1. 多项式ADT链表实现：

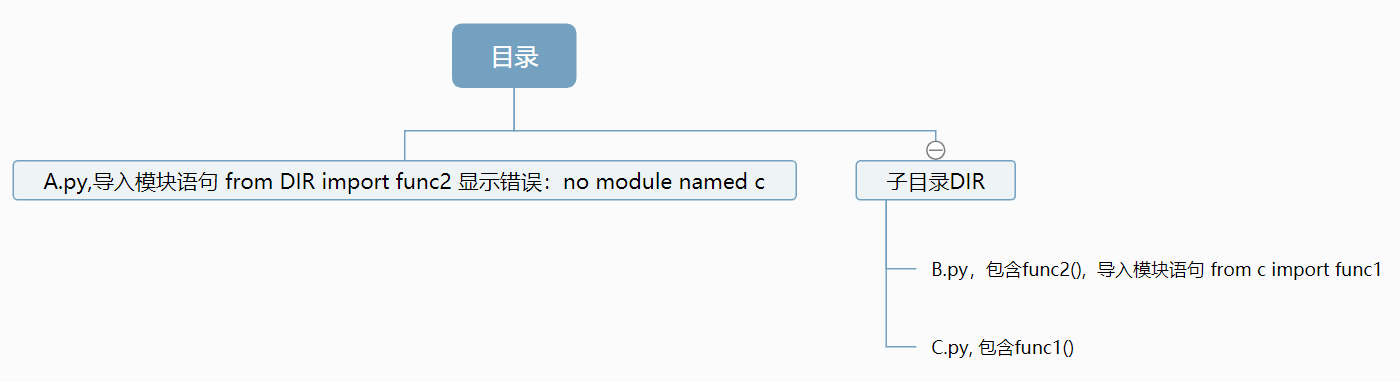
* **完成：**多项式的链表实现、多项式加法、多项式乘法
* **遗留：**多项式乘法中得到的结果链表未合并同类项，也未按照指数降序排列

1. 装饰器内的全局变量

在装饰器外定义变量，装饰器内部函数修改变量时无需加global关键字，为什么？

1. **def** print\_score**(**self**):**
2. **print** '%s: %s' **%** **(**self**.**name**,** self**.**score**)**
3. 链表的应用

* **基数排序：**通过链表实现桶式排序及基数排序性
* **多重表：**通过链表实现多重表

1. 循环链表和python实现，双向链表已实现
2. 不同层级下的模块调用。A调用B模块，B调用C模块，B、C位于同一包下，直接运行B，import语句没问题，但运行A，显示B中import语句有问题，B中的import语句为：from c import func1()，A中的import语句为：from b import func2()
3. 栈的应用：方法调用
4. 在python中使用数组，并以数组实现链表、栈、队列
5. 返回节点Node的时候直接打印显示\_\_str\_\_需要打印字符串，而返回值类型是Node，但是直接打印Node实例是可以的
6. 二叉树应用：表达式树
7. 二叉树的序列化与反序列化

序列化容易，遍历即可，反序列化目前存在困难

1. Python时间格式化输出

strftime()函数接收以时间元组(struct\_time对象)，并返回以可读字符串表示的当地时间，格式由参数format决定

1. t **=** time**.**time**()//**获得以秒为单位的时间
2. **print(**time**.**strftime**(**"%b %d %Y %H:%M:%S"**,** time**.**gmtime**(**t**)))//**gmtime获得struct\_time对象

格式参数format，详见[附录3.python中时间日期格式化符号](#日期格式)

1. 格式化输出

格式输出详见参考文档 [《python的格式化输出》](file:///C:\Users\Alta\Desktop\编程笔记\python的格式化输出.pdf)

1. Windows环境下文件路径表示

Python代码里面，反斜杠“\”是转义符，例如“\n”表示回车，采用以下三种方式表示路径：

* 斜杠“/”，如“c:/test.txt”
* 两个反斜杠“\\”，如“c:\\test.txt”
* 字符串前面加上字母r，表示后面是一个原始字符串raw string，如“r“c:\\test.txt””

1. Python接收命令行参数

导入argv，结果即为参数列表

1. **from** sys **import** argv
2. **print(**argv**)**
3. **>>>**python xx**.**py xxx
4. **>>>[**'xx.py'**,**'xxx'**]**
5. '\u'前缀字符串

\u4f60十六进制代表对应汉字的utf-16编码

1. 定义1个元素的tuple

定义1个元素的tuple: t = (1,)，加上一个逗号，避免成为数学意义上的括号

1. 创建生成器(generator)的两种方式

方式1: g = (x \* x for x in range(10))，列表生成式的[]更改为()

方式2: 如果一个函数定义中包含yield关键字，那么函数就不再是一个普通函数，而是一个generator，函数是顺序执行，遇到return语句或者最后一行函数语句返回。而generator在每次调用next()的时候执行，遇到yield语句返回，再次执行时从上次返回的yield语句处继续执行

