



School of Economics and Management, Beihang University

现代程序设计技术

赵吉昌

jichang@buaa.edu.cn

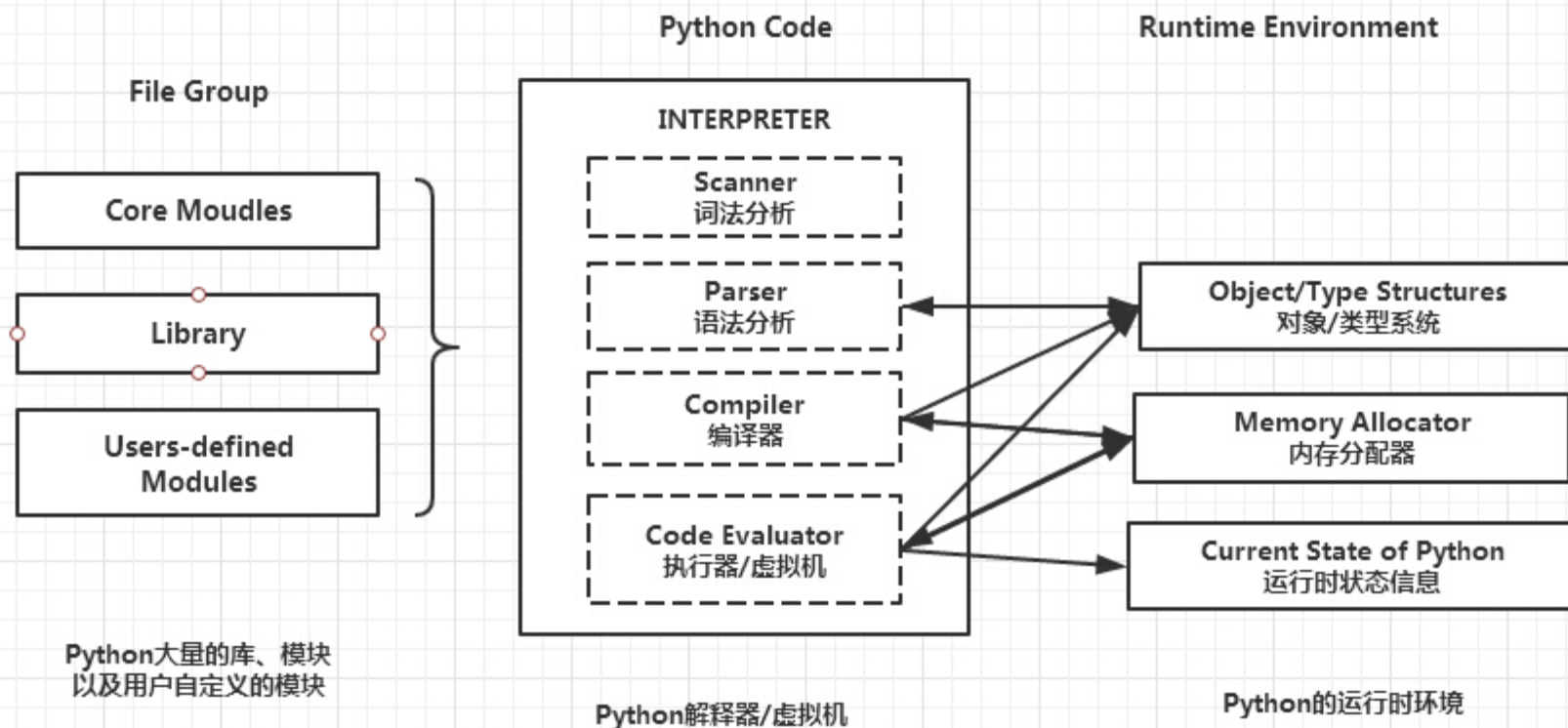
本周内容



- 考核方式
- 期终总结
- 答疑

- 平时小作业
 - 10分
- 平时大作业
 - 30分 (A)
 - 25分 (B)
 - 20分 (C)
- 闭卷考试
 - 60分
 - 判断 (约10分, 10题)
 - 选择 (约10分, 10题)
 - 编程题 (约40分, 4-5题) 不涉及算法和数据结构, 考察面向对象编程与设计, 类型和难度均在课堂演示的范围内)

• Python解释器



- Python代码格式

- 缩进

- 缩进空格数可变，但是同一个代码块的语句必须包含相同的缩进空格数

- 多语句

- 可以用；在同一行显示多条语句
 - 语句很长时可使用\来实现多行语句
 - 在 [], {}, 或 () 中的多行语句不需要使用\

- 注释

- 单行注释用#
 - 多行注释用'''或"""

- Python数据类型
 - 变量所引用的内存中对象的类型
 - 不可变类型
 - Number(int, bool, float, complex)
 - String(str)
 - Tuple(tuple)
 - 可变类型
 - List(list)
 - Set(set)
 - Dictionary(dict)
 - 类型的查询与判断
 - type, isinstance, isinstance

- Python函数
 - 没有switch-case语句
 - 不可变类型传值，如整数、字符串和元组
 - 可变类型传引用，如列表，字典，集合
 - 参数默认值只被赋值一次，当默认值是可变对象时会在后续调用中不断累积
 - 如何避免？
 - 不定长参数(*, **)
 - 嵌套函数
 - `lambda` 如何从外部作用域引用变量？
 - 闭包
 - 每次开启内函数时都在使用同一闭包变量
 - `nonlocal`

