可执行的 Python 文件

理论上来讲，你最终可以把任何一个程序，无论大小，都封装（或者囊括）到仅仅一个函数之中。按照惯例（Convention），这个函数的名称叫做 `main()`：

```
def routine_1():
    print('Routine 1 done.')

def routine_2():
    sub_routine_1()
    sub_routine_2()
    print('Routine 2 done.')

def sub_routine_1():
    print('Sub-routine 1 done.')

def sub_routine_2():
    print('Sub-routine 2 done.')

def main():
    routine_1()
    routine_2()
    print('This is the end of the program.')

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

```
Routine 1 done.
Sub-routine 1 done.
Sub-routine 2 done.
Routine 2 done.
This is the end of the program.
```

当一个模块（其实就是存有 Python 代码的 `.py` 文件）被导入，或者被执行的时候，这个模块的 `__name__` 被设定为 `__main__`。

把一个程序整个封装到 `main()` 之中，而后在模块代码里加上：

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

这么做的结果是：

1. 当 Python 文件被当作模块，被 `import` 语句导入时，`main()` 函数不被直接运行；

> 2. 当 Python 文件被 `python -m` 执行的时候，`main()` 才被执行。

还记得那个 Python 的彩蛋吧？`this.py` 的代码如下：

```
s = """Gur Mra bs Clguba, ol Gvz Crgref
Ornhgvshy vf orggre guna htyl.
Rkcyvpvg vf orggre guna vzcyvpvg.
Fvzcyr vf orggre guna pbzcyrk.
Pbzcyrk vf orggre guna pbzcyvpngrq.
Syng vf orggre guna arfgrq.
Fcnefr vf orggre guna qrafr.
Ernqnovyvgl pbhagf.
Fcrpvny pnfrf nera'g fcrpvny rabhtu gb oernx gur ehyrf.
Nygubhtu cenpgvpnyvgl orngf chevgl.
Reebef fubhyq arire cnff fvyragyl.
Hayrff rkcyvpvgyl fvyraprq.
Va gur snpr bs nzovthvgl, ershfr gur grzcgngvba gb thrff.
Gurer fubhyq or bar-- naq cersrenoyl bayl bar --boivbhf jnl gb qb vg.
Nygubhtu gung jnl znl abg or boivbhf ng svefg hayrff lbh'er Qhgpu.
Abj vf orggre guna arire.
Nygubhtu arire vf bsgra orggre guna *evtug* abj.
Vs gur vzcyrzragngvba vf uneq gb rkcynva, vg'f n onq vqrn.
Vs gur vzcyrzragngvba vf rnfl gb rkcynva, vg znl or n tbbq vqrn.
Anzrfcnprf ner bar ubaxvat terng vqrn -- yrg'f qb zber bs gubfr!"""

d = {}
for c in (65, 97):
    for i in range(26):
        d[chr(i+c)] = chr((i+13) % 26 + c)

print("".join([d.get(c, c) for c in s]))
```

所以，只要 `import this`，`this.py` 中的代码就被执行：

```
import this
```

我在当前目录下，保存了一个文件 `that.py`，它的内容如下 —— 其实就是把 `this.py` 之中的代码封装到 `main()` 函数中了：

```
# %load that.py
def main():
    s = """Gur Mra bs Clguba, ol Gvz Crgref
    Ornhgvshy vf orggre guna htyl.
    Rkcyvpvg vf orggre guna vzcyvpvg.
    Fvzcyr vf orggre guna pbzcyrk.
    Pbzcyrk vf orggre guna pbzcyvpngrq.
    Syng vf orggre guna arfgrq.
    Fcnefr vf orggre guna qrafr.
```

