# 基本概念

面向对象编程的作业

1. 面向过程编程与面向对象编程的区别与应用场景？

面向过程编程强调将问题分解为一系列步骤或函数的顺序执行，适用于简单的、线性的问题，如脚本编程或数值计算领域。它关注算法和步骤。

面向对象编程将问题抽象为对象的集合，通过对象之间的交互来解决问题。它适用于复杂的、交互复杂的问题，如大型软件开发或图形用户界面开发。它具有更好的可维护性、可扩展性和代码重用性。

1. 初始化函数、析构函数与普通函数的区别？

初始化函数：初始化函数是在创建对象时自动调用的特殊方法。它用于初始化对象的属性和设置对象的初始状态。初始化函数在对象创建后立即执行，通常用于执行必要的设置或准备工作。

析构函数：析构函数是在对象被销毁时自动调用的特殊方法。它用于清理对象所占用的资源和执行必要的收尾工作。析构函数在对象销毁前被调用，通常用于释放内存、关闭文件或断开连接等操

作。

普通函数：普通函数是一般的函数，不属于特殊的初始化函数或析构函数。它可以在任何时候被调用，接收参数并执行特定的任务或操作。普通函数通常用于执行一系列操作、完成特定的任务或返回结果。

1. 绑定对象的函数、绑定类的函数、静态函数的区别？

绑定对象的函数：绑定对象的函数是类中定义的普通函数，可以访问和操作对象的属性。它们通过对象进行调用，自动将对象本身作为第一个参数传递给函数（通常命名为self）。绑定对象的函数可以访问和修改对象的状态，并且可以使用对象的属性和方法。

绑定类的函数：绑定类的函数是类中定义的普通函数，无需访问或操作特定对象的属性。它们通过类进行调用，自动将类本身作为第一个参数传递给函数（通常命名为cls）。绑定类的函数通常用于定义类级别的操作或处理类级别的属性。

静态函数：静态函数是类中定义的特殊函数，与特定对象和类无关。它们不会自动传递任何对象或类参数，因此没有self或cls参数。静态函数可用于定义与类相关但不需要访问实例或类属性的独立功能。

1. 类的属性和对象的属性有什么区别？

类的属性：类的属性是定义在类中的变量，被该类的所有对象所共享，可以通过类名直接访问，也可以通过类的任意对象访问。当类的属性被修改时，会影响到所有使用该类的对象。

对象的属性：对象的属性是定义在对象中的变量，每个对象都有自己的属性副本，只能通过对象访问，无法通过类名直接访问。每个对象的属性可以独立地被修改，不会影响其他对象的属性。

1. 类和对象的属性与方法在内存中是如何保存的（ dict ）。

在Python中，类和对象的属性与方法在内存中是通过字典（dict）来保存的。每个类和对象都有一个字典，用于存储其属性和方法。

1. 描述继承、派生和组合的基本概念、适用场景，及典型用法示例。

\*\*继承（Inheritance）\*\*是面向对象编程中的一种机制，通过创建一个新类（称为子类或派生类），从一个现有类（称为父类或基类）继承属性和方法。子类可以继承父类的特征，并可以在其基础上添加新的特征或修改已有特征。继承可以实现代码的重用和层次化组织。适用场景：当存在一个通用类，而需要创建一个更具体的类时，可以使用继承来扩展现有类的功能。当多个类之间存在共同的属性和方法时，可以将这些共同的部分提取到一个父类，然后让子类继承父类。

\*派生（Derivation）\*\*是继承的过程，通过创建子类来派生或继承父类的属性和方法。派生是继承的一种具体操作，用于创建子类。

适用场景：

当存在一个通用类，而需要创建一个更具体、特定功能的类时，可以派生一个子类。子类可以继承父类的通用属性和方法，并在此基础上添加新的特征或修改已有特征，从而实现特化的功能。通过派生子类，可以重用父类的代码，避免重复编写相似的功能。子类可以继承父类的属性和方法，从而减少代码冗余，并提高代码的可维护性和可扩展性。

