第 7 章　Blender 游戏引擎 Python 脚本设计

Python 是一门面向对象的、交互的解释型编程语言。它集成了模块、异常、动态类型、高水平的动态数据类型和类。Python 兼具强大的功能和清晰的语法。Python 脚本是一种强大而灵活的用于扩展 Blender 功能的方法。Blender 的大部分功能都可以脚本化，包括动画、渲染、导入与导出、创建物体和自动重复任务的脚本。脚本可以利用紧密集成的 API（Application Programming Interface）与 Blender 进行交互。

Python 是一个高层次的结合了解释性、编译性、互动性和面向对象的脚本语言。Python 程序具有很强的可读性，具有比其他语言更具特色的语法结构。

（1）Python 是一种解释型语言：这意味着开发过程中没有编译这个环节。类似于 PHP 语言和 Perl 语言。

（2）Python 是交互式语言：这意味着，您可以在一个 Python 提示符（>>>）后直接执行代码。

（3）Python 是面向对象语言：这意味着，Python 是支持将代码封装在对象的编程技术。

（4）Python 是初学者的语言：Python 对初级程序员而言，是一种伟大的语言，它支持广泛的应用程序开发，从简单的文字处理到 WWW 浏览器甚至游戏。

Python 是由 Guido van Rossum 于 20 世纪 80 年代末和 90 年代初，在荷兰国家数学和计算机科学研究所设计出来的。Python 本身也是由诸多其他语言发展而来的，其中包括 ABC、Modula-3、C、C++、Algol-68、SmallTalk、UNIX shell 和其他的脚本语言等。像 Perl 语言一样，Python 源代码同样遵循 GPL（GNU General Public License）协议。现在 Python 由一个核心开发团队在维护，Guido van Rossum 仍然占据着至关重要的作用并指导其进展。

Python 特点：

（1）易于学习：Python 有相对较少的关键字，及结构简单且明确定义的语法，学习起来更加简单。

（2）易于阅读：Python 代码定义得更清晰。

（3）易于维护：Python 的成功在于它的源代码是相当容易维护的。

（4）一个广泛的标准库：Python 的最大的优势之一是具有丰富的库、跨平台等优点，在操作系统 UNIX、Windows 和 Macintosh 兼容很好。

（5）互动模式：互动模式的支持，可以从终端输入执行代码并获得结果的语言，互动的测试和调试代码片段。

（6）可移植：基于其开放源代码的特性，Python 已经被移植（也就是使其工作）到许多平台。

（7）可扩展：如果需要一段运行很快的关键代码，或者是想要编写一些不愿开放的算法，可以使用 C 或 C++ 完成那部分程序，然后从 Python 程序中调用。

（8）数据库：Python 提供所有主要的商业数据库的接口。

（9）GUI 编程：Python 支持 GUI 可以创建和移植到许多系统进行调用。

（10）可嵌入：可以将 Python 嵌入到 C/C++ 程序，让程序用户获得“脚本化”的能力。

7.1 Blender 游戏引擎文本编辑器

Blender 有一个窗口类型叫文本编辑器，用于编写 Blender 脚本文件。通过窗口类型菜单选择或按快捷键 Shift + F11 即可进入该编辑器窗口。Blender 文本编辑器窗口，如图 7-1 所示。

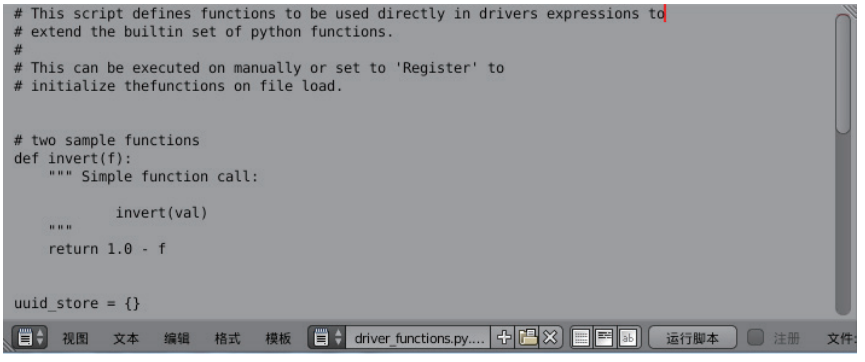

图 7-1　Blender 文本编辑器窗口

在标题栏 3 中，选择“时间线”→“文本编辑器”。显示视图、文本、模板、新建、打开以及文本功能选项等。

当单击“新建”按钮时，已经打开一个文件的文本工具栏。展开文本编辑器为视图、文本、编辑、格式、模板、打开文本、文本功能选项、运行脚本以及注册文字等。

文本编辑器类型采用标准编辑器选择按钮，菜单使用编辑器菜单，而文本用于选择文本或创建新文本的数据块菜单，使用之后标题栏将发生变化。显示紧跟着的三个按钮分别

开启显示选项：行号、文本换行和语法高亮。按快捷键 Alt + P 可运行脚本 / 脚本节点更新执行文本作为 Python 脚本。注册加载时，注册当前文本数据块为模块，扩展名必须为 *.py。更多关于 Python 模块注册的内容请参考 API 文档。

新建文本编辑器各种功能详解。

视图：包含文件底部、文件顶部、属性。文件底部：将视图和光标移动到文本的末尾，快捷键为 Ctrl + 结束；文件顶部：将视图和光标移动到文本的开头，快捷键为 Ctrl + Home；属性：切换文本属性栏显示，快捷键为 Ctrl + T。

文本：包括创建文本块、打开文本块、重载、保存、另存为、加载为内部文件、运行脚本等。创建文本块：创建一个新的内部文本；打开文本块：打开文件浏览器，载入一个文本，快捷键为 Alt + O；重载：重新打开（重新载入）当前文本缓存（会丢失所有未保存修改），快捷键为 Alt + R；保存：保存已打开文件，快捷键为 Alt + S；另存为：打开文件浏览器，保存未保存文本为文本文件，快捷键为 Shift + Ctrl + Alt + S；加载为内部文件：将文本存储在混合文件中；运行脚本：执行文本作为 Python 脚本，快捷键为 Alt + P。

编辑：包括剪切、复制、粘贴、复制行、将行上移、将行下移、选择、跳转、查找、文本自动补全、将文本转换为 3D 物体等。剪切：剪切选中文本至文本剪贴板，快捷键为 Ctrl + X；复制：复制选中文本至文本剪贴板，快捷键为 Ctrl + C；粘贴：粘贴剪贴板文本至文本窗口光标位置，快捷键为 Ctrl + V；复制行：复制当前行，快捷键为 Ctrl + D；将行上移：交换当前行与上一行，快捷键为 Shift + Ctrl + Up；将行下移：交换当前行与下一行，快捷键为 Shift + Ctrl + Down；选择：包含全选和连接行号。全选，快捷键为 Ctrl + A，连接行号，快捷键为 Shift + Ctrl + A；跳转：显示跳转弹出窗口，可以选择跳转到的行号；查找：在侧栏中显示查找面板；文本自动补全：显示文本中已有的匹配文字供选择，快捷键为 Ctrl + 空格键；将文本转换为 3D 物体：包含单一物体和每行生成一个物体。

格式：包含缩进、取消缩进、注释、取消注释、转换空格等。缩进：缩进选中行，快捷键为 Tab；取消缩进：缩进选中行，快捷键为 Shift + Tab；注释：将所选行转换为 Python 注释；取消注释：取消所选行的注释；转换空格：在标签或空格缩进之间转换。

模板：包括 Python 和开放式着色语言（OSL）。

Python 脚本使用 Blender 内置的解释器解析缓冲区的内容。Blender 配有一个内置的功能齐全的 Python 解释器，并具有许多 Blender 特有的模块，如脚本与扩展 Blender 部分所述。

7.2　Blender 游戏引擎 Python 控制台

Python 控制台因其对 Python API、历史记录和自动补全的完整访问而成为一个快速执行命令的途径。可以先通过控制台来探索脚本的各种可能性，然后将脚本粘贴到更复杂的

脚本中。

访问内置的 Python 控制台，在标题栏 3 中，选择“时间线”→“Python 控制台”。或在任何 Blender 编辑器类型（如 3D 视图，时间线等）下，按快捷键 Shift + F4 可以将其切换为控制台编辑器。命令提示符使用常用的 Python 3.x，解释器已加载并准备接受提示符（>>>）后的命令。

设置 Python 控制台背景颜色，选择“文件”→“用户设置”→“主题”→“Python 控制台”，设置窗口背景 = 1.0 1.0 1.0 白色，行输入 = 0.0 0.0 0.0 黑色。Python 控制台操作面板，如图 7-2 所示。

