软件构造2

P61 T5

模块化的基本原则是什么？如何评价程序的模块化?

- **模块化的基本原则：**

1. 单一职责原则：每个模块应只负责一个明确的功能或任务，避免功能耦合。
2. 开闭原则：模块应 “对扩展开放，对修改关闭”。通过抽象接口或继承机制，允许新增功能时无需修改原有代码。
3. 依赖倒置原则：高层模块不应依赖低层模块，两者都应依赖抽象；抽象不应依赖细节，细节应依赖抽象。
4. 接口隔离原则：模块提供的接口应最小化，避免客户端依赖不需要的接口。
5. 高内聚，低耦合：模块内部组件（函数、类）应紧密相关，共同完成单一功能；模块之间的依赖应尽可能少且松散，通过明确定义的接口交互。
6. 模块化设计的可复用性：模块应设计为可独立复用的单元，通过参数化或配置化适应不同场景。

- **如何评价程序的模块化：**

1. 模块独立性：内聚度（模块内部功能的关联性（如功能内聚、顺序内聚、偶然内聚等，功能内聚最优））；耦合度（模块间依赖的紧密程度（如数据耦合、控制耦合、内容耦合等，数据耦合最优））
2. 可维护性：修改一个模块时，是否无需改动其他模块，或改动范围最小。
3. 可扩展性：新增功能时，是否可通过新增模块或扩展现有模块实现，而非重构整个系统。
4. 可测试性：模块是否可独立进行单元测试，无需依赖其他模块的具体实现。
5. 可读性与可理解性：模块的功能边界是否清晰，命名是否直观，是否符合 “自文档化” 原则。
6. 复用率：模块是否被多个场景复用，减少重复开发。

P61 T16

如何改变案例程序，使其可以产生任意整数数值范围、任意个二元运算的习题?实现程序,并测试:①产生 50 道[0.200]的加法或减法的二元运算;②产生 50 道[-100,100]的加法或减法的二元运算。

1. **操作数生成模块**

   负责生成指定范围内的随机整数

```
```

```python
def generate_operand(min_value, max_value):
    return random.randint(min_value, max_value)
```

2. **运算符生成模块**

   生成随机的二元运算符

代码块

```python
def generate_operator(operators=['+', '-']):
    return random.choice(operators)
```

3. **算式处理模块**

   封装算式的结构、计算逻辑和字符串表示

代码块

```python
class MathEquation:
    """表示一个二元运算算式及其结果"""
    def __init__(self, left_operand, operator, right_operand):
        self.left = left_operand
        self.operator = operator
        self.right = right_operand
        self.result = self.calculate()
    def calculate(self):
        """计算算式结果"""
        if self.operator == '+':
            return self.left + self.right
        elif self.operator == '-':
            return self.left - self.right
        else:
            raise ValueError(f"不支持的运算符: {self.operator}")
    def __str__(self):
        """返回算式的字符串表示形式"""
        return f"{self.left}{self.operator}{self.right}={self.result}"
    def __eq__(self, other):
        """用于判断两个算式是否相同（去重）"""
        if not isinstance(other, MathEquation):
            return False
        return (self.left == other.left and
                self.operator == other.operator and
                self.right == other.right)
```

4. **习题集生成模块**

按要求批量生成不重复的习题

代码块

```
def generate_exercise_set(count, min_val, max_val, operators=['+', '-']):
    exercises = []
    while len(exercises) < count:
        # 生成操作数和运算符
        left = generate_operand(min_val, max_val)
        op = generate_operator(operators)
        right = generate_operand(min_val, max_val)
        # 创建算式对象
        equation = MathEquation(left, op, right)
        # 确保习题不重复
        if equation not in exercises:
            exercises.append(equation)
    return exercises
```

5. **输出展示模块**

格式化显示习题集，提升可读性

代码块

```
def display_exercises(exercises, columns=5):
    for i, exercise in enumerate(exercises, 1):
        print(f"{exercise}", end="\t")
        if i % columns == 0:
            print()
    print()
```

6. **测试模块**

验证各模块协同工作的正确性

代码块

```
def test():
    # 测试①：50道[0, 200]的加法或减法
    print("1")
    exercises1 = generate_exercise_set(50, 0, 200)
    display_exercises(exercises1)

    # 测试②：50道[-100, 100]的加法或减法
    print("2")
    exercises2 = generate_exercise_set(50, -100, 100)
    display_exercises(exercises2)
```