笔记

作者：Jeff

一、函数递归【了解】

1.概念

递归函数：一个会调用自身的函数【在一个函数的内部，自己调用自己】

递归调用

递归中包含了一种隐式的循环，他会重复指定某段代码【函数体】，但这种循环不需要条件控制

使用递归解决问题思路：

a.找到一个临界条件【临界值】

b.找到相邻两次循环之间的关系

c.一般情况下，会找到一个规律【公式】

2.使用

代码演示：

```
# 函数自己调用自己就是递归
# 需求:通过递归的方式,封装一个函数,计算 n! (n是传入的数字)


# 5! ===> 5*4*3*2*1==>5 * 4!
# 4! ===> 4*3*2*1 ===>4 * 3!
# 3! ===> 3*2*1====>  3 * 2!
# 2! ===> 2*1  ===>   2 * 1!
# 1! ===> 1

#总结如下:
# n! = n * (n-1)!
'''
a.找到一个临界值(临界条件)   1! = 1
b.找到两个循环之间的关系
c.总结规律: n! = n * (n-1)!
'''
def digui(n):
   if n == 1:
      return 1
   return n * digui(n-1)
print(digui(6))
```

```
# 通过递归封装一个函数,传入一个数字m,得到第m个斐波那契数列
# 1 1 2 3 5 8 13 21 34 55 89
def fn(m):
  if m < 3:
     return 1
  return fn(m-1) + fn(m-2)
print(fn(8))
```

注意：以后在实际项目中尽量少用递归，如果隐式循环的次数太多，会导致内存泄漏【栈溢出】 python默认的递归次数限制为1000次左右.以免耗尽计算机的内存.

优点：简化代码，逻辑清晰

二、列表推导式和生成器【掌握】

1.列表生成式/列表推导式 【掌握】

list comprehension

系统内置的用于创建list的方式

range(start,end,step)缺点:生成的列表一般情况下都是等差数列

代码演示：

```
# 最基本的列表
# 1.生成1-10之间所有的数字
list1 = list(range(1,11))
print(list1)

# 需求:通过程序的方式生成列表 [1,4,9,16,25]
#第一种方法:使用原始的方式生成
list2 = []
for i in range(1,6):
list2.append(i ** 2)
print(list2)

# 第二种方法:使用列表生成式
list3 = [i**2 for i in range(1,6)]
print(list3)   # [1, 4, 9, 16, 25]

# 使用列表生成式 生成1-10之间所有的奇数
list4 = [i for i in range(1,11) if i % 2 == 1]
print(list4)

# 使用列表生成式 生成1-10之间所有的奇数并且能被3整除的数字
list5 = [i for i in range(1,11) if i % 2 == 1 and i % 3 == 0]
print(list5)
```

```
# 列表生成式中使用双重循环
list6 = [i + j for i in "xyz" for j in "987"]
print(list6)

# 字典生成式:(了解)
dict1 = {i:i*i for i in range(1,6)}
print(dict1)

# 集合生成式:(了解)
set1 = {i*i for i in range(1,6)}
print(set1)
```

2.生成器【掌握】

> generator
>
> next()
>
> 代码演示:

```
# 定义生成器: 只需要把列表生成式的[] 换成() 即可
g = (i ** 2 for i in range(1,10))
print(g)
print(type(g))  #<class 'generator'>

# 生成器得到的数据不能直接输出,要想访问里面的数据,必须使用next() 进行遍历,每调用一次next() 就会输出一个数据
print(next(g))
print(next(g))
print(next(g))
print(next(g))
print(next(g))
print(next(g))
print(next(g))
print(next(g))
print(next(g))

# 当通过next() 将生成器中的数据遍历完毕以后,接着调用next(),就会出现StopIteration
# print(next(g))   # StopIteration

# 生成器的应用场景:
'''
当需要生成无穷多的数据时,一般使用生成器. 若使用列表生成无穷多的数,会占用大量的空间,不合适.
'''

# 生成器相关的函数 yield
'''
1.yield存在于函数中,当普通函数中有了yield关键词,该普通函数变为了生成器函数.
2.调用生成器函数不会执行代码,需要使用next
```

```
3.可以在函数的内部不断地返回值,但是不会终止函数的执行(与return不同)
4.每次调用next以后,程序会在yield处暂停
'''

def test(m):
  print("m=",m)
  yield m * 2
  print("m+1=",m+1)
  yield (m + 1) * 2
  print("m+2",m+2)

g = test(23) # 在普通函数test中加入了关键字yield,那该函数变成了生成器函数,直接调用不会执行.
print(type(g))   # <class 'generator'>
print(next(g))    # 函数执行到第一个yiled处暂停
print(next(g))    # 函数执行到第二个yiled处暂停
```

三、迭代器【了解】

1.可迭代对象

可迭代对象【实体】：可以直接作用于for循环的实体【Iterable】

可以直接作用于for循环的数据类型：

a.list,tuple,dict,set,string

b.generator【() 和yield】

isinstance:判断一个实体是否是可迭代的对象

代码演示：

```
# 导入模块的方式:
# 第一种: import 模块名
# 第二种:from 模块名 import 函数

# 使用可迭代对象    需要导入collections.abc
from collections.abc import Iterable

# 通过 isinstance:判断一个实体是否是可迭代的对象     返回值是bool类型,True或者False
# 可迭代对象:能够被for 循环遍历的实体就是可迭代对象

print(isinstance([],Iterable))   # True
print(isinstance((),Iterable))   # True
print(isinstance({},Iterable))   # True
print(isinstance("loha",Iterable)) # True
# 生成器
print(isinstance((i for i in range(1,10)),Iterable))  # True

# 列表\元组\字典\字符串\生成器  都是可迭代对象
```

```
print(isinstance(11,Iterable))  # False
print(isinstance(True,Iterable))  # False

# 整型\浮点型\布尔类型 都不是可迭代对象
```

