

系统开发工具基础第三次实验报告

张烨 23020007162

2024 年 9 月 12 日

目录

一、 实验目的	3
二、 实验内容	3
(一) python	3
1. python 基本语句	3
2. 导入包（库）	6
3. if 语句	6
4. 循环语句 while、for、break、continue、pass	8
5. 数据类型和数据类型转换	10
6. 字符串	11
7. 列表	12
8. 元组	13
9. 字典	15
10. 集合	17
11. 迭代器与生成器	19
12. 日期	21
13. 函数	22
14. 文件	22
15. os 模块	23
16. 面向对象	23
(二) python 计算机视觉	24
1. PIL:Python 图像处理类库	24

目录	2
2. Matplotlib	25
3. NumPy 将图像转化为数组	27
4. SciPy 图像模糊	27
三、 心得体会	28
四、 相关练习、报告和代码查看链接	28

一、实验目的

1. 学习 python, 掌握 python 基础语法, 能独立完成 python 基础练习。
2. 学习 python 计算机视觉编程。

二、实验内容

(一) python

1. python 基本语句

1. 注释

注释的方法有行注释和块注释

行注释以#开头

```
#这是注释
```

图 1: 行注释

块注释可以用多个#、三单引号、三双引号

```
#这  
#块  
#是  
#注  
#释  
  
....  
这块是注释  
....  
  
....  
这块是注释  
....
```

图 2: 块注释

2. 输出语句在 python 中, 用 print 输出语句, 下面来输出一行 helloworld

```
print("Hello world!")
```

图 3: helloworld

```
D:\pythonProject4\venv\Scripts\python.exe D:\pythonProject4\3.1.py
Hello world!
```

图 4: helloworld

3. 标识符

- (1) 标识符的第一个字符必须是字母表中的字母或下划线
- (2) 标识符的其他部分由字母、数字、下划线组成
- (3) 标识符对大小写敏感

```
a="早上吃什么？"
b="早上吃三明治"

print(a)
print(b)
print(a+b)
```

图 5: 标识符

```
早上吃什么？
早上吃三明治
早上吃什么？早上吃三明治
```

图 6: 标识符

4. 多行语句

在编写代码中,如果变量名很长,可以用反斜杠\来实现多行语句。在[]、{}、()里的多行语句,不需要使用反斜杠\

```
text1="明天天气"
text2="怎么样？"
text3="是晴天"
text4="还是雨天？"
print(text1+\n
      text2+\n
      text3+\n
      text4)

list=[text1+text2+text3+text4]
print(list)
```

图 7: 多行语句

```
明天天气怎么样？是晴天还是雨天？
['明天天气怎么样？是晴天还是雨天？']
```

图 8: 多行语句

5. 行与缩进

在 python 中，用缩进来表示代码块，要保证正确的缩进

6. 关键字

python 中有 33 个关键字，其中 False、None、True 的首字母大写，其他还

有 class、finally、is、return、continue、for、lambda、try、def、from、nonlocal、while、and、del、global

7. 数据类型

python 中的数据类型有字符串、整型、列表、元组、字典、布尔型等。

跟之前学过的 c 语言和 c++ 不同，以整型为例：在 python 中有两种写法：

```
counter = 100
```

```
counter = int(100)
```

浮点型、字符串型也类似

布尔型是整型的子类型，只有两个取值——True 和 False，对应的整型分别为 1 和 0

8. 运算符

表 1: 运算符

符号	描述
==	等于
!=	不等于
<=	小于等于
>=	大于等于
>	大于
<	小于
and	与
or	或
not	非

关于逻辑运算：

- (1) 与运算 and 一假则假
- (2) 或运算 or 一真则真
- (3) 非运算 not 真假倒转

2. 导入包（库）

在 python 中用 import 或者 from...import 来导入相应的模块

将整个模块 (somemodule) 导入，格式为：import somemodule

从某个模块中导入某个函数，格式为：from somemodule import somefunction

从某个模块中导入多个函数，格式为：from somemodule import firstfunc, secondfunc, thirdfunc

将某个模块中的全部函数导入，格式为：from somemodule import *

将某个模块改名 (改为 s)，格式为：import somemodule as s

3. if 语句

1.if 判断语句格式为：

if:

 执行语句

下面是一个例子：

```
a=0  
b=1  
if a<b:  
    print(a,"<",b)
```

图 9: if 判断语句

```
0 < 1
```

图 10: if 判断语句

2.if else 分支语句格式为:

```
if:  
    执行语句
```

```
else:  
    执行语句
```

下面是一个例子:

```
a=0  
b=1  
if a<b:  
    print(a,"<",b)  
else:  
    print(a,">",b)
```

图 11: if 分支语句

```
0 < 1
```

图 12: if 分支语句

3.if elif else 多分支语句格式为:

```
if:  
    执行语句
```

```
elif:  
    执行语句
```

```
else:  
    执行语句
```

下面是一个例子:

```

num = 5
if num == 3:
    print('boss')
elif num == 2:
    print('user')
elif num == 1:
    print('worker')
elif num < 0:
    print('error')
else:
    print('roadman')

```

图 13: if 多分支语句

roadman

图 14: if 多分支语句

因为 python 不支持 switch 语句，所以多个条件判断，只能用 elif 来实现

4. 循环语句 while、for、break、continue、pass

1. while 循环语句

(1) while 循环语句

下面是一个例子：

```

count = 0
while count < 9:
    print('The count is:', count)
    count = count + 1
print("Goodbye!")

```

图 15: while 循环语句

```

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
Goodbye!

```

图 16: while 循环语句

(2) while 循环语句 else

下面是一个例子：

```
count = 0
while count < 5:
    print(count, "is less than 5")
    count = count + 1
else:
    print(count, "is not less than 5")
```

图 17: while 循环语句 else

```
0 is less than 5
1 is less than 5
2 is less than 5
3 is less than 5
4 is less than 5
5 is not less than 5
```

图 18: while 循环语句 else

2. for 循环

(1) for 循环语句

下面是两个例子：

```
for letter in 'Python':#迭代字符串中的每个字母
    print("当前字母:%s" % letter)

fruits = ['banana', 'apple', 'mango']#迭代水果列表中的每个水果
for fruit in fruits:
    print('当前水果:%s' % fruit)
print("Goodbye!")
```

图 19: for 循环

```
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
Goodbye!
```

图 20: for 循环

(2) for 循环语句 else

下面是一个例子：

```
for num in range(10,20):
    for i in range(2,num):
        if num%i == 0:
            j=num/i
            print('%d 等于 %d * %d' %(num,i,j))
            break
    else:
        print('%d 是一个质数' % num)
```

图 21: for 循环 else

```
10 等于 2 * 5
11 是一个质数
12 等于 2 * 6
13 是一个质数
14 等于 2 * 7
15 等于 3 * 5
16 等于 2 * 8
17 是一个质数
18 等于 2 * 9
19 是一个质数
```

图 22: for 循环 else

(3) break 语句用来终止循环语句，而 continue 语句用来跳过当前循环的剩余语句，然后继续进行下一轮循环，此外还有一个 pass 语句，但是 pass 语句是空语句，不做任何事情，一般用做占位语句

5. 数据类型和数据类型转换

整型 int, 长整型 long, 浮点型 float, 复数 complex。其中，复数由实数部分和，虚数部分构成，可以用 $a+bi$ 或者 $complex(a,b)$ 表示， a 是实部， b 是虚部。

数据类型转换：

- int(x) 将 x 转换为一个整数
- long(x) 将 x 转换为一个长整数
- float(x) 将 x 转换为一个浮点数
- str(x) 将 x 转换为字符串
- list(s) 将序列 s 转换为一个列表
- ...

