = 25$



Рекурсия

```
def factorial(n):  
    if n == 1:  
        return 1  
    else:  
        return n * factorial(n-1)  
  
x = 4  
print("Факториал %i! = %i" % (x, factorial(x)))
```

Факториал $4! = 24$



Аргументы по умолчанию, позиционные и keyword аргументы

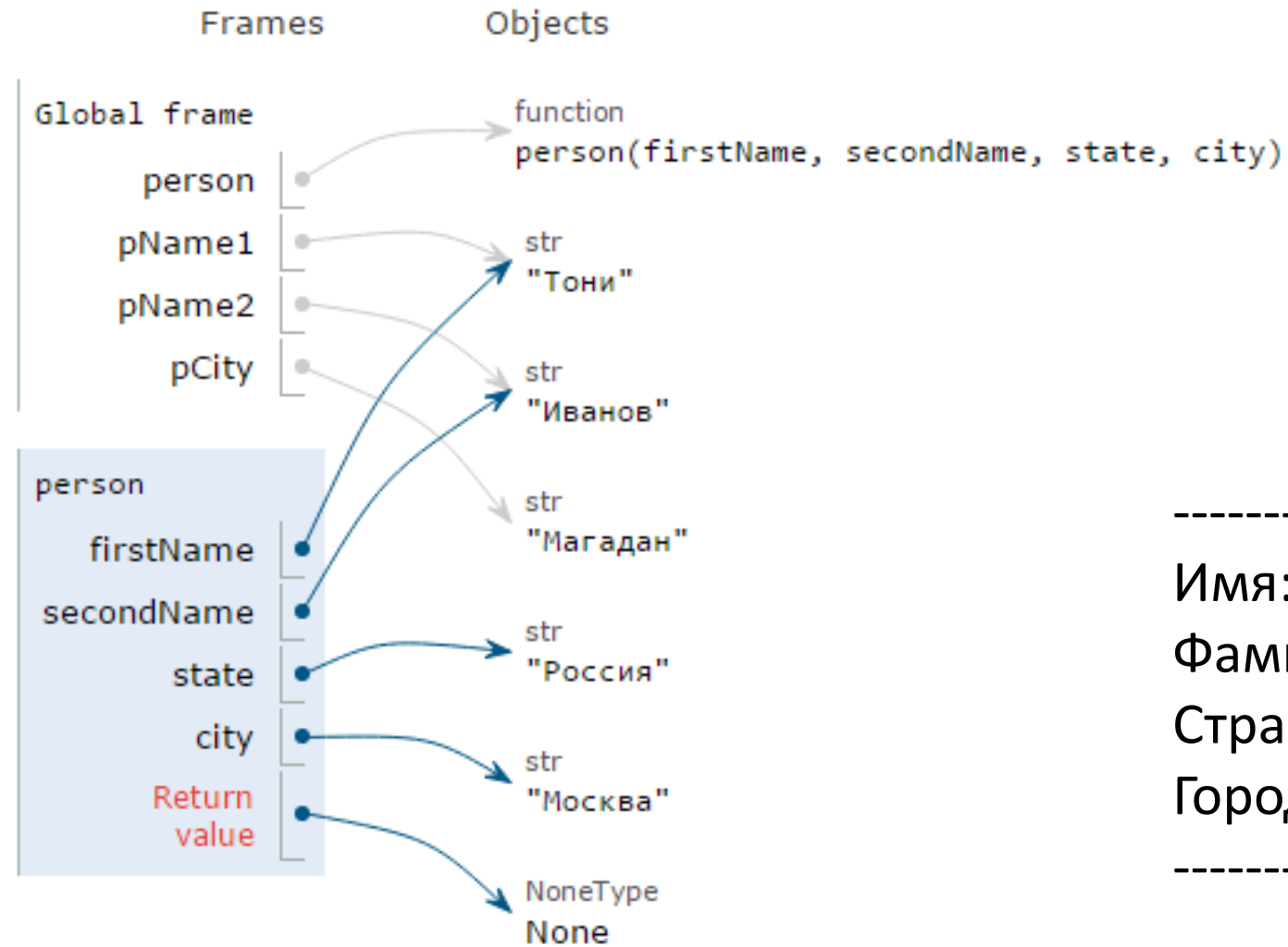
```
def person(firstName, secondName, city="Москва", state="Россия"):  
    print("-----")  
    print("Имя: " + firstName)  
    print("Фамилия: " + secondName)  
    print("Страна: " + state)  
    print("Город: " + city)  
    print("-----")  
    return
```

```
pName1 = "Тони"  
pName2 = "Иванов"  
pCity = "Магадан"
```

```
1 person(pName1, pName2)  
2 person(pName1, pName2, pCity)  
3 person(pName1, pName2, city="Мадрид")  
4 person(pName1, pName2, "Мадрид", "Испания")  
5 person(pName1, pName2, state = "Испания", city="Мадрид")  
6 person(state = "Испания", city="Мадрид", firstName = pName1, secondName =  
pName2)
```

