Python函数

内置函数

1. 常用内置函数

```
a = [1, -1, 0, 5]
# 求最大
print(max(a))
# 求最小
print(min(a))
```

```
5
-1
```

2. 类型转换

```
# # 字符串转整型
# print(int('520'))
# # 浮点转整型
# print(int(3.94))

# # # 字符串转浮点型
# print(float('3.14'))

# # # 浮点型转字符
# print(str(1.23))
# # # 整型转字符串
# print(str(100))

# 整型转布尔型
# print(bool(-1))
# print(bool(0))
# print(bool(' '))
# print(bool(''))
print(bool([None]))
print(bool([]))
```

```
True
False
```

定义函数

接口与函数

- 定义函数时，确定了参数名称和位置，就完成接口函数的定义
- 调用者不用关心函数内部逻辑，只需要指定如何传参，返回值是什么，就可以了

定义函数

- 定义函数要素：def，函数名，括号，参数，冒号，函数体，return返回内容
- 执行函数要素：函数名，括号，参数

```
# def xx(a):
#     print('hello', a)
# a = xx(1222)
# print(a, type(a))
# print(a)
def my_abs(x=1):
    '''计算绝对值'''
    if not isinstance(x, int):
        return None
    if x >= 0:
        return x
    else:
        return -x
# 不使用括号，直接得到函数对象，而不会执行函数
# my_abs()
# print(my_abs(-10))
# # # 自动抛出异常
print(my_abs('a'))
```

```
None
```

- 参数检查，程序自动报错不准确，可以手动抛出异常

```
# isinstance('a', (float, int))


def my_abs(x):
    # 判断变量类型后，手动抛出异常
    if not isinstance(x, (int, float)):
        # 抛出异常
        raise TypeError('您传入的参数不是浮点，或者整型')
    if x >= 0:
        return x
    else:
        return -x
print(my_abs('d'))
```

- 是返回多个值，例如：返回屏幕的坐标

```
# 导入math模块，以使用sin/cos函数
import math
# 接受(x, y)起始坐标，step位移，angle角度(默认为0)
def move(x, y, step, angle=0):
    nx = x + step * math.cos(angle)
    ny = y - step * math.sin(angle)
    return nx, ny
# 分开获取返回值
x, y = move(0, 0, 10, math.pi / 6)
print(x, y)
# 获取返回元组
r = move(0, 0, 10, math.pi / 6)
print(r)
print(r[0], r[1])
```

```
8.660254037844387 -4.999999999999999
(8.660254037844387, -4.999999999999999)
8.660254037844387 -4.999999999999999
```

- 遇到return就会返回结果，而不会执行后面的内容

```
# return的位置，一定要准确
# def test(x):
#     if x >= 5:
#         print("hello")
#         return '大于5'
#     if x >= 10:
#         print("world")
#         return '大于10'
# print(test(11))

def test_1():
    a = 10
    b = 20
    c = a + b
    return c
    print(c)

h1 = test_1()
print(h1)
```

```
30
```

- 小结：

1. 定义函数时，需要先确定函数名和参数个数
2. 如有必要，先对参数的数据类型做校验
3. 函数内部遇到return则会返回，而不会执行后面的结果
4. 如果函数没有return语句，则默认return None
5. 函数可以同时返回多个值，但本质上是一个元组

函数的参数

- 定义参数类型类型：必选参数，默认参数，可变参数，关键字参数
- 目的：简化代码，方便调用

默认参数

- 默认参数可以降低调用的难度，即：少传或不传参数

```
# 固定参数，位置参数
def enroll(name, gender):
    print('name:', name)
    print('gender:', gender)

# enroll('duoceshi', 'M')
# enroll('duoceshi')
```

```
# 默认参数
def enroll(name, gender, age=6, city='Shenzhen'):
    print('name:', name)
    print('gender:', gender)
    print('age:', age)
    print('city:', city)

# enroll()
# 默认参数可不传
# enroll('duoceshi', 'M')
# 默认参数可顺序传
# enroll('duoceshi', 'M', 4, 'guangzhou')
# 指定某个默认参数
# enroll('duoceshi', 'M', 'guangzhou')  # 位置不对
enroll('duoceshi', 'M', city='guangzhou')
```

```
name: duoceshi
gender: M
age: 6
city: guangzhou
```

