06-Python函数

- 定义函数
- 传递实参
- 返回值
- 传递列表参数
- 传递任意数量的参数
- 定义模块
- 函数编写指南

定义函数

- def表示要定义一个函数
- greet_user()是定义的函数名，必须要有圆括号
- 函数名后面的后面是冒号，表示函数代码块开始
- 函数体的代码块必须有缩进
- 按照代码规范，第一句是函数的文档注释(docstring)

In [ ]:
```
def greet_user():
    """Display a simple greeting."""
    print("Hello!")
```

In [ ]:
```
greet_user()
```

形参和实参

- 变量username是一个形参(parameter)。
- 值'jesse'是一个实参（argument）。
- 函数在被调用时，实参的值会被赋值给形参。

In [ ]:
```
def greet_user(username):
    """Display a simple greeting."""
    print(f"Hello, {username.title()}!")

greet_user('jesse')
```

形参和实参的值如果发生变化，会相互影响吗?

In [4]:
```
def plus(x, y):
    print("Print ID in the function before assignment.")
    print(id(x), id(y), "\n")
    x = 20
    y = 10
    print("Print ID in the function after assignment.")
    print(id(x), id(y))
    print(f"x + y = {x + y}")

a, b = 1, 2
print("Print ID out of the function.")
print(id(a), id(b))
print(f"a={a}, b={b}\n")
plus(a, b)
```

```
Print ID out of the function.
1226196123888 1226196123920
a=1, b=2

Print ID in the function before assignment.
1226196123888 1226196123920

Print ID in the function after assignment.
1226196124496 1226196124176
a + b = 30
```

两种参数类型:

1. 位置实参
2. 关键字实参

位置实参

当函数有多个参数时，基于实参的顺序关联到对应位置的形参，这种关联方式称为位置实参。

In [5]:
```
def describe_pet(animal_type, pet_name):
    """Display information about a pet."""
    print(f"\nI have a {animal_type}.")
    print(f"My {animal_type}'s name is {pet_name.title()}.")

describe_pet('hamster', 'harry')
describe_pet('dog', 'willie')
describe_pet('harry', 'hamster')
```

```
I have a hamster.
My hamster's name is Harry.

I have a dog.
My dog's name is Willie.

I have a harry.
My harry's name is Hamster.
```

关键字实参

- 关键字实参是传递给函数的名称-值对。直接在实参中将名称和值关联起来，因此向函数传递实参时不会混淆。
- 关键字实参顺序不再重要。
- 代码可读性更好。

In [7]:
```
def describe_pet(animal_type, pet_name):
    """Display information about a pet."""
    print(f"\nI have a {animal_type}.")
    print(f"My {animal_type}'s name is {pet_name.title()}.")

describe_pet(animal_type="hamster", pet_name="harry")
describe_pet(pet_name="harry", animal_type="hamster")
```

```
I have a hamster.
My hamster's name is Harry.

I have a hamster.
My hamster's name is Harry.
```

参数默认值

- 编写函数时，可给每个形参指定默认值。
- 在调用函数中给形参提供了实参时，Python将使用指定的实参值；否则，将使用形参的默认值。
- 形参列表中的带有默认值的形参必须放在没有默认值的形参后面。

In [8]:
```
def describe_pet(pet_name, animal_type='dog'):
    """Display information about a pet."""
    print(f"\nI have a {animal_type}.")
    print(f"My {animal_type}'s name is {pet_name.title()}.")

describe_pet(pet_name='willie')
```

```
I have a dog.
My dog's name is Willie.
```

In [9]:
```
def describe_pet(animal_type='dog', pet_name, ):
    """Display information about a pet."""
    print(f"\nI have a {animal_type}.")
    print(f"My {animal_type}'s name is {pet_name.title()}.")
```

```
  Input In [9]
    def describe_pet(animal_type='dog', pet_name, ):
                                        ^

SyntaxError: non-default argument follows default argument
```