6. 字符串

字符串是 python 中最常用的数据类型，使用单引号或双引号来创建字符串。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，可以用方括号 [] 来截取字符串。

表 2: 字符串运算符

符号	描述
+	字符串连接
*	重复输出字符串
[]	通过索引获取字符串中字符
[:]	截取字符串中的一部分，遵循左闭右开原则，例如 str[0:2] 是不包含第 3 个字符的
in	成员运算符—如果字符串中包含给定的字符返回 True
not in	成员运算符—如果字符串中不包含给定的字符串返回 True
r/R	原始字符串 - 原始字符串：所有的字符串都是直接按照字面的意思来使用，没有转义特殊或不能打印的字符。原始字符串除在字符串的第一个引号前加上字母 r（可以大小写）以外，与普通字符串有着几乎完全相同的语法。

下面是一些例子：

```
a = "python3.0"
b = "python"
print("a + b输出结果: ", a + b)
print("a * 2输出结果: ", a * 2)
print("a[1]输出结果: ", a[1])
print("a[1:4]输出结果: ", a[1:4])
if("H" in a):
    print("H在变量a中")
else:
    print("H不在变量a中")
if("M" not in a):
    print("M不再变量a中")
else:
    print("M在变量a中")
print(r'\n')
print(R'\n')
```

图 23: 字符串运算符

```
a + b输出结果:  python3.0python
a * 2输出结果:  python3.0python3.0
a[1]输出结果:  y
a[1:4]输出结果:  yth
H不在变量a中
M不再变量a中
\n
\n
```

图 24: 字符串运算符

7. 列表

列表的数据项不需要具有相同的类型，下面是一个列表：

```
List = [ 'a' , 'b' , 1 , 2 ]
```

图 25: 列表

索引：序列中的每个值都有对应的位置值，称之为索引，第一个索引是 0，第二个索引是 1，索引也可以从尾部（负索引）开始，最后一个元素的索引为 -1，往前一位为 -2，以此类推。

```
List = ['index1' , 'index2' , 'index3' , 'index4' , 'index5' ]
print("这是第一个索引",List[0])
print("这是第二个索引",List[1])
print("这是第三个索引",List[2])
print("这是倒数第一个索引",List[-1])
print("这是倒数第二个索引",List[-2])
print("这是倒数第三个索引",List[-3])
```

图 26: 索引

```
这是第一个索引 index1
这是第二个索引 index2
这是第三个索引 index3
这是倒数第一个索引 index5
这是倒数第二个索引 index4
这是倒数第三个索引 index3
```

图 27: 索引

切片：使用方括号 [] 截取字符

```
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(nums[1:7])
```

图 28: 切片

```
[2, 3, 4, 5, 6, 7]
```

图 29: 切片

添加: 使用 append() 方法来添加列表项

```
List = ['spring', 'summer', 'fall']
List.append('winter')
print("更新后的列表: ", List)
```

图 30: 添加

```
更新后的列表: ['spring', 'summer', 'fall', 'winter']
```

图 31: 添加

删除: 使用 del 语句来删除列表元素

```
List = ['spring', 'summer', 'fall']
List.append('winter')
print("更新后的列表: ", List)
del List[1]
print("更新后的列表", List)
```

图 32: 删除

```
更新后的列表 ['spring', 'fall', 'winter']
```

图 33: 删除

8. 元组

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[]

索引: 元组使用下标索引来访问元组中的值

```
tup1 = ('一', '二', '三', '四', '五')
tup2 = (1, 2, 3, 4, 5)

print("tup1[0]:", tup1[0])
print("tup2[1:5]", tup2[1:5])
```

图 34: 索引

```
tup1[0]: ...
tup2[1:5] (2, 3, 4, 5)
```

图 35: 索引

修改:

```
tup1 = (12, 34.56)
tup2 =('abc', 'xyz')
tup3 = tup1 + tup2
print(tup3)
```

