《数字图像处理》课程作业(基于 Python)

(Version 1.1)

丁广太，李凯悦，童蓝蓝，卿青，孙春红，高博睿

上海大学计算机工程与科学学院，上海，中国，200444

gtding@shu.edu.cn

2019 年 9 月

（内部资料）

目录

1 引言

2019 年 9 月开设了《数字图像处理》课程，要求学生完成不少于 5 个图像处理算法设计题，编程语言可以选择 Matlab 或 Python.

本文是为选修本课程、基于 Python 语言进行程序设计的学生所写.请基于 Python3.X，选择如下功能，编程实现.

本手册仅供选修《数字图像处理》的学生参考和使用.内部资料，未经作者同意，不要散布.

2 功能

2.1 文件

2.1.1 文件操作

打开图像

保存图像

另存为

2.1.2 退出软件

2.2 编辑

2.2.1 操作的撤销和恢复

撤销图像操作

恢复撤销的图像操作

2.2.2 图像整体缩放

2.3 绘画、裁剪和度量

2.3.1 绘画

画笔、橡皮

2.3.2 旋转图像

2.3.3 局部图像缩放

2.3.4 裁剪

2.3.5 度量（长度和面积）

2.4 图像处理

2.4.1 图像分割（点检测）
2.4.2 图像分割（线检测）
2.4.4 图像分割（边缘检测）
2.4.4 图像分割（基于区域的分割）
2.4.5 图像分割（使用距离变换的分水岭分割）
2.4.6 图像分割（使用梯度的分水岭分割）
2.4.7 图像分割（控制标记符的分水岭分割）
2.4.8 小波变换
2.4.9 图像拼接
2.4.10 特征提取算法
2.4.11 图像增强（空间域：平滑、锐化）
2.4.12 图像增强（频域：平滑、锐化）
2.4.13 仅有噪声的图像复原（空间滤波）
2.4.14 图像复原（直接逆滤波）
2.4.15 图像复原（维纳滤波）
2.4.16 图像格式转换
2.4.17 GIF 图像制作
2.4.18 视频图像转换成图像序列
2.4.19 彩色图像分量分离
2.4.20 彩色图像颜色模式转换
2.4.20 彩色图像不同颜色模式的分量直方图

3 要求

3.1 知识产权风险规避

（1）不允许照抄网上代码；
（2）按照统一的代码格式进行编写（如下所示）.

3.2 代码格式

一个 Python 程序的结构一般是（以模块形式书写）：

```
# 起始行
```

```
# 模块文档
# 导入模块
# 定义全局变量
# 定义类
# 定义函数
# 在模块代码文件的尾部
    def function1():
    #...
    def function2():
    #...
    def functionX():
    #...
   ...
    #
    def main():
    #...
    pass
    # 根据__name__ 判断是否执行函数 main()
    if  __name__ == "__main__":
        main()
```

4 案例

4.1 图像文件读写和显示

使用 Image.open(filename) 打开一张图像文件, show( )函数显示所读取的图像文件, save(filename，formation) 将图像文件以指定文件格式保存（formation 为图像格式，如'jpeg', 'png' 等）

实现代码：

```
#/usr/bin/env Python
# coding=utf-8
"""
图像读取和保存.需要安装 pillow 包.
"""
from PIL import Image
def main():
```

```
    Img=Image.open('a.jpg') #读取图像文件(读当前目录下的 a.jpg 图像文件)
    Img.show() #显示图像文件
    Img.save('b.jpg','jpeg') #保存图像文件(将 Img 对象写入当前目录下的文件 b.jpg)
if __name__ == "__main__":
    main()
```

4.2 图像格式转换

convert( )函数转换图像格式，可以在'RGB'(真彩图)、'L'(灰度图)、'CMYK'(压缩图)等格式之间转换.

实现代码：

```
#/usr/bin/env Python
"""
图像格式转换.
"""
from PIL import Image
from pylab import *
def main():
    Img=Image.open('a.jpg') #读取图像文件(读当前目录下的 a.jpg 图像文件)
    Img_L=Img.convert("L")  #将图像转换为灰度图
    figure()
    subplot(1,2,1)
    imshow(Img)
    subplot(1,2,2)
    imshow(Img_L)
    show()
if __name__ == "__main__":
    main()
```

4.3 图像滤波

实现图像的中值滤波、均值滤波、高斯滤波.

（1）中值滤波：本方法是一种非线性平滑技术；将每一像素的灰度值设置为该像素滤波窗口中所有像素灰度值的中值.OpenCV 的中值滤波算法由函数 medianBlur(img, ksize)实现，其中，img 表示原始图像，ksize 表示滤波器大小，滤波窗口大小必须是正奇数.该函数使用滤波窗口大小为 ksize*ksize 的中值滤波器平滑图像.

实现代码：

```
#/usr/bin/env Python
#coding=utf-8
"""
```

```
图像滤波.需要安装 python-opencv 包.
"""
import cv2
import numpy as np
from matplotlib import pyplot as plt
if __name__ == '__main__':
    #读取目标图片
    img = cv2.imread("a.jpg")
    #均值滤波
    medIMG = cv2.medianBlur(img,7)
    #显示图像
    plt.subplot(1,2,1), plt.imshow(img), plt.title('Original')
    plt.xticks([]),plt.yticks([])
    plt.subplot(1,2,2), plt.imshow(medIMG), plt.title('Blurred')
    plt.xticks([]), plt.yticks([])
    plt.show()
```

输出：

图 1 原始图像（左）和中值滤波后图像（右）

（2）**均值滤波**：是一种线性滤波算法；通过将图像与标准化的盒式滤波器进行卷积完成滤波.例如，一个 3x3 标准化的盒式滤波器如下所示：

$$K=\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

OpenCV 中的均值滤波算法由函数 blur(img,(width, height))实现，其中 img 表示输入图像，width 和 height 是盒式滤波器（区域）的宽度和高度.

实现代码：

```
#/usr/bin/env Python
#coding=utf-8
"图像滤波：均值滤波"
import cv2
import numpy as np
```

```
from matplotlib import pyplot as plt
if __name__ == '__main__':
    #读取目标图片
    img = cv2.imread("a.jpg")
    #均值滤波
    blurIMG = cv2.blur(img, (5, 5))
    #显示图像
    plt.subplot(1,2,1), plt.imshow(img), plt.title('Original')
    plt.xticks([]),plt.yticks([])
    plt.subplot(1,2,2), plt.imshow(blurIMG), plt.title('Blurred')
    plt.xticks([]), plt.yticks([])
    plt.show()
```

输出：

图 2 原始图像（左）和均值滤波后图像（右）

(3)高斯滤波：对整幅图像进行加权平均，每个像素的值由其本身和邻域内的其他像素值经过加权平均后得到.

