Case Study: Python

Functions and Scope

Dr. Nguyen Hua Phung

HCMC University of Technology, Viet Nam

08, 2020

Functions

```
def <func-name>(<parameter-list>):
  <stmt-list>
```

- nested function
- pass-by-value (pointer)
- matched by position and by name
- default vaule
- arbitrary parameters
- arbitrary keyword parameters
- return statement

Nested Functions

```
def outer(x):
    y = x + 1
    def inner(z):
        return z + 1
    return inner(y)
print(outer(3))
print(inner(2))
```

Nested Functions

```
def outer(x):
    y = x + 1
    def inner(z):
        return z + 1
    return inner(y)
print(outer(3)) => 5
print(inner(2))
```

Nested Functions

```
def outer(x):
    y = x + 1
    def inner(z):
        return z + 1
    return inner(y)
print(outer(3))  => 5
print(inner(2))  => wrong
```

Nested Functions

```
def outer(x):
    y = x + 1
    def inner(z):
        return z + 1
    return inner(y)
print(outer(3))  => 5
print(inner(2))  => wrong
```

Nested Functions

```
def outer(x):
    y = x + 1
    def inner(z):
        return z + 1
    return inner(y)
print(outer(3))  => 5
print(inner(2))  => wrong
```

- **inner** function is visible inside **outer** but invisible outside **outer**

Match by position and by name

```
def foo(param1, param2 = 0):
    print(param1, param2)
print(foo(1,2))
print(foo(param2 = 2,param1 = 1))
print(foo(1))
```

Match by position and by name

```
def foo(param1, param2 = 0):
    print(param1, param2)
print(foo(1,2)) => 1 2
print(foo(param2 = 2,param1 = 1))
print(foo(1))
```

Match by position and by name

```
def foo(param1, param2 = 0):
    print(param1, param2)
print(foo(1,2)) => 1 2
print(foo(param2 = 2,param1 = 1)) => 1 2
print(foo(1))
```

Match by position and by name

```
def foo(param1, param2 = 0):
    print(param1, param2)
print(foo(1,2)) => 1 2
print(foo(param2 = 2,param1 = 1)) => 1 2
print(foo(1)) => 1 0
```

Arbitrary Arguments, *args

```
def my_func(*kids):
    print("My third child is" + kids[2])
my_func('Tuong', 'Ca', 'Mam', 'Muoi')
```

Arbitrary Arguments, *args

```
def my_func(*kids):
    print("My third child is" + kids[2])
my_func('Tuong', 'Ca', 'Mam', 'Muoi')
```

- Allow arbitrary number of arguments

Arbitrary Arguments, *args

```
def my_func(*kids):
    print("My third child is" + kids[2])
my_func('Tuong', 'Ca', 'Mam', 'Muoi')
```

- Allow arbitrary number of arguments
- Access the parameter as a *tuple*

Arbitrary Arguments, *args

```
def my_func(*kids):
    print("My third child is" + kids[2])
my_func('Tuong', 'Ca', 'Mam', 'Muoi')
```

- Allow arbitrary number of arguments
- Access the parameter as a *tuple*
- Define after normal parameters

Arbitrary keyword Arguments, **kwargs

```
def my_func(**rec):
    for x,y in rec.items():
        print(x,y)
my_func(ho='nguyen',ten='thi ha',
    namsinh=1996,mssv='0123456')
```

Arbitrary keyword Arguments, **kwargs

```
def my_func(**rec):
    for x,y in rec.items():
        print(x,y)
my_func(ho='nguyen',ten='thi ha',
    namsinh=1996,mssv='0123456')
```

- Allow arbitrary number of keyword arguments

Arbitrary keyword Arguments, **kwargs

```
def my_func(**rec):
    for x,y in rec.items():
        print(x,y)
my_func(ho='nguyen',ten='thi ha',
    namsinh=1996,mssv='0123456')
```

- Allow arbitrary number of keyword arguments
- Access the parameter as a *dictionary*

Arbitrary keyword Arguments, **kwargs

```
def my_func(**rec):
    for x,y in rec.items():
        print(x,y)
my_func(ho='nguyen',ten='thi ha',
    namsinh=1996,mssv='0123456')
```

- Allow arbitrary number of keyword arguments
- Access the parameter as a *dictionary*
- Define after normal and arbitrary parameters

Return Statement

- Syntax:

  return (<exp> (, <exp>)*)?