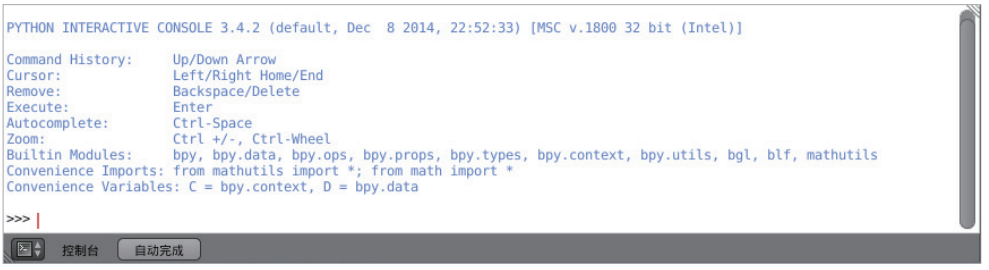

图 7-2　Python 控制台操作面板

Python 控制台常用命令：

在提示符下键入 dir() 并执行，可以检测到已经加载 Python 控制台解释器环境模块。Python 控制台操作 dir() 命令，如图 7-3 所示。

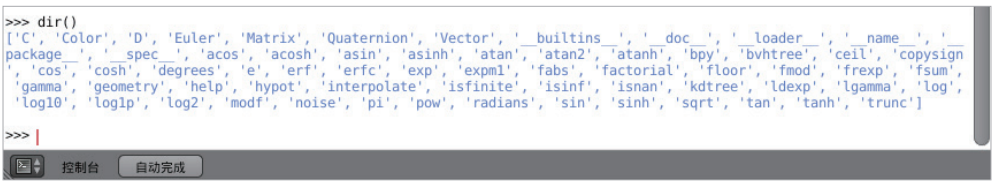

图 7-3　Python 控制台操作 dir() 命令

bpy 命令全称 Blender Python API，是 Blender 使用 Python 与系统执行数据交换和功能调用的接口模块。通过调用这个模块的函数，可以实现以下功能：代替界面操作去完成对物体的修改，如修改网格属性或添加修改器。自定义系统的相关配置，如重设快捷键或修改主题的色彩。自定义工具的参数配置，如自定义雕刻笔刷的参数。自定义用户界面，如修改面板的外观和按钮的排列效果。创建新的工具，如 Surface Sketching（表面绘制）工具。创建交互式工具，如游戏的逻辑脚本。创建与外置渲染器的接口调用，如配置 Vray 等外置渲染器。

在提示符下键入 bpy. 并执行，然后单击自动完成按钮或者按快捷键 Ctrl + 空格键，会看到控制台的自动补全功能已经生效。显示 bpy 子模块的列表，这些模块作为一组非常强

大的工具。bpy 命令接口模块函数，如图 7-4 所示。

图 7-4　bpy 命令接口模块函数

用同样的方法列出 bpy.app. 等模块的所有内容。在启用自动补全后，命令提示符上方的绿色输出。看到的是自动补全功能列出的可能结果。以上列表中所列出的内容都是模块属性名称和函数名。

如果在 3D 视图查看默认的 Blender 场景，将注意到三个物体：立方体、灯光和摄像机。

（1）所有对象的都存在上下文，以及各种模式及其对应的操作。

（2）在任何情况下，只有一个物体处于活动状态，并且可以有多个选定对象。

（3）所有物体都是 blend 文件中的数据。

（4）存在创建和修改这些对象的操作 / 函数。

对于以上所简要列出的内容，并非全部列出，要注意 bpy 模块提供了访问和修改数据的相关功能。

Python 控制台 bpy.context 命令案例：

- 在提示符下键入 >>> bpy.context.mode，将显示当前 3D 视图所处于的模式，如物体、编辑、雕刻等。
- 在提示符下键入 >>> bpy.context.object 或 bpy.context.active_object 将获得对 3D 视图中当前活动对象的访问。
- 在提示符下键入 >>> bpy.context.selected_objects 访问选择上的对象列表，可以同时选择多个对象。
- 在提示符下键入 >>> bpy.context.selected_objects[0]，访问列表中第一个对象的名称。Python 控制台 bpy.context 命令设计，如图 7-5 所示。

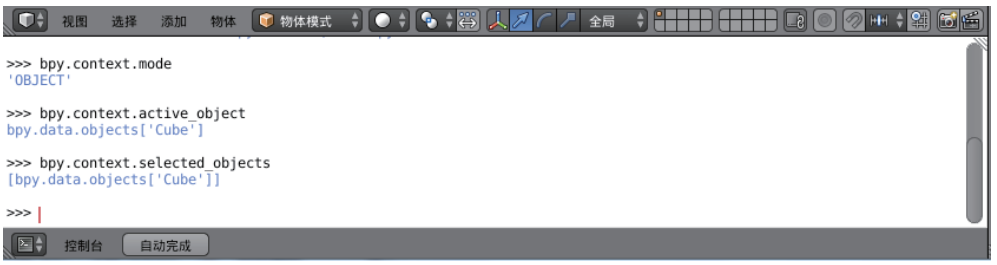

图 7-5　Python 控制台 bpy.context 命令设计

Python 控制台 bpy.context 命令控制 3D 物体移动实例：

- 在提示符下键入 >>> bpy.context.object.location.x = 2，将在 X 坐标位置移动 2 个数值单位。
- 在提示符下键入 >>> bpy.context.object.location.x + = 3，将物体从前一个 X 位置继续移动 3 个单位，3D 物体从坐标原点实际移动到 X = 5 的位置。

Python 控制台 bpy.context.object.location 命令设计，如图 7-6 所示。

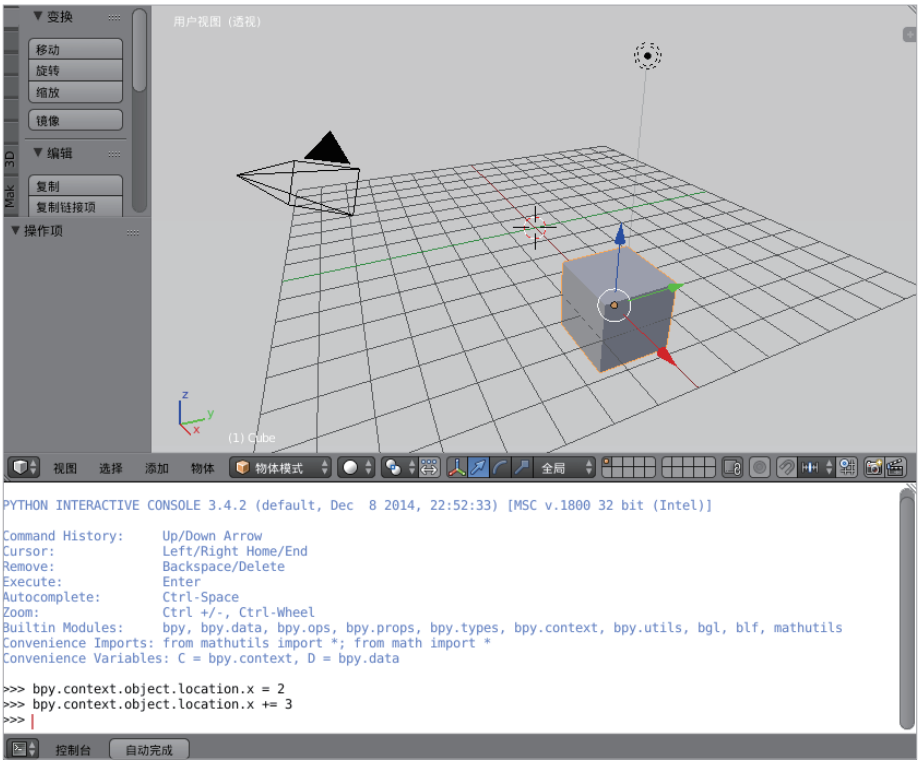

图 7-6　Python 控制台 bpy.context.object.location 命令设计

Python 控制台 bpy.context 命令修改 3D 物体坐标位置实例：

- 在提示符下键入 >>> bpy.context.object.location = (1, 2, 3)，将修改 X、Y、Z 坐标位置移动 3D 物体。
- 在提示符下键入 >>> bpy.context.object.location.xy = (5, -5)，只修改 X，Y 分量坐标位置移动 3D 物体。
- 在提示符下键入 >>> type(bpy.context.object.location)，将获得物体位置的数据类型。
- 在提示符下键入 >>> bpy.context.selected_objects，可以访问所有选定对象的全部列表。
- 在提示符下键入 >>> dir(bpy.context.object.location)，可以访问到许多的数据。

