Python 面试题

- python2/ python3的区别
  - python3的字符串都是Unicode
  - python3的Print 是函数, 而不是语句
  - python3中使用了range 代替xrange, 删除了xrange函数
  - python3中 的 类 (class) 都是新类型 (new style)
  - python3中支持更简单的unpack方式
- 什么是修饰器(Decorator)? 你能说出他的用途吗?
  - 装饰器本质上就是一个函数, 这个函数接收其他函数作为参数, 并将其以一个新的修改后的函数进行替换.
  - 修饰器允许你包装函数和类方法, 在执行原始代码之前和之后执行其他的代码
  - 装饰器需要注意的几点:
    - 函数的属性变化
    - 使用insperct 获取函数参数
    - 多个装饰器的调用顺序
    - 给装饰器传递参数
  - 装饰器的使用场景
    - 注入参数(提供默认参数, 生成参数)
    - 记录函数行为(日志, 缓存, 计时)
    - 预处理/后处理(配置上下文)
    - 修改调用时的上下文(线程异步或者并行,类方法)
- 你能列出列表(list)和元组(tuple)的区别吗 ? 举例子说明用法的不同。
  - 列表和元祖是最基本的两个数据类型
  - 首先, 列表的对象是可变的, 元祖不是
  - 其次, 元祖可以被哈希, 例如可以用做字典对象的键值(key)
  - 例子: 地图上的地理坐标可以用二元祖表示, 地图上的路径可以用坐标点 列表来表示
    - 一个点,至少要知道x,y轴位置,地理上就是经/纬, 如(20, 30)表示一个点;
    - [(20,30), (22, 38), (44, 20)] 这不就是三个点了, 三个点连起来不就成线了?
      [(20,30), (22, 38), (44, 20)] 这些点连成的线就是地理路径
- 你知道 range 和 xrange的区别吗?
  - range函数返回的是一个列表对象, xrange 返回的是一个 xrange 对象; 主要区别是:
    - 内存的占用不同. 列表对象已经在内存中存在了, 而 xrange对象,永远占用同样的内存大小, 无论需要生成的range有多大
    - xrange 对象不能使用列表对象的切片函数 , 也就是说 range(10)[3:5] 能工作, 而 xrange(10)[3:5]就不工作了
- 参数是如何传递的 ? 传值还是传引用 ?
  - 问题的实质是当一个传入的参数在函数体内被更改, 那么在函数返回后, 函数体外的这个参数变量的值是否改变.
    - 最简单的回答: 都不是 事实上 Python里是传递对象的 (call by object)
    - 要弄明白这个变量的值是否在函数体外发生变化,需要明白两个概念:
      - Python里的对象分为 可变和不可变对象: 数字(int, float) , 字符串 元祖(tuple) 是不可变对象, 列表, 字典表 是可变对象
      - 所有的变量都是一个对象的引用. 无论对象是否可变, 这个变量都可以赋值给另外一个对象

    - 因此: 1. 如果传入的参数变量指向一个不可变对象, 那么在函数体外的这个对象的内容永远不会发生变化;
    - 2. 如果传入的参数变量指向一个可变对象, 那么 这个对象是否发生变化, 取决于函数体内部是否改变了这个可变对象的内容

- 什么是列表和字典推导
    - 列表推导是典型的Pythonic 的代码书写方式
    - 列表/字典推导是简化列表和字典的生成
    - 列表推导的执行速度快于通常的循环
    - a = [x *2 for x in range(10)] a = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
    - a = { x: x*2 for x in range(10)} a = {0: 0, 1: 2, 2: 4, 3: 6, 4: 8, 5: 10, 6: 12, 7: 14, 8: 16, 9: 18}
    - >>> a = {x **2 for x in range(10) if x % 2 == 0} >>>a = {0, 64, 4, 36, 16}
- 如何计算列表里所有元素的和？如何计算列表里所有元素的乘积？
    - >>> s = 0 for x in range(10) s += x the basic way
    - >>> s = sum(range(10)) the right way
    - >>> from opreator import add s = reduce(add, range(10)) the other way
    - >>> s = 1 for x in range(10) s = s * x the basic way
    - >>> from opreator import mul reduce(mul, range(1, 10))
- 什么是WSGI
    - WSGI , web server gateway interface, WSGI 不是服务器,不是API, 不是Python 模块, 更不是什么框架, 而是一种服务器和客户端交互的接口规范
    - wsgi是将web server参数python化，封装为request对象传递给apllication命名的func对象并接受其传出的response参数，由于其处理了参数封装和结果解析，才有python世界web框架的泛滥，在python下，写web框架就像喝水一样简单:)
    - WSGI里的组件分为『Server』，『Middleware』和『Application』三种，其中的『Middleware』是『设计模式』里的Decorator（装饰器）。
- GET和POST，有什么区别？
    - 误区一：POST可以比GET提交更多更长的数据？
        - 实际上，URL不存在参数上限的问题，HTTP协议规范没有对URL长度进行限制。这个限制是特定的浏览器及服务器对它的限制
        - 同时，POST是没有大小限制的，HTTP协议规范也没有进行大小限制。POST数据是没有限制的，起限制作用的是服务器的处理程序的处理能力。
    - 误区二：POST比GET安全？
        - GET将提交到服务器的数据添加到URL中了，可见；
        - 虽然POST的数据，你肉眼看不到，你抓个包看看，
        - 在HTTP包的包体中，我们提交的数据时仍然可见的；
        - 所以说，从这方面来说，POST也是以五十步笑百步了。
    - 清晰概念场景
        - 你使用GET方法时，浏览器可能会缓存你的地址等信息，还会留下历史记录，而对于POST方法呢，则不会进行缓存。
        - 在开发中，一定要分清楚GET和POST的使用场合，什么时候要使用GET，什么时候要使用POST，自己做到心中有数
- python的可变与不可变数据类型 简单解释
    - 可变数据类型：列表list和字典dict；
        - 可变数据类型是允许同一对象的内容，即值可以变化，但是地址是不会变化的。
        - 但是需要注意一点，对可变数据类型的操作不能是直接进行新的赋值操作，比如说a = [1, 2, 3, 4,

