24. Python和课程总结

- Python语言综述
- 内置标准类型
- 内置标准函数
- Python和编程
- 课程总结
- 考试问题

Python语言

- 本课程使用Python语言，讨论程序设计和用计算机解决问题
  - 课程中讨论了各种程序设计技术
  - 下面综合考察Python语言本身
- Python通常被看作是一种脚本编程语言（scripting language）
  - 经常用于写较短小的（计算）脚本，通过丰富的数据结构和库的支持，很短的程序也可能做有意义的工作
  - 不需要“编译”就可以直接解释执行
- 实际上，Python是一种通用编程语言
  - 人们已经用Python写了许多相当大规模的系统，一些系统长期运行，提供各种重要的服务
  - 由于有性能优异的库模块支持，Python也完全可以用于一些计算密集型的应用领域，例如数值计算等

Python语言

- Python标准库包提供了许多重要功能，如前面介绍的基本GUI包和并发执行支持包，大家自学的Turtle包等，还有很多

  如果需要做特定方面的应用，应该找一找相应的专用程序包

  例如，科学计算包NumPy实现高效的数值计算，SciPy把几个与科学计算、可视化等有关的重要的包组合在一起。有人根据具体需要制作了可以独立安装的打包文件，见scipy.org/

- Python程序的执行
  - Python被称为解释性语言，也就是说，用Python写的程序不需要加工（翻译），就可以在一定环境中直接执行
  - 能执行Python程序的环境称为Python解释器（或Python系统），我们用的cpython是官方实现，用C语言开发。存在若干其他实现：www.python.org/download/alternatives/

Python语言

- 人们都说Python是解释执行的，其实Python系统并不是看一行代码解释一行（执行一行），而是采用另一种方式：
  - 执行单位是一个代码块（block）。代码块包括：模块，函数定义，类定义，交互方式输入的一个语句
  - Python深入分析整个代码块，弄清其中各作用域层次中的变量命名（定义）和引用关系，实现该代码块要求的计算
  - 注意：如果一个代码块由一系列语句（包括函数定义和类定义）组成，这些语句将被顺序执行，逐一完成
- 一些情况：
  - 函数定义和类定义中可能用到其他功能，但在定义时，其中语句并不执行，因此即使调用的功能还没定义也没有问题
  - 一旦执行中使用的功能没定义，就会报错

Python语言

- Python并不是“看着”程序代码一条条执行语句
  - 程序里有循环，函数可能被多次调用，如果每次都直接解释执行源程序语句，就可能多次处理程序里的一些结构
  - Python解释器处理一个代码块时，通常先将其翻译为一种方便解释（计算机容易处理）的内部代码形式（bytecode），而后调用一个内部解释器（称为Python虚拟机）去执行
  - 一些情况下Python自动把一些模块的内部代码形式保存下来，下次执行时就直接使用（保存为 .pyc文件，是一种策略）
  - 可调用py-compile包的compile函数从py文件生成内码文件
- 在Python系统中，这种翻译和执行是透明的
  - 通常不需要人工干预
  - 系统保证翻译前后的两种程序形式之间的语义等价性

Python程序结构和名字约束

- **Python程序中的最大单位是模块**
  - 模块层语句定义的变量具有模块的全局作用域，函数和类定义也在全局作用域建立名字约束（就像给其名字赋值）
  - 模块层的函数/类定义形成全局作用域内部嵌套的作用域，在函数和类里还可以有内部嵌套的（函数和类）定义
- 在一个作用域里有定义的名字，在这个作用域可以直接使用
  - 函数里有定义的局部名字（函数参数或函数中语句引进的名字），其定义延伸到函数内部嵌套的作用域（除非被遮蔽）
  - 类中定义的局部名字在类中嵌套的作用域里不能使用，只能通过类名和圆点方式引用（从全局作用域开始描述）
  - 如果把某个名字声明为**global**，所在作用域里的这个名字都表示相应全局约束。**global**声明应该出现在所有使用之前