- 3D 物体从原点移动到（1, 2, 3）坐标位置，再从该点移动到 (5, -5) 坐标位置，并获得物体位置的数据类型，接着显示所有选定对象的列表。

Python 控制台 bpy.context.object.location 位置移动命令设计，如图 7-7 所示。

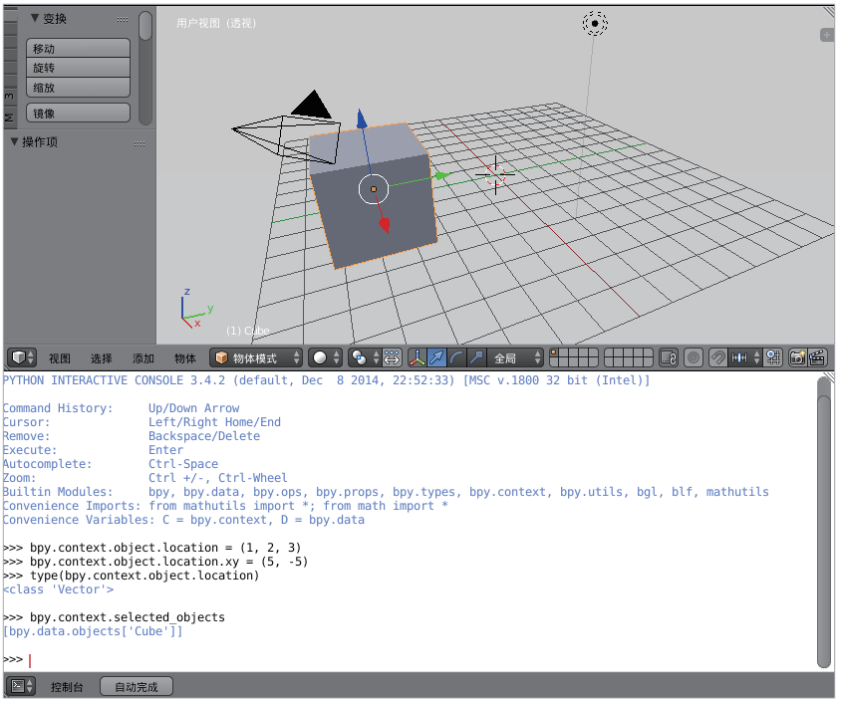

图 7-7　Python 控制台 bpy.context.object.location 位置移动命令设计

bpy.data 具有访问 .blend 文件中所有数据的函数和属性。访问当前 .blend 文件中的以下数据：对象、网格、材质、纹理、场景、窗口、声音、脚本等。

Python 控制台 bpy.data 命令控制对象、场景和材质等实例：

- 在提示符下键入 >>> bpy.data.objects，将获得数据对象信息。
- 在提示符下键入 >>> bpy.data.scenes，将获得数据场景信息。
- 在提示符下键入 >>> bpy.data.materials，将获得数据材质信息。

Python 控制台 bpy.data 命令控制，如图 7-8 所示。

图 7-8　Python 控制台 bpy.data 命令控制

Python 控制台通过集合中的方法添加和删除数据 bpy.data 案例设计：

- 在提示符下键入 >>> mesh = bpy.data.meshes.new(name = "MyMesh")，添加网格数据信息。
- 在提示符下键入 >>> print(mesh)，将显示打印网格信息。
- 在提示符下键入 >>>bpy.data.meshes.remove(mesh)，删除网格数据信息。
- 在提示符下键入 >>> print(mesh)，将显示打印网格信息。

Python 控制台通过集合中的方法添加和删除数据 bpy.data 设计，如图 7-9 所示。

图 7-9　Python 控制台通过集合中的方法添加和删除数据 bpy.data 设计

7.3　扩展 Blender 脚本功能

插件是 Blender 中用于扩展功能的脚本，可以在用户设置中启用。在 Blender 执行程序以外，还有很多其他人写的数以百计的插件。

Blender 开发版会包含一些其他测试中的插件，而官方正式版则不会有这些。这些插件中很多都能可靠工作且非常有用，但无法保证在正式版中的稳定性。

Blender 内置插件见“插件”文档。脚本除插件之外，还有其他可以用来扩展 Blender 功能的脚本：

（1）模块：用于导入其他脚本的实用工具库。

（2）预设：Blender 工具和关键配置的设置。

（3）启动文件：启动 Blender 时载入的文件。这些文件定义了大多数 Blender 的用户界面和一些附带的核心操作。

（4）自定义脚本：与插件不同，这些往往是通过文本编辑器编写的一次性脚本。

保存脚本的文件位置，所有脚本都从本地、系统和用户路径下的 scripts 文件夹载入。可以在文件路径、用户设置、文件路径设置额外的脚本查找路径。

安装脚本插件，通过 Blender 用户设置可以很方便地安装插件。单击“安装”按钮，并选择 .py 或 .zip 文件。

手动安装脚本或插件，可视具体类型将其放置到 add-ons、modules、presets 或 startup 目录。还可以在文本编辑器中载入并运行脚本。

7.4　Python 函数和内置函数设计

Python 函数和内置函数设计包含 Python 函数设计和内置函数设计两个部分。Python 函数设计包含函数的定义、函数的参数设计、返回值等信息；内置 dict() 字典函数、help() 帮助函数、setattr() 函数、dir() 模块的属性列表、hex() 十六进制转换函数等功能设计。

7.4.1　Python 函数设计

Python 函数方法设计原则是指在函数设计中，每个函数功能只完成一件事、函数简练、使用输入、输出参数以及返回 return 语句等。

Python 函数主要用于以下两种情况：

（1）代码模块重复时，必须考虑用到函数降低程序的冗余度。

（2）代码模块复杂时，可以考虑用到函数增强程序的可读性。

当程序代码非常繁杂时，就要考虑使用函数。在 Python 中做函数设计，如何将函数组合在一起，主要考虑到函数大小、聚合性、耦合性三个方面，这三者应该归结于规划与设计的范畴。高聚合、低耦合是 Python 函数设计的总体原则，具体而言有以下几点：

（1）如何将任务分解成更有针对性的函数从而增强聚合性。

（2）如何设计函数间的通信又涉及耦合性。

（3）如何设计函数的大小从而加强聚合性、降低耦合性。

1. 函数的基本定义

```
def 函数名称（参数）
        执行语句
        return 返回值
```

def：定义函数的关键字。

函数名称：顾名思义，就是函数的名字，可以用来调用函数，不能使用关键字来命名，最好是用这个函数的功能的英文名命名，可以采用驼峰法与下画线法。

参数：用来给函数提供数据，有形参和实参的区分。

执行语句：也称为函数体，用来编写一系列的程序设计语句和逻辑运算。

返回值：执行完函数后，返回给调用者的数据，默认为 None，所以没有返回值时，

可以不写 return。

2. 函数的普通参数

函数的参数最直接的是一对一关系的参数关系。

函数的普通参数设计：

```
def  fun_ex(a,b):              #a,b 是函数 fun_ex 的形式参数，也叫形参
          sum=a+b
          print('sum =',sum)
          fun_ex(2,3)          #1,3 是函数 fun_ex 的实际参数，也叫实参

# 运行结果
sum = 5
```

3. 函数的默认参数

给函数参数定义一个默认值，如果调用函数时，没有给指定参数，则函数使用默认参数，默认参数需要放在参数列表的最后。

函数的默认参数设计：

```
def fun_ex(a,b=7):    # 默认参数放在参数列表最后，如 b=6 只能在 a 后面
          sum=a+b
          print('sum =',sum)
          fun_ex(1,2)
          fun_ex(1)
# 运行结果
sum = 3
sum = 8
```

4. 函数的返回值

函数的返回值是指运行一个函数，一般都需要从函数的运算结果中得到某个信息，这时就需要使用 return 来获取返回值。

函数的返回值设计：

```
def fun_ex(a,b):
          sum=a+b
          return sum      # 返回 sum 值

re=fun_ex(2,3)
print('sum =',re)

# 运行结果
sum = 5
```

7.4.2 Python 内置函数设计

Python 内置了一些非常巧妙而且功能强大的函数，这些内置函数都是经典的而且是经过严格测试的。使用内置函数后，代码不仅简洁易读了很多，还可以省下很多时间，减少用户的编程工作量。详细的内置函数如表 7-1 所示。

