

# 系统基础开发工具

## 实验二

姓 名: 陈 佳 玲  
学 号: 23100021002  
专 业: 23 级环境工程  
2025 年 9 月 15 日

## 目录

<b>1</b>	<b>实验目的</b>	<b>3</b>
<b>2</b>	<b>实验内容</b>	<b>3</b>
2.1	命令行环境 . . . . .	3
2.1.1	任务控制 . . . . .	3
2.1.2	别名 . . . . .	4
2.2	python . . . . .	5
2.2.1	中文输入 . . . . .	5
2.2.2	基础语法 . . . . .	5
2.2.3	变量类型 . . . . .	6
2.2.4	运算符 . . . . .	7
2.2.5	条件语句 . . . . .	10
2.2.6	循环语句 . . . . .	11
2.2.7	break 语句 . . . . .	13
2.2.8	字符串 . . . . .	16
2.2.9	列表 . . . . .	17
2.2.10	元组 . . . . .	18
2.2.11	字典 . . . . .	20
2.2.12	日期和时间 . . . . .	22
2.2.13	编辑模式 . . . . .	24
2.2.14	时间 . . . . .	24
2.2.15	日期 . . . . .	26

2.3	Python 计算机视觉 . . . . .	28
2.3.1	Pillow . . . . .	28
2.3.2	Numpy . . . . .	29
2.3.3	Scipy . . . . .	31
<b>3</b>	<b>心得体会</b>	<b>32</b>
<b>4</b>	<b>实验代码查看链接</b>	<b>33</b>

## 1 实验目的

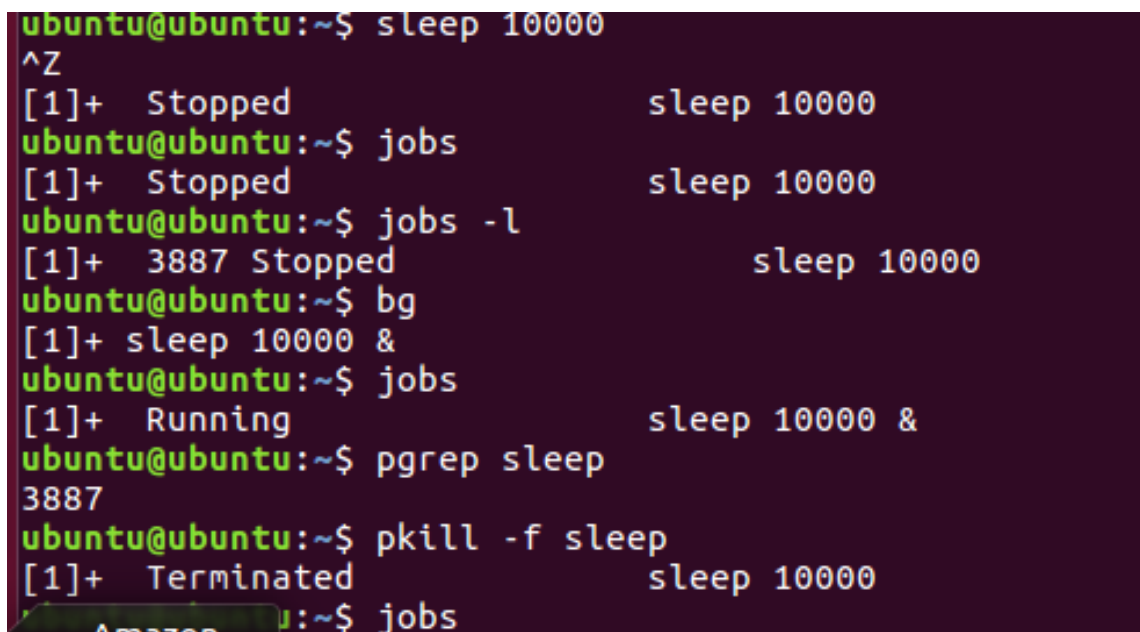
1. 学习 Shell 工具和脚本：通过课堂学习和课下探索，熟悉 shell 基本指令，认识脚本以及一些基本语法知识，强化对于 bash 命令的熟练度，掌握在 Linux 环境下使用 Shell 进行基本操作和自动化任务的能力。
2. 掌握编辑器 Vim 的使用：通过实际操作，学习 Vim 的基本操作模式、常用命令和高级功能，能够熟练使用 Vim 进行文本编辑和代码编写，提高在命令行环境下的编辑效率。
3. 培养系统开发基础能力：通过 Shell 脚本编写和 Vim 编辑器使用的结合，培养在 Unix/Linux 系统环境下进行软件开发的基础能力，为后续课程和项目开发打下坚实基础。

## 2 实验内容

### 2.1 命令行环境

#### 2.1.1 任务控制

(1) 在终端中执行 `sleep 10000` 这个任务。然后用 `Ctrl-Z` 将其切换到后台并使用 `bg` 来继续允许它。现在，使用 `pgrep` 来查找 `pid` 并使用 `pkill` 结束进程而不需要手动输入 `pid`。(提示：：使用 `-af` 标记)。



```
ubuntu@ubuntu:~$ sleep 10000
^Z
[1]+  Stopped                  sleep 10000
ubuntu@ubuntu:~$ jobs
[1]+  Stopped                  sleep 10000
ubuntu@ubuntu:~$ jobs -l
[1]+  3887 Stopped              sleep 10000
ubuntu@ubuntu:~$ bg
[1]+ sleep 10000 &
ubuntu@ubuntu:~$ jobs
[1]+  Running                  sleep 10000 &
ubuntu@ubuntu:~$ pgrep sleep
3887
ubuntu@ubuntu:~$ pkill -f sleep
[1]+  Terminated              sleep 10000
Amazon:~$ jobs
```

图 1: sleep 10000

(2) 使用 `sleep 60 &` 作为先执行的程序。一种方法是使用 `wait` 命令。尝试启动这个休眠命令，然后待其结束后再执行 `ls` 命令。

```

[1]+  Done                  sleep 60
ubuntu@ubuntu:~$ sleep 60 &
[1] 3870
ubuntu@ubuntu:~$ PID = 3870
PID: command not found
ubuntu@ubuntu:~$ PID=3870
ubuntu@ubuntu:~$ wait $PID;ls
[1]+  Done                  sleep 60
0.py          b.py          hello.sh
    t2.txt
1.sh          buflab        html_root
    tar
2-1.py        chen          kjv12.txt
    Templates
2-2.py        c.py          lab
    test
2-3.py        Desktop       lab0
    test1.txt
=2.5          Documents     lab1-handout
    test.c
2.sh          Downloads     lab1-handout.t
ar test_dir
3-1.py        d.py          lab3
    test_normal
3-2.py        e.py          mapreduce.py
    test_paranoid
3-3.py        examples.desktop marco_history.
log test_unsafe
3-4.py        false         marco.sh
    Videos
3-5.py        file          Music
    x.py
aaa.c         file.txt       Pictures
    y.py
a.py          f.py          Public
    z2.py
bomb_23100021002.tar fu             t1.c
    z.py
bomb73        hello1.txt     t1.txt

