第 5 章

CHAPTER 5

脚　　本

在编写程序时，直接向终端逐条输入命令的效率较低，此时可以在脚本中提前编写好需要的代码，然后直接运行脚本，即可节省大量时间。

SageMath 脚本分为多种，每种脚本的运行方式和编写方式均有区别。

5.1　Sage 文件

Sage 文件需要借助 load()函数运行。编写 Sage 文件 example.sage，代码如下：

```
#第 5 章/example.sage
print("Hello World")
print(2^3)
```

运行 Sage 文件 example.sage，代码如下：

```
sage: load("example.sage")
Hello World
8
```

5.2　spyx 文件

spyx 文件需要按照 Cython 语法编写，用于编译并运行 C 语言程序。

以 C 语言科学计算程序为例，首先编写文件 test.c，在这个文件中定义了一个 C 语言函数 add_one()，用于将一个数字加 1 并返回，代码如下：

```
/* 第 5 章/test.c */
#include <stdio.h>
int add_one(int n) {
   return n + 1;
}
```

然后编写文件 test.spyx，先在文件 test.c 中声明有 C 语言函数 add_one()，再编写 Python 语言函数 test()，用于从 Python 语言层面调用 C 语言函数 add_one()，代码如下：

```
#第 5 章/test.spyx
cdef extern from "test.c":
    int add_one(int n)

def test(n):
    return add_one(n)
```

编译文件 test.spyx,代码如下:

```
sage: attach("test.spyx")
Compiling ./test.spyx...
```

在编译成功后,即可调用编译后的 test()函数,代码如下:

```
sage: test(10)
11
```

5.3 可独立运行的脚本

SageMath 脚本可以不在命令行中运行,而直接在操作系统的终端运行。可独立运行的脚本在写法上略有不同。首先需要在脚本中写入#!/usr/bin/sage 以指定运行环境,然后还需要在脚本中写入 from sage.all import * 来导入 SageMath 的全部库,以访问 SageMath 中的变量、函数和类等属性。

编写一个独立的 SageMath 脚本,代码如下:

```
#!/usr/bin/sage
#第 5 章/standalone_script

import sys
from sage.all import *

print("hello world!")
```

然后在不借助 SageMath 的情况下运行此脚本,结果如下:

```
$ ./standalone_script
hello world!
```

5.4 脚本传参

一般而言,如果不使用其他模块处理传入的参数,就需要借助 sys.argv 参数来接收从命令行传入的变量。编写一个脚本,用于打印所有从命令行传入的变量,代码如下:

```
#!/usr/bin/sage
#第5章/print_argv.py
import sys
print(sys.argv)
```

在运行此脚本时传入参数1、2和3，运行结果如下：

```
$ sage print_argv.py 1 2 3
['print_argv.py', '1', '2', '3']
```

如果脚本需要以更复杂的方式处理传入的参数，则推荐使用argparse模块。argparse.ArgumentParser类用于解析命令行参数和选项，支持的参数如下。

(1) prog：指定程序的名称，默认为sys.argv[0]。

(2) usage：指定程序的用法信息。

(3) description：指定注释。

(4) epilog：指定程序的结尾信息。

(5) parents：指定一个ArgumentParser对象列表，用于共享参数。

(6) formatter_class：指定帮助信息的格式化类。

(7) prefix_chars：指定可接受的命令行选项前缀字符，默认为“-”。

(8) fromfile_prefix_chars：指定读取参数值的文件的前缀字符，默认为None。

(9) argument_default：指定参数的默认值，默认为None。

(10) conflict_handler：指定冲突处理策略，默认为"error"。

(11) add_help：指定是否添加"-h/--help"选项，默认值为True。

将程序的名称指定为test，将程序的用法信息指定为test usage，将注释指定为test description，将程序的结尾信息指定为test epilog，代码如下：

```
#!/usr/bin/sage
#第5章/arg_parser.py
import argparse
parser = argparse.ArgumentParser(
   prog = 'test',
   usage = 'test usage',
   description = 'test description',
   epilog = 'test epilog',
   )
```

使用argparse.ArgumentParser类创建一个ArgumentParser对象后，可以通过运行add_argument()函数来添加命令行参数和选项。add_argument()函数支持的参数如下。

(1) name or flags：指定参数的名称或选项列表。

(2) action：指定参数的动作，例如存储值、计数等。

(3) nargs：指定参数的数量。

(4) const：指定常量值。

(5) default：指定参数的默认值。

（6）type：指定参数的类型。

（7）choices：指定参数的可选值。

（8）required：指定参数是否是必需的。

（9）help：指定参数的帮助信息。

将一个参数的名称指定为 test，将参数的数量指定为＋，代码如下：

```
#第 5 章/arg_parser_2.py
parser.add_argument('test', nargs = '+')
```

将一个参数的名称指定为-t 或--test，将参数的类型指定为 int 并将默认值指定为 10，代码如下：

```
#第 5 章/arg_parser_2.py
parser.add_argument('-t', '--test', type = int, default = 10)
```

此时在运行 arg_parser_2.py 脚本时必须传入 test 参数，代码如下：

```
$ sage arg_parser_2.py test
Namespace(test = ['test'])
```

在 test 参数之后还可以追加传入若干参数。追加传入 1、2 和 3 的代码如下：

```
$ sage arg_parser_2.py test 1 2 3
Namespace(test = ['test', '1', '2', '3'])
```

如果不传入 test 参数，则报错，代码如下：

```
$ sage arg_parser_2.py
usage: test [-h] [-t TEST] test [test ...]
test: error: the following arguments are required: test
```

由于-t 或--test 参数包含默认值，因此在不显式传入此参数时默认传入“-t 10”或“--test 10”参数。显式传入“-t 5” 或“--test 5”参数，代码如下：

```
$ sage arg_parser_2.py -t 5 test
Namespace(test = ['test'])
$ sage arg_parser_2.py --test 5 test
Namespace(test = ['test'])
```