Аргументы по умолчанию

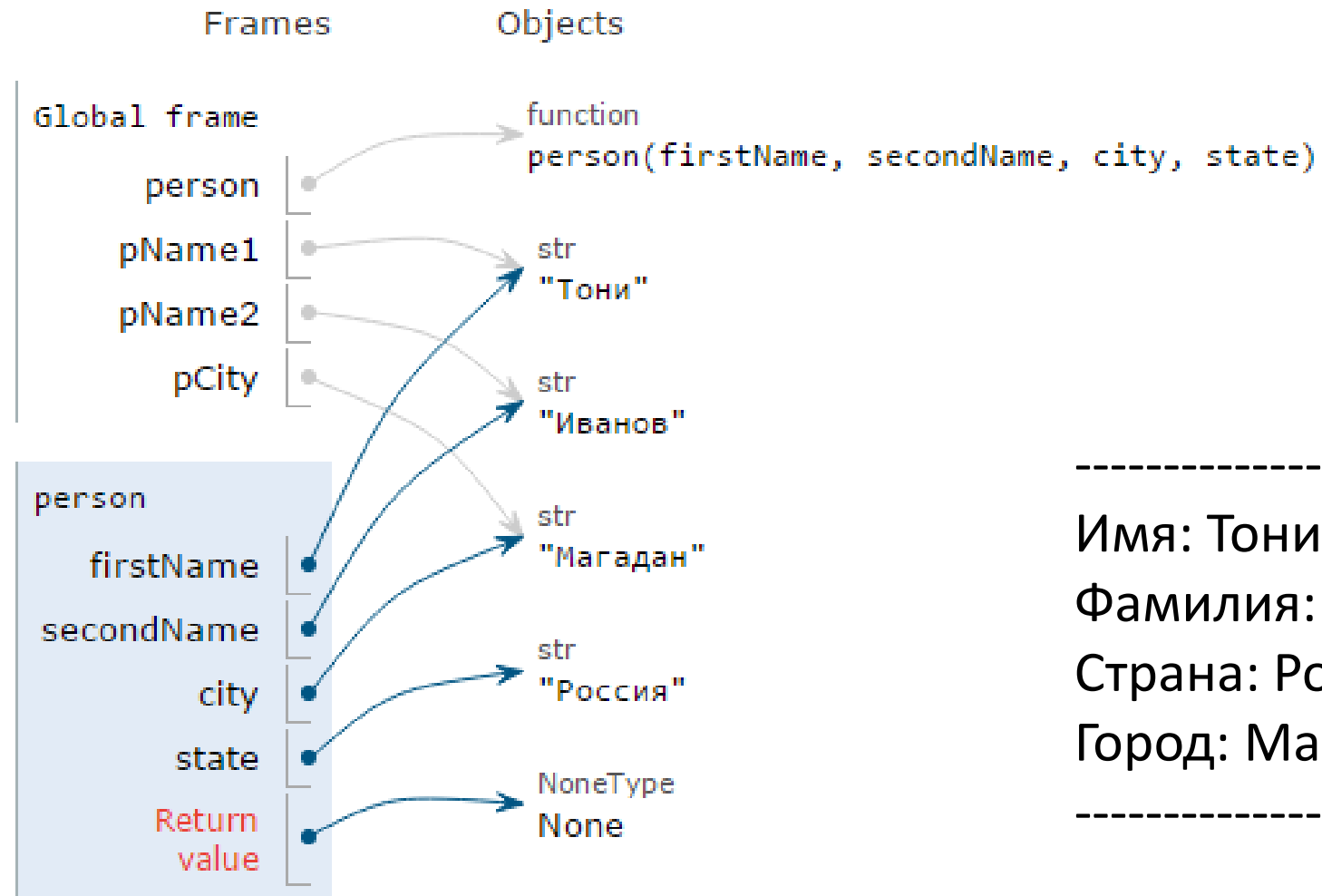
1 `person(pName1, pName2)`



Имя: Тони
Фамилия: Иванов
Страна: Россия
Город: Москва

Аргументы по умолчанию

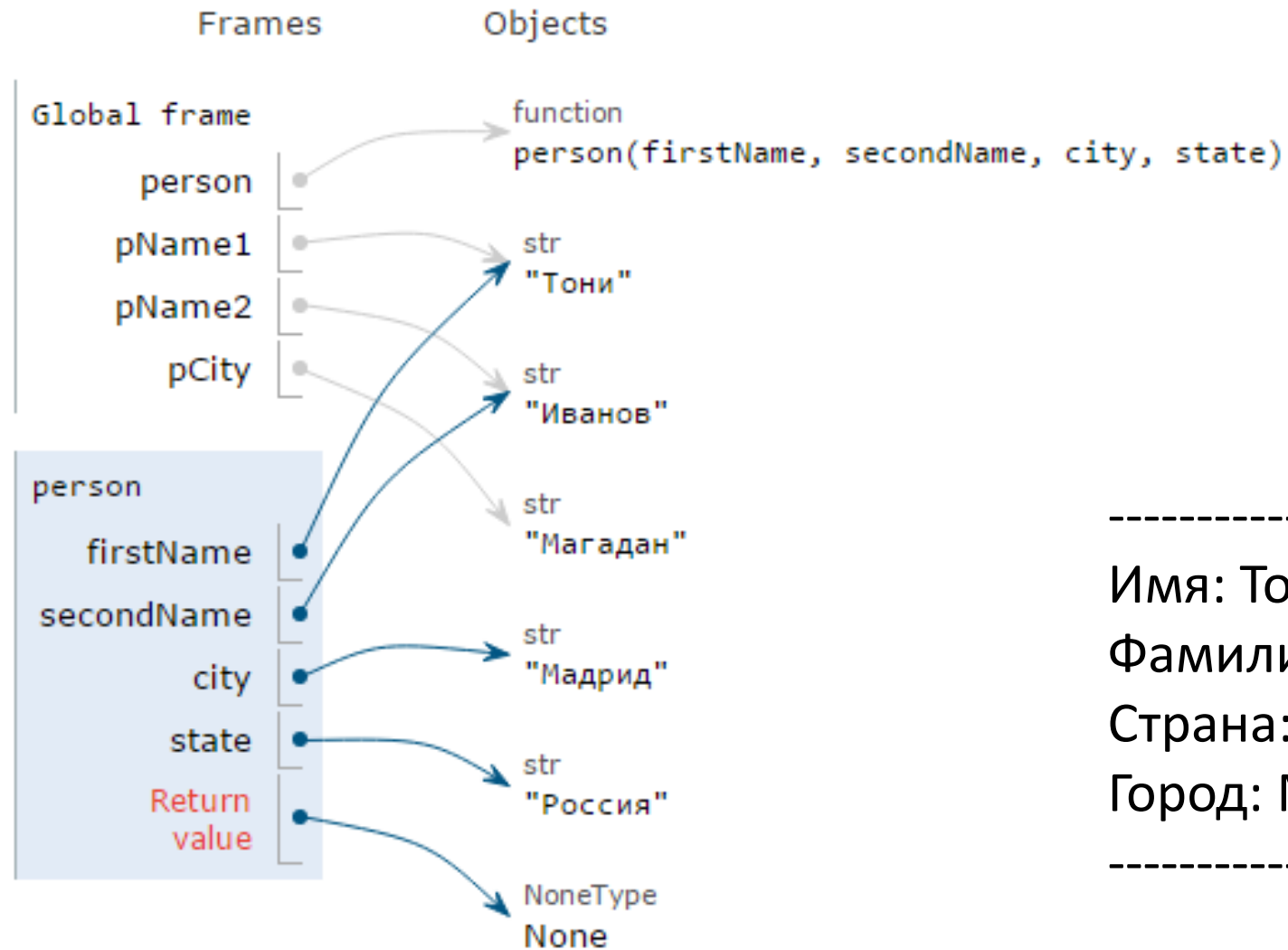
2 `person(pName1, pName2, pCity)`



Имя: Тони
Фамилия: Иванов
Страна: Россия
Город: Магадан

Анонимные функции

3 person(pName1, pName2, city="Мадрид")