```

图 2: sleep 60

### 2.1.2 别名

创建一个 `v` 别名，它的功能是当我们将 `vim` 输入为 `v` 时也能正确执行。

```
ubuntu@ubuntu:~$ alias v=vim
ubuntu@ubuntu:~$ v a.txt
```

图 3: alias

## 2.2 python

### 2.2.1 中文输入

文件开头加入 `# -*- coding: UTF-8 -*-` 或者 `# coding=utf-8` 就行了

```
#!/user/bin/python
# -*- coding: UTF-8 -*-

print("你好")
~
~
```

图 4: 中文输入

### 2.2.2 基础语法

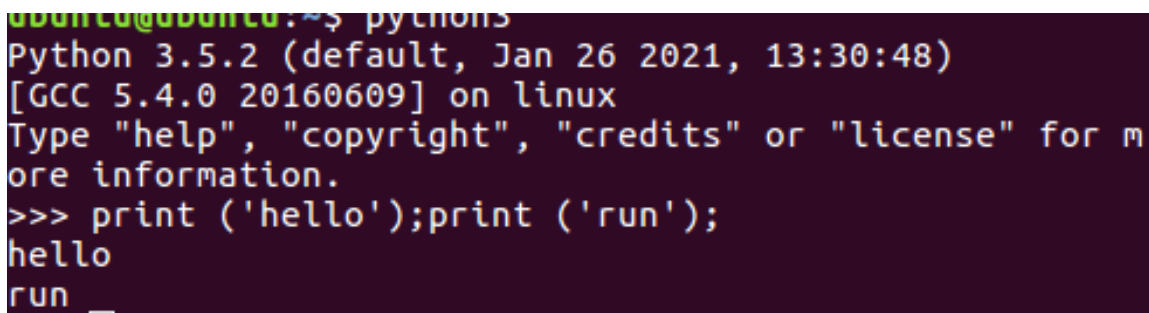
1. 交互式编程: 交互式编程不需要创建脚本文件, 是通过 Python 解释器的交互模式进来编写代码

脚本式编程: 通过脚本参数调用解释器开始执行脚本, 直到脚本执行完毕

```
ubuntu@ubuntu:~$ python3 pro1.py
Ubuntu Software python3 pro1.py
你好
ubuntu@ubuntu:~$ python3
Python 3.5.2 (default, Jan 26 2021, 13:30:48)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for m
```

图 5: 交互式编程与脚本式编程

2. 标识符: 在 Python 里, 标识符由字母、数字、下划线组成。在 Python 中, 所有标识符可以包括英文、数字以及下划线(`_`), 但不能以数字开头。Python 中的标识符是区分大小写的。



```
ubuntu@ubuntu:~$ python3
Python 3.5.2 (default, Jan 26 2021, 13:30:48)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print('hello');print('run');
hello
run
```

图 6: 标识符

3. 行和缩进学习 Python 与其他语言最大的区别就是，Python 的代码块不使用大括号来控制类，函数以及其他逻辑判断。python 最具特色的就是用缩进来写模块。缩进的空白数量是可变的，但是所有代码块语句必须包含相同的缩进空白数量，这个必须严格执行

### 2.2.3 变量类型

变量可以指定不同的数据类型，这些变量可以存储整数，小数或字符

- (a) Python 数字: 数字数据类型用于存储数值。他们是不可改变的数据类型，这意味着改变数字数据类型会分配一个新的对象。当你指定一个值时，Number 对象就会被创建。Python 支持四种不同的数字类型：int（有符号整型）、long（长整型，也可以代表八进制和十六进制）、float（浮点型）、complex（复数）
- (b) Python 字符串: 字符串或串 (String) 是由数字、字母、下划线组成的一串字符。
- (c) Python 列表: List（列表）是 Python 中使用最频繁的数据类型。  
列表可以完成大多数集合类的数据结构实现。它支持字符，数字，字符串甚至可以包含列表（即嵌套）。  
列表用 [] 标识，是 python 最通用的复合数据类型。
- (d) Python 元组: 元组是另一个数据类型，类似于 List（列表）。  
元组用 () 标识。内部元素用逗号隔开。但是元组不能二次赋值，相当于只读列表

```
#!/user/bin/python
# -*- coding: UTF-8 -*-
#数字
var1=1
var2=10

#字符串
s="a1a2a3...an"
print s

#列表
list=['run',786,2.23,'join',70.2]
print list

#元组
tuple = ( 'runoob', 786 , 2.23, 'john', 70.2 )
tinytuple = (123, 'john')

#字典
dict = {}
dict['one'] = "This is one"
dict[2] = "This is two"

tinydict = {'name': 'runoob','code':6734, 'dept': 'sales'}
```

图 7: 变量类型

#### 2.2.4 运算符

Python 语言支持以下类型的运算符: 算术运算符、比较（关系）运算符、赋值运算符、逻辑运算符、位运算符、成员运算符、身份运算符、运算符优先级

(a) Python 算术运算符: 加法: +、

减法: -、

乘法: \*、

除法: /、

取模: %

幂运算: \*\*、

取整除: //

(b) 比较运算符: 比较两个值, 返回布尔值。

等于: ==、

不等于: !=、

大于: >、

小于: <、

大于等于:  $\geq$ 、  
小于等于:  $\leq$   
: 用于条件组合。  
与: and、  
或: or、  
非: not

(c) 赋值运算符: 用于给变量赋值。

简单赋值:  $=$ 、  
加法赋值:  $+=$ 、  
减法赋值:  $-=$ 、  
乘法赋值:  $*=$ 、  
除法赋值:  $/=$ 、  
取模赋值:  
幂赋值:  $**=$ 、  
取整除赋值:  $//=$

