实验操作结果

姓名：

学号：

实验内容：

1. python部分

* [第一个Python程序](https://www.liaoxuefeng.com/wiki/1016959663602400/1016966022717728)、

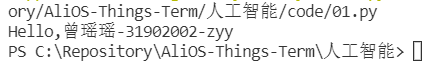
[使用文本编辑器](https://www.liaoxuefeng.com/wiki/1016959663602400/1017024645952992)、用print打印，('hello,自己的学号和姓名的拼音')

程序截图



输入和输出

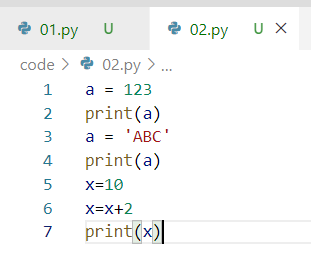
程序截图

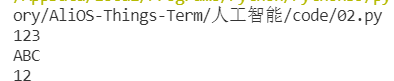


* [Python基础](https://www.liaoxuefeng.com/wiki/1016959663602400/1017063413904832)

[数据类型和变量](https://www.liaoxuefeng.com/wiki/1016959663602400/1017063826246112)

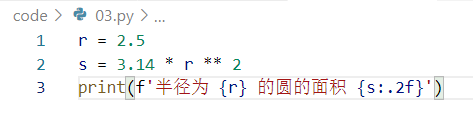
程序截图





[字符串和编码](https://www.liaoxuefeng.com/wiki/1016959663602400/1017075323632896)

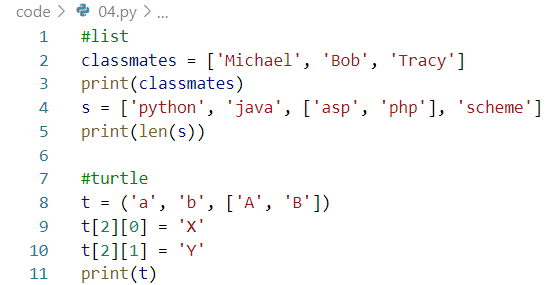
程序截图





[使用list和tuple](https://www.liaoxuefeng.com/wiki/1016959663602400/1017092876846880)

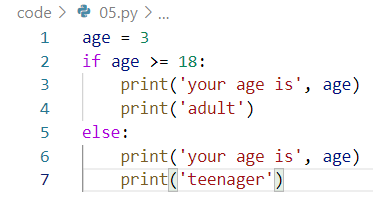
程序截图

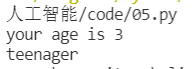




[条件判断](https://www.liaoxuefeng.com/wiki/1016959663602400/1017099478626848)

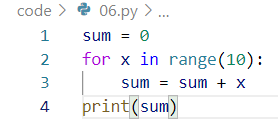
程序截图





[循环](https://www.liaoxuefeng.com/wiki/1016959663602400/1017100774566304)

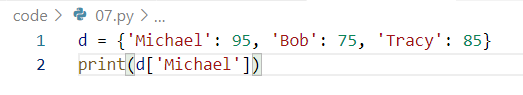
程序截图





[使用dict和set](https://www.liaoxuefeng.com/wiki/1016959663602400/1017104324028448)

程序截图





图形界面

* 海龟绘图

自己绘制一副美观好看的图形

程序截图

#!/usr/bin/env python

# -\*- coding:utf-8 -\*-

import turtle

import time

# 画心形圆弧

def hart\_arc():

    for i in range(200):

        turtle.right(1)

        turtle.forward(2)

def move\_pen\_position(x, y):

    turtle.hideturtle()     # 隐藏画笔（先）

    turtle.up()     # 提笔

    turtle.goto(x, y)    # 移动画笔到指定起始坐标（窗口中心为0,0）

    turtle.down()   # 下笔

    turtle.showturtle()     # 显示画笔

love = input("请输入表白话语，默认为‘I Love You’：")

signature = input("请签署你的大名，不填写默认不显示：")

if love == '':

    love = 'I Love You'

# 初始化

turtle.setup(width=800, height=500)     # 窗口（画布）大小

turtle.color('red', 'pink')     # 画笔颜色

turtle.pensize(3)       # 画笔粗细

turtle.speed(1)     # 描绘速度

# 初始化画笔起始坐标

move\_pen\_position(x=0,y=-180)   # 移动画笔位置

turtle.left(140)    # 向左旋转140度

turtle.begin\_fill()     # 标记背景填充位置

# 画心形直线（ 左下方 ）

turtle.forward(224)    # 向前移动画笔，长度为224

# 画爱心圆弧

hart\_arc()      # 左侧圆弧

turtle.left(120)    # 调整画笔角度

hart\_arc()      # 右侧圆弧

# 画心形直线（ 右下方 ）

turtle.forward(224)

turtle.end\_fill()       # 标记背景填充结束位置

# 在心形中写上表白话语

move\_pen\_position(0,0)      # 表白语位置

turtle.hideturtle()     # 隐藏画笔

turtle.color('#CD5C5C', 'pink')      # 字体颜色

# font:设定字体、尺寸（电脑下存在的字体都可设置）  align:中心对齐

turtle.write(love, font=('Arial', 30, 'bold'), align="center")

