2.4 函数与模块

2023 年 9 月 18 日

函数与模块

一、函数

在 Python 语言中，函数是一段可重复使用的代码块，用于完成特定的功能。函数可以接受输入参数（也称为参数或参数）并返回一个结果。

函数的主要优点是代码的可重用性和模块化。通过将代码封装在函数中，可以在程序中多次调用函数，避免重复编写相同的代码。函数还可以提高代码的可读性和维护性，因为它们将复杂的逻辑划分为较小的、可理解的部分。

以下是一个简单的 Python 函数的示例：

[1]:
```
def greet(name):
    """ 打招呼的函数"""
    print("Hello, " + name + "!")

# 调用函数
greet("Alice")
greet("Bob")
```

```
Hello, Alice!
Hello, Bob!
```

在上面的示例中，greet 函数接受一个参数 name，并在函数体中打印出相应的问候语。通过调用函数并传递不同的参数，可以多次执行问候的操作。

1.1 函数如何定义

在 Python 中，定义一个函数要使用 `def` 语句，依次写出函数名、括号、括号中的参数和冒号:，然后，在缩进块中编写函数体，函数的返回值用 `return` 语句返回。

```
def 函数名 (参数1, 参数2, ···):     # 注意冒号是语法必备元素
    """ 本函数的介绍"""              # 函数的注释
    函数语句块                      # 缩进表明语句块内的所有语句都在本函数作用域下
    或称函数体                      # 函数体中的代码可以是任意有效的 Python 代码，包括变量声明、条件
    return 表达式                   # 如果函数没有指定返回值，它将默认返回 None
```

函数可以接受零个或多个参数。参数是函数定义中用于接受输入值的变量。在函数调用时，可以向函数传递实际的参数值。

我们以自定义一个求绝对值的 `my_abs` 函数为例：

[2]:
```
def my_abs(x):
    """ 这是一个求绝对值的函数"""
    if x >= 0:
        return x
    else:
        return -x
```

现在我们调用它看看，给其传入一个负数，例如-99：

[3]:
```
my_abs(-99)
```

[3]: 99

练习

请设计一个比较大小的函数，compare(x, y)，使用 return 语句返回两者中较大的值。

[ ]:
```

```

二、模块

在 Python 中，**模块（module）**是指一个包含了 Python 代码的文件。模块可以包含函数、变量和类等定义，以及可执行的代码。

模块的主要目的是将相关的代码组织在一起，以便在需要时进行重复使用。通过将代码分解为模块，可以提高代码的可维护性、可读性和重用性。

要创建一个模块，只需创建一个以.py 为扩展名的 Python 脚本文件，并在其中编写相应的代码。

2.1 import 模块名

当您使用 import 语句加载模块时，可以直接指定模块的名称，例如：

```
import module_name
```

这将加载名为 module_name 的模块，并使其在当前代码中可用。您可以使用模块中定义的函数、变量和类等。

我们以 datetime 模块为例，datetime 模块，提供当日的日期，试运行以下代码：

[4]:
```
import datetime
```

[5]:
```
now = datetime.datetime.today()
print(now)
```

```
2023-09-17 18:21:48.763584
```

2.2 from 模块名 import 函数名

此外，还可以使用 from 关键字结合 import 语句来选择性地加载模块或包中的特定内容。例如：

```
from module_name import function_name
```

这将从 module_name 模块中加载 function_name 函数，并使其在当前代码中可用。

例如，

[6]:
```
from math import sqrt
sqrt(2)
```

[6]:
```
1.4142135623730951
```

练习

1. 在当前目录下，点击右上方蓝色 ☐ 号，新建一个 Python 文件
2. 在文件中放入上一练习的函数 compare(x, y)，保存文件名称为 my_func.py
3. 在下方空格处，导入该文件中的 compare 函数
4. 调用 compare 函数，比较 2 的 3 次方和 3 的 2 次方的大小。

[ ]:
```

```

三、包

3.1 包的概念

Package（包）是一个包含多个模块的目录。包用于组织和管理相关的模块，可以形成层次结构。包目录中通常包含一个特殊的 __init__.py 文件，用于标识该目录为一个包。