(d) 位运算符: 按二进制位进行操作

按位与:  $\&$ 、  
按位或:  $|$ 、  
按位异或:  $\wedge$ 、  
按位取反:  $\sim$ 、  
左移:  $\ll$ 、  
右移:  $\gg$



```
def main(): 1个用法
    print("=" * 50)
    print("=" * 50)
    # 1. 算术运算符
    print("\n1. 算术运算符")
    a = 15
    b = 4
    print(f"a = {a}, b = {b}")
    print(f"a + b = {a + b}") # 加法
    print(f"a - b = {a - b}") # 减法
    # 2. 比较运算符
    print("\n2. 比较运算符")
    x = 10
    y = 20
    print(f"x = {x}, y = {y}")
    print(f"x == y: {x == y}") # 等于
    print(f"x < y: {x < y}") # 小于
    # 3. 逻辑运算符
    print("\n3. 逻辑运算符")
    is_weekend = True
    is_sunny = False
    print(f"is_weekend = {is_weekend}, is_sunny = {is_sunny}")
    print(f"is_weekend and is_sunny: {is_weekend and is_sunny}") # 与
    print(f"is_weekend or is_sunny: {is_weekend or is_sunny}") # 或
    print(f"not is_weekend: {not is_weekend}") # 非
    # 4. 赋值运算符
    print("\n4. 赋值运算符")
    num = 10
    print(f"初始值: num = {num}")
    num += 5 # num = num + 5
    # 7. 位运算符
    print("\n7. 位运算符")
    p = 60 # 二进制: 0011 1100
    q = 13 # 二进制: 0000 1101
    print(f"p & q = {p & q}") # 按位与: 12 (0000 1100)
    print(f"p | q = {p | q}") # 按位或: 61 (0011 1101)
```

图 8: 运算符

### 1. 算术运算符

```
a = 15, b = 4
```

```
a + b = 19
```

```
a - b = 11
```

### 2. 比较运算符

```
x = 10, y = 20
```

```
x == y: False
```

```
x < y: True
```

### 3. 逻辑运算符

```
is_weekend = True, is_sunny = False
```

```
is_weekend and is_sunny: False
```

```
is_weekend or is_sunny: True
```

```
not is_weekend: False
```

### 4. 赋值运算符

```
初始值: num = 10
```

### 7. 位运算符

```
p & q = 12
```

```
p | q = 61
```

图 9: 运算结果

#### 2.2.5 条件语句

Python 条件语句是通过一条或多条语句的执行结果（True 或者 False）来决定执行的代码块。

Python 编程中 if 语句用于控制程序的执行，基本形式为：

if 判断条件：

    执行语句……

else：

    执行语句……

```
#!/bin/bash
```

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

# 例1: if 基本用法

flag = False
name = 'luren'
if name == 'python': # 判断变量是否为 python
    flag = True # 条件成立时设置标志为真
    print ('welcome boss' ) # 并输出欢迎信息
else:
    print (name) # 条件不成立时输出变量名称
```

hello x

D:\PythonProject\.venv\Scripts\python.exe D:\PythonProject\hello.py  
luren

进程已结束，退出代码为 0

图 10: 条件语句 if

## 2.2.6 循环语句

(a) While 循环语句

```
1  #!/usr/bin/python
2
3  count = 0
4  while (count < 9):
5      print ('The count is:', count)
6      count = count + 1
7
8  print ("Good bye!")
```

运行 hello x

D:\PythonProject\.venv\Scripts\python.exe D:\PythonProject\hello.py

The count is: 0  
The count is: 1  
The count is: 2  
The count is: 3  
The count is: 4  
The count is: 5  
The count is: 6  
The count is: 7  
The count is: 8  
Good bye!

进程已结束，退出代码为 0

图 11: while 循环语句

(b) for 循环语句



```
hello.py x
1  #!/usr/bin/python
2  # -*- coding: UTF-8 -*-
3
4  for letter in 'Python': # 第一个实例
5      print("当前字母: %s" % letter)
6
7  fruits = ['banana', 'apple', 'mango']
8  for fruit in fruits: # 第二个实例
9      print('当前水果: %s' % fruit)
10
11 print("Good bye!")
```

运行 hello x

D:\PythonProject\.venv\Scripts\python.exe D:\PythonProject\hello.py

当前字母: P  
当前字母: y  
当前字母: t  
当前字母: h  
当前字母: o  
当前字母: n  
当前水果: banana  
当前水果: apple  
当前水果: mango  
Good bye!

进程已结束, 退出代码为 0

图 12: for 循环语句

### 2.2.7 break 语句

(1) break 语句: Python break 语句, 就像在 C 语言中, 打破了最小封闭 for 或 while 循环。break 语句用来终止循环语句, 即循环条件没有 False 条件或者序列还没被完全递归完, 也会停止执行循环语句。break 语句用在 while 和 for 循环中。

```
1  #!/usr/bin/python
2  # -*- coding: UTF-8 -*-
3
4  for letter in 'Python': # 第一个实例
5      if letter == 'h':
6          break
7      print
8      '当前字母 :', letter
9
10 var = 10 # 第二个实例
11 while var > 0:
12     print ('当前变量值 :', var)
13     var = var - 1
14     if var == 5: # 当变量 var 等于 5 时退出循环
15         break
16
17 print ("Good bye!")
```

运行 hello x

D:\PythonProject\.venv\Scripts\python.exe D:\PythonProject\h

当前变量值 : 10  
当前变量值 : 9  
当前变量值 : 8  
当前变量值 : 7  
当前变量值 : 6  
Good bye!

进程已结束, 退出代码为 0

图 13: break 语句

(2) continue 语句: Python continue 语句跳出本次循环, 而 break 跳出整个循环。continue 语句用来告诉 Python 跳过当前循环的剩余语句, 然后继续进行下一轮循环。continue 语句用在 while 和 for 循环中。

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

for letter in 'Python': # 第一个实例
    if letter == 'h':
        continue
    print ('当前字母 :', letter)

var = 10 # 第二个实例
while var > 0:
    var = var - 1
    if var == 5:
        continue
    print ('当前变量值 :', var)
print ("Good bye!")
```

hello x

D:\PythonProject\.venv\Scripts\python.exe D:\PythonProject\h

当前字母 : P  
当前字母 : y  
当前字母 : t  
当前字母 : o  
当前字母 : n  
当前变量值 : 9  
当前变量值 : 8  
当前变量值 : 7  
当前变量值 : 6  
当前变量值 : 4  
当前变量值 : 3  
当前变量值 : 2  
当前变量值 : 1  
当前变量值 : 0

图 14: continue 语句

### 2.2.8 字符串

(1) Python 访问字符串中的值：Python 不支持单字符类型，单字符在 Python 中也是作为一个字符串使用。Python 访问子字符串，可以使用方括号来截取字符串，如下实例：



```
1  #!/usr/bin/python
2
3  var1 = 'Hello World!'
4  var2 = "Python Runoob"
5
6  print ("var1[0]: ", var1[0])
7  print ("var2[1:5]: ", var2[1:5])
```

运行 hello x

D:\PythonProject\.venv\Scripts\python.exe D:\PythonProject\h  
var1[0]: H  
var2[1:5]: ytho

图 15: 字符串访问

(2) Python 字符串连接：我们可以对字符串进行截取并与其他字符串进行连接



```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

var1 = 'Hello World!'

print ("输出 :- ", var1[:6] + 'Runoob!')
```

hello x

D:\PythonProject\.venv\Scripts\python.exe D:\PythonProject\h  
输出 :- Hello Runoob!