# 签写署名

if signature != '曾瑶瑶':

    turtle.color('red', 'pink')

    time.sleep(2)

    move\_pen\_position(180, -180)

    turtle.hideturtle()  # 隐藏画笔

    turtle.write(signature, font=('Arial', 20), align="center")

# 点击窗口关闭程序

window = turtle.Screen()

window.exitonclick()

结果截图



1. Tensorflow部分

使用tensorflow实现手写数字识别

# -\*- coding: utf-8 -\*-

"""MNIST手写数字图像识别.ipynb

Automatically generated by Colaboratory.

Original file is located at

    https://colab.research.google.com/drive/1FB6OZpImQ5yefJ1ZzTMuuNZYyIEstyF-

# \*\*Tensorflow2.X学习（一） — Keras基础应用（MNIST手写数字图像识别）\*\*

tensorflow：2.6.0

---

python：3.7.11

---

\*\*一、版本检查\*\*

"""

tf.\_\_version\_\_

import platform

print(platform.python\_version())

"""\*\*二.MNIST的数据查看\*\*

1.相关包导入

"""

import tensorflow as tf

from tensorflow.keras import layers

import matplotlib.pyplot as plt

import pandas as pd

"""2.读取MNIST数据。"""

#分别读入MNIST数据集的训练集数据和测试集数据

(X\_train\_image,y\_train\_label),(X\_test\_image,y\_test\_label) = tf.keras.datasets.mnist.load\_data()

"""3.打印MNIST数据的尺寸大小"""

print(len(X\_train\_image))

print(len(X\_test\_image))

print(X\_train\_image.shape)

"""60000张图像是训练集的数目，10000张则是测试集的数目，（60000，28，28）中的60000代表的是Batch\_size, 28和28分别代表一张图像的长和宽。

4.定义函数显示数字的图像

"""

def plot\_image(image):

    fig=plt.gcf()    #图表生成

    fig.set\_size\_inches(3,3)  #设置图表大小

    plt.imshow(image,cmap='binary') #以黑白灰度显示图片

    plt.show()    #开始绘图

plot\_image(X\_train\_image[0]) #显示一张图片

"""5.为了更加方便查看图像、标签及预测结果之间对应的关系，我们定义一个函数来显示。

"""

def plot\_image\_labels\_prediction(images,labels,prediction,idx,nums=10):

    fig = plt.gcf()

    fig.set\_size\_inches(12,14)  #设置图表大小

    if nums>25: nums=25 #最多显示25张图像

    for i in range(0,nums):

        ax = plt.subplot(5,5,1+i) #子图生成

        ax.imshow(images[idx],cmap='binary') #idx是为了方便索引所要查询的图像

        title = 'label=' + str(labels[idx]) #定义title方便图像结果对应

        if(len(prediction)>0): #如果有预测图像，则显示预测结果

            title += 'prediction='+ str(prediction[idx])

        ax.set\_title(title,fontsize=10) #设置图像title

        ax.set\_xticks([]) #无x刻度

        ax.set\_yticks([]) #无y刻度

        idx+=1

    plt.show()

plot\_image\_labels\_prediction(X\_train\_image,y\_train\_label,[],0,10) #显示前10张的图像

#下图为最后一行命令的显示结果：

"""\*\*三、MNIST的数据预处理\*\*

1.将image转化为一维向量, 转化过程是图像大小相乘，结果为：28x28 = 784, 以便于能作为神经网络的第一层输入。

"""

X\_train = X\_train\_image.reshape(60000,784).astype('float32') #二维转一维

X\_test = X\_test\_image.reshape(10000,784).astype('float32')

"""2.对图像数据进行归一化。归一化是必要操作，可以提高准确率。原因是如果没有归一化那么图像的数据值是0-255，不同的图像之间的值差异很大，在训练过程中，小的值很容易被忽略。因此我们需要对数据进行缩放，常见的一种方法是将数据的值压缩到(0,1)区间内，在本例中，直接将数据除以255即可。"""

X\_train\_normalize = X\_train/255 #归一化

X\_test\_normalize = X\_test/255

"""3.对数据标签进行One-hot编码（独热编码），我们都知道数字识别的结果是0-9，但为了满足神经网络的最后一层的10个神经元输出，我们需要将数字转换为二进制的10个0和1的组合，具体可看下面代码例子，第一个为数据的原标签，第二个是One-hot编码后的标签。"""

y\_train\_label[:5] #显示前5个图像的对应标签数字

"""之后大家可以直接调用keras的utils中的to\_categorical函数进行直接数字编码。"""

y\_TrainOneHot = tf.keras.utils.to\_categorical(y\_train\_label) #One-Hot编码

y\_TestOneHot = tf.keras.utils.to\_categorical(y\_test\_label)

y\_TrainOneHot[0:5] #显示前5个数据编码后的结果

"""上图为编码结果，其中1在第几个位置则代表了数字的大小是多少。

\*\*四.模型的搭建\*\*

1.建立线性堆叠模型Sequential，使用它我们可以搭建出很多复杂的神经网络模型，后面的层就犹如叠面包一样一层一层通过add指令叠加上去就好了，十分方便。

"""

model = tf.keras.models.Sequential() # 调用Sequential模型

"""2.不断叠加层上去，Dense层是全连接层，指的是神经前后前后两层神经元中两两都有互相连接，而Dropout的层的作用是为了防止过拟合（过拟合概念可自行查询），随机使多少比例的神经元失活，等于是断了之间的连接关系。