\*\*组合（Composition）\*\*是将多个类组合在一起以创建一个更大的类或对象。组合关系表示一个类包含其他类的对象作为其成员变量，以实现更复杂的功能。组合关系强调类之间的包含关系，一个类的对象包含了其他类的对象。适用场景：当需要创建一个更复杂的对象，该对象由多个较小的对象组成时，可以使用组合。当一个类需要使用另一个类的功能，但不符合继承关系时，可以使用组合。

1. 描述抽象和多态的基本概念及适用场景，python “鸭子类型”的含义，及典型用法。

\*\*抽象（Abstraction）\*\*是面向对象编程中的概念，指的是将复杂的现实世界问题简化为适合于程序设计的模型。抽象通过定义类的接口（属性和方法）来隐藏实现的细节，只暴露必要的信息给外部使用者。抽象类和接口是实现抽象的常见方式。适用场景：当需要将问题领域的概念和行为抽象为类的形式时，可以使用抽象。当需要隐藏类的具体实现细节，只提供必要的接口给外部使用者时，可以使用抽象。\*\*多态（Polymorphism）\*\*是面向对象编程的一个特性，指的是相同的接口可以有多种不同的实现方式。多态使得可以使用统一的接口来处理不同类型的对象，提高了代码的灵活性和可扩展性。适用场景：当存在多个具有相同接口的类，但实现方式不同，并且需要统一处理它们时，可以使用多态。当需要编写通用算法或方法能够处理不同类型的对象时，多态可以简化代码逻辑。

\*\*鸭子类型（Duck Typing）\*\*是Python中一种动态类型系统的概念，根据对象的行为（方法和属性），而不是对象的类型来确定对象的适用性。鸭子类型的含义是：“如果它走起路来像鸭子，叫起来也像鸭子，那么它就是鸭子。”在Python中，鸭子类型意味着不关心对象的具体类型，只关心对象是否具有特定的方法和属性。如果一个对象具有与所需方法和属性一致的行为，即使它的类型与所需类型不同，也可以被视为适用的对象。

1. 描述封装和接口的基本概念，封装一共分为几种类型，各自的适用场景，及典型用法示例。

封装是将数据和操作封装在一个单元（类）中，通过限制对数据的直接访问，提供公共的方法（接口）来间接访问和操作数据。封装将数据和操作进行了隐藏，只暴露必要的接口，从而实现了信息的隐藏和保护。封装可以提高代码的安全性、可维护性和可扩展性。封装可以分为以下几种类型：私有封装：将属性或方法标记为私有，只能在类的内部访问，外部无法直接访问。可以通过提供公共的方法来间接访问私有成员。保护封装：将属性或方法标记为受保护的，只能在类的内部和子类中访问，外部无法直接访问。同样可以通过公共的方法来间接访问受保护成员。公共封装：没有对属性或方法进行特殊的封装限制，可以在类的内部和外部任意访问。

适用场景：当需要对数据进行保护或隐藏时，封装可以提供安全性和信息隐藏。当需要控制对数据的访问和操作时，封装可以通过定义公共接口来限制直接访问，提供更好的数据管理和操作。

# 编程操作（必做）

1. 用面向对象理念，重构“进击的小鸟”
2. 面向对象编程的基本练习：
   * 编写一个课程类(Course)
     + 包含属性：名称(name, string)，教师列表（teachers, list）， 学生列表（students，

list)，成绩（scores，dict）

* + - 主要方法：添加/删除学生， add\_student/drop\_student，添加/删除教师， add\_teacher/drop\_teacher，添加/查找某位同学的分数：set\_score/get\_score
  + 编写一个学生类(Student)，
    - 包含属性：姓名(name , string)，性别(sex, int) ，学号(student\_id , int)，已选修课程(courses，列表)
    - 主要方法： 选修/退出某课程 select\_course/ drop\_course，查看自己某个课