图 36: 修改

```
(12, 34.56, 'abc', 'xyz')
```

图 37: 修改

删除: 元组中的元素是不允许删除的, 但我们可以使用 `del` 语句来删除整个元组

```
tup = ('一', '二', '三', '四', '五')
print(tup)
del tup
print("删除后的元组tup:")
print(tup)
```

图 38: 删除

```
删除后的元组tup:
Traceback (most recent call last):
  File "D:\pythonProject\3.1.py", line 140, in <module>
    print(tup)
    ^
NameError: name 'tup' is not defined. Did you mean: 'tup1'?
```

图 39: 删除

通过报错信息可以看到，元组已经被删除，已经没有这个元组了。

元组同样也可以使用 + 和 * 来进行运算，还可以索引，截取。

同时，元组还有内置函数：

表 3: 元组内置函数

符号	描述
len(tuple)	计算元组元素个数
max(tuple)	返回元组中元素最大值
min(tuple)	返回元组中元素最小值
tuple(iterable)	将可迭代系列转换为元组

下面是一些例子：

```
tuple1 = ('Google', 'Baidu', 'Taobao')
print(len(tuple1))
tuple2 = ('5', '4', '8')
print(max(tuple2))
print(min(tuple2))
list1 = ['Google', 'Taobao', 'Baidu']
tuple1 = tuple(list1)
print(tuple1)
```

图 40: 元组内置函数

```
3
8
4
('Google', 'Taobao', 'Baidu')
```

图 41: 元组内置函数

9. 字典

字典是另一种可变容器模型，可存储任意类型对象。字典中的每个键值 key=>value 对用冒号: 分割，每个对之间用逗号，分割，整个字典包括在花括号中。

下面是一个字典：

```
tinydict = {'name': 'zy', 'grade': 100, 'age': 18}
print(tinydict)
```

图 42: 字典

```
{'name': 'zy', 'grade': 100, 'age': 18}
```

图 43: 字典

如果要访问字典里的值，把相应键放入到方括号中：

```
tinydict = {'name': 'zy', 'grade': 100, 'age': 18}
print(tinydict)

print("tinydict['name']:", tinydict['name'])
```

图 44: 访问字典

```
{'name': 'zy', 'grade': 100, 'age': 18}
tinydict['name']: zy
```

图 45: 访问字典

修改字典：向字典添加新内容的方法是增加新的键/值对，修改或删除已有键/值对

```
tinydict = {'name': 'zy', 'grade': 100, 'age': 18}
print(tinydict)

print("tinydict['name']:", tinydict['name'])
tinydict['age'] = 19
tinydict['school'] = "OUC"
print(tinydict)
```

图 46: 修改字典

```
{'name': 'zy', 'grade': 100, 'age': 19, 'school': 'OUC'}
```

图 47: 修改字典

清空字典用`.clear()`，删除一个键用`del`

```
del tinydict['name'] #删除键'name'
tinydict.clear() #清空字典
del tinydict #删除字典
print(tinydict)
```

图 48: 字典

表 4: 字典内置函数

符号	描述
len(dict)	计算字典元素个数，即键的总数
str(dict)	输出字典，可以打印的字符串表示
type(variable)	返回输入的变量类型，如果变量是字典就返回字典类型

下面是字典内置函数：

下面是例子：

```
tinydict = {'name': 'zy', 'grade': 100, 'age': 18}
print(len(tinydict))
print(str(tinydict))
print(type(tinydict))
|
```

图 49: 字典函数

```
3
{'name': 'zy', 'grade': 100, 'age': 18}
<class 'dict'>
```

图 50: 字典函数

10. 集合

集合（set）是一个无序的不重复元素序列。集合其实也可以看作是一个没有键，只有值的字典。

假设现在有两个集合 a、b

a-b (集合 a 中包含而集合 b 中不包含的元素)

a|b (集合 a 或 b 中包含的所有元素)

a&b (集合 a 和 b 中都包含了的元素)

a^b (不同时包含于 a 和 b 的元素)