表 7-1 内置函数

abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

Python 内置函数可用于数学计算、数据类型转换、字符串处理等。

1. 数学计算

（1）在提示符下键入 >>> abs(−15)，计算绝对值。

（2）在提示符下键入 >>> max([1,2,3,4,5])、min([1,2,3,3,4,5])，计算最大、最小值。

（3）在提示符下键入 >>> len('abcdefg')、len([1,2,3,4,5])、len((1,2,3,4,5))，计算序列长度。

（4）在提示符下键入 >>> round(10)//1.0，计算浮点数。

Python 常用内置函数学计算设计，如图 7-10 所示。

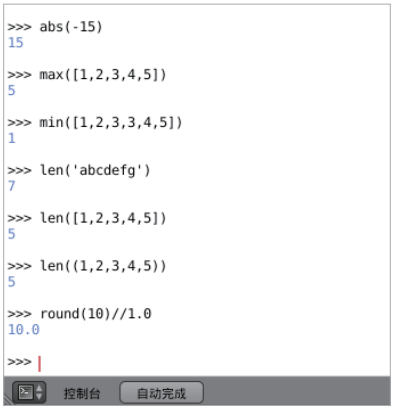

图 7-10　Python 常用内置函数学计算设计

2. 数据类型转换

（1）在提示符下键入 >>> x = 100、int(x)，数据类型转换为整形。

（2）在提示符下键入 >>> x = 200、float(x)，数据类型转换为浮点数。

（3）在提示符下键入 >>> x = 300、complex(x)，数据类型转换为复数。

（4）在提示符下键入 >>> x = 500、hex(x)，将数值转换为十六进制数。

（5）在提示符下键入 >>> x = 500、oct(x)，将数值转换为十进制数。

（6）在提示符下键入 >>> x = 65、chr(x)，返回 x 对应的字符。

（7）在提示符下键入 >>> x = 'A'、ord(x)，返回字符对应的 ASC 码数字编号。

Python 常用数据类型转换设计，如图 7-11 所示。

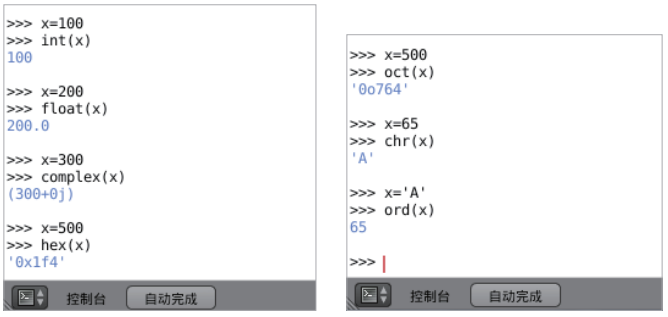

图 7-11　Python 常用数据类型转换设计

3. 字符串处理

（1）把字符串的字母全部变成大写：upper。在提示符下键入 >>>'hello World'.upper()。

（2）把字符串的字母全部变成小写：lower。在提示符下键入 >>>'HELLO World'.lower()。

（3）修改首字母大写：str.capitalize。在提示符下键入 >>>'hello'.capitalize()。

（4）字符串替换：str.replace。在提示符下键入 >>> a = 'hello world'、a.replace('l', 'a', 2)，需要传三个参数，第三个参数为替换次数。

（5）字符串切割：str.split。在提示符下键入 >>> a = 'hello world'、a.split('w')，需要传二个参数，第二个参数为切割次数。

Python 内置字符串处理设计，如图 7-12 所示。

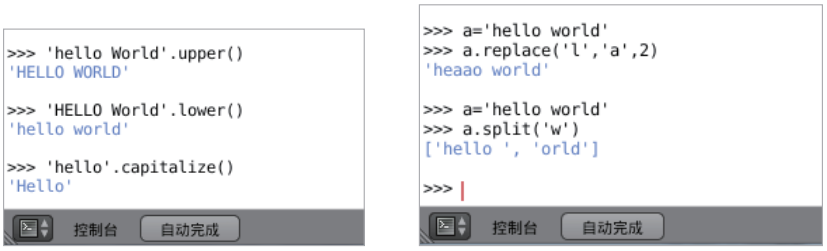

图 7-12　Python 内置字符串处理设计

7.5　Python 脚本创建 3D 模型案例设计

使用 Python 脚本创建 3D 模型群设计，步骤如下：

- 启动 Blender 仿真游戏引擎，显示默认立方体。
- 在标题栏 3 中，添加一个文本，选择“时间线”→“文本编辑器”→“新建”。
- 在文本编辑中，输入 Python 脚本创建 3D 模型，在对应 x、y、z 坐标范围内随机创建 200 个立方体。

```
import bpy
from random import randint
bpy.ops.mesh.primitive_cube_add()
# 创建模型的数量
count = 200
for c in range(0,count):
    x = randint(-100,100)
    y = randint(-100,100)
    z = randint(-100,100)
    bpy.ops.mesh.primitive_cube_add(location=(x,y,z))
```

- 在标题栏 3 中，单击运行脚本按钮，或按快捷键 Alt + P，会生成 200 个立方体模型，如图 7-13 所示。
- 在标题栏 3 中，选择“文本”→“另存为”，将内容保存到 blender-python-cube-1-1 文件中。

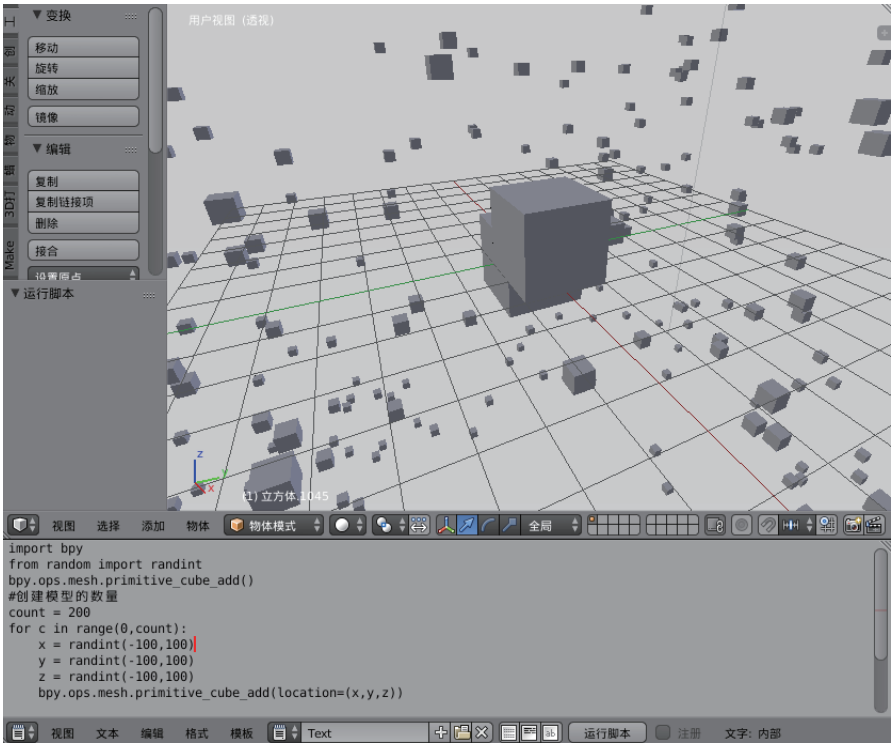

图 7-13　用 Python 脚本创建 3D 模型群设计

7.6　Python 脚本构建点、线、面模型案例设计

Python 脚本可以利用点、线、面创建 3D 网格造型。Python 脚本构建几何网格数据原理如下：

Python 脚本构建网格数据“点”：假设三维空间中有两个点，坐标为：verts = [(0,0,0), (1,0,0)]。Blender 的 Python 脚本中点的坐标是作为列表数据存储的，列表中每个点依次赋予一个索引值（Index），上面两个点的索引值就分别为 0 和 1。

Python 脚本构建网格数据“线”：在 Blender 的 Python 脚本数据中，上述两个点的连线表示为：edge = [[0,1]]，Blender 的 Python 脚本中线的数据也是作为列表存储的，两点成线，取两个顶点的索引值，得到了一条位于 X 轴的线段。

Python 脚本构建网格数据“面”：由三个点才能构成一个平面，所以要得到一个面，必须至少有 3 个点：verts = [(0,0,0),(1,0,0), (0,1,0)]，然后用一个索引列表把这三个点连起来，得到一个面的索引列表：face = [[0,1,2]]。Blender 的 Python 脚本中面的数据也是作为列表存储的，把一个面包含的所有点作为面数据的一个子列表，Blender 会自动将其闭合，然后得到一个位于 XY 平面的三角面。

下面用 Python 脚本创建一个金字塔模型，金字塔模型是由 5 个点组成的，具有 4 个

三角面和 1 个矩形面，步骤如下：

- 启动 Blender 仿真游戏引擎，删除默认立方体。
- 在标题栏 3 中，添加一个文本，选择“时间线”→“文本编辑器”→“新建”。
- 在文本编辑中，输入 Python 脚本顶点、边线、面程序，创建一个金字塔 3D 模型。