- 默认参数必须指向不变对象：字符串，整型，浮点型，布尔型，空类型等
    1. 原因：默认参数变量L指向列表，而列表是可变长的，每次程序调用的时候，都会去取默认列表，但每次执行完毕都会改变这个默认列表，导致实际使用的是更改后的列表，而不是最初

的空列表[]
2. 结论：尽量使用不变对象当默认参数，因为不变对象一旦创建，对象内部的数据就不能修改，减少错误

```
def add_end(L=[]):
    L.append('duoceshi')
    return L
print(add_end())
# # 每次调用都会累计之前的变量
print(add_end())
# print(add_end())
```

```
['duoceshi']
['duoceshi', 'duoceshi']
```

```
# 改良
def add_end(a, L=None):
    if L is None:
        L = []
    L.append('duoceshi')
    return L

print(add_end(1))
print(add_end(1))
```

```
['duoceshi']
['duoceshi']
```

- 默认参数使用说明：
    1. 必选参数在前，默认参数在后，否则Python的解释器会报错
    2. 变化大的参数放前面，变化小的参数放后面，变化小的作为默认参数，username/passwd

可变参数

- 参数的个数不确定的时候，考虑把列表传进去
- 可变长可为0或多个

```python
# 加个*号就变可变长
def req_h(*user):
    print("user is", user)
    for i in user:
        print(i)

a = ['duoceshi', 'Tom', 'Jerry']
# 错误调用，输出整个列表
req_h(a)
# # 正确调用1，一个一个传
# req_h(a[0], a[1], a[2], "ahasdf", 'asdf ')
# 正确调用2，使用*号，后面还可以添加
req_h(*a)
```

```
user is (['duoceshi', 'Tom', 'Jerry'],)
['duoceshi', 'Tom', 'Jerry']
user is ('duoceshi', 'Tom', 'Jerry')
duoceshi
Tom
Jerry
```

关键字参数

使用场景

- 使用场景：注册的可选项，比如：地址，邮编；构造测试数据的时候，需要指定多个扩展参数
- 关键字参数可传入0个或多个键值对

```python
# 两个**定义关键字参数
def person(name, age, **kw):
    print('name:', name, 'age:', age, 'other:', kw)

extra = {'city': 'Shenzhen', 'job': 'Engineer', "home": "hunan"}
# 错误调用
# person('duoceshi', 4, extra)
# 正确调用，关键字参数得到的是一个字典
person('duoceshi', 4, city=extra['city'], job=extra['job'])
# 正确调用
person('duoceshi', 4, **extra)
```

```
name: duoceshi age: 4 other: {'city': 'Shenzhen', 'job': 'Engineer', 'home':
'hunan'}
```

命名关键字参数

- 为了防止调用者传入不受限制的参数，命名关键字参数

```
# 指定只接收特定的关键字参数，使用*分隔符
def person(name, age, *, city="guangzhou", job):
    print(name, age, city, job)
extra = {'job': 'Engineer'}
# 提示不接受该参数
# person('duoceshi', 4, addr='nanshan')
# 仅接收指定的参数
person('duoches', '3', **extra)
```

```
duoches 3 guangzhou Engineer
```

- 如果已经包含了可变长参数，则不需要再加分割符了

```
# 指定只接收特定的关键字参数，使用*分隔符
def person(name, age, *args, city, job):
    print(name, age, args, city, job)
extra = {'city': 'Shenzhen', 'job': 'Engineer'}
# 仅接收指定的参数
person('duoceshi', 4,*[1,1,1,1,1], **extra)
```

```
duoceshi 4 (1, 1, 1, 1, 1) Shenzhen Engineer
```

参数组合

- 参数组合顺序为：必选参数、默认参数、可变参数、命名关键字参数和关键字参数
- 在函数调用的时候，Python解释器自动按照参数位置和参数名把对应的参数传进去

```
def f1(a, b, c=0, *args, **kw):
    print('a =', a, 'b =', b, 'c =', c, 'args =', args, 'kw =', kw)

def f2(a, b, c=0, *, d, **kw):
    print('a =', a, 'b =', b, 'c =', c, 'd =', d, 'kw =', kw)

f1(1, 2, 3, 'a', 'b', x=99)
# # 通过元组+字典调用
# f1(*(1, 2, 3, 'a', 'b'), **{"x": 99})

# f2(1, 2, d=99, ext=None)
```

```
a = 1 b = 2 c = 3 args = ('a', 'b') kw = {'x': 99}
a = 1 b = 2 c = 3 args = ('a', 'b') kw = {'x': 99}
```

