DDLC Mod 开发指南

DDLC MOD Development Guide

（Ver. 0.1.0）

PartyParrot 著

前言

Ren'Py 是一种视觉小说语言。《*心跳！心跳！文学部！*》*(Doki Doki Literature Club!*，下文简称 DDLC) 正是基于 Ren'Py 编写的。因此，要编写 DDLC 的 Mod 就必须学习 Ren'Py。本书旨在通过简单明了的例子教会读者 DDLC Mod 的开发。另外，由于 Ren'Py 与 Python 是密不可分的，本书在介绍 Ren'Py 7 的同时也会介绍部分 Python 知识。本书中部分内容对于部分读者可能会有一定的编程门槛，请善用百度。

实例代码

本书包含大量的示例代码，由于 Ren'Py 的特殊性，大部分代码需要放在相应环境中才能运行。本书代码只适用于 Ren'Py 7/8，大部分代码理论上可以在 Ren'Py 6 上运行，但由于其过时性，本书将不再针对 Ren'Py 6 进行介绍与适配。

本书示例代码及注释样式

为区别普通文本，本书对于实例代码做出以下规定:

- 代码英文使用 Hack 字体, 中文使用思源等宽字体, 字号为 14 点。背景为 (235, 235, 235)。如:

```
# 这是一行注释
```

- 需要您输入的内容将以粗体出现。如:

```
$ renpy.input()
# 输入：22
```

- 表示代码输出结果的将以斜体出现。如:

```
>>> 1 + 2
# 输出：3
```

- 语法中的占位符将用尖括号括起来。您应使用实际的参数、变量等替换占位符。如:

```
define <变量名称> = <值：整型、浮点型等>
```

您应将其替换类似的例子:

```
define a = 2
```

- 当代码中不含有 »> 或... 则表示您应该使用文件运行代码，而非 Python 交互模式。
- 本书中只能在 Python 代码块中运行的语法，将会含有 **[Python Only]** 标签。如下例:
for 循环 **[Python Only]**

同时，本书分为 4 种注释类型:

- 普通注释背景使用 25% 色调青色，边框使用 75% 青色，如:

> **注释**
>
> 这是一行注释。

- 扩展知识背景使用 25% 色调绿色，边框使用 RGB 颜色（105, 190, 78）如:

> **扩展知识**
>
> 这是一行扩展知识。

- 警告背景使用 25% 色调黄色，边框使用 RGB 颜色（150, 150, 0）如:

> **警告**
>
> 这是一行警告。

- 必须注意的内容背景使用 25% 色调红色，边框使用 75% 红色，如:

> **必须注意**
>
> 您必须注意此内容。

获取最新版指南

https://wwyc.lanzouq.com/b02fb2saj
密码:ddlc
https://github.com/DanilJeston/DDLC-Chinese-Modding-Guide

联系方式

我们的联系邮箱是: team_ninety@outlook.com。

如果您对本书有任何疑问或建议, 请发邮件给我们。若您有兴趣参与本书的编写、完善, 可以邮件给我们。同时, 若您发现有人未经 CC BY-NC-SA 4.0 方式分发本书, 请发邮件给我们。若本书存在部分代码出现错误、无法运行等, 请发送邮件给我们。

最后, 祝您学习愉快!

目录

前言 i

第一章 预备知识 1

1.1 关于 Ren'Py . . . 1

1.1.1 Ren'Py 概述 . . . 1

1.2 下载 Ren'Py . . . 1

1.2.1 下载 Ren'Py SDK 7 . . . 2

第二章 开始学习 Ren'Py 4

2.1 准备开发环境 . . . 4

2.1.1 下载 DDLC Mod 中文模版 . . . 4

2.1.2 配置文本编辑器 . . . 5

2.1.3 设置您的游戏 . . . 7

2.2 进入 Ren'Py 世界 . . . 7

2.2.1 say 语句 . . . 7

2.2.2 show、hide、scene 语句 . . . 9

2.2.2.1 show 语句 . . . 9

2.2.2.2 hide 语句 . . . 13

2.2.2.3 scene 语句 . . . 13

2.2.3 play、stop、voice 语句 . . . 14

2.2.3.1 play 语句 . . . 14

2.2.3.2 stop 语句 . . . 15

2.2.3.3 voice 语句 . . . 15

2.2.4 menu 语句 . . . 16

第三章 Python 与 Ren'Py 18

3.1 Python 中的几种数据类型 . . . 19

3.1.1 数字 . . . 19

3.1.2 字符串 . . . 20

3.1.3 列表与元组 . . . 21

3.1.3.1 列表 . . . 21

3.1.3.2 元组 . . . 23

3.1.4 字典 . . . 23

3.1.5 布尔类型 . . . 24

3.2 函数与类 **[Python Only]** . . . 25
3.2.1 函数 . . . 25
3.2.1.1 函数的定义 . . . 25
3.2.1.2 函数命名空间 . . . 26
3.2.2 类 . . . 27
3.2.2.1 类的定义 . . . 27
3.2.2.2 类的使用 . . . 27
3.3 在 Ren'Py 中使用 Python 语句 . . . 28
3.3.1 Python 语句块 . . . 28
3.3.2 单行 Python 语句 . . . 29
3.3.3 init python 语句 . . . 29
3.4 使用变量 . . . 31
3.4.1 python 语句块 . . . 31
3.4.2 define 语句 . . . 31
3.4.3 default 语句 . . . 32
3.4.4 持久化数据 . . . 32
3.5 流程控制 . . . 32
3.5.1 脚本标签 . . . 32
3.5.1.1 label 语句 . . . 32
3.5.1.2 call 语句与 jump 语句 . . . 33
3.5.2 if 判断 . . . 35
3.5.3 循环 . . . 36
3.5.3.1 while 循环 . . . 37
3.5.3.2 for 循环 **[Python Only]** . . . 37
3.5.4 错误和异常 **[Python Only]** . . . 40
3.6 等效语句 . . . 42
3.6.1 对话 . . . 42
3.6.2 图像显示 . . . 43
3.6.2.1 show . . . 43
3.6.2.2 hide . . . 43
3.6.2.3 scene . . . 43
3.6.2.4 with 从句 . . . 43
3.6.3 call 和 jump . . . 43

第四章 增添资源 **44**
4.1 增添资源 . . . 44
4.1.1 定义角色 . . . 44
4.1.2 增加、定义图片 . . . 46
4.1.2.1 角色立绘 . . . 47
4.1.2.2 背景图像 . . . 48
4.1.3 音频资源 . . . 48
4.1.3.1 节选播放 . . . 48

第五章 特殊效果及特殊脚本 **50**
5.1 特殊脚本 50
5.1.1 bsod.rpy 50
5.1.2 cgs.rpy 50
5.1.3 console.rpy 51
5.1.4 glitchtext.rpy 51
5.1.5 poems_special.rpy\poems-tl.rpy\poems.rpy 51

后记：DDLC Cn 路在何方? **53**

第一章　预备知识

Ren'Py 是一种视觉小说语言, 名字是恋爱（れんあい，即恋爱）与 Python 两词混合而成。Python 是 Ren'Py 使用的编程语言。而《*心跳! 心跳! 文学部!*》(*Doki Doki Literature Club!*, 下文简称 DDLC) 正是基于 Ren'Py 编写的。想要编写 DDLC Mod, 我们就必须先学习 Ren'Py 相关知识。

本章将为您介绍 Ren'Py 的来源及如何安装 Ren'Py 与代码编辑器。

1.1　关于 Ren'Py

1.1.1　Ren'Py 概述

Ren'Py 几乎支持视觉小说所应该具有的功能, 如: 分支故事、存储与加载游戏、回退到之前故事的存储点、多样性的场景转换等。其首次发布于 2004 年 8 月 24 日。Ren'Py 拥有类似电影剧本的语法, 并且能够允许用户编写 Python 代码来增加新的功能。除此之外, 游戏引擎内附的出版工具能提供基本的脚本加密与压缩游戏素材。

Ren'Py 建构于 Python 软件库 Pygame 之上, 而它又基于了 SDL。Ren'Py 官方支持 Windows、Linux 以及较新版 Mac OS X, 并可通过 Arch Linux、Ubuntu、Debian 或 Gentoo 的软件包管理系统安装。它可以在 Windows、macOS、Linux、Android、OpenBSD、iOS 和 wasm 的 HTML5 下建置。

利用 Ren'Py 结合剧本及 Python, 我们可以制作出各种各样的游戏。Ren'Py 也有一些电子角色扮演游戏框架的示例, 但相对来说, 制作 RPG 游戏会比较困难。

1.2　下载 Ren'Py

目前主流的 Ren'Py 分为 3 个版本:Ren'Py 6、Ren'Py 7、Ren'Py 8。其中 Ren'Py 6 与 Ren'Py 7 兼容 Python 2 且 Ren'Py 7 已经支持大部分 Python 3 特性, 而 Ren'Py 8 完全兼容 Python 3。需要注意的是,Ren'Py 6 已停止支持。

Ren'Py 几乎可在所有主流系统上运行。由于 Ren'Py 8 目前稳定性存疑, 故本书主要使用 Ren'Py 7 进行教学。

> 注释
>
> Ren'Py 7.4.4 及以后的版本均不支持 Windows XP 及更早的系统。

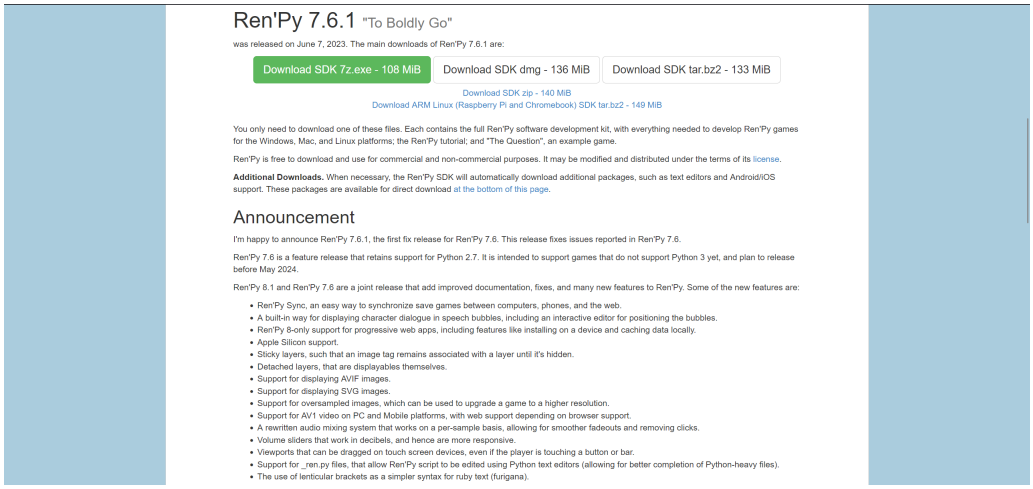

图 1.1: Ren'Py 官网

1.2.1 下载 Ren'Py SDK 7

> **注释**
>
> 本章编写时 DDLC Mod 中文模板支持的最新版 Ren'Py SDK 7 为 7.6.1。

使用任意浏览器进入 https://www.renpy.org/release/7.6.1 网页 (如图1.1), 您将在本页面看到三个按钮，分别是：Download SDK 7z.exe、Download SDK dmg 与 Download SDK tar.bz2。Windows 用户请下载第一个，macOS 用户请下载第二个，Linux 用户请下载第三个。

