第 6 周 Python 代码组织（初级）

任务目标

上周“对象类型”主要是 Python **表达式** (expression) 方面的概念，而本周“代码组织”则主要是 Python **语句** (statement)、**模块** (module) 和 **软件包** (package) 等方面的概念。和写文章相似，最基础的固然是一个个词汇，但划分段落、谋篇布局也是很重要的。编写程序和写文章虽然有很多相似的地方（比如都可以虚构，都需要组织），但有个重要的区别是：因为编程追求自动化，所以编程有个基本原则是“代码不要重复” (Don't Repeat Yourself, DRY)。这就要求我们在代码编写中通过 *抽象* (abstraction) 和 *复用* (reuse) 来减少重复的逻辑或数据，提高代码的可维护性。同理，我们安装和利用第三方软件包 (PyPI)，也是为了“不要重新发明轮子” (Don't Reinvent the Wheel)。

Fork 第 06 周打卡 仓库至你的名下，然后将你名下的这个仓库 Clone 到你的本地计算机

用 VS Code 打开项目目录，新建一个 environment.yml 文件，指定安装 Python 3.12，然后运行 conda env create 命令创建 Conda 环境

创建一个 guessing_game.py 文件，复制粘贴以下代码，运用 pdb 调试器理解其运行流程：

```
import random


def guessing_game():
    # 生成 1 到 100 之间的随机整数
    secret_number = random.randint(1, 100)
    n = 0

    print("欢迎来到猜数字游戏！我已经想好了一个 1 到 100 之间的数字，你可以开始猜啦。")

    while True:
        n += 1
        # 获取玩家输入
        guess = input(f"(第 {n} 次尝试) 请输入你猜的数字 (输入整数，或者输入 q 回车退出): ")
        guess = guess.strip()  # 去除多余空白字符

        if guess == "q":
            break

        try:
            guess = int(guess)
        except ValueError:
            print("输入无效🙅，请输入一个整数。")
            continue
```

```
            if guess < 1 or guess > 100:
                print("输入无效🙅，输入值应该在 1 ~ 100 之间。")
                continue

            if guess == secret_number:
                print("恭喜你🎉，猜对了！")
                break

            if guess < secret_number:
                print("猜的数字太小了，再试试⤴️。")
                continue

            if guess > secret_number:
                print("猜的数字太大了，再试试⤵️。")
                continue

            raise NotImplementedError

        print("游戏结束，再见👋。")


if __name__ == "__main__":
    guessing_game()
```

创建一个 flow_controls.py 文件，让豆包（或 DeepSeek 等任何大模型）生成例子，尝试运行，体会理解以下 Python 流程控制语句：

for 迭代循环（iteration loop）
while 条件循环（conditional loop）
break 打断跳出循环
continue 跳至下一轮循环
for...else 循环未被打断的处理
if 条件分支
if...elif[...elif] 多重条件分支
if...else 未满足条件的处理
try...except[...except...else...finally] 捕捉异常的处理
raise 主动抛出异常

```
# ==============================
# 流程控制与异常处理示例
# ==============================

# ====== for 迭代循环 ======
print("1. for 循环示例：遍历列表")
fruits = ["apple", "banana", "cherry"]
```

```
for fruit in fruits:
    print(fruit)

# ====== while 条件循环 ======
print("\n2. while 循环示例：计数器")
count = 0
while count < 3:
    print(count)
    count += 1

# ====== break 打断跳出循环 ======
print("\n3. break 示例：提前退出循环")
for number in range(1, 6):
    if number == 4:
        break
    print(number)

# ====== continue 跳至下一轮循环 ======
print("\n4. continue 示例：跳过偶数")
for number in range(1, 6):
    if number % 2 == 0:
        continue
    print(number)

# ====== for...else 结构 ======
print("\n5. for...else 示例：未被打断的情况")
numbers = [1, 2, 3]
for num in numbers:
    print(num)
else:
    print("循环正常结束")

# ====== if 条件分支 ======
print("\n6. if 示例：判断年龄是否成年")
age = 18
if age >= 18:
    print("成年人")

# ====== if...elif[...elif] 多重条件分支 ======
print("\n7. if...elif 示例：成绩等级判断")
score = 85
if score >= 90:
    print("A")
elif score >= 80:
```

```
        print("B")
elif score >= 70:
        print("C")
else:
        print("D")

# ====== if...else 示例 ======
print("\n8. if...else 示例：奇偶性判断")
number = 7
if number % 2 == 0:
        print("偶数")
else:
        print("奇数")

# ====== try...except...else...finally 异常处理 ======
print("\n9. try...except 示例：除以零错误")
try:
        result = 10 / 0
except ZeroDivisionError as e:
        print("捕获异常：不能除以零")
else:
        print("计算结果为：", result)
finally:
        print("无论是否出错都会执行")

# ====== raise 主动抛出异常 ======
print("\n10. raise 示例：主动抛出异常")
def check_age(age):
        if age < 0:
                raise ValueError("年龄不能为负数")
        print("年龄合法")

try:
        check_age(-5)
except ValueError as ve:
        print("ValueError 捕获：", ve)
```

创建一个 mylib.py 模块 (module)，在里面定义以下函数，再创建一个 myjob.py 脚本 (script)，从 mylib.py 导入函数并尝试调用：