"""

#units是该层神经元数量，input\_dim是数据输入大小，kernel\_initializer=normal指的是用正态分布来初始化神经元的权重和偏差，activation指的是激活函数

model.add(layers.Dense(units=256,input\_dim=784,kernel\_initializer='normal',activation='relu'))

model.add(layers.Dropout(0.5)) #使50%的神经元失活

#最终输出是0-9，因此是10个神经元，softmax激活函数可以将输出转化为预测每个数字的概率

model.add(layers.Dense(units=10,kernel\_initializer='normal',activation='softmax'))

"""3.通过summary我们可以查看我们刚才搭建的模型的概要。

"""

print(model.summary()) #显示模型信息

"""大家可以从上图看到我们刚建立的模型信息，其中Param指的是该层神经元总的参数数量，具体的计算公式是 Param=（上一层神经元数量）x（本层神经元数量）+（本层神经元数量）

比如图中的200960是由（784x256+256）得到的。

\*\*五.模型的训练\*\*

1.在训练模型之间，我们需要定义一些训练的相关参数，比如优化器、损失函数和评估模型的方式。

"""

#loss这里采用的交叉熵损失，优化器采用的是adam优化器，评估方式为准确率

model.compile(loss='categorical\_crossentropy',optimizer='adam',metrics=['accuracy'])

"""2.开始训练，其中validation\_split是将原数据集划分一部分为验证集，epoch是训练次数，batch\_size是每一批训练的数据量，verbose=2代表显示所有训练过程。

"""

train\_history = model.fit(x=X\_train\_normalize,y=y\_TrainOneHot,validation\_split=0.2,epochs=10,batch\_size=200,verbose=2)

"""3.我们可以通过train\_history查看训练过程，之前的训练步骤的值都保存在这里面。这里共有loss,accuracy,val\_loss,val\_accuracy四个参数。

"""

train\_history.history

"""4.我们还可以定义函数，显示训练的曲线图。

"""

def show\_train\_history(train\_history,train,validation):

    plt.plot(train\_history.history[train]) #绘制训练数据的执行结果

    plt.plot(train\_history.history[validation]) #绘制验证数据的执行结果

    plt.title('Train History') #图标题

    plt.xlabel('epoch') #x轴标签

    plt.ylabel(train) #y轴标签

    plt.legend(['train','validation'],loc='upper left') #添加左上角图例

show\_train\_history(train\_history,'accuracy','val\_accuracy')

"""\*\*六.模型的测试\*\*

1.我们通过evaluate可以使用模型在测试集上进行测试。

"""

scores = model.evaluate(X\_test\_normalize,y\_TestOneHot)

print('accuracy=',scores[1]) #显示测试准确率

"""2.对模型进行预测，并使用之前我们定义的函数plot\_image\_labels\_prediction，可以方便查看预测结果之间的对应关系。

"""

prediction = model.predict(X\_test) #结果预测

plot\_image\_labels\_prediction(X\_test\_image,y\_test\_label,prediction,idx=340)

"""3.我们还可以通过pandas中的crosstab调用混淆矩阵，来查看哪些结果最容易混淆，对角线部分，为正确预测的部分，其它部分则是预测错误的部分。

使用pandas的crosstab函数查看混淆矩阵。这里要注意的是pandas.crosstab的输入必须是一维数组，所以传入的prediction和测试集的label都需要是一维的，如果不是一维数组，需要用reshape转为一维数组。

"""

import numpy as np

prediction=np.argmax(prediction,axis=1)

print(prediction.shape)

print(y\_test\_label.shape)

pd.crosstab(y\_test\_label,prediction,rownames=['label'],colnames=['predict'])

"""4.我们还可以通过pandas的DataFrame来查看预测错误数据的位置。

"""

df = pd.DataFrame({'label':y\_test\_label,'predict':prediction})

print(df[(df.label==5)&(df.predict==3)]) #查看是5预测错成3的数据位置

"""或者通过我们先前定义的plot\_image\_labels\_prediction函数，也可以直接查看预测错误的图像。

"""

plot\_image\_labels\_prediction(X\_test\_image,y\_test\_label,prediction,idx=340,nums=1)

"""以上便是Keras第一个实例的操作内容，该实例比较完整的演示了一个简单数据集在训练、测试及预测的整体流程，过段时间会分享后续的系列内容。"""

实现结果截图