Python 脚本源程序代码如下。

```
import bpy # 加载 bpy，导入 Blender 仿真引擎 Python 脚本
# 顶点
verts = [(1,1,0),
        (-1,1,0),
        (-1,-1,0),
        (1,-1,0),
        (0,0,2)]
# 边
edges = [(0,1),
        (1,2),
        (2,3),
        (3,0),
        (0,4),
        (1,4),
        (2,4),
        (3,4)]
# 面
faces = [(0,1,4),
        (1,2,4),
        (2,3,4),
        (3,0,4),
        (0,1,2,3)]
mesh = bpy.data.meshes.new('Pyramid_Mesh')          # 新建网格
mesh.from_pydata(verts, edges, faces)               # 载入网格数据
mesh.update()                                       # 更新网格数据
pyramid = bpy.data.objects.new('Pyramid', mesh)     # 新建物体 "Pyramid"，并使
                                                      用 "mesh" 网格数据
scene = bpy.context.scene
scene.objects.link(pyramid)                         # 将物体链接至场景
```

- 在标题栏 3 中，单击运行脚本按钮或按快捷键 Alt + P，会生成一个金字塔 3D 模型，如图 7-14 所示。
- 如果将载入网格数据代码中的面数据改为空列表（[]）：mesh.from_pydata(verts，edges，[])，再运行脚本，可以在 3D 视图窗口中将获得线框模型的金字塔造型。
- 在标题栏 3 中，选择“文本”→“另存为”，将内容保存到 blender-python- 点线面 -1-1 文件中。

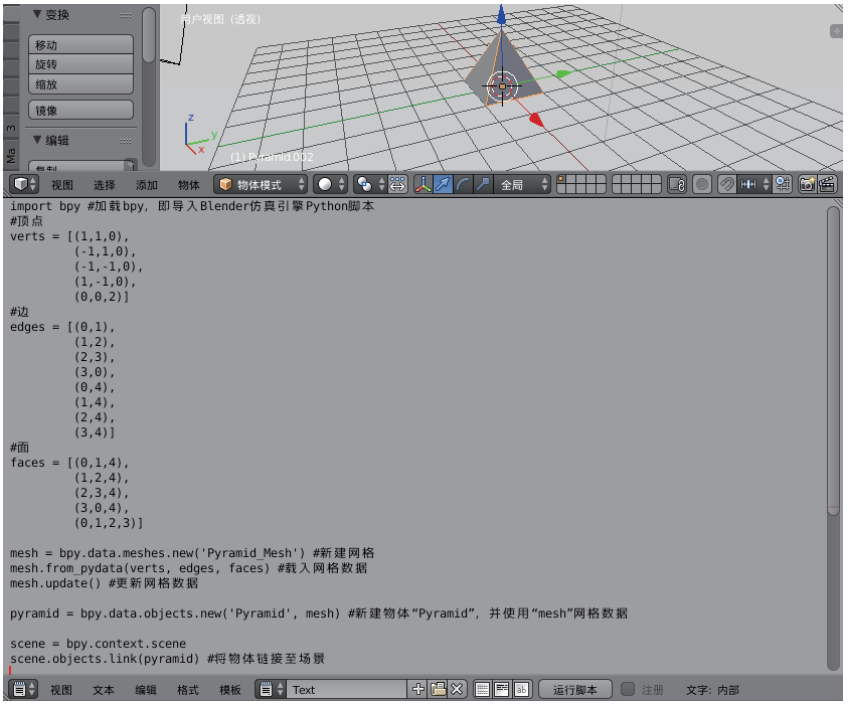

图 7-14　用 Python 脚本创建一个金字塔 3D 模型设计

7.7　Python 脚本创建一个平面案例设计

使用 Python 脚本创建一个平面。先创建一个几何体网格平面，再定义三维坐标来确定该平面位置，步骤如下：

- 启动 Blender 仿真游戏引擎，删除默认立方体。
- 在 3D 视图右上角拖动一个视图窗口。
- 在第 2 个视图窗口中，选择“文本编辑器”→“新建”→“编写源代码 / 粘贴源代码”。

Python 脚本源程序代码如下。

```
    1 import bpy                    # 加载 bpy，即导入 Blender 仿真引擎 Python 脚本
    2 from bpy import context       # 从 bpy 导入上下文
    3 from math import sin, cos, radians        # 从数学导入正弦，余弦，弧度
    4 x = 0                                     # 为变量 x 赋初值
    5 y = 0                                     # 为变量 y 赋初值
    6 z = 2                                     # 为变量 z 赋初值
    7 bpy.ops.mesh.primitive_plane_add(radius=1, view_align=False,
    8 enter_editmode=False, location=(x, y, z),
    9 layers=(True, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False))
```

第 1 行代码：加载 Blender Python 模块，即导入 Blender 仿真引擎 Python 脚本。

第 2 行代码：从 Blender Python 模块中导入上下文。

第 3 行代码：从数学模块中导入正弦、余弦以及弧度函数。

第 4 ～ 6 行代码：为变量赋初值，即 x = 0、y = 0、z = 2。

第 7 行代码：为几何平面添加一个半径 = 1，视图对齐 = False。

第 8 行代码：输入编辑模式 = False，位置 =（x，y，z）。

第 9 行代码：层 =（真，假，假，假，假，假，假，假，假，假，假，假，假，假，假，假，假，假，假，假）。

- 单击运行脚本按钮。在第 1 个 3D 视图中，显示一个平面造型；在第 2 个视图窗口中，显示 Python 脚本程序，如图 7-15 所示。

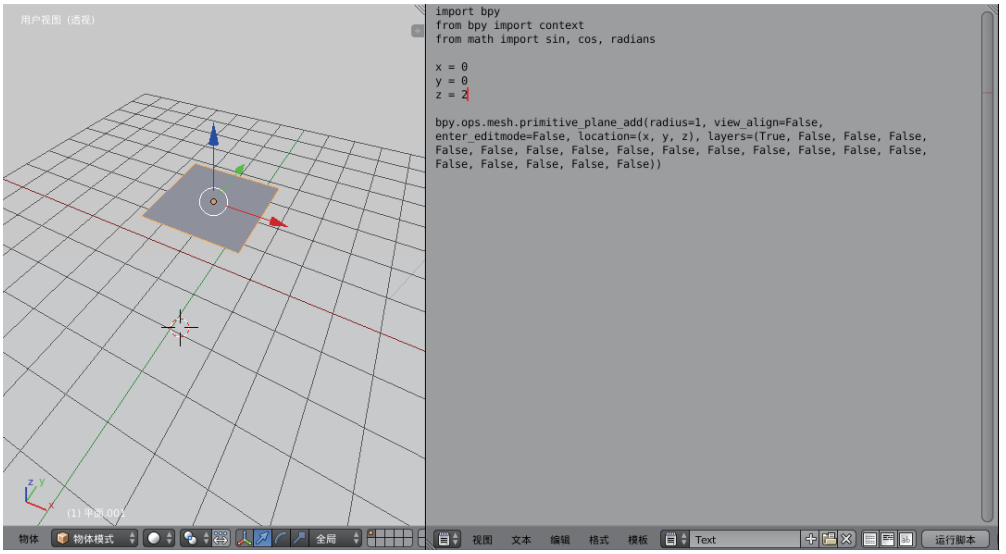

图 7-15　用 Python 脚本创建一个平面设计

7.8　Python 脚本创建一个圆锥体和圆锥棱台案例设计

利用 Python 脚本创建一个圆锥体和圆锥棱台，步骤如下：

- 启动 Blender 仿真游戏引擎，删除默认立方体。
- 在标题栏 3 中，选择“文本编辑器”→“新建”→“编写源代码 / 粘贴源代码”。

Python 脚本源程序代码如下。

```
1 import bpy
2 from bpy import context
3 from math import sin, cos, radians
4 x = 0
5 y = 0
```

```
    6 z = 1
    7 bpy.ops.mesh.primitive_cone_add(vertices=32, radius1=2, radius2=0.0,
depth=2, end_fill_type='NGON', view_align=False, enter_editmode=False,
location=(x,y,z), rotation=(0, 0 ,0))
```

第 1 行代码：加载 Blender Python 模块，即导入 Blender 仿真引擎 Python 脚本。

第 2 行代码：从 Blender Python 模块中导入上下文。

第 3 行代码：从数学模块中导入正弦、余弦以及弧度函数。

第 4 ～ 6 行代码：为变量赋初值，即 x = 0、y = 0、z = 1。

第 7 行代码：添加一个几何网格圆锥体，并赋值为顶点 = 32，半径 1 = 2，半径 2 = 0.0，深度 = 2，结束填充 = 'NGON'，视图对齐 = False，输入编辑模式 = False，位置 =（x，y，z），旋转 =（0，0，0）