Python程序的执行

- Python程序或从一个主模块开始执行，或以交互方式执行。首先建立全局名字空间，在这里建立全局作用域要求的约束。名字空间是类似字典的结构，记录本作用域的名字/值约束
- 执行import mdl（mdl是模块名）或类定义在当前名字空间引进相应模块名/类名，其值是模块/类对象，是类似字典的结构

  这种对象将模块成员/类属性映射到相应的值（可以是任何对象，包括函数对象等），可以通过圆点记法访问

  from-import为当前作用域引进模块中定义的（指定）约束
- 执行函数定义在当前名字空间建立函数名到函数对象的约束
  - 只有在函数对象被调用执行时，才建立函数的局部名字空间，每个调用建立一个独立的局部名字空间
  - 创建函数的名字空间时需记录其外围名字空间，nonlocal声明的名字需要在外围名字空间中找到定义（约束）

Python程序和执行

- 创建类的实例对象也建立一个局部名字空间
  - 实例对象里（隐式地）记录着其所属的类，因此可以找到类中定义的方法，也支持**isinstance**和**type**函数
  - 实例对象也是类似字典的结构，属性值约束于属性名
  - 通过类的对象调用实例方法时，该对象本身作为方法的第一个参数约束于方法的**self**参数
- 方法是特殊的函数，两者都在调用时建立的局部名字空间中执行，可按作用域规则访问非局部名。方法能修改调用对象的属性，函数能修改**nonlocal**变量的值，两者都能修改**global**变量的值
- 程序在执行中不断创建各种对象，或基于已有对象创建新对象

  程序员不需要销毁对象。**del x**只是取消变量**x**的约束。如果一个对象失去所有变量约束，所占的存储最终将被系统回收

Python内置类型

- 标准库手册第4章说明了所有Python内置类型的情况
  - 最基本的类型包括bool和各种数值类型
  - 模块、类、函数、方法、None等各有自己的类型
- 数值类型除了各种算术运算外还有几个特殊的操作，包括
  - int类型有一组按位（bitwise）运算（4.1.1节），这些运算把整数看作是二进制位序列进行操作，包括：按位与 &，按位或 |，按位异或 ^，按位否定 ~，左移 << 和右移 >>
  - int对象的bit_length()方法求出整数的二进制表示位数（不考虑正负号，从值非0的最高位开始计）
  - float对象也有几个方法：方法f.is_integer()判断f是否整数，f.as_integer_ratio返回一对整数，其比值正好等于f而且分母非负（相当于转换为值相同的最简有理数）

Python内置类型

- 迭代器是Python语言中的一个核心概念：for循环语句和各种结构类型的描述式都依赖于迭代器的概念
  - 任何支持__next__()方法的类型都是（都可作为）迭代器，在没有下一迭代值时该方法应引发StopIteration异常
  - 任何类型可以定义__iter__()方法返回该类对象的迭代器
  - 生成器函数是定义迭代器的最方便方式，Python的内置序列类型（range/tuple/list）和str的对象、文件对象都是迭代器，set/frozenset和dict都可以作为迭代器使用
- range/tuple/list和str是内置序列类型，支持一组公共序列操作，其中list是可变序列类型，支持变动操作和sort方法
- set/frozenset和dict也是容器类型，元素没有显式定义的顺序，用作迭代器时按某种内部顺序给出元素（对dict是给出关键码），set和dict是可变容器类型，frozenset是不变容器类型

Python内置类型

- 另外一种重要类型称为上下文管理器（context manager）
  - 用于在with语句中管理资源，保证正确释放
  - 打开文件得到的文件类型的对象，用于并发控制的Lock对象，都是系统（或标准库）已定义的上下文管理器
  - 任何类型只需支持__enter__()和__exit__()方法，其对象就可以作为上下文管理器，用在with语句里
- 下面简单介绍几个前面没介绍过的标准内置类型

