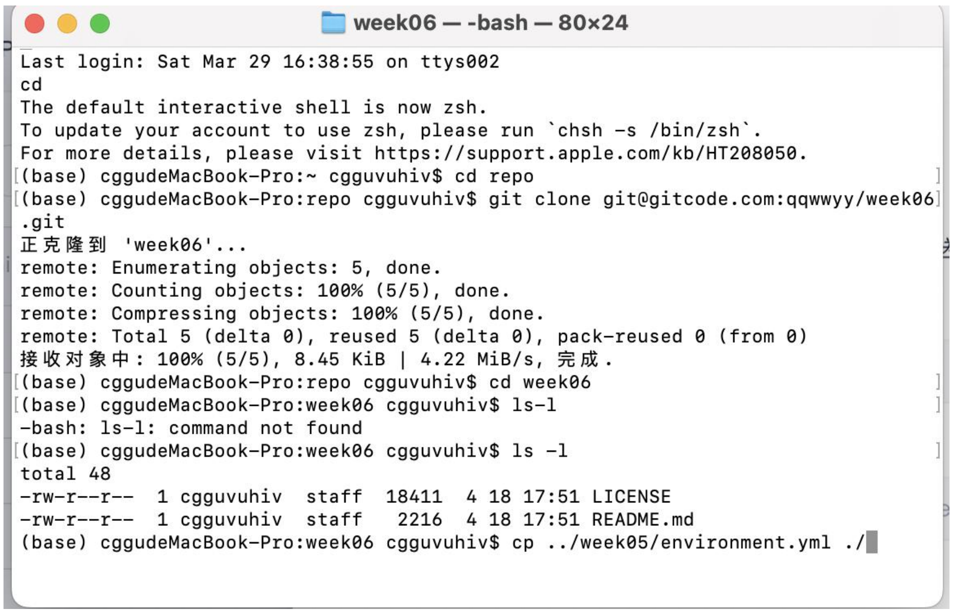
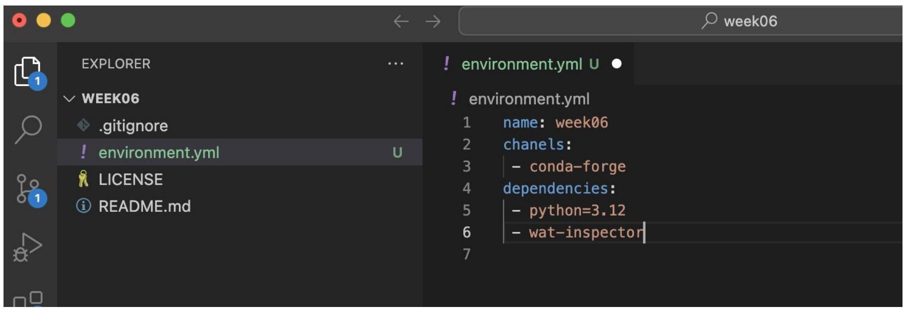
1.