例如，您可以创建一个名为 my_package 的包，其中包含多个模块文件:

```
my_package/
    __init__.py
    module1.py
    module2.py
```

可以通过导入包和其中的模块来使用其中的功能:

```
import my_package.module1
import my_package.module2


my_package.module1.function1()
my_package.module2.function2()
```

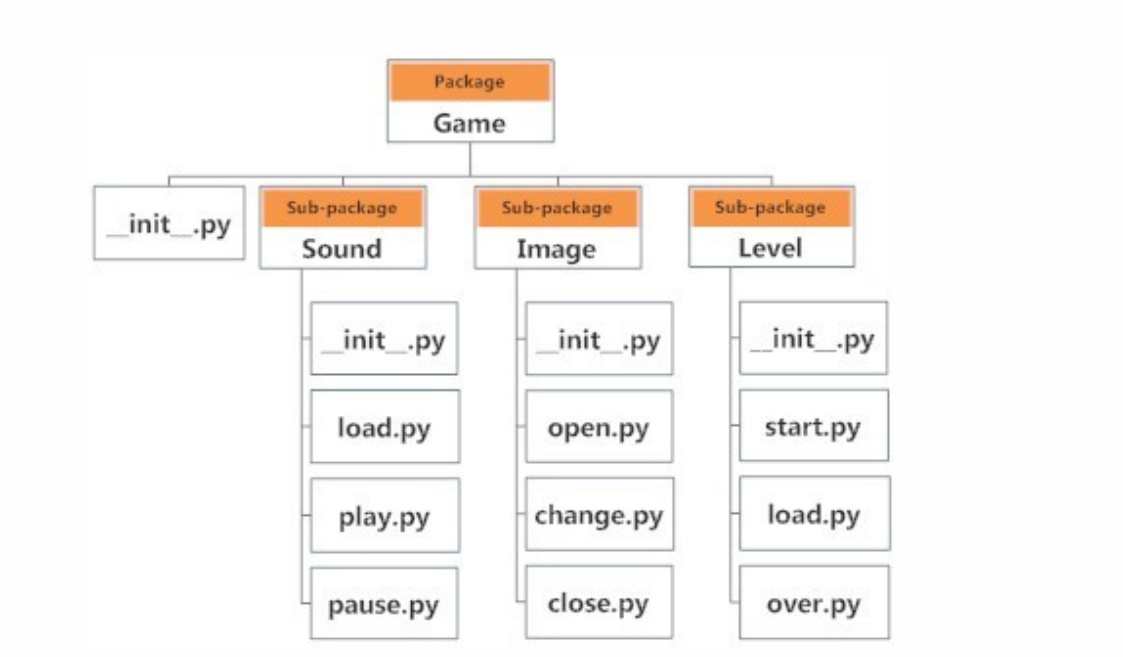
Python 中的 import 语句既可以用于加载模块（module），也可以用于加载包（package）。您也可以从包中加载特定的模块或其他内容。例如:

```
from package_name import module1
```

这将从 package_name 包中加载 module1 模块，并使其在当前代码中可用。

3.2 包的结构

以下图为例子，这是一个和游戏相关的程序，它的文件结构如下:



这里 Game 文件夹叫包（Package），Sound、Image 和 Level 这三个文件夹叫子包 (Sub-package)。它们各自包含自己的.py 文件，这些文件叫模块。

从这里我们可以看到，包是一个有层次的文件目录结构，它定义了由 n 个模块或 n 个子包组成的 python 应用程序执行环境。通俗一点：包是一个包含 __init__.py 文件的目录，该目录下一定得有这个 __init__.py 文件和其它模块或子包。

注意，每一个包中的 `__init__.py` 文件是必须存在的，否则，Python 就会把这个目录当成普通目录 (文件夹)，而不是一个包。`__init__.py` 可以是空文件，也可以有 Python 代码，因为 `__init__.py` 本身就是一个模块，而它的模块名就是对应包的名字。

