

Week06

复制 environment.yml 文件到 week06, 修改名称

```
(base) 86139@LAPTOP-J150R7EU MINGW64
$ cd week06

(base) 86139@LAPTOP-J150R7EU MINGW64
$ cp ../week05/environment.yml ./

! environment.yml
1  name: week06
2  channels:
3    - conda-forge
4  dependencies:
5    - python=3.12
6    - wat-inspector
```

创建 conda 环境、删除 conda 指令

```
$ conda env remove -n prj1
$ conda env remove -n prj2
```

```
(base) 86139@LAPTOP-J150R7EU MINGW64 ~/repo/week06 (main)
$ conda env list
# conda environments:
#
base                * C:\Users\86139\anaconda3
myproject           C:\Users\86139\anaconda3\envs\myproject
prj1                 C:\Users\86139\anaconda3\envs\prj1
prj2                 C:\Users\86139\anaconda3\envs\prj2
week04               C:\Users\86139\anaconda3\envs\week04
week05               C:\Users\86139\anaconda3\envs\week05
week06               C:\Users\86139\anaconda3\envs\week06

$ conda env list
# conda environments:
#
base                * C:\Users\86139\anaconda3
myproject           C:\Users\86139\anaconda3\envs\myproject
week04              C:\Users\86139\anaconda3\envs\week04
week05              C:\Users\86139\anaconda3\envs\week05
week06              C:\Users\86139\anaconda3\envs\week06
```

创建 guessing_game.py, 复制代码, 运用 pdb 调试器

```
86139@LAPTOP-J150R7EU MINGW64 ~/repo/week06 (main)
$ python guessing_game.py
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。
(第 1 次尝试) 请输入你猜的数字 (输入整数, 或者输入 q 回车退出): 78
猜的数字太大了, 再试试。
(第 2 次尝试) 请输入你猜的数字 (输入整数, 或者输入 q 回车退出): 56
猜的数字太小了, 再试试。
(第 3 次尝试) 请输入你猜的数字 (输入整数, 或者输入 q 回车退出): 67
猜的数字太大了, 再试试。
(第 4 次尝试) 请输入你猜的数字 (输入整数, 或者输入 q 回车退出): 62
猜的数字太小了, 再试试。
(第 5 次尝试) 请输入你猜的数字 (输入整数, 或者输入 q 回车退出): 65
猜的数字太小了, 再试试。
(第 6 次尝试) 请输入你猜的数字 (输入整数, 或者输入 q 回车退出): 66
恭喜你 🎉, 猜对了!
游戏结束, 再见 🙋。
```

For 迭代循环 【iteration loop】

```

fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    fruit += ",ok"
    print(fruit)

message = "Hello"
for char in message:
    print(char)

person = {"name": "John", "age": 30, "city": "New York"}
for key in person:
    print(key, ":", person[key])

for i in range(5):
    print(i)

for i in range(3):
    for j in range(2):
        print(i, j)

```

```

$ python flow_control.py
apple,ok
banana,ok
cherry,ok
H
e
l
l
o
name : John
age : 30
city : New York
0
1
2
3
4
0 0
0 1
1 0
1 1
2 0
2 1

```

While 条件循环 【conditional loop】

```

# while语句
count = 0
while count < 5:
    print(count)
    count = count + 1

user_input = ""
while user_input != "quit":
    user_input = input("请输入内容 (输入 'quit' 退出) : ")
    if user_input != "quit":
        print(f"你输入的是: {user_input}")

numbers = [1, 2, 3, 4, 5]
while numbers:
    current_number = numbers.pop(0)
    print(f"正在处理数字: {current_number}")

```

```

0
1
2
3
4
请输入内容 (输入 'quit' 退出) : 11
你输入的是: 11
请输入内容 (输入 'quit' 退出) : quit
正在处理数字: 1
正在处理数字: 2
正在处理数字: 3
正在处理数字: 4
正在处理数字: 5

```

Break 打断 跳出循环

```
# break语句
numbers = [1, 2, 3, 4, 5]
for num in numbers:
    if num == 3:
        break
    print(num) # 在 for 循环中使用 break 语句
```

```
1
2
```

Continue 跳至下一循环

```
# Continue语句
numbers = [1, 2, 3, 4, 5]
for num in numbers:
    if num % 2 == 0:
        continue
    print(num)
```

```
1
3
5
```

For...else 循环未被打断处理

```
# For...else循环
fruits = ['apple', 'banana', 'cherry']
search_fruit = 'mango'
for fruit in fruits:
    if fruit == search_fruit:
        print(f"找到了 {search_fruit}。")
        break
else:
    print(f"没有找到 {search_fruit}。")
```

```
0
没有找到 mango。
```

If 条件分支

```
# If语句
age = 20
is_student = True
if age >= 18 and is_student:
    print("你是成年学生。")
```

```
你是成年学生。
```