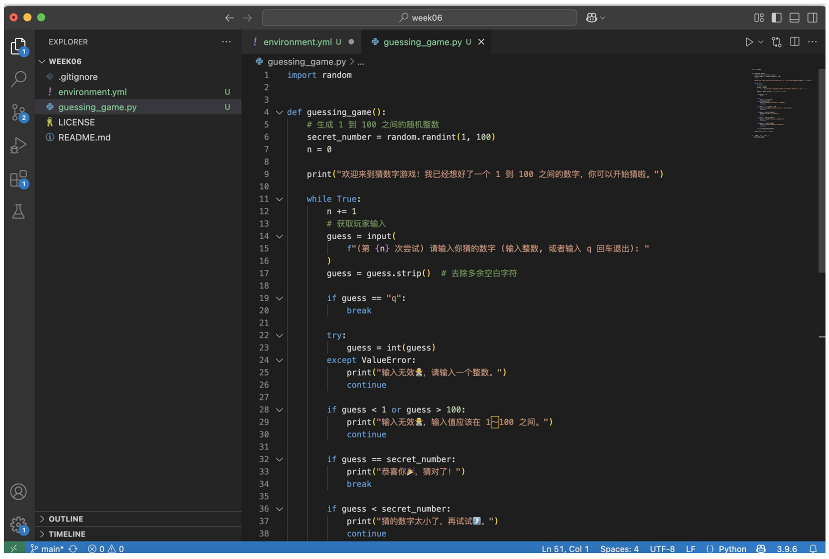
```
Last login: Sat Mar 29 16:38:55 on ttys002
cd
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
(base) cggudeMacBook-Pro:~ cgguvuhiv$ cd repo
(base) cggudeMacBook-Pro:repo cgguvuhiv$ git clone git@gitcode.com:qqwwyy/week06
.git
正克隆到 'week06'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 5 (delta 0), reused 5 (delta 0), pack-reused 0 (from 0)
接收对象中: 100% (5/5), 8.45 KiB | 4.22 MiB/s, 完成.
(base) cggudeMacBook-Pro:repo cgguvuhiv$ cd week06
(base) cggudeMacBook-Pro:week06 cgguvuhiv$ ls-l
-bash: ls-l: command not found
(base) cggudeMacBook-Pro:week06 cgguvuhiv$ ls -l
total 48
-rw-r--r--  1 cgguvuhiv  staff  18411  4 18 17:51 LICENSE
-rw-r--r--  1 cgguvuhiv  staff   2216  4 18 17:51 README.md
(base) cggudeMacBook-Pro:week06 cgguvuhiv$ cp ../week05/environment.yml ./
```

2.



```yaml
name: week06
chanels:
  - conda-forge
dependencies:
  - python=3.12
  - wat-inspector
```

3.



```python
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
```

4.pdb 调解器:

功能特点

设置断点：能在程序代码的特定行设置断点，当程序执行到断点时会暂停，方便开发者检查当前程序的状态，包括变量的值、调用栈信息等。例如在一个复杂的函数内部设置断点，就能查看函数执行到该点时各个变量的取值情况。

单步执行：支持逐行执行代码，每次执行一行，让开发者清晰了解程序的执行流程，观察每一行代码执行后的变化，从而发现逻辑错误。

查看变量：在调试过程中随时查看程序中变量的值，这对于检查程序运行过程中数据的正确性至关重要。可以直观看到变量是否按照预期进行了赋值和变化。

回溯调用栈：当程序出现异常或错误时，PDB 调试器能展示调用栈信息，帮助开发者快速定位错误发生的位置和原因，了解函数的调用关系和执行顺序。

使用方式

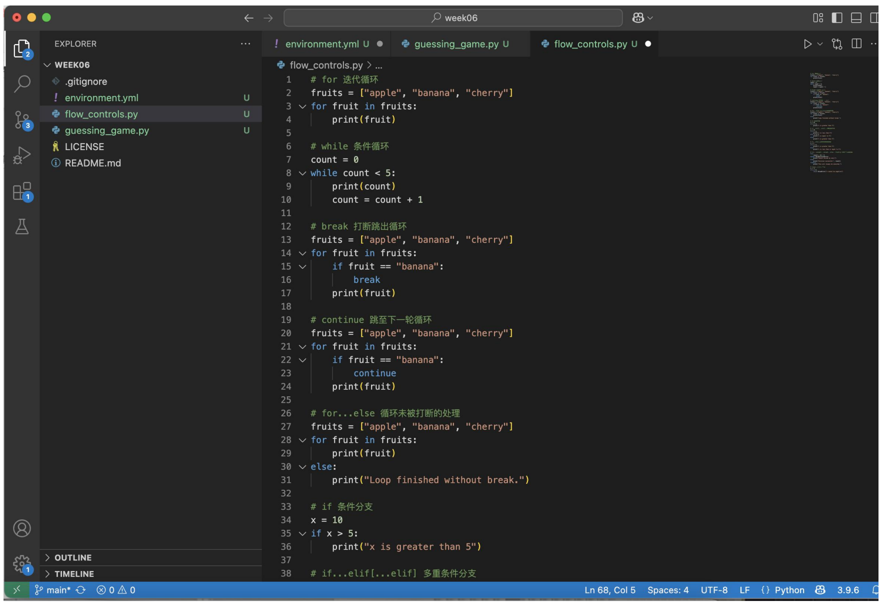
命令行方式：在命令行中运行 Python 程序时，通过 python -m pdb your_script.py 这种形式启动调试。进入调试环境后，使用一系列命令进行调试操作，如 b（设置断点）、n（单步执行）、s（进入函数）、c（继续执行）、p（打印变量值）等。