Имя: Тони
Фамилия: Иванов
Страна: Россия
Город: Мадрид

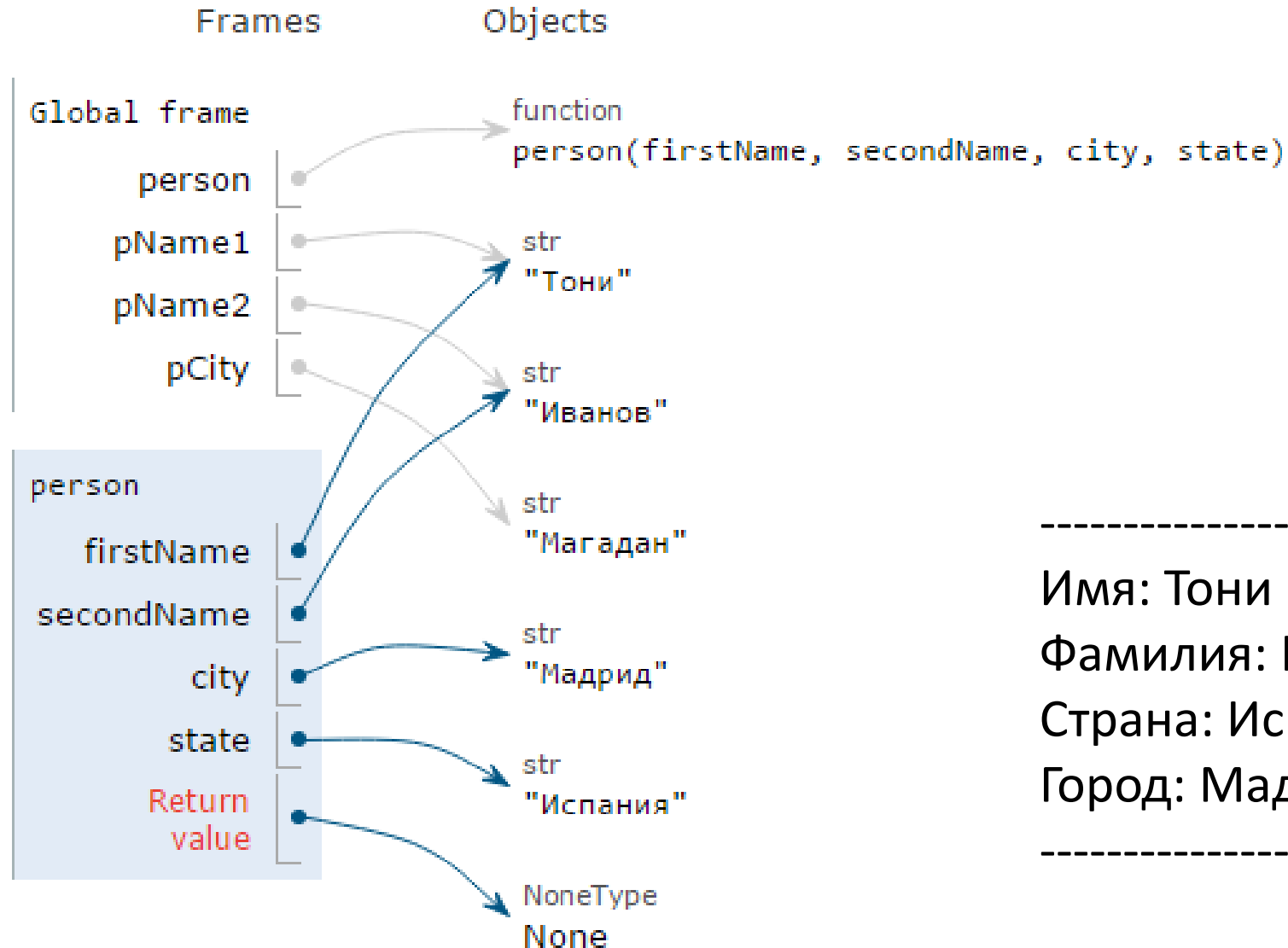
Анонимные функции

4 `person(pName1, pName2, "Мадрид", "Испания")`



Анонимные функции

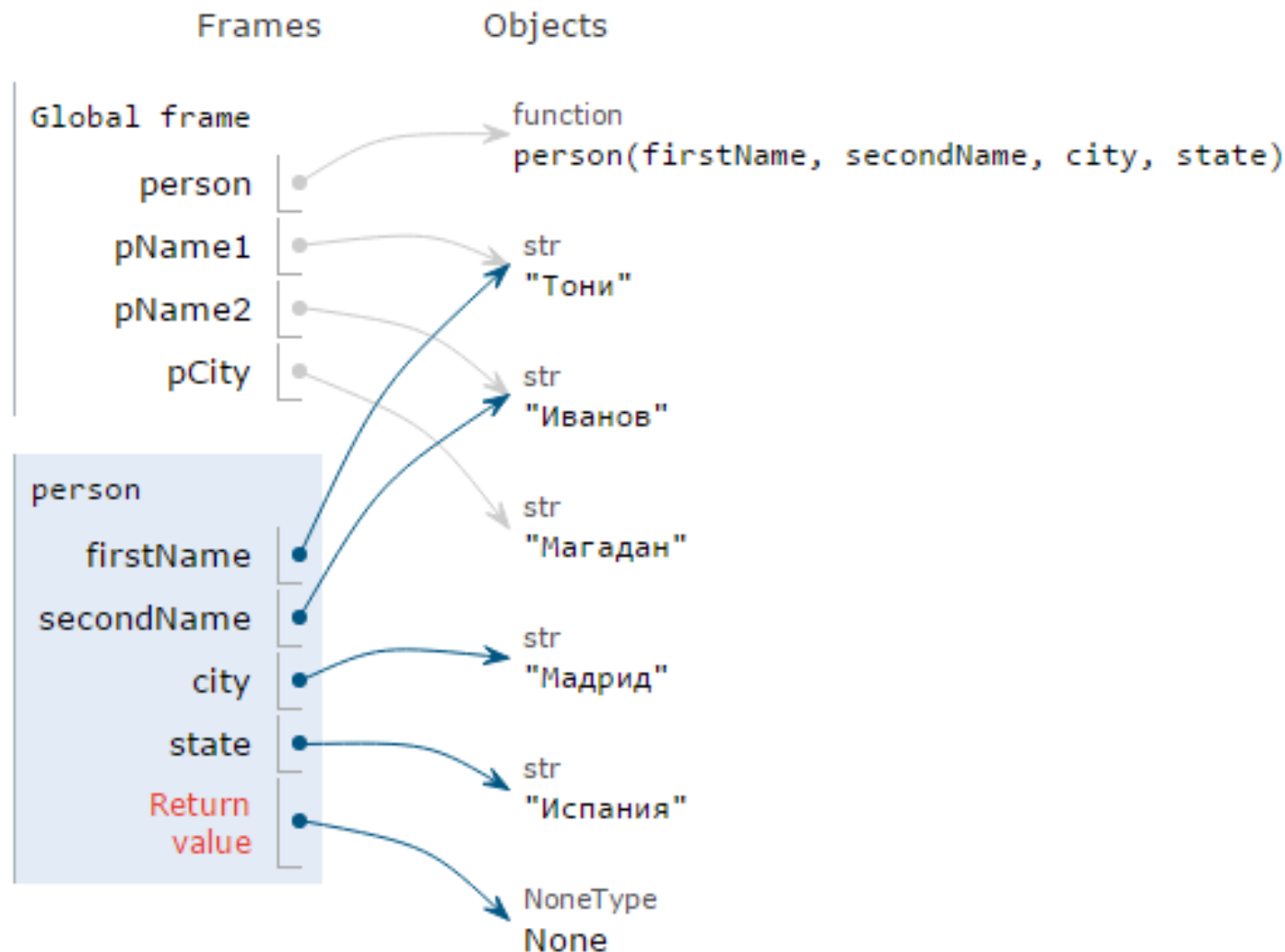
5 `person(pName1, pName2, state = "Испания", city="Мадрид")`



Имя: Тони
Фамилия: Иванов
Страна: Испания
Город: Мадрид

Анонимные функции

6 `person(state = "Испания", city="Мадрид", firstName = pName1, secondName = pName2)`



Имя: Тони
Фамилия: Иванов
Страна: Испания
Город: Мадрид

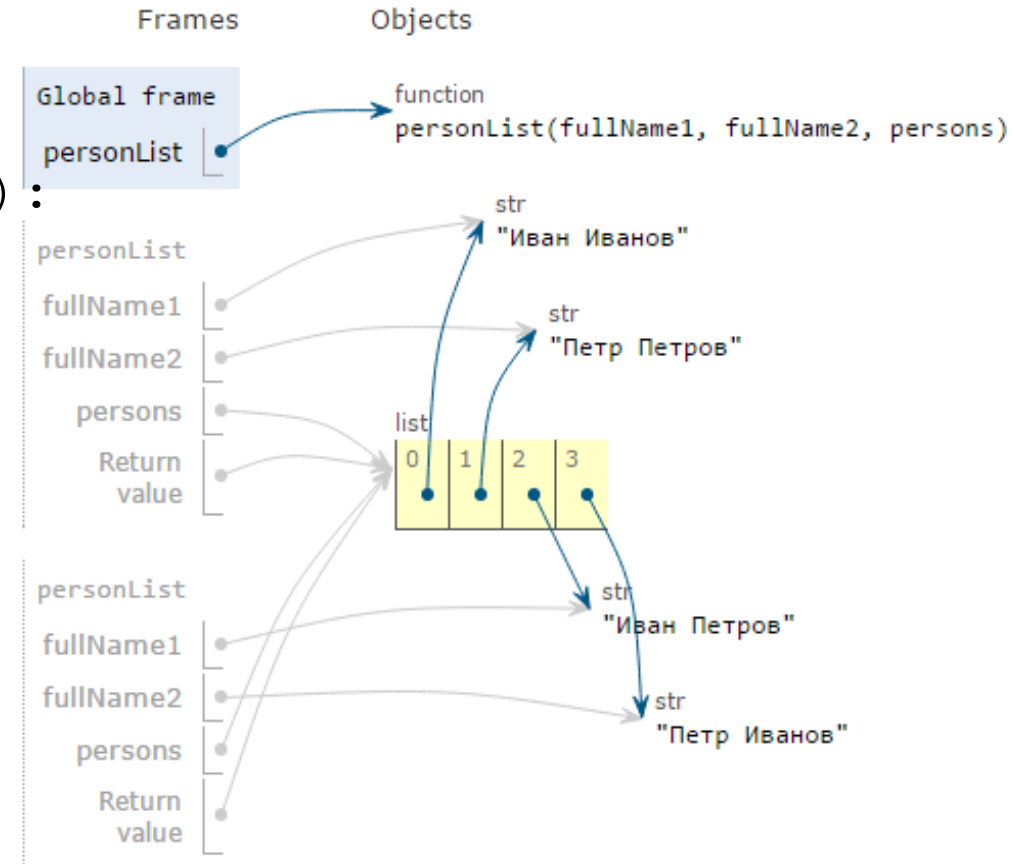
Аргументы по умолчанию: список, словарь, класс