> **注释**
>
> 若无法打开文件，请点击第二行小字 Download SDK zip 下载 ZIP 文件

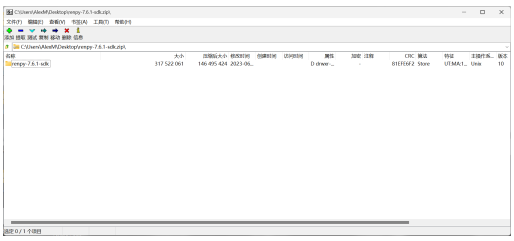

图 1.2: ZIP 文件

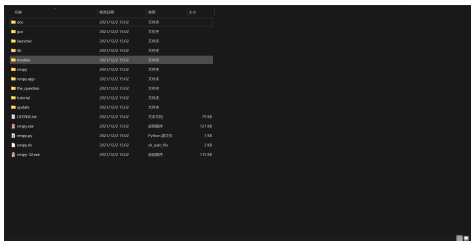

图 1.3: 文件夹内容

以 ZIP 文件为例，下载完成后打开压缩包（如图1.2）将文件夹中的所有内容解压, 得到如下内容。（如图1.3）

Windows 用户请双击 renpy.exe 或 renpy-32.exe(在某些情况下，它可能显示为 renpy 或 renpy-32)；MacOS 用户若使用驱动器镜像 (dmg) 安装 Ren'Py, 请运行 renpy, 否则运行 renpy.sh；Linux 用户请运行 renpy.sh。

打开 Ren'Py 后，正常来讲，您会见到如图1.4所示的界面。点击简体中文后，您会见到如图1.5所示的界面。

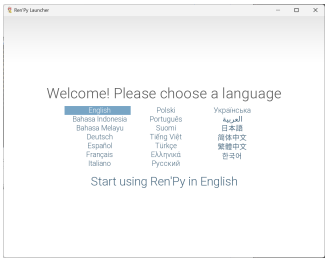

图 1.4: 选择语言

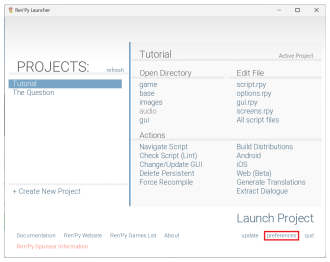

图 1.5: 调整语言

若您没有见到该界面, 而是直接见到类似图1.5，请点击右下角的 preferences，在右下角 Language 中选中简体中文即可调整为简体中文。

第二章　开始学习 Ren'Py

本节目标包括：

- 使用 DDLC Mod 中文模版创造一个项目；
- 学会使用 say 语句；
- 学会使用 show、hide、scene 语句；
- 学会使用 play、stop 语句；
- 学会使用 menu 语句；

要建造一个属于我们的房屋，就必须要先打地基、搭框架。如果说 Ren'Py 是项目的地基，那么 Mod 模板就是一个框架，我们可以在这个框架中快速地放置门、窗等，还可以自定义这个框架，而不必先调制水泥，然后从地基一步步开始。通过模版，我们可以省去很多步骤。本章我们将在 DDLC Mod 中文模版的基础上将对 Ren'Py 进行初步学习。

2.1　准备开发环境

DDLC Mod 模板是由 GanstaKingofSA 编写的一个方便开发 Mod 的模板。imgradeone 对其进行了本土化。目前，DDLC Mod 中文模板主要流行三个大版本:1.0 版本，2.0 版本 (即 Next 分支) 与 4.0 版本 (即 Future 分支)。

- 1.0 版本仅支持 Ren'Py 6，且没有什么特别功能；
- 2.0 版本增加支持了 Ren'Py 7、Android 移植等功能;
- 4.0 版本增加支持了 Ren'Py 8 与 Python 3，增加额外屏幕 (Extra Screen) 功能等，但稳定性存疑。

本书将使用 2.0 版本进行讲解。

2.1.1　下载 DDLC Mod 中文模版

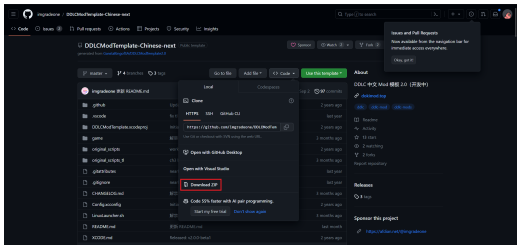

图 2.1: 项目页面

使用任意浏览器打开 github.com/imgradeone/DDLCModTemplate-Chinese-next，将压缩包下载下来 (如图2.1所示)。

下载完成后解压至在第1.2.1节中您下载的 Ren'Py SDK 的目录下。完成该步骤后，您的 Ren'Py SDK 中项目一栏应出现 DDLCModTemplate-Chinese-next。

从 https://ddlc.moe/中下载原版 DDLC,打开压缩包后将 DDLC-1.1.1-pc/game/下的 audio.rpa、images.rpa、fonts.rpa 解压至 Mod 中文模板下的 game/文件夹中。此时，Mod 中文模板下的 game/文件夹结构应如图2.2所示。

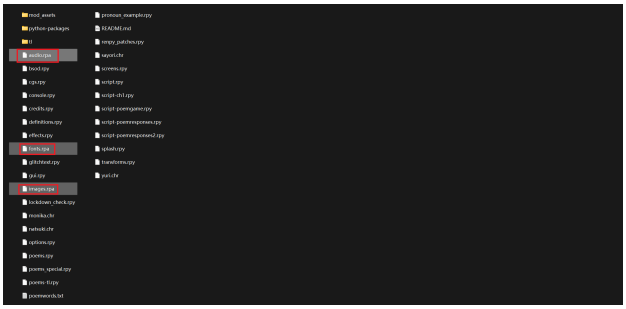

图 2.2: 项目结构

至此，您就完成了准备工作的第一部分——下载 DDLC Mod 中文模板并完成配置。

2.1.2　配置文本编辑器

有了 Mod 中文模板，我们还需要一个趁手的编辑器。你大可选择记事本，不过 Ren'Py SDK 也为我们提供了两种文本编辑器:Visual Studio Code 和 Atom。本书将以 Visual Studio Code 为例配置文本编辑器。

打开 Ren'Py SDK，点击右下角的设置。在一般选项卡中选择文本编辑器，此时点击第一个选项 Visual Studio Code。Ren'Py 会开始下载 Visual Studio Code 与 Ren'Py 插件。耐心等待一段时间后，会返回到设置界面。此时点击返回，点击 DDLCModTemplate-Chinese-next，点击编辑选项卡下的“打开项目”会打开 Visual Studio Code(如图2.3所示)。

此时点击扩展选项卡，输入 Chinese，点击第一个搜索结果中的 Install。此时右下角会弹出提示框询问是否切换语言并重启。点击 Change Language and Restart。重启后 Visual Studio Code 就会切换成中文。

打开 Ren'Py SDK，选中 DDLCModTemplate-Chinese-next，点击编辑文件选项卡中的“打开项目”，Ren'Py SDK 会自动帮我们打开 Visual Studio Code 并定位到本项目。

随后，展开左侧资源管理器中的“game”文件夹，打开 script-ch1.rpy 文件。保留文件第一行，删除其他内容即可。至此，我们就正式完成了开发的准备工作。

> **注释**
>
> 若您打开 Visual Studio Code 时出现如图2.4所示的提示框，请点击“是，我信任此作者”。

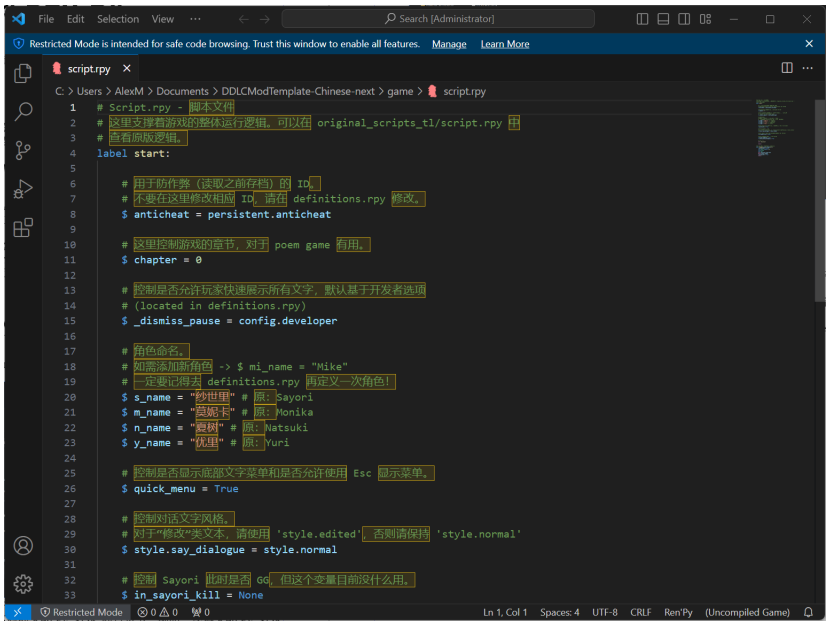

图 2.3: Visual Studio Code

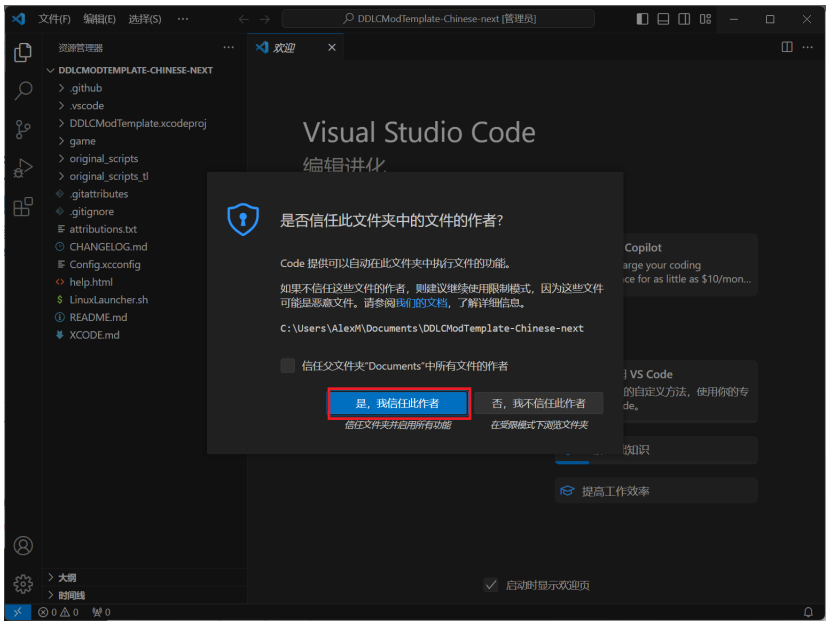

图 2.4: Visual Studio Code

2.1.3　设置您的游戏

做好了上述准备工作，我们还有最后一项任务需要完成———配置你的游戏设置。游戏设置关系到游戏的正常运行、存读档。现在，在 Visual Studio Code 左侧资源管理器中 game 目录下的 options.rpy 文件

现在，将根据下表，将原有内容替换或进行配置。

2.2　进入 Ren'Py 世界

在 Ren'Py 的世界里，剧本与视觉内容都围绕着代码展开。要编写脚本，我们就必须先学习 Ren'Py 的语法。

2.2.1　say 语句

say 语句是一个十分重要的语法，它可以让视觉小说角色拥有对话的能力。不过不用担心，say 语句的语法十分简单。

还记得上一课我们打开的 script-ch1.rpy 吗？现在另起一行，按下 Tab 键 (或打四个空格，不过 Visual Studio Code 会自动为我们将 Tab 转化为空格)，输入以下代码片段:

```
1    "快到学园祭了"
2    m "各位！我们得开始准备了！"
```

警告

您必须使用英文引号来包住文字。另外，请务必注意缩进问题，错误的缩进将会导致代码运行失败。

这是一段非常简单的 Ren'Py 代码，它只包含文字，不显示任何图像或播放任何音效。现在，切换回 Ren'Py SDK，点击启动项目，开始游戏，您应该会看到一行旁白以及莫妮卡的一行对话。

