Python 基础

Day 05. Python 函数 & 函数式编程

1. 调用函数

Python 内置函数

> Built-in Functions

```
print(dir('__builtins__'))
```

```
print(abs(-1))

print(max(1, 2))
print(min(1, 2, 3))
```

数据类型转换函数

函数	作用
int(x)	将 x 转换为整数
float(x)	将 x 转换到浮点数
str(x)	将对象 x 转换为字符串
tuple(s)	将序列 s 转换为元组
list(s)	将序列 s 转换为列表
set(s)	将序列 s 转换为集合

```
a = 1
b = float(a)
print(b)
# 1.0
print(type(b))
# <class 'float'>

a = [1, 2, 3]
b = tuple(a)
```

```
print(b)
# (1, 2, 3)
c = set(a)
d = set(b)
print(c,d)
# {1, 2, 3} {1, 2, 3}

a = '12'
b = int(a)
print(b)
# 12
print(type(b))
# <class 'int'>
c = float(a)
print(c)
# 12.0
a = 123
b = str(a)
print(b,type(b))
# 123 <class 'str'>
```

函数别名

> 函数名是指向一个函数对象的引用

```
absolute = abs

print(absolute(-1))
```

2. 定义函数

```
def function_name(parameter(s)):
    """ documentation """
    function body
    return value
```

编程规范

- 函数名：小写，下划线区分单词
- 函数上下各留两空行

返回控制

- 没有值返回，默认返回 `None` ，可以写为 `return`
- 返回多个值，本质 `tuple`

```
def multi_return(x, y):
    """ this is a document... """
    return x, y


a, b = multi_return(1, 2)

print(a, b)

print(multi_return(1, 2))
```

导入函数

```
from python_file_name import function_name
```

```
from function_test import multi_return

print(multi_return(1, 2))
```

空函数

```
def some_function():
    pass
```

异常处理

```
def my_abs(x):
    if not isinstance(x, (int, float)):
        raise TypeError('Error, message...')
    if x >= 0:
        return x
    else:
        return -x
```

3. 函数参数

位置参数

- 必须传入
- 保证顺序

```
def power(x):
    return x * x
```

```
def power(x, n):
    p = 1
    while n > 0:
        p *= x
        n -= 1
    return p
```

默认参数

```
def power(x, n=2):
    p = 1
    while n > 0:
        p *= x
        n -= 1
    return p
```

- 默认参数可按顺序调用，不必写参数名
- 默认参数也可不按顺序调用，写参数名

```
def fn_default(x, y=1, z=2):
    return x + y - z


print(fn_default(0, 1))
print(fn_default(0, z=1))
```

- 位置参数在前，默认参数在后
- 把变化小的参数作为默认参数
- 默认参数必须指向不变的对象

```
def fn_append(array=[]):  # default argument value is mutable
    array.append('END')
    return array


print(fn_append([1, 2, 3]))

print(fn_append())
print(fn_append())
```

```
def fn_append(array=None):
    if array is None:
        array = []
    array.append('END')
    return array


print(fn_append([1, 2, 3]))

print(fn_append())
print(fn_append())
```

可变参数

- `*` + 参数名
- 参数 **个数** 可变
- 可变参数被组装为 `tuple`

```
def fn_sum(*numbers):
    s = 0
    for n in numbers:
        s += n
    return s


print(fn_sum())
print(fn_sum(1))
print(fn_sum(1, 2))

num = [1, 2, 3]  # list
print(fn_sum(num[0], num[1], num[2]))
print(fn_sum(*num))  # pass each list element as parameter
```

关键字参数

keyword arguments

- ** + 参数名
- 可传入任意含参数名的参数
- 关键字参数被组装为 dict

```
def fn_keywords(email, password, **kv):
    print(email, password, kv)


fn_keywords('tom@tom.com', '123', age=18, married=False)

props = {'age': 22, 'married': True}
fn_keywords('jerry@tom.com', '123', **props)
```