Аргумент с изменяемым объектом (список, словарь, класс) будет общим для всех вызовов функции.

```
def personList(fullName1, fullName2, persons=[]):  
    persons.append(fullName1)  
    persons.append(fullName2)  
    return persons
```

```
print(personList("Иван Иванов", "Петр Петров"))  
print(personList("Иван Петров", "Петр Иванов"))
```

```
['Иван Иванов', 'Петр Петров']  
['Иван Иванов', 'Петр Петров', 'Иван Петров', 'Петр Иванов']
```



Аргументы по умолчанию: список, словарь, класс

```
def personList(fullName1, fullName2, persons=None):
```

```
    if (persons == None):
```

```
        persons = []
```

```
    persons.append(fullName1)
```

```
    persons.append(fullName2)
```

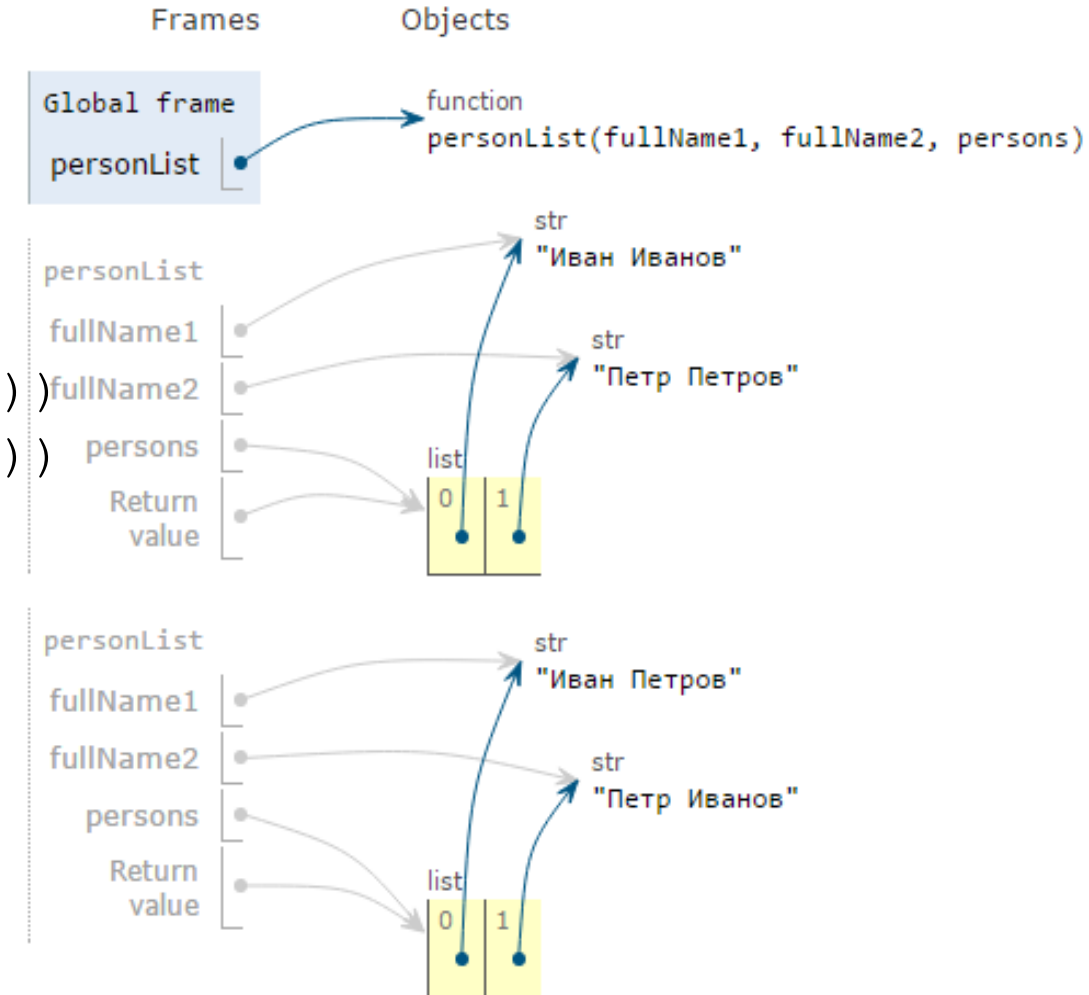
```
    return persons
```

```
print(personList("Иван Иванов", "Петр Петров"))
```

```
print(personList("Иван Петров", "Петр Иванов"))
```

```
['Иван Иванов', 'Петр Петров']
```

```
['Иван Петров', 'Петр Иванов']
```



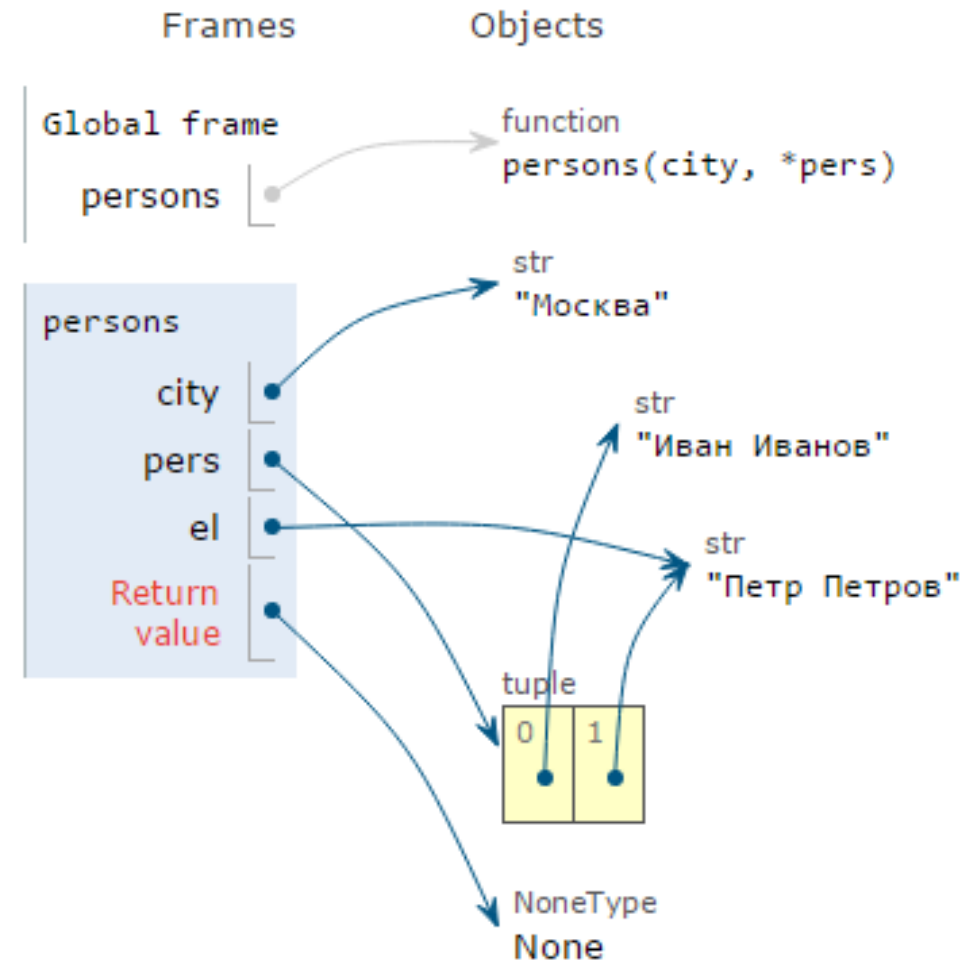
Аргумент в виде списка

```
def persons(city, *pers):  
    print(city+"-----")  
    for el in pers:  
        print(el)  
persons("Москва", "Иван Иванов", "Петр Петров")
```