在理解以上基础内容之后，我们可以添加更多对话。不过在那之前，你得先了解 say 语句的基础语法:

```
<角色名> "<说话内容>"
```

此处的角色名应对应下列表格:

角色名	对应角色
m	莫妮卡（Monika）
s	纱世里（Sayori）
y	优里（Yuri）
n	夏树（Natsuki）
mc	主人公（Main Character）
留空	叙述者（Narrator）

表 2.1: 角色名表格

现在，尝试理解下列代码并运行:

```
# Chapter 0
label ch0_start:
    "快到学园祭了。"
    m "各位！我们得开始准备了！ "
    s "好耶！！！！ "
    n "啊，我都等不及学园祭了。"
    n "肯定会很棒的！ "
    y "..."
    return
```

代码 2.1: scipt-ch1.rpy

在 Ren'Py 中，我们使用脚本标签（label）来声明代码块。（关于脚本标签的知识，我们将会在后续内容进行详细讲解）。通常，在 Ren'Py 项目中都含有一个名为 start 的脚本标签。但是在 DDLC Mod 模板中，script.rpy 文件已经为我们声明好了 start 脚本标签。在本段代码中，script-ch1.rpy 包含以下几件事:

- 第一行的注释;
- 创造一个名为 ch0_start 的代码块;
- say 语句创造对话;
- return 语句返回到上级。

注释

在 Ren'Py 中，我们使用与 Python 相同的“#”来表示本行为注释。注释行的内容将不会被 Ren'Py 或 Python 执行。

当我们需要在对话中提及玩家的名字、需要将某些内容突出显示或对某部分文字需要进行特殊处理时，应当怎么办呢？读代码2.2，尝试理解其用法，并在游戏中运行，看看与自己的猜测是否相同。

```
# Chapter 0
label ch0_start:
    "{cps=20} 快到 {b} 学园祭 {/b} 了。{/cps}{w=.5}{nw}"
    m "各位！我们得开始准备了！"
    s "好耶！！！！"
    n "啊，我都等不及学园祭了。"
    n "肯定会很棒的！"
    y "..."
    m "那么，是时候来进行分工了。"
    m "[player]，你想要做什么？"
    return
```

代码 2.2: scipt-ch1.rpy

扩展知识

player 是一个记录了玩家名字的变量。不理解变量是什么？没关系，本书将在后面为您介绍变量的性质与作用。在这里，您只需要知道当需要提及玩家名字时使用 player 变量即可。
您或许也注意到了 {/cps} 和 {/b}。这是对前面 cps 标签和 bold 标签的闭合，代表着其适用范围仅为自身标签与闭合标签内的所有文字。

不难发现，当需要使用变量时，我们只需要使用中括号将变量名括住。当我们需要将某段文字进行加粗等操作，可以对文字打上标签 (既花括号)。若需要在对话中使用这些字符，则需要连续出现两次。如在代码中要出现花括号，则应该使用 {{}} 代替 {}

以下为常用的文字标签：

标签	作用
{b}	将文本渲染为粗体
{i}	将文本渲染为斜体
{color=<16 进制颜色 >}	将文本渲染为指定颜色
{font=< 字体文件 >}	使用指定字体渲染文本
{cps=< 数字 >}	以指定速度渲染文本
{nw}	不等待用户操作，渲染完本行文字后立即渲染下一行
{w=< 数字 >}	等待一定秒数后或用户操作后继续渲染文本

表 2.2: 常用文本处理标签

2.2.2　show、hide、scene 语句

在2.2.1课中，我们学习了如何在视觉小说中编写对话。但现在有一个很大的问题——没有图像。在视觉小说中，视觉便是重点。在本课中，我们将学习如何在视角小说中显示图像。

2.2.2.1　show 语句

show 语句是 Ren'Py 游戏中的一个重要语句。它可以让一个图像显示在屏幕上。这个图像可以是背景，也可以是角色。

还记得上一课中的代码吗？请阅读以下代码并尝试理解，在 Ren'Py 中运行，看看运行实际效果与自己的理解是否正确。

```
1  # Chapter 0
2  label ch0_start:
3      show bg club_day
4      "{cps=20}快到{b}学园祭{/b}了。{/cps}{w=.5}{nw}"
5      show monika 1a at t41 zorder 1
6      m "各位！我们得开始准备了！"
7      show sayori 1a at t42 zorder 1
8      s "好耶！！！！"
9      show natsuki 1a at t43 zorder 1
10     n "啊，我都等不及学园祭了。"
```

```
    n "肯定会很棒的！"
    show yuri 1a at t44 zorder 1
    y "..."
    m "那么，是时候来进行分工了。"
    m "[player]，你想要做什么？"
    return
```

代码 2.3: script-ch1.rpy

由此可知，show 语句的基础语法为：

```
show <图片名称> <附加参数>
```

在游戏中，图片名称可以大致分为三类：

- 角色立绘
- 背景立绘
- 毛刺 (Glitch) 立绘

角色立绘 对于角色立绘，图片名称的组成为：

```
<角色名> <立绘编号>
```

DDLC 原版中，一个显示在屏幕上的角色立绘可以分为两个部分：身体姿势与面部表情。（实际上，DDLC 中的立绘为三个部分：左侧身体姿势、右侧身体姿势、面部表情）

模版中已经将所有可能的身体姿势为我们组合好了。除优里外的所有角色都有五个姿势（1-4 为正常站位，5 为特殊站位），而优里只有 4 个姿势（1-3 为正常站位，4 为特殊站位）。

面部表情使用英文小写字母来表示（请注意，不是所有的角色的面部表情都表示到 z。如莫妮卡的面部表情就远少于夏树的面部表情）。

同时，除莫妮卡外的所有角色都可以使用日常服，只需要在身体姿势后加上 b 即可使用日常服。如：1b

对于角色立绘的列举，您可以前往 https://docs.dokimod.top/pages/0a59cf/ 查看。

毛刺立绘 毛刺立绘则是指在二周目出现的大多数“错误”立绘，如错乱的莫妮卡等。对于毛刺立绘，其图片名称的构成为：

```
<角色名> <毛刺编号>
```

警告

我们不建议您使用毛刺立绘，因为这对于某些厨可能极不友好。

一般来说，毛刺编号为 glitch。但是请注意，莫妮卡的毛刺立绘编号为 g1 与 g2；夏树的毛刺立绘则包括 ghost_blood、ghost1、ghost2 等。

若要使用优里的自杀立绘，则毛刺编号为 stab_+ 数字 1-6。

背景立绘　对于背景立绘，其图片名称构成为：

```
bg <背景编号>
```

背景编号即意义可以在 definitions.rpy 文件中找到。这里不再进行过多叙述。

> 扩展知识
>
> 通常我们不使用 show 来显示背景，而是使用 scene。对于 scene 的详细介绍请看第2.2.2.3课。

附加参数　对于附加参数，常用的有：

语法	作用
at <transform>	将一个变换 (transform) 套用到立绘上
with <transform>	显示图像时使用指定的转场动画 (transform)
zordr < 数字 >	控制图像在 Z 轴上的位置

表 2.3: 常用 show 语句附加参数

> 扩展知识
>
> show 语句也可以使用 behind 从句、as 从句、onlayer 从句。但由于在游戏中这些从句很少会使用，故不再展开讲解。有兴趣者可前往 https://doc.renpy.cn/zh-CN/displaying_images.html 了解详细内容。

at、with 从句　当变换作为角色变化时，应使用 at 从句。一个关于变换的例子：t41。在这个变换中，t 表示在这个位置上的角色静止站立在原地；4 表示一共有 4 个站位，会将屏幕等分成 4 份；1 表示从左往右该变换是所有站位中的第 1 个。

所有角色状态有：

编号	意义
t	角色静止站立在原地
i（instant）	角色突然出现
f（focus）	角色成为屏幕焦点
s（sink）	角色下沉
h（hop）	角色跳跃
hf（hop and focus）	角色跳跃并成为屏幕焦点
d（dip）	角色向下倾斜然后升起
l（left）	角色从左侧飞入
r（right）	角色从右处飞入

表 2.4: 角色状态表

对于站位总数，共有 1 到 4。

当变换作为转场动画来使用时，应使用 with 从句。游戏中实现定义好的转场动画有：

编号	意义
dissolve	溶解效果
dissolve_cg	适用于 CG 的溶解效果
dissolve_scene	适用于 scene 的溶解效果
dissolve_scene_full	使屏幕自行溶解为黑色，显示另一个场景
dissolve_scene_half	溶解屏幕一段时间，显示下一个场景
wipeleft	从屏幕左侧擦除隐藏当前图像
wipeleft_scene	从屏幕左侧擦除屏幕为黑色，然后显示下一场景
wiperight	从屏幕右侧擦除隐藏当前图像
wiperight_scene	从屏幕右侧擦除屏幕为黑色，然后显示下一场景

表 2.5: 常用转场动画编号及其效果

zorder 从句 对于 zorder 从句，zorder 后的数字越大，图像在 Z 轴上的距离越远，即离屏幕越远，反之亦然。但由于 DDLC 的变换已经为我们设置好了图像的大小，所以 zorder 其实在后续版本不再需要。

临时性变化与对话属性 在视觉小说中，每一次对话角色的神态、姿势可能都会发生变化，那我们岂不是要重复使用很多次 show 语句、hide 语句? 所以为了方便开发，Ren'Py 给 say 语句添加了对话属性。

如下例:

```
# Chapter 0
label ch0_start:
    show bg club_day
    "{cps=20}快到{b}学园祭{/b}了。{/cps}{w=.5}{nw}"
    show monika 1a at t41 zorder 1
    m "各位！我们得开始准备了！"
    show sayori 1a at t42 zorder 1
    s "好耶！！！！"
    show natsuki 1a at t43 zorder 1
    n 2d "啊，我都等不及学园祭了。"
    n "肯定会很棒的！"
    show yuri 1a at t44 zorder 1
    y "..."
    m "那么，是时候来进行分工了。"
    m 4k "[player]，你想要做什么？"
    return
```

运行后，我们发现，当夏树说出“啊，我都等不及学园祭了。”会变换一次立绘，以及后面莫妮卡说出“[player]，你想要做什么?”也会变换立绘，这就是 say 语句的对话属性作用。

不难发现，只需要在角色名后添加立绘编号即可使角色在说出这句话时变换为指定立绘。

临时性变化则是指角色在说出本句前会变换为指定立绘，在本句结束后则换回原来的立绘。要启用临时性变化，只需要在角色名后的立绘编号前加上 @ 即可。如：

```
    n @2d "啊，我都等不及学园祭了。"
```

2.2.2.2　hide 语句

hide 语句有着与 show 语句相反的作用——从屏幕上隐藏一个图像。如：

```
hide monika
```

则会将莫妮卡从当前屏幕上隐藏。

hide 语句同样可以使用附加参数，但只能使用 with 从句，使用方法与 show 语句的 with 从句相同，详细请见第2.2.2.1课。

扩展知识

扩展知识：hide 语句同样可以使用 onlayer 从句，用于隐藏对应图层上的图像。但在游戏中很少会涉及到对图层的操作，故本书不会对 onlayer 从句展开讲解。有兴趣者课前往 https://doc.renpy.cn/zh-CN/displaying_images.html 了解详细内容。

2.2.2.3　scene 语句

scene 语句类似于 show 语句，但与 show 语句有一个很明显的差异——使用 scene 语句会清空当前屏幕上的所有图像。scene 语句的显示方式和特性的使用效果与 show 语句一致。