图 16: 字符串连接

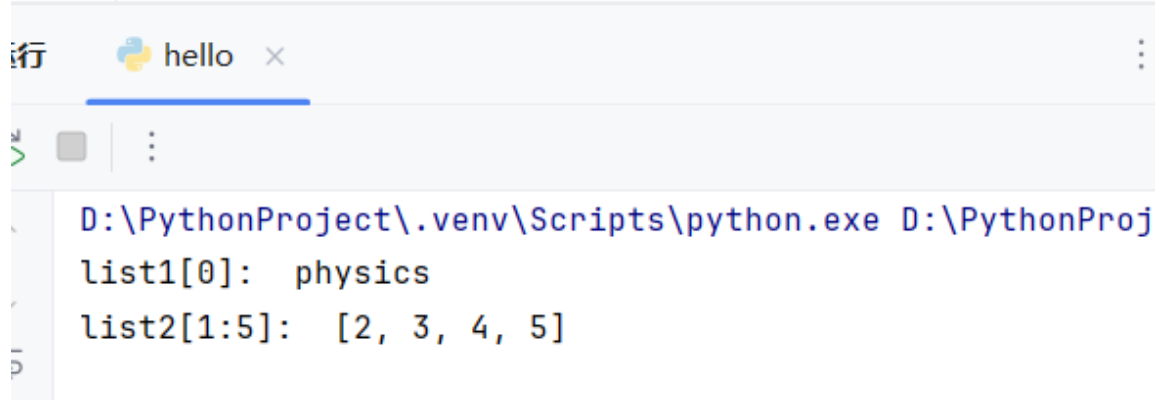


### 2.2.9 列表

序列是 Python 中最基本的数据结构。序列中的每个元素都分配一个数字 - 它的位置，或索引，第一个索引是 0，第二个索引是 1，依此类推。Python 有 6 个序列的内置类型，但最常见的是列表和元组。序列都可以进行的操作包括索引，切片，加，乘，检查成员。此外，Python 已经内置确定序列的长度以及确定最大和最小的元素的方法。列表是最常用的 Python 数据类型，它可以作为一个方括号内的逗号分隔值出现。列表的数据项不需要具有相同的类型创建一个列表，只要把逗号分隔的不同的数据项使用方括号括起来即可。

(1) 访问列表中的值

```
3 list1 = ['physics', 'chemistry', 1997, 2000]
4 list2 = [1, 2, 3, 4, 5, 6, 7]
5
6 print ("list1[0]: ", list1[0])
7 print ("list2[1:5]: ", list2[1:5])
```



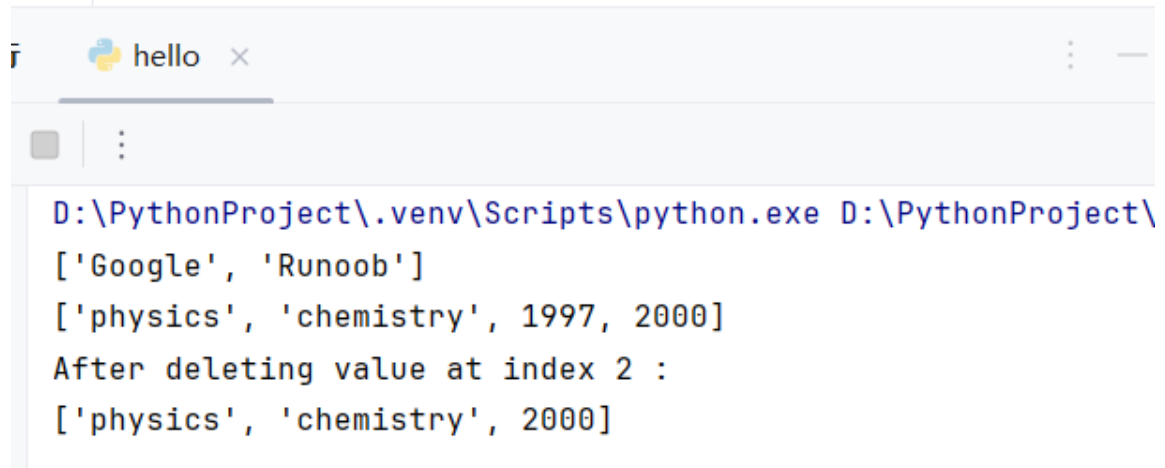
```
D:\PythonProject\.venv\Scripts\python.exe D:\PythonProj
list1[0]:  physics
list2[1:5]:  [2, 3, 4, 5]
```

图 17: 访问列表中的值

(2) 更新和删除列表中的元素

```
#更新列表
list = [] ## 空列表
list.append('Google') ## 使用 append() 添加元素
list.append('Runoob')
print (list)

#删除列表元素
list1 = ['physics', 'chemistry', 1997, 2000]
print (list1)
del list1[2]
print ("After deleting value at index 2 : ")
print (list1)
```



```
D:\PythonProject\.venv\Scripts\python.exe D:\PythonProject\
['Google', 'Runoob']
['physics', 'chemistry', 1997, 2000]
After deleting value at index 2 :
['physics', 'chemistry', 2000]
```