1. **def** fib**(**max**):**
2. n**,** a**,** b **=** 0**,** 0**,** 1
3. **while** n **<** max**:**
4. **yield** b
5. a**,** b **=** b**,** a **+** b
6. n **=** n **+** 1
7. **return** 'done'
8. python代码规范

请参考[《python代码规范》](C:\\Users\\Alta\\Desktop\\编程笔记\\python代码规范.pdf)

1. 迭代器总结

* 凡是可作用于for循环的对象都是Iterable类型；
* 凡是可作用于next()函数的对象都是Iterator类型，它们表示一个惰性计算的序列；
* 集合数据类型如list、dict、str等是Iterable但不是Iterator，不过可以通过iter()函数获得一个Iterator对象。
* Python的for循环本质上就是通过不断调用next()函数实现的

1. Map 函数总结

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，返回新的Iterator。

map(function, iterable1,iterable2 ...)

例如三个列表相乘：

1. list1 **=** **[**1**,**2**,**3**,**4**,**5**,**6**,**7**,**8**,**9**]**
2. list2 **=** **[**1**,**2**,**3**,**4**,**5**,**6**,**7**,**8**,**9**]**
3. list3 **=** **[**9**,**8**,**7**,**6**,**5**,**4**,**3**,**2**,**1**]**
4. **def** foo**(**l1**,**l2**,**l3**):**
5. **return** l1**\***l2**\***l3
6. **print(**list**(**map**(**foo**,**list1**,**list2**,**list3**)))**
7. **>>>[9,32,63,96,125,144,147,128,81]**
8. Reduce函数总结

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。

例如序列求和：

1. **>>>** **from** functools **import** reduce
2. **>>>** **def** add**(**x**,** y**):**
3. **...** **return** x **+** y
4. **...**
5. **>>>** reduce**(**add**,** **[**1**,** 3**,** 5**,** 7**,** 9**])**
6. 25
7. 匿名函数(lambda)总结

lambda parameters: expression

parameters：可选，如果提供，通常是逗号分隔的变量表达式形式，即位置参数。

expression：**不能包含分支或循环（但允许条件表达式）**，也不能包含return（或yield）函数。如果为元组，则应用圆括号将其包含起来。调用lambda函数，**返回的结果是对表达式计算产生的结果**

1. #根据参数是否为1 决定s为yes还是no
2. **>>>** s **=** **lambda** x**:**"yes" **if** x**==**1 **else** "no"
3. 全局变量的使用

* 使用到的全局变量只是作为引用，不在函数中修改它的值的话，不需要加global关键字

1. a **=** 1
2. **def** func**():**
3. **if** a **==** 1**:**
4. **print(**"a: %d" **%**a**)**

* 使用到的全局变量，需要在函数中修改的话，就涉及到歧义问题，因此，需要修改全局变量a，可以在"a = 2"之前加入global a声明

1. a **=** 1
2. **def** func**():**
3. **global** a
4. a **=** 2
5. **print(**"in func a:"**,** a)
6. python 定义类时，内部方法的互相调用

每次调用内部的方法时，方法前面加 self

1. class MyClass:
2. def \_\_init\_\_(self):
3. pass
4. def func1(self):
5. print('a')
6. self.common\_func()
7. def func2(self):
8. self.common\_func()
10. def common\_func(self):
11. pass
12. filter函数总结

filter()也接收一个函数和一个序列。filter()把传入的函数依次作用于每个元素，然后根据返回值是True还是False决定保留还是丢弃该元素

1. sorted函数总结

sorted()函数也是一个高阶函数，它还可以接收一个key函数来实现自定义的排序，key指定的函数将作用于list的每一个元素上，并根据key函数返回的结果进行排序

1. **>>>** sorted**([**36**,** 5**,** **-**12**,** 9**,** **-**21**],** key**=**abs**)**
2. **[**5**,** 9**,** **-**12**,** **-**21**,** 36**]**
3. 返回函数(闭包)
4. 装饰器

这种在代码运行期间动态增加功能的方式，称之为“装饰器”（Decorator）。本质上，decorator就是一个返回函数的高阶函数。

1. **def** log**(**func**):**
2. **def** wrapper**(\***args**,** **\*\***kw**):**
3. **print(**'call %s():' **%** func**.**\_\_name\_\_**)**
4. **return** func**(\***args**,** **\*\***kw**)**
5. **return** wrapper

如果将上述的log函数作为装饰器，则在其装饰的函数前添加@log

1. @log
2. **def** now**():**
3. **print(**'2015-3-25'**)**
4. >>> now()
5. call now():
6. 2015-3-25

装饰器原理及引入机制较为复杂，详情可参考[《python装饰器解释》](file:///C:\Users\Alta\Desktop\编程笔记\python装饰器解释.pdf)