- Python模块

- 模块的模块名（字符串）可以由全局变量 `__name__` 得到

- `python module.py`时，其 `__name__` 被设置为 `"__main__"`

- 模块搜索路径为当前目录或存储在 `sys.path` 的路径

- 模块可以包含可执行语句以初始化，仅在第一次被导入的地方执行一次

- 目录必须包含 `__init__.py` 文件才能成为包
 - 该文件里一般包含哪些内容

- Python命名空间与作用域
 - 局部的命名空间->全局命名空间->内置命名空间
 - Python 中只有模块 (module) , 类 (class) 以及函数 (def、lambda) 才会引入新的作用域
 - if-elif-else, try-except, for, while 并不能引入
 - 四种作用域 : L -> E -> G -> B
 - `a = 10`
 - `def fa():`
 - `a = a + 1`
 - `print(a)`
 - `fa()`

- 面向对象程序设计
 - 一种程序设计范式
 - 程序由对象组成，每个对象包含对用户公开的特定功能和隐藏的实现部分
 - 对象是数据与相关行为的集合
 - 不必关心对象的具体实现，只要能满足用户的需求即可
 - 与结构化程序设计的差异
 - 将数据搁在第一位
 - 更加适用于规模大的问题
 - 基本特点：封装，继承，多态

- 类
 - 对象的类型，用来描述对象、构造对象的模板
 - 定义了该集合中每个对象所共有的属性和方法
 - 由类构造对象的过程称之为实例化，或创建类的实例
- 类对象
 - 类对象支持属性引用和实例化，属性引用可以是类变量，也可以是类方法
- 实例对象
 - 实例对象仅能进行属性引用（数据或方法）
- 类变量
 - 类变量在整个实例化的对象中是公用共享的，类变量定义在类中且在函数体之外，类变量通常不作为实例变量使用
- 实例变量
 - 每个实例独有，第一次使用时自动生成，与类变量重名时仍作为实例变量

- 类的方法与私有性

- `__new__` () 为构造方法，参数为将要构造的类，无 `self` 参数，返回新创建的实例对象
- `__init__` () 为初始化方法，在类实例化时会自动调用，有 `self` 参数，只能返回 `None`
- 严格来讲，python 类的属性和方法都是对外公开的
- 通过一些 “约定” 来约束外部的访问
 - 在属性或方法前加 `_` 或 `__`
 - 命名改装 (name mangling), 外部访问时需要加上 `_<类名>` 的前缀

- 类的继承
 - 基类与派生类
 - 多继承
 - `super()` 方法
 - 类的专有方法
 - `__str__`
 - `__iter__`
 - `__next__`
 - `__le__`
 - 运算符重载

- Python异常处理

- 异常捕获

- `try/except/finally`
 - 多个捕获子句

- 异常抛出

- `raise`的异常必须是一个异常实例或异常类
 - `raise`也能重新抛出异常

- 断言

- 用于判断表达式并在其`False`的时候触发`AssertionError`

- 自定义异常

- 直接或间接的从`Exception`类派生，命名多以 “Error”结尾

- 预定义清理行为 (`with...as`)

- 异常处理的原则

- 利用异常层次结构；不压制异常；早抛出、晚捕获

- 装饰器

- 通过函数实现装饰器

- 如果装饰器本身需要参数，则需要通过高阶函数实现
 - 如何保持函数名称不发生变化？

- `functools.wraps(func)`

- 通过类实现装饰器

- `__call__(self, func)`

- 装饰器的顺序

- Python抽象类

- 元类

- 控制类的创建行为，即先定义metaclass，再创建类，最后创建实例
 - 类可以看成metaclass创建的“实例”
 - 元类是类的模板，所以必须从type类型派生
 - 通过关键字参数metaclass来指定元类来创建类

- 抽象类

- 只能被继承，不能被实例
 - 从不同的类中抽取相似的行为作为抽象方法
 - 子类必须实现抽象方法
 - 借助abc模块实现（指定元类并装饰）
 - 继承/注册

- Python生成器与迭代器
 - 一边循环一边计算元素的机制
 - 通过`next()`函数获得下一个元素值
 - 通过定义函数构建生成器
 - `yield`
 - 获取返回值需要捕获`StopIteration`异常
 - 把一个类作为一个迭代器使用需要在类中实现两个方法 `__iter__()` 与 `__next__()`
 - `__iter__()` 返回一个特殊的迭代器对象，这个迭代器对象实现了 `__next__()` 方法
 - `__next__()` 方法返回下一个元素并通过 `StopIteration` 异常标识迭代的完成

- Python多进程
 - 创建多进程 (`Process`)
 - 进程的同步：互斥锁，信号量，事件，条件，队列 (`Queue`, `JoinableQueue`)
 - 进程的通信 (管道)
 - 进程的内存共享
 - 进程池
 - `ProcessPoolExecutor`

- Python多线程
 - Cpython解释器中，同一个进程下开启的多线程，同一时刻只能有一个线程执行，无法利用多核优势
 - 创建多线程 (Thread)
 - 线程同步 (锁，信号量，事件，条件，定时器，Barrier，队列)
 - 线程池 (ThreadPoolExecutor)

- Python协程
 - 异步IO
 - 协程在本质上只有一个线程
 - 协程的实现
 - `yield`, `gevent`, `asyncio`
 - 协程的优缺点
 - 无需线程上下文切换的开销，避免了无意义的调度，无需原子操作锁定及同步的开销
 - 程序员必须自己承担调度的责任，协程仅能提高IO密集型程序的效率，但对于CPU密集型程序无能为力

- Python网络编程
 - Socket：应用层与TCP/IP协议族通信的中间软件抽象层
 - TCP/UDP
- Python数据库编程
 - 关系数据库
 - 非关系数据库
 - ORM

- 设计模式
 - 单例模式
 - 工厂模式
 - 代理模式
 - 观察者模式
 - 适配器模式