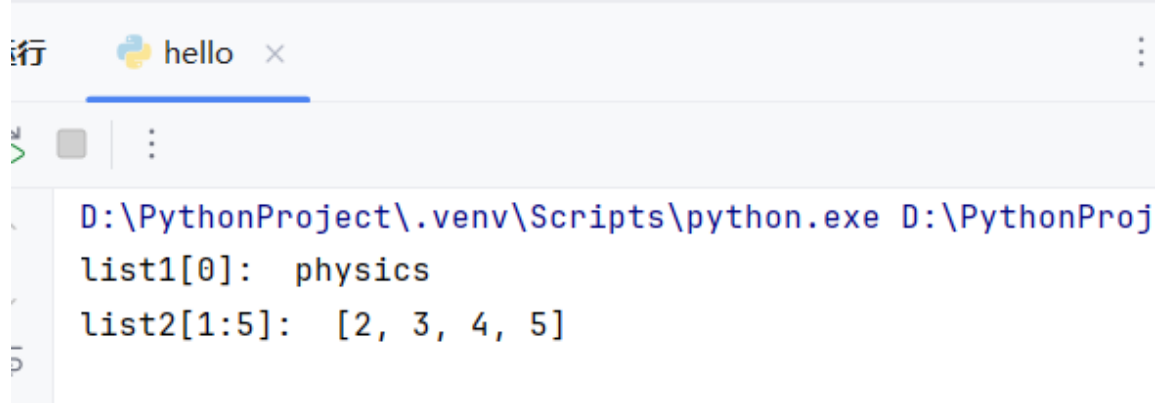
图 18: 更新和删除列表中的元素

### 2.2.10 元组

Python 的元组与列表类似，不同之处在于元组的元素不能修改。元组使用小括号，列表使用方括号。元组创建很简单，只需要在括号中添加元素，并使用逗号隔开即可。

#### (1) 访问元组

```
3 list1 = ['physics', 'chemistry', 1997, 2000]
4 list2 = [1, 2, 3, 4, 5, 6, 7]
5
6 print ("list1[0]: ", list1[0])
7 print ("list2[1:5]: ", list2[1:5])
```

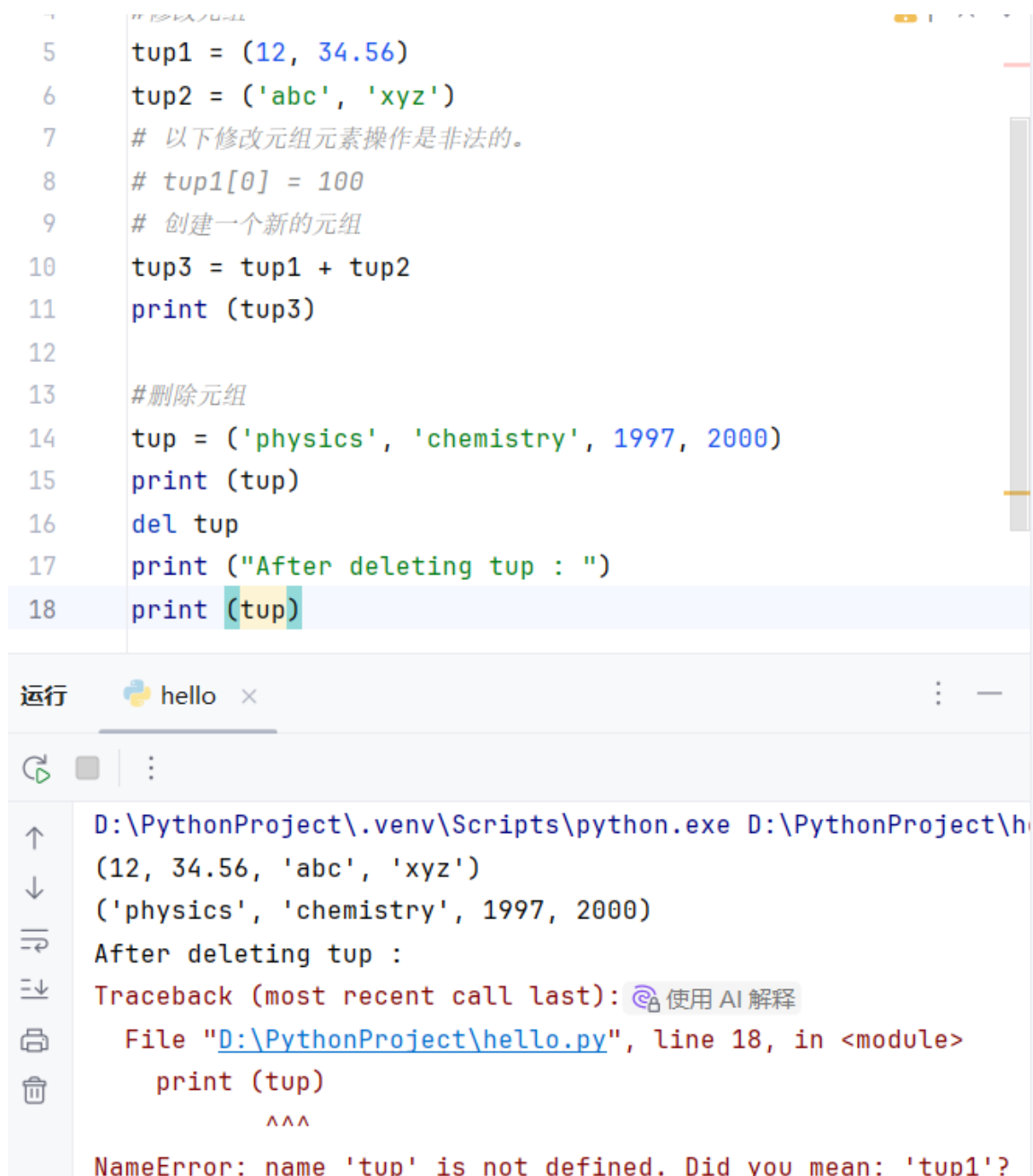


The screenshot shows a Python IDE with a file named 'hello'. The code defines two lists: `list1` containing `'physics', 'chemistry', 1997, 2000` and `list2` containing `1, 2, 3, 4, 5, 6, 7`. It then prints `list1[0]` and a slice of `list2` from index 1 to 5. The output in the console is:

```
D:\PythonProject\.venv\Scripts\python.exe D:\PythonProj
list1[0]:  physics
list2[1:5]:  [2, 3, 4, 5]
```

图 19: 访问列表中的值

## (2) 删除和修改元组



```
5 tup1 = (12, 34.56)
6 tup2 = ('abc', 'xyz')
7 # 以下修改元组元素操作是非法的。
8 # tup1[0] = 100
9 # 创建一个新的元组
10 tup3 = tup1 + tup2
11 print (tup3)
12
13 #删除元组
14 tup = ('physics', 'chemistry', 1997, 2000)
15 print (tup)
16 del tup
17 print ("After deleting tup : ")
18 print (tup)
```