OpenCV 中的高斯滤波算法由函数 GaussianBlur(img, (width,height), sigmaX,sigmaY)实现，其中 img 表示原始图像，width 和 height 是滤波器的宽度和高度，它应该是正奇数，sigmaX 和 sigmaY 分别是 X 和 Y 方向的标准偏差；如果仅指定了 sigmaX，则 sigmaY 等于 sigmaX.如果两者都为零，则根据滤波窗口大小自动计算它们.

实现代码：

```
#/usr/bin/env Python
#coding=utf-8
"图像滤波:高斯滤波"
import cv2
import numpy as np
from matplotlib import pyplot as plt
if __name__ == '__main__':
    #读取目标图片
    img = cv2.imread("a.jpg")
    #高斯滤波
    blurIMG = cv2.GaussianBlur(img,(5,5),0)
    #显示图像
```

```
plt.subplot(1,2,1), plt.imshow(img), plt.title('Original')
plt.xticks([]),plt.yticks([])
plt.subplot(1,2,2), plt.imshow(blurIMG), plt.title('Blurred')
plt.xticks([]), plt.yticks([])
plt.show()
```

输出：

图 3 原始图像（左）和高斯滤波后图像（右）

4.4 图像特征提取

使用 SIFT 经典算法可以提取图像的 SIFT 特征.SIFT 算法在一定程度上可解决：

（1）目标的旋转、缩放、平移（RST）

（2）图像仿射/投影变换（视点 viewpoint）

（3）光照影响（illumination）

（4）目标遮挡（occlusion）

（5）杂物场景（clutter）

（6）噪声

SIFT 算法的实质是在不同的尺度空间上查找关键点(特征点)，并计算出关键点的方向.SIFT 查找到的关键点是一些十分突出，不会因光照，仿射变换和噪音等因素而变化的点，如角点、边缘点、暗区的亮点及亮区的暗点等.

实现代码：

```
#/usr/bin/env Python
"this is an image feature extraction module"
import cv2
if __name__ == '__main__':
    #读取图像
    img = cv2.imread("test.jpg")
    #将图像转化为灰度图
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    #创建 SIFT object: 若要下面语句顺利运行，
    #需要安装 opencv-contrib-python(3.4.2.16)
```

```
sift = cv2.xfeatures2d.SIFT_create()
#寻找图像中的关键点
kp = sift.detect(gray, None)
#在原始图像中画出关键点，用小圆圈表示
img = cv2.drawKeypoints(gray, kp, img)
#计算描述符
kp, des = sift.compute(gray, kp)
# 输出特征
print(des)
#保存已画关键点的图像
cv2.imwrite('sift_keypoints.jpg', img)
```

输出：

```
[[ 0. 0. 0. ... 6. 3. 1.]
[ 12. 4. 38. ... 19. 66. 50.]
[ 23. 22. 22. ... 8. 15. 14.]
...
[ 92. 151. 74. ... 0. 0. 0.]
[ 0. 1. 0. ... 0. 7. 17.]
[ 0. 0. 0. ... 24. 0. 0.]]
```

参考链接[1]：

OpenCV-Python Tutorials's documentation

https://opencv-python-tutroals.readthedocs.io/en/latest/index.html#

图 4 原始图像（左）及其角点分布（右）

4.5 图像分割

图像分割[2]：利用图像的灰度、颜色、纹理、形状等特征，把图像分成若干个互不重叠的区域，并使这些特征在同一区域内呈现相似性，在不同的区域之间存在明显的差异性.然后就可以将分割的图像中具有独特性质的区域提取出来用于不同的研究.

（1）传统分水岭算法分割

分水岭算法[3]：把图像中低密度的区域（变化很少）想象成山谷，图像中高密度的区域（变化很多）想象成山峰.开始向山谷注入水直到不同的山谷中的水开始汇聚.为了阻止不同的山谷的水汇聚，可以设置一些栅栏，最后得到的栅栏就是图像分割.

环境配置：Windows10；Python3.6；OpenCV 3.3.1.

实现步骤：

1)加载图像并转为灰度，设置一个阈值，将图像分为黑色部分和白色部分.

2)通过 morphologyEx()变换来去除噪声数据.

3)使用 dilate()函数得到大部分都是背景的区域.

4)使用 distanceTransform()函数的到最可能是前景的区域，并应用一个阈值来判断最可能为前景的区域.

5)前景与背景区域相减得到边界区域.

实现代码：

```
#/usr/bin/env Python
"This is an image segmentation module"
import numpy as np #用于矩阵运算
import cv2 #用于导入 opencv 库
from matplotlib import pyplot as plt #用于导入绘图库
if __name__ == '__main__':
    img = cv2.imread('test.jpg')
    #转化为灰度图
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)
    #去除噪声
    kernel = np.ones((3, 3), np.uint8)
    opening = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel, iterations=2)
    #确定背景区域
    sure_bg = cv2.dilate(opening, kernel, iterations=3)
    #寻找确定的前景区域
    dist_transform = cv2.distanceTransform(opening, cv2.DIST_L2, 5)
    ret, sure_fg = cv2.threshold(dist_transform, 0.7 * dist_transform.max(), 255,
0)
    #寻找未知区域
```

```
sure_fg = np.uint8(sure_fg)
unknown = cv2.subtract(sure_bg, sure_fg)
#标记标签
ret, markers = cv2.connectedComponents(sure_fg)
#在所有标签中添加一个，确保背景不是 0，而是 1
markers = markers + 1
#用 0 标记未知区域
markers[unknown == 255] = 0
markers = cv2.watershed(img, markers)
img[markers == -1] = [255, 0, 0]
plt.imshow(img)
#显示结果图像
plt.show()
```

输出：

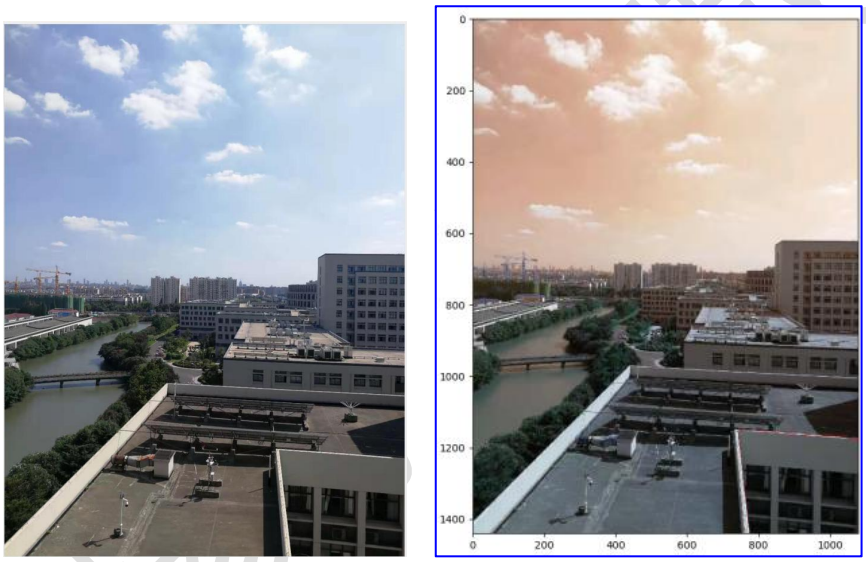


图 5 原始图像（左）和分水岭算法分割后图像（右）

（2）基于聚类的“图像分割”