  写比较简单的程序时，一般不会用到这些类型

  一些特殊的与计算机底层关系密切的程序，或者存储效率要求较高的程序，可能用到这些类型

字节序列类型

- **bytes类型（字节串）：与str类似，但其元素是字节，元素大小统一（str的元素是unicode字符，字符占用的存储可能不统一）**
  - **字面量：b开头的字符串表达形式，同样允许两种单个引号的描述形式和两种连续三个引号的描述形式，如**

    b'How Are you?'　b"I am fine!"　b"""Thank you!\n"""
  - **bytes是不变类型，建立的字节串是不变对象。实际上相当于取值范围在[0, 256)之内的小整数串**
  - **bytes支持类似str的操作集合，详情见第4.8.3节**
- **创建bytes对象的其他方式：**
  - **bytes(10)：建立一个包含10个字节的全0字节串**
  - **bytes(range(20))：基于可迭代对象创建字节串**
  - **bytes(obj)：基于其他合适的对象创建字节串**

字节数组类型

- **bytearray**（字节数组）类型是可变的字节序列类型。没有字面量形式，只能通过类型名构造，例如：
  - **bytearray()**，建立一个空字节数组
  - **bytearray(20)**，建立给定大小的全零字节数组
  - **bytearray(range(20))**，基于适当的迭代器创建
  - **bytearray(b'I am fine')**，基于字节串创建
- **bytearray**支持所有**str**操作，还支持所有不变序列和可变序列操作，包括通过下标赋值，切片赋值等修改字节数组内容的操作

  bytearray对象的**str**操作都创建新**bytearray**对象（不变操作）

  例如，**barray.join(s)**构造一个新字节数组，其中内容相当于字符数组**barray**的内容后跟串**s**的内容。这里**s**不仅可以是字节串或字节数组，也可以是任何元素值都合适的可迭代对象

字节串和字节数组

- 对字节串和字节数组，取元素操作的语义与str不同：
  - 取元素操作bytes1[3]得到一个int类型的值
  - 切片bytes1[3:4]得到的是一个元素的字节串
  - bytearray的情况与此类似，取元素也得到int类型的值，取包含一个元素的切片，得到只包含一个元素的bytearray
- memoryview类型是为高效使用bytes/bytearray或其他类似对象而设计的一套机制。bytes/bytearray可能保存大量信息，如巨型图像/矩阵，标准库array包用于建立整数或浮点数数组，处理中不希望建立中间拷贝。memoryview就是为解决这类问题
- 与上面情况类似，字典也需要尽可能避免拷贝。dict类的方法 dict.keys(), dict.values() 和 dict.items() 返回称为dictionary view objects的对象，它们都不是另外建立的关键码序列（或其他），得到的值称为view，能动态反映具体字典的变化

标准函数

- 标准库第2节给出了所有68个标准函数及其解释，可以大致分类
- 与对象和类型有关：type(), id(), isinstance(), issubclass(), super(), staticmethod(), classmethod()
- 具体类型有关：bool(),int(),float(),complex(),str(),range(),tuple(),list(), set(), frozenset(), dict(), bytes(), bytearray(), memoryview(), object(), callable()
- 计算：abs(), divmod(), sum(), pow(), min(), max(), round(), any(), all(), ord(), chr(), hash(), len()
- 变量和属性：globals(), locals(), vars(), setattr(), getattr(), hasattr(), delattr(), property()
- 串和IO：bin(),oct(),hex(),repr(),ascii(),format(),input(),print(),open()
- 序列和迭代：filter(),map(),enumerate(),zip(),sorted(),reversed(),next(),iter()
- 其他：help(), dir(), exec(), eval(), compile(), slice(), __import__()
- 注意：hash从每个内部对象算出一个整数值，evel把参数字符串作为Python表达式求值，exec把参数字符串作为Python代码执行（也可以是callable对象）

什么是计算

- 朴素的认识，计算的思想和实际的发展
- 图灵机
  - 基本结构
  - 图灵机可计算
  - 通用图灵机
  - 图灵论题（丘齐-图灵论题）
  - 不可计算
- 理论的和实际的计算
  - 计算复杂性
  - **P == NP ?**