运行 hello x

↑ ↓ ↺ ↻

```
D:\PythonProject\.venv\Scripts\python.exe D:\PythonProject\h
(12, 34.56, 'abc', 'xyz')
('physics', 'chemistry', 1997, 2000)
After deleting tup :
Traceback (most recent call last): 使用 AI 解释
  File "D:\PythonProject\hello.py", line 18, in <module>
    print (tup)
          ^^^
NameError: name 'tup' is not defined. Did you mean: 'tup1'?
```

图 20: 元组的修改与删除

### 2.2.11 字典

字典是另一种可变容器模型，且可存储任意类型对象。字典的每个键值 key:value 对用冒号: 分割，每个键值对之间用逗号, 分割，整个字典包括在花括号 {} 中，格式如下所示：d = key1 : value1, key2 : value2 (1) 访问字典元素



```
3 tinydict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
4
5 print ("tinydict['Name']: ", tinydict['Name'])
6 print ("tinydict['Age']: ", tinydict['Age'])
```

运行 hello x

D:\PythonProject\.venv\Scripts\python.exe D:\PythonProject\h  
tinydict['Name']: Zara  
tinydict['Age']: 7

图 21: 访问字典中的值

## (2) 删除和修改元组

```
4 #修改字典
5 tinydict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
6 tinydict['Age'] = 8 # 更新
7 tinydict['School'] = "RUNOOB" # 添加
8
9 print ("tinydict['Age']: ", tinydict['Age'])
10 print ("tinydict['School']: ", tinydict['School'])
11
12 #删除字典元素
13 tinydict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
14
15 del tinydict['Name'] # 删除键是'Name'的条目
16 print (tinydict)
17 tinydict.clear() # 清空字典所有条目
18 del tinydict # 删除字典
19
20 #print ("tinydict['Age']: ", tinydict['Age'])
21 #print ("tinydict['School']: ", tinydict['School'])
```

运行

hello x



```
D:\PythonProject\.venv\Scripts\python.exe D:\PythonProject\h
tinydict['Age']: 8
tinydict['School']: RUNOOB
{'Age': 7, 'Class': 'First'}
```

图 22: 字典的修改与删除

### 2.2.12 日期和时间

#### (1) 访问元组

```
3 list1 = ['physics', 'chemistry', 1997, 2000]
4 list2 = [1, 2, 3, 4, 5, 6, 7]
5
6 print ("list1[0]: ", list1[0])
7 print ("list2[1:5]: ", list2[1:5])
```

The screenshot shows a Python IDE with a file named 'hello'. The code in the editor is as follows:

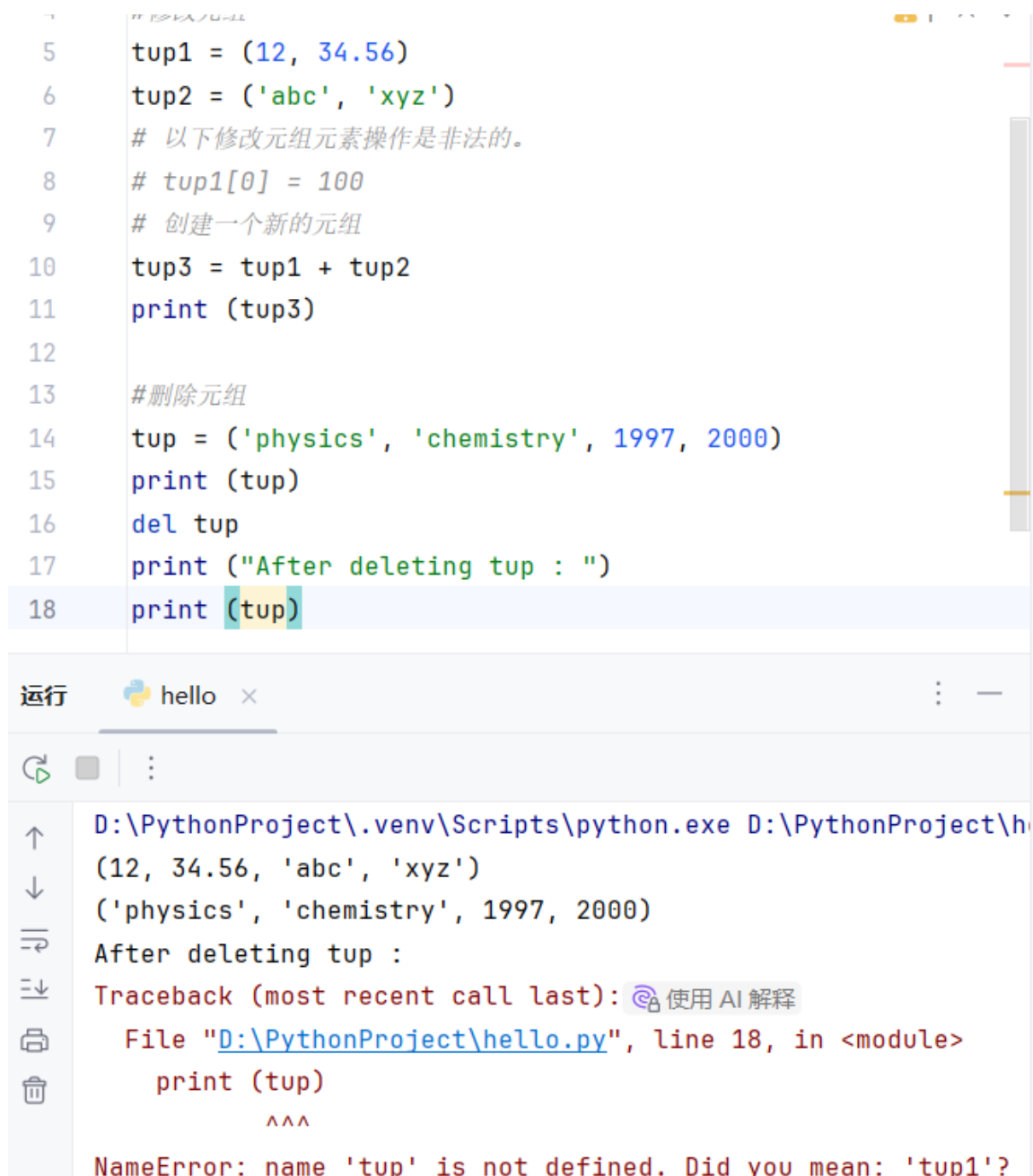
```
3 list1 = ['physics', 'chemistry', 1997, 2000]
4 list2 = [1, 2, 3, 4, 5, 6, 7]
5
6 print ("list1[0]: ", list1[0])
7 print ("list2[1:5]: ", list2[1:5])
```