命名关键字参数

named keyword arguments

- *, + 参数名
- 限制关键字参数名
- 必须传入参数名
- 函数定义中有可变参数，可省略 *

```
def fn_named_keywords(email, password, *, age, married=False):
    print(email, password, age, married)


fn_named_keywords('tom@tom.com', '123', age=18)


# *args
def fn_named_keywords(email, password, *args, age, married=False):
    print(email, password, args, age, married)


fn_named_keywords('tom@tom.com', '123', 1, 2, 3, age=18)
```

参数组合

- 参数定义的顺序必须是：

> 必选参数, 默认参数, 可变参数, 命名关键字参数, 关键字参数

```
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)
```

4. 递归函数

> 函数在内部调用函数本身

```
def fn_recursive(n):
    if n == 1:
        return 1
    else:
        return n * fn_recursive(n - 1)  # recursive call
```

```
print(fn_recursive(5))
```

5. 高阶函数

> 函数可以接收另一个函数作为参数，这个函数就是高阶函数 `Higher-Order Functions`

```
def fn_higher(f, *numbers):
    s = 0
    for number in numbers:
        s += f(number)
    return s

print(fn_higher(abs, 1, 2, -1, -2, 0))
```

map

> `map` 函数接收两个参数，一个是函数，一个是 Iterable
> `map` 将传入的函数依次作用到序列的每个元素，并把结果作为新的 Iterator 返回

```
def power(x, n=2):
    p = 1
    while n > 0:
        p *= x
        n -= 1
    return p


numbers = [1, 2, 3, 4]

print(list(map(power, numbers)))

print(list(map(str, numbers)))
```

reduce

```
reduce(f, [x1, x2, x3, x4]) = f(f(f(x1, x2), x3), x4)
```

> `reduce` 把一个函数作用在一个序列 `[x1, x2, x3, ...]` 上，这个函数必须接收两个参数，`reduce` 把结果继续和序列的下一个元素做累积计算

```
from functools import reduce


def power(x, n=2):
    p = 1
    while n > 0:
        p *= x
        n -= 1
    return p


numbers = [1, 2, 3, 4]

print(reduce(pow, numbers))


from functools import reduce
```

```
def fn(x, y):
    return x * 10 + y


print(reduce(fn, [1, 3, 5, 7, 9]))
```

filter

> `filter` 函数接收两个参数，一个是函数，返回布尔值，一个是 Iterable
> `filter` 将传入的函数依次作用到序列的每个元素，并根据返回结果决定元素去留，最后返回新的 Iterator 返回

```
def fn_filter(x):
    return x < 10


print(list(filter(fn_filter, [1, 2, 100, 9, 11])))
```

sorted

> 根据 `key` 指定的排序算法函数进行排序

```
print(list(sorted([1, 2, -100, 9, -11], key=abs)))
```

6. 作业

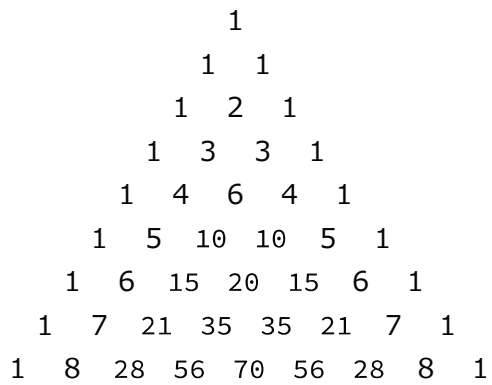
1. 定义函数，统计一个字符串中大写，小写字母的个数
2. 定义函数，把一个 list 的元素去重，返回新 list
3. 斐波那契数列第 n 项 `Fibonacci sequence`

> 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

4. 汉诺塔 n 圆盘移动步骤 `Tower of Hanoi`

```
n = 2
A - B
A - C
B - C
```

5. 打印杨辉三角



6. 参数组合练习

7. 高阶函数 `map` / `reduce` / `filter` / `sorted` 练习