

# 使用库

**库** 就是别人已经写好了的代码, 可以让我们直接拿来用.

荀子曰: "君子性非异也, 善假于物也"

一个编程语言能不能流行起来, 一方面取决于语法是否简单方便容易学习, 一方面取决于生态是否完备.

所谓的 "生态" 指的就是语言是否有足够丰富的库, 来应对各种各样的场景.

实际开发中, 也并非所有的代码都自己手写, 而是要充分利用现成的库, 简化开发过程.

按照库的来源, 可以大致分成两大类

- 标准库: Python 自带的库. 只要安装了 Python 就可以直接使用.
- 第三方库: 其他人实现的库. 要想使用, 需要额外安装.

咱们自己也可以实现 "第三方库" 发布出去, 交给别人来使用.

## 标准库

### 认识标准库

Python 自身内置了非常丰富的库.

在 Python 官方文档上可以看到这些库的内容.

<https://docs.python.org/3.10/library/index.html>

简单来说, 主要是这些部分:

- 内置函数 (如 print, input 等)
- 内置类型 (针对 int, str, bool, list, dict 等类型内置的操作).
- 文本处理
- 时间日期
- 数学计算
- 文件目录
- 数据存储 (操作数据库, 数据序列化等).
- 加密解密
- 操作系统相关
- 并发编程相关 (多进程, 多线程, 协程, 异步等).
- 网络编程相关
- 多媒体相关 (音频处理, 视频处理等)
- 图形化界面相关
- .....

我们不需要把这些库的内容都背下来, 只要大概知道里面有啥, 需要用的时候能够找到即可.

## 使用 import 导入模块

使用 import 可以导入标准库的一个 模块

```
import [模块名]
```

所谓 "模块", 其实就是一个单独的 `.py` 文件.

使用 import 语句可以把这个外部的 `.py` 文件导入到当前 `.py` 文件中, 并执行其中的代码.

## 代码示例: 日期计算

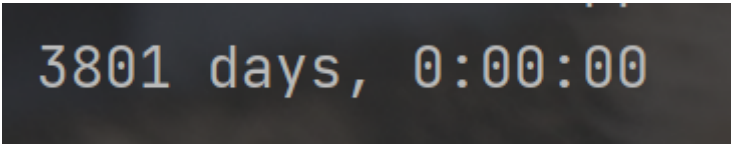
输入任意的两个日期, 计算两个日期之间隔了多少天.

- 使用 `import` 语句导入标准库的 `datetime` 模块
- 使用 `datetime.datetime` 构造两个日期. 参数使用 年, 月, 日 这样的格式.
- 两个日期对象相减, 即可得到日期的差值.

```
import datetime

date1 = datetime.datetime(2012, 2, 14)
date2 = datetime.datetime(2022, 7, 12)

print(date2 - date1)
```



```
3801 days, 0:00:00
```

快来看一下你和心爱的人已经认识了多少天了.

关于 `datetime` 的更多操作, 可以参考官方文档

<https://docs.python.org/3/library/datetime.html>

## 代码示例: 字符串操作

字符串是 Python 的内置类型, 字符串的很多方法不需要导入额外的模块, 即可直接使用.

### 1) 剑指offer 58, 翻转单词顺序

<https://leetcode.cn/problems/fan-zhuan-dan-ci-shun-xu-lcof/>

输入一个英文句子, 翻转句子中单词的顺序, 但单词内字符的顺序不变. 为简单起见, 标点符号和普通字母一样处理. 例如输入字符串 "I am a student.", 则输出 "student. a am I".

- 使用 `str` 的 `split` 方法进行字符串切分, 指定 空格 为分隔符. 返回结果是一个列表.
- 使用列表的 `reverse` 方法进行逆序.
- 使用 `str` 的 `join` 方法进行字符串拼接. 把列表中的内容进行合并.

```
def reversewords(s):
    tokens = s.split()
    tokens.reverse()
    return ' '.join(tokens)

print(reversewords('I am a student.'))
```

## 2) leetcode 796, 旋转字符串

<https://leetcode.cn/problems/rotate-string/>

给定两个字符串,  $s$  和  $goal$ 。如果在若干次旋转操作之后,  $s$  能变成  $goal$ , 那么返回  $true$ 。

$s$  的 旋转操作 就是将  $s$  最左边的字符移动到最右边。

例如, 若  $s = 'abcde'$ , 在旋转一次之后结果就是  $'bcdea'$ 。

- 使用 `len` 求字符串的长度. 如果长度不相同, 则一定不能旋转得到.
- 将  $s$  和 自己 进行拼接, 然后直接使用 `in` 方法来判定  $goal$  是否是  $s + s$  的子串.

```
def rotateString(s, goal):
    return len(s) == len(goal) and goal in s + s

print(rotateString('abcde', 'cdeab'))
```