Scikit-learn(sklearn)[4]是机器学习中常用的第三方模块，对常用的机器学习方法进行了封装，包括回归(Regression)、降维(Dimensionality Reduction)、分类(Classfication)、聚类(Clustering)等方法.并且 sklearn 拥有着完善的文档，上手容易，具有着丰富的 API，在学术界颇受欢迎.sklearn 已经封装了大量的机器学习算法，包括 LIBSVM 和 LIBINEAR.同时 sklearn 内置了大量数据集，节省了获取和整理数据集的时间.

该实验使用 sklearn 中的 KMeans 聚类方法[5]，进行图像分割.实验基于 Python3.

环境配置：Windows10；Python3.6.

实现步骤：

1)建立工程并导入 sklearn 包

2)加载图片并进行预处理

3)加载 Kmeans 聚类算法

4)对像素点进行聚类并输出

实验代码：

```
#/usr/bin/env Python
"this is a image segmentation module"
import numpy as np #用于矩阵运算
import PIL.Image as image #用于导入图像处理库
from sklearn.cluster import KMeans #用于导入聚类算法 KMeans
def load_data(file_path):
    f = open(file_path,'rb') #二进制打开
    data = []
    img = image.open(f) #以列表形式返回图片像素值
    m,n = img.size #活的图片大小
    for i in range(m):
        for j in range(n): #将每个像素点 RGB 颜色处理到 0-1 范围内并存放 data
            x,y,z = img.getpixel((i,j))
            data.append([x/256.0,y/256.0,z/256.0])
    f.close()
    return np.mat(data),m,n #以矩阵型式返回 data，图片大小
if __name__ == '__main__':
    img_data,row,col = load_data('test.jpg')
    label = KMeans(n_clusters=3).fit_predict(img_data) #聚类中心的个数为 3
    label = label.reshape([row,col]) #聚类获得每个像素所属的类别
    pic_new = image.new("L",(row,col)) #创建一张新的灰度图保存聚类后的结果
    for i in range(row): #根据所属类别向图片中添加灰度值
        for j in range(col):
            pic_new.putpixel((i,j),int(256/(label[i][j]+1)))
    pic_new.save('result.jpg') #保存结果图
```

输出：

图 6 原始图像（左）和基于聚类算法分割后图像（右）

4.6 字符识别

MNIST 手写数字数据库（Mixed National Institute of Standards and Technology database）包含 70000 张手写数字图片[6].这些数字是通过美国国家统计局的员工和美国高校的学生收集的.每张图片都是 28x28 的灰度图.

Keras 是一个非常方便的深度学习框架[7]，它以 TensorFlow 或 Theano 为后端，支持 GPU/CPU 的切换.用它可以快速地搭建深度网络，灵活地选取训练参数来进行网络训练.

该实验使用 Keras 框架，基于 MNIST 数据集，搭建全连接神经网络，进行手写数字识别[8]的实验，基于 Python3.

环境配置：Windows10；Python 3.6；keras 2.1.5.

实现步骤：

1)建立工程并导入 sklearn 包

2)加载图片并进行预处理

3)加载 Kmeans 聚类算法

4)对像素点进行聚类并输出

实验代码：

```
#!/usr/bin/env Python
#-*- coding: utf-8 -*-
"""
字符识别模块。需要安装 Keras 或高版本的 Tensorflow.
参见：https://blog.csdn.net/li528405176/article/details/83857286
      https://keras-cn.readthedocs.io/en/latest/
      https://blog.csdn.net/macair123/article/details/79509366
"""
from keras.models import Sequential   #采用贯序模型
from keras.layers import Input, Dense, Dropout, Activation   #导入 Keras 的几个重要函数
from keras.models import Model     #导入 Keras 的 model 类
from keras.optimizers import SGD #导入优化器 随机梯度下降法 SGD
from keras.datasets import mnist #导入 mnist 数据集
import numpy as np #用于矩阵运算
batchSize = 128   #每组包含的样本数量

"""第一步：选择模型"""
model = Sequential()   #采用贯序模型
```

"""第二步：构建网络层，并定义网络的参数"""

#构建的第 1 层作为输入层

#Dense 这是第一个隐藏层，并附带定义了输入层，该隐含层有 500 个神经元.输入则是 784 个节点

model.add(Dense(500, input_shape=(784,))) #输入层，28*28=784 输入层将二维矩阵换成了一维向量输入

model.add(Activation('tanh')) #激活函数是 tanh 为双曲正切 tanh(x) = sinh(x)/cosh(x) = (e^x - e^(-x))/(e^x + e^(-x))

model.add(Dropout(0.5)) #采用 50%的 dropout 随机取一半进行训练

#构建的第 2 层作为隐藏层 2， （如果加上输入层，实际上是第三层）

model.add(Dense(500)) #隐藏层节点 500 个

model.add(Activation('tanh'))

model.add(Dropout(0.5))

model.add(Dense(500)) #隐藏层 3，节点 500 个

model.add(Activation('tanh'))

#构建的第 3 个层作为输出层

model.add(Dense(10)) #输出结果是 10 个类别，所以维度是 10

model.add(Activation('softmax')) #最后一层用 softmax 作为激活函数

"""第三步：网络优化和编译"""

#lr：大于 0 的浮点数，学习率

#momentum：大于 0 的浮点数，动量参数

#decay：大于 0 的浮点数，每次更新后的学习率衰减值

#nesterov：布尔值，确定是否使用 Nesterov 动量

sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True) #优化函数，设定学习率(lr)等参数

#通过编译，建立模型(model)使用 categorical_crossentropy 交叉熵作为多分类损失函数，使用随机梯度下降法 SGD 作为优化器

model.compile(loss='categorical_crossentropy', optimizer=sgd)

"""第四步：训练"""