- 单击运行脚本按钮。在 3D 视图中，实现一个圆锥体造型设计，在下面文本编辑器中，显示 Python 脚本程序，如图 7-16 所示。

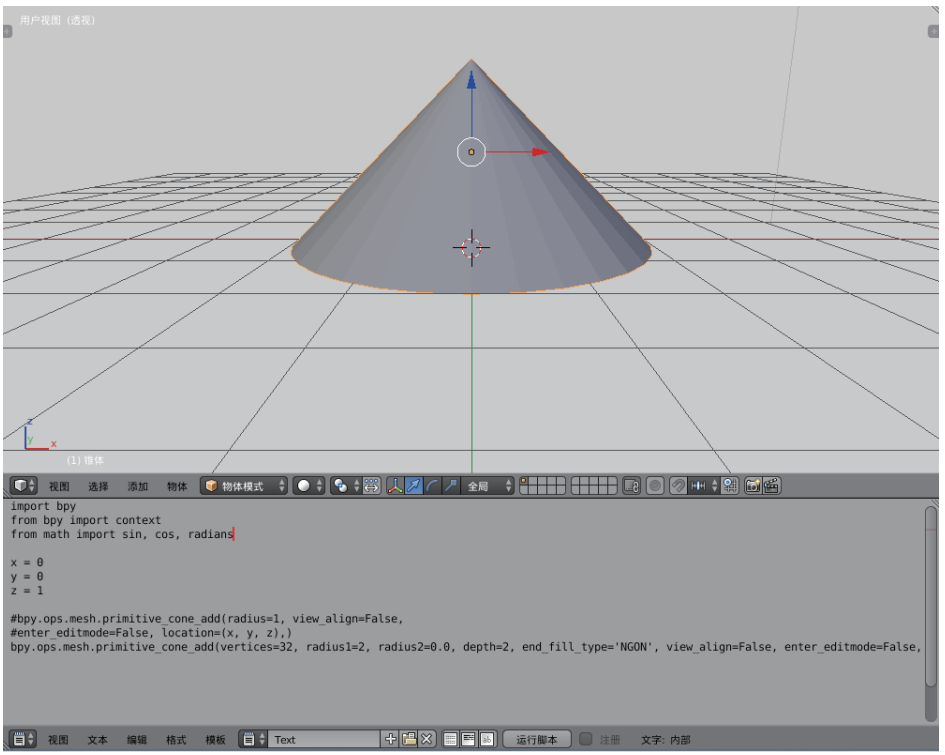

图 7-16　用 Python 脚本创建一个圆锥体设计

- 将圆锥体改变为圆锥棱台。修改第 7 行参数源代码为：

```
    7 bpy.ops.mesh.primitive_cone_add(vertices=32, radius1=2, radius2=1.0,
depth=2, end_fill_type='NGON', view_align=False, enter_editmode=False,
location=(x,y,z), rotation=(0,0 ,0))
```

第 7 行代码：添加一个几何网格圆锥体，并赋值为顶点 = 32，半径 1 = 2，半径 2 = 1.0，深度 = 2，结束填充 = 'NGON'，视图对齐 = False，输入编辑模式 = False，位置 =（x，y，z），旋转 =（0，0，0）

- 单击运行脚本按钮。在 3D 视图中，实现一个圆锥棱台造型设计，在下面文本编辑器中，显示 Python 脚本程序，如图 7-17 所示。

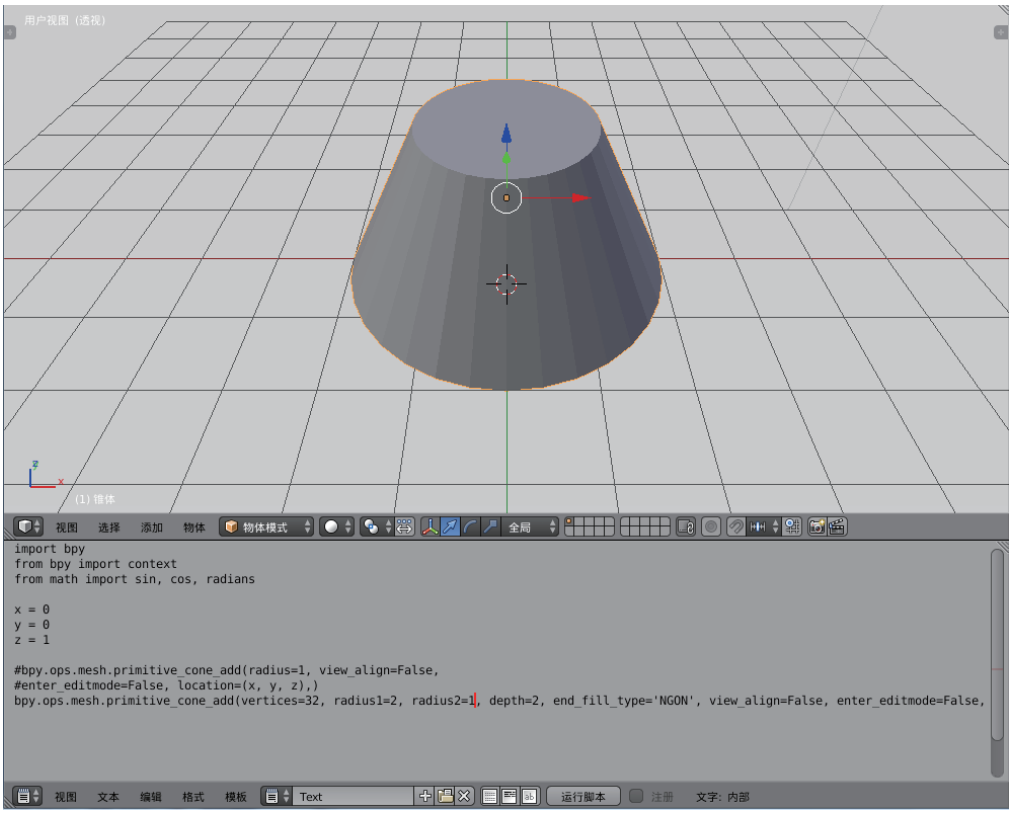

图 7-17　用 Python 脚本创建一个圆锥棱台设计

7.9　Python 脚本创建一个圆柱体案例设计

利用 Python 脚本创建一个圆柱体，步骤如下：

- 启动 Blender 仿真游戏引擎，删除默认立方体。
- 在标题栏 3 中，选择“文本编辑器”→“新建”→“编写源代码 / 粘贴源代码”。

Python 脚本源程序代码如下。

```
1 import bpy
2 import math
3 from math import *
4 x = 0
```

```
    5 y = 0
    6 z = 0
    7 bpy.ops.mesh.primitive_cylinder_add(vertices=16, radius=1, depth=2,
location=(x,y,z),end_fill_type='NGON', view_align=False, enter_editmode=False)
```

第 1 行代码：加载 Blender Python 模块，即导入 Blender 仿真引擎 Python 脚本。

第 2 行代码：从 Blender Python 模块中导入数学模块。

第 3 行代码：从数学模块中导入全部函数。

第 4 ～ 6 行代码：为变量赋初值，即 x = 0、y = 0、z = 0。

第 7 行代码：添加一个几何网格圆柱体，并赋值为顶点 = 16，半径 = 1，深度 = 2，结束填充类型 = 'NGON'，视图对齐 = False，输入编辑模式 = False。