代码嵌入方式：在代码中通过 import pdb; pdb.set_trace()语句在需要调试的位置插入调试器。运行程序时，一旦执行到该语句，程序就会暂停进入调试状态，开发者可使用调试命令进行调试。

5.# for 迭代循环

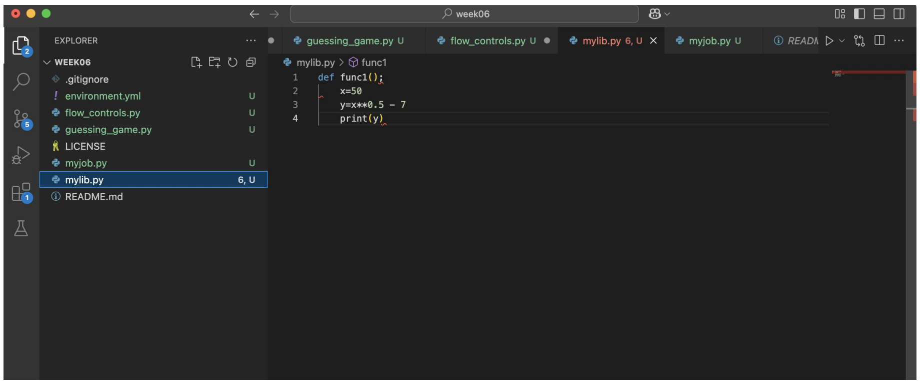
```
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)
# while 条件循环
count = 0
while count < 5:
    print(count)
    count = count + 1
# break 打断跳出循环
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    if fruit == "banana":
        break
    print(fruit)
# continue 跳至下一轮循环
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    if fruit == "banana":
        continue
    print(fruit)
# for...else 循环未被打断的处理
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)
else:
    print("Loop finished without break.")
# if 条件分支
x = 10
if x > 5:
    print("x is greater than 5")
# if...elif[...elif] 多重条件分支
x = 10
if x < 5:
    print("x is less than 5")
elif x == 5:
    print("x is equal to 5")
else:
    print("x is greater than 5")
# if...else 未满足条件的处理
x = 3
```

```
if x > 5:
    print("x is greater than 5")
else:
    print("x is less than or equal to 5")
# try...except[...except...else...finally]  捕捉异常的处理
try:
    result = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero!")
else:
    print("Division successful:", result)
finally:
    print("This will always be executed.")
# raise  主动抛出异常
x = -1
if x < 0:
    raise ValueError("x cannot be negative")
```