The output of the program is displayed in the console:

```
D:\PythonProject\.venv\Scripts\python.exe D:\PythonProj
list1[0]:  physics
list2[1:5]:  [2, 3, 4, 5]
```

图 23: 访问列表中的值

## (2) 删除和修改元组



```
5 tup1 = (12, 34.56)
6 tup2 = ('abc', 'xyz')
7 # 以下修改元组元素操作是非法的。
8 # tup1[0] = 100
9 # 创建一个新的元组
10 tup3 = tup1 + tup2
11 print (tup3)
12
13 #删除元组
14 tup = ('physics', 'chemistry', 1997, 2000)
15 print (tup)
16 del tup
17 print ("After deleting tup : ")
18 print (tup)
```

运行 hello x

D:\PythonProject\.venv\Scripts\python.exe D:\PythonProject\h  
(12, 34.56, 'abc', 'xyz')  
( 'physics', 'chemistry', 1997, 2000)  
After deleting tup :  
Traceback (most recent call last): 使用 AI 解释  
File "D:\PythonProject\hello.py", line 18, in <module>  
 print (tup)  
 ^^^  
NameError: name 'tup' is not defined. Did you mean: 'tup1'?

图 24: 元组的修改与删除

### 2.2.13 编辑模式

### 2.2.14 时间

Python 程序能用很多方式处理日期和时间，转换日期格式是一个常见的功能。Python 提供了一个 `time` 和 `calendar` 模块可以用于格式化日期和时间。时间间隔是以秒为单位的浮点小数。每个时间戳都以自从 1970 年 1 月 1 日午夜（历元）经过了多长时间来表示。Python 的 `time` 模块下有很多函数可以转换常见日期格式。(1) 如函数 `time.time()` 用于获取当前时间戳



```
3 list1 = ['physics', 'chemistry', 1997, 2000]
4 list2 = [1, 2, 3, 4, 5, 6, 7]
5
6 print ("list1[0]: ", list1[0])
7 print ("list2[1:5]: ", list2[1:5])
```

运行 hello x

D:\PythonProject\.venv\Scripts\python.exe D:\PythonProject\h  
list1[0]: physics  
list2[1:5]: [2, 3, 4, 5]

图 25: 访问列表中的值

(2) 获取本地时间以及时间格式化

```
import time # 引入time模块
#获取当前时间
localtime = time.localtime(time.time())
print ("本地时间为 :", localtime)

#获取格式化时间
localtime = time.asctime( time.localtime(time.time()) )
print ("本地时间为 :", localtime)
```

运行 hello x

D:\PythonProject\.venv\Scripts\python.exe D:\PythonProject\h  
本地时间为 : time.struct\_time(tm\_year=2025, tm\_mon=9, tm\_mday=15, tm\_mhour=16, tm\_min=06, tm\_sec=43)  
本地时间为 : Mon Sep 15 16:06:43 2025

图 26: 获取本地时间以及时间格式化

### 2.2.15 日期

Python 程序能用很多方式处理日期和时间，转换日期格式是一个常见的功能。Python 提供了一个 `time` 和 `calendar` 模块可以用于格式化日期和时间。(1) 格式化日期格式

```
import time

# 格式化2016-03-20 11:45:39形式
print (time.strftime( format: "%Y-%m-%d %H:%M:%S", time.localtime()))

# 格式化Sat Mar 28 22:24:24 2016形式
print (time.strftime( format: "%a %b %d %H:%M:%S %Y", time.localtime()))

# 将格式字符串转换为时间戳
a = "Sat Mar 28 22:24:24 2016"
print (time.mktime(time.strptime(a, format: "%a %b %d %H:%M:%S %Y")))
```



```
hello x
D:\PythonProject\.venv\Scripts\python.exe D:\PythonProject\hello.py
2025-09-15 16:10:52
Mon Sep 15 16:10:52 2025
1459175064.0
```

图 27: 格式化日期

(2) 获取某月日历

```
4 import calendar
6 cal = calendar.month( theyear: 2025, themonth: 9)
7 print ("以下输出2025年9月份的日历:")
8 print (cal)
```

运行 hello x

D:\PythonProject\.venv\Scripts\python.exe D:\PythonProject\hello.py

以下输出2025年9月份的日历:

September 2025

Mo	Tu	We	Th	Fr	Sa	Su
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

图 28: 获取某月日历

## 2.3 Python 计算机视觉



图 29: 原图

### 2.3.1 Pillow

```
from PIL import Image, ImageEnhance
img_original = Image.open("dark.jpg")
img_original.show("Original Image")
img = ImageEnhance.Contrast(img_original)
img.enhance(3.8).show("Image With More Contrast")
```

