

第六周

一、老师笔记

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)。

1. 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
2. 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
3. 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, 再见手。")
```

```
if __name__ == "__main__":
```

```
guessing_game()
```

4. 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

- o `try...except[...except...else...finally]` 捕捉异常的处理
- o `raise` 主动抛出异常

5. 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:

- o 定义函数 `func1`, 没有形参, 没有返回值
- o 定义函数 `func2`, 没有形参, 有返回值
- o 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)
 - o 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)
 - o 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
 - o 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
 - o 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
 - o 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)
 - o 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`，接受一个命名形参，两个命名形参，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

6. 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

二、我的笔记

	循环	<ul style="list-style-type: none"> ○ <code>for</code> 迭代循环 (iteration loop) ○ <code>while</code> 条件循环 (conditional loop) ○ <code>break</code> 打断跳出循环 ○ <code>continue</code> 跳至下一轮循环
--	----	--

		进入 While 循环后，遇到 break，则直接打断，跳出该循环，进行该循环下面的代码 Continue 是停止，然后开始本循环。
For 循环		
	For a in b	<pre>fruits = ["apple", "banana", "cherry"] for fruit in fruits: print(fruit)</pre> <p>列表 【】：可迭代变量 In 后面是一个表达式，它能够返回一个可迭代的对象，那么就会向他要下一个。取来了下一个以后，每取来一个就叫一个 fruit 结果就是 “apple\banana\cherry” 随机出一个</p>
		<pre>person = {"name": "John", "age": 30, "city": "New York"} for key, value in person.items(): print(f"{key}: {value}") person = {"name": "John", "age": 30, "city": "New York"} for key in person.keys(): print(key) person = {"name": "John", "age": 30, "city": "New York"} for value in person.values(): print(f"{key}: {value}")</pre> <p>分别是：键值对、键、值 要求： In 后面只要可迭代就行</p>
while 条件循环 (conditional loop)		
		<pre>count = 0 while count < 5: print(count) count = count + 1</pre> <p>While 后面可以跟任何对象，并判断 true 或 false，只有 true 才能往下走，false 就直接进行其他的了</p>
<p>break 打断跳出循环 continue 跳至下一轮循环</p> <p>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</p>		
		可以找豆包，问例子，自学学
		python while 循环 例子
		python 用 try 和 raise 配合流程控制，举例子

cd /D/biancheng/4.8jiaozuoye/week06

conda activate week06

code .

	<p>创建两个写代码的文件：(mylib 作为模块直接写代码不规范，规范版在后面呢)</p> <p>在 mylib 里写代码，在 Myjob 里调用 mylib 里的代码 (def): import mylib</p> <pre> ! environment.yml U guessing_game.py U mylib.py U myjob.py U X + myjob.py 1 import mylib # noqa: F401 2 3 mylib.func1() 4 </pre>
	<ul style="list-style-type: none"> 定义函数 func1，没有形参，没有返回值 <pre> def func1(): x = 50 y = x**0.5 - 7 print(y) </pre> <pre> y = mylib.func1() print(y) </pre>
	<ul style="list-style-type: none"> 定义函数 func2，没有形参，有返回值 <pre> def func2(): x = 70 y = x**0.5 - 7 print(y) return y </pre> <pre> y = mylib.func2() print(y) </pre>
	<ul style="list-style-type: none"> 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) <p>定义函数的时候，这个参数叫做形参。这个参数它只是一个形式上的一个东西，现在你不知道 X 是什么。在调用的时候才会给形参赋值叫做实参。</p> <p>定义函数：给位置形参：</p> <pre> def func3(x): y = x**0.5 - 7 return y </pre> <p>调用函数：用位置实参、命名实参（两个都可，没区别）：</p> <pre> y = mylib.func3(45) print(y) </pre> <pre> y = mylib.func3(x=47) print(y) </pre>
	<ul style="list-style-type: none"> 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) <p>定义函数：给命名形参：</p> <pre> def func4(x=50): y = x**0.5 - 7 return y </pre> <p>调用函数：用位置实参、命名实参（两个都可，没区别）：</p>

	<pre>y = mylib.func4(48) print(y) y = mylib.func4(x=49) print(y)</pre>	
	<ul style="list-style-type: none"> 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 <pre>def calculate(a, b, operation="add"): if operation == "add": return a + b elif operation == "subtract": return a - b else: return None</pre> <p>A b 是位置形参，operation 是命名形参，位置参数必须在命名参数之前</p>	
	<ul style="list-style-type: none"> 定义函数 func6，在形参列表中使用 / 来限定只接受位置实参 定义函数 func7，在形参列表中使用 * 来限定只接受命名实参 <pre>def func7(a, /, b, *, operation="add"):</pre> <p>调用函数： */前必须是用位置实参 */后必须是用位置实参</p>	
	<ul style="list-style-type: none"> 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) <pre>def func8(*numbers): breakpoint() total = 0 for num in numbers: total = total + num return total</pre> <pre>print(mylib.func8(4, 8, 12))</pre> <ul style="list-style-type: none"> 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) 	
	<ul style="list-style-type: none"> 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，接受一个命名形参，两个命名形参，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 <p>这些都可以直接文豆包，自己学： Python+上面的东西+给我例子</p>	
	<ul style="list-style-type: none"> 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 <p>内嵌文档：</p> <pre>def func12(arg1, arg2, named_arg="default"): """多个参数的调用例子""" print(f"位置实参 arg1: {arg1}") print(f"位置实参 arg2: {arg2}") print(f"命名实参 named_arg: {named_arg}")</pre> <p>类型注解：</p>	

在行参的后面加上一个冒号: xx 类型。这就可以给这个调用的人, 来给她个提示。
在往进传实参的时候, 最好传什么样的类型, 那么它叫做类型的注解