Return Statement

- Syntax:

  return (<exp> (, <exp>)*)?

- Example:

```
def my_func(x):
    x = 2
    return x,x+2
a,b = my_func()
print(a,b)
```

Return Statement

- Syntax:

  return (<exp> (, <exp>)*)?

- Example:

```
def my_func(x):
    x = 2
    return x,x+2
a,b = my_func()
print(a,b) =>2 4
```

Return Statement

- Syntax:

```
return (<exp> (, <exp>)*)?
```

- Example:

```
def my_func(x):
    x = 2
    return x,x+2
a,b = my_func()
print(a,b) =>2 4
```

- Stop executing of a function call and return the result

Return Statement

- Syntax:

  return (<exp> (, <exp>)*)?

- Example:

```
def my_func(x):
    x = 2
    return x,x+2
a,b = my_func()
print(a,b) =>2 4
```

- Stop executing of a function call and return the result
- If no expression after **return**, *None* is returned

Return Statement

- Syntax:

  return (<exp> (, <exp>)*)?

- Example:

```
def my_func(x):
    x = 2
    return x,x+2
a,b = my_func()
print(a,b) =>2 4
```

- Stop executing of a function call and return the result
- If no expression after **return**, *None* is returned
- If many expressions after **return**, a *tuple* is returned

Scope

- Read: Block rule, where a function is a block
  - ⇓ Local
  - ⇓ Nonlocal
  - ⇓ Global
    - Built-in or imported environments
- Write: **global**, **nonlocal**

Scope Example

```
from functools import *
x = 3
def f():
    y = 4
    def g():
        t = 2
        print(z)
```

Scope Example

```
from functools import * => imported env.
x = 3
def f():
    y = 4
    def g():
        t = 2
        print(z)
```

Scope Example

```
from functools import *   => imported env.
x = 3                     => global
def f():
    y = 4
    def g():
        t = 2
        print(z)
```

Scope Example

```
from functools import *
x = 3
def f():
    y = 4
    def g():
        t = 2
        print(z)
```

=> imported env.

=> global

=> nonlocal of g

Scope Example

```
from functools import *     => imported env.
x = 3                       => global
def f():
    y = 4                   => nonlocal of g
    def g():
        t = 2               => local of g
        print(z)
```

Scope Example

```
from functools import *     => imported env.
x = 3                       => global
def f():
    y = 4                   => nonlocal of g
    def g():
        t = 2               => local of g
        print(z)
```

Scope Example

```
from functools import *     => imported env.
x = 3                       => global
def f():
    y = 4                   => nonlocal of g
    def g():
        t = 2               => local of g
        print(z)
```

⇓ declaration of z is looked firstly in local environment

Scope Example

```
from functools import *     => imported env.
x = 3                       => global
def f():
    y = 4                   => nonlocal of g
    def g():
        t = 2               => local of g
        print(z)
```

- ⇓ declaration of z is looked firstly in local environment
- ⇓ and then in nonlocal environments that enclose the local

Scope Example

```
from functools import *   => imported env.
x = 3                     => global
def f():
    y = 4                 => nonlocal of g
    def g():
        t = 2             => local of g
        print(z)
```

- ⇓ declaration of z is looked firstly in local environment
- ⇓ and then in nonlocal environments that enclose the local
- ⇓ and then in global environment

Scope Example

```
from functools import *   => imported env.
x = 3                     => global
def f():
    y = 4                 => nonlocal of g
    def g():
        t = 2             => local of g
        print(z)
```