- 单击运行脚本按钮。在 3D 视图中，实现一个圆柱体造型设计；在下面文本编辑器中，显示 Python 脚本程序，如图 7-18 所示。

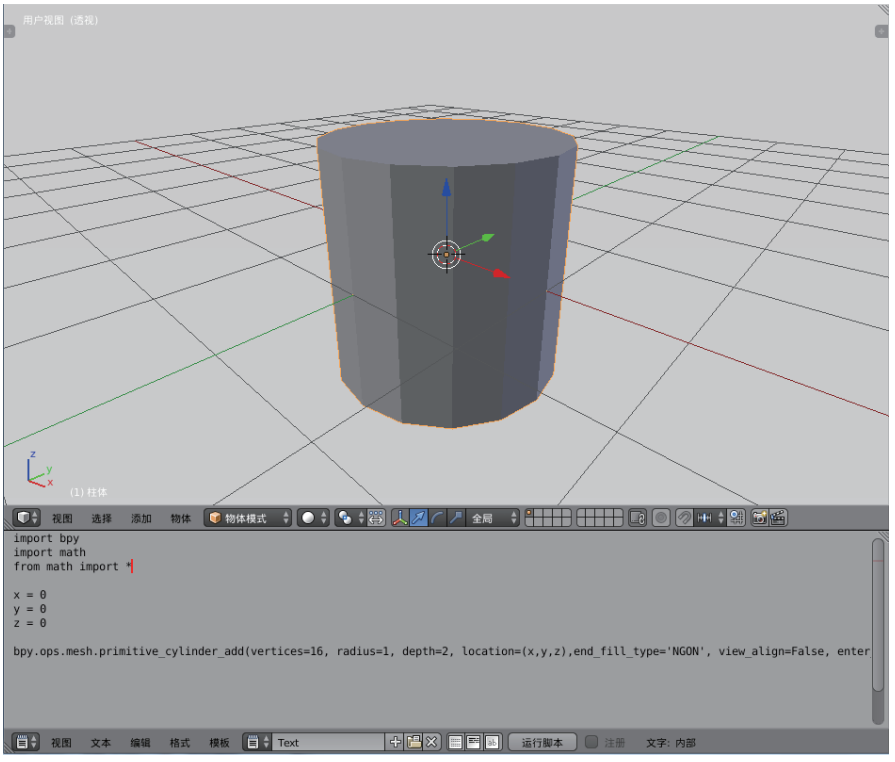

图 7-18　用 Python 脚本创建一个圆柱体设计

- 将物体模式改变为编辑模式。修改第 7 行参数源代码为：

```
    7 bpy.ops.mesh.primitive_cylinder_add(vertices=16, radius=1, depth=2,
location=(x,y,z),end_fill_type='NGON', view_align=False, enter_editmode=True)
```

第 7 行代码：添加一个几何网格圆锥体，并赋值为顶点 = 16，半径 = 1，深度 = 2，

结束填充类型 = 'NGON'，视图对齐 = False，输入编辑模式 = True。

- 单击运行脚本按钮。在 3D 视图中，实现圆柱体造型编辑模式，在下面文本编辑器中，显示 Python 脚本程序，如图 7-19 所示。

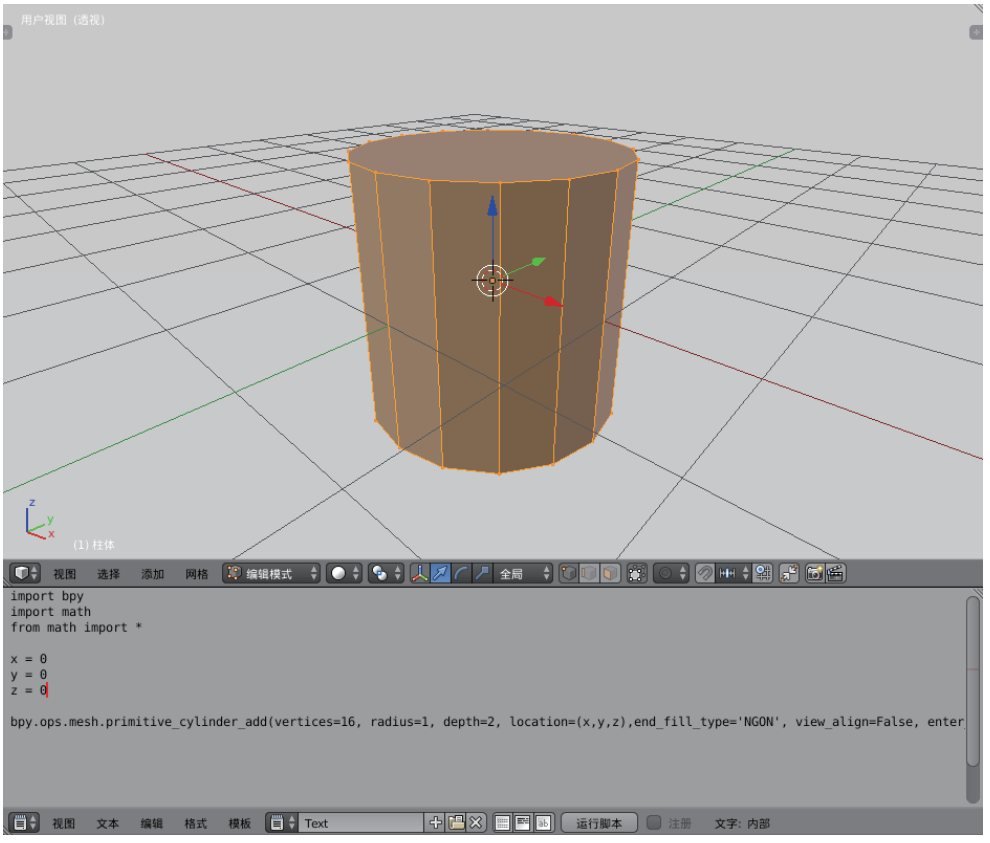

图 7-19　用 Python 脚本创建圆柱体编辑模式设计

7.10　Python 脚本创建一个立方体矩阵案例设计

利用 Python 脚本创建一个立方体矩阵，步骤如下：

- 启动 Blender 仿真游戏引擎，删除默认立方体。
- 在标题栏 3 中，选择“文本编辑器”→“新建”→“编写源代码 / 粘贴源代码”。

Python 脚本源程序代码如下。

```
    1 import bpy
    2 for x in range(20):
    3     for y in range(20):
    4         bpy.ops.mesh.primitive_cube_add(radius = .1 + (x*y*.0005),
location=(x, y, (x*y*.02)))
```

第 1 行代码：加载 Blender Python 模块，即导入 Blender 仿真引擎 Python 脚本。

第 2 行代码：设计循环变量 x，范围设定为 20。

第 3 行代码：设计循环变量 y，范围设定为 20。

第 4 行代码：设计循环体，创建几何网格立方体，参数赋值半径 = 1 +（x*y*.0005），位置 =（x，y，（x*y*.02））。

- 单击运行脚本按钮。在 3D 视图中，实现立方体矩阵造型设计，在下面文本编辑器中，显示 Python 脚本程序，如图 7-20 所示。

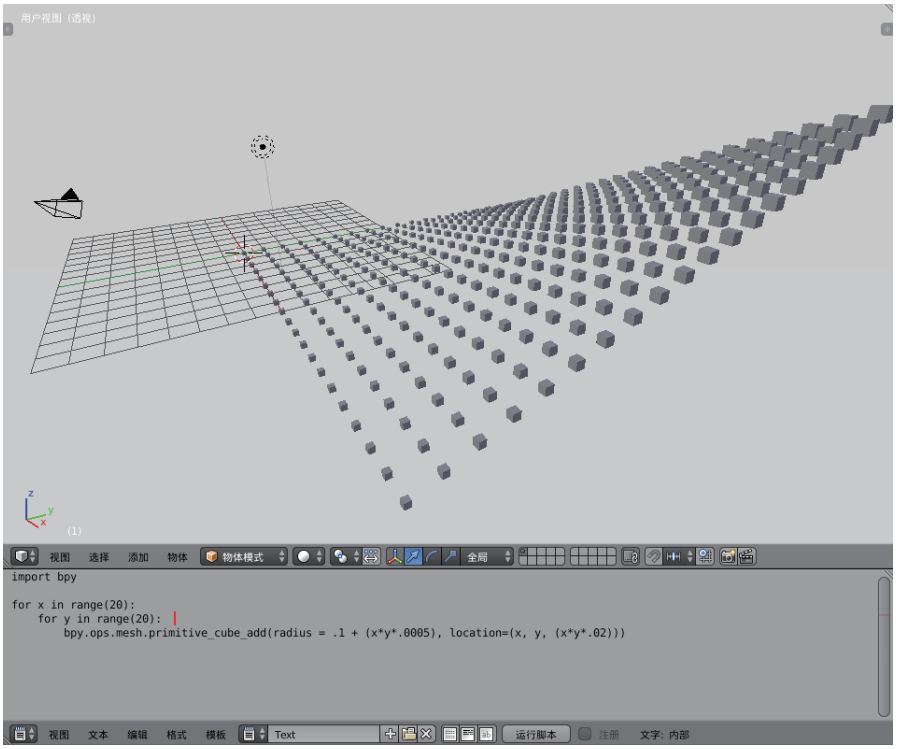

图 7-20　用 Python 脚本创建立方体矩阵设计

7.11　Python 脚本创建一个猴头和平面案例设计

利用 Python 脚本创建一个猴头和平面场景，步骤如下：

- 启动 Blender 仿真游戏引擎，删除默认立方体。
- 在 3D 视图右上角拖动一个视图窗口。
- 在第 2 个视图窗口中，选择“文本编辑器”→“新建”→“编写源代码 / 粘贴源代码”。