同时输出结果不是固定的，是个无序的不重复元素序列。

```

a = set('abcccd')
print(a)
b = set('ad')
c = a - b
d = a|b
e = a&b
f = a^b
print(c)
print(d)
print(e)
print(f)

```

图 51: 集合

```

{'a', 'c', 'b', 'd'}
{'c', 'b'}
{'a', 'd', 'c', 'b'}
{'a', 'd'}
{'c', 'b'}

```

图 52: 集合

集合中常用的几个方法:

表 5: 集合

符号	描述
add()	将元素添加到集合中, 如果元素已存在, 则不进行任何操作
update()	添加元素, 且参数可以是列表, 元组, 字典等
remove()	将元素从集合中移除, 如果元素不存在, 则会发生错误
discard()	将元素从集合中移除, 且如果元素不存在, 不会发生错误
pop()	将集合进行无序的排列, 然后将这个无序排列集合的左面第一个元素进行删除
len()	计算集合元素个数
clear()	清空集合
in	判断元素是否在集合中, 存在返回 True, 不存在返回 False

下面是一些例子:

```
fruit = set(["apple", "orange", "pear"])
fruit.add("banana")
print(fruit)
fruit.update({1,2,3})
print(fruit)
fruit.remove("apple")
print(fruit)
fruit.discard("banana")
print(fruit)
fruit.pop()
print(fruit)
print(len(fruit))
print("apple" in fruit)
fruit.clear()
print(fruit)
```

图 53: 集合

```
{'banana', 'apple', 'pear', 'orange'}
{'banana', 1, 2, 'pear', 'orange', 3, 'apple'}
{'banana', 1, 2, 'pear', 'orange', 3}
{1, 2, 'pear', 'orange', 3}
{2, 'pear', 'orange', 3}
4
False
set()
```

图 54: 集合

11. 迭代器与生成器

迭代器是一个可以记住遍历的位置的对象。迭代器对象从集合的第一个元素开始访问，直到所有的元素被访问结束。迭代器只能往前不会后退。迭代器有两个基本的方法：iter() 和 next()。字符串，列表或元组对象都可用于创建迭代器。迭代器对象可以使用常规 for 语句、while 语句等进行遍历。

下面是一个例子：

```
list = [1,2,3,4]
it = iter(list)#创建迭代器对象
print(next(it))#输出迭代器的下一个元素
for x in it:
    print(x,end=" ")
```

图 55: 迭代器

```
1  
2 3 4
```

图 56: 迭代器

StopIteration 异常用于标识迭代的完成，防止出现无限循环的情况。在 next() 方法中我们可以设置在完成指定循环次数后触发 StopIteration 异常来结束迭代。下面是一个迭代二十次停止执行的例子：

```
class MyNumbers:  
    def __iter__(self):  
        self.a = 1  
        return self  
    def __next__(self):  
        if self.a <= 20:  
            x = self.a  
            self.a += 1  
            return x  
        else:  
            raise StopIteration  
myclass = MyNumbers()  
myiter = iter(myclass)  
for x in myiter:  
    print(x)
```

图 57: StopIteration

```
13  
14  
15  
16  
17  
18  
19  
20  
|  
Process finished with exit code 0
```

图 58: StopIteration

生成器：使用 yield 的函数被成为生成器。其实生成器也是一个迭代器。在调用生成器运行的过程中，每次遇到 yield 时函数会暂停并保存当前所有的运行信息，返回 yield 的值，并在下一次执行 next() 方法时从当前位置继续运行。调用一个生成器函数，返回的是一个迭代器对象。

下面是一个例子实现斐波那契数列：

```

import sys
usage
def fibonacci(n): # 生成器函数 - 斐波那契
    a, b, counter = 0, 1, 0
    while True:
        if (counter > n):
            return
        yield a
        a, b = b, a + b
        counter += 1
f = fibonacci(10) # f 是一个迭代器, 由生成器返回生成
while True:
    try:
        print_(next(f), end=" ")
    except StopIteration:
        sys.exit()