## 3) leetcode 2255, 统计是给定字符串前缀的字符串数目

<https://leetcode.cn/problems/count-prefixes-of-a-given-string/>

给你一个字符串数组  $words$  和一个字符串  $s$ , 其中  $words[i]$  和  $s$  只包含 小写英文字母。

请你返回  $words$  中是字符串  $s$  前缀 的 字符串数目。

一个字符串的 前缀 是出现在字符串开头的子字符串。子字符串 是一个字符串中的连续一段字符序列。

- 依次遍历  $words$  中的字符串
- 直接使用字符串的 `startswith` 方法即可判定当前字符串是否是  $s$  的前缀.

```
def countPrefixes(words, s):
    res = 0 # 符合要求字符串个数
    for word in words:
        if s.startswith(word):
            res += 1
    return res

print(countPrefixes(["a", "b", "c", "ab", "bc", "abc"], "abc"))
```

关于字符串的更多操作, 参考官方文档

<https://docs.python.org/3/library/stdtypes.html#str>

## 代码示例: 文件查找工具

指定一个待搜索路径, 同时指定一个待搜索的关键字.

在待搜索路径中查找是否文件名中包含这个关键字.

- 使用 `os.walk` 即可实现目录的递归遍历.
- `os.walk` 返回一个三元组, 分别是 当前路径, 当前路径下包含的目录名 (多个), 当前路径下包含的文件名 (多个)

```
import os

inputPath = input('请输入待搜索路径: ')
pattern = input('请输入待搜索关键词: ')

for dirpath, dirnames, filenames in os.walk(inputPath):
    for f in filenames:
        if pattern in f:
            print(f'{dirpath}/{f}')
```

关于 `os` 模块的更多操作, 参考官方文档

<https://docs.python.org/3/library/os.html>

## 第三方库

### 认识第三方库

第三方库就是别人已经实现好了的库, 我们可以拿过来直接使用.

虽然标准库已经很强大了, 但是终究是有限的. 而第三方库可以视为是集合了全世界 Python 程序猿的智慧, 可以说是几乎无穷无尽.

问题来了, 当我们遇到一个需求场景的时候, 如何知道, 该使用哪个第三方库呢?

就需要用到下面几个网站了:



当我们确定了该使用哪个第三方库之后, 就可以使用 `pip` 来安装第三方库了.

# 使用 pip

pip 是 Python 内置的 **包管理器**.

所谓 **包管理器** 就类似于我们平时使用的手机 app 应用商店一样.

第三方库有很多, 是不同的人, 不同的组织实现的. 为了方便大家整理, Python 官方提供了一个网站 PyPI <https://pypi.org/> 来收集第三方库.

其他大佬写好的第三方库也会申请上传到 PyPI 上.

这个时候就可以方便的使用 pip 工具来下载 PyPI 上的库了.

pip 在我们安装 Python 的时候就已经内置了. 无需额外安装.

pip 是一个可执行程序, 就在 Python 的安装目录中.

打开 cmd, 直接输入 pip. 如果显示以下帮助信息, 说明 pip 已经准备就绪.

```
C:\Users\72770>pip

Usage:
  pip <command> [options]

Commands:
  install           Install packages.
  download          Download packages.
  uninstall         Uninstall packages.
  freeze            Output installed packages in requirements format.
  list              List installed packages.
  show              Show information about installed packages.
  check             Verify installed packages have compatible dependencies.
  config            Manage local and global configuration.
  search            Search PyPI for packages.
  cache             Inspect and manage pip's wheel cache.
  index             Inspect information available from package indexes.
  wheel             Build wheels from your requirements.
  hash              Compute hashes of package archives.
  completion        A helper command used for command completion.
  debug             Show information useful for debugging.
  help              Show help for commands.
```

如果最开始按照要求在安装 Python 的时候勾选了

☒ Add Python 3.10 to PATH

那么 pip 就是默认可用的.

如果提示

'pip' 不是内部或外部命令, 也不是可运行的程序  
或批处理文件。

则说明没有正确的把 pip 加入到 PATH 中, 可以手动把 pip 所在的路径加入到 PATH 环境变量中. (参考 <https://www.jianshu.com/p/1de0acf7185d>)

或者卸载重装 Python, 记得勾上上述选项, 也许是更简单的办法.