- ⇓ declaration of z is looked firstly in local environment
- ⇓ and then in nonlocal environments that enclose the local
- ⇓ and then in global environment
- and lastly in imported environments

global Example

```
1  x,y = 3,4
2  def f():
3      x = 2
4      return x + y
5  def g():
6      global x
7      x = 2
8      return 2 + x
9  f()
10 print(x)
11 g()
12 print(x)
```

global Example

```
1  x,y = 3,4
2  def f():
3      x = 2
4      return x + y
5  def g():
6      global x
7      x = 2
8      return 2 + x
9  f()
10 print(x)
11 g()
12 print(x)
```

=> x,y: global

global Example

```
1  x,y = 3,4
2  def f():
3      x = 2
4      return x + y
5  def g():
6      global x
7      x = 2
8      return 2 + x
9  f()
10 print(x)
11 g()
12 print(x)
```

=> x,y: global

=> x: local of f

global Example

```
1  x,y = 3,4                    => x,y: global
2  def f():
3      x = 2                    => x: local of f
4      return x + y             => x: local of f; y global
5  def g():
6      global x
7      x = 2
8      return 2 + x
9  f()
10 print(x)
11 g()
12 print(x)
```

global Example

```
1  x,y = 3,4                    => x,y: global
2  def f():
3      x = 2                    => x: local of f
4      return x + y             => x: local of f; y global
5  def g():
6      global x
7      x = 2                    => x: global
8      return 2 + x
9  f()
10 print(x)
11 g()
12 print(x)
```

global Example

```
1  x,y = 3,4                    => x,y: global
2  def f():
3      x = 2                    => x: local of f
4      return x + y             => x: local of f; y global
5  def g():
6      global x
7      x = 2                    => x: global
8      return 2 + x
9  f()
10 print(x)                     => 3
11 g()
12 print(x)
```

global Example

```
1  x,y = 3,4                    => x,y: global
2  def f():
3      x = 2                    => x: local of f
4      return x + y             => x: local of f; y global
5  def g():
6      global x
7      x = 2                    => x: global
8      return 2 + x
9  f()
10 print(x)                     => 3
11 g()
12 print(x)                     => 2
```

global Example

```
1  x,y = 3,4                    => x,y: global
2  def f():
3      x = 2                    => x: local of f
4      return x + y             => x: local of f; y global
5  def g():
6      global x
7      x = 2                    => x: global
8      return 2 + x
9  f()
10 print(x)                     => 3
11 g()
12 print(x)                     => 2
```

global Example

```
1  x,y = 3,4                    => x,y: global
2  def f():
3      x = 2                    => x: local of f
4      return x + y             => x: local of f; y global
5  def g():
6      global x
7      x = 2                    => x: global
8      return 2 + x
9  f()
10 print(x)                     => 3
11 g()
12 print(x)                     => 2
```

- firstly assigning to a variable makes the declaration of the variable in the current environment

global Example

```
1  x,y = 3,4                    => x,y: global
2  def f():
3      x = 2                    => x: local of f
4      return x + y             => x: local of f; y global
5  def g():
6      global x
7      x = 2                    => x: global
8      return 2 + x
9  f()
10 print(x)                     => 3
11 g()
12 print(x)                     => 2
```

- firstly assigning to a variable makes the declaration of the variable in the current environment
- to assign to a global variable in a function, the declaration of **global** is required

nonlocal Example

```
1  x,y = 3,4
2  def f():
3      x,z = 2,5
4      def g():
5          nonlocal x
6          x = 2 * y
7          return z + x
8      print(g())
9      print(x)
10 f()
11 print(x)
```

- like **global** but for **nonlocal** variables

nonlocal Example

```
1  x,y = 3,4
2  def f():
3      x,z = 2,5
4      def g():
5          nonlocal x
6          x = 2 * y
7          return z + x
8      print(g())
9      print(x)
10 f()
11 print(x)
```