2.迭代器

迭代器: 不但可以作用于for循环，还可以被next函数遍历【不断调用并返回一个元素，直到最后一个元素被遍历完成，则出现StopIteration】

目前为止，只有生成器才是迭代器【Iterator】

结论：迭代器肯定是可迭代对象，但是，可迭代对象不一定是迭代器

isinstance:判断一个实体是否是迭代器

代码演示：

```
from collections.abc import Iterator

# 迭代器:既能够通过 for循环遍历,也能next函数遍历的实体叫做迭代器.
# 目前只有生成器是迭代器

# 使用 isinstance() 判断某个实体是否是迭代器

print(isinstance([],Iterator))  #False
print(isinstance((),Iterator))  #False
print(isinstance({},Iterator))  #False
print(isinstance("halo",Iterator))  #False
print(isinstance(11,Iterator))  #False
print(isinstance(False,Iterator))   #False

# 只有生成器是迭代器
print(isinstance((i for i in range(1,10)),Iterator))  # True
```

3.可迭代对象和迭代器之间的转换

可以将 可迭代对象转换为迭代器：iter()

代码演示：

虽然list tuple str dict set 这些可迭代对象不能使用next() 函数遍历,但是可以通过iter() 函数将可迭代对象转换为迭代器.

```
# 可迭代对象转换为迭代器: iter()
list = [12,4,5,78]
# print(next(list))   # 'list' object is not an iterator
list1 = iter(list)   # 将列表转换为 迭代器
print(next(list1))   # 12
print(next(list1))   # 4
print(next(list1))   # 5
print(next(list1))   # 78
```

四、包和模块【掌握】

1.概述

为了解决维护问题，一般情况下，在一个完整的项目中，会将特定的功能分组，分别放到不同的文件中，在使用的过程中，可以单独维护，各个不同的文件之间互不影响，每个.py文件就被称为一个模块，通过结合包的使用来组织文件

封装思路: 函数 => 类 => 模块 => 包 => 项目

优点:

a.提高了代码的可维护性

b.提高了代码的复用性【当一个模块被完成之后，可以在多个文件中使用】

c.引用其他的模块【第三方模块】

d.避免函数名和变量的命名冲突

```
# 模块
'''
模块就是一个python文件
模块分类:
1.内置模块: python 本身提供的模块    比如:os  random  time
2.自定义模块: 我们自己根据项目的需求,自己书写的模块
3.第三方模块: 别人写好的具有特殊功能的模块
   a.在使用第三方模块的时候,需要先安装  比如:numpy  pandas  requests
   b.导入和使用
'''

# 模块导入的方式:
# 第一种: import 模块名
# 第二种: from 模块名 import 模块名里面的方法
# 示例:
# 导入内置模块
import os
from random import randint
```

2.自定义模块【掌握】

2.1自定义import模块

> 代码演示：

```
# 导入自定义模块
# import module1
# 使用自定义模块的变量
# print(module1.name)
# print(module1.age)
# print(module1.love)
```

2.2自定义from-import模块

> 代码演示：

```
from module1 import *
```

2.3自定义from-import*模块

> 代码演示：

```
# * 表示通配符  模糊导入,使用*号后,可以直接使用模块中的所有的内容,比如变量,方法等(不推荐使用)
'''
from module1 import *
print(name)
print(age)
print(love)
'''

# 精确导入 (推荐)
from module1 import name,age
print(name)
print(age)

# 给模块起别名 as
import random as r
# print(r.randint(1,10))
```

> 总结：在python中，每个py文件其实都是一个模块，如果跨模块调用函数，则采用导入的方式
>
> 将不同的功能进行划分，调用函数的时候相对比较方便的

2.4第三方模块

```
第三方模块：别人写好的具有特殊功能的模块
   a.在使用第三方模块的时候,需要先安装  比如:numpy  pandas
   b.导入和使用

# 更新pip 版本
# python -m pip install --upgrade pip

第一种操作第三方模块的方式：
    pip  专门用来安装和卸载python相关扩展的工具
        pip -V 查看当前pip的版本号
        pip list 查看当前项目安装的所有的扩展
        pip install 扩展名(包名)   安装指定的包
        设置临时镜像   pip install 扩展名(包名) -i 国内的镜像源
            一般我们在安装包的时候,会使用国内的镜像

        设置永久镜像：  pip config set global.index-url
https://mirrors.aliyun.com/pypi/simple/

          安装numpy框架
          pip install numpy -i https://mirrors.aliyun.com/pypi/simple/

          练习：尝试去安装 numpy pandas flask  requests  lxml

        pip uninstall 扩展名(包名)
           pip uninstall flask

        pip show 扩展名(包名)

第二种操作第三方模块的方式：
  通过pycharm编辑器：文件->设置->项目->python解释器：
    + 表示安装包
    - 表示卸载包
```