定义函数 func1，没有形参，没有返回值
定义函数 func2，没有形参，有返回值
定义函数 func3，只有一个 位置形参 (positional parameter)，先尝试传入 位置实参 (positional argument) 调用，再尝试传入 命名实参 (named argument) 调用，再尝试不传实参 (会报错)

定义函数 func4，只有一个 命名形参 (named parameter)，先传入 位置实参 调用，再传入命名实参 调用，再尝试不传实参 (取默认值)
定义函数 func5, 接受多个位置形参和命名形参, 尝试以位置/命名各种不同方式传入实参, 注意位置参数必须排在命名参数之前
定义函数 func6，在形参列表中使用 / 来限定只接受位置实参的形参
定义函数 func7，在形参列表中使用 * 来限定只接受命名实参的形参
定义函数 func8，在位置形参的最后，在形参名称前使用 * 允许传入任意数量的位置实参 (被打包为元组)
定义函数 func9，在命名形参的最后，在形参名称前使用 ** 允许传入任意数量的命名实参 (被打包为字典)
定义函数 func10，接受两个位置形参，一个命名形参，尝试在调用时使用 * 将可迭代对象 (如元组或列表) 自动解包，按位置实参传入
定义函数 func11，接受一个位置形参，两个命名形参，尝试在调用时使用 ** 将映射对象 (如字典) 自动解包，按命名实参传入
定义函数 func12, 给函数添加 内嵌文档 (docstring), 给形参和返回值添加 类型注解 (type annotation)，提高函数签名的可读性

创建模块 `mylib.py`

首先，我们需要创建一个名为 `mylib.py` 的文件，并在其中定义你所提到的所有函数。以下是如何实现这些要求的代码：

```
# mylib.py

def func1():
    """func1 不接受参数，不返回任何值"""
    print("This is func1")

def func2():
    """func2 不接受参数，返回一个字符串"""
    return "Hello from func2"

def func3(pos_param):
    """func3 接受一个位置参数"""
    print(f"Received parameter: {pos_param}")

def func4(named_param="Default"):
    """func4 接受一个命名参数，默认值为 'Default'"""
    print(f"Received named parameter: {named_param}")

def func5(*args, **kwargs):
    """func5 接受多个位置参数和命名参数"""
    print(f"Positional arguments: {args}")
    print(f"Keyword arguments: {kwargs}")
```

```
def func6(a, b, /):
    """func6 使用 / 限定形参只能以位置实参传入"""
    print(f"a: {a}, b: {b}")

def func7(*, c, d):
    """func7 使用 * 限定形参只能以命名实参传入"""
    print(f"c: {c}, d: {d}")

def func8(*args):
    """func8 允许传入任意数量的位置实参"""
    print(f"Args: {args}")

def func9(**kwargs):
    """func9 允许传入任意数量的命名实参"""
    print(f"Kwargs: {kwargs}")

def func10(x, y, z=0):
    """
    func10 接受两个位置参数和一个可选的命名参数
    尝试使用 * 来解包可迭代对象
    """
    print(f"x: {x}, y: {y}, z: {z}")

def func11(a, *, b=0, c=0):
    """
    func11 接受一个位置参数和两个命名参数
    尝试使用 ** 来解包字典
    """
    print(f"a: {a}, b: {b}, c: {c}")

def func12(x: int, y: int) -> int:
    """
    func12 是一个带有类型注解的函数，
    它接收两个整数并返回它们的和。
    """
    return x + y
```

创建脚本 `myjob.py`

接下来，我们创建一个名为 `myjob.py` 的文件来导入并调用 `mylib.py` 中定义的函数。

```
# myjob.py

from mylib import *
```

```
# 调用 func1 和 func2
func1()
print(func2())

# 调用 func3
func3(10)  # 使用位置实参
func3(pos_param=20)  # 使用命名实参
try:
    func3()  # 应该会抛出 TypeError 异常
except TypeError as e:
    print(e)

# 调用 func4
func4()  # 使用默认值
func4(30)  # 使用位置实参
func4(named_param=40)  # 使用命名实参

# 调用 func5
func5(1, 2, 3, key1='a', key2='b')

# 调用 func6
func6(1, 2)  # 正确调用
try:
    func6(a=1, b=2)  # 应该会抛出 TypeError 异常
except TypeError as e:
    print(e)

# 调用 func7
func7(c=1, d=2)
try:
    func7(1, 2)  # 应该会抛出 TypeError 异常
except TypeError as e:
    print(e)

# 调用 func8
func8(1, 2, 3)

# 调用 func9
func9(name="Alice", age=30)

# 调用 func10
t = (5, 6)
func10(*t, z=7)
```

```
# 调用 func11
d = {'b': 2, 'c': 3}
func11(1, **d)

# 调用 func12
    print(func12(5, 3))
```

把 mylib 模块转变为 软件包 (package) 安装进当前的 Conda 环境来使用

把 myjob.py 脚本移动至 scripts/myjob.py，再次尝试运行，会发现 import mylib 失败，这是由于 mylib 并没有打包成 软件包 (package) 安装
将 mylib.py 模块移动至 src/mypkg/mylib.py，创建 src/mypkg/__init__.py 文件，准备好软件包的源代码
创建 pyproject.toml 配置文件，按照 文档 填写基本的软件包信息
在 pyproject.toml 配置文件里，按照 文档 填写软件包的 构建 (build) 配置
使用 pip install -e . 以本地可编辑模式把当前软件包安装进当前 Conda 环境
修改 environment.yml 文件，使得 conda env create 自动安装本地可编辑软件包