#获取 mnist 数据集

(X_train, y_train), (X_test, y_test) = mnist.load_data() #使用 Keras 自带的 mnist 工具读取数据

#由于 mist 的输入数据维度是(num, 28, 28)，这里需要把后面的维度直接拼起来变成 784

维

```
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1] * X_train.shape[2])
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1] * X_test.shape[2])

#这个能生成一个 OneHot 的 10 维向量，作为 Y_train 的一行，这样 Y_train 就有 60000 行 OneHot 作为输出
Y_train = (np.arange(10) == y_train[:, None]).astype(int)  # 整理输出
Y_test = (np.arange(10) == y_test[:, None]).astype(int)  # np.arange(5) = array([0,1,2,3,4])
'''
    训练模型（fit）的一些参数 （可分别设置不同参数查看最终的准确率）
    batch_size：对总的样本数进行分组，每组包含的样本数量
    epochs ：训练次数
    shuffle：是否把数据随机打乱之后再进行训练
    validation_split：拿出百分之多少用来做交叉验证
    verbose：屏显模式 0：不输出  1：输出进度  2：输出每次的训练结果
'''
model.fit(X_train, Y_train, batch_size=batchSize, epochs=50, shuffle=True, verbose=2, validation_split=0.3)
# model.evaluate(X_test, Y_test, batch_size=200, verbose=0)

"""第五步：输出"""
print("test set")
#误差评价 ：按 batch 计算在 batch 用到的输入数据上模型的误差
scores = model.evaluate(X_test, Y_test, batch_size=batchSize, verbose=0)
print("The test loss is %f" % scores)
#根据模型获取预测结果  为了节约计算内存，也是分组（batch）load 到内存中的，
result = model.predict(X_test, batch_size=batchSize, verbose=1)
#预测结果，即找到每行最大的序号
result_max = np.argmax(result, axis=1) #axis=1 表示按行  取最大值如果 axis=0 表示按列取最大值 axis=None 表示全部
test_max = np.argmax(Y_test, axis=1)  #这是结果的真实序号
result_bool = np.equal(result_max, test_max)  #预测结果和真实结果一致的为真（按元素比较）
true_num = np.sum(result_bool)  #正确结果的数量
print("The accuracy of the model is %f" % (true_num / len(result_bool)))  #验证结果的准
```

确率

4.7 语音处理

Python 读取 WAV 音频文件，并绘制 WAV 文件波形图，处理步骤：

1）将 WAV 文件导入到 Python 的工作环境中；

2）设置参数，声音信号(时间、振幅、频率)；

3）将这些信息通过 matplotlib.pyplot 提供的接口绘画出来.

环境配置：Windows10，Python3.6，matplotlib 3.1.1，numpy1.16.3，scipy 1.1.0，Python 自带 wave 包.

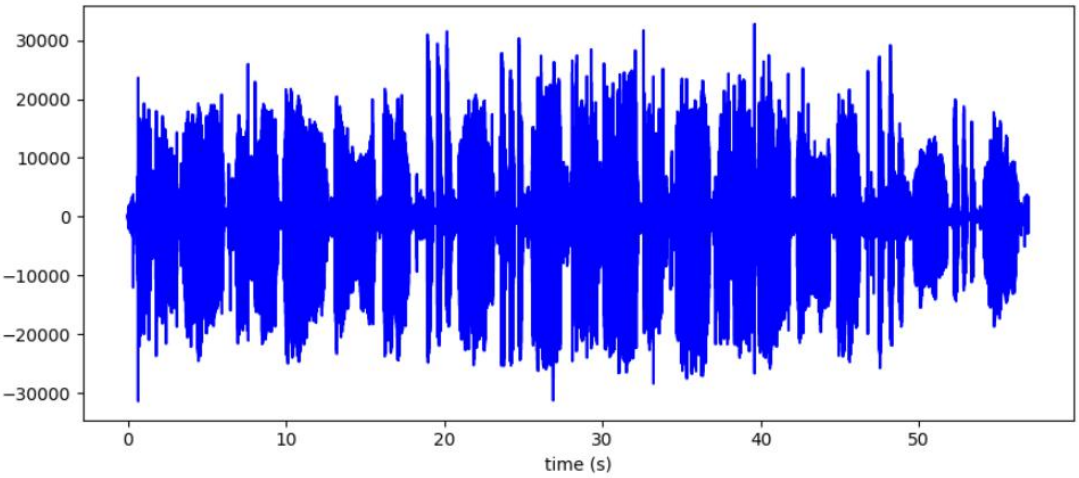
实现代码：

```
# /usr/bin/env Python
"this is a speech processing module"
import wave                          #用于解析.wav 文件
import matplotlib.pyplot as plt      #用于绘制波形图        (3)模块导入
import numpy as np                   #用于矩阵运算
from scipy.io import wavfile         #用于读取并解析文件
def wav_analysis():

    file_path = './test1.wav'
    wav_file = wave.open(file_path, 'rb')    #’rb’为只读模式，‘wb’为只写模式
    frame = wav_file.getnframes()            #获取音频的总帧数
    framerate = wav_file.getframerate()      #获取采样频率
    sample_time = 1 / framerate              #采样点的时间间隔
    time = frame / framerate
    sample_frequency, audio_sequence = wavfile.read(file_path)   #获取音频的采
                                             #样率和音频数据的 numpy 数组
    x_seq = np.arange(0, time, sample_time)      #获取 x 轴的标度
    plt.plot(x_seq, audio_sequence, 'blue')    #输入音频数据使用 matplotlib 进行
                                           #绘图
    plt.xlabel('time(s)')                      #  设置 x 轴的标签
    plt.show()                                 #显示所绘制的图像

if __name__ == '__main__':
    wav_analysis()
```

输出：



参考链接[9]：

https://docs.scipy.org/doc/scipy-0.19.0/reference/generated/scipy.io.wavfile.read.html

https://docs.python.org/3/library/wave.html

4.10 视频图像处理

视频的处理和图片的处理是类似的，而区别就是视频处理需要连续处理一系列的图片.一般有两种视频源，一种是直接从本地加载视频，另一种是获取摄像头的视频.对于视频的读取，OpenCV 提供了接口 VideoCapture.要想编程实现视频的读取与显示，需要熟悉一下该类的构造函数和成员函数.

本案例利用 OpenCV 模块，实现了对本地视频的读取、写入和显示.

