# python 语法总结

# python简介

Python是著名的“龟叔”Guido van Rossum在1989年圣诞节期间，为了打发无聊的圣诞节而编写的一个编程语言。

Python就为我们提供了非常完善的基础代码库，覆盖了网络、文件、GUI、数据库、文本等大量内容，被形象地称作“内置电池（batteries included）”。用Python开发，许多功能不必从零编写，直接使用现成的即可。

除了内置的库外，Python还有大量的第三方库，也就是别人开发的，供你直接使用的东西。当然，如果你开发的代码通过很好的封装，也可以作为第三方库给别人使用。

许多大型网站就是用Python开发的，例如YouTube、Instagram，还有国内的豆瓣。很多大公司，包括Google、Yahoo等，甚至NASA（美国航空航天局）都大量地使用Python。

龟叔给Python的定位是“优雅”、“明确”、“简单”，所以Python程序看上去总是简单易懂，初学者学Python，不但入门容易，而且将来深入下去，可以编写那些非常非常复杂的程序。

总的来说，Python的哲学就是简单优雅，尽量写容易看明白的代码，尽量写少的代码。如果一个资深程序员向你炫耀他写的晦涩难懂、动不动就几万行的代码，你可以尽情地嘲笑他。

## 缺点

第一个缺点就是运行速度慢，和C程序相比非常慢，因为Python是解释型语言，你的代码在执行时会一行一行地翻译成CPU能理解的机器码，这个翻译过程非常耗时，所以很慢。而C程序是运行前直接编译成CPU能执行的机器码，所以非常快。

但是大量的应用程序不需要这么快的运行速度，因为用户根本感觉不出来。例如开发一个下载MP3的网络应用程序，C程序的运行时间需要0.001秒，而Python程序的运行时间需要0.1秒，慢了100倍，但由于网络更慢，需要等待1秒，你想，用户能感觉到1.001秒和1.1秒的区别吗？这就好比F1赛车和普通的出租车在北京三环路上行驶的道理一样，虽然F1赛车理论时速高达400公里，但由于三环路堵车的时速只有20公里，因此，作为乘客，你感觉的时速永远是20公里。

第二个缺点就是代码不能加密。如果要发布你的Python程序，实际上就是发布源代码，这一点跟C语言不同，C语言不用发布源代码，只需要把编译后的机器码（也就是你在Windows上常见的xxx.exe文件）发布出去。要从机器码反推出C代码是不可能的，所以，凡是编译型的语言，都没有这个问题，而解释型的语言，则必须把源码发布出去。

这个缺点仅限于你要编写的软件需要卖给别人挣钱的时候。好消息是目前的互联网时代，靠卖软件授权的商业模式越来越少了，靠网站和移动应用卖服务的模式越来越多了，后一种模式不需要把源码给别人。

再说了，现在如火如荼的开源运动和互联网自由开放的精神是一致的，互联网上有无数非常优秀的像Linux一样的开源代码，我们千万不要高估自己写的代码真的有非常大的“商业价值”。那些大公司的代码不愿意开放的更重要的原因是代码写得太烂了，一旦开源，就没人敢用他们的产品了。

# 安装

当我们编写Python代码时，我们得到的是一个包含Python代码的以.py为扩展名的文本文件。要运行代码，就需要Python解释器去执行.py文件。

由于整个Python语言从规范到解释器都是开源的，所以理论上，只要水平够高，任何人都可以编写Python解释器来执行Python代码（当然难度很大）。事实上，确实存在多种Python解释器。

**CPython**

当我们从Python官方网站下载并安装好Python 3.x后，我们就直接获得了一个官方版本的解释器：CPython。这个解释器是用C语言开发的，所以叫CPython。在命令行下运行python就是启动CPython解释器。

CPython是使用最广的Python解释器。教程的所有代码也都在CPython下执行。

## 输入输出

```python

# 输出

print('hello, world')

print(300)

print(100 + 200)

print('100 + 200 =', 100 + 200)

```

hello, world

300

300

100 + 200 = 300

```python

# 输入

name = input()

print(name)