- 小结：

    1. 虽然可以组合多达5种参数，但不要同时使用太多的组合，否则函数接口的可理解性很差
    2. 对于任意函数，都可以通过类似func(*args, **kw)的形式调用它，无论它的参数是如何定义

的

总结

1. Python参数形态很灵活，支持传递复杂参数
2. 默认参数一定要用不可变对象
3. 可变参数：*args是可变参数，args接收的是一个tuple
4. 关键字参数：**kw是关键字参数，kw接收的是一个dict
5. 可变参数两种传参方式：a. 直接传入：`func(1, 2, 3)` b. 通过列表或元组传：`func(*(1, 2, 3))`
6. 关键字参数两种传参方式：a. 直接传入：`func(a=1, b=2)` b. 通过字典传参数：`func(**{'a': 1, 'b': 2})`
7. 使用`*args`和`**kw`是习惯写法，可以用其他名称，但最好用这两个写法
8. 命名关键字参数是为了限制调用者乱传参数，同时可以提供默认值
9. 定义命名的关键字参数在没有可变参数的情况下不要忘了写分隔符`*`，否则定义的将是位置参数

提升练习

写一个函数，随机生成字符串，要求：字符串可以是：数字，字母，中文，特殊字符的随机组合，而且要求可以指定某个单独的字符类型出现的次数，默认是0： 如：def create_str(a, b, c, d):
create_str(1,2,2,3) 生成：中!个s1d&*

```
import random, string
def create_str(num_int=0, num_letters=0, num_zh=0, num_pun=0):
    ran_list = []
    for i in range(num_int):
        # ran_list.append(str(random.randint(0, 9)))
        ran_list.append(random.choice((string.digits)))
    for i in range(num_letters):
        ran_list.append(random.choice(string.ascii_letters))
    for i in range(num_zh):
        ran_list.append(chr(random.randint(0x4e00, 0x9fbf)))
    for i in range(num_pun):
        ran_list.append(random.choice(string.punctuation))
    random.shuffle(ran_list)
    return ''.join(ran_list)
print(string.punctuation)
create_str(1,2,3,4)
```

```
!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
```

```
'埻(猨|伙@0E*t'
```

递归函数

- 一个函数(方法)在内部调用自己本身
- 计算阶乘n! = 1 x 2 x 3 x ... x n，用函数fact(n)表示，可以看出：fact(n) = n! = 1 x 2 x 3 x ... x (n-1) x n = (n-1)! x n = fact(n-1) x n

```
# 递归
def fact(n):
    return n * fact(n - 1)
```

真实案例

- 1. 从接口返回的结果中筛选自己需要的结果

```
# 格式化数据，即：接口请求后得到的结果
dcs = {
    "name": "duoceshi",
    "age": "4",
    "grades1": {
        "number": 1,
        "teachers": ["pansir", "chensir"],
        "students": ["xiaohong", "xiaoming", "xiaobai"],
        "content": "python2"
    },
    "grades2": {
        "number": 2,
        "teachers": ["pansir", "chensir"],
        "students": ["tom", "jerry", "bob"],
        "content": "python3"
    },
    "grades3": {
        "number": 3,
        "teachers": ["pansir", "chensir"],
        "students": ["阿毛", "阿珍", "阿强"],
        "content": "java"
    }
}
# 取出所有课程，组成列表
def get_content():
    all_stu = []
    # 遍历所有字典的key
    for key in dcs:
        if "grade" in key:
    #         print(key)
```

```
    #         print(dcs[key]["content"])
    #         print(dcs[key]["content"])
            all_stu.append(dcs[key]["content"])
print(all_stu)
```

```
['python2', 'python3', 'java']
```