图 30: 处理代码

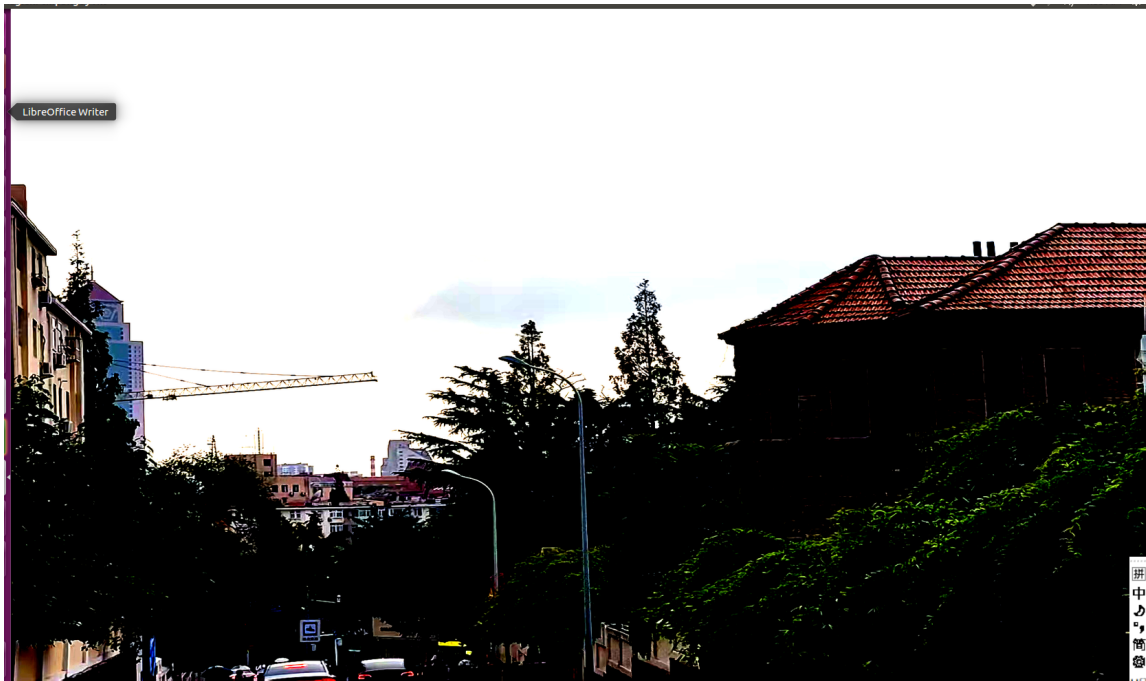


图 31: 运行效果图

### 2.3.2 Numpy

```
from PIL import Image
import numpy as np
img = np.array(Image.open('0.jpg'))
img_red = img.copy()
img_red[:, :, (1, 2)] = 0
img_green = img.copy()
img_green[:, :, (0, 2)] = 0
img_blue = img.copy()
img_blue[:, :, (0, 1)] = 0
img_ORGB = np.concatenate((img, img_red, img_green, img_blue), axis=1)
img_converted = Image.fromarray(img_ORGB)
img_converted.show() ## Combine Image Contains all four images

~
~
~
```

图 32: numpy 代码

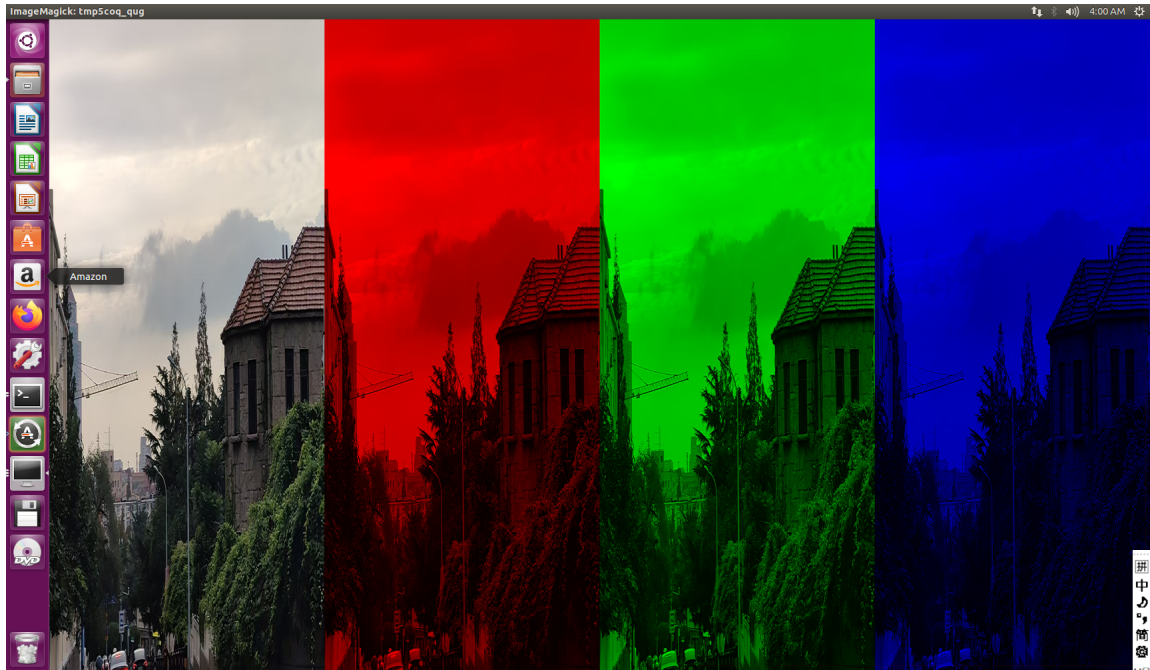


图 33: numpy 操作效果图



## 2.3.3 Scipy

```
import matplotlib.pyplot as plt
from scipy import ndimage, misc
import numpy as np

# 创建图形
fig = plt.figure()
plt.gray() # 以灰度显示

# 创建子图
ax1 = fig.add_subplot(121) # 左侧子图
ax2 = fig.add_subplot(122) # 右侧子图

# 尝试获取测试图像，如果不可用则创建一个简单的测试图像
try:
    img = misc.ascent()
except AttributeError:
    # 如果misc.ascent不可用，创建一个简单的测试图像
    x, y = np.meshgrid(np.linspace(0, 1, 100), np.linspace(0, 1, 100))
    img = np.sin(10*x) * np.cos(10*y) * 255
    img = img.astype(np.uint8)

# 应用高斯滤波
result = ndimage.gaussian_filter(img, sigma=5)

# 显示图像
ax1.imshow(img)
ax1.set_title('Original Image')

ax2.imshow(result)
ax2.set_title('Gaussian Filtered')

plt.show()
~
~
```

图 34: 获取某月日历



图 35: 获取某月日历

### 3 心得体会

通过本次实验，我收获颇丰，不仅掌握了 Shell 工具和脚本编程的基本技能，还深入学习了 Python 编程语言及其在计算机视觉中的应用。在 Shell 工具和脚本学习过程中，我认识到命令行环境的高效性和强大功能。在 Python 学习方面，我从基础语法到高级特性进行了系统性的实践。通过编写和调试各种类型的 Python 代码，我不仅加深了对 Python 语言特性的理解，更重要的是培养了编程思维和解决问题的能力。特别是对于 Python 独特的特点，如缩进规则、动态类型、丰富的内置数据结构等，有了更加直观和深刻的认识。在计算机视觉部分，通过实际操作 Pillow、Numpy 和 Scipy 等库，我体验到了 Python 在图像处理领域的强大能力。从基本的图像裁剪、缩放，到复杂的图像滤波和处理，我感受到了计算机视觉技术的魅力，也为将来可能的环境监测图像处理应用奠定了基础。这次实验培养了我系统开发的基础能力。



从环境配置、工具使用到代码编写和调试，我经历了完整的开发流程，这对我综合能力的提升有很大帮助。特别是在解决各种安装和运行问题的过程中，我提高了问题排查和解决的能力。通过这次实验，我不仅掌握了技术知识，更重要的是学会了如何学习和探索新技术。我相信这些经验和技能将对我未来的专业学习和科研工作产生积极影响，特别是在环境工程领域的数据处理和监测分析方面。

## 4 实验代码查看链接

本次报告相关练习、报告和代码均可以在 <https://github.com/chen2-spec/my-latex-report> 查看