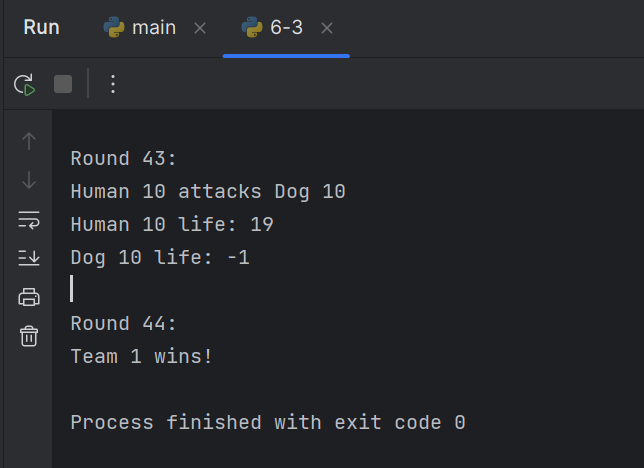
程的成绩, get\_score

* + 编写一个教师类(Teacher)：
    - 包含属性：姓名(name , string)，性别(sex , int) ，拥有的课程(courses，列表)
    - 主要方法：选修/退出某课程 select\_course/ drop\_course，给某个课程和某个课程打成绩, set\_score。
  + 完成如下操作：
    - 定义 10 个学生，3 个教师、5 门课程。
    - 某一名学生选择某一门课程
    - 某一名教师选择某一门课程
    - 教师打分。
    - 学生查询分数。