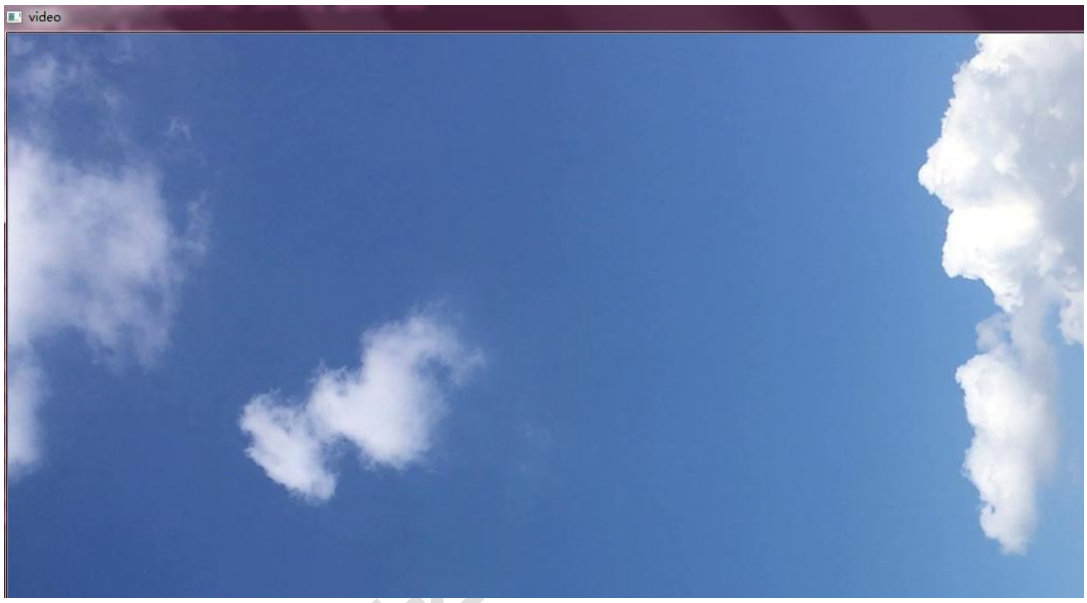
环境配置：Windows 7，Python3.7，opencv-contrib-python 3.4.2.16

实现代码：

```
#/usr/bin/env Python
"this is a video image processing module"
import cv2
def test():
    cap = cv2.VideoCapture('1.MOV')   #创建一个 VideoCapture 对象，并读取视频文件
    while(cap.isOpened()):    #读取视频
        ret, frame = cap.read()    #每一帧的捕获读取
        if ret == True:
            cv2.imshow('video',frame)   #展示读取完整的视频
            if cv2.waitKey(25) & 0xFF == ord('q'):    #播放时可以按 q 键退出播放
                break
```

```
            else:   #否则播放完毕后自动关闭
                break
        cap.release()   #最后释放 VideoCapture 对象
        cv2.destroyAllWindows()   #关闭播放的窗口
if __name__ == '__main__':
    test()
```

输出：



参考链接[10]：https://www.linuxidc.com/Linux/2015-08/121400.htm

参考文献

[1] OpenCV-Python Tutorials's documentation, https://opencv-python-tutroals.readthedocs.io/en/latest/index.html#

[2] 周莉莉，姜枫. 图像分割方法综述研究[J]. 计算机应用研究, 2017(7).

[3] 王国权，周小红，蔚立磊. 基于分水岭算法的图像分割方法研究[J]. 计算机仿真, 2009, 26(5):255-258.

[4] Keras: 基于 Python 的深度学习库，https://keras-cn.readthedocs.io/en/latest/

[5] 冯超. K-means 聚类算法的研究[D]. 大连理工大学, 2007.

[6] http://yann.lecun.com/exdb/mnist/

[7] https://keras.io/zh/#keras-python

[8] 郑样明，史耀武. 数字图象识别技术综述[J]. 湖北汽车工业学院学报, 1999(1).

[9] https://docs.scipy.org/doc/scipy-0.19.0/reference/generated/scipy.io.wavfile.read.html

[10] https://docs.python.org/3/library/wave.html

[11] OpenCV 官方教程中文版（For Python），
https://www.linuxidc.com/Linux/2015-08/121400.htm

[12] Jack LDZ，Tensorflow 简介以及与 Keras 的关系、常用机器学习框架一览，
https://blog.csdn.net/li528405176/article/details/83857286

[13] Keras 官方中文文档：Keras 安装和配置指南(Windows)
https://blog.csdn.net/macair123/article/details/79509366

[14]

