第九章 - 函数

张建章

阿里巴巴商学院

杭州师范大学

2024-09

1 抽象的概念及意义

2 自定义函数的定义与调用

3 函数的参数

4 局部变量与全局变量

抽象是计算机科学中的核心概念，旨在隐藏复杂的实现细节，突出核心功能，从而提高程序的可读性、可维护性和扩展性。

在编程实践中，函数抽象是实现这一目标的主要手段之一。通过将重复的逻辑封装在函数中，开发者可以减少代码冗余，提升代码的模块化程度。

例如，考虑一个需要计算多个矩形面积的场景。如果不使用函数，可能会多次编写相同的计算逻辑：

```
# 计算第一个矩形的面积
width1 = 5
height1 = 10
area1 = width1 * height1

# 计算第二个矩形的面积
width2 = 3
height2 = 7
area2 = width2 * height2
# 计算第三个矩形的面积
```

上述代码存在明显的重复，增加了维护难度。通过定义一个计算矩形面积的函数，可以有效地抽象出重复的逻辑：

```
def calculate_area(width, height):
    return width * height

# 使用函数计算面积
area1 = calculate_area(5, 10)
area2 = calculate_area(3, 7)
area3 = calculate_area(6, 9)
```

通过这种方式，计算面积的逻辑被封装在 `calculate_area` 函数中，调用者只需提供不同的参数即可复用该逻辑。这不仅减少了代码冗余，还提高了代码的清晰度和可维护性。

函数的定义使用 `def` 关键字，后跟函数名和括号中的参数列表。函数内部的代码块通常由缩进的语句组成，而 `return` 语句用于返回函数的结果。

1. 函数的定义与调用

函数的基本定义格式如下：

```
def function_name(parameters):
    # 执行的代码块
    return result
```

在定义函数时，`def` 后接函数名，再由圆括号包围的参数列表，可以为函数的参数指定默认值。如果函数没有参数，可以省略。如果函数需要返回一个结果，可以使用 `return` 语句将计算结果返回给调用者。

2. 参数的传递

Python 支持多种参数传递方式，包括位置参数、关键字参数和混合方式。位置参数是最常见的类型，调用时传入的值按照位置匹配到相应的参数。

```
def add(a, b):
    return a + b

result = add(3, 5)
print(result)  # 输出 8
```

除了位置参数，Python 还支持使用关键字参数来进行函数调用，这使得传递参数时不受位置顺序的限制。例如：

```
def describe_pet(animal_type, pet_name):
    print(f"I have a {animal_type} named {pet_name}.")

describe_pet(animal_type="dog", pet_name="Buddy")
```

3. 混合方式

混合使用位置参数和关键字参数进行函数调用时，位置参数必须在关键字参数前面。

```
def multiply(a, b):
    return a * b

print(multiply(4, b = 5))   # 输出 20
print(multiply(a = 4, 5))   # 错误，位置参数必须在关键字参数前面
```

4. 返回值的使用

函数可以返回值，`return` 语句用于指定返回值。如果函数没有 `return` 语句，它会默认返回 `None`。返回的值可以用于后续的计算或处理。例如：

```
def multiply(a, b):
    return a * b
```

5. 示例代码：带参数的函数

以下是一个包含多个参数、返回值以及默认参数的完整示例：

```python
def calculate_area(length, width=1):
    """ 计算矩形的面积，宽度参数有默认值"""
    return length * width

# 使用位置参数
area1 = calculate_area(5, 3)
print(f"Area 1: {area1}")  # 输出 Area 1: 15

# 使用默认参数
area2 = calculate_area(5)
print(f"Area 2: {area2}")  # 输出 Area 2: 5
```

在此示例中，`width` 参数有默认值，因此在调用 `calculate_area(5)` 时，`width` 会自动取默认值 `1`。

5. 文档字符串

文档字符串（docstring）和函数注解（function annotation）是提高代码可读性和可维护性的关键工具。文档字符串用于描述模块、类或函数的功能和用法，而函数注解用于为函数的参数和返回值提供类型提示。

文档字符串是位于模块、类或函数定义内部的字符串字面量，通常使用三重引号（`"""  """`）包裹。文档字符串应简洁明了，首行应为简短的描述，后续可包含更详细的说明。

```python
def add(a, b):
    """
    返回两个数的和。

    参数:
    a (int): 第一个加数。
    b (int): 第二个加数。

    返回:
    int: 两个数的和。
    """
    return a + b
```