```
def func12(arg1: str, arg2: int, named_arg: str = "default") -> None:  
    """多个参数的调用例子"  
    print(f"位置实参 arg1: {arg1}")  
    print(f"位置实参 arg2: {arg2}")  
    print(f"命名实参 named_arg: {named_arg}")
```

规范版

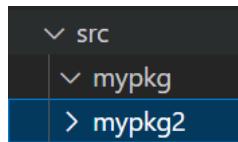
. 把 `mylib` 模块转变为 软件包 (package) 安装进当前的 Conda 环境来使用

不知道怎么样把自己的模块给打包成软件包安装:

软件包的打包步骤:

1. 新建文件夹 (随意命名: `src`) —— 用于把软件包们放在这里

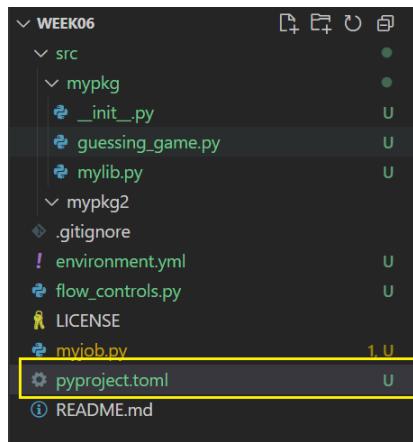
2. `src` 下面, 建立文件夹 (随意命名: `mypkg`), 把 `Mylib` (模块) (模块里可以定义很多函数, 代码都写在这里) 放进来



3. 在 `Mypkg` 里新建一个文件: `__init__.py`

有了这个特殊的文件, `python` 就知道 `Mypkg` 这个文件夹是一个软件包
把

4. 在最外面, 创建 `pyproject.toml` 配置文件,



第一, 按照 文档 填写基本的软件包信息

```
1 [project]
2   name = "mypackage"
3   version = "2025.4.14"
4   dependencies = [
5     "openpyxl",
6   ]
7   authors = [
8     {"name = "xy", email = "626907497@qq.com"},  

9   ]
10  description = "测试用的软件包"
11
12  [project.optional - dependencies]
13  dev = [
14    "pytest",
15  ]
```

第二，按照[文档](#) 填写软件包的 构建 (build) 配置

```
17 [build-system]
18 requires = ["hatchling"]
19 build-backend = "hatchling.build"
20
21 [tool.hatch.build.targets.wheel]
22 packages = [
23   'src/mypkg',
24 ]
```

黄色要写对

总预览:

