系 统 开 发 工 具 实 验 报 告

所 在 学 院：信息科学与工程学部　专　　业：计算机科学与技术

学 生 姓 名：陈一韬　　学　　号：23010022003

2024/8/27

主　题：Python 基础与 Python 图像处理

1　实验目的

1. 学习 Python 的基本语法与相关数据类型的使用
2. 学习使用 Python 相关库实现图像处理

2　课后实例练习

Pyhon 基本编程部分

注：python 编程与先前学过的 C++ 编程有许多共通点，所以这篇报告更聚焦于二者的差异

1. def 用于定义函数。
   不同于 C++ 的函数，Python 中定义函数和变量时都不用声明类型，主机会自动判断。这种特性使得在写代码时效率很高

```
def BMI(weight, height):
    bmi = weight / (height ** 2)
    print("BMI指数: " + str(bmi))
```

2. print() 函数，用于输出内容
   Python 语言的 print 函数相较于 C++ 的 cout 更加灵活，因为 python 会自动判断输出的内容类型，并且会在输出语句后自动加上换行这种高封装度省去了很多繁琐的步骤。

```
def BMI(weight, height):
    bmi = weight / (height ** 2)
    print("BMI指数: " + str(bmi))
```

3. 条件选择语句 if · · · elif · · · else
   python 的条件选择语句与 C++ 的书写很相似，但在条件判断方面，python 可以判断如 0 < x < 100 这样的连续不等号，并且 python 靠缩进来规范判断语句后的内容，而 C++ 则用大括号

```python
if bmi < 18.5:
    print("偏瘦")
elif 18.5 <= bmi < 25:
    print("正常")
elif 25 <= bmi < 30:
    print("偏胖")
else:
    print("肥胖")
return bmi
```

4. try · · · except

   在 Python 中，try 语句用于处理异常（也称为错误），它允许程序在遇到运行时错误时能够继续运行，而不是直接终止。使用 try 语句，可以捕获和处理可能出现的异常，这相较于 C++ 来说在 debug 方面强了许多

```python
try:
    input_weight = float(input("请输入体重(kg):"))
    input_height = float(input("请输入身高(m):"))
    result = BMI(input_weight, input_height)
except ValueError:
    print("请输入有效的数字。")
```

5. 循环语句 for i in range (x,y)

   Python 的 for 循环语法更清晰并且和与迭代器紧密结合，使得代码通常更加简洁和易读。而 C++ 的 for 循环则给程序员更多的控制和灵活性，适用于更复杂的逻辑，但相对较冗长且容易出错。

```python
for _ in range(60):  # 对于每种卡牌
    for j in range(n, -1, -1):  # 从后向前遍历
        for k in range(1, max_per_card + 1):  # 选择 1 到 max_per_card
            if j >= k:
                dp[j] += dp[j - k]
```

6. 字典类型

   字典数据类型引入了“键：值”对这一经典的概念，使得检索变得十分简单。并且与元组的相结合更让字典的应用场景更加灵活。在 C++ 中少有类似的数据结构。

```
pokemon_dict ={"妙蛙种子":"草宝可梦，初代御三家，全国编号001",
               "小火龙":"火龙宝可梦，初代御三家，全国编号004",
               "杰尼龟":"水宝可梦，初代御三家，全国编号007",
               "皮卡丘":"黄皮耗子，小智御用宝可梦，全国编号025"}
query = input("请输入您要查询的宝可梦：")
if query in pokemon_dict:
```

7. len() 函数返回对象（如字符串、列表、字典等）的元素数量。

   在 C++ 中，对不同的数据类型往往要采用不同的方法来获取其元素长度。如对 vector 的.size 方法和对字符串的 strlen() 方法；而在 python 中 len() 可返回多种对象的元素数量

```
random_list = generate_random_list(size, lower_bound, upper_bound)
print("随机生成的整数列表:", random_list)
print("列表长度:", len(random_list))  # 使用 len 函数
```

8. sum() 函数返回可迭代对象中所有元素的总和。

   C++ 没有内建的 sum() 函数，通常需要使用 std::accumulate 函数需要包含 <numeric> 头文件。

```
evens = list(filter(lambda x: x % 2 == 0, numbers))
# 使用 map 计算偶数的平方
squares = list(map(lambda x: x ** 2, evens))
# 使用 sum 计算平方和
return sum(squares)
```

9. map() 函数返回一个迭代器，使得多个输入的每个元素都通过指定的函数进行处理，适合用于函数式编程风格。

   C++ 没有直接的 map() 函数，但是可以使用 std::transform 来进行类似的操作。

```
# 使用 map 计算偶数的平方
squares = list(map(lambda x: x ** 2, evens))
```

10. filter() 函数创建一个迭代器，其中包含通过了测试的元素。常用于过滤可迭代对象。

    C++ 也没有类似的内建 filter() 函数，但可以使用 std::copy_if 结合 STL 容器来实现。

```python
def calculate_even_squares_sum(numbers):
    # 使用 filter 过滤出偶数
    evens = list(filter(lambda x: x % 2 == 0, numbers))
```

Python 图像处理部分

1. cv2.imread（）

```python
import cv2

# 读取图像
image = cv2.imread('image.png')  # 确保这个路径存在一个图像文件
```

filename：要读取的图像文件的路径。
flags：可选参数，用于指定图像读取的模式。常用的标志包括：
cv2.IMREAD_COLOR：加载彩色图像。图像的透明度将被忽略，即使图像文件包含 alpha 通道，也会被忽略。这是默认模式。
cv2.IMREAD_GRAYSCALE：以灰度模式加载图像。
cv2.IMREAD_UNCHANGED：加载图像的 alpha 通道。如果图像文件包含 alpha 通道，则该通道将被加载。

2. cv2.imshow(window_name, image)：
在新窗口中显示图像。

```python
# 显示缩小的图像
cv2.imshow( winname: 'Resized Image', resized_image)
```

3. cv2.waitKey(delay)：
等待指定的毫秒数，直到用户按下某个键。如果不指定，窗口会一直保持。

```python
# 等待用户按键
print("按 'r' 将处理图像，按 'q' 退出。")
key = cv2.waitKey(0)
```

4. cv2.destroyAllWindows()：
关闭所有创建的窗口。

```
            print("程序退出。")
    else:
        print("程序退出。")
else:
    print("程序退出。")

# 关闭所有窗口
cv2.destroyAllWindows()
```

5. cv2.cvtColor(image, code)：

   将图像从一种颜色空间转换为另一种。例如，从 BGR 转换为灰度。

```
if key == ord('g'):
    # 转换为灰度图像
    gray_image = cv2.cvtColor(resized_image, cv2.COLOR_BGR2GRAY)
    cv2.imshow( winname: 'Gray Image', gray_image)
```

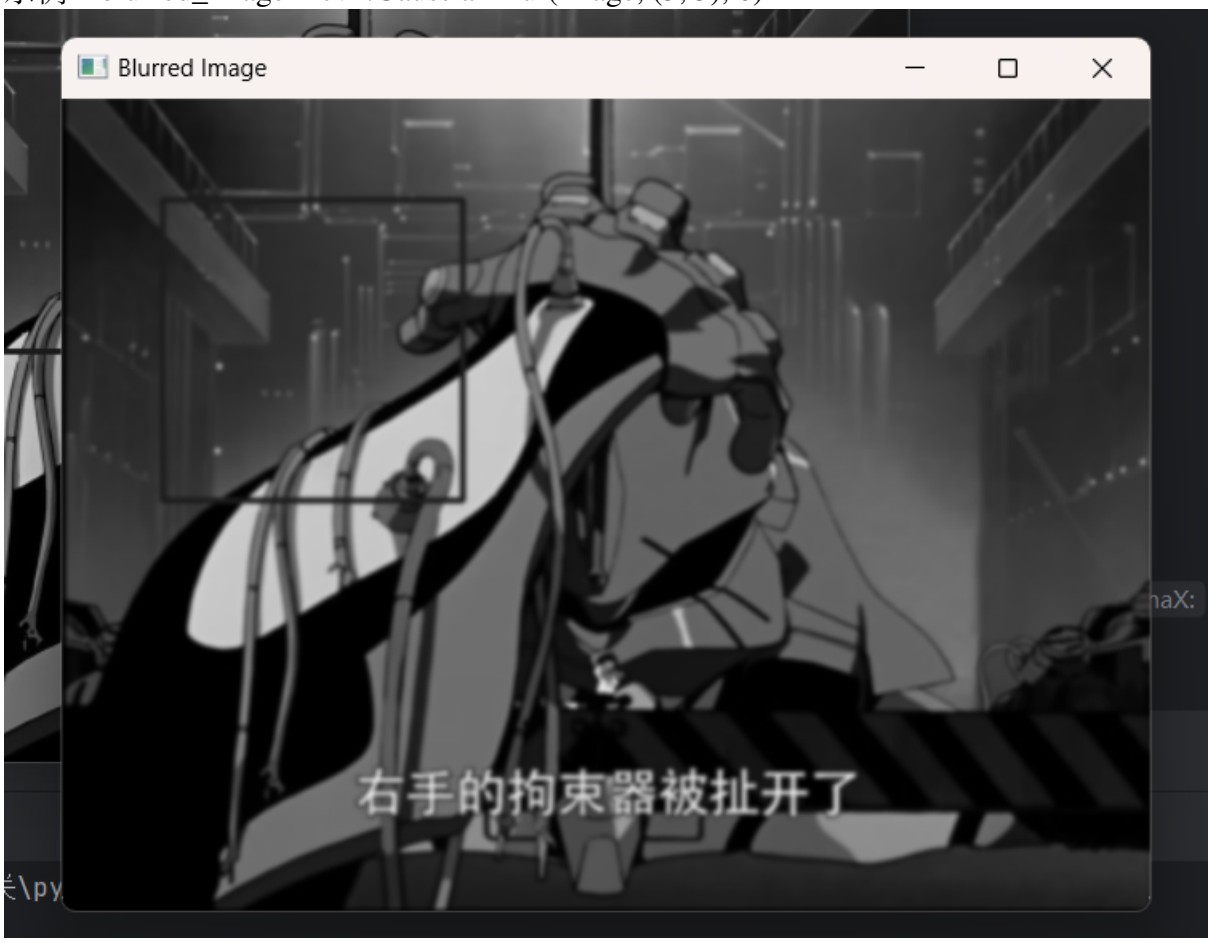
   示例：gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

6. cv2.GaussianBlur(src, ksize, sigmaX)：

使用高斯滤波器对图像进行模糊处理。

```
if key == ord('b'):
    # 使用高斯模糊处理
    blurred_image = cv2.GaussianBlur(gray_image, ksize: (5, 5), sigmaX:
    cv2.imshow( winname: 'Blurred Image', blurred_image)
```

示例：blurred_image = cv2.GaussianBlur(image, (5, 5), 0)

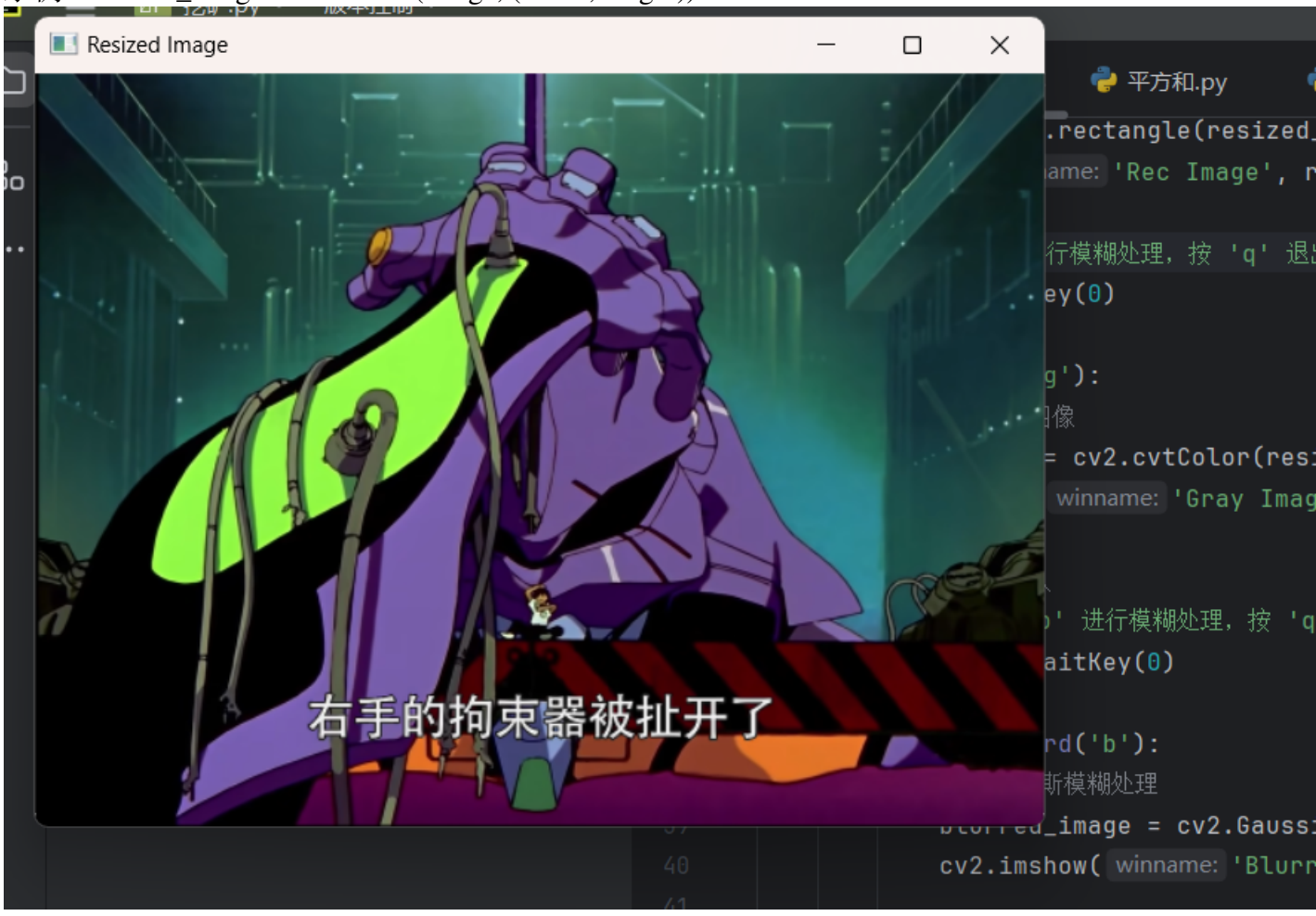


7. cv2.resize(src, dsize)：

调整图像的大小。

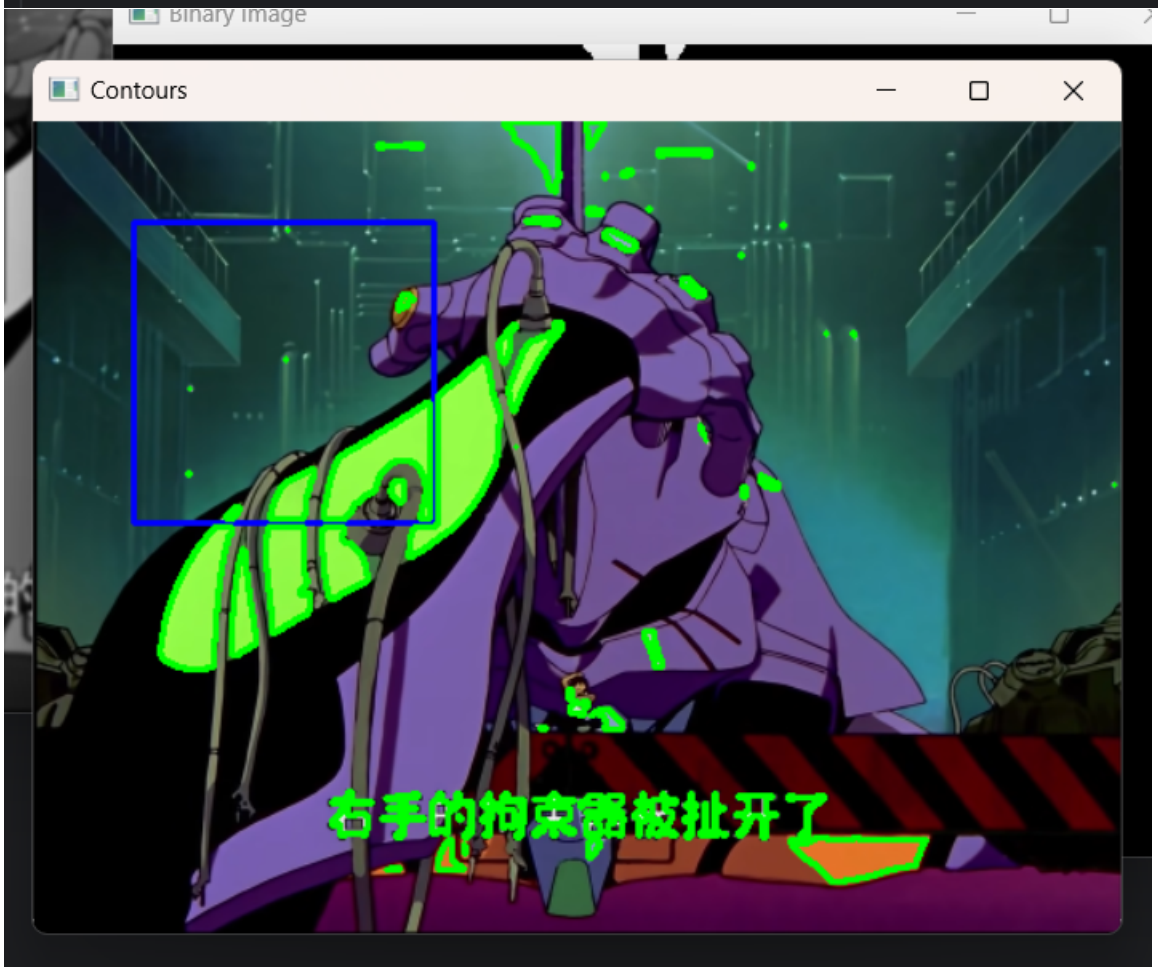
```python
# 缩小图像
scale_percent = 50  # 按照50%缩小
width = int(image.shape[1] * scale_percent / 100)
height = int(image.shape[0] * scale_percent / 100)
dim = (width, height)

resized_image = cv2.resize(image, dim, interpolation=cv2.INTER_AREA)

# 显示缩小的图像
cv2.imshow( winname: 'Resized Image', resized_image)
```

示例：resized_image = cv2.resize(image, (width, height))



8. cv2.findContours(image, mode, method)：
   找到图像中的轮廓，并返回轮廓的列表。

```python
if key == ord('c'):
    # 查找轮廓
    contours, _ = cv2.findContours(binary_image, cv2.RETR_EXTERNAL, cv
    # 在缩小的图像上绘制轮廓
    cv2.drawContours(resized_image, contours, -1, color: (0, 255, 0),
    cv2.imshow( winname: 'Contours', resized_image)
```