附录　Python 语言概要

1. 概述

开发环境：

Python 底层代码支持环境、解释器、编辑器（源代码编辑和调试）

解释器：

官方标配：Python3.X（CPython），Anaconda3 等

编译运行解释器：Pypy

Java 混合开发：Jython

.NET 混合开发：IronPython

开发工具：

编写代码的 IDE 工具、结合解释器解释运行 Python 程序(Pycharm，VSCode 等）

2. 标准数据类型

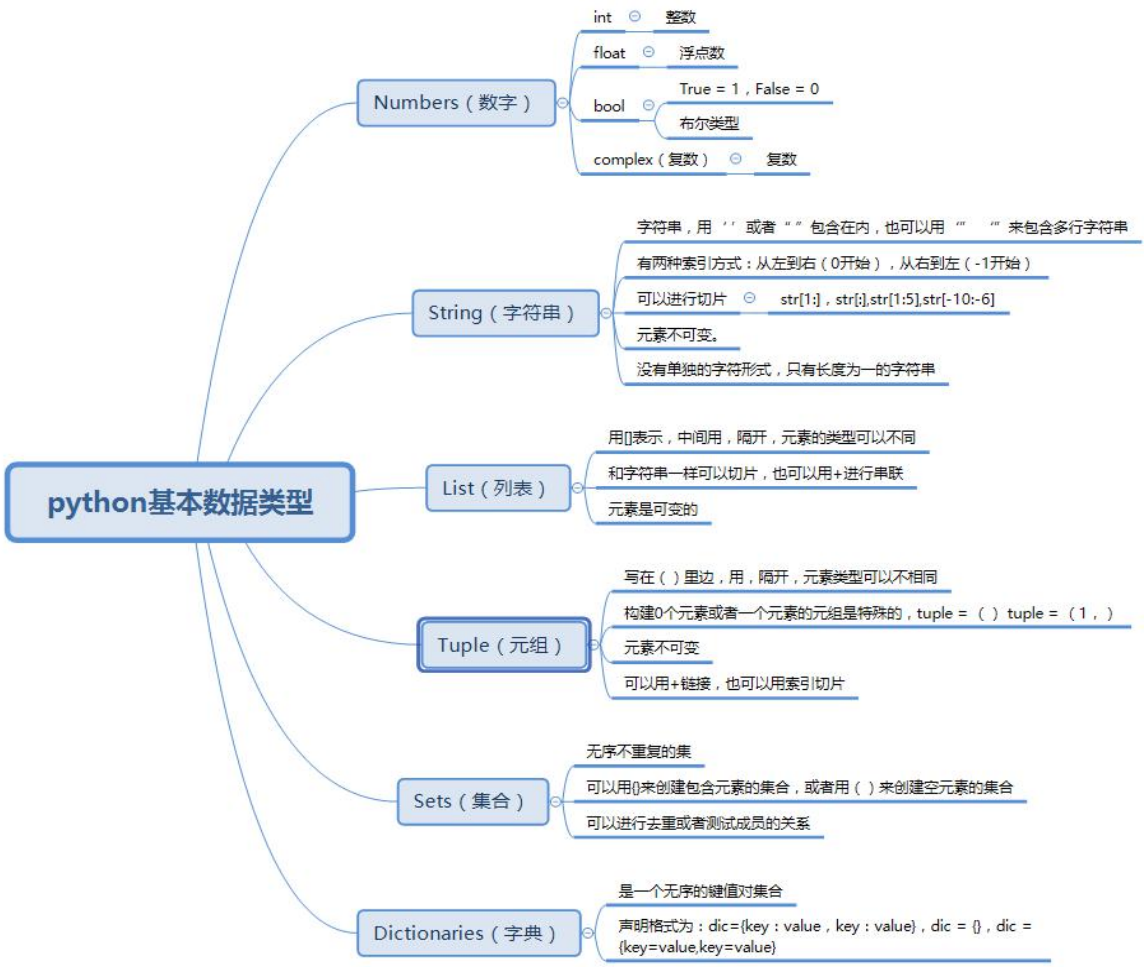


图 1 标准数据类型（https://www.cnblogs.com/chen-jun552/p/11200150.html）

Python3 中有六个标准的数据类型：

Number（数字）

String（字符串）

List（列表）

Tuple（元组）

Set（集合）

Dictionary（字典）

Python3 的六个标准数据类型中：

不可变数据(3 个)：Number（数字）、String（字符串）、Tuple（元组）；

可变数据(3 个)：List（列表）、Dictionary（字典）、Set（集合）。

3. 程序结构

顺序结构

python 是解释型的语言，代码会逐行解释运行

选择结构

if-else 选择结构：

单分支、双分支、多分支、嵌套方式

三元操作选择结构：

结果 1 if 条件，else 结果 2

条件为 True 时输出结果 1，条件为 False 时输出结果 2

循环结构

普通循环条件控制：while 循环

用于序列数据循环：for 循环

跳出循环：break

结束本次循环，开始下一次循环：continue

循环正常结束时才会执行的代码：else（循环中 break 一旦执行：else 中的代码就不会执行）

(https://blog.csdn.net/weixin_44606231/article/details/87110937)

关于程序的结构知识体系的描述，参见图 2.

从上面我们可以大概看出，一个完整的 py 程序结构大概如图 3 所示.

（1）起始行

#!/usr/bin/python

说明脚本语言是 Python 的，用/usr/bin 下面的程序（工具）python 作为解释器，解释运行 python 脚本.

```
# -*- coding: utf-8 -*-
```

指定文件编码为 utf-8 的.

Python3 注释

Python 中的注释有单行注释和多行注释：Python 中单行注释以 # 开头，多行注释用三个单引号 ''' 或者三个双引号 """ 将注释内容括起来.

（2）模块文档

为函数、类、方法等编写文档（只要在函数、类、方法定义后定义一个字符串即可）.借助于 Python 自带的 pydoc 模块，可以非常方便地查看、生成帮助文档，该文档是 HTML 格式的(http://c.biancheng.net/view/2674.html).例如：

```
python -m pydoc "C:\Users\DGT\Desktop\DLN\bin\test001.py"
```

（3）模块导入

模块（module）：用来从逻辑（实现一个功能）上组织 Python 代码（变量、函数、类），本质就是*.py 文件.文件是物理上组织方式"module_name.py"，模块是逻辑上组织方式"module_name".

包（package）：定义了一个由模块和子包组成的 Python 应用程序执行环境，本质就是一个有层次的文件目录结构（必须带有一个__init__.py 文件）.

导入方法：

```
# 导入一个模块
import model_name
# 导入多个模块
import module_name1,module_name2
# 导入模块中的指定的属性、方法（不加括号）、类
from moudule_name import moudule_element [as new_name]
```

方法使用别名时，使用"new_name()"调用函数，文件中可以再定义"module_element()"函数.

import 本质：路径搜索和搜索路径.

（4）变量定义

变量是一个存储数据的内存空间对象.定义一个变量，即向内存申请一个带地址的访问空间对象，用来存储数据，通过变量名找到（指向）这个值.python 是一种动态类型语言，定义一个变量不需要声明变量类型，变量类型取决于值的类型.几条规则：

- 同时定义多个变量：变量名与值用逗号隔开，一一对应
- 互相交换两个变量的值：变量 1，变量 2 = 变量 2，变量 1
- 值本身就有类型，不需要声明变量名类型，查看类型用内置函数 type()
- 查看变量的内存空间存储地址，用 id(变量名)方法查看
- 删除变量，用 del + 变量名 ，但删除的仅是一个指向存储对象的引用，不会删除存储对象.

（5）类定义

Python 的类定义由类头（指 class 关键字+类名+冒号（:））和统一缩进的类体构成，在类体中最主要的两个成员就是类变量和方法.格式如下：

```
class 类名：
    执行语句...
    零个到多个类变量...
    零个到多个方法...
```

1）类(Class)：用来描述具有相同的属性和方法的对象的集合.它定义了该集合中每个对象所共有的属性和方法.对象是类的实例.

2）方法：类中定义的函数.

3）类变量(属性)：类变量在整个实例化的对象中是公用的.类变量定义在类中且在函数体(方法)之外.类变量通常不作为实例变量使用，类变量也称作属性

4）数据成员：类变量或者实例变量用于处理类及其实例对象的相关数据.

5）实例变量：定义在方法中的变量，只作用于当前实例的类.

6）实例化：创建一个类的实例，类的具体对象.

7）对象：通过类定义的数据结构实例.

