迭代器与生成器

1. 迭代器

1.1 迭代

每一次对过程的重复称为一次“迭代”，而每一次迭代得到的结果会作为下一次迭代的初始值。例如：循环获取容器中的元素。

1.2 可迭代对象iterable

(1) 定义：具有__iter__函数的对象，可以返回迭代器对象。

(2) 语法

```
# 创建：
class 可迭代对象名称：
    def __iter__(self):
        return 迭代器
# 使用：
for 变量名 in 可迭代对象：
    语句
```

(3) 原理：

```
迭代器 = 可迭代对象.__iter__()
while True:
    try:
        print(迭代器.__next__())
    except StopIteration:
        break
```

(4) 演示：

```
message = "我是花果山水帘洞孙悟空"
# for item in message:
#     print(item)

# 1. 获取迭代器对象
```

```
iterator = message.__iter__()
# 2. 获取下一个元素
while True:
    try:
        item = iterator.__next__()
        print(item)
        # 3. 如果停止迭代则跳出循环
    except StopIteration:
        break
```

练习1：创建列表，使用迭代思想，打印每个元素。

练习2：创建字典，使用迭代思想，打印每个键值对。

1.3 迭代器对象iterator

(1) 定义：可以被__next()__函数调用并返回下一个值的对象。

(2) 语法

```
class 迭代器类名:
    def __init__(self, 聚合对象):
        self.聚合对象= 聚合对象

    def __next__(self):
        if 没有元素:
            raise StopIteration
            return 聚合对象元素
```

(3) 说明：聚合对象通常是容器对象。

(4) 作用：使用者只需通过一种方式，便可简洁明了的获取聚合对象中各个元素，而又无需了解其内部结构。

(5) 演示：

```
class StudentIterator:
    def __init__(self, data):
        self.__data = data
        self.__index = -1
```

```
    def __next__(self):
        if self.__index == len(self.__data) - 1:
            raise StopIteration()
        self.__index += 1
        return self.__data[self.__index]


class StudentController:
    def __init__(self):
        self.__students = []

    def add_student(self, stu):
        self.__students.append(stu)

    def __iter__(self):
        return StudentIterator(self.__students)

controller = StudentController()
controller.add_student("悟空")
controller.add_student("八戒")
controller.add_student("唐僧")

# for item in controller:
#     print(item) #
iterator = controller.__iter__()
while True:
    try:
        item = iterator.__next__()
        print(item)  #
    except StopIteration:
        break
```

练习1：遍历图形控制器

```
class GraphicController:
    pass

controller = GraphicController()
controller.add_graphic("圆形")
controller.add_graphic("矩形")
controller.add_graphic ("三角形")

for item in controller:
    print(item)
```

练习2：创建自定义range类，实现下列效果

```
class MyRange:
    pass

for number in MyRange(5):
    print(number)# 0 1 2 3 4
```

2. 生成器generator

(1) 定义：能够动态(循环一次计算一次返回一次)提供数据的可迭代对象。

(2) 作用：在循环过程中，按照某种算法推算数据，不必创建容器存储完整的结果，从而节省内存空间。数据量越大，优势越明显。以上作用也称之为延迟操作或惰性操作，通俗的讲就是在需要的时候才计算结果，而不是一次构建出所有结果。

2.1 生成器函数

(1) 定义：含有yield语句的函数，返回值为生成器对象。

(2) 语法

```
# 创建：
def 函数名():
    …
    yield 数据
    …

# 调用：
for 变量名 in 函数名():
    语句
```

(3) 说明：

-- 调用生成器函数将返回一个生成器对象，不执行函数体。

-- yield翻译为”产生”或”生成”

(4) 执行过程：

a. 调用生成器函数会自动创建迭代器对象。

b. 调用迭代器对象的**next**()方法时才执行生成器函数。

c. 每次执行到yield语句时返回数据，暂时离开。

d. 待下次调用**next**()方法时从离开处继续执行。

(5) 原理：生成迭代器对象的大致规则如下

a. 将yield关键字以前的代码放在next方法中。

b. 将yield关键字后面的数据作为next方法的返回值。

(6) 演示：

```
def my_range(stop):
    number = 0
    while number < stop:
        yield number
        number += 1

for number in my_range(5):
    print(number)  # 0 1 2 3 4
```

练习1：定义函数,在列表中找出所有偶数

[43,43,54,56,76,87,98]

练习2：定义函数,在列表中找出所有数字

[43,"悟空",True,56,"八戒",87.5,98]

2.2 内置生成器

1、枚举函数enumerate

(1) 语法：

```
for 变量 in enumerate(可迭代对象):
    语句

for 索引，元素in enumerate(可迭代对象):
    语句
```

(2) 作用：遍历可迭代对象时，可以将索引与元素组合为一个元组。

(3) 演示：

```
list01 = [43, 43, 54, 56, 76]
# 从头到尾读            -- 读取数据
for item in list01:
    print(item)

# 非从头到尾读          -- 修改数据
for i in range(len(list01)):
    if list01[i] % 2 == 0:
        list01[i] += 1

for i, item in enumerate(list01):  # -- 读写数据
    if item % 2 == 0:
        list01[i] += 1
```

练习：将列表中所有奇数设置为None，所有偶数自增1

2、zip

(1) 语法：

```
for item in zip(可迭代对象1, 可迭代对象2):
    语句
```

(2) 作用：将多个可迭代对象中对应的元素组合成一个元组，生成的元组个数由最小的可迭代对象决定。

(3) 演示：

```
list_name = ["悟空", "八戒", "沙僧"]
list_age = [22, 26, 25]

# for 变量 in zip(可迭代对象1,可迭代对象2)
for item in zip(list_name, list_age):
    print(item)
# ('悟空', 22)
# ('八戒', 26)
# ('沙僧', 25)
```

```
# 应用:矩阵转置
map = [
    [2, 0, 0, 2],
    [4, 2, 0, 2],
    [2, 4, 2, 4],
    [0, 4, 0, 4]
]
# new_map = []
# for item in zip(map[0],map[1],map[2],map[3]):
#     new_map.append(list(item))
# print(new_map)

# new_map = []
# for item in zip(*map):
#     new_map.append(list(item))

new_map = [list(item) for item in zip(*map)]
print(new_map)
# [[2, 4, 2, 0], [0, 2, 4, 4], [0, 0, 2, 0], [2, 2, 4, 4]]
```

练习：使用学生列表封装以下三个列表中数据

list_student_name = ["悟空", "八戒", "白骨精"]

list_student_age = [28, 25, 36]

list_student_gender = ["男", "男", "女"]

2.3 生成器表达式

(1) 定义：用推导式形式创建生成器对象。

(2) 语法：

```
变量 = (表达式 for 变量 in 可迭代对象 if 条件)
```

练习1：使用生成器表达式在列表中获取所有字符串.

list01 = [43, "a", 5, True, 6, 7, 89, 9, "b"]

练习2：在列表中获取所有整数,并计算它的平方.