函数返回值

当函数没有return语句时，函数默认返回None。

In [10]:
```
def get_formatted_name(first_name, last_name):
    """Return a full name, neatly formatted."""
    full_name = f"{first_name} {last_name}"
    return full_name.title()

musician = get_formatted_name('jimi', 'hendrix')
print(musician)
```

```
Jimi Hendrix
```

In [11]:
```
def get_formatted_name(first_name, middle_name, last_name):
    """Return a full name, neatly formatted."""
    full_name = f"{first_name} {middle_name} {last_name}"
    return full_name.title()

musician = get_formatted_name('john', 'lee', 'hooker')
print(musician)
```

```
John Lee Hooker
```

In [ ]:
```
get_formatted_name('jimi', 'hendrix')
```

In [14]:
```
def get_formatted_name(first_name, last_name, middle_name=''):
    """Return a full name, neatly formatted."""
    if middle_name:
        full_name = f"{first_name} {middle_name} {last_name}"
    else:
        full_name = f"{first_name} {last_name}"

    return full_name.title()

print(get_formatted_name('jimi', 'hendrix'))
print(get_formatted_name('john', 'hooker', 'lee'))
```

```
Jimi Hendrix
John Lee Hooker
```

函数返回字典

In [ ]:
```
def build_person(first_name, last_name, age=None):
    """Return a dictionary of information about a person."""
    person = {'first': first_name, 'last': last_name}
    if age:
        person['age'] = age
    return person
```

函数返回多个值：返回的是一个元组

In [16]:
```python
def get_first_last_name(fullname):
    """Return a full name, neatly formatted."""
    names = fullname.split()
    first_name = names[0]
    last_name = names[-1]
    return first_name, last_name # return a tuple here

first, last = get_first_last_name('John Lee Hooker')
print(f'First name: {first}, Last name: {last}')
```

```
First name: John, Last name: Hooker
```

传递列表参数

In [14]:
```python
def greet_users(names):
    """Print a simple greeting to each user in the list."""
    for name in names:
        msg = f"Hello, {name.title()}!"
        print(msg)

usernames = ['hannah', 'ty', 'margot']
greet_users(usernames)
greet_users(usernames[:]) # 复制列表传入函数，防止列表内容被修改
```

```
Hello, Hannah!
Hello, Ty!
Hello, Margot!
```

在函数中修改列表

In [15]:
```
def print_models(unprinted_designs, completed_models):
    """
    Simulate printing each design, until none are left.
    Move each design to completed_models after printing.
    """
    while unprinted_designs:
        current_design = unprinted_designs.pop()
        print(f"Printing model: {current_design}")
        completed_models.append(current_design)

def show_completed_models(completed_models):
    """Show all the models that were printed."""
    print("\nThe following models have been printed:")
    for completed_model in completed_models:
        print(completed_model)

unprinted_designs = ['iphone case', 'robot pendant', 'dodecahedron']
completed_models = []
print_models(unprinted_designs, completed_models)
show_completed_models(completed_models)
```

```
Printing model: dodecahedron
Printing model: robot pendant
Printing model: iphone case

The following models have been printed:
dodecahedron
robot pendant
iphone case
```

传递任意数量的位置实参： *

In [16]:
```
def make_pizza(*toppings):
    """print all the toppings that have been requested."""
    print(toppings)

make_pizza('pepperoni')
make_pizza('mushrooms', 'green peppers', 'extra cheese')
```

```
('pepperoni',)
('mushrooms', 'green peppers', 'extra cheese')
```

任意数量的位置实参必须放到普通位置实参后面

In [24]:
```
def make_pizza(size, *toppings):
    """Describe the pizza"""
    print(f"\nMaking a {size}-inch pizza with the following toppings:")
    for topping in toppings:
        print(f"- {topping}")

make_pizza(16, "pepperoni")
make_pizza(12, "mushroom", "green peppers", "extra cheese")
make_pizza(12, "mushroom", "green peppers", "extra cheese")
```

```
Making a 16-inch pizza with the following toppings:
- pepperoni

Making a 12-inch pizza with the following toppings:
- mushroom
- green peppers
- extra cheese

Making a 12-inch pizza with the following toppings:
- mushroom
- green peppers
- extra cheese
k, v = price, 16
```