9. cv2.threshold(src, thresh, maxval, type)：
   应用阈值操作，转换为二值图像。

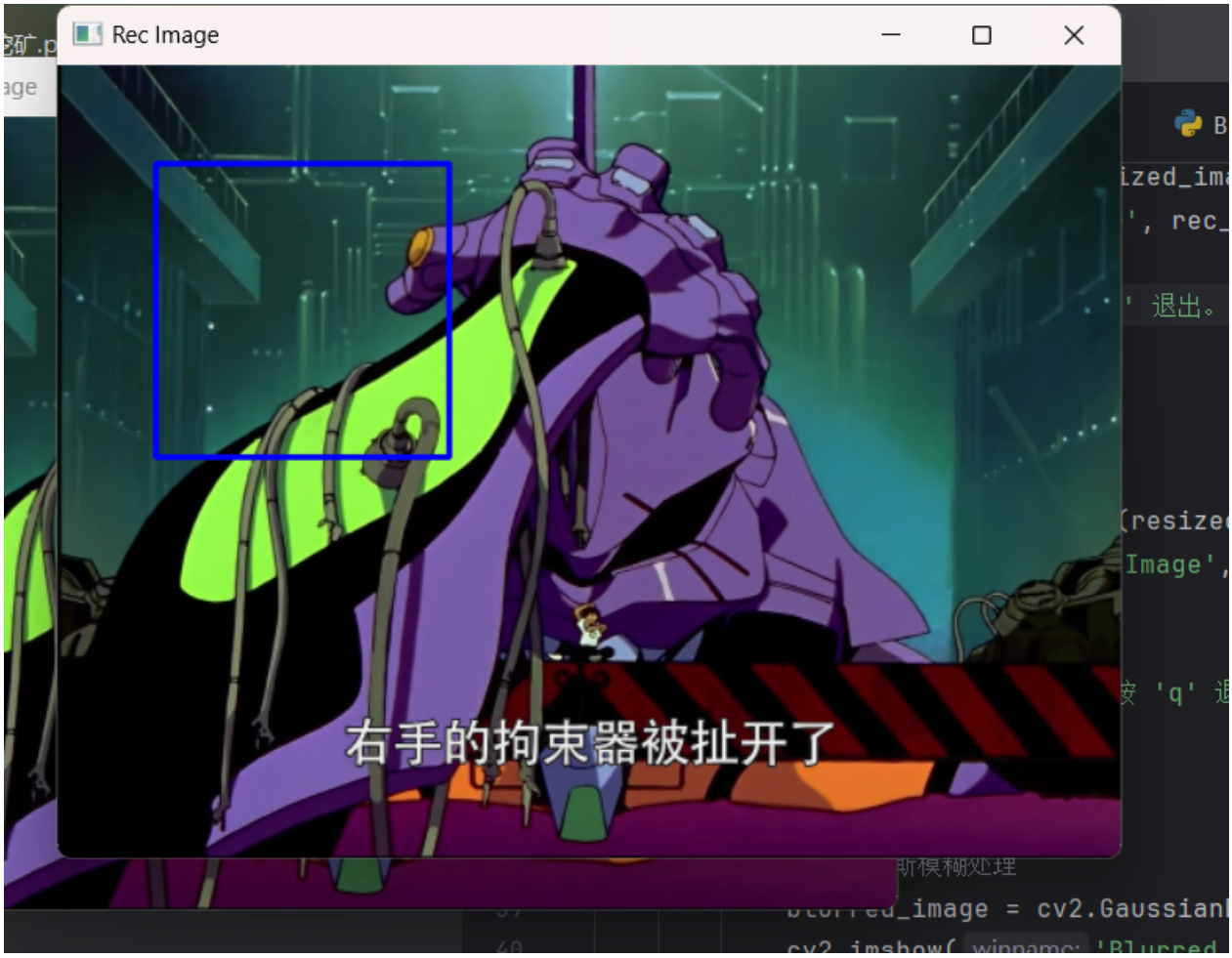
```python
if key == ord('t'):
    # 应用阈值
    _, binary_image = cv2.threshold(blurred_image, thresh: 127, maxval: 2
    cv2.imshow( winname: 'Binary Image', binary_image)
```