尝试在游戏中运行以下代码，并猜测运行结果：

```
1  # Chapter 0
2  label ch0_start:
3      scene bg club_day
4      "{cps=20}快到{b}学园祭{/b}了。{/cps}{w=.5}{nw}"
5      show monika 1a at l41 zorder 1
6      m "各位！我们得开始准备了！"
7      show sayori 1a at h42 zorder 1
8      s "好耶！！！！"
9      show natsuki 1a at t43 zorder 1
10     n 2d "啊，我都等不及学园祭了。"
11     n "肯定会很棒的！"
12     show yuri 1a at s44 zorder 1
13     y "..."
14     scene bg club_day
15     show monika 2a at t11 zorder 1
16     m "那么，是时候来进行分工了。"
17     m 4k "[player],你想要做什么？"
18     return
```

代码 2.4: script-ch1.rpy

不难发现，执行 scene 语句时会将整个屏幕清空。scene 语句的语法为:

```
scene <图片编号> <with从句>
```

with 从句的使用方法与 hide、show 语句相同。详细请见2.2.2.1。

2.2.3 play、stop、voice 语句

现在，在我们的视觉小说中，图像有了，人物立绘与背景也有了，但还是缺了一些东西，比如音乐与音效。在接下来，我们将会进入对于 play 和 stop 语句的学习。

2.2.3.1 play 语句

play 语句主要用于播放音频、音效。请注意，play 语句会覆盖当前通道所播放的音频。Ren'Py 中，我们主要使用三种已经定义好的音频频道:

- music - 音乐播放通道;
- sound - 声音播放通道;
- voice - 语音播放通道。

请阅读以下代码并尝试理解 play 语句的语法:

```
1    play music audio.t1
2    play sound audio.closet_open
```

不难看出，play 语句的基本语法如下:

```
play <频道名> <文件名>
```

> 扩展知识
>
> 在上述例子中，audio.t1 是一个变量，且被定义为"<loop 22.073>bgm/1.ogg"。故此处的文件名也可以是一个指向文件名的变量。

播放列表 play 语句的文件名部分也可以是一个列表，且遵循从首到尾的顺序。

读下列代码，尝试理解并运行:

```
1    play music [audio.t1, audio.t2]
2    play sound [audio.closet_open, "<silence .5>", audio.
         closet_open]
```

特殊效果 play 也可以使用淡出、淡入、循环等效果。

读下列代码，尝试理解并运行。

```
1    play music audio.main_menu loop
2    play music audio.t1 fadein .5 fadeout 1 noloop
```

从实际效果可以看出，loop 可以使一段音频重复播放；fadein 则可以使音频在指定时间内淡入；fadeout 则使音频在指定时间内淡出；noloop 意味不重复播放这段音频。

扩展知识

当 play 语句既没有出现 loop 分句也没有出现 noloop 分句时，Ren'Py 会根据音频的默认配置决定音频的播放。

play 语句同样也可以调整音频的音量。

```
play music audio.t1 volume .55 loop
```

请注意，volume 分句后的值应大于等于 0 小于等于 1

2.2.3.2　stop 语句

stop 语句以关键词 stop 开头，后接想要停止播放的通道名。如：

```
stop music fadeout 2
stop sound
```

2.2.3.3　voice 语句

Ren'Py 同样支持语音功能。要使用语音功能可以使用 voice 语句。
读下列代码，尝试理解 voice 语句的用法：

```
voice 'hello.ogg'
m "你好！"

define voice.a = "happy_birthday.ogg"
voice voice.a
m "生日快乐！"
```

请注意，当用户进行互动行为时，语音将会被打断。使用 sustain 特性则可以保证语音完整播放不被打断。

```
voice 'arguing.ogg'
s "你不能这样做！！"

voice sustain
s "...小蛋糕是我的！！！！"
```

自动语音　Ren'Py 同时还提供了自动语音功能，省去了麻烦的 voice 语句。
要实现这个功能，语音文件名必须跟对话脚本标识号严格匹配。
对话脚本标示名，需要将对话脚本导出为一个表格。操作如下：

1. 在启动器上选择“Extract Dialogue”
2. “Tab-delimited Spreadsheet (dialogue.tab)”
3. “Continue”
4. 使用表格程序打开 dialogue.tab。

表格的第一列就是需要使用的标识号，其他列则是对话的更多别的信息。

要自动播放语音，请将 config.auto_voice 设置为一个包含 {id} 的字符串。当开始对话时，{id} 会被对话脚本标识符替换，并自动组成一个音频文件名。若音频文件名对应的文件真实存在，则那个文件就会播放。

```
    config.auto_voice = "mod_assets/voice/{id}.ogg"
```

对话标识号是ch1_n_end_24e41fcf,那么当对应的对话显示时,Ren'Py 会寻找文件 mod_assets/voice/ch1_n_end_
如果文件存在，Ren'Py 会播放这个文件。

2.2.4 menu 语句

在视觉小说中，往往会有很多选择。这些选择通常能够影响剧情的走向，进入不同的分支。在本节中，你将会学习 menu 语句的基本用法。

读下列代码并尝试运行，看看实际的运行效果：

```
# Chapter 0
label ch0_start:
    scene bg club_day
    "{cps=20}快到{b}学园祭{/b}了。{/cps}{w=.5}{nw}"
    show monika 1a at l41 zorder 1
    m "各位！我们得开始准备了！"
    show sayori 1a at h42 zorder 1
    s "好耶！！！！"
    show natsuki 1a at t43 zorder 1
    n 2d "啊，我都等不及学园祭了。"
    n "肯定会很棒的！"
    show yuri 1a at s44 zorder 1
    y "..."
    scene bg club_day
    show monika 2a at t21 zorder 1
    show sayori 2a at t22 zorder 1
    m "那么，是时候来进行分工了。"
    m 4k "[player],你想要做什么？"
    menu:
        "做小蛋糕":
            s "夏树的小蛋糕最好吃了！"
        "布置教室":
            m "那我们可得抓紧时间了！"
    return
```

代码 2.5: script-ch1.rpy

menu 的基本语法为：

```
    menu:
        "这是一个选项示例" # 文本框内显示的内容。
```

```
        "选项 1": # 选项名称
            # 选择此项后所的脚本。
        # 可以添加更多选项。
```

第三章　Python 与 Ren'Py

本节目标包括：

- 了解 Python 的基本数据类型；
- 学会使用函数与类；
- 学会在 Ren'Py 代码中嵌入 Python 代码；
- 学会使用变量；
- 学会进行判断控制；
- 学会使用等效语句；

Ren'Py 与 Python 是一个不可分割的整体，在 Ren'Py 中，许多复杂的操作都需要依靠 Python 来完成。本章我们将会初步学习如何在 Ren'Py 中使用 Python 语句。

必须注意

本章只会浅显的介绍 Python 的用法，若要真正的学习 Python，您可以前往 https://www.runoob.com/python3/python3-tutorial.html 进行更深入的学习。切记，本书不以 Python 为主。

警告

如果您只是想要简单开发一个 Mod，不需要做什么复杂的处理，如只是扩展一下原作剧情，或是给玩家讲述另一个故事，那么您大概率可以跳过本章关于函数、类的学习。但如果您需要开发更复杂的 Mod，比如与 DDLC 有关的 AVG 游戏，或是像 Monika After Story 一样的 Mod，那么函数与类无疑会方便您后期的开发。

但是在学习前，您需要注意一些问题。学习 Python 的难度也比 Ren'Py 要大许多。而且在 Ren'Py 中使用 Python 代码还要格外小心。如在 Python 中，有几种数据类型需要格外注意。这些数据类型轻则导致代码出现意料之外的运行结果，重则使 Ren'Py 不稳定崩溃。

最后，虽然很多 Ren'Py 语句都有等效的 Python 代码，但对于 Mod 来说，不应该使用 Python 代替 Ren'Py 代码。Python 的作用是方便开发者进行更复杂的操作而不用修改 Ren'Py 的底层逻辑。但对于大部分的 Mod 来说，完全可以使用纯 Ren'Py 代码。更不用说 Python 代码在一定程度上会增加脚本的复杂度。Python 学好了，用好了，就是锦上添花；如果用不好，就会造成开发难度直线上升，Debug 及其困难，还会使游戏崩溃给玩家带来负面体验。

3.1 Python 中的几种数据类型

3.1.1 数字

在大多数编程语言中，数字可以分为两类——整型（int）与浮点类（float）型。整型顾名思义，就是指 3、2、6、-2 等不包含小数的数字，浮点型则与之相反，即包含小数的数字，如-2.9、7.1、3.213 等。

数字可以进行运算，如加减乘除。在 Python 中，加法运算使用 + 号，减法运算使用–号，乘法运算使用 * 号，除法运算使用/号。对于其他的运算符，详细请见表3.1。如下例：

```
>>> 1 + 2
# 输出：3

>>> 2 / 2
# 输出：1.0

>>> 10 - 8
# 输出：2

>>> 2 * 7
# 输出：14
```

扩展知识

我们可以使用变量将想要将数据保存。使用 = 操作符对变量进行赋值，如：

```
>>> a = 1
>>> a
# 输出：1

>>> a = a * 2
>>> a
# 输出：2
```

对于乘方、整除、取模（余数）计算，则分别使用 ** 运算符、//运算符与% 运算符。如下例：

```
>>> 1 ** 2
# 输出：2

>>> 2 ** 3
# 输出：8

>>> 10 // 8
# 输出：1
```

```
>>> 14 // 7
# 输出: 2

>>> 14 % 7
# 输出: 0

>>> 25 % 4
# 输出: 1
```

3.1.2 字符串

除了数字，Python 还可以处理字符串（str）。在 Python 中可以使用双引号会单引号括起来表示字符串，也可以使用反斜线操作符对特殊字符转义。

```
>>> 'Hello'
# 输出: 'Hello'

>>> "Hi!"
# 输出: "Hi!"

>>> 'I\'m fine.'
# 输出: "I'm fine."

>>> "I'm fine."
# 输出: "I'm fine."

>>> print("This is a backslash: \\")
# 输出: This is a backslash: \
```

在 Python 中，字符串同样支持一些运算功能。+ 号能将两个字符串连接起来。* 则用来重复字符串。如：

```
>>> 'Hello! ' * 3
# 输出: 'Hello! Hello! Hello!'

>>> "Hi! " + "How are you?"
# 输出: "Hi! How are you?"
```

操作符	描述	示例
+	加法运算，将运算符两侧值相加	1 + 2; "str" + "string"
-	减法运算，将运算符两侧值相减	1 - 2
*	乘法运算，将操作符两侧值相称	4 * 2； "str" * 3
/	除法运算，左操作数除以右操作数	3 / 1

操作符	描述	示例
%	取模运算，左操作数除以右操作数的余数部分	7 % 2
**	幂运算，左操作数的右操作数次幂	2 ** 3
//	整除运算，左操作数整除以右操作数	3 // 2
+=	将左侧值加上右侧值	a += 100
-=	将左侧值减去右侧值	a -= 100
*=	将左侧值乘上右侧值	a *= 100
/=	将左侧值除以右侧值	a /= 100

表 3.1: 常见算数与赋值运算符与示意

3.1.3 列表与元组

3.1.3.1 列表

在 Python 中，列表（list）是一种常见的数据类型。在列表可以将多种数据组合在一起。如：

```
>>> ['1', '2', '3']
# 输出: ['1', '2', '3']

>>> ['This', 'is', 'a', 'list']
# 输出: ['This', 'is', 'a', 'list']
```

在列表中，我们可以使用索引（index）来访问指定数据。如:

```
>>> name_list = ['Sayori', '莫妮卡', 'Yuri', 'Natsuki']
>>> name_list[0]
# 输出: 'Sayori'

>>> name_list[3]
# 输出: 'Natsuki'
```