1. 函数参数

* 必选参数(位置参数)：
* 默认参数：如def power(x, n=2)，x即为位置参数，n为默认参数，必选参数在前，默认参数在后
* 可变参数：传入的参数个数是可变的，不必自行将所有参数组装成一个list或tuple，如def calc(\*numbers)。定义可变参数和定义一个list或tuple参数相比，仅仅在参数前面加了一个\*号。在函数内部，参数numbers接收到的是一个tuple

1. **def** calc**(\***numbers**):**
2. sum **=** 0
3. **for** n **in** numbers**:**
4. sum **=** sum **+** n **\*** n
5. **return** sum

* 关键字参数:可变参数在函数调用时自动组装为一个tuple。而关键字参数允许你传入0个或任意个含参数名的参数，这些关键字参数在函数内部自动组装为一个dict:

1. **def** person**(**name**,** age**,** **\*\***kw**):**
2. **print(**'name:'**,** name**,** 'age:'**,** age**,** 'other:'**,** kw**)**
4. **>>>** person**(**'Michael'**,** 30**)**
5. name**:** Michael age**:** 30 other**:** **{}**
6. **>>>** person**(**'Bob'**,** 35**,** city**=**'Beijing'**)**
7. name**:** Bob age**:** 35 other**:** **{**'city'**:** 'Beijing'**}**
8. 字符串里面的引号

单引号' '定义字符串的时候，它就会认为你字符串里面的双引号" "是普通字符，从而不需要转义。反之当你用双引号定义字符串的时候，就会认为你字符串里面的单引号是普通字符无需转义

1. Str1 = "We all know that 'A' and 'B' are two capital letters."
2. Str2 = 'The teacher said: "Practice makes perfect" is a very famous proverb.'
3. 偏函数

functools.partial的作用就是，把一个函数的某些参数给固定住（也就是设置默认值），返回一个新的函数，调用这个新函数会更简单

1. **>>>** **import** functools
2. **>>>** int2 **=** functools**.**partial**(**int**,** base**=**2**)**
3. **>>>** int2**(**'1000000'**)**
4. 64
5. 算法速度的定义

* 大O：T(N) = O(f(N))，T增长率小于等于f
* **Ω**：T(N) =Ω (f(N))，T增长率大于f
* **Θ**：T(N) =Θ (f(N))，T增长率等于f
* 小o：T(N) =o (f(N))，T增长率小于f

1. 计算运行时间的一般法则

* 法则1：for循环：一个for循环的运行时间至多是该循环内语句的运行时间乘以迭代次数
* 法则2：嵌套for循环：嵌套循环内部的一条语句总运行时间为该语句运行时间乘以所有for循环的大小，如下程序片段运行时间为O(N2)

1. **for** i **in** range**(**N**):**
2. **for** j **in** range**(**N**):**
3. j **+=** 1

* 法则3：顺序语句：各个语句运行时间求和
* 法则4：if/else语句：运行时间至多是判断时间+S1或S2中运行时间长者

1. **if** **(**condition**):**
2. S1
3. **else:**
4. S2

导入自定义模块

**附录**

1. 方法解析顺序（Method Resolution Order, MRO）列表

MRO 列表的顺序遵循以下三条原则：

* 子类永远在父类前面
* 如果有多个父类，会根据它们在列表中的顺序被检查
* 如果对下一个类存在两个合法的选择，选择第一个父类

1. 查询模块的帮助文档

* 先导入模块，再查询普通模块的使用方法：help(module\_name)，例：help(math)
* 先导入sys，再查询系统内置模块的使用方法：sys.bultin\_modulenames
* 查看模块下所有函数：dir(module\_name)，例：dir(math)
* 查看模块下特定函数：help(module\_name.func\_name)，例：help(math.sin)
* 查看函数信息的另一种方法：print(func\_name.\_\_doc\_\_)，例：print(sin.\_\_doc\_\_)

1. python中时间日期格式化符号

* %y 两位数的年份表示（00-99）
* %Y 四位数的年份表示（000-9999）
* %m 月份（01-12）
* %d 月内中的一天（0-31）
* %H 24小时制小时数（0-23）
* %I 12小时制小时数（01-12）
* %M 分钟数（00=59）
* %S 秒（00-59）
* %a 本地简化星期名称
* %A 本地完整星期名称
* %b 本地简化的月份名称
* %B 本地完整的月份名称
* %c 本地相应的日期表示和时间表示
* %j 年内的一天（001-366）
* %p 本地A.M.或P.M.的等价符
* %U 一年中的星期数（00-53）星期天为星期的开始
* %w 星期（0-6），星期天为星期的开始
* %W 一年中的星期数（00-53）星期一为星期的开始
* %x 本地相应的日期表示
* %X 本地相应的时间表示
* %Z 当前时区的名称
* %% %号本身