

Python入门1

Python入门1

什么是Python

Python的2、3之争

开始使用

代码如何被执行？ -- Python解释器

Windows安装

Ubuntu(Debian)安装

Python的力量之源

代码写在哪？

写在最后

Python网络教程

Python入门读物

人生的经验

最后的最后 -- Python的哲学

什么是Python

Python is powerful... and fast;

plays well with others;

runs everywhere;

is friendly & easy to learn;

is Open.

These are some of the reasons people who use Python would rather not use anything else.

[Welcome to Python.org](https://www.python.org/)

- Python是一种解释型¹计算机程序设计语言
- 他优雅、简洁、高效²、通用，他能让你更专注于解决具体问题
- 大量的内置的和第三方的库将使你事半功倍，只需几行代码便能实现复杂的功能
- ...

[Python - Official Site](https://www.python.org/)

[Python - 维基百科，自由的百科全书](#)

[Python 百度百科](#)

总结一句，他很好玩！

1989的圣诞节，.....可能那时还没有冬季特惠吧

试看以下这两段代码

```
# first_example.py
str = input('Please input your string: ')
print('Your string in reverse order:', str[::-1])
```

```
// first_example.c
#include <stdio.h>

int main()
{
    int n = 0, i;
    char ch, str[1000];

    printf("Please input your string: ");
    while ((ch = getchar()) != '\n')
        str[n++] = ch;

    printf("Your string in reverse order: ");
    for (i = 0; i < n; i++)
        printf("%c", str[n - i - 1]);
    printf("\n");
    return 0;
}
```

他们的功能几乎是一样的。但是用C语言写的比用Python写的要复杂得多³

其实你们真的不用在意代码写了啥

以上的比较还仅仅是单纯从语法上，如果Python使用一些强大的库，那将极大地提高效率

```
# try_flask.py
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

if __name__ == "__main__":
    app.run()
```

你肯定不会想到，以上这7行代码就实现了一个简单的Web服务器。当然，他引入了一个很棒的库 `flask`

```

# font_image.py
from PIL import Image

IMG = 'filename'
WIDTH = 80
HEIGHT = 80
OUTPUT = './font_image.txt'

gray_char = list("$@B%8&WM#*oahkbdpqwmZO0QLCJUYXzcvunxrjft/\|()1{}[]?-_+~<>i!lI;:;,\"^' ".)

def rgb_to_char(r, g, b, alpha = 1):
    gray = int((0.2126 * r + 0.7152 * g + 0.0722 * b) * alpha + (1 - alpha) * 255)
    unit = 256 / len(gray_char)
    return gray_char[int(gray / unit)]

if __name__ == '__main__':
    im = Image.open(IMG)
    im = im.resize((WIDTH, HEIGHT), Image.NEAREST)
    txt = ""

    for i in range(HEIGHT):
        for j in range(WIDTH):
            txt += rgb_to_char(*im.getpixel((j, i)))
        txt += '\n'
    print(txt)

    if OUTPUT:
        with open(OUTPUT, 'w') as f:
            f.write(txt)
    else:
        with open("output.txt", 'w') as f:
            f.write(txt)

```

以上是一个有趣的小玩意，使用了第三方库 `pillow`。

你们可以试着拿C语言写一下，我反正不会233

Python的2、3之争

争论永无止息，所以我们还是应该朝前看

不过最起码，Python3的字符编码处理比Python2好多了。

Python2.7.13: 更丰富的教学资源、更多支持且稳定的库、更稳定的语言特性⁴

Python3.6.2: 目前最高的Python版本，拥有更长久支持与维护。或许有更广阔的发展空间？或许更酷？

Python2.7系列以及目前的**Python3.6**系列的代码是相互不可兼容的，学习之前最好能多了解，明确自己的方向。（我们还是尽量用Python3.6吧）