警告

从上例中，我们可以知道索引从 0 开始。但请注意：索引最大值不可以超出列表的长度。即在 name_list 这个列表中，索引最大只能为 3，因为此时从 0 往后数 4 个数为 3。如果索引最大值超出了列表长度，Python 就会抛出 IndexError 错误。如：

```
>>> name_list = ['Sayori', 'Monika']
>>> name_list[2]

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

扩展知识

索引的值也可以为负数，此时 Python 会从列表尾部向前寻找索引。但请注意，若要使 Python 从尾部开始寻找，则索引从-1 开始，且同样不可超出列表最大长度。如下例：

```
>>> name_list = ['Sayori', 'Monika', 'Natsuki', 'Yuri']
>>> name_list[-1]
# 输出: 'Yuri'

>>> name_list[-4]
# 输出: 'Sayori'

>>> name_list[-10]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

扩展知识

索引也可用于字符串，但无法修改字符串的数据。字符串中的索引用法与列表中的索引用法相同。

同时，我们可对列表中的数据进行修改。要修改数据，只需要使用索引指定修改的数据，然后使用 = 重新赋值。如：

```
>>> name_list = ['Sayori', '莫妮卡', 'Natsuki', 'Yuri']
>>> name_list[1] = 'Monika'
>>> name_list

# 输出: ['Sayori', 'Monika', 'Natsuki', 'Yuri']
```

若要添加数据，则可以使用 insert 函数与 append 函数。如

```
>>> name_list = ['Monika', 'Sayori']
>>> name_list.insert(1, 'Natsuki')
>>> name_list.append('Yuri')
>>> name_list

# 输出: ['Monika', 'Natsuki', 'Sayori', 'Yuri']
```

对于 insert 函数，它接受两个参数。第一个参数为一个整数，代表在列表的指定索引处添加一个数据。第二个参数则是添加的数据。

对于 append 函数，它接受一个参数，即要添加的内容。append 函数会在列表末尾添加数据。

若要删除数据，则可以使用 pop 函数与 remove 函数。pop 函数与 remove 函数都只接受一个参数。pop 函数接受一个整数参数，可以删除列表中指定索引处的数据。remove 函数接

受一个任意类型的数据，它会先检查列表中是否存在一个数据与参数相同，如果存在则移除，如果不存在则抛出 ValueError 错误。如：

```
>>> name_list = ['Monika', 'Sayori', 'Yuri', 'Natsuki']
>>> name_list.pop(0)
>>> name_list
# 输出: ['Sayori', 'Yuri', 'Natsuki']

>>> name_list.remove("Sayori")
# 输出: ['Yuri', 'Natsuki']

>>> name_list.remove("Monika")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: list.remove(x): x not in list
```

3.1.3.2 元组

元组（tuple）是一种不可变的数据类型。元组支持列表除修改、添加、删除外的所有功能。如：

```
>>> name_list = ('Sayori', 'Monika', 'Natsuki', 'Yuri')
>>> name_list[3]
# 输出: 'Yuri'

>>> name_list.append("Main Character")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'append'
```

3.1.4 字典

字典（dict）就像它的名字一样，可以像查字典一样取查找。如：

```
>>> point = {'Sayori': 0}
>>> point
# 输出: {'Sayori': 0}

>>> point['Sayori'] = 2
>>> point['Monika'] = 1
>>> point['Natsuki'] = 3
>>> point['Yuri'] = 1

>>> point
# 输出: {'Sayori': 2, 'Monika': 1, 'Natsuki': 3, 'Yuri': 1}
```

```

>>> point['Sayori']
# 输出：2
```

3.1.5 布尔类型

布尔类型是最简单的一种类型。布尔类型只包括两个值真（True）、假（False）。如：

```
>>> is_act_two = False
>>> is_act_two
# 输出：False

>>> is_act_two = True
>>> is_act_two
# 输出：True
```

Python 中的内置数据类型均可进行逻辑运算和比较。如：

```
>>> a = False
>>> a is True
# 输出：False

>>> not a
# 输出：True

>>> b = 3
>>> b != 2
# 输出：True

>>> c = 2
>>> c <= b
# 即 2 <= 3

# 输出：True
```

详细逻辑操作符请见表3.2。

表 3.2: 常见逻辑、比较运算符与示意

操作符	描述	示例
==	比较两个对象是否相等	2 == 2; a == b
!=	比较两个对象是否不等	1 != 2; a != b
>	比较左对象是否大于右对象	3 > 2
<	比较左对象是否小于右对象	2 < 3
>=	比较左对象是否大于等于右对象	3 >= 2; 3 >= 3

操作符	描述	示例
<=	比较左对象是否小于等于右对象	2 <= 3; 3 <= 3
is	比较两个对象内存是否相等（更加严格的 ==）	a is True
not	逻辑非，用于反转操作数的逻辑状态。即 True 则为 False，False 则为 True	not True
and	逻辑与，当只有左操作数与右操作数皆为真时，条件为真	1 == 2 and 3 < 4
or	逻辑或，当左操作数和右操作数中有一个为真时，条件为真	1 == 2 or 10 > 6

扩展知识

此处只对 Python 的一些常见类型做了简单的介绍。Python 中还有其他类型，如可调用类（callable）等。本书目的以教学 Ren'Py 为主，故不会涉及 Python 过多。有兴趣者可以前往 https://www.runoob.com/python3/python3-tutorial.html 进行更深入的学习。

3.2 函数与类 [Python Only]

3.2.1 函数

在编程中，我们往往会重复执行一段代码或进行类似的操作。为了减少代码的重复，我们可以使用函数。函数的作用就是把相对独立的某个功能抽象出来，成为一个独立的个体。

3.2.1.1 函数的定义

定义一个函数，只需要开头为 def 即可。如下例：

```
def test(arg1, arg2):
    print("Arg1 is: " + arg1)
    print("Arg2 is: " + arg2)
    return
```

其中，test 为这个函数的名字，arg1、arg2 则为这个函数接受的参数。若留空，在代表该函数不接受参数。引号后的部分被称为函数主体，是调用该函数后具体的一些代码。return 语句则是函数运行成功后返回的值，可以留空。

调用函数也非常简单，如下例：

```
test(1, 2)
# 输出: Arg1 is: 1
# 输出: Arg2 is: 2

test(5, arg2=1)
# 输出: Arg1 is: 5
# 输出: Arg2 is: 1

```

```
test(arg2=4, arg1=2)
# 输出: Arg1 is: 2
# 输出: Arg2 is: 4

test(arg1=7, arg2=-7)
# 输出: Arg1 is: 7
# 输出: Arg2 is: -7
```

由此可见，在给函数传递参数时，可以直接传递，也可以使用“参数名 = 参数”的方式传递。

3.2.1.2 函数命名空间

所有在函数中的变量，都位于一个独立的命名空间内。该命名空间只能在该函数内使用，函数外的代码都无法读取或修改函数命名空间内的变量。同时，函数内也无法直接修改全局命名空间的变量。

如下例:

```
>>> a = 1
>>> def test():
...     a = 2
...     print(a)
>>> print(a)
# 输出: 1
>>> test()
# 输出: 2
>>> print(a)
# 输出: 1
```

如果要在函数内修改全局变量，则应使用 global 语句声明要使用的变量，如下例:

```
>>> a = 1
>>> def test():
...     global a
...     a = 2
...     print(a)


>>> print(a)
# 输出: 1
>>> test()
# 输出: 2
>>> print(a)
# 输出: 2
```

3.2.2 类

Python 是一门面向对象的语言，而类是一种用来描述具有相同属性和方法（函数）的对象的集合。它定义了该集合中每个对象共同具有的方法。对象是类的实例化。

3.2.2.1 类的定义

定义一个类，只需要开头为 class 即可。如下例：

```
class Test:
    def __init__(self):
        self.a = 1

    def counter(self):
        self.a += 1
        print(self.a)
```

上述例子中，定义了一个名为 Test 的类，这个类中有一个变量为 a，且有一个 counter 方法用于打印 a 的值。

> 扩展知识
>
> 在类中，以双下划线开头的、具有特殊的方法名的方法叫魔法方法（Magic Methods）。上述例子中的 __init__ 特殊方法用于在实例化一个类时会运行的初始化函数。类似的魔法方法还有 __eq__，__ne__ 等。
> self 是一个特殊的参数。当类被实例化后，无论调用其中的哪一个方法，Python 都会将这个对象自己传递给第一个参数。

3.2.2.2 类的使用

在使用类前，需要对类进行实例化。实例化后，类就会变成对象。创造对象和创造变量类似。如下例：

```
>>> test = Test()
>>> test.counter()
# 输出：2
>>> test.counter()
# 输出：3
```

第一行的“test = Test()”创造了一个 Test 对象。剩下两行代码则是在调用这个对象的 counter 方法。

如果要在类中如果要定义或使用一个属性，必须使用“self.”的方式进行赋值，否则这个属性就只会存在于方法的命名空间而不是对象的命名空间。

> 扩展知识
>
> Python 中类与函数的使用远远不止这些，您可以前往 https://www.runoob.com/python3/python3-function.html 与 https://www.runoob.com/

python3/python3-class.html 了解更多。

3.3 在 Ren'Py 中使用 Python 语句

在 Ren'Py 中有多种方式可以使用 Python 语句：Python 语句块（block）、单行 Python 语句、init python 语句。

3.3.1 Python 语句块

Python 语句块是使用 Python 的最方便的一种形式。一个 Python 语句块包含两个部分：

- 开头的声明
- Python 代码

如下例：

```
# Chapter 0
label ch0_start:
    python:
        # affection: 好感度
        n_aff = 0
        m_aff = 0
    scene bg club_day
    "{cps=20}快到{b}学园祭{/b}了。{/cps}{w=.5}{nw}"
    show monika 1a at l41 zorder 1
    m "各位！我们得开始准备了！"
    show sayori 1a at h42 zorder 1
    s "好耶！！！！"
    show natsuki 1a at t43 zorder 1
    n 2d "啊，我都等不及学园祭了。"
    n "肯定会很棒的！"
    show yuri 1a at s44 zorder 1
    y "..."
    scene bg club_day
    show monika 2a at t21 zorder 1
    show sayori 2a at t22 zorder 1
    m "那么，是时候来进行分工了。"
    m 4k "[player],你想要做什么？"
    menu:
        "做小蛋糕":
            s "夏树的小蛋糕最好吃了！"
            python:
                $ n_aff += 1
                $ m_aff -= 1
```

```
        "布置教室":
            m "那我们可得抓紧时间了！"
            python:
                $ m_aff += 1
                $ n_aff -= 1
    return
```

当出现多个 Python 语句块时，Ren'Py 会根据先后顺序依次执行。同时，我们可以使用 hide 与 in 关键词来改变 Python 语句块的行为。

hide 关键词会使 Python 语句块在一个独立的环境（命名区）下运行，即不与其他 Python 语句块共用一个环境。在具有 hide 关键词的 Python 语句块中的所有变量将不会被保存。

in 关键词则可以让 Python 语句块在一个独立的环境下运行。其他环境可以通过“储存区. 变量”来使用其他环境的内容。

如下例：

```
python:
    a = 2

python hide:
    a = 1

python in c:
    a = 4

python:
    print(c.a)
    print(a)
```

最后的运行结果为：4、2。

3.3.2 单行 Python 语句

大多数情况下，我们只有一行 Python 语句需要执行。此时使用 Python 语句块无疑会对开发者造成多余的输入。为了让编写只有一行的 Python 更方便快捷，Ren'Py 提供了单行 Python 语句。