Python 脚本源程序代码如下。

```
1 import bpy
2 # 取消选择对象
3 bpy.ops.object.select_all(action='DESELECT')
```

```
4 ## 删除名称中包含多维数据集的所有对象 ##
5 for object in bpy.data.objects:
6     if  "Cube"  in object.name:
7         object.select = True
8         bpy.ops.object.delete()
9 ## 添加猴子对象
10 bpy.ops.mesh.primitive_monkey_add(location=(0, 0, 0), radius = 3)
11 ## 添加平面对象
12 bpy.ops.mesh.primitive_plane_add(location=(0, 0, -3), radius = 10)
```

第 1 行代码：加载 Blender 仿真引擎 Python 脚本。

第 2 行代码：注释行。

第 3 行代码：操作对象全选（action = ' 取消选择 '）。

第 4 行代码：注释行。

第 5 行代码：for 循环中的 bpy 脚本数据对象。

第 6 行代码：If 条件判断，如果“立方体”在对象名称中。

第 7 行代码：对象选择 = 真。

第 8 行代码：bpy 脚本操作对象被删除。

第 9 行代码：注释行。

第 10 行代码：添加猴子几何网格模型设计。

第 11 行代码：注释行。

第 12 行代码：添加平面几何网格模型设计。

- 单击运行脚本按钮。在第 1 个 3D 视图中，显示一个猴头和平面造型，在第 2 个视图窗口中，显示 Python 脚本程序，如图 7-21 所示。

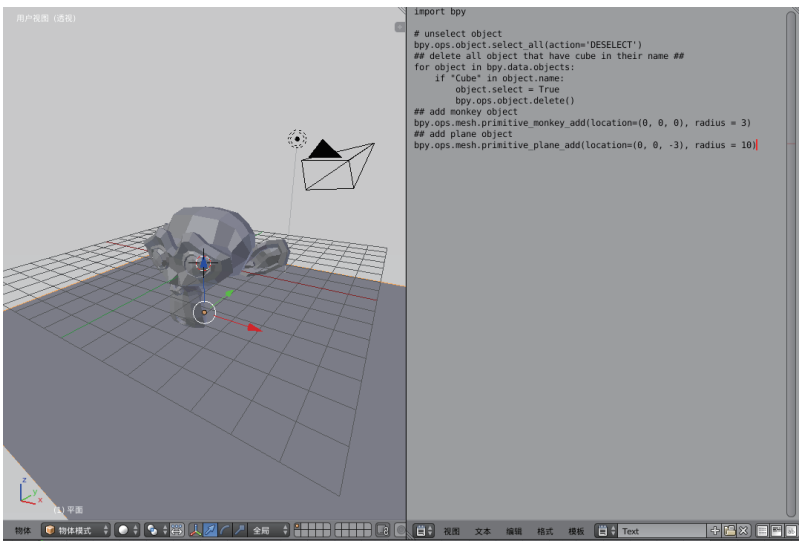

图 7-21　用 Python 脚本创建一个猴头和平面设计

7.12 Python 脚本创建一个多米诺骨牌案例设计

利用 Python 脚本创建一个多米诺骨牌动画，游戏场景包含一个平面、一组多米诺骨牌，步骤如下：

- 启动 Blender 仿真游戏引擎，删除默认立方体。
- 在 3D 视图窗口中，单击右上角小三角的斜线标志，向左拖动可以创建一个窗口。
- 在右侧第 2 个 3D 视图窗口的标题栏 2 中，选择“文本编辑”→“新建”。
- 在文本编辑栏中，输入 Python 脚本程序，或粘贴源代码即可，如图 7-22 所示。

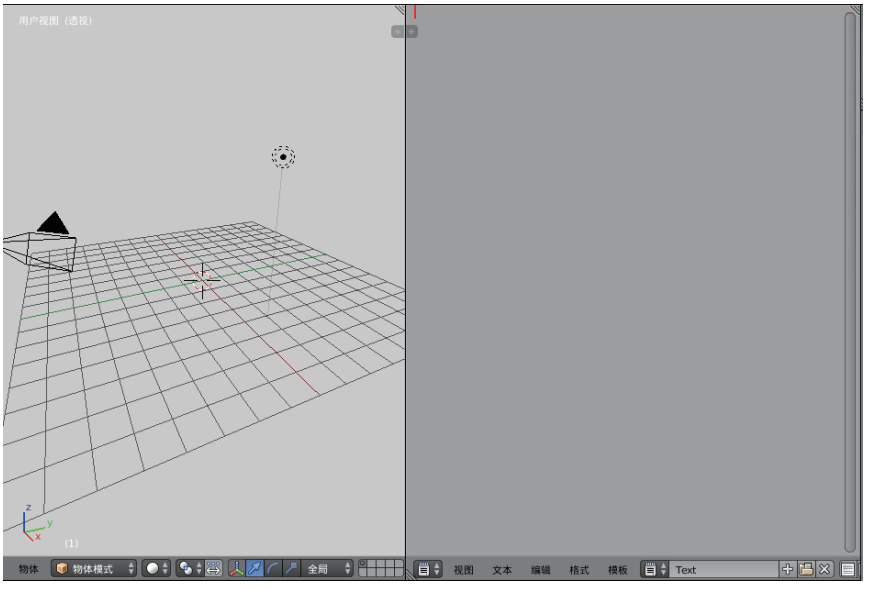

图 7-22　创建文本编辑

需要创建一个平面设置为“被动”的刚体，作为一个游戏地面。上面放一组均匀排放的长方体物体，并设置为刚体，移动初始立方体运动形成多米诺骨牌效应，实现多米诺骨牌动画设计效果。Python 脚本多米诺骨牌源程序代码如下。

```
1 import bpy
2 bpy.ops.mesh.primitive_plane_add(radius=145, location=(0, 0, 0))
3 bpy.ops.rigidbody.object_add()
4 bpy.context.object.rigid_body.type = 'PASSIVE'
5 for i in range(0, 25):
6     bpy.ops.mesh.primitive_cube_add(radius=1, location=(6*i, 0, 10))
7     bpy.ops.rigidbody.object_add()
8     bpy.ops.transform.resize(value=(1, 6, 10), constraint_axis=(False,
      True, True))
9     if i==0:
10        bpy.ops.transform.rotate(value=0.4, axis=(0, 1, 0))
```

第 1 行代码：加载 Blender Python 模块。

第 2 行代码：创建一个平面。

第 3 行代码：为平面添加一个刚体属性。

第 4 行代码：为平面添加一个刚体类型为被动属性。

第 5 ～ 7 行代码：创建一个由长方体构建的多米诺骨牌，并为其添加刚体。

第 8 行代码：调整多米诺骨牌块的大小值和约束轴等。

第 9 ～ 10 行代码：旋转第一块多米诺骨牌，使其翻转坠落状态并开始多米诺骨牌链式反应。

- 在左侧的第 1 个 3D 视图窗口中，按快捷键 Alt + A 或单击“播放”按钮，实现多米诺骨牌动画设计，如图 7-23 所示。

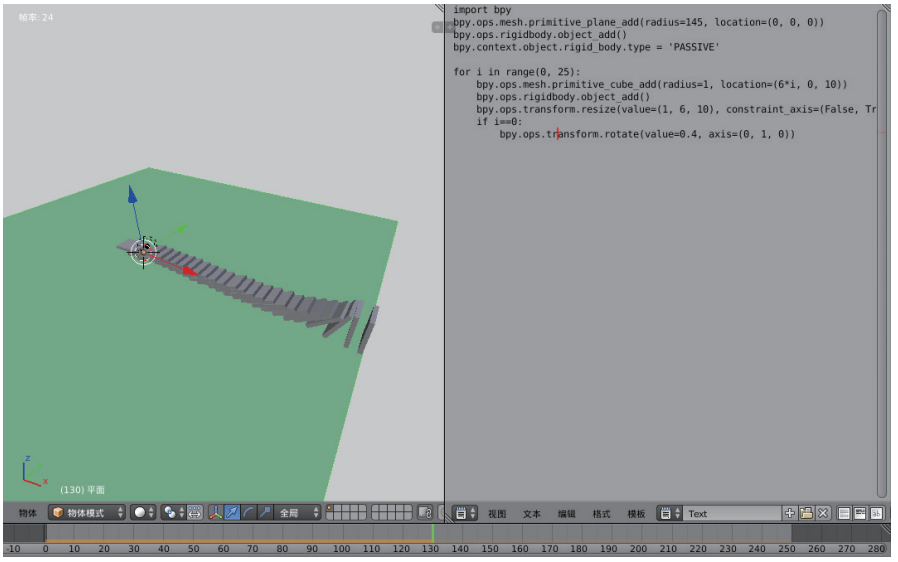

图 7-23　实现多米诺骨牌动画设计效果