在 Python 中，有几种方法可以让程序主动终止自己的执行。下面介绍这些方法及其适用场景：

1. 使用 sys.exit()

适用场景：在脚本或程序的任何位置强制退出。

特点：会引发 SystemExit 异常，可以被捕获。

python

运行

import sys

# 正常退出（状态码0表示成功）

sys.exit(0)

# 异常退出（非零状态码表示错误）

sys.exit("发生错误：文件不存在") # 等价于 sys.exit(1)

注意：如果在 try 块中调用 sys.exit()，可以用 except SystemExit: 捕获并处理。

1. 使用 os.\_exit()

适用场景：在子进程或需要立即终止的场景（如守护进程）。

特点：直接终止进程，不执行任何清理操作（如 finally 块、atexit 注册的函数）。

python

运行

import os

# 立即终止程序，状态码为1

os.\_exit(1)

1. 引发 SystemExit 异常

适用场景：与 sys.exit() 等价，但更显式地表明意图。

python

运行

raise SystemExit("程序退出") # 等同于 sys.exit("程序退出")

1. 在主程序中使用 return

适用场景：从 main() 函数返回，隐式终止程序。

python

运行

def main():

# 程序逻辑  
  
if error\_condition:  
  
 return 1 # 返回非零状态码表示错误  
  
return 0 # 返回0表示成功

if **name** == "**main**":

sys.exit(main()) # 将main()的返回值作为退出状态码

5. 在交互式环境中使用 exit() 或 quit()

适用场景：仅用于 Python 解释器或 Jupyter 等交互式环境，不建议在脚本中使用。

python

运行

exit() # 交互式环境中退出

选择建议

常规退出：使用 sys.exit(0) 或 return（从 main() 函数）。

紧急终止：使用 os.\_exit()（如在信号处理函数中）。

异常处理：捕获 SystemExit 异常以执行清理操作。

示例：优雅地退出程序

python

运行

import sys

def cleanup\_resources():

print("清理资源...")

try:

# 主程序逻辑  
  
if some\_condition:  
  
 raise Exception("发生错误")

except Exception as e:

print(f"错误：{e}")  
  
sys.exit(1)

finally:

cleanup\_resources() # 无论是否异常，都会执行清理

这个示例展示了如何结合异常处理和 sys.exit() 实现优雅退出。