一張含有 文字, 螢幕擷取畫面, 軟體, 多媒體軟體 的圖片

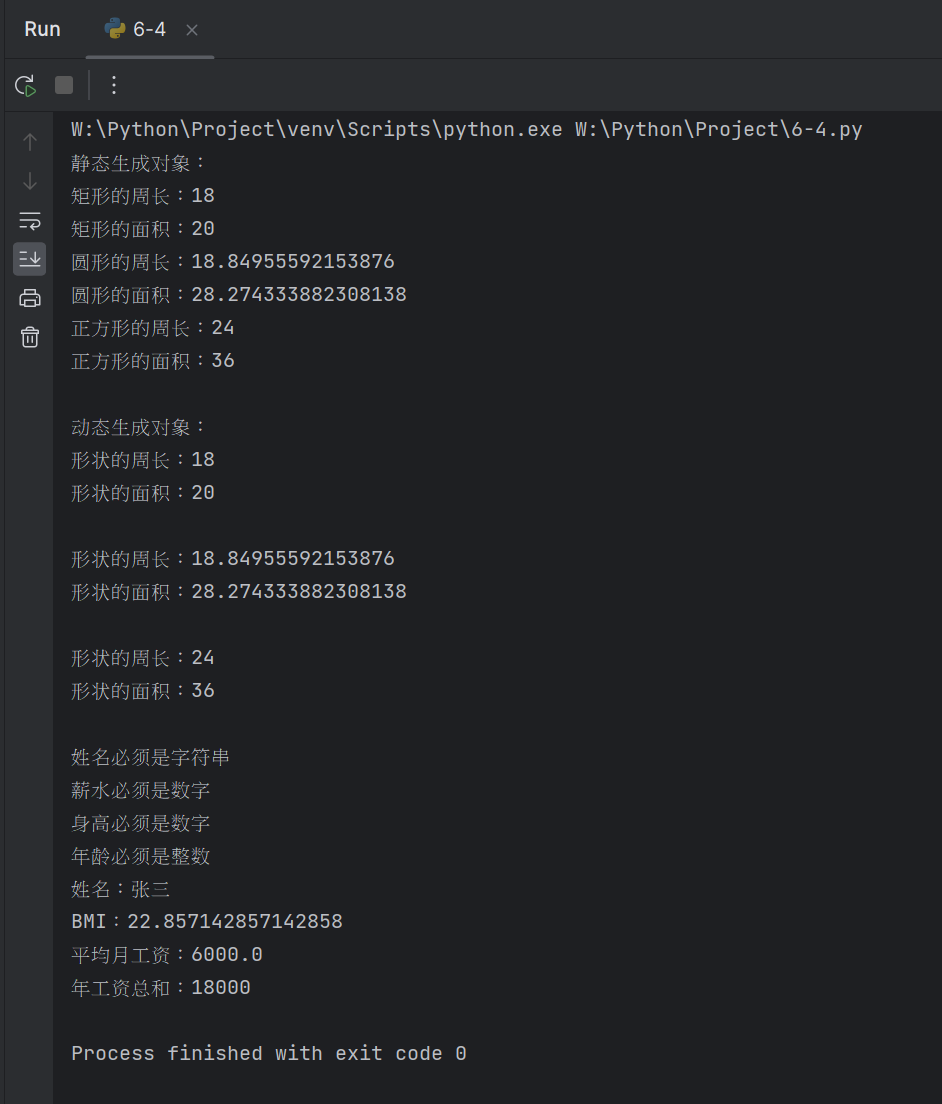
自動產生的描述

1. 继承的练习：
   * 定一个动物类 Animal，再定义两个子类：Human、Dog
     + Animal，属性： id，name, life, power, weight, pos， 方法： move(self, pos)
     + Human，新增加属性： father, mother，新增加方法: shout(self, info)， attack(self, dog)
     + Dog， 新增加属性：price, 新增加方法: bark(self, info) ， bite(self, human)
   * 生成两个队列， 各有 10 名成员，随机碰撞，碰到则互相进攻，血条降为 0， 则死亡，最终剩余的一队，宣布胜利。

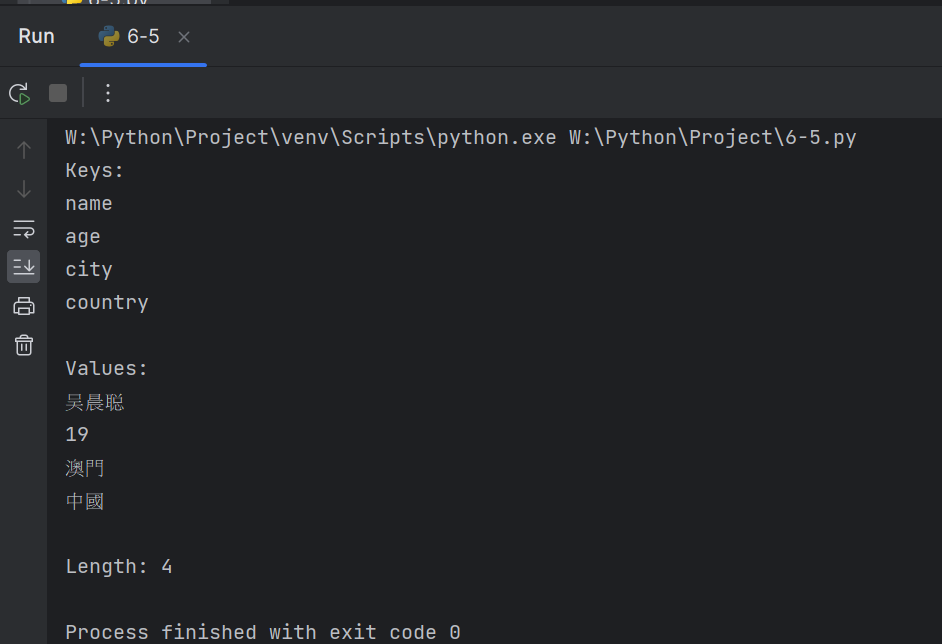


1. 多态的练习：几何形状
   * 定义 3 个类： Rect, Circle, Square
   * 实现两个函数功能的多态化：
     + 求解周长：perimeter()
     + 求解面积：area()
   * 两种方式来生成对象：静态、动态。