从问题到程序

- 作为计算机和信息领域的基础课程，首要任务是帮助大家理解和掌握从问题出发，做出可以解决问题的程序的整个过程
  - 问题通常用非形式的语言表述，或直接来自实际。其表示往往不精确、不完全，甚至有错误，依赖于人的知识
  - 需要通过深入分析，把问题严格化。这一工作阶段出现不但在实际中，也会出现在解决初级问题的练习中
  - 有了清晰的需求后，需要设计好整个处理过程（计算过程，这是程序的设计阶段），对复杂的问题，这一步骤更加重要
  - 设法落实（设计出）实现中各主要处理工作（处理步骤）的算法，注意考虑算法的性质和情况
  - 实现程序（编码），通过试验运行确认程序能完成所需工作
  - 评价和改进（例如效率，实现方法等）

从问题到程序

- 工作中的每一步，都可能遇到多种选择（不唯一），例如：
  - 问题精确化常存在多种合理假设。只要不与实际情况冲突，沿着每种假设做下去都可能做出一个符合问题需求的程序
  - 问题分解（设计）时常有许多不同方式；解决重要步骤可能有多种可行算法，实现时存在许多不同设计，等等
- 选择的关键：
  - 考虑各种选择的优缺点，为什么有这些优缺点等
  - 评价各种可能性，根据情况做出最好的选择
- 最后还要分析和评价
  - 一些作业的程序虽然对，但有重要缺点，有很大改进余地
  - 用Python写程序相对容易，也容易写出极差的程序

编程基础

- 编程是为了实现计算（让计算机做事），解决问题
  - 用图灵机定义，太低级；用计算机的 CPU 指令也太低级
  - 绝大多数程序是用高级语言写的

    C 是一种常用的高级语言；Python 是比 C 更高级的语言，提供了许多高级的程序机制
- 常用的高级语言有许多共性：
  - 基本操作。如一批内置操作和常用函数，赋值等基本语句
  - 组合机制。用运算符和括号构造复杂的表达式，用各种控制语句（复合语句）组合出复杂的控制流结构
  - 抽象机制，把复杂描述定义为命名整体，使之能像基本操作一样使用。函数是控制的抽象机制，类是数据的抽象机制

编程基础

- 常规语言通过变量/赋值，不断修改程序状态的方式完成计算
  - 称命令式（**imperative**）语言或过程式（**procedural**）语言

    在这里编程称为命令式编程或过程式编程
  - 存在其他种类的语言和编程范式，那里思考计算的方式不同

    如函数式编程，逻辑式编程等
- 命令式语言里的计算描述有两个层次：
  - 表达式，基于基本运算和函数描述如何从已有的值算出新值
  - 语句，基于基本操作（如赋值）描述修改状态的操作和顺序

    控制语句描述在函数内部的操作的执行顺序的手段；函数调用（和返回）是高级控制手段

    异常处理机制实现更大跨度的控制转移

编程基础

- 编程的另一重要方面是数据的组织
  - 简单数值计算中问题比较简单
  - 如果涉及一批数据，特别是成组的数据，数据之间有错综复杂的联系，数据的组织问题影响重大
  - 良好的数据组织能更好地支持计算过程的（设计和）进行
- 不同编程语言都为数据组织提供了专门机制
  - **Python**组合类型用于将简单的数据组合起来

    dict是更高级的组合功能，实现关键码到值的映射，可以看作一种定义域有穷的函数
  - 类定义机制使人可以根据需要定义自己的数据类型，包括定义该类型的（实例）对象的信息表示方式，并为这种对象定义适当的操作（称为实例方法）

编程基础

- 表示和解释，输入和输出
  - 用程序能处理的形式表达解决实际问题时需要关注的信息，称为“表示”，与之相对（互逆）的是“解释”