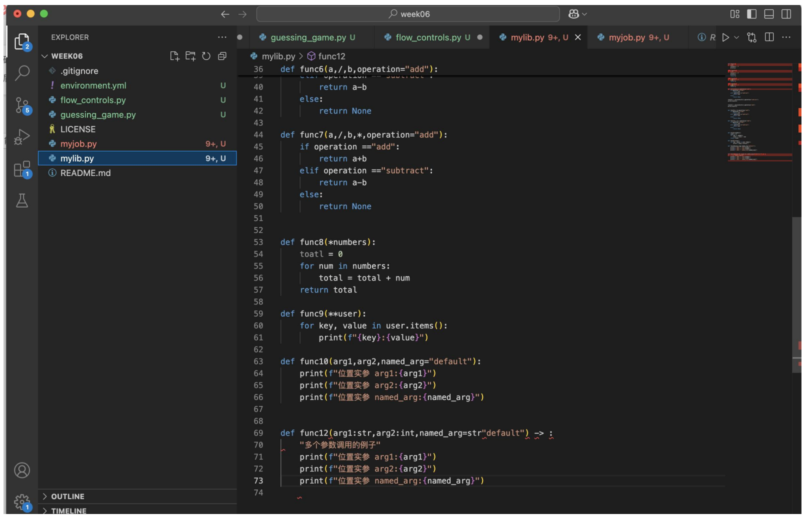
```python
# for 迭代循环
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)

# while 条件循环
count = 0
while count < 5:
    print(count)
    count = count + 1

# break 打断跳出循环
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    if fruit == "banana":
        break
    print(fruit)

# continue 跳至下一轮循环
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    if fruit == "banana":
        continue
    print(fruit)

# for...else 循环未被打断的处理
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)
else:
    print("Loop finished without break.")

# if 条件分支
x = 10
if x > 5:
    print("x is greater than 5")

# if...elif[...elif] 多重条件分支
```

6.

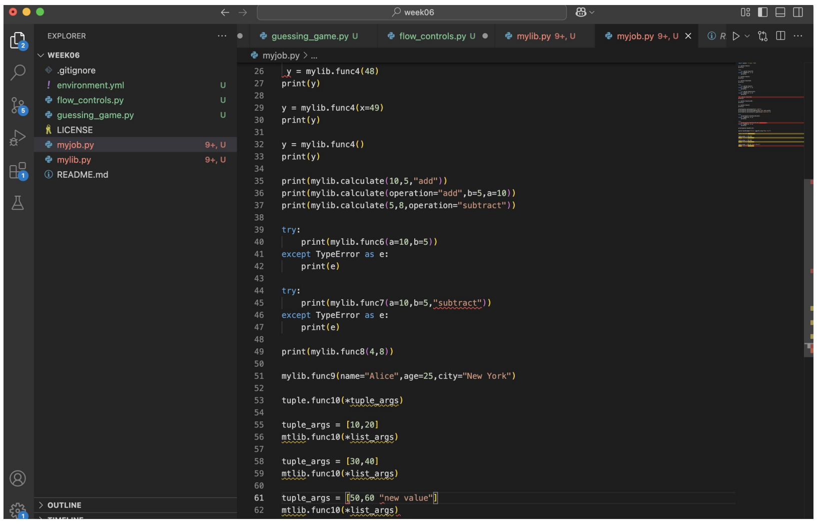


```python
def func1():
    x=50
    y=x**0.5 - 7
    print(y)
```

7.



```python
def func6(a,/,b,operation="add"):
        return a-b
    else:
        return None

def func7(a,/,b,*,operation="add"):
    if operation =="add":
        return a+b
    elif operation =="subtract":
        return a-b
    else:
        return None


def func8(*numbers):
    toatl = 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"位置实参 named_arg:{named_arg}")


def func12(arg1:str,arg2:int,named_arg=str"default") -> :
    "多个参数调用的例子"
    print(f"位置实参 arg1:{arg1}")
    print(f"位置实参 arg2:{arg2}")
    print(f"位置实参 named_arg:{named_arg}")

```



```python
 y = mylib.func4(48)
print(y)

y = mylib.func4(x=49)
print(y)

y = mylib.func4()
print(y)

print(mylib.calculate(10,5,"add"))
print(mylib.calculate(operation="add",b=5,a=10))
print(mylib.calculate(5,8,operation="subtract"))

try:
    print(mylib.func6(a=10,b=5))
except TypeError as e:
    print(e)

try:
    print(mylib.func7(a=10,b=5,"subtract"))
except TypeError as e:
    print(e)

print(mylib.func8(4,8))

mylib.func9(name="Alice",age=25,city="New York")

tuple.func10(*tuple_args)

tuple_args = [10,20]
mtlib.func10(*list_args)

tuple_args = [30,40]
mtlib.func10(*list_args)

tuple_args = [50,60 "new value"]
mtlib.func10(*list_args)
```