day21

- 可迭代对象

```
range() list tuple dict set str frozenset
元素可以一个个取出来。迭代 遍历 逐个获取
- 拥有迭代器的对象，称为 可迭代对象
    l = [10,20,30]
    iter(l) # 获取 迭代器
```

- 迭代器

```
1. 实现了 __iter__() __next__() 方法的 对象 ，称为 迭代器
2. 迭代器 本身也是一个 可迭代对象
3. 实际上 在使用 for循环时，就是利用了 迭代器，而只不过 for循环比较友好，在迭代完毕之后，则程序完成，不会以报错形式结束。
4. 迭代器 使用 next() 迭代出来元素。
        - next() 每调用一次会迭代出来一个元素，并且游标会记录当前位置
        - 直到 报错 stopIteration异常，则证明 迭代完成。
```

- 生成器

```
1. 生成器 也是 迭代器
    - 可以通过 next() 迭代出每个元素
2. 生成器 可迭代对象
    - for循环
```

- 生成器 VS. 列表

```
g1 = (i for i in range(5))
l1 = [0,1,2,3,4]
优势：
    - 生成器 会 节省内存。生成器元素如果需要，才会next()迭代出来，而不用像列表，初始创建时就已经开辟空间
    - 不需要考虑 下标 ，不需要依靠下标来取元素.
缺点：
    - 不能灵活的获取某个元素。第三个元素 list[2]
    - 没有下标 有可能会造成 操作时 不便捷
```

- 生成器创建

1. 使用 推导式方式
    - (变量 for 变量 in 可迭代对象 if 布尔表达式)
        - 流程一：for循环 迭代出 每个元素 变量
        - 流程二： 针对 if 表达式 进一步 筛选
        - 流程三：最终 形成 生成器的 元素
2. 函数 + yield
    - yield: 关键字 意义：生产，退出 退让
    - 功能：
        - 返回函数 停止 类似return, 需要注意： 暂停
        - 可以 传递给 外界 数据
        - 可以 接收 外界 发送过来的数据 使用send()
    - 激活 使用 生成器，可以 借助 next(),send()

- 协程

微线程
进程：
线程：轻量级进程 进程下 基本 资源分配 任务执行 基本单元
协程： 比 线程单位 更细化
    - 是 为 非抢占式多任务执行计算机程序的功能组件。

vs. 线程：
    - 避免了 多线程工作时，上锁问题 数据脏读
    - 多个协程 存在于 一个线程中。
    - 避免了 多线程工作时 线程之间 切换，从而节省系统资源开销，并且 速度也会有所提升
    - 执行速度快，相较于线程，更轻量。

缺点：

- 一旦当前程序发生阻塞，整个程序都会阻塞。

使用yield完成