```
    Ernqnovyvgl pbhagf.
    Fcrpvny pnfrf nera'g fcrpvny rabhtu gb oernx gur ehyrf.
    Nygubhtu cenpgvpnyvgl orngf chevgl.
    Reebef fubhyq arire cnff fvyragyl.
    Hayrff rkcyvpvgyl fvyraprq.
    Va gur snpr bs nzovthvgl, ershfr gur grzcgngvba gb thrff.
    Gurer fubhyq or bar-- naq cersrenoyl bayl bar --boivbhf jnl gb qb vg.
    Nygubhtu gung jnl znl abg or boivbhf ng svefg hayrff lbh'er Qhgpu.
    Abj vf orggre guna arire.
    Nygubhtu arire vf bsgra orggre guna *evtug* abj.
    Vs gur vzcyrzragngvba vf uneq gb rkcynva, vg'f n onq vqrn.
    Vs gur vzcyrzragngvba vf rnfl gb rkcynva, vg znl or n tbbq vqrn.
    Anzrfcnprf ner bar ubaxvat terng vqrn -- yrg'f qb zber bs gubfr!"""
    d = {}
    for c in (65, 97):
        for i in range(26):
            d[chr(i+c)] = chr((i+13) % 26 + c)
    print("".join([d.get(c, c) for c in s]))


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

于是，当你在其它地方导入它的时候，`import that`，`main()` 函数的内容不会被执行：

```
import that
```

但是，你在命令行中，用 `python that.py`，或者 `python -m that` 将 `that.py` 当作可执行模块运行的时候，`main()` 就会被执行 —— 注意，不要写错，`python -m that.py` 会报错的 —— 有 `-m` 参数，就不要写文件尾缀 `.py`：

```
%%bash
python that.py
```

```
%%bash
python -m that
```

像 `that.py` 那样把整个程序放进 `main()` 函数之后，`import that` 不会自动执行 main 函数里的代码。不过，你可以调用 that.main() ：

```
import that
that.main()
```

当然，`that.py` 之中没有任何 Docstring，所以 `help(that)` 的结果是这样的：

```
import that
help(that)
```

所以，之前那个从 37 万个词汇中挑出 3700 个字母加起来等于 100 的词汇的程序，也可以写成以下形式：

```
#!/usr/bin/env python

def sum_of_word(word):
    sum = 0
    for char in word:
        sum += ord(char) - 96
    return sum
def main(wordlist, result):
    with open(result, 'w') as result:
        with open(wordlist, 'r') as file:
            for word in file.readlines():
                if sum_of_word(word.strip()) == 100:
                    result.write(word)

if __name__ == '__main__':
    main('words_alpha.txt', 'results.txt')
```

至于以上代码中的第一行，`#!/usr/bin/env python` 是怎么回事，建议你自己动手解决一下，去 Google：

> `python3 script executable`

你会很快弄明白的……

另外，再搜索一下：

> `python3 script executable parameters retrieving`

你就可以把以上程序改成在命令行下能够接收指定参数的 Python 可执行文件……

顺带说，`import this` 的彩蛋有更好玩的玩法：

```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

import this
love = this
this is love                              # True
love is True                              # False
love is False                             # False
love is not True or False                 # True
love is not True or False; love is love # True True
```

```
True
False
False
True
True
True
```

在 Terminal 里输入 `python ⏎` 而后在 Interactive Shell 里逐句输入试试。`love = this` 后面的每一句，都是布尔运算，想想看为什么是那样的结果？

```
import this
love = this

this is love
# True, 试试看，id(this) 和 id(love) 是同一个值
# 即，它们的内存地址相同

love is True
# False, id(love) 和 id(True) 不是同一个值
love is False
# 同上


love is not True or False
# is not 的优先级比 or 高；所以相当于是：
# (love is not True) or False，于是返回 True

love is not True or False; love is love
# 重复一次上一句 —— `;` 是语句分隔符
# 而后 love is love 当然是 True
```

注意以下代码中，`id()` 函数的输出结果：

```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

import this
love = this
this is love
love is True
love is False
love is not True or False
love is not True or False; love is love
id(love)
id(this)
id(True)
id(False)
love is not True
```

```
True
False
False
True
True
True
4345330968
4345330968
4308348176
4308349120
True
```

Python 的操作符优先级，完整表格在这里：

> Operator precedence

Python 的更多彩蛋：

> Python Easter Eggs