5, 6, 7]，这样的操作就不是改变值了，而是新建了一个新的对象，这里的可变只是对于类似于append、+=等这种操作。

- 不可变数据类型：整型int、浮点型float、字符串型string和元组tuple。
  - 不可变数据类型的优点就是内存中不管有多少个引用，相同的对象只占用了一块内存
  - 它的缺点就是当需要对变量进行运算从而改变变量引用的对象的值时，由于是不可变的数据类型，所以必须创建新的对象，这样就会使得一次次的改变创建了一个个新的对象
  - 不过不再使用的内存会被垃圾回收器回收。
- 总结:
  - “python中的不可变数据类型，不允许变量的值发生变化，如果改变了变量的值，相当于是新建了一个对象，而对于相同的值的对象，在内存中则只有一个对象，内部会有一个引用计数来记录有多少个变量引用这个对象；
  - 可变数据类型，允许变量的值发生变化，即如果对变量进行append、+=等这种操作后，只是改变了变量的值，而不会新建一个对象，变量引用的对象的地址也不会变化，不过对于相同的值的不同对象，在内存中则会存在不同的对象，即每个对象都有自己的地址，相当于内存中对于同值的对象保存了多份，这里不存在引用计数，是实实在在的对象。”

- 语句和表达式的区别
  - 语句和表达式的区别：“表达式就是某件事”，“语句是做某件事”。
    - 19+2*4-8/2是一个表达式
    - print a 是一个语句 按照严格的说法，是打印变量a所对应的对象的值

- Python 不能不知道的日常应用模块
  - Crontab/Queue/ sched模块下的schedulerl类
    - 定时任务 crontab 管理任务很规矩，到点执行(精确度不那么高)
    - 假如是动态任务, 也就是不一定啥时候就来排上队约定一个事件等着执行
      - 这个时候选择Queue模块
    - 加上优先级策略并能取消某个指定的已经放入队列的任务
      - sched模块中的scheduler类就是一个这样的通用的事件调度类
  - 多线程和多进程之前怎么选择
    - 首先分清楚任务是CPU密集型还是IO密集型, 用下面两个方法分别试试
      - from multiprocessing import Pool
      - from multiprocessing.dummy import Pool
      - 那个速度快就用那个, 在多线程/多进程之间切换非常方便
  - 优先级别标准库
    - argparse 用来替代optparse的命令行解析库. 如果你考虑用更直观的, 推荐docopt它使用docstring所见即所得实现命令行解析
    - collections 包含了一些额外的数据类型.
      - 其中的OrderedDict(有序列的字典),
      - defaultdict(带有默认值的字典),
      - namedtuple(通过创建带有字段属性的元祖子类)
      - deque(高效实现插入和删除操作的双向列表)
    - functools 带有非常有用的工具
      - partial (偏函数)
      - wraps(将被包装函数的信息拷贝过来)
      - total_ording(只需要定义2个__xx__方法就可实现对象对比的类装饰器)
      - cmp_to_key(将老式的比较函数转化为关键字函数)

        - multiprocessing 多进程模块
        - Queue 这个模块用于多线程编程 它是一个线程安全的FIFO(先进先出)的队列实现.
            - 多进程编程选用 multiprocessing.queues中的Queue, SimpleQueue, JoinableQueue 这三个队列实现
        - threading 多线程模块
        - os 模块 例子获取环境变量
            - os.environ.get('pythonpath')
            - os.getenv('pythonpath')
        - SimpleHTTPServer 最简单地HTTP Server实现 不使用web框架,就可以运行起来的静态服务
            - python -m SimpleHTTPServer PORT

- python 为什么性能差
    - 编程语言的效率分为：
        - 1. 开发效率 就是对程序员而言, 完成编码所需要的时间
        - 2. 运行效率 对计算机而言, 完成计算任务所需要的时间
    - python更在乎的是 编码效率 从而接受 运行效率低的事实 life is short, we use python
    - Python 运算效率低 具体原因,简单说明
        - Python是动态语言
            - 一个变量所指向的对象的类型在运行时才确定, 编译器做不了任何预测, 也就无从优化
            - 属性查找 访问对象的某个属性是一个非常复杂的过程, 而且通过同一个变量访问到的Python对象还都可能不一样
        - Python 是解释执行, 但不支持JIT(just in time compiler)
        - python 中一切都是对象, 每个对象都需要维护引用计数, 增加了额外的 工作
        - PYTHON GIL
            - 因为GIL Python中的多线程并不能真正的并发,
                - 如果是在IO bound的业务场景, 这个问题并不大
                - 但是在 CPU bound的场景, 就很致命
                - 即使在单线程 GIL也会有很大的性能影响, 因为Python每执行100个opcode(默认) 就会尝试线程的切换
                - 工作中使用Python多线程的情况并不多, 一般都是使用多进程(prefork), 或者加上协程
        - 垃圾回收
            - python采用标记和分代的垃圾回收策略, 每次垃圾回收的时候都会中断正在执行的程序,造成所谓的顿卡

以上内容整理于 幕布