在上述示例中，函数 `add` 的文档字符串清晰地描述了函数的功能、参数和返回值。这有助于用户快速理解函数的用途和使用方法。

6. 函数注解（Function Annotation）

函数注解是 Python 3 引入的特性，用于为函数的参数和返回值添加元数据，通常用于类型提示。注解的语法是在参数名后使用冒号加类型提示，返回值注解则在参数列表后使用箭头加类型提示。函数注解仅用于提供信息，不会影响函数的实际行为。

```
def add(a: int, b: int) -> int:
    return a + b
```

在此示例中，函数 `add` 的参数 `a` 和 `b` 以及返回值均被注解为整数类型。这为阅读代码的人提供了关于参数和返回值类型的有用信息。

1. 形式参数与实际参数

函数的定义和调用涉及两个关键概念：形式参数（formal parameters）和实际参数（actual parameters）。形式参数是在函数定义时指定的变量，用于接收传入的数据；实际参数则是在函数调用时提供的具体值或表达式。

```
def multiply(a, b):
    return a * b

result = multiply(4, 7)
print(result)  # 输出: 28
```

在此示例中，`a` 和 `b` 是形式参数，`4` 和 `7` 是实际参数。调用 `multiply(4, 7)` 时，实际参数 `4` 和 `7` 被传递给形式参数 `a` 和 `b`，函数返回它们的乘积 `28`。

2. 默认参数（Default Arguments）

默认参数是函数定义时为参数指定的默认值。如果在调用时没有为该参数提供值，函数将使用默认值。

```
def greet(name, age=25):
    print(f"Hello, {name}! You are {age} years old.")

greet("Bob")  # 使用默认值
greet("Alice", 30)  # 使用提供的值
```

在这个例子中，`age` 参数有一个默认值 `25`。如果在调用函数时没有传入 `age` 的值，默认值 `25` 将被使用。

3. 不定长参数（Arbitrary Arguments）

当不确定传入函数的参数个数时，可以使用不定长参数。Python 提供了 `*args` 和 `**kwargs` 两种方式来处理这种情况。

- `*args` 用于传递任意数量的位置参数。
- `**kwargs` 用于传递任意数量的关键字参数。

```python
def sum_numbers(*numbers):
    total = sum(numbers)
    print(f"Sum: {total}")

sum_numbers(1, 2, 3, 4)  # 传入多个位置参数
```

在这个例子中，`*numbers` 接受了多个位置参数并将它们放入一个元组 `numbers` 中，之后通过 `sum()` 函数计算它们的和。

在 Python 编程中，变量的作用域决定了变量的可访问范围。主要分为局部变量和全局变量两种类型。

1. 局部变量（Local Variables）

局部变量是在函数内部定义的变量，其作用域仅限于该函数内部。当函数被调用时，局部变量被创建；函数执行结束后，局部变量被销毁。局部变量在函数外部无法访问。

```
def example_function():
    local_var = "I am a local variable"
    print(local_var)

example_function()
print(local_var)  # 试图在函数外部访问局部变量
```

上例中，`local_var` 是在函数 `example_function` 内部定义的局部变量。在函数内部打印该变量时，输出正常。然而，当尝试在函数外部访问 `local_var` 时，Python 抛出 `NameError`，提示未定义该变量。

2. 全局变量（Global Variables）

全局变量是在函数外部定义的变量，其作用域覆盖整个模块。全局变量可以在函数内部和外部访问。然而，在函数内部如果需要修改全局变量的值，必须使用 `global` 关键字声明，否则 Python 会将其视为新的局部变量。

```
global_var = "I am a global variable"

def example_function():
    print(global_var)  # 在函数内部访问全局变量

example_function()
print(global_var)  # 在函数外部访问全局变量
```

在上例中，`global_var` 是在函数外部定义的全局变量。在函数 `example_function` 内部和外部均可访问该变量，且输出结果一致。

3. 在函数内部修改全局变量

如果需要在函数内部修改全局变量的值，必须使用 `global` 关键字声明该变量。否则，Python 会在函数内部创建一个同名的局部变量，而不会影响全局变量的值。

```
global_var = "I am a global variable"

def example_function():
    global global_var
    global_var = "I have been modified"
    print(global_var)

example_function()
print(global_var)  # 检查全局变量是否被修改
```

上例中，使用 `global` 关键字声明 `global_var`，表示在函数内部对全局变量进行修改。因此，函数内部和外部的 `global_var` 值均被修改。

未完待续