```

图 59: 斐波那契数列

```
0 1 1 2 3 5 8 13 21 34 55
```

图 60: 斐波那契数列

12. 日期

python 提供了一个 time 和 calendar 模块可以用于格式化日期和时间。time 模块下有很多函数可以转换常见日期格式，函数 time.time() 用于获取当前时间戳。

```

import time
print(time.time())

```

图 61: 时间戳

```
1726137689.4677157
```

图 62: 时间戳

可以使用 time 模块的 strftime 方法来格式化日期：

```

import time # 格式化成YYYY-MM-DD HH:MM:SS形式
print_(time.strftime( format: "%Y-%m-%d %H:%M:%S", time.localtime()) ) # 格式化成
print_(time.strftime( format: "%a %b %d %H:%M:%S %Y", time.localtime()) ) # 将格式字
a = "Sat Mar 28 22:24:24 2016"
print_(time.mktime(time.strptime(a, format: "%a %b %d %H:%M:%S %Y")))

```

图 63: 格式化日期

```
2024-09-12 18:44:16  
Thu Sep 12 18:44:16 2024  
1459175064.0
```

图 64: 格式化日期

13. 函数

函数以 def 关键词开头, 后面是函数名和括号, 括号里面是要传入的参数。使用 return 结束函数。在下面直接用函数名调用函数。

```
def func(str):  
    """打印传入的字符串"""  
    print(str)  
    return  
func("hhh")  
func("ggg")
```

图 65: 函数

```
hhh  
ggg
```

图 66: 函数

14. 文件

打开文件用 open() 函数, 关闭用 close() 函数。write() 方法可以将任何字符串写入一个打开的文件, read() 方法可以从一个打开的文件中读取一个字符串。下面是一个例子:

```
fo = open("foo.txt", "w")  
fo.write("hello!\nworld\n")  
fo.close()  
fo = open("foo.txt", "r+")  
str = fo.read(10)  
print("读取的字符串是: ", str)  
fo.close()
```

图 67: 文件

```
读取的字符串是: hello!  
wor
```

图 68: 文件

15.os 模块

python 的 os 模块提供了执行文件操作的方法，比如重命名和删除文件。要使用这个模块，要先导入它，然后才可以调用相关功能。使用 `import os` 语句导入。

重命名是 `rename()` 方法，它需要两个参数，当前的文件名和新文件名。删除文件用 `remove()` 方法，需要提供要删除的文件名作为参数。下面是一个将我之前创建的 `foo` 文件重命名为 `foo1` 的操作：

```
import os  
os.rename( src: "foo.txt", dst: "foo1.txt")
```

图 69: 重命名



图 70: 重命名

可以看到，重命名成功。

下面我来删除这个文件：

```
os.remove("foo1.txt")
```

图 71: 删除

然后到原来文件所在的目录就可以看到这个文件已经没有了。

16. 面向对象

下面简单介绍类：

使用 `class` 语句创建一个新类，类之后为类的名称并以冒号结尾。下面用一个实例来具体介绍：

```
class Employee:  
    '''所有员工的基类'''  
    empCount = 0  
  
    def __init__(self, name, salary):  
        self.name = name  
        self.salary = salary  
        Employee.empCount += 1  
  
    def displayEmployee(self):  
        print("Total Employee %d" % Employee.empCount)  
        2 usages  
  
    def displayEmployee(self):  
        print("Name : ", self.name, ", Salary: ", self.salary)  
    '''创建 Employee 类的第一个对象'''  
    emp1 = Employee(name="Zara", salary=2000)  
    '''创建 Employee 类的第二个对象'''  
    emp2 = Employee(name="Manni", salary=5000)  
    emp1.displayEmployee()  
    emp2.displayEmployee()  
    print("Total Employee %d" % Employee.empCount)
```

图 72: 类

```
Name : Zara , Salary: 2000  
Name : Manni , Salary: 5000  
Total Employee 2
```