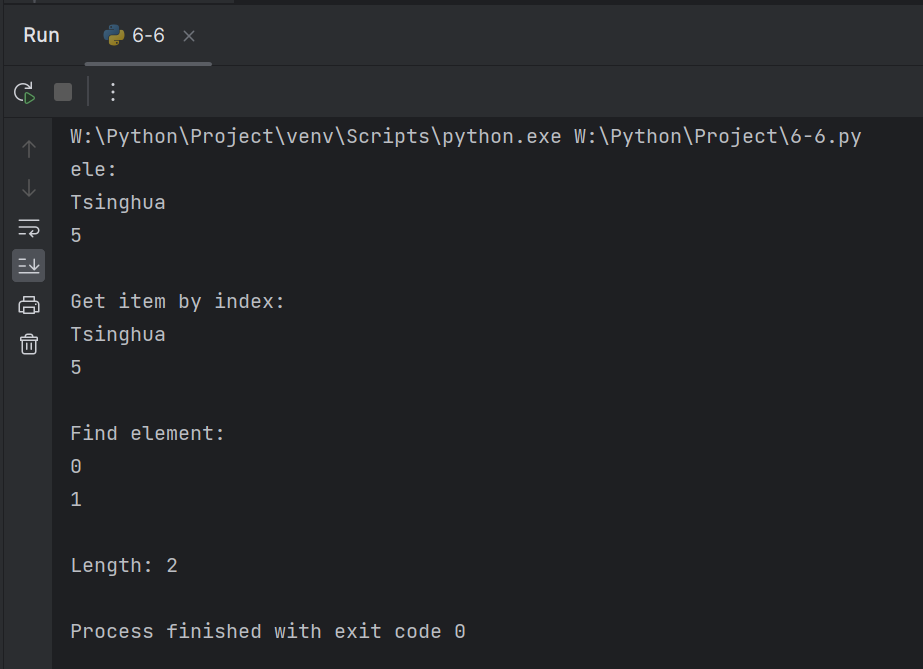
* 封装的练习：
  + 定一个类（Person)：
    - 把年龄和体重属性隐藏起来
    - 通过 property，确保 name 不允许非字符串式赋值。。。
    - 通过 property，确保输入每个月的 salary 值不允许非数字赋值
    - 通过 property，确保 height 不允许非数字赋值
    - 通过 property，确保 age 不允许非整形数字赋值
    - 通过 property，返回人体的 BMI 指标（体重千克数除以身高米数的平方得出的数字）
    - 通过 property，返回当年的平均月工资、年工资总和。



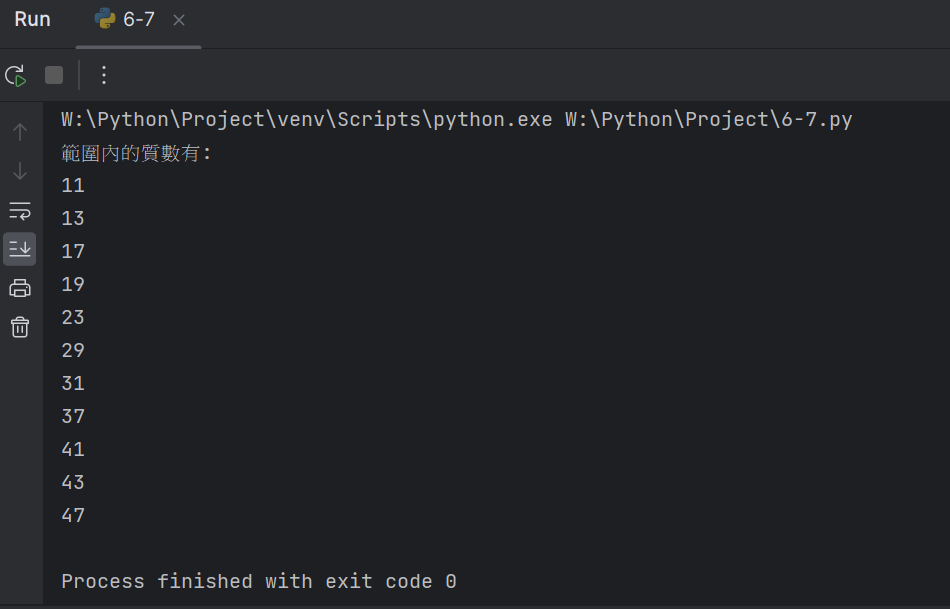
1. 自己封装 1 个 MyDict，支持以下操作：
   * 支持 for … in
   * [] 操作， 访问 key-value 值，类似于：dic[key] = value
   * 通过.操作，访问 key-value 值，类似于：dic.key = value
   * 要求: key 只能是字符串（字符串的字符构成只能是下划线、数字或字母）
   * 支持 get 操作， 输入 key， 返回 value…
   * 支持 len() 操作