3.3 Python 的标准库

Python 的标准库是一组预先安装在 Python 解释器中的模块和包，它们提供了广泛的功能和工具，可用于各种常见的任务和应用程序开发。

Python 的标准库既包含模块，也包含包。标准库中的一些功能是通过单个模块提供的，例如 `math` 模块用于数学计算，`random` 模块用于生成随机数，`datetime` 模块用于处理日期和时间等。您可以使用 `import` 语句来加载这些模块，并在您的代码中使用它们提供的功能。

另一方面，标准库中的一些功能是通过包的形式组织的。包是一个包含多个模块的目录，通常还包括一个特殊的 __init__.py 文件来标识该目录为一个包。这些包提供了一组相关的模块，用于解决特定的问题或提供特定的功能。例如，`os` 包提供了与操作系统交互的功能，`re` 包提供了正则表达式操作的功能。您可以使用 `import` 语句加载这些包，并使用包中的模块和功能。

这里只列举了部分，更多请见官方文档: 链接

模块或包	功能
os	提供了许多与操作系统交互的函数
datetime	模块提供了以简单和复杂的方式操作日期和时间的类
re	为高级字符串处理提供正则表达式工具
random	模块提供了进行随机选择的工具
doctest	用于扫描模块并验证程序文档字符串中嵌入的测试

3.4 Python 的第三方包

Python 的第三方包是由 Python 社区开发并提供的，不属于 Python 标准库的软件包。这些包通常由第三方开发者创建和维护，并提供了各种额外的功能和工具，可以扩展 Python 的能力。

第三方包可以用于各种用途，例如数据处理、网络通信、Web 开发、科学计算、机器学习、图像处理等。它们提供了高级功能和库，使开发人员能够更快速、更高效地开发应用程序。

Python 的第三方包可以通过包管理工具（如 pip）进行安装。安装这些包后，您可以在自己的 Python 项目中使用它们。常见的第三方包有：

Python 的第三方包	功能
Numpy	提供了数组、矩阵和数值计算功能
Pandas	提供了强大、便捷的数据分析工具
Matplotlib	用于绘制图表和可视化数据的库
Scipy	提供了科学计算、统计等相关工具
Statsmodels	提供了多元回归和分析的工具
Requests	用于发送 HTTP 请求的库
Django	用于构建 Web 应用程序的高级框架
Flask	用于构建轻量级 Web 应用程序的框架
TensorFlow	用于机器学习和深度学习的库
Scikit-learn	提供了各种机器学习算法和工具
BeautifulSoup	用于解析和提取 HTML/XML 数据的库

这只是一小部分常见的第三方包示例，Python 社区有数以千计的第三方包可供选择，覆盖了各种不同的领域和用途。

通过使用第三方包，您可以利用 Python 生态系统中其他开发者的工作成果，加快开发速度，避免重复造轮子，并且能够更加便捷地实现复杂的功能。

3.5 pip 包管理器

在 Python 中，pip 是一个用于管理和安装第三方包的包管理工具。它是Python Package Index（PyPI）的默认包管理器。

使用 pip，您可以轻松地搜索、安装、升级和卸载 Python 包。

以下是一些常用的 pip 命令示例，在 shell 命令行环境下，输入

1. 安装包：

```
pip install package_name
```

该命令会从 PyPI 下载并安装指定的包。

2. 升级包：

```
pip install --upgrade package_name
```

该命令会升级已安装的包到最新版本。

3. 卸载包：

```
pip uninstall package_name
```

该命令会卸载指定的包。

4. 列出已安装的包：

```
pip list
```

该命令会列出当前环境中已安装的所有包。

pip 还支持其他一些命令和选项，您可以使用 `pip --help` 命令或查阅 pip 的文档来获取更多信息。

需要注意的是，pip 默认安装包到 Python 的全局环境中，如果你想安装到本用户下，在命令中追加使用'-u' 参数。如果您使用虚拟环境（virtual environment），您可以在虚拟环境中使用 pip 来管理包，以避免与全局环境中的包冲突。