

## 泛型 Generics

### 字符串格式化 (string formatting methods)

- 1) 使用加号 (常见的合并字符串的方式) → 这里的 %s 表示字符串; 亦可用 %.nf 表示保留 n 位小数的浮点数等.
- 2) 使用百分号 a='Hello' print("%s,%s" % (a,a)) >> Hello,Hellopython
- 3) 使用 str.format() 方法 一个在 python 2.6 中引进的方法, 较为麻烦
- 4) 使用字符串. 已经讲过, 不再赘述
  - + 字符串中保留 n 位小数的方法是 变量名:.nf
  - 对变量的输出同样调用了 \_\_str\_\_() 方法, 可以与 "Special Objects Methods" 结合起来考虑.
  - + 字符串的许多衍生方法可以询问 chatGPT (误)

## 泛型

```
def map(items, func):  
    mapped = []  
    for item in items:  
        mapped.append(func(item))  
    return mapped
```

对于这个函数, 传入的参数 items 可以是任意可迭代对象.  
我们说 "The function map is generic in the type of items."

Duck typing: 对于一个泛型函数 func, 它须通过 duck test.

我们不必具体关注变量的具体实现方法, 只要具有一些必需的性质即可.

例如: 对于一个物体, 我们只需考察它能否像鸭子一样走, 不必通过 DNA 检测证明.

当对 Link 对象调用上述定义的 map 函数时, 由于 Link 对象是不可迭代的, 会报错

为了让它们通过 ducking test, 可以为 Link 对象添加 \_\_iter\_\_() 方法.

```
def __init__(self):  
    current = self  
    while current is not Link.empty:  
        yield current.first  
        current = current.rest
```

通过在 Iterator and Generator 中学习的生成迭代器的方法, 我们可以很容易地写出对应的 \_\_iter\_\_() 方法.

通过对对象的自定义泛型语法可以让它们通过 ducking test. 可以为它们定义 \_\_add\_\_(),  
\_\_radd\_\_(), ... 等方法来增加对象的泛用性.

让它们的行为看上去像鸭子, 尽管它们本身不是鸭子, 又有什么关系呢?

调用方法