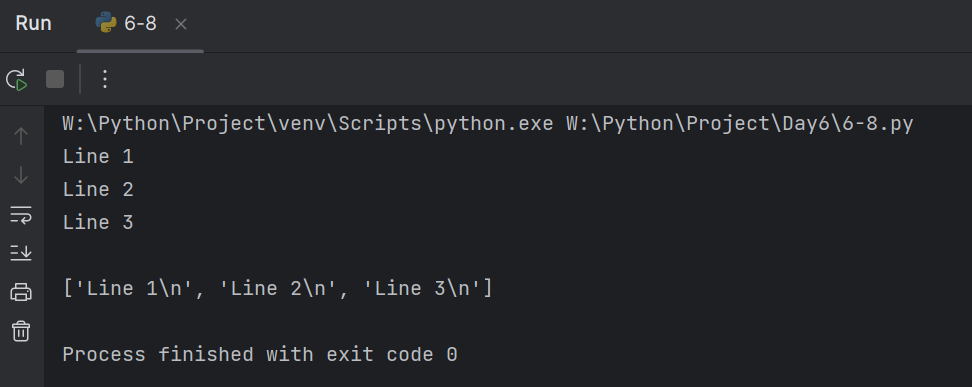
1. 自己封装 1 个 MyList，支持以下操作：
   * 支持 for … in
   * [] 操作， 输入下标，返回下标对应的结果，比如：lst[1] = value
   * 元素只支持字符串、数字，不支持其他类型的变量。
   * 支持 find()操作
   * 支持 len() 操作



1. 自己封装 1 个素数发生器 MyPrimeGenerator 的类：
   * 支持设置最小值、最大值
   * 支持 for … in



1. 自己封装 1 个文件接口类 File：
   * 支持 with 语法
   * 支持 open、close 操作
   * 支持 read、readline、readlines 等操作
   * 支持 write、writeline、writelines 等操作
   * 支持几种典型的异常操作。



# 编程操作（选做）

1. 定义 MyBTree、MyBNode 类
   * MyBTree 要求：
     + 成员变量： root, 含义：根节点
     + 用 property 去封装/访问 私有变量
     + 成员函数：添加节点：add\_node(node)、查找节点：find\_node(node, style = “wfs”)
   * MyBNode 要求：
     + 成员变量：name(名称), value(值), parent(父节点)、 left(左节点)、 right(右节点)
     + 用 property 去封装/访问 私有变量
     + name 只是能字符串, value 只能是整形或浮点型数。
     + 成员函数：add\_node(node)、find\_node(node, style = ‘wfs’) ，封装函数：
       - 广度优先搜索： find\_node\_wfs(node)
       - 深度优先搜索： find\_node\_dfs(node)
       - 利用 2 叉树特性： find\_node\_btree(node)