单行 Python 语句以美元符号（$）开头。如：

```
$ s_aff = 0

$ winner = 'monika'
```

3.3.3 init python 语句

init python 语句在 Ren'Py 初始化阶段运行，会早于其他代码。这种功能可以用于定义类和函数或者配置变量。

扩展知识

在 init 与 python 之间还可以放一个运行优先级，默认为 0。init 语句将会按照从低到高的顺序执行。即优先级为-1 的代码会优先于优先级为 0 的代码执行。在优先级相同的情况下，Ren'Py 会根据本代码所在文件的 Unicode 码顺序执行。即在代码优先级都在 0 的情况下，a.rpy 内的代码总会优先于 b.rpy 内的代码执行。

必须注意

为了避免与 Ren'Py 自身代码冲突，您只应使用-999 到 999 范围内作为优先级。同时，原则上除特殊代码外，init 优先级应全部大于等于 0。

init python 语句也可以使用 hide 或 in 分句，与普通 python 语句用法相同。

在 init python 语句中的变量不会被存档。因此，在 init 语句中定义的应为常量。

```
# Chapter 0
init -1 python:
    demo = True

label ch0_start:
    scene bg club_day
    "{cps=20}快到{b}学园祭{/b}了。{/cps}{w=.5}{nw}"
    show monika 1a at l41 zorder 1
    m "各位！我们得开始准备了！"
    show sayori 1a at h42 zorder 1
    s "好耶！！！！"
    show natsuki 1a at t43 zorder 1
    n 2d "啊，我都等不及学园祭了。"
    n "肯定会很棒的！"
    show yuri 1a at s44 zorder 1
    y "..."
    scene bg club_day
    show monika 2a at t21 zorder 1
    show sayori 2a at t22 zorder 1
    m "那么，是时候来进行分工了。"
    m 4k "[player],你想要做什么？"

    if demo:
        "Demo 版剧情到此结束。"
        return

    menu:
        "做小蛋糕":
            s "夏树的小蛋糕最好吃了！"
```

```
            python:
                $ n_aff += 1
                $ m_aff -= 1
        "布置教室":
            m "那我们可得抓紧时间了！"
            python:
                $ m_aff += 1
                $ n_aff -= 1
    return
```

扩展知识

Python 语句块还可以使用 early 分句。在 Ren'Py 的运行生命周期中，python early 是被最早运行的。python early 中的代码将会早于所有代码运行，因此，python early 语句适合用来修改 Ren'Py 底层处理机制。但由于修改底层处理代码可能会导致一系列问题，我们不会在这里展开讲解。有兴趣者可以前往 https://doc.renpy.cn/zh-CN/lifecycle.html 了解 python early 语句的使用。

3.4 使用变量

3.4.1 python 语句块

关于如何在 Python 语句块中定义变量，请阅读第3.3章

3.4.2 define 语句

define 语句在初始化时将一个变量赋值。此变量视为一个常量，初始化之后不应再改变。例如：

```
define demo = True
define isActTwo = False
```

这段代码的运行效果等于：

```
init python:
    demo = True
    isActTwo = False
```

扩展知识

在 define 语句中同样可以指定优先级，只需要在 define 关键词后添加优先级即可，如：define 3 demo = True。

define 还可以为我们创建一个储存区，只需要将 define 关键词后的变量改为“储存区.变量”的形式即可。

3.4.3 default 语句

default 语句会给一个未被定义的变量初始赋值。default 语句适合用来定义在游戏过程中会变化的变量。如下例：

```
default demo = False
```

当 demo 这个变量在游戏开始后没有被定义，则将等价于在 start 脚本标签中定义 demo，且值为 False。若在存档加载后没有被定义，则等价于在 after_load 魔法标签中定义 demo，且值为 False。总而言之，若 demo 这个变量在游戏开始时没有被定义，那么它的值就是 False

3.4.4 持久化数据

持久化数据是 Ren'Py 中的一个储存区，无论用户怎样存档、读档，持久化数据区的数据总是独立于 Ren'Py 的存档数据的。如 DDLC 中，对于用户是否在一周目走过了每一条支线、解锁了每一个 CG 的字典（dict），就存储在持久化数据中，避免因读档存档导致数据消失。简单来说，持久化数据就是不会随着用户存档、读档而改变的数据。

一般来说，在使用 DDLC 中文 Mod 模板制作且没有对模版进行设置的游戏，在 Ren'Py 标准位置里会有一个名为 DDLCModTemplateZh 的文件夹，里面通常有一个名为 persistent 的文件，那就是存储持久化数据的存档文件。

持久化数据的用法就和普通变量一样，只不过在前面需要加上“persistent.”前缀。简单来说，持久化数据的语法如下：

```
1 persistent.<变量名> = <Python 数据类型>
```

例如：

```
1 default persistent.monika_deleted = False
```

无论用户如何重启游戏、读档、存档，除非在代码中对持久化数据进行修改或删除 persistent 文件，persistent.monika_deleted 的值永远都只会是 False。

3.5 流程控制

流程控制控制了程序运行的步骤。流程控制包括顺序控制、条件控制和循环控制。顺序控制，顾名思义，就是按照代码的先后顺序，从上到下依次执行代码。

3.5.1 脚本标签

在 Ren'Py 中，我们可以使用 label 语句，用自定义的标签名声明一个程序点位。这些标签用于调用或者跳转，可以使用在 Ren'Py 脚本、python 函数及各类界面中。

3.5.1.1 label 语句

在游戏中，故事常常会有多个走向，这是我们就需要编写多个分支。如果剧情只围绕一个分支来讲述故事，那么一定是很枯燥的。同时，如果我们把所有的代码都写在一个 label 里，无疑会对编写与后期的维护造成不必要的麻烦。这时候，就需要定义多个 label。

label 语句的基本语法为：

```
label <标签名>(参数 1, 参数 2):
    <语句 1>
    <语句 2>
    <语句 3>
    ...
```

如下例子:

```
label ch0_end:
    scene bg club_day
    "多么美好的一天啊！"
    return
```

警告

通常在 label 语句末尾，我们都会使用 return 来返回到上一个调用栈（stack）。在这里您可以理解为回到之前执行的函数、label 继续运行游戏。
如果没有 return 语句，在执行完本 label，Ren'Py 会继续调用在本 label 定义之后的 label。

label 可以在不同的文件内定义。例如我们现在在 game 目录下创造一个名为 script-ch0_tasks.rpy 的文件，并向这个文件中写入以下内容:

```
label ch0_monika:
    scene bg club_day
    show monika 1a at t11
    m "想好要做什么了吗？"
    return

label ch0_natsuki:
    scene bg club_day
    show natsuki 1a at t11
    n "不过，你知道怎么做小蛋糕吗？"
    return
```

代码 3.1: script-ch0_tasks.rpy

此时，我们在 ch0_start 标签中可以调用 ch0_monika 与 ch0_natsuki 标签。

3.5.1.2 call 语句与 jump 语句

现在，让我们修改一下 script-ch0.rpy 中的内容:

```
default n_aff = 0
default s_aff = 0
default demo = False

label ch0_start:
```

```
    scene bg club_day
    "{cps=20}快到{b}学园祭{/b}了。{/cps}{w=.5}{nw}"
    show monika 1a at l41 zorder 1
    m "各位！我们得开始准备了！"
    show sayori 1a at h42 zorder 1
    s "好耶！！！！"
    show natsuki 1a at t43 zorder 1
    n 2d "啊，我都等不及学园祭了。"
    n "肯定会很棒的！"
    show yuri 1a at s44 zorder 1
    y "..."
    scene bg club_day
    show monika 2a at t21 zorder 1
    show sayori 2a at t22 zorder 1
    m "那么，是时候来进行分工了。"
    m 4k "[player],你想要做什么？"

    if demo:
        "Demo 版剧情到此结束。"
        return

    menu:
        "做小蛋糕":
            s "夏树的小蛋糕最好吃了！"
            python:
                $ n_aff += 1
                $ m_aff -= 1
            call ch0_natsuki
        "布置教室":
            m "那我们可得抓紧时间了！"
            python:
                $ m_aff += 1
                $ n_aff -= 1
            call ch0_monika
    return
```

代码 3.2: script-ch0.rpy

运行上述代码，我们会发现在玩家做出选择后，执行了在 script-ch1_tasks.rpy 中的 label 中的内容。这就依赖于 call 语句和 jump 语句为我们提供的跳转功能了。

call 语句和 jump 语句可以将程序跳转到一个指定的脚本标签处，并且当指定的脚本标签执行完毕后，会自动返回到主控标签继续运行下面的代码。

call 语句和 jump 语句的语法如下:

```
call/jump <标签名>
```

或者

```
call/jump expression <label expressions>
```

如下例:

```
label main:
    scene bg club_day
    m "你现在正在主标签内。"
    $ today_winner = "sayori"
    call test_natsuki
    m "哦，你回来了？（跳转到 test_natsuki 标签后返回主标签。）"
    call expression "test" + today_winner
    m "你刚刚又去哪里了？（跳转到 test_sayori 标签后再次返回主标
        签。）"
    jump no_way
    return

label test_natsuki:
    n "你跳转到了 test_natsuki 标签内。"
    return

label test_sayori:
    s "你跳转到了 sub2 标签内。"
    return

label no_way:
    y "好吧，看起来你回不去了。"
    return
```

运行上述代码，我们会发现我们一开始会运行 main 标签中的内容。接着，我们会跳转到 test_natsuki 标签内并运行代码，运行完成后我们会返回到 main 标签中，随后再次跳转到 test_sayori 标签内，然后我们又回到了 main 标签中，最后，我们跳转到了 no_way 标签中，并且不再返回 main 标签，而是回到开始界面。

你或许注意到了，在 main 中我们并没有直接使用 call test_sayori 语句，而是使用了一个简单的表达式，然后把这个运算结果传递给了 call。这就是 expression 选项的作用。使用 expression 选项, 我们不用把标签名写死在程序里, 可以立即运算表达式的结果并传递给 call。这样做的好处是可以方便地在同一日的多个分支中跳转。

3.5.2 if 判断

大多数游戏中都具有多条剧情线。但面对一些只希望玩家触发多条剧情中的一条时，我们就可以利用 if 语句判断玩家的剧情线路。在 Python 和 Ren'Py 中，if 语句的基本语法为:

```
if <表达式>:
    <语句 1>
    <语句 2>
```

```
elif <表达式>:
    <语句 3>
else:
    <语句 4>
```

每一个 if 语句中的表达式都应当返回一个 True 或 False（见表3.2）。结果为 True 时，将会执行 if 语句块中的代码，如果结果为 False，Python 就会忽略 if 语句块内的所有代码。

扩展知识

表达式也可以是一个数字、一个字符串、或定义了 __bool__ 魔术方法的对象。
不为 0 的数字、非空的字符串以及 __bool__ 方法返回 True 的对象都会被视为 True

如下例：

```
if 1:
    print('1 is True.')

x = True

if x:
    print('x is True.')

if 1 + 1 == 2:
    print('Math is still correct.')
```

上述代码的输出结果为：

```
1 is True.
x is True.
Math is still correct.
```

当有多个表达式需要同时进行判断或当表达式为 False 时需要执行一些代码，我们就可以使用 elif 和 else 语句。

当 if 语句中的表达式为 False 时，会执行 else 语句的内容。请注意，if、elif 或 else 都必须跟在一起。如下例：

```
if 1 + 1 != 2:
    print('Math crashes!')
elif 1 + 2 == 2:
    print('Math crashes again!')
else:
    print("It's OK. Nothing crazy happened.")
```

3.5.3 循环

循环允许我们重复执行一段代码而不需要编写更多的代码。Python 中存在两种循环：while 循环与 for 循环。

3.5.3.1 while 循环

while 循环是 Python 和 Ren'Py 中最简单的循环。它的语法结构如下：

```
while <表达式>:
    <语句 1>
    <语句 2>
    ...
```

while 循环的表达式与 if 循环的表达式一样。只有表达式为 True 时才会执行 while 内的语句。例如：

```
i = 0
while i < 10:
    print(i)
    i += 1
```