Москва-----

№1: Иван Иванов

№1: Петр Петров

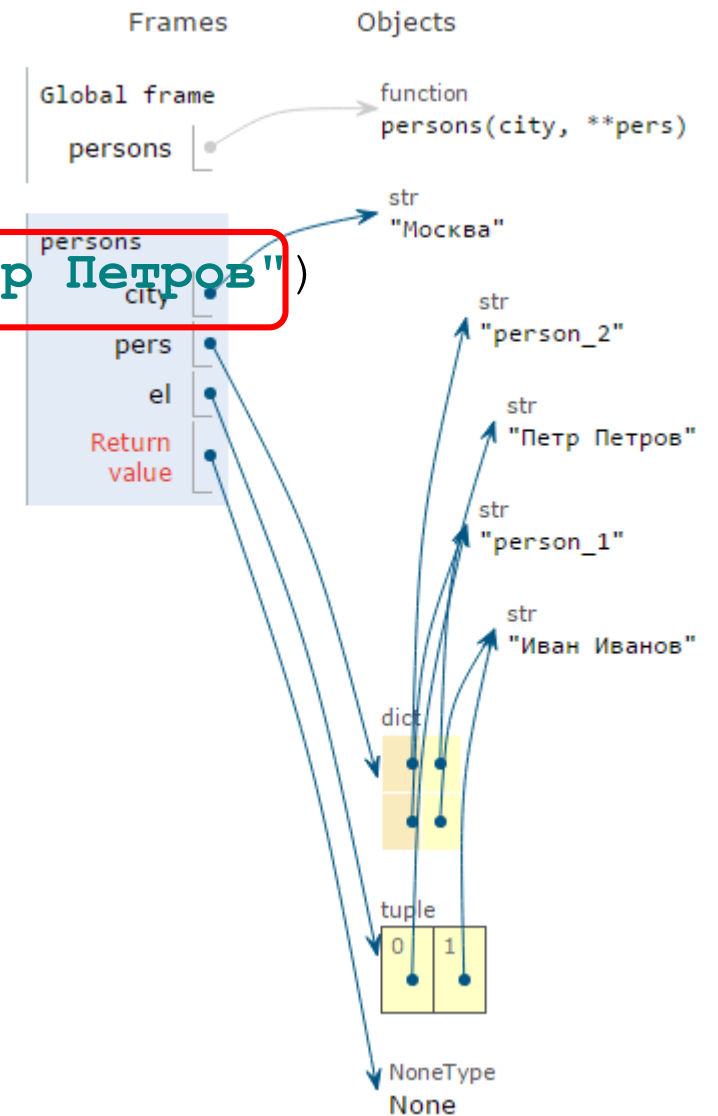


Аргумент в виде словаря

```
def persons(city, **pers):  
    print(city+"-----")  
    for el in pers.items():  
        print(el[0] + " = " + el[1])
```

```
persons("Москва", person_1="Иван Иванов", person_2="Петр Петров")
```

Москва-----
person_2 = Петр Петров
person_1 = Иван Иванов

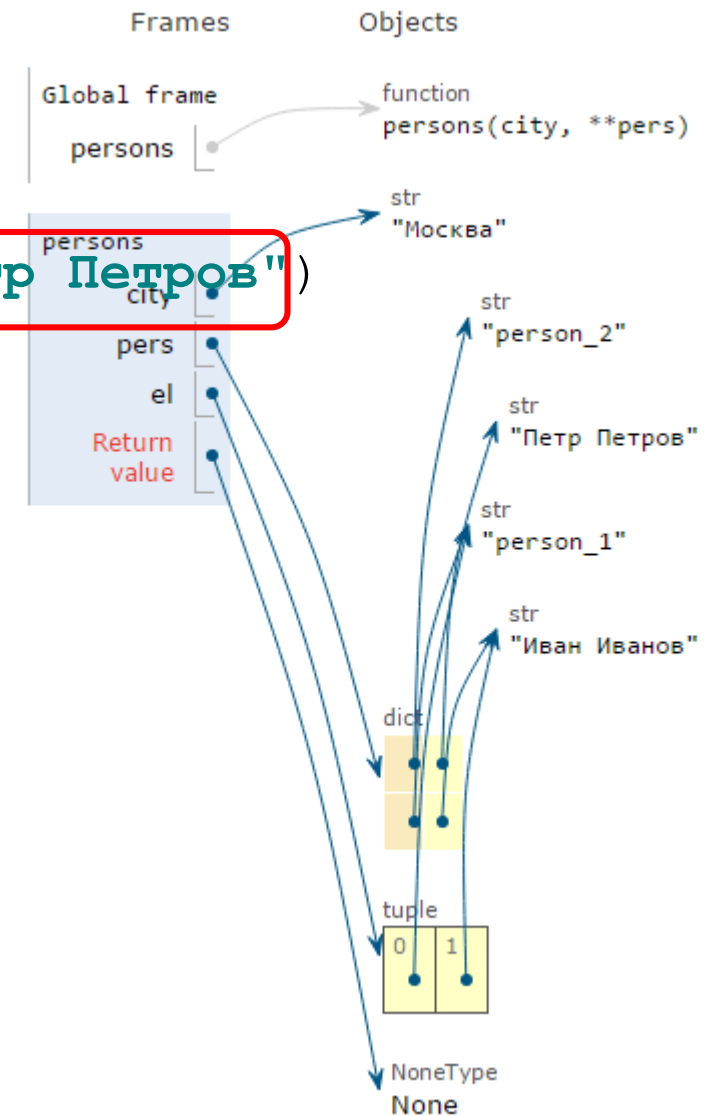


Аргумент в виде словаря

```
def persons(city, **pers):  
    print(city+"-----")  
    for el in pers.items():  
        print(el[0] + " = " + el[1])
```

```
persons("Москва", person_1="Иван Иванов", person_2="Петр Петров")
```

Москва-----
person_2 = Петр Петров
person_1 = Иван Иванов

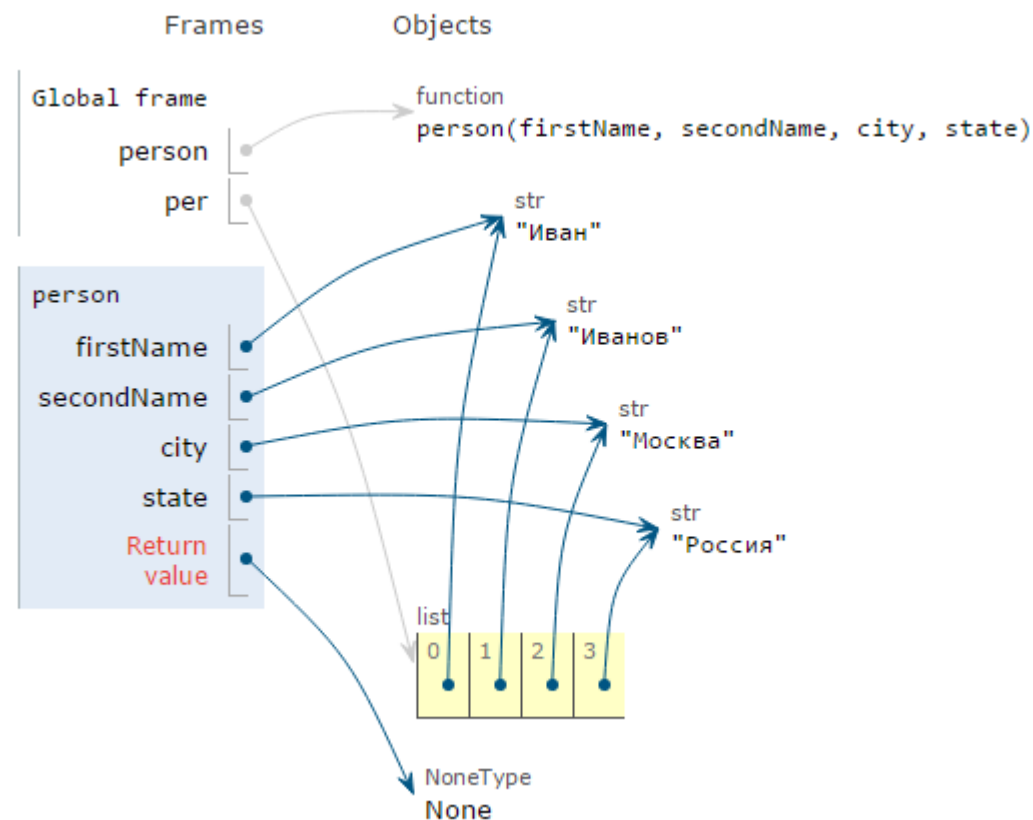


Распаковка списка аргументов: список

```
def person(firstName, secondName, city, state):  
    print("-----")  
    print("Имя: " + firstName)  
    print("Фамилия: " + secondName)  
    print("Страна: " + state)  
    print("Город: " + city)  
    print("-----")  
    return
```

```
per = ["Иван", "Иванов", "Москва", "Россия"]  
person(*per)
```

```
-----  
Имя: Иван  
Фамилия: Иванов  
Страна: Россия  
Город: Москва  
-----
```