    例：把多项式表示为表，把表解释（看作）为多项式
  - 常需要把数据送入程序，转换为某种内部表示；或者把计算得到的结果转换到某种外部形式，例如人易读的形式

    例：读入一批数据，把它们作为元素构造一个矩阵

    例：把多项式计算得到的结果用某种易读形式输出
- 在人（外部）易提供/易读/易用的信息形式（例如文本等）与计算机易处理的内部表示之间需要往复转换，可能简单或复杂
  - 从外部到程序内部是输入，从程序内部到外部是输出
  - 语言提供基本类型数据的转换功能，常需要通过编程扩充

Python 语言机制

- 期中时已经对Python的基本表达式，多数基本语句和控制语句，基本函数定义做了总结，这里不再重复
- 下半学期讨论的控制方面的机制：异常处理，生成器函数，函数的函数参数，控制语句的扩充部分，命令行参数，with语句
- 异常处理：
  - 作用：描述程序在遇到错误时的处理过程
  - 结构：try语句（包含except部分）和raise语句

    try语句可以有可选的finally和else部分
  - 理解运行时异常导致的控制转移，查找异常处理器的过程
  - 基本使用方式：用try结构包裹可能出现异常的代码

    在本try段处理，或让异常传到外围作用域或调用函数

Python 语言机制

- 生成器函数：
  - 定义具有内部状态，可能反复生成值的对象（迭代器）
  - 定义方式：在函数定义里使用yield语句
  - 执行中遇到yield语句就求出其表达式的值返回，再次调用从yield语句之后继续，遇return或函数体结束时生成器终止
  - 使用：常作为可迭代对象用在for语句或各种描述式里；也可以通过生成器的__next__()函数显式调用
- 其他控制机制，应了解基本情况，如：
  - 怎样定义允许任意多个实际参数的函数？
  - 函数调用的分拆参数
  - 循环语句的else部分，import的as部分

Python 语言机制

- 下半学期课程的重要内容是各种数据机制：字符串，表，元组，字典，“序列”，集合（set和frozenset），文件
- 字符串（str）：
  - 文字量写法（4种形式），特殊字符
  - 基本字符串操作
  - 格式化操作
- 表（list）：
  - 最常用的数据组合机制，没有特殊要求时应该用表或元组
  - 可变对象，允许任意元素，可以属于不同类型
  - 表的描述（直接描述的表），表描述式（推导式）
  - 基本操作，修改元素的操作，使用

Python语言机制

- 元组（tuple）：
  - 任意元素组合，不变对象
  - 隐式地使用在许多地方，如元组赋值
  - 包装和拆分，如需要函数返回一组值，多个值的赋值
- 字典（dict）：
  - 表示映射，从关键码到值
  - 关键码只能用不变对象，值可以任意
  - 字典的描述，字典推导式（描述式）
- 集合（set 和 frozenset）：
  - 可变（或不变）对象，元素没有顺序且唯一表示
  - 集合操作，集合描述和集合推导式（描述式）

Python语言机制

- “序列”：
  - 序列不是类型，代表一些类型的共性
  - 序列操作（不变序列操作，可变序列操作）
  - 序列都是可迭代对象，可直接用于 **for** 语句的迭代描述，或作为各种推导式（描述式）的迭代描述
- 文件（**file**）：
  - **file**对象代表程序建立起来的与外存文件联系，用于在程序里访问外存文件，使用外存里的数据
  - 使用外存文件前要先用**open**函数打开，**open**建立一个**file**对象，在程序里通过**file**对象访问文件
  - 文本文件的基本使用方法，文件使用完毕应该用**close**关闭
  - 文件不是序列，但**Python**支持在文本文件上按行迭代

Python 编程技术

- 基本编程技术大都在上半学期介绍了，包括：
  - 表达式描述（理解一个表达式确定的计算）
  - 赋值等基本语句的使用
  - 几种控制结构
  - 定义函数，建立控制抽象
  - 函数的函数参数和函数返回值（高阶函数）
  - 函数嵌套结构（内部函数），作用域规则
  - 函数和变量的作用域，**global** 和 **nonlocal**
- 后半学期讨论的新技术（不多）：
  - 复杂数据对象的构造和操作
  - 异常处理，文件处理，生成器