使用以下命令, 即可安装第三方库

```
pip install [库名]
```

**注意:** 这个命令需要从网络上下载, 使用时要保证网络畅通.

安装成功后, 即可使用 `import` 导入相关模块, 即可进行使用.

**注意:** 如果使用 pip 安装完第三方库之后, 在 PyCharm 中仍然提示找不到对应的模块, 则检查 Settings -> Project -> Python Interpreter, 看当前 Python 解释器设置的是否正确. (如果一个机器上安装了多个版本的 Python, 容易出现这种情况).

## 代码示例: 生成二维码

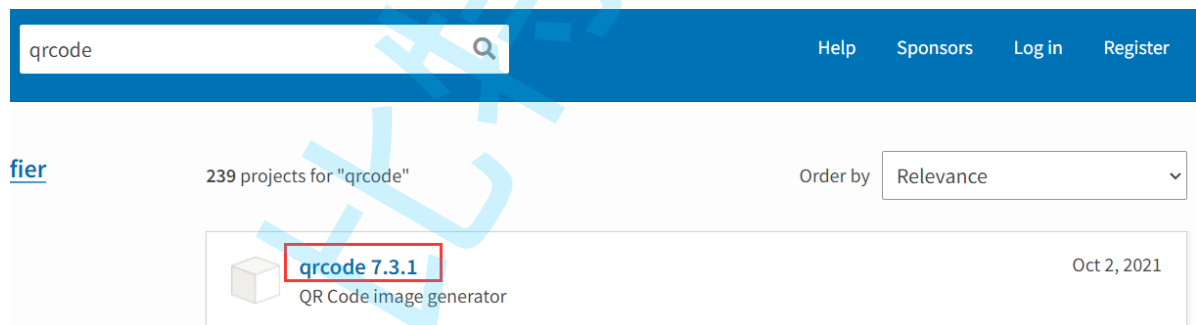
### (1) 通过搜索引擎, 确定使用哪个库

使用 python 生成二维码

得到情报, `qrcode` 这个库, 可以用来生成二维码.

### (2) 查看 qrcode 文档

在 PyPI 上搜索 `qrcode`



点击则进入 qrcode 的详情页.

文档开头描述了如何安装 qrcode

```
pip install qrcode[pil]
```

页面中央位置描述了 qrcode 库的使用方法

```
import qrcode
img = qrcode.make('Some data here')
type(img) # qrcode.image.pil.PilImage
img.save("some_file.png")
```

### (3) 使用 pip 安装

```
pip install qrcode[pil]
```

**注意:** pip 安装的时候可能会有警告, 提示使用的 pip 版本太低. 这个警告我们不必处理, 不影响我们正常使用.

### (4) 编写代码

按照文档给出的示例, 模仿一段代码.

```
import qrcode

img = qrcode.make('汤老湿真帅')
img.save('qrcode.png')
```

运行完毕后, 得到结果如下:



使用微信或者其他工具扫码, 即可看到二维码内容.

所谓二维码, 本质上就是使用黑白点阵表示一个字符串.

我们日常使用的二维码内部一般是一个 URL, 扫码后会自动跳转到对应的地址, 从而打开一个网页.

## 代码示例: 操作 excel

读取 excel 可以使用 `xlrd` 模块. 文档地址: <https://xlrd.readthedocs.io/en/latest/>

修改 excel 可以使用 `xlwt` 模块. 文档地址: <https://xlwt.readthedocs.io/en/latest/>

此处以 `xlrd` 为例, 演示 excel 的基本操作.

**需求** 有如下 excel 表格 `d:/test.xlsx`

	A	B	C
1	姓名	班级	分数
2	张三	100	68
3	李四	100	76
4	王五	100	74
5	赵六	101	89
6	田七	101	75
7	周八	100	90
8	刘九	101	57

求 100 班的同学的平均分.

虽然 excel 自身支持很强大的功能, 也可以求和, 求平均值. 但是如果是稍微复杂的需求, 操作起来可能就没那么方便了.