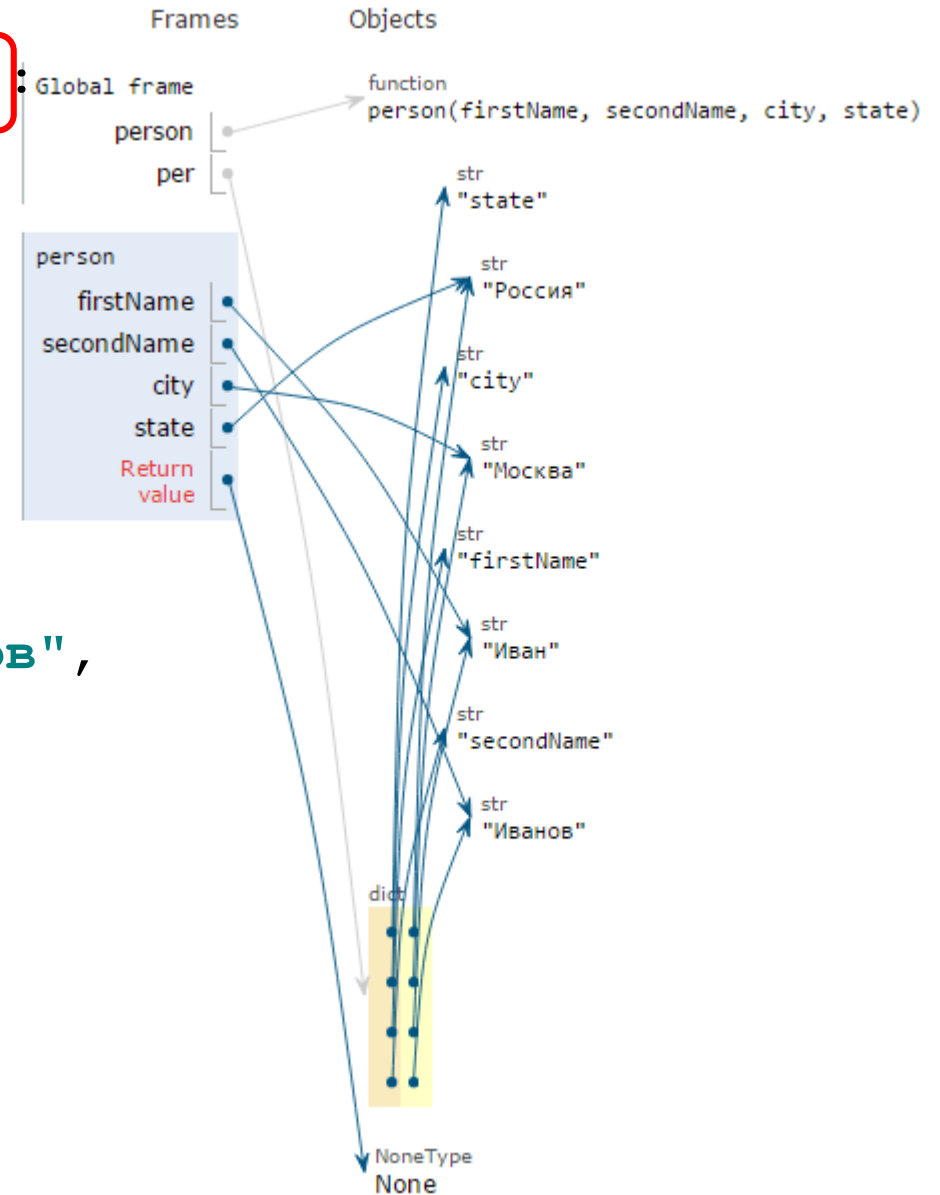


Распаковка списка аргументов: словарь

```
def person(firstName, secondName, city, state):  
    print("-----")  
    print("Имя: " + firstName)  
    print("Фамилия: " + secondName)  
    print("Страна: " + state)  
    print("Город: " + city)  
    print("-----")  
    return
```

```
per = {"firstName": "Иван", "secondName": "Иванов",  
      "city": "Москва", "state": "Россия"}  
person(**per)
```

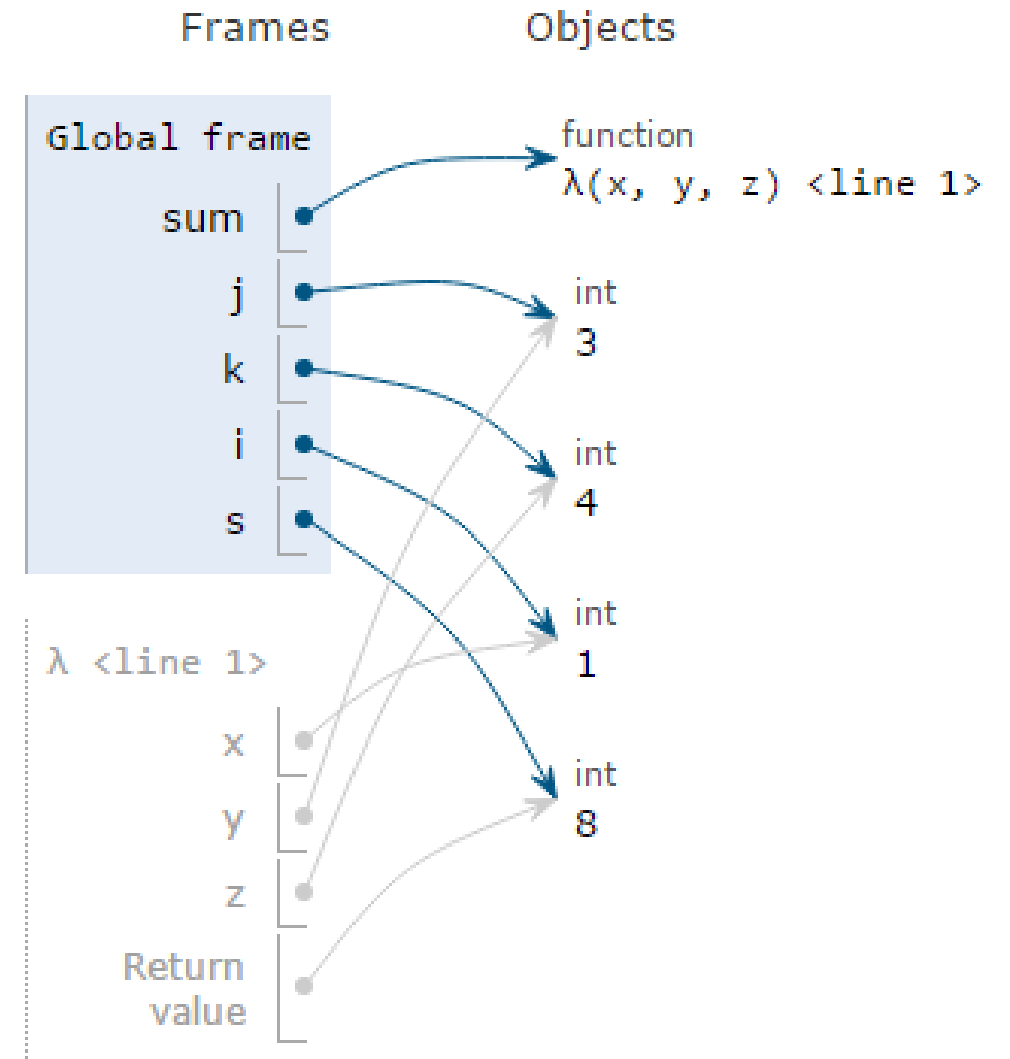
```
-----  
Имя: Иван  
Фамилия: Иванов  
Страна: Россия  
Город: Москва  
-----
```