输出结果为：

```
0
1
2
3
4
5
6
7
8
9
```

> **警告**
>
> 请注意，一般表达式不为 True，否则就会出现无限循环或死循环。如将上例中的 i+= 1 删去，就会导致 while 的表达式始终为 True，程序卡死在 while 循环。在后文中我们将会介绍 break 和 continue 语句来打破死循环。

3.5.3.2 for 循环 [Python Only]

for 循环比 while 循环的使用方法更加丰富。它的语法结构如下：

```
for <变量名> in <序列>:
    <语句 1>
    <语句 2>
    ...
```

这里的序列可以是列表、元组等可迭代对象。当序列中不再有变量后，for 循环会停止运行。如下例：

```
t1 = ('Sayori', 'Monika', 'Yuri', 'Natsuki')
```

```
l1 = [0, 0, 2, 0]

for i in t1:
    print(i)

for i in l1:
    print(i)
```

输出结果为:

```
Sayori
Monika
Yuri
Natsuki
0
0
2
0
```

扩展知识

range 函数是 Python 的内部函数之一，它可以为我们生成一个生成器（类似于列表，但比列表的性能更好）。使用 range 函数，我们可以快速生成一个从 0 开始到某数结束的一个生成器。当然，我们也可以指定 range 函数的起始数字与结束数字以及步长。如下例:

```
    for i in range(10):
        print(i)

    print("================================================")

    for i in range(10, 0, -1):
        print(i)
```

输出结果为:

```
    0
    1
    2
    3
    4
    5
    6
    7
    8
```

```
9
==========================================
10
9
8
7
6
5
4
3
2
1
```

上述第二个例子中的 10 就是起始数字，0 则为截止数字，-1 就是步长。

在部分情况中，我们希望可以跳过循环或退出循环体，这时我们就可以使用 break 和 continue 语句了。break 可以立即退出循环体，如下例：

```
for i in range(10):
    print(i)
    if i > 5:
        break
```

输出结果为：

```
0
1
2
3
4
5
```

continue 可以跳过当前的循环，如下例：

```
for i in range(10):
    print(i)
    if i == 5:
        continue
```

输出结果为：

```
0
1
2
3
4
6
7
```

```
8
9
```

同时，在 Python 中，循环也可以使用 else 语句。在 while 语句中的 else 语句会在 while 表达式为 False 时被执行。如下例：

```
i = 0
while i <= 9:
    print(i)
    i += 1
else:
    print(i, " is bigger than 9")
```

输出结果为：

```
0
1
2
3
4
5
6
7
8
9
10 is bigger than 9
```

同时，请注意 break 导致的循环体退出不会执行 else 语句中的内容。

3.5.4 错误和异常 [Python Only]

在 Python 中，不正常或语法错误的代码将会抛出异常。异常会使程序停止运行、崩溃、闪退等。常见的异常有：NameError（尝试使用一个未定义的变量）、IndexError（尝试访问在列表或元组范围外的索引）、TypeError（试图将两个不支持运算的类型进行运算）等。

为了处理这些异常，我们可以使用 try-except 语句。它的语法结构如下：

```
try:
    <可能抛出错误的语句>
except <错误类型>:
    <当错误被捕获后的语句>
```

如下例子：

```
try:
    x = 1 / 0
except ZeroDivisionError:
    print('Error: Cannot divide by zero.')
```

输出结果为：

```
Error: Cannot divide by zero.
```

进阶的语法包含 finally 从句或 else 从句。请注意，finally 从句与 else 从句不可并存。

在 try-except-finally 中，无论 try 代码块中的代码是否抛出异常，finally 从句中的代码都一定会被执行。try-except-finally 的语法如下：

```
try:
    <可能抛出错误的语句>
except <错误类型>:
    <当错误被捕获后的语句>
finally:
    <无论是否抛出错误都会执行的语句>
```

如下例子：

```
div = 0
def func():
    global div
    return 1 / div

try:
    x = func()
except ZeroDivisionError:
    print('Error: Cannot divide by zero.')
finally:
    x = 10

print(x)

div = 10
try:
    x = func()
except ZeroDivisionError:
    print('Error: Cannot divide by zero.')
finally:
    x = 100
print(x)
```

输出结果为：

```
Error: Cannot divide by zero.
10
100
```

try-except-else 中，只有当 try 语句块中的内容没有抛出异常时，else 中的内容才会被执行。try-except-else 的语法如下：

```
try:
```

```
    <可能抛出错误的语句>
except <错误类型>:
    <当错误被捕获后的语句>
else:
    <当没有错误被捕获时执行的语句>
```

如下例：

```
try:
    x = 1 / 0
except ZeroDivisionError:
    print('Error: Cannot divide by zero.')
else:
    print("The value of x is ", x)

try:
    x = 1 / 10
except ZeroDivisionError:
    print('Error: Cannot divide by zero.')
else:
    print("The value of x is ", x)
```

输出结果为：

```
Error: Cannot divide by zero.
The value of x is 0.1
```

3.6 等效语句

等效语句是在 Python 中使用 Ren'Py 语法的一种方式。等效语句只能在 Python 代码中运行。

3.6.1 对话

Ren'Py 的 say 语句对应两种等效语句。如下例：

```
m "你好！"
```

这段代码不仅等效于下列代码：

```
$ m("你好！")
```

同时也等效于：

```
$ renpy.say(m, "你好")
```

不过，为了保持语义上的通畅，我们常使用后者。

3.6.2 图像显示

3.6.2.1 show

show 语句的等效语句的语法如下:

```
renpy.show(<name>, at_list=<position>, zorder=<zorder number>)
```

3.6.2.2 hide

hide 语句的等效语句的语法如下:

```
renpy.hide(<name>)
```

3.6.2.3 scene

scene 语句的等效语句的语法如下:

```
renpy.scene()
renpy.show(<name>)
```

3.6.2.4 with 从句

with 语句的等效语句的语法如下:

```
renpy.with_statement(<name>)
```

3.6.3 call 和 jump

call 和 jump 语句的等效语句的语法如下:

```
renpy.call(script=<script name>)
renpy.jump(screen=<script name>)
```

对于更多关于等效语句的介绍,请查阅https://doc.renpy.cn/zh-CN/statement_equivalents.html

第四章　增添资源

本节目标包括：

- 学会增添图像、音频资源；
- 学会使用 image 语句定义图像；
- 学会定义音频

在开发中，我们往往会想增添一些资源，如更多的背景音乐、更丰富的角色表情与背景、为角色增添更多服装等。在本章中，我们将重点学习如何管理模组资源，往模组中增添新的图像、音频等。

4.1　增添资源

一般来说，在 Mod 工程里的 game 文件夹下会有一个名叫 mod_assets 的文件夹。所有在 mod_assets 文件夹里的内容都会在生成发行版时被打包成一个 rpa 文件。（在后续的章节里，我们将会详细学习如何控制哪些文件打包、哪些不打包，以及如何生成发行版）

扩展知识

目前，常见的资源获取方法有：从 DDLC Community Assets 获取（Google Drive）、从一些开发 QQ 群提供的资源、Reddit 等。

必须注意

在使用资源前，请务必注意版权问题。一般来说，在 DDLC Community Assets 中的文件只需要在感谢名单中添加作者的名字，但也有部分资源会有更多的要求。请记住：在一个资源必须得到作者授权的情况下，没有得到授权就使用该资源的行为是侵权行为。保护作者的知识产权，不仅是在维护作者的权利，更是促进 DDLC 社区的更好发展。

4.1.1　定义角色

在一些模组中，我们常常需要添加一些角色来丰富故事线，或是推动故事情节的发展。要增加角色，我们需要先了解如何定义一个角色。

定义一个角色有两种方式：Character 与 DynamicCharacter。两种定义方式几乎没有区别，唯一的区别在于 Character 的名字是固定的，无法更改的；而 DynamicCharacter 则使用一个变量作为角色名，是动态的，可以更改。由于 DDLC 中角色都使用 DynamicCharacter 来

定义角色，故本书不会介绍如何使用 Character 定义，感兴趣者可以前往 Ren'Py 中文文档了解。

DynamicCharacter 的语法如下：

```
define <变量名> = DynamicCharacter('<存储角色名的变量名>', image='<say语句的对话属性图像名>', what_prefix='"', what_suffix='"', ctc="ctc", ctc_position="fixed")
```

警告

对于 say 语句的对话属性来说，如果 image 后的图像名错误，那么将会导致其无法使用。所以请务必确保 image 后的图像名与您定义的角色立绘名一致。对于角色立绘名，请参考2.2.2.1

扩展知识

在某些情况下，我们可能改变角色说话内容的双引号为其他符号，我们可以通过修改 what_prefix 和 what_suffix 实现效果。

正常情况下，除旁白外所有的角色说的话都会被双引号包住。如：

```
m "你好"
```

那么在游戏中的效果为："你好"。如果我们想要修改为：「你好」，那么只需要找到莫妮卡角色的定义，并将 what_prefix 改为'「', what_suffix 改为'」'即可实现。实际代码如下：

```
define m = DynamicCharacter('m_name', image='monika', what_prefix='「', what_suffix='」', ctc='ctc', ctc_position='fixed')
```

如我现在需要定义一个名为 Charlie 的角色，其使用的图像名为 charlie，存储角色名的变量名叫 c_name：

```
define c_name = "Charlie"
define c = DynamicCharacter('c_name', image='charlie', what_prefix='"', what_suffix='"', ctc="ctc", ctc_position="fixed")
```

现在，我们就成功定义了 Charlie 这个角色。那么这段代码应该放在哪里呢？答案是 game 目录下的 definitions.rpy 文件内。这个文件里储存着所有在游戏中需要用到的资源的定义。

打开编辑器的查找功能，搜索："define s = DynamicCharacter"，随后在这一行上面或下面增添例如上述的代码。

现在，definitions.rpy 文件内的代码应该长这样：

```
    # 角色变量

    define narrator = Character(ctc="ctc", ctc_position="fixed")
```

```
4     define mc = DynamicCharacter('player', what_prefix='"',
          what_suffix='"', ctc="ctc", ctc_position="fixed")
5     define s = DynamicCharacter('s_name', image='sayori',
          what_prefix='"', what_suffix='"', ctc="ctc", ctc_position=
          "fixed")
6     define m = DynamicCharacter('m_name', image='monika',
          what_prefix='"', what_suffix='"', ctc="ctc", ctc_position=
          "fixed")
7     define n = DynamicCharacter('n_name', image='natsuki',
          what_prefix='"', what_suffix='"', ctc="ctc", ctc_position=
          "fixed")
8     define y = DynamicCharacter('y_name', image='yuri',
          what_prefix='"', what_suffix='"', ctc="ctc", ctc_position=
          "fixed")
9     define ny = Character('夏树 & 优里', what_prefix='"',
          what_suffix='"', ctc="ctc", ctc_position="fixed")
10    define c = DynamicCharacter('c_name', image='charlie',
          what_prefix='"', what_suffix='"', ctc="ctc", ctc_position=
          "fixed")
11
12    # ...
13
14    # Default Name Variables
15    default s_name = "纱世里"
16    default m_name = "莫妮卡"
17    default n_name = "夏树"
18    default y_name = "优里"
19    default a_name = "Charlie"
```

4.1.2 增加、定义图片

现在，在 mod_assets 文件夹下创造一个名为 images 的文件夹。在后续的教程中，我们将会把所有的图片资源都存储在这个 images 文件夹内。

警告

请注意，对于所有图片，其格式都应为 PNG，且背景应该是透明的，图像编号不能和已有的重复。角色立绘资源的分辨率应为 960x960 以保证兼容性，且对于一个完整的立绘应遵循原版 DDLC 的原则分成三个图像：头部、左半身、右半身。背景图像的尺寸应为 16:9，分辨率应该为 1280x720。如果图像资源分辨率扩大，您可以使用软件降低图像分辨率或使用 size 属性解决：

```
1 size (1280,720) # 添加 size 属性
```