## 1) 安装 xlrd

```
pip install xlrd==1.2.0
```

**注意:** 此处要指定版本号安装. 如果不指定版本号, 则安装最新版. 最新版里删除了对 `xlsx` 格式文件的支持.

## 2) 编写代码

- 使用 `open_workbook` 方法打开一个 excel 文件.
- 使用 `xlsx.sheet_by_index(0)` 获取到 0 号标签页.
- 使用 `table.nrows` 获取到表格的行数.
- 使用 `table.cell_value(row, col)` 获取到表格中 row, col 位置的元素值.

```
import xlrd

# 1. 打开 xlsx 文件
xlsx = xlrd.open_workbook('d:/test.xlsx')
# 2. 获取 0 号标签页. (当前只有一个标签页)
table = xlsx.sheet_by_index(0)
# 3. 获取总行数
nrows = table.nrows
# 4. 遍历数据
count = 0
total = 0
for i in range(1, nrows):
    # 使用 cell_value(row, col) 获取到指定坐标单元格的值.
    classId = table.cell_value(i, 1)
    if classId == 101:
        total += table.cell_value(i, 2)
        count += 1
print(f'平均分: {total / count}')
```



# 代码示例: "程序猿鼓励师"

监听键盘按键, 每按键 20 下, 就自动播放一个音频, 鼓励一下辛苦搬砖的自己.

## 1) 安装第三方依赖

- `pynput` 用于监听键盘按键. 注意版本不要用最新.
- `playsound` 用于播放音频.

```
pip install pynput==1.6.8
pip install playsound==1.2.2
```

## 2) 准备音频文件

此处准备了一个 `ding.mp3` 放到和 py 代码同级目录中.

## 3) 编写代码

- 使用 `from import` 的格式直接导入模块中的指定对象/函数.
- 使用 `keyboard.Listener` 监听键盘按键. 其中 `on_release` 会在释放按键时被调用.
- 使用 `listener.start` 启动监听器. 为了防止程序直接退出, 使用 `listener.join` 让程序等待用户按键.
- 使用 `count` 计数, 每隔 10 次, 调用 `playsound` 播放音频文件.

```
from pynput import keyboard
from playsound import playsound

count = 0

def on_release(key):
    print(key)
    global count
    count += 1
    if count % 10 == 0:
        playsound('ding.mp3')

listener = keyboard.Listener(
    on_release=on_release)
listener.start()
listener.join()
```

运行程序, 即可感受到效果.

## 4) 改进代码

上述代码在执行过程中, 会感觉到播放音频会导致按键卡顿. 可以使用多线程解决这个问题.

关于多线程的知识, 在此处不详细介绍.

- 使用 `threading.Thread` 引入多线程类.
- 使用 `Thread` 的构造函数来构造一个线程. `target` 表示线程要执行的任务, `args` 表示 `target` 中要调用函数的参数.
- 使用 `Thread.start()` 启动线程.

```
from pynput import keyboard
from playsound import playsound
from threading import Thread

count = 0

def on_release(key):
    print(key)
    global count
    count += 1
    if count % 10 == 0:
        t = Thread(target=playsound, args=('ding.mp3',))
        t.start()
        #playsound('ding.mp3')

listener = keyboard.Listener(
    on_release=on_release)
listener.start()
listener.join()
```

## 综合案例: 学生管理系统

### 需求说明

实现一个命令行版本的学生管理系统

功能:

- 新增学生
- 显示学生
- 查找学生
- 删除学生
- 存档到文件

### 创建入口函数

- 使用一个全局列表 `students` 表示所有学生信息.
- 使用 `menu` 函数和用户交互. 这是一个自定义函数.
- 使用 `insert`, `show`, `find`, `delete` 这几个自定义函数完成增删查操作.
- 使用 `sys.exit` 实现程序退出.

```
# 使用列表表示所有的学生
students = []
```

```
def main():
    """
    程序的入口函数
    """

    print('+-----+')
    print('|      欢迎来带学生管理系统!      |')
    print('+-----+')
    while True:
        choice = menu()
        if choice == 0:
            sys.exit()
        if choice == 1:
            insert()
        elif choice == 2:
            show()
        elif choice == 3:
            find()
        elif choice == 4:
            delete()
        else:
            print('您的输入有误! 请重新输入!')

main()
```

## 实现菜单函数

```
def menu():
    """
    显示程序菜单
    """

    print(" 1. 新增学生信息")
    print(" 2. 显示所有同学信息")
    print(" 3. 根据名字查找学生信息")
    print(" 4. 删除学生信息")
    print(" 0. 退出程序")
    choice = input(" 请输入您的选择: ")
    return int(choice)
```

## 实现增删查操作

### 1. 新增学生

```
def insert():
    print("[新增学生] 开始!")
    studentId = input("请输入学生的学号: ")
    name = input("请输入学生的姓名: ")
    gender = input("请输入学生的性别: ")
    if gender not in ('男', '女'):
        print("性别不符合要求! 新增学生失败!")
```

```

        return
className = input("请输入学生的班级：")
# 使用一个字典表示学生信息
student = {
    'studentId': studentId,
    'name': name,
    'gender': gender,
    'className': className
}
# 把字典添加到学生列表中
global students
students.append(student)
print("[新增学生] 完毕!")