- 2. 当字段内容变更，str变字典；或字典值变更，grades1变为level1，都会造成原有的方法不可用

```
# 格式化数据，即：接口请求后得到的结果
dcs = {
    "name": "duoceshi",
    "age": "4",
    "grades1": {
        "number": 1,
        "teachers": ["pansir", "chensir"],
        "students": ["xiaohong", "xiaoming", "xiaobai"],
        "content": {"theory": "auto-test", "language": "python2"}
    },
    "grades2": {
        "number": 2,
        "teachers": ["pansir", "chensir"],
        "students": ["tom", "jerry", "bob"],
        "content": {"theory": "auto-test", "language": "python3"}
    },
    "grades3": {
        "number": 3,
        "teachers": ["pansir", "chensir"],
        "students": ["阿毛", "阿珍", "阿强"],
        "content": {"theory": "auto-test", "language": "java"}
    }
}
# 旧方法已经不能满足需求，需要重写
# all_stu = []
# for key in dcs:
#     if "grade" in key:
#         print(dcs[key]["content"])
#         all_stu.append(dcs[key]["content"])
# print(all_stu)
# 重写方法
all_stu = []
for key in dcs:
    if "grade" in key:
        print(dcs[key]["content"]["language"])
        all_stu.append(dcs[key]["content"]["language"])
print(all_stu)
```

```
python2
python3
java
['python2', 'python3', 'java']
```

- 3. 引入递归来写一个通用方法，来解析任何字段；遍历多重字典，找到需要的key

```
# 格式化数据，即：接口请求后得到的结果
dcs = {
    "name": "duoceshi",
    "age": "4",
    "grades1": {
        "number": 1,
        "teachers": ["pansir", "chensir"],
        "students": ["xiaohong", "xiaoming", "xiaobai"],
        "content": {"theory": "auto-test", "language": "python2"}
    },
    "grades2": {
        "number": 2,
        "teachers": ["pansir", "chensir"],
        "students": ["tom", "jerry", "bob"],
        "content": {"theory": "auto-test", "language": "python3"}
    },
    "grades3": {
        "number": 3,
        "teachers": ["pansir", "chensir"],
        "students": ["阿毛", "阿珍", "阿强"],
        "content": {"theory": "auto-test", "language": "java"}
    }
}


# 1. 遍历1个字典，getkey我们需要的那个key
# def get1(getkey, res_dict):
#     for k, v in res_dict.items():
#         print(k, v)


# get1("grades", dcs)


# # 2. 增加递归逻辑，增加字典判断，深度遍历完所有的字段
def get2(getkey, res_dict):
    if isinstance(res_dict, dict):
        for k, v in res_dict.items():
            print(k, v)
```

```
#             get2(getkey, v)
get2("grades", dcs)

# # get2("language", dcs)

# # 3. 把需要的字段取出来
# exp = []


# def get3(getkey, res_dict):
#     if isinstance(res_dict, dict):
#         for k, v in res_dict.items():
#             if k == getkey:
#                 print(v)
#                 # 把找到数据增加到exp中
#                 # exp.append(v)
#                 # 如果每个v只要找一个，可以手动中断
#                 # break
#             get3(getkey, v)


# get3("language", dcs)
# print(exp)
```

```
name duoceshi
age 4
grades1 {'number': 1, 'teachers': ['pansir', 'chensir'], 'students':
['xiaohong', 'xiaoming', 'xiaobai'], 'content': {'theory': 'auto-test',
'language': 'python2'}}
number 1
teachers ['pansir', 'chensir']
students ['xiaohong', 'xiaoming', 'xiaobai']
content {'theory': 'auto-test', 'language': 'python2'}
theory auto-test
language python2
grades2 {'number': 2, 'teachers': ['pansir', 'chensir'], 'students': ['tom',
'jerry', 'bob'], 'content': {'theory': 'auto-test', 'language': 'python3'}}
number 2
teachers ['pansir', 'chensir']
students ['tom', 'jerry', 'bob']
content {'theory': 'auto-test', 'language': 'python3'}
theory auto-test
language python3
grades3 {'number': 3, 'teachers': ['pansir', 'chensir'], 'students': ['阿毛',
'阿珍', '阿强'], 'content': {'theory': 'auto-test', 'language': 'java'}}
number 3
teachers ['pansir', 'chensir']
students ['阿毛', '阿珍', '阿强']
```

```
content {'theory': 'auto-test', 'language': 'java'}
theory auto-test
language java
```

```python
a = 1
print(type(a))
# 1. 变量  2. 类型
# 判断数据类型，如果符合，则返回True
# int string dict list
isinstance(a, list)
# 1
b = {"name": "duoceshi", "age": 4}
for i in b:
    print(i)
    print(b[i])
# for k, v in b.items():
#     print(k)
#     print(v)
```

```
<class 'int'>
name
duoceshi
age
4
```

