Python中的7种基础数据类型，看这4500字就够了！

Python 语言以其简洁、高效和强大的功能，成为了无数开发者和编程爱好者的首选。无论是数据分析、人工智能、网络开发还是自动化脚本，Python 都能以其优雅的语法和丰富的库支持，让编程变得更加简单而有趣。

但正如建造一座大厦需要坚实的地基，掌握 Python 编程也需要从理解其基础数据类型开始。数据类型是编程语言中用于定义变量所存储数据种类的一种方式，它们是构建程序逻辑的基本构建块。在 Python 中，有一系列基础数据类型，它们是理解语言特性和进行高效编程的关键。

本文将带你学习 Python 的基础数据类型，从数值到布尔，从字符串到列表，再到集合和字典，我们将逐一探索这些数据类型的特性和应用。通过实际的代码示例，学会如何在 Python 中自如地使用这些数据类型，为我们的编程之路打下坚实的基础。

一 数值类型：Python编程的基石

在编程的世界里，数值是最基本的元素之一。Python 提供了两种主要的数值类型：整数（`int`）和浮点数（`float`）。让我们来详细了解一下它们。

整数（`int`）：计数的基础

整数是没有小数部分的数字，可以是正数、负数或零。在 Python 中，你可以很容易地创建一个整数：

```
age = 30    # 一个正整数
balance = -50  # 一个负整数
zero = 0  # 零也是整数
```

整数在编程中常用于计数和执行数学运算，如加法、减法、乘法和除法。

浮点数（`float`）：精确到小数点后

浮点数与整数不同，它们可以有小数部分，适用于需要更精确表示的场合：

```
price = 19.99  # 一个表示价格的浮点数
weight = 72.5  # 一个表示重量的浮点数
```

浮点数同样可以进行数学运算，但由于它们的精度限制，有时可能会遇到一些微小的误差。

数值运算：基础但强大

Python 支持对整数和浮点数进行多种数学运算，包括：

* 加法：`+`
* 减法：`-`
* 乘法：`*`
* 除法：`/`
* 取模（求余数）：`%`
* 幂运算（指数）：`**`

例如：

```
result = 10 + 5 * 2  # 结果为20，先进行乘法运算
remainder = 11 % 3   # 结果为2，11除以3的余数
power = 2 ** 3       # 结果为8，2的3次方
```

注意事项

* 在 Python 中，当你进行除法运算时，结果总是一个浮点数。即使你除以两个整数，结果也会是浮点数：

```
division = 7 / 2  # 结果为3.5，而不是整数3
```

* 如果你需要进行整数除法（忽略余数），可以使用 `//` 运算符：

```
integer_division = 7 // 2  # 结果为3，整数除法
```

* 在进行数值运算时，要注意数据类型的转换。例如，如果你将一个整数和一个浮点数相加，结果将是一个浮点数：

```
mixed_result = 5 + 3.0  # 结果为8.0，一个浮点数
```

掌握整数和浮点数的使用，是学习 Python 编程的基础。它们在变量赋值、数学运算和逻辑判断中扮演着重要角色。理解了数值类型，你就已经为进一步探索 Python 的其他数据类型打下了坚实的基础。

二 布尔类型（`bool`）：逻辑判断的守护者

在编程中，逻辑判断是控制程序流程的关键。布尔类型（`bool`）是实现这些逻辑判断的基础。布尔值只有两个：`True` 和 `False`，它们代表了逻辑