* 操作符表示的是分解

In [20]:
```
numbers = [1, 2, 3, 4, 5]
print(*numbers)
msg = "hello"
print(*msg)
```

```
1 2 3 4 5
h e l l o
```

In [22]:
```
a, b, *rest = numbers
print(a, b)
print(*rest)
print(rest)
a, _, b, _, *rest = numbers
```

```
1 3
5
[5]
```

传递任意数量的关键字实参： **

你经常会看到形参名 **kwargs ， 它让函数接受任意数量的关键字实参。

In [5]:
```
def build_profile(first, last, **user_info):
    """Build a dictionary containing everything we know about a user."""
    user_info['first_name'] = first
    user_info['last_name'] = last
    return user_info

user_profile = build_profile('albert', 'einstein',
                             location='princeton',
                             field='physics')
print(user_profile)
```

```
{'location': 'princeton', 'field': 'physics', 'first_name': 'albert', 'last_name': 'einste
in'}
```

** 操作符表示的是分解字典

In [9]:
```
def dump(**kwargs):
    return kwargs
```

In [10]:
```
dump(**{'a': 1}, b=2, **{'c':3, 'd':4})
```

Out[10]:
```
{'a': 1, 'b': 2, 'c': 3, 'd': 4}
```

将函数保存在模块中

在Python中，一个.py文件就是一个模块。

In [11]:
```
import pizza

pizza.make_pizza(16, 'pepperoni')
pizza.deliver_pizza("hotel")
```

```
Making a 16-inch pizza with the following toppings:
- pepperoni

Delivering the pizza to hotel.
```

In [12]:
```
from pizza import make_pizza, deliver_pizza
make_pizza(16, 'pepperoni')
deliver_pizza("hotel")
```

```
Making a 16-inch pizza with the following toppings:
- pepperoni

Delivering the pizza to hotel.
```

In [13]:
```
from pizza import make_pizza as mp, deliver_pizza as dp
mp(16, 'pepperoni')
dp("hotel")
```

```
Making a 16-inch pizza with the following toppings:
- pepperoni

Delivering the pizza to hotel.
```

In [14]:
```
import pizza as p
p.make_pizza(16, 'pepperoni')
p.deliver_pizza("hotel")
```

```
Making a 16-inch pizza with the following toppings:
- pepperoni

Delivering the pizza to hotel.
```

使用*可以导入模块中所有的函数，但是不推荐这样使用

In [15]:
```
from pizza import *
make_pizza(16, 'pepperoni')
deliver_pizza("hotel")
```

```
Making a 16-inch pizza with the following toppings:
- pepperoni

Delivering the pizza to hotel.
```

用于序列的内置函数

- enumerate
- zip

我们经常在循环的时候需要用到序列的索引

In [ ]:
```
def createDict(keys, values):
    index = 0
    result = {}
    for key in keys:
        if index < len(values):
            result[key] = values[index]
        else:
            result[key] = None
        index += 1
    return result
```

这时候我们可以用内置函数enumerate,这样就不需要自己定义一个index来记录索引了

In [1]:
```
def createDict(keys, values):
    result = {}
    for index, key in enumerate(keys):
        if index < len(values):
            result[key] = values[index]
        else:
            result[key] = None
    return result

keys = ['a', 'b', 'c', 'd']
values = [1, 2, 3]
createDict(keys, values) # {'a': 1, 'b': 2, 'c': 3, 'd': None}
```

Out[1]:
```
{'a': 1, 'b': 2, 'c': 3, 'd': None}
```

zip函数：将多个序列压缩成一个序列

In [4]:
```
numbers = [1, 2, 3, 4, 5]
letters = ['a', 'b', 'c', 'd', 'e']
zipped = zip(numbers, letters)
print(list(zipped))  # zipped 只能生产一次数据
print(list(zipped))  # zipped is now empty
```

```
[(1, 'a'), (2, 'b'), (3, 'c'), (4, 'd'), (5, 'e')]
[]
```