```

## 2. 显示学生

```

def show():
    print("[显示学生] 开始!")
    for s in students:
        print(f"
[{s['studentId']}] \t {s['name']} \t {s['gender']} \t {s['className']}")
    print(f"[显示学生] 完毕! 共显示了 {len(students)} 条记录!")

```

## 3. 查找学生

```

def find():
    print("[查找学生] 开始!")
    name = input("请输入要查找的同学姓名：")
    count = 0
    for s in students:
        if name == s['name']:
            print(f"
[{s['studentId']}] \t {s['name']} \t {s['gender']} \t {s['className']}")
            count += 1
    print(f"[查找学生] 完毕! 共查找到 {count} 条记录!")

```

## 4. 删除学生

```

def delete():
    print("[删除学生] 开始!")
    studentId = input("请输入要删除的同学学号：")
    count = 0
    for s in students:
        if studentId == s['studentId']:
            print(f"删除 {s['name']} 同学的信息!")
            students.remove(s)
            count += 1
    print(f"[删除学生] 完毕! 共删除 {count} 条记录!")

```

# 加入存档读档

## 1. 约定存档格式

约定存档文件放到 `d:/record.txt` 文件中.

并且以行文本的方式来保存学生信息. 格式如下:

```
学号\t名字\t性别\t班级  
学号\t名字\t性别\t班级  
学号\t名字\t性别\t班级
```

- 每个同学占一行.
- 每个同学的信息之间使用 `\t` 制表符进行分隔.

## 2. 实现存档函数

```
def save():  
    """  
    存档函数  
    """  
    with open('d:/record.txt', 'w') as f:  
        for s in students:  
            f.write(f"  
{s['studentId']}\t{s['name']}\t{s['gender']}\t{s['className']}\n")  
        print(f"[存档成功] 共存储了 {len(students)} 条记录!")
```

在 `insert` 和 `delete` 末尾, 调用 `save` 函数进行存档.

```
# 执行存档  
save()
```

## 3. 实现读档函数

```
def load():  
    """  
    读档函数  
    """  
    # 如果存档文件不存在, 则跳过读档环节  
    if not os.path.exists('d:/record.txt'):  
        return  
    # 先清空全局变量里的数据  
    global students  
    students = []  
    with open('d:/record.txt', 'r') as f:  
        for line in f:  
            # 去除末尾的换行符  
            line = line.strip()
```

```

tokens = line.split('\t')
if len(tokens) < 4:
    print(f"文件格式有误! line={line}")
    continue
student = {
    'studentId': tokens[0],
    'name': tokens[1],
    'gender': tokens[2],
    'className': tokens[3]
}
students.append(student)
print(f"[读档成功] 共读取了 {len(students)} 条记录!")

```

在 `main` 函数开头的地方, 调用 `load` 加载存档.

```
load()
```

## 打包成 exe 程序发布

当前虽然已经实现了一个管理系统, 但是 `.py` 的文件只能在安装了 Python 环境的机器上运行.

为了能够更好的部署到其他主机上, 可以借助 `pyinstaller` 来把 Python 程序打包成 exe 程序.

### 1. 安装 pyinstaller

```
pip install pyinstaller
```

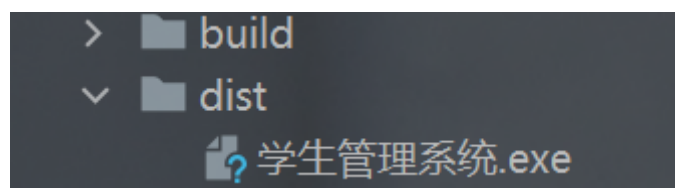
### 2. 打包程序

- `-F` 表示打包成单个 exe (不带动态库)

```
pyinstall -F 学生管理系统.py
```

**注意:** 如果提示找不到 `pyinstaller` 命令, 则需要重启一下 PyCharm.

稍等片刻, 很快打包完成.





此时就可以把这个程序拷贝给其他机器使用了. 无需 Python 环境即可运行.

## 后续扩展

### python cookbook

python 经典进阶书籍. 针对各种典型场景提供了一些解决方案.



### awesome-python

整理了 Python 的一些非常实用的程序库.

<https://gitee.com/awesome-lib/awesome-python>

## 500 Lines or Less

使用简短的 Python 代码来实现一些有意思的程序.

<https://github.com/aosabook/500lines>

比特就业课