```

shankai

shankai

任何计算机程序都是为了执行一个特定的任务，有了输入，用户才能告诉计算机程序所需的信息，有了输出，程序运行后才能告诉用户任务的结果。

输入是Input，输出是Output，因此，我们把输入输出统称为Input/Output，或者简写为IO。

input()和print()是在命令行下面最基本的输入和输出，但是，用户也可以通过其他更高级的图形界面完成输入和输出，比如，在网页上的一个文本框输入自己的名字，点击“确定”后在网页上看到输出信息。

好，最后。只要输入输出可以运行，说明python环境已经搭好。

# 函数

在Python中，定义一个函数要使用def语句，依次写出函数名、括号、括号中的参数和冒号:，然后，在缩进块中编写函数体，函数的返回值用return语句返回。

我们以自定义一个求绝对值的my\_abs函数为例：

```python

# 这是自定义的一个求绝对值的函数

def my\_abs(x):

if x >= 0:

return x

else:

return -x

```

```python

print(my\_abs(-99))

```

99

# 模块

Python本身就内置了很多非常有用的模块，只要安装完毕，这些模块就可以立刻使用。

```python

#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

' a test module '

\_\_author\_\_ = 'shankai'

import sys

def test():

args = sys.argv

if len(args)==1:

print('Hello, world!')

elif len(args)==2:

print('Hello, %s!' % args[1])

else:

print('Too many arguments!')

if \_\_name\_\_=='\_\_main\_\_':

test()

```

Too many arguments!

第1行和第2行是标准注释，第1行注释可以让这个hello.py文件直接在Unix/Linux/Mac上运行，第2行注释表示.py文件本身使用标准UTF-8编码；

第4行是一个字符串，表示模块的文档注释，任何模块代码的第一个字符串都被视为模块的文档注释；

第6行使用\_\_author\_\_变量把作者写进去，这样当你公开源代码后别人就可以瞻仰你的大名；

以上就是Python模块的标准文件模板，当然也可以全部删掉不写，但是，按标准办事肯定没错。

后面开始就是真正的代码部分。

你可能注意到了，使用sys模块的第一步，就是导入该模块：

```python

import sys

```

导入sys模块后，我们就有了变量sys指向该模块，利用sys这个变量，就可以访问sys模块的所有功能。

sys模块有一个argv变量，用list存储了命令行的所有参数。argv至少有一个元素，因为第一个参数永远是该.py文件的名称，例如：

运行python3 hello.py获得的sys.argv就是['hello.py']；

运行python3 hello.py Michael获得的sys.argv就是['hello.py', 'Michael]。

# 面向对象

面向对象最重要的概念就是类（Class）和实例（Instance），必须牢记类是抽象的模板，比如Student类，而实例是根据类创建出来的一个个具体的“对象”，每个对象都拥有相同的方法，但各自的数据可能不同。

仍以Student类为例，在Python中，定义类是通过class关键字：

```python

class Student(object):

pass

```

class后面紧接着是类名，即Student，类名通常是大写开头的单词，紧接着是(object)，表示该类是从哪个类继承下来的，继承的概念我们后面再讲，通常，如果没有合适的继承类，就使用object类，这是所有类最终都会继承的类。

定义好了Student类，就可以根据Student类创建出Student的实例，创建实例是通过类名+()实现的：

```python

bart = Student()

print(bart)

print(Student)

```

<\_\_main\_\_.Student object at 0x7fb1e0f7a780>

<class '\_\_main\_\_.Student'>

可以看到，变量bart指向的就是一个Student的实例，后面的0x10a67a590是内存地址，每个object的地址都不一样，而Student本身则是一个类。

可以自由地给一个实例变量绑定属性，比如，给实例bart绑定一个name属性：

```python

bart.name = 'Bart Simpson'

print(bart.name)

```

Bart Simpson

由于类可以起到模板的作用，因此，可以在创建实例的时候，把一些我们认为必须绑定的属性强制填写进去。通过定义一个特殊的\_\_init\_\_方法，在创建实例的时候，就把name，score等属性绑上去：

```python

class Student(object):

#注意：这里init前后是两个下划线

def \_\_init\_\_(self, name, score):

self.name = name

self.score = score