=> x,y: global

- like **global** but for **nonlocal** variables

nonlocal Example

```
1  x,y = 3,4                    => x,y: global
2  def f():
3      x,z = 2,5                => x,z: local of f
4      def g():
5          nonlocal x
6          x = 2 * y
7          return z + x
8      print(g())
9      print(x)
10 f()
11 print(x)
```

- like **global** but for **nonlocal** variables

nonlocal Example

```
1  x,y = 3,4                    => x,y: global
2  def f():
3      x,z = 2,5                => x,z: local of f
4      def g():
5          nonlocal x
6          x = 2 * y            => x: nonlocal
7          return z + x
8      print(g())
9      print(x)
10 f()
11 print(x)
```

- like **global** but for **nonlocal** variables

nonlocal Example

```
1  x,y = 3,4                    => x,y: global
2  def f():
3      x,z = 2,5                => x,z: local of f
4      def g():
5          nonlocal x
6          x = 2 * y            => x: nonlocal=>local of f;
7          return z + x
8      print(g())
9      print(x)
10 f()
11 print(x)
```

- like **global** but for **nonlocal** variables

nonlocal Example

```
1  x,y = 3,4                          => x,y: global
2  def f():
3      x,z = 2,5                      => x,z: local of f
4      def g():
5          nonlocal x
6          x = 2 * y                  => x: nonlocal=>local of f;y:global
7          return z + x
8      print(g())
9      print(x)
10 f()
11 print(x)
```

- like **global** but for **nonlocal** variables

nonlocal Example

```
1  x,y = 3,4                    => x,y: global
2  def f():
3      x,z = 2,5                => x,z: local of f
4      def g():
5          nonlocal x
6          x = 2 * y            => x: nonlocal=>local of f;y:global
7          return z + x         => x,z:nonlocal
8      print(g())
9      print(x)
10 f()
11 print(x)
```

- like **global** but for **nonlocal** variables

nonlocal Example

```
1  x,y = 3,4                    => x,y: global
2  def f():
3      x,z = 2,5                => x,z: local of f
4      def g():
5          nonlocal x
6          x = 2 * y            => x: nonlocal=>local of f;y:global
7          return z + x         => x,z:nonlocal=>local of f;
8      print(g())
9      print(x)
10 f()
11 print(x)
```

- like **global** but for **nonlocal** variables

nonlocal Example

```
1  x,y = 3,4                    => x,y: global
2  def f():
3      x,z = 2,5                => x,z: local of f
4      def g():
5          nonlocal x
6          x = 2 * y            => x: nonlocal=>local of f;y:global
7          return z + x         => x,z:nonlocal=>local of f;
8      print(g())               => 13
9      print(x)
10 f()
11 print(x)
```

- like **global** but for **nonlocal** variables

nonlocal Example

```
1  x,y = 3,4                    => x,y: global
2  def f():
3      x,z = 2,5                => x,z: local of f
4      def g():
5          nonlocal x
6          x = 2 * y            => x: nonlocal=>local of f;y:global
7          return z + x         => x,z:nonlocal=>local of f;
8      print(g())               => 13
9      print(x)                 => 8
10 f()
11 print(x)
```

- like **global** but for **nonlocal** variables

nonlocal Example

```
1  x,y = 3,4                    => x,y: global
2  def f():
3      x,z = 2,5                => x,z: local of f
4      def g():
5          nonlocal x
6          x = 2 * y            => x: nonlocal=>local of f;y:global
7          return z + x         => x,z:nonlocal=>local of f;
8      print(g())               => 13
9      print(x)                 => 8
10 f()
11 print(x)                     => 3
```

- like **global** but for **nonlocal** variables

References I

[1] Python Tutorial, http:w3schools.com/python, 10 08 2020.

[2] Python Programming Language, https://www.geeksforgeeks.org/python-programming-language/, 10 08 2020.

[3] Python Tutorial, https://www.tutorialspoint.com/python, 10 08 2020.

[4] Introduction to Python 3, https://realpython.com/python-introduction/, 10 08 2020.