Python 编程技术

- 复杂数据对象的构造
  - 直接描述（“字面量”，成分是表达式，可以包含计算）
  - 描述式（推导式），以迭代的形式描述组合对象的元素
  - 通过计算逐步构造（更复杂的对象）
- 组合对象的操作
  - 不变对象（修改不变对象将导致执行中出错）
    - 检查元素，提取性质（如元素个数）
    - 基于已有对象构造新对象
  - 可变对象
    - 增加：修改对象的操作
  - 组合对象的比较

Python 编程技术

- 组合对象上的循环和有关处理
  - 若要在循环里修改与循环控制有关的组合对象，应该用**while**
  - **while**用于一般的循环，**for** 用于基本的规范的循环
  - 在 **for** 循环体里不应去修改头部引进的循环变量。**for**循环头部牵涉到循环体里修改的变动对象，计算容易出问题
- 对象拷贝，建立已有对象的复本
  - 浅拷贝（**shallow copy**）：只建立最上一层拷贝，共享元素
  - 深拷贝（**deep copy**）：一直拷贝到底，只共享基本对象
  - **Python** 所有内部定义拷贝操作都是做浅拷贝
  - 如果需要做更深的拷贝，需要自己写程序完成

Python 编程技术

- 异常处理技术
  - 捕捉和处理可能发生的异常
  - 异常处理器的排列顺序
  - 检查状态，发现异常情况时主动引发异常，要求外围处理
  - **else和finally块的使用**
- 文件处理技术
  - 打开和关闭
  - 缓冲区问题，冲刷（**flush**）
  - 文本文件处理技术
- 生成器的定义和使用

类的定义和使用

- 类定义，用于定义自己需要的类型
  - 类和对象，继承，类的实例等相关概念
  - 基本语法，通过继承定义派生类
  - 初始化和对象的创建，在类中定义的实例方法、静态方法、类方法，几种方法的意义和用途
  - 方法的继承与覆盖；类层次结构，方法查找；通过逐步扩充或修改，定义一组相关的类型
- 有关GUI和并发程序的内容是简单介绍，希望帮助大家了解基本情况和一些与计算领域最近发展有关的“常识”

  课堂上只做了简单介绍，考试中不准备涉及

课程情况

- 课程进行中情况
  - 讲授为主，课堂气氛不太活跃
  - 同学提问不多，答疑的同学也不多
  - 课下一些同学提出了很好的问题，对我也很有帮助
  - 需要老师和同学的共同努力
  - 作业提交情况较好
  - 期中考试反映出一些问题

    希望期中考得不太好的同学注意
- 本课程进行中，一些同学积极参加了课程论坛的讨论
  - 20多位同学或多或少参加了讨论
  - 有关情况会反映在课程成绩的评定中

考试的若干具体问题

- 题目类型与期中类似。不用抄题，只需注明题目号
- 有编程题，经常要求定义函数，请按题目要求完成

  定义函数可能要求写出代码说明函数的使用，这时只需写出调用函数的语句实例，说明函数的用法
- 程序尽可能写清楚
  - 格式清晰，能看清层次
  - 采用合适的名字（函数名、变量名等）
  - 必要时加注释，或在考卷上另外给出说明
- 简答题的回答应简明扼要，说明问题
  - 这种题都不需要长篇大论，只需要关键性的几句话
  - 如：A是什么，A和B有什么关系，A和B比较等

下学期课

- 下学期有一门与本课程配套的“数据结构与算法”

  孙猛老师将开设基于Python的课程。学院另开设一门基于C语言的课程，如去另一门课需要自己做些准备

- 目前国内尚无合适教材

  在Amazon有几本名字有关的书，国内网站能找到一本外文“Data Structures and Algorithms Using Python”电子版

  张乃孝老师的“算法与数据结构”作为参考书，该书基于C语言讨论相关内容，各章的基本概念部分可参考

  我基于去年的课程编写了一本教材，应该能在8月出版

- 课程准备：

  假期做一点Python程序，特别是用它去解决自己遇到的问题

  浏览一下Python文档，查阅其他有关材料（书籍/网上资料等）