In [3]:
```
# 很容易将zip对象转换为字典
print(dict(zip(numbers, letters)))
```

```
{1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'e'}
```

zip函数可以接受任意多个序列，然后返回一个元组(tuple)的列表

In [5]:
```
numbers = [1, 2, 3, 4, 5]
lowers = ['a', 'b', 'c', 'd', 'e']
uppers = ['A', 'B', 'C', 'D', 'E']
zipped = zip(numbers, lowers, uppers)
print(list(zipped))
```

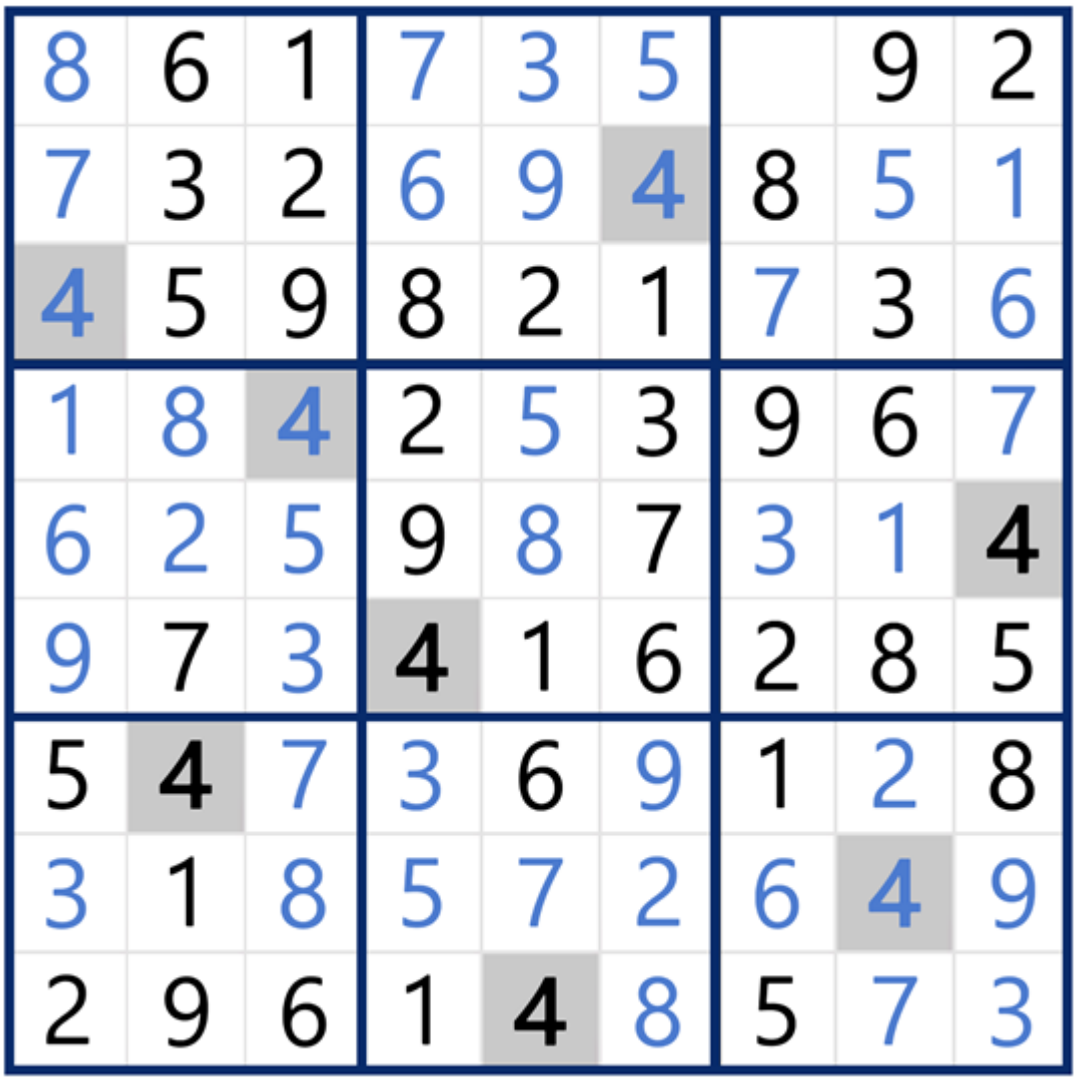
```
[(1, 'a', 'A'), (2, 'b', 'B'), (3, 'c', 'C'), (4, 'd', 'D'), (5, 'e', 'E')]
```