If...elif...elif 多重条件分支

```
# If...elif[...elif]多重条件分支
score = 85

if score >= 90:
    print("成绩等级:A")
elif score >= 80:
    print("成绩等级:B")
elif score >= 70:
    print("成绩等级:C")
elif score >= 60:
    print("成绩等级:D")
else:
    print("成绩等级:F")
```

成绩等级: B

If...else 未满足条件的处理

```
# If...else未满足条件的处理
number = -5
if number >= 0:
    print(f"{number} 是正数或零。")
else:
    print(f"{number} 是负数。")
```

-5 是负数。

Try...except[...except...else...finally]捕捉异常的处理

```
# Try...except[...except...else...finally]捕捉异常的处理
try:
    num1 = int(input("请输入一个被除数: "))
    num2 = int(input("请输入一个除数: "))
    result = num1 / num2
except ValueError:
    print("输入无效, 请输入有效的整数。")
except ZeroDivisionError:
    print("错误: 除数不能为零。")
else:
    print(f"结果是: {result}")
finally:
    print("操作完成。")
```

请输入一个被除数: 12.9
输入无效, 请输入有效的整数。
操作完成。

Raise 主动抛出异常

```
# Raise主动抛出异常
class CustomError(Exception):
    pass

def check_number(num):
    if num % 2 != 0:
        raise CustomError("输入的数字必须是偶数。")
    return num

try:
    number = check_number(3)
    print(f"输入的数字是: {number}")
except CustomError as e:
    print(f"发生自定义错误: {e}")
```

发生自定义错误：输入的数字必须是偶数。

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：

```
mylib.py > ...
1 def func1():
2     x = 50
3     y = x**0.5 - 7
4     print(y)
```

定义函数 **func1**，没有形参，没有返回值

```
myjob.py > ...
1 import mylib
2
3 y = mylib.func1
4 print(y)
```

\$ python myjob.py
<function func1 at 0x000002168717B060>
(weakref)

定义函数 **func2**，没有形参，有返回值

```
def func2():
    x = 70
    y = x**0.5 - 7
    print(y)
    return y
```

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)

```
def func3():  
    y = x**0.5 - 7  
    return y
```

```
y = mylib.func3(45)  
print(y) # 位置实参  
  
y = mylib.func3(x=47)  
print(y) # 命名实参
```

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)

```
def func4(x=50):  
    y = x**0.5 - 7  
    return y
```

```
y = mylib.func4(47)  
print(y) # 位置实参  
  
y = mylib.func4(x=49)  
print(y) # 命名实参  
  
y = mylib.func4()  
print(y) # 不传实参
```

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

```
def caculate(a, b, operation="add"):  
    if operation == "add":  
        return a + b  
    elif operation == "subtract":  
        return a - b  
    else:  
        return None
```

```
print(mylib.caculate(5, 10, "add"))  
print(mylib.caculate(operation="add", b=5, a=10))  
print(mylib.caculate(b=5, a=10))  
print(mylib.caculate(5, 10, operation="subtract"))
```

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

```
def func6(a, /, b, operation="add"):
    if operation == "add":
        return a + b
    elif operation == "subtract":
        return a - b
    else:
        return None
```

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

```
def func7(a, /, b, *, operation="add"):
    if operation == "add":
        return a + b
    elif operation == "subtract":
        return a - b
    else:
        return 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)

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

```
def func8(*numbers):
    total = 0
    for num in numbers:
        total = total + num
    return total

def func9(**user):
    for key, value in user.items():
        print(f"{key}:{value}")

def func10(arg1, arg2, named_arg="default"):
    print(f"位置实参  arg1:{arg1}")
    print(f"位置实参  arg2:{arg2}")
    print(f"命名实参  name_arg:{named_arg}")
```

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

```
def func11(arg1, arg2):
    print(f"arg1 的值是:{arg1}")
    print(f"arg2 的值是:{arg2}")

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  name_arg:{named_arg}")
```

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

```
pyproject.toml
1  [project]
2  name = "mypackage"
3  version = "2025.4.14"
4  dependencies = [
5  | "openpyxl",
6  | ]
7  authors = [
8  | {name = "Vicky", email = "1796757476@163.com"},
9  | ]
10 description = "测试用的软件包"
11
12 [project.optional-dependencies]
13 dev = [
14 | "pytest",
15 | ]
16
17 [build-system]
18 requires = ["hatchling"]
19 build-backend = "hatchling.build"
20
21 [tool.hatch.build.target.wheel]
22 packages = [
23 | "src/mypkg",
24 | ]
```

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

```
! environment.yml
1  name: week06
2  channels:
3    - conda-forge
4  dependencies:
5    - python=3.12
6    - wat-inspector
7    - pip
8    - pip:
9      - "-e ."
```