LAMBDA

```
sum = lambda x,y,z: x+y+z  
i = 1; j = 3; k = 4  
s = sum(i, j, k)  
print("1 + 3 + 4 = " + str(s))
```

1 + 3 + 4 = 8



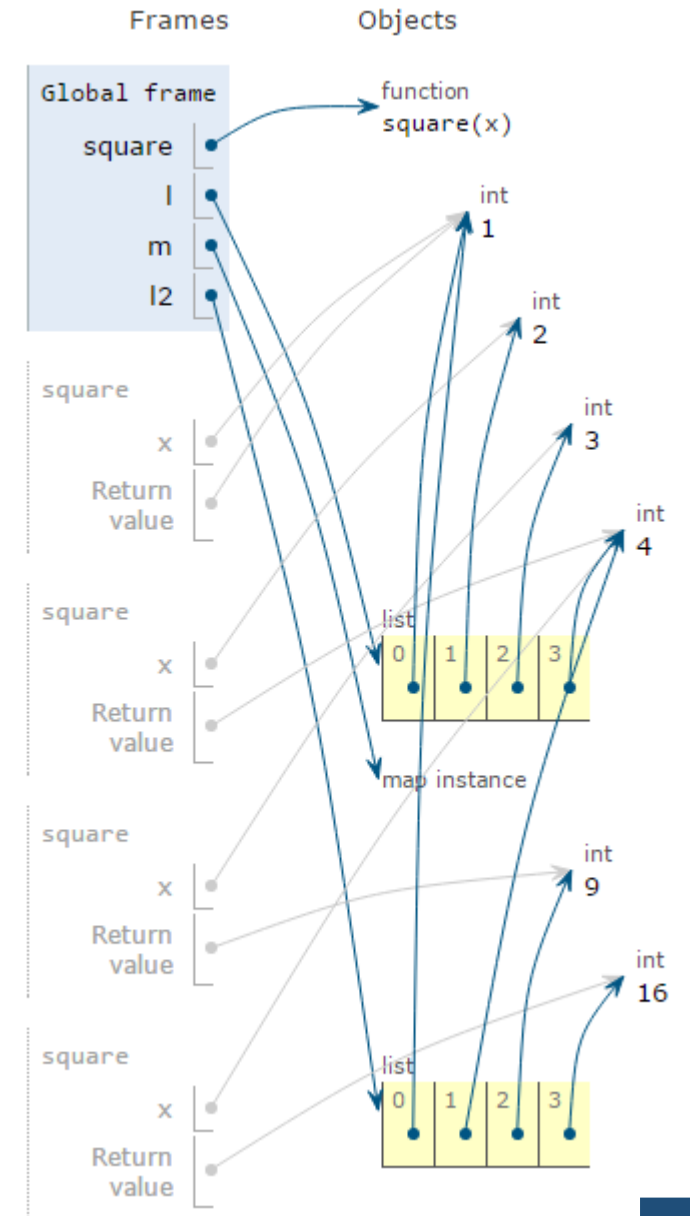
MAP

```
def square(x):  
    return x**2
```

```
l = [1, 2, 3, 4]  
m = map(square, l)  
l2 = list(m)
```

```
print(l2)
```

[1, 4, 9, 16]

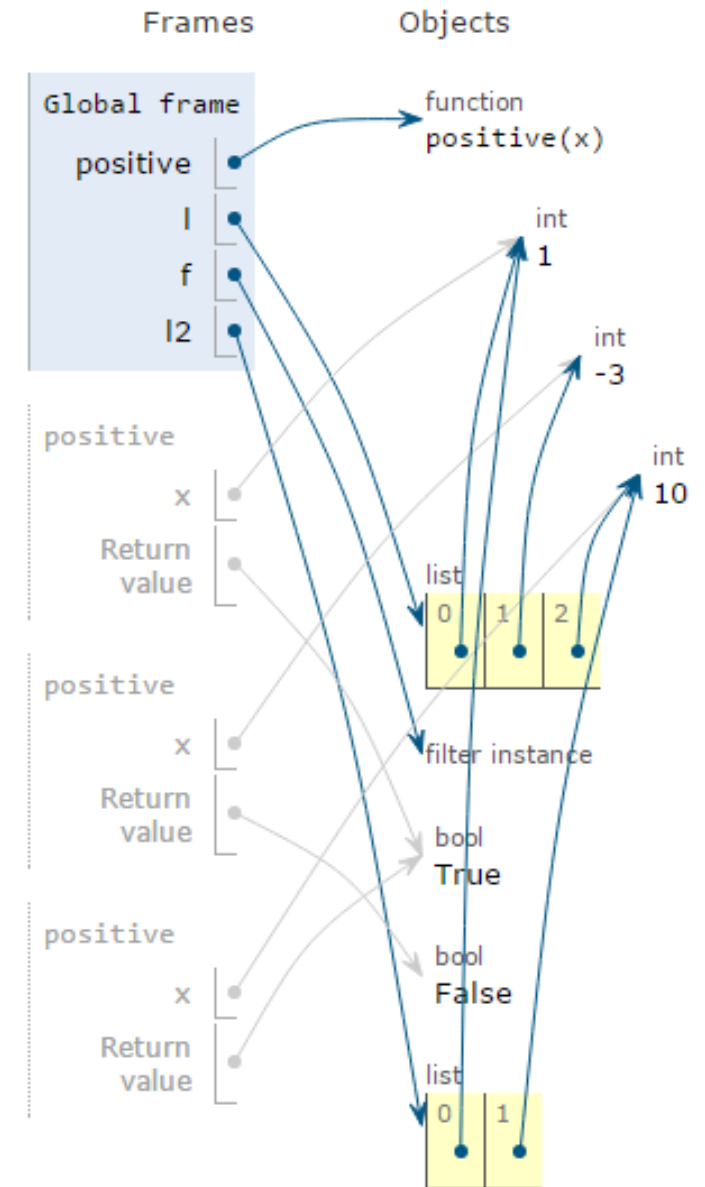


FILTER

```
def positive(x):  
    if(x > 0):  
        return True  
    return False
```

```
l = [1, -3, 10]  
f = filter(positive, l)  
l2 = list(f)  
print(l2)
```

[1, 10]



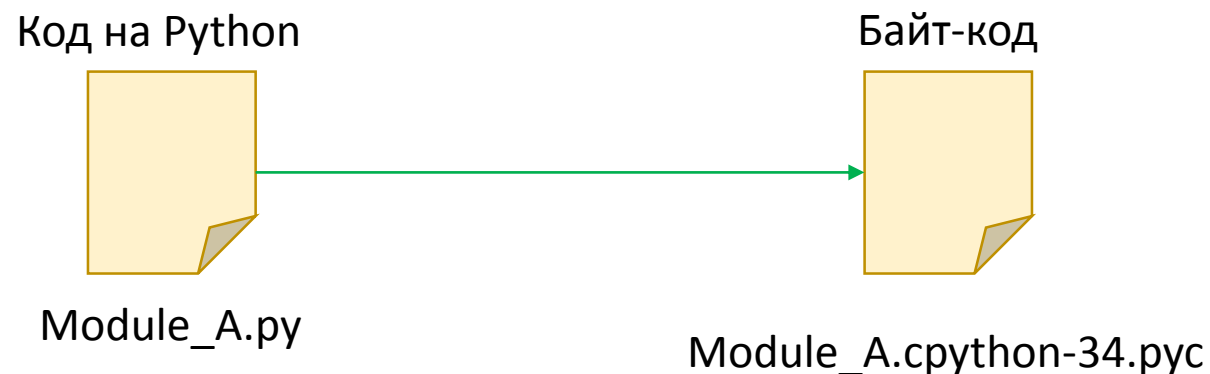
Модули и пакеты

Модули

Модули (Module) используются для облегчения разработки и сопровождения кода за счет его разделения на несколько отдельных частей с возможностью повторного использования.

Модуль (Module) – это файл с кодом на Python, имя модуля – имя файла без .py.

Для ускорения загрузки код модуля компилируется в байт-код



Подключение модулей

import *имя_модуля*

Module_A.py

```
def sum3Num(x, y, z):  
    return x+y+z
```

```
def mult3Num(x, y, z):  
    return x*y*z
```

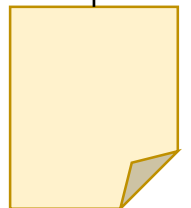
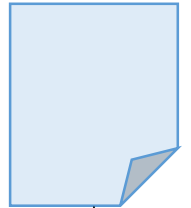
Main.py

```
import Module_A
```

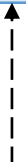
```
sum3 = Module_A.sum3Num(1, 2, 3)  
mult3 = Module_A.mult3Num(1, 2, 3)
```

```
print("1+2+3 = %i" % sum3)  
print("1*2*3 = %i" % mult3)
```

Main.py



Module_A.py



Подключение модулей

import *имя_модуля* as *имя*

Module_A.py

```
def sum3Num(x, y, z):  
    return x+y+z
```

```
def mult3Num(x, y, z):  
    return x*y*z
```

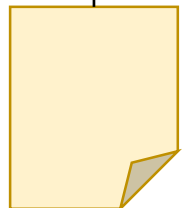
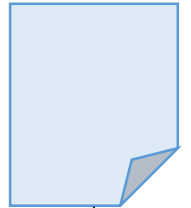
Main.py

```
import Module_A as mA
```

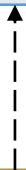
```
sum3 = mA.sum3Num(1, 2, 3)  
mult3 = mA.mult3Num(1, 2, 3)
```

```
print("1+2+3 = %i" % sum3)  
print("1*2*3 = %i" % mult3)
```

Main.py



Module_A.py



Подключение модулей

from *имя_модуля* import *имя_функции*

1+2+3 = 6

1*2*3 = 6

Module_A.py

```
def sum3Num(x, y, z):  
    return x+y+z
```

```
def mult3Num(x, y, z):  
    return x*y*z
```

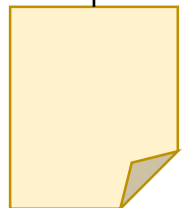
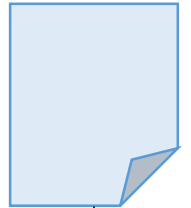
Main.py

```
from Module_A import sum3Num, mult3Num
```

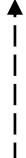
```
sum3 = sum3Num(1, 2, 3)  
mult3 = mult3Num(1, 2, 3)
```

```
print("1+2+3 = %i" % sum3)  
print("1*2*3 = %i" % mult3)
```

Main.py



Module_A.py



Подключение модулей

from *имя_модуля* import *

1+2+3 = 6

1*2*3 = 6

Module_A.py

```
def sum3Num(x, y, z):  
    return x+y+z
```

```
def mult3Num(x, y, z):  
    return x*y*z
```

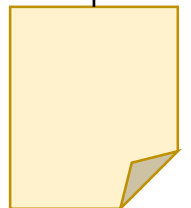
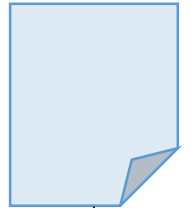
Main.py

```
from Module_A import *
```

```
sum3 = sum3Num(1, 2, 3)  
mult3 = mult3Num(1, 2, 3)
```

```
print("1+2+3 = %i" % sum3)  
print("1*2*3 = %i" % mult3)
```

Main.py

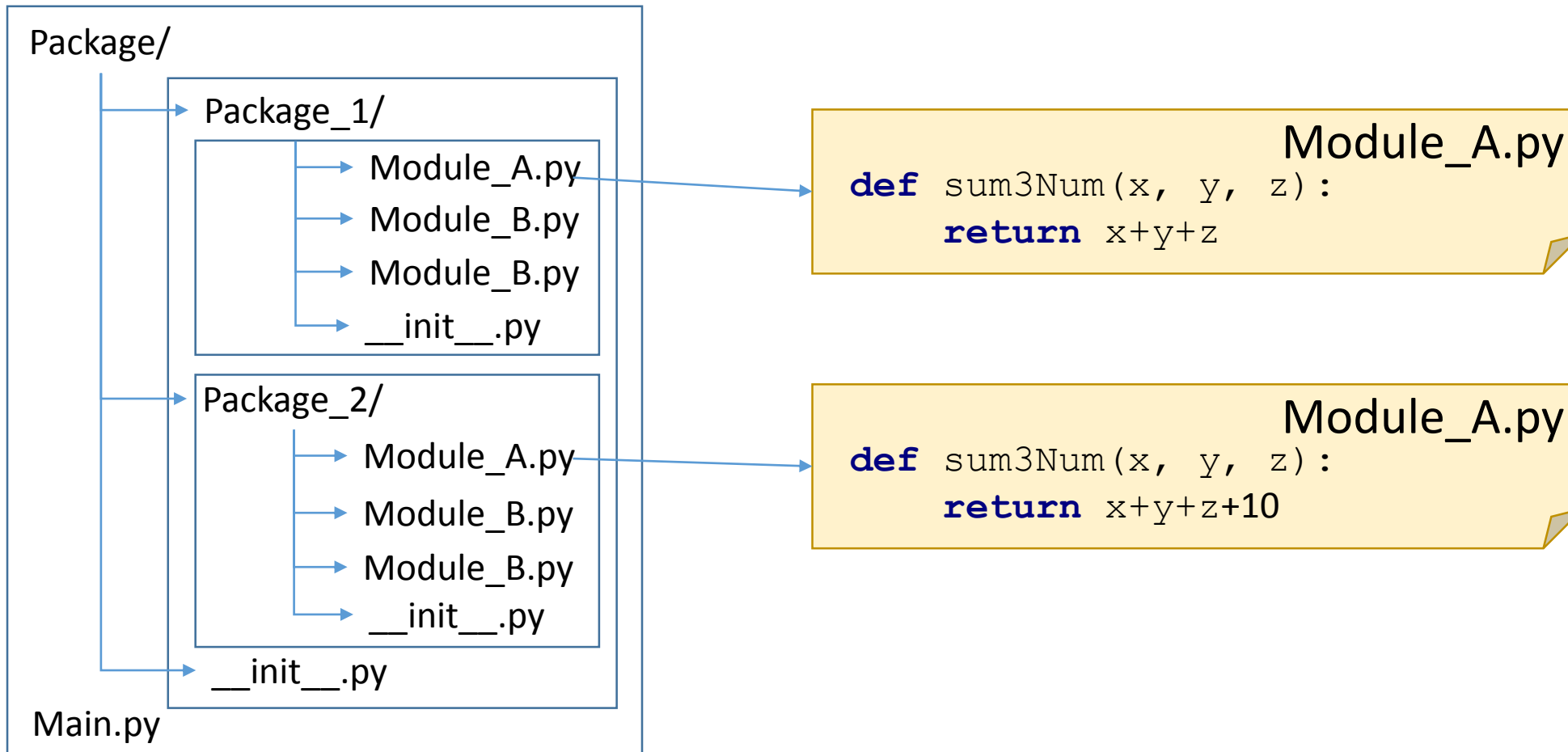


Module_A.py

Копируются все имена кроме тех, которые начинаются с нижнего подчеркивания (_)

Пакеты (Packages) используются для структурировать пространства имен модулей

Для пакетной организации создаются папки, в которых хранят модули



`import имя_пакета.имя_модуля`

`from имя_пакета import имя_модуля`

`from имя_пакета.имя_модуля import имя_функции`

Main.py

```
import Package.Package_1.Module_A as p1A
import Package.Package_2.Module_A as p2A
```

```
sum1A = p1A.sum3Num(1, 2, 3)
```

```
sum2A = p2A.sum3Num(1, 2, 3)
```

1. Программирование на Python. Часть 2: Строки в питоне

2. 6. Modules

3. More Control Flow Tools

4. Recursive Functions

5. Lambda, filter, reduce and map