矩阵转置

In [7]:
```
matrix = [[1, 2, 3],
          [4, 5, 6],
          [7, 8, 9]]
print(*matrix)
print(list(zip(*matrix)))
```

```
[1, 2, 3] [4, 5, 6] [7, 8, 9]
[(1, 4, 7), (2, 5, 8), (3, 6, 9)]
```

判断数独是否合法



8	6	1	7	3	5		9	2
7	3	2	6	9	4	8	5	1
4	5	9	8	2	1	7	3	6
1	8	4	2	5	3	9	6	7
6	2	5	9	8	7	3	1	4
9	7	3	4	1	6	2	8	5
5	4	7	3	6	9	1	2	8
3	1	8	5	7	2	6	4	9
2	9	6	1	4	8	5	7	3

In [22]:
```
from itertools import product
list(product(range(3), range(3)))
```

Out[22]:
```
[(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2)]
```

In [23]:
```
print([(x, y) for x in range(0,7,3) for y in range(0,7,3)])
```

```
[(0, 0), (0, 3), (0, 6), (3, 0), (3, 3), (3, 6), (6, 0), (6, 3), (6, 6)]
```

In [12]:
```
from itertools import product

def validate_sudoku(board):
    """Sudoka solution validator"""
    # Check each row
    for row in range(9):
        if set(board[row]) != set(range(1, 10)):
            return False

    # Check each column
    for column in zip(*board):
        if set(column) != set(range(1, 10)):
            return False

    # Check each 3x3 sub-grid
    for i, j in product(range(3), range(3)):
        sub_grid = []
        for line in board[i*3:i*3+3]:
            sub_grid += line[j*3:j*3+3]
        if set(sub_grid) != set(range(1, 10)):
            return False

    # check each sub-grid
    # for i,j in [(0,0), (0,3), (0,6), (3,0), (3,3), (3,6), (6,0), (6,3), (6,6)]:
    for i,j in [(x, y) for x in range(0,7,3) for y in range(0,7,3)]:
        sub_grid = [board[x][y] for x in range(i, i+3) for y in range(j, j+3)]
        if set(sub_grid) != set(range(1, 10)):
            return False

    # magic squares
    for i in range(3, 10, 3):
        for j in range(3, 10, 3):
            if set(range(1, 10)) != {(board[q][w]) for w in range(j-3, j) for q in range(i-
                return False

    # if everything is ok
    return True
```

In [13]:
```
board1 =[[8,4,7,2,6,5,1,9,3],
         [1,3,6,7,9,8,2,4,5],
         [9,5,2,1,4,3,8,6,7],
         [4,2,9,6,7,1,5,3,8],
         [6,7,8,5,3,2,9,1,4],
         [3,1,5,4,8,9,7,2,6],
         [5,6,4,9,1,7,3,8,2],
         [7,8,1,3,2,4,6,5,9],
         [2,9,3,8,5,6,4,7,1]]  # True
print(validate_sudoku(board1))

board2 =[[8,4,7,2,6,5,1,0,3],
         [1,3,6,7,0,8,2,4,5],
         [0,5,2,1,4,3,8,6,7],
         [4,2,0,6,7,1,5,3,8],
         [6,7,8,5,3,2,0,1,4],
         [3,1,5,4,8,0,7,2,6],
         [5,6,4,0,1,7,3,8,2],
         [7,8,1,3,2,4,6,5,0],
         [2,0,3,8,5,6,4,7,1]] # a valid board, but with 0 instead of 9
print(validate_sudoku(board2))
```

```
True
False
```