图 73: 类

`__init__`方法是一种特殊的方法，被称为类的构造函数或初始化方法，当创建了这个类的实例时就会调用该方法。`self`代表类的实例，`self`在定义类的方法时是必须有的，虽然在调用时不必传入相应的参数。其他的创建和访问属性都和 C 和 C++ 相似。

(二) python 计算机视觉

1.PIL:Python 图像处理类库

下面使用 `convert()` 方法将一个图片转换成灰色

```
from PIL import Image  
  
# 打开图像文件  
pil_im = Image.open('hh.jpg')  
  
# 将图像转换为灰度模式  
gray_im = pil_im.convert('L')  
  
# 显示图像  
gray_im.show()  
  
# 保存转换后的图像  
gray_im.save('hh_gray.jpg')
```

图 74: 灰色



图 75: 原图片



图 76: 灰色图片

2. Matplotlib

下面在原图上绘制几个点和一条线段：



图 77: 原图片

```
from PIL import Image
from pylab import *
# 读取图像到数组中
im = array(Image.open('empire.jpg'))
# 绘制图像
imshow(im)
# 一些点
x = [100,100,400,400]
y = [200,500,200,500]
# 使用红色星状标记绘制点
plot(*args: x,y, 'r*')
# 绘制连接前两个点的线
plot(*args: x[:2],y[:2])
# 添加标题，显示绘制的图像
title('Plotting: "empire.jpg"')
#axis('off')
show()
```

图 78: 方法

绘制后的图片：

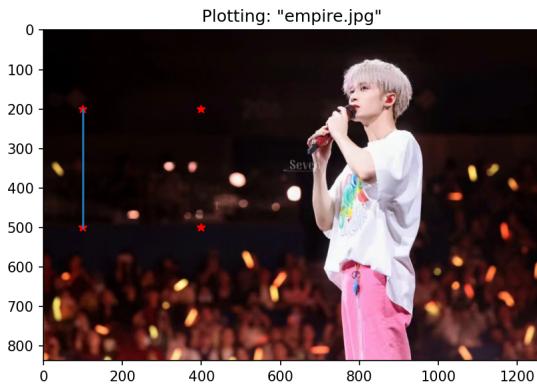


图 79: 图片 1

如果不显示坐标轴，增加 axis('off')，下面是改进后的图片：



图 80: 图片 2

3. NumPy 将图像转化为数组

将图片转化为数组:

```
from PIL import Image
from pylab import *
im = array(Image.open('empire.jpg'))
print(im.shape, im.dtype)
im = array(Image.open('empire.jpg').convert('L'), dtype='f')
print(im.shape, im.dtype)
```

图 81: 数组

```
(838, 1260, 3) uint8
(838, 1260) float32
```

图 82: 数组

其中因为图像通常被编码成无符号八位整型，所以在第一种情况下，数组的数据类型为“unit8”。在第二种情况下，对图像进行灰度化处理，并且在创建数组时使用额外的参数“f”；该参数将数据类型转换为浮点型 float32。

4. SciPy 图像模糊

下面进行模糊图片：

```
from PIL import Image
from numpy import *
from scipy.ndimage import gaussian_filter
# 打开图像并转换为灰度模式
im = array(Image.open('empire.jpg').convert('L'))
# 应用高斯滤波
im2 = gaussian_filter(im, sigma=5)
# 将处理后的图像转换为 PIL 图像对象并显示
im2_pil = Image.fromarray(im2)
im2_pil.show()
```

图 83: 模糊图片



图 84: 模糊图片

三、心得体会

本次学习主要学习了 python，并了解了一些 python 计算机视觉的相关内容，通过使用 PIL 处理图像、numpy 进行数值操作以及 scipy.ndimage 的高斯滤波等，我掌握了图像的基本处理技术，很有收获。

四、相关练习、报告和代码查看链接

本次报告相关练习、报告和代码均可以在 <https://kkgithub.com/zhangtantan77/work> 查看