```

注意到\_\_init\_\_方法的第一个参数永远是self，表示创建的实例本身，因此，在\_\_init\_\_方法内部，就可以把各种属性绑定到self，因为self就指向创建的实例本身。

有了\_\_init\_\_方法，在创建实例的时候，就不能传入空的参数了，必须传入与\_\_init\_\_方法匹配的参数，但self不需要传，Python解释器自己会把实例变量传进去：

```python

bart = Student('Bart Simpson', 59)

print(bart.name)

print(bart.score)

```

Bart Simpson

59

# python进行机器学习的一般步骤

## 常用库

numpy 矩阵操作库

pandas 类数据库的库

matplotlib 绘制图形图像的可视化库

seaborn 绘制图像的库

## 读取数据集

一般是csv文件，csv文件是一种逗号分隔的二维表，类似于数据库中的表。一般使用pandas.read\_csv()读取

## 选择相应的机器学习算法

python 提供 scikit-learn 机器学习算法库，内置了基本的常用算法

### 分类

多层感知机（传统的神经网络）

from sklearn.neural\_network import MLPClassifier

KNN算法

from sklearn.neighbors import KNeighborsClassifier

支持向量机

from sklearn.svm import SVC

高斯分类

from sklearn.gaussian\_process import GaussianProcessClassifier

径向基网络

from sklearn.gaussian\_process.kernels import RBF

决策树

from sklearn.tree import DecisionTreeClassifier

随机森林和adaboost

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier

高斯朴素贝叶斯

from sklearn.naive\_bayes import GaussianNB

### 6.3.2 回归

线性回归

from sklearn.linear\_model import LinearRegression

随机梯度下降

from sklearn.linear\_model import SGDRegressor

岭回归

sklearn.linear\_model.Ridge

### 聚类

kmeans聚类算法

sklearn.cluster.KMeans

DBSCAN算法

sklearn.cluster.DBSCAN

MeanShift算法，多用与图像处理

sklearn.cluster.MeanShift

### 关联

apriori算法，实现较简单，sklearn不提供

## 训练和预测

训练

model.fit()

预测

model.predict()

## 可视化

# 一个机器学习例子

```python

# numpy 和 np 是等价的，为什么要用别名？ 因为可以少打字母，省力气

import numpy as np

import matplotlib.pyplot as plt

```

```python

从模块中引用函数，LineCollection等我们没有def，可以直接调用

```

```python

from matplotlib.collections import LineCollection

from sklearn.linear\_model import LinearRegression

from sklearn.isotonic import IsotonicRegression

from sklearn.utils import check\_random\_state

```

人工生成一些数据，y = log（1，x），但是这里我们掺杂了噪声

```python

n = 100

x = np.arange(n)

rs = check\_random\_state(0)

y = rs.randint(-50, 50, size=(n,)) + 50. \* np.log1p(np.arange(n))

```

```python

使用两种方式对这些点进行拟合。IsotonicRegression和LinearRegression现不用管他，这是两个类

```

```python

# #############################################################################

# Fit IsotonicRegression and LinearRegression models

ir = IsotonicRegression()

y\_ = ir.fit\_transform(x, y)

lr = LinearRegression()

lr.fit(x[:, np.newaxis], y) # x needs to be 2d for LinearRegression

```

LinearRegression(copy\_X=True, fit\_intercept=True, n\_jobs=1, normalize=False)

将人工生成的数据点和两个拟合的曲线画出来

```python

# #############################################################################

# Plot result

segments = [[[i, y[i]], [i, y\_[i]]] for i in range(n)]

lc = LineCollection(segments, zorder=0)

lc.set\_array(np.ones(len(y)))

lc.set\_linewidths(np.full(n, 0.5))

fig = plt.figure()

plt.plot(x, y, 'r.', markersize=12)

plt.plot(x, y\_, 'g.-', markersize=12)

plt.plot(x, lr.predict(x[:, np.newaxis]), 'b-')

plt.gca().add\_collection(lc)

plt.legend(('Data', 'Isotonic Fit', 'Linear Fit'), loc='lower right')

plt.title('Isotonic regression')

plt.show()

```