   示例：_, binary_image = cv2.threshold(gray_image, 127, 255, cv2.THRESH_BINARY)

10. cv2.rectangle(img, pt1, pt2, color, thickness)：
在图像上绘制矩形框。

```
if key == ord('r'):
    #在图像上绘制方框
    rec_image = cv2.rectangle(resized_image, (50, 50), (200, 200), (255
    cv2.imshow( winname: 'Rec Image', rec_image)
```

示例：cv2.rectangle(image, (50, 50), (200, 200), (255, 0, 0), 2)



3 解题感悟

在完成了这次关于 Python 基础入门、Python 视觉应用的实例操作后我深刻体会到了 Python 作为一门强大而灵活的编程语言所带来的无限可能. 以下是我对这次学习的一些感悟与体会:

一、Python 入门的惊鸿一瞥

Python 以其简洁明了的语法、丰富的库支持和强大的社区资源，迅速成为了我学习编程路上的得力助手。从最初的变量定义、数据类型、控制结构到函数定义与调用，每一步都显得那么自然流畅，仿佛是在与一位老朋友对话。这种低门槛的入门体验，极大地激发了我对编程的兴趣和热情。

在解题过程中，我深刻感受到了 Python 在数据处理、字符串操作以及文件读写等方面的便捷性。尤其是其内置的字典推导式等高级特性，让我在解决复杂问题时能够事半功倍，体验到编程带来的乐趣和成就感。

二、视觉应用的探索之旅

当课程深入到视觉应用领域时，我仿佛打开了一个全新的世界。通过 Python 的图像处理库（OpenCV），我能够轻松地对图像进行加载、显示、处理和分析。从简单的图像缩放、转换到复杂的边缘检测、特征提取，每一次实践都让我对计算机视觉有了更深的理解。

更令我兴奋的是，学习到通过结合机器学习库（如 scikit-learn、TensorFlow 或 PyTorch），我还能够尝试构建简单的图像分类模型，实现图像内容的自动识别与分类。可惜时间过短，没能完全掌握这一应用，希望未来能有机会对 Python 视觉相关的库进行综合运用。

三、Python 与 C、C++ 的对比感悟

回顾之前学习 C、C++ 的经历，我深刻感受到了 Python 与这两种语言在多个方面的显著差异。

首先，在语法层面，Python 的简洁性远胜于 C、C++。Python 的语法更加直观易懂，减少了大量繁琐的语法规则和细节处理，使得开发者能够更专注于算法逻辑本身。而 C、C++ 则以其严谨性和高效性著称，但这也意味着需要编写更多的代码来处理各种边界情况和性能优化。

其次，在库支持方面，Python 拥有庞大的第三方库生态系统，几乎涵盖了所有常见的编程需求。这些库经过精心设计和优化，能够极大地提高开发效率和质量。相比之下，C、C++ 虽然也有丰富的库资源，但往往需要开发者自行搜索、筛选和整合。

最后，在应用领域上，Python 因其简单易学、功能强大的特点，在数据分析、机器学习、Web 开发等多个领域得到了广泛应用。而 C、C++ 则因其高效性和可移植性，在操作系统、游戏开发、嵌入式系统等对性能要求极高的领域占据主导地位。

总之，这次系统开发工具课程的学习经历让我对 Python 有了更深入的了解和认识。在未来的学习和工作中，我将继续探索 Python 的无限可能，同时也将保持对 C、C++ 等语言的关注和学习，以便更好地应对各种编程挑战

本次实验报告的 github 地址：https://github.com/Shiny-Magikarp/-.git(相关.py 文件也已上传）