实例方法的定义与函数的方法定义基本相同，只是实例方法的第一个参数会被绑定到方法的调用者（该类的实例），因此实例方法至少应该定义一个参数，该参数通常会被命名为 self.（注意：实例方法的第一个参数并不一定要叫 self，其实完全可以叫任意参数名，只是约定俗成地把该参数命名为 self，这样具有最好的可读性）.例

```
class people:
    #定义基本属性
    name = ''
    age = 0
    #定义私有属性,私有属性在类外部无法直接进行访问
    __weight = 0
    #定义构造方法
    def __init__(self,n,a,w):
        self.name = n
        self.age = a
        self.__weight = w
    def speak(self):
        print("%s 说: 我 %d 岁." %(self.name,self.age))
# 实例化类
p = people('runoob',10,30)
p.speak()
```

（6）函数定义

Python 提供了许多内建函数，比如 print().但你也可以自己创建函数，即用户自定义函数.定义一个函数的规则：

- 函数代码块以 def 关键词开头，后接函数标识符名称和圆括号().
- 传入参数和自变量必须放在圆括号中间.圆括号之间可以用于定义参数.
- 函数的第一行语句可以选择性地使用文档字符串—用于存放函数说明.
- 函数内容以冒号起始，并且缩进.

- return [表达式] 结束函数，选择性地返回一个值给调用方.不带表达式的 return 相当于返回 None.

例子：

```
def functionname( parameters ):
    "函数_文档字符串"
    function_suite
    return [expression]
```

（7）主程序

Python 使用缩进对齐组织代码的执行，所有没有缩进的代码（非函数定义和类定义），都会在载入时自动执行，这些代码，都可以认为是 Python 的 main 函数.每个文件（模块）都可以任意写一些没有缩进的代码，并且在载入时自动执行，为了区分主执行文件还是被调用的文件，Python 引入了一个变量__name__，当文件是被调用时，__name__的值为模块名，当文件被执行时，__name__为'__main__'.因此，在之前的代码中，我们基本上可以使用这种方式来设置 main 函数的调用.（也就是说，如果是写成模块方式，则应该使用__name__变量编写主程序）