> 为了兼容性，我们建议您新建的所有文件（夹）名称全部为英文小写字母，且不使用中文。

4.1.2.1　角色立绘

在获取角色立绘后，在 images 文件夹下创建一个文件夹，名为您想要添加立绘的角色名，如 sayori。将图像放进您刚才创建的文件夹内，同时我们建议您将图像名改为 26 个小写字母或 0-9 数字中的任意一个（建议从 a 开始到 z 依次排列并遵循2.2.2.1中的定义）。

增添图像后，接下来我们就该定义角色立绘了。定义图像的语法如下：

```
image <图像名> = "<资源地址>"
```

使用 image 定义后的图像就可以被 show 语句使用了。

由于 DDLC 将角色立绘拆分成为了三部分（请参考2.2.2.1），故对于角色立绘的定义会非常的复杂。

您可以按照以下步骤判断选取相应的代码进行修改：

1. 您添加的立绘是 DDLC 原有角色：

   (a) 您只添加了表情：

   ```
   image <角色名> <立绘编号> = im.Composite((960, 960), (0,
       0), "<角色名>/<数字>l.png", (0, 0), "<角色名>/<数字>r.
       png", (0, 0), "mod_assets/images/<角色名>/<图像资源>.
       png")
   ```

   (b) 您只添加了身体姿势：

   ```
   image <角色名> <立绘编号> = im.Composite((960, 960), (0,
       0), "mod_assets/images/<角色名>/<数字>l.png", (0, 0), "
       mod_assets/images/<角色名>/<数字>l.png", (0, 0), "<角色
       名>/<字母>l.png")
   ```

   (c) 您同时添加了表情与身体姿势：

   ```
   image <角色名> <立绘编号> = im.Composite((960, 960), (0,
       0), "mod_assets/images/<角色名>/<数字>l.png", (0, 0), "
       mod_assets/images/<角色名>/<数字>l.png", (0, 0), "
       mod_assets/images/<角色名>/<图像资源>.png")
   ```

2. 您添加的立绘是新定义的的角色：

   ```
   image <角色名> <立绘编号> = im.Composite((960, 960), (0, 0),
       "mod_assets/images/<角色名>/<数字>l.png", (0, 0), "
       mod_assets/images/<角色名>/<数字>l.png", (0, 0), "
       mod_assets/images/<角色名>/<图像资源>.png")
   ```

例如现在为优里增添了新的服装与表情，且图像资源名分别为 1l, 1r, happy.png 那么您使用的代码应该长这个样子：

```
# m means mod
images yuri m1a = im.Composite((960, 960), (0, 0), "mod_assets/
    images/yuri/1l.png", (0, 0), "mod_assets/images/yuri/1r.png",
    (0,0), "mod_assets/images/yuri/happy.png")
```

对于角色立绘的列举，您可以前往https://docs.dokimod.top/pages/0a59cf/查看。

4.1.2.2 背景图像

在获取背景图像后，在 images 文件夹下创建一个名为 bg 文件夹。将背景放进您刚才创建的文件夹内，同时我们建议您将图像名改为小写的单词。

增添背景后，接下来我们就该定义背景了。定义背景的语法如下：

```
image bg <背景名> = "mod_assets/images/bg/<背景图像资源名>.png"
```

扩展知识

这里的 bg 指 background，它实际上是图像名的一部分，其目的是为了区分角色立绘与背景立绘。

4.1.3 音频资源

现在，在 mod_assets 文件夹下创造一个名为 audio 的文件夹。将音频放进您刚才创建的文件夹内，同时我们建议您将音频名改为小写的单词。

警告

Ren'Py 只支持以下几种音频格式：

- Opus
- Ogg
- MP3
- WAV

然后，打开 definitions.rpy 文件，新增以下代码：

```
define audio.<音频名> = "mod_assets/audio/<音频文件名>.<后缀名>"
```

请注意，音频名不能和已有的定义重复，否则会覆盖定义。

4.1.3.1 节选播放

Ren'Py 支持将音频切割成一段播放。其中，共有三种行为：

- from
- to
- loop

from to　from 与 to 标签用于指定播放文件的起始位置和终止位置。其基本用法为：

```
<from [开始] to [结束]>
```

例如，现在有一个音频名为“festival.ogg”存放在 mod_assets 文件夹中。现在我们想要从这个音频的第 5 秒开始播放，到第 15 秒停止，我们可以修改音频的定义为：

```
define festival = "<from 5 to 15>mod_assets/festival.ogg"
```

这样，在播放 festival 音频时就会从第 5 秒开始，到第 15 秒结束。

loop　loop 标签用于指定音频循环播放。其基本用法为：

```
<loop [开始，默认为0]>
```

例如，现在有一个音频名为“music.ogg”，我们想要从第 10 秒开始播放，并循环播放这个音频，可以修改音频的定义为：

```
define bg_music = "<loop 10>mod_assets/music.ogg"
```

第五章　特殊效果及特殊脚本

本节目标包括：

- 了解 DDLC 中一些二周目的特殊效果；
- 了解 DDLC 中的一些特殊脚本；
- 学会定义一首新诗歌并对其进行调用。

在 Mod 中，二周目的一些特殊效果往往是有用的，特别是一些解谜类或恐怖类的 Mod。在本章中，我们将了解 DDLC 模板中的一些特殊脚本与 DDLC 二周目中的一些特殊效果。同时，我们也会在本章学习如何定义一首新的诗歌并调用。

5.1　特殊脚本

打开 game 文件夹，我们能够发现许多 rpy 文件。在本节中，我们将会了解这些文件内所定义的内容、它们的作用与使用方法。

5.1.1　bsod.rpy

bsod.rpy 中定义了 DDLC 二周目中“电脑蓝屏”的特殊效果。代码会自动检测当前电脑的类型、版本，并生成对应的“死机”效果。

使用方法：

```
call screen bsod()

call screen bsod(bsodCode="<错误代码>", bsodFile="<蓝屏文件，默认
    libGLESv2.dll>", rsod=<bool值，默认为False>, chinese_screen=<
    bool值，默认为True>)
```

其中，bsodCode 是一串伪造的错误代码。默认为 DDLC_ESCAPE_PLAN_FAILED。

bsodFile 是导致蓝屏的文件名，默认为 libGLESv2.dll。

rsod 表示是否在将“蓝屏”替换为“红屏”，默认为 False。

chinese_screen 表示是否在使用中文版本，默认为 True。

5.1.2　cgs.rpy

这个文件中定义了 DDLC 中的所有 CG 内容。

5.1.3　console.rpy

这个文件中定义了假结局中莫妮卡删除 CG 时使用的控制台。其使用方法为:

```
call updateconsole(text="<命令>", history="<提示消息>") # 第一次调用控制台

call updateconsolehistory(text="<提示消息>") # 更新提示消息

call hideconsole  # 隐藏控制台
```

例如:

```
call updateconsole(text="print('Hello, World'), history="Hello, World")
```

运行代码, 我们发现在游戏的左上角出现了控制台, 并自动输入了“print('Hello, World')”这串代码，执行结果为“Hello, World”

5.1.4　glitchtext.rpy

glitchtext.rpy 中定义了乱码文字。我们可以通过调用 glitchtext 函数生成一段乱码文字。其使用方法如下:

```
$ <变量名> = glitchtext(<长度>)
```

例如我想要生成一串长度为 200 的乱码字符，并把它赋值给一个名为 gtext 的变量，可以用以下的代码:

```
$ gtext = glitchtext(200)
```

5.1.5　poems_special.rpy\poems-tl.rpy\poems.rpy

这三个文件共同存储了在游戏中出现的诗歌。其中 poems_special.rpy 定义了二周目的特殊诗歌，poems-tl.rpy 存储着中文版本的诗歌，poems.rpy 则是原版的诗歌。

我们可以按照以下模板定义一首新的诗歌:

```
poem_<编号> = Poem(
    author = <作者名>,
    title = <标题>,
    text = """\
<具体诗歌内容>"""
    )
```

同时我们也可以通过 showpoem 来展示诗歌。如下例:

```
poem_a = Poem(
    author = "mc",
    title = "Empty",
    text = """\
```

```
婚姻，热情，童年，快乐，色彩，希望，友人，可爱，松软，单纯，糖
    果，购物，狗狗，猫咪，云朵，决意，自杀，想象，秘密，活泼，存
    在，璀璨，绯红，飓风"""
)

call showpoem(poem_a)
```

后记：DDLC Cn 路在何方?

说起来，我是从 2019 年左右接触到 DDLC 这个圈子的。那时 DDLC Cn 还不像现在这样，各种二创、模组到处都是，心跳心跳文学部吧还没有被封，每天都在贴吧不是催更二创小说就是水经验。后来的事情各位也都知道了，DDLC 被封杀，成为“禁游”，贴吧被封、视频被下架、小说也看不到了。然后我就慢慢淡出了 DDLC 这个圈子，把它抛到了脑后。

后来有一天，接触到了 Salvation 这个模组，那个时候第一次装模组啥也不会，整了半天发现是英文人都傻了。后面又找了半天都没找到汉化，就干脆拿翻译器一点点的生啃。结局是怎样的我已经记不清了，但当时对我影响特别的大，以至于我下定了决心想要创造属于自己的一个 Good Ending。

后面就是学习 Python、学习 Ren'Py，当时完全不知道有 DokiMod 开发文档，只知道有个中文模板，完全靠 Ren'Py 中文文档和 B 站上的速成 Ren'Py 视频学的。但 DDLC 的很多东西不按 Ren'Py 的套路走，比如说角色定义吧，Ren'Py 中文文档中使用的是 Character 进行定义，但为了做到角色名动态，DDLC 使用的是 DynamicCharacter，这个东西文档里根本就没有，导致我刚开始学的时候懵逼了好久这个 DynamicCharacter 怎么用的。后面各种解包别人的模组看别人代码怎么写的，对于 Ren'Py 也是渐渐的熟悉起来了。然后一次无聊偶然发现了 DokiMod 开发文档，个人认为是一篇很好的教程，但后面想要学一些更高级的内容就完全只能靠自己。（后面还为这个文档写过好几篇教程呢，真不知道自己那个时候技术力没多少怎么好意思给别人写教程的，写的内容又低脂技术力又低，还特别没有逻辑，多亏 Dobby233Liu 给我删减修改才让我的文章不至于那么不堪入目。）

到了今天，其实 DDLC Mod 开发已经很成熟了，但就是缺乏一个系统的教程指引初学者一步步学习，这也是我为什么要写这本指南的原因————让初学者能够更轻松容易的学习。但如今 DDLC Cn Modder 少之又少，知名的 Mod 似乎都是国外的，是什么导致了如今圈内 Mod 如此少呢，我想原因之一可能就是缺乏一个系统的指南。很多人其实都想要创作自己的 Mod，但都止步于第一步：从哪里开始？写这本书的原因就是为了帮助人克服这第一个困难。但其实第一步迈开之后，你会发现后路更加困难了，你是否有足够的时间、经历、兴趣去开发 Mod。包括我自己的 Mod 其实都已经很长时间没有继续编写了，因为我其实没有那么大的兴趣去坚持我走下去，促使我去写代码。写代码是一个极其耗费精力、极其考验人的东西，但我想这并不意味着我放弃了我的 Mod。

总之，感谢你看到这里，希望你能够坚持走下 Mod 开发这条道路，也希望有一日我能看到 DDLC 这个圈子像 19 年那样如此富有活力。