- 4. 场景复杂化，字典的值为列表，如何实现递归?

```python
# 格式化数据，即：接口请求后得到的结果
dcs = {
    "name": "duoceshi",
    "age": "4",
    "grades1": {
        "number": 1,
        "teachers": ["pansir", "chensir"],
        "students": [{"shenzhen": "xiaohong"}, {"guangzhou": "xiaobai"}],
        "content": {"theory": "auto-test", "language": "python2"}
    },
    "grades2": {
        "number": 2,
        "teachers": ["pansir", "chensir"],
        "students": [{"shenzhen": "tom"}, {"guangzhou": "jerry"}],
        "content": {"theory": "auto-test", "language": "python3"}
    },
    "grades3": {
        "number": 3,
        "teachers": ["pansir", "chensir"],
```

```
        "students": [{"shenzhen": "阿珍"}, {"guangzhou": "阿强"}],
        "content": {"theory": "auto-test", "language": "java"}
    }
}
# 1. 无法满足需求，因为值为列表，而目前的递归只能找到 字典下的字典的值
exp = []


def get3(getkey, res_dict):
    if isinstance(res_dict, dict):
        for k, v in res_dict.items():
            if k == getkey:
                print(v)
                # 把找到数据增加到exp中
                exp.append(v)
                # 如果每个v只要找一个，可以手动中断
                # break
            get3(getkey, v)


# 发现找不出来 shenzhen字段到值
get3("shenzhen", dcs)


# 2. 扩展递归函数
def get4(getkey, res_dict):
    if isinstance(res_dict, dict):
        for k, v in res_dict.items():
            if isinstance(v, list):
                for ele in v:
                    get4(getkey, ele)
            if k == getkey:
                print(v)
                # 把找到数据增加到exp中
                exp.append(v)
                # 如果每个v只要找一个，可以手动中断
                # break
            get4(getkey, v)


# get4("shenzhen", dcs)
print(exp)
```

```
xiaohong
tom
阿珍
['xiaohong', 'tom', '阿珍']
```

- 5. 优化逻辑，一个方法，可以查询任何字段的值，而不论结构

```
# 格式化数据，即：接口请求后得到的结果
dcs = {
    "name": "duoceshi",
    "age": "4",
    "grades1": {
        "number": 1,
        "teachers": ["pansir", "chensir"],
        "students": [{"shenzhen": "xiaohong"}, {"guangzhou": "xiaobai"}],
        "content": {"theory": "auto-test", "language": "python2"}
    },
    "grades2": {
        "number": 2,
        "teachers": ["pansir", "chensir"],
        "students": [{"shenzhen": "tom"}, {"guangzhou": "jerry"}],
        "content": {"theory": "auto-test", "language": "python3"}
    },
    "grades3": {
        "number": 3,
        "teachers": ["pansir", "chensir"],
        "students": [{"shenzhen": "阿珍"}, {"guangzhou": "阿强"}],
        "content": {"theory": "auto-test", "language": "java"}
    },
    # 列表下有列表，列表中再包含字典
    "extra": [[{"a": 1}, {"b": 2}], [{"a": "hello"}, {"b": "world"}]]
}

exp = []


# 2. 扩展递归函数
def get4(getkey, res_dict):
    if isinstance(res_dict, dict):
        for k, v in res_dict.items():
            if isinstance(v, list):
                for ele in v:
                    get4(getkey, ele)
            if k == getkey:
                print(v)
                # 把找到数据增加到exp中
                exp.append(v)
                # 如果每个v只要找一个，可以手动中断
```

```
                # break
            get4(getkey, v)


# 发现找不出来 shenzhen字段到值
# get4("a", dcs)


# 3. 优化逻辑
def get5(getkey, res_dict):
    if isinstance(res_dict, dict):
        for k, v in res_dict.items():
            if k == getkey:
                # 把找到数据增加到exp中
                exp.append(v)
                # 如果每个v只要找一个，可以手动中断
                # break
            get5(getkey, v)
    elif isinstance(res_dict, list):
        for ele in res_dict:
            get5(getkey, ele)
# 可以找出任何字段
get5("b", dcs)

print(exp)
```

```
[2, 'world']
```

- 小结
    1. 使用递归函数的优点是逻辑简单清晰，缺点是过深的调用会导致栈溢出。
    2. Python标准的解释器没有针对尾递归做优化，任何递归函数都存在栈溢出的问题。
- 课堂练习题：通过递归函数，解析接口请求结果：获取所有的图片url链接，并将图片下载到本地。url地址：https://tieba.baidu.com/hottopic/browse/topicList