开始使用

在结束了一轮介绍之后，现在我们先来做一些比较实际的工作

代码如何被执行？ -- Python解释器

- CPython*
- PyPy
- Jython
-

Windows安装

1. [Python官网](#)下载、安装
2. 设置环境变量（可自动完成，留心勾选就可以避免很多麻烦）
3. 使用 `py` 启动器选择启动版本（`-2` 或 `-3`），使用 `Python` 启动随缘版本的Python

Ubuntu(Debian)安装

打开终端，使用apt命令安装py2或py3

```
apt-get install python
apt-get install python3
```

你可能（肯定）需要使用 `root` 用户权限，比如在命令前加上 `sudo`

使用 `python` 启动Python2.x，或 `python3` 启动python3.x

Python的力量之源

前面说过Python拥有大量的官方和第三方库，这是高效的源泉（之一）。嗯，所以管理他们很重要。

pip就是一个*好的Python包管理器

以上涉嫌违反广告法的部分已用*代替

Windows使用 [pip-get.py](#) 脚本安装

```
python get-pip.py
py -3 get-pip.py
```

Ubuntu可以通过apt工具安装

```
apt-get install python-pip
apt-get install python3-pip
```

安装完成之后可以使用pip命令安装Python第三方库，比如一个优雅的HTTP库 `requests`

```
pip install requests
pip3 install requests
```

代码写在哪？

安装好Python后，我们就该开始写点代码了，我们主要有以下三种方式

- 命令行的Python交互式环境（Linux终端、PowerShell、CMD、...）

- 文本编辑器 + 命令行（[sublime text 3](#)、Notepad ++、Visual Studio Code、...）
- IDE（[PyCharm](#)、IDLE、Visual Studio（这一定是在开玩笑）、...）
- [pythonanywhere](#)等在线编译器
- [QPython](#)等移动端编译器

以上前三种方式各有优劣，我们并不打算给你推荐其中一种，而且恰恰相反，我觉得他们都是必须的。

交互式环境可以做简单测试，文本编辑器可以小修小补，IDE可以做大事

第四、五种还是算了吧，玩玩就好。

写在最后

Python网络教程

[廖雪峰的官方网站](#)

Python入门读物

《Python基础教程（第二版 修订版）》

主要基于Python3，主要关注语法

《Python核心编程（第二版）》

主要基于Python2，前三分之二是语法后面是一些应用

《Python核心编程（第三版）》

主要基于Python2，主要关注应用

第三版并不是第二版的超集

.....鄙人书读得少，欢迎强者补充

人生的经验

很多概念，在没有接触过足够多的事物，没有经过足够多的思考，没有足够的知识积累之前是没法理解的

所以我们先不用理解，只要知道 *什么样？* 以及 *怎么用？* 即可

以及，多看多想！

最后的最后 -- Python的哲学

```
import this
```

The Zen of Python, by Tim Peters

Beautiful is better than ugly.

Explicit is better than implicit.

Simple is better than complex.

Complex is better than complicated.

Flat is better than nested.

Sparse is better than dense.

Readability counts.

Special cases aren't special enough to break the rules.

Although practicality beats purity.

Errors should never pass silently.

Unless explicitly silenced.

In the face of ambiguity, refuse the temptation to guess.

There should be one-- and preferably only one --obvious way to do it.

Although that way may not be obvious at first unless you're Dutch.

Now is better than never.

Although never is often better than *right* now.

If the implementation is hard to explain, it's a bad idea.

If the implementation is easy to explain, it may be a good idea.

Namespaces are one honking great idea -- let's do more of those!

1. <https://zh.wikipedia.org/wiki/%E7%9B%B4%E8%AD%AF%E8%AA%9E%E8%A8%80> [↵](#)

2. 此处是指写代码比较高效，而非运行得高效 [↵](#)

3. 行数多600%，字符数多240% [↵](#)

4. 自2016-12-17更新至Python2.7.13至今Python2.x再无更新，极可能以后也不再有更新 [↵](#)