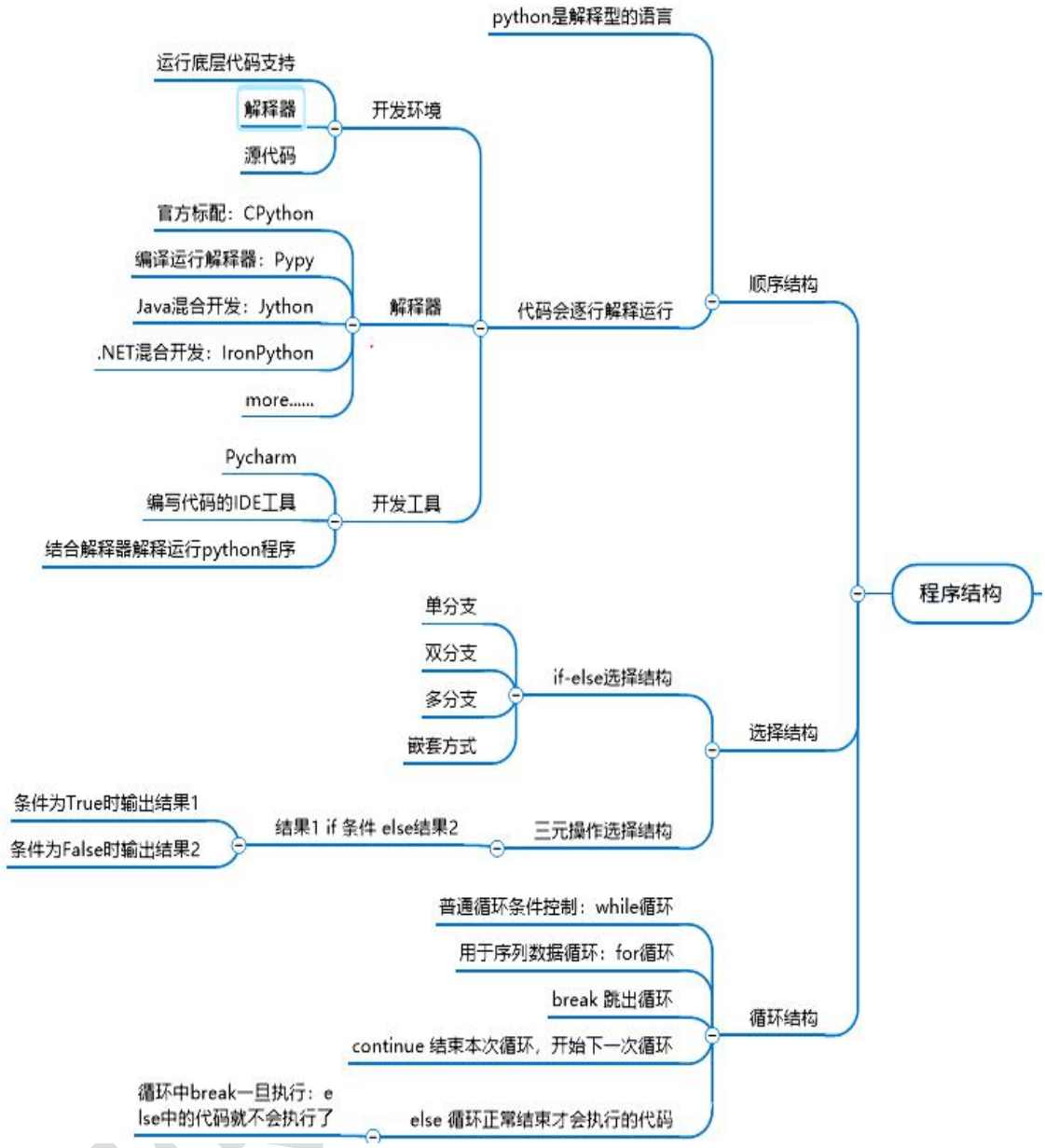


图 2 Python 程序知识体系

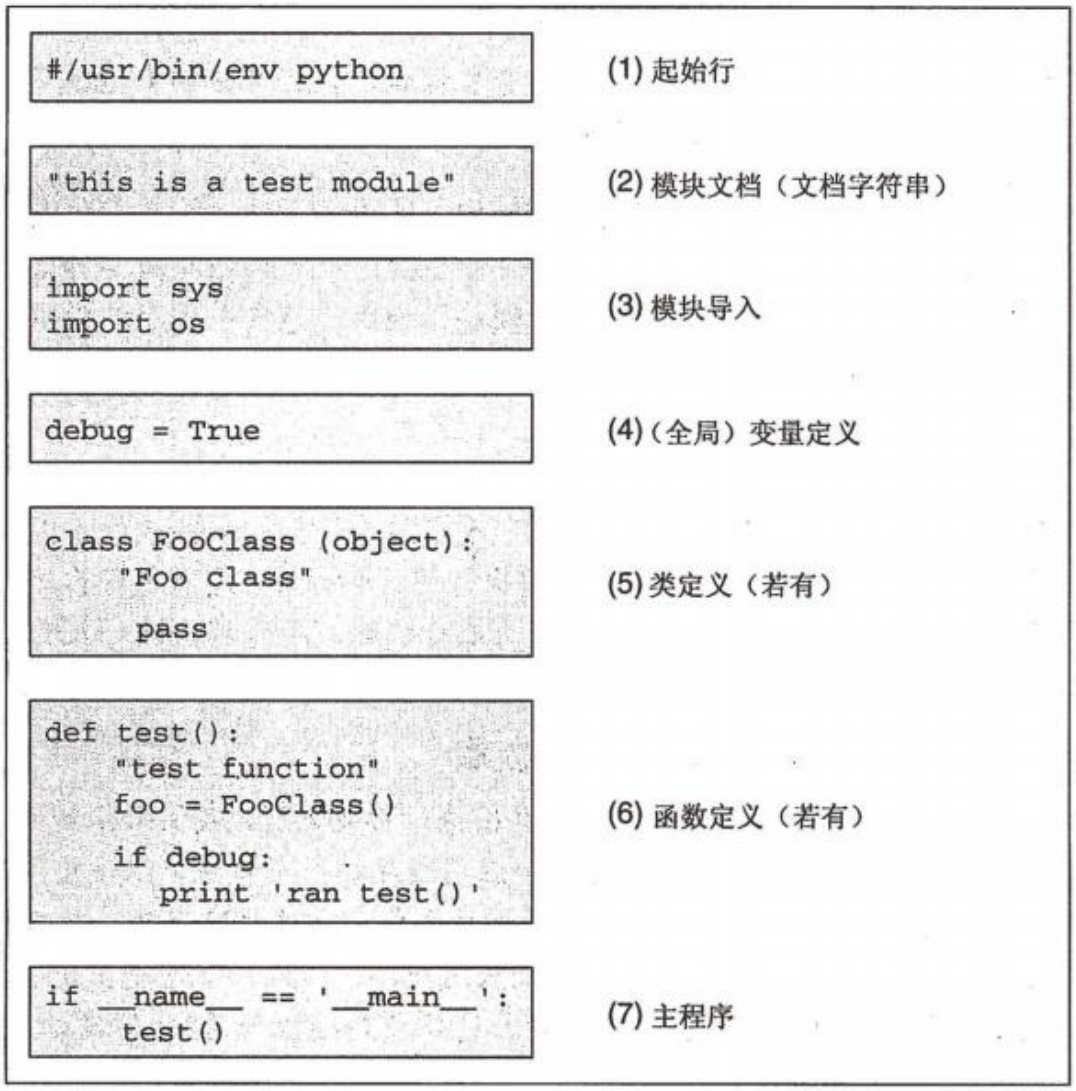


图 3 程序结构详情

也就是说，一个 Python 程序的结构一般是（以模块形式书写）：

```
# 起始行
# 模块文档
# 导入模块
# 定义全局变量
# 定义类
# 定义函数
# 在模块代码文件的尾部
    def main():
    #...
    pass
    # 根据__name__ 判断是否执行如下代码
    if __name__ == "__main__":
        main()
```

（8）概念辨析

函数：拥有名称、参数和返回值的代码块.被调用才会执行，通过参数和返回值与其它程序进行交互.

类：具有特定属性和方法的对象集合.

对象：类的实例

方法：类中定义的函数

类变量（属性）：类中定义在函数体（方法）之外的变量.

实例化：创建对象.

模块（package）：一个以模块方式书写的 Python 程序文件，扩展名为.py，该文件里面定义了一些函数和变量，通过 import 可以导入这些模块，进而使用该模块里的代码功能.

包（module）：模块的集合.为了方便代码管理，可将模块文件进行打包.包本质上是文件目录.包目录下第一个文件是 __init__.py，其它是一些模块文件和子目录，假如子目录中也有 __init__.py，那么该子目录是这个包的子包.

库：具有相关功能的模块的集合.

Python 模块/包/库安装（？）

4. 数据类型详解

Dictionary 字典

列表是有序的对象集合，字典是无序的对象集合.两者之间的区别在于：字典当中的元素是通过键来存取的，而不是通过偏移存取.

- 字典是一种映射类型，字典用 { } 标识，它是一个无序的 **键(key):值(value)** 的集合.
- 键(key)必须使用不可变类型（Number、String、Tuple）.
- 在同一个字典中，键(key)必须是唯一的.

字典类型也有一些内置的函数，例如 clear()、keys()、values()等.

Tuple 元组

元组（tuple）与列表类似，不同之处在于元组的元素不能修改.元组写在小括号 () 里，元素之间用逗号隔开.元组中的元素类型也可以不相同.

List 列表

引用语法：变量[头下标:尾下标].索引值以 0 为开始值，-1 为从末尾的开始位置.和字符串一样，列表同样可以被索引和截取，列表被截取后返回一个包含所需元素的新列表.与字符串不同的是 List 中的元素是可以改变的.List 内置了有很多方法，例如 append()、pop()等等.

- List 写在方括号之间，元素用逗号隔开.
- 和字符串一样，list 可以被索引和切片.
- List 可以使用+操作符进行拼接.
- List 中的元素是可以改变的.

String 字符串

引用语法：变量[头下标:尾下标].索引值以 0 为开始值，-1 为从末尾的开始位置.Python 中的字符串有两种索引方式，从左往右以 0 开始，从右往左以-1 开始，且字符串不能改变.Python 字符串不能被改变.向一个索引位置赋值，比如 word[0] = 'm'会导致错误.

- 反斜杠可以用来转义，使用 r 可以让反斜杠不发生转义.
- 字符串可以用+运算符连接在一起，用*运算符重复.
- Python 字符串有两种索引方式，从左往右以 0 开始，从右往左以-1 开始.
- Python 中的字符串不能改变.

Number 数字

Python 3 支持 int、float、bool、complex（复数）.数值类型的赋值和计算都是很直观的，就像大多数语言一样.内置的 type()函数可以用来查询变量所指的对象类型.

- Python 可以同时为多个变量赋值，如 a, b = 1, 2.
- 一个变量通过赋值可以指向不同类型的对象.
- 数值的除法（/）总是返回一个浮点数，要获取整数使用//操作符.
- 在混合计算时，Pyhton 会把整型转换成为浮点数.

Sets 集合

集合（set）是一个无序不重复元素的集.基本功能是进行成员关系测试和消除重复元素.可以使用大括号或者 set()函数创建 set 集合，注意：创建一个空集合必须用 set() 而不是 { }，因为{ }是用来创建一个空字典.

- set 集合中的元素不重复，重复了它会自动去掉.
- set 集合可以用大括号或者 set()函数创建，但空集合必须使用 set()函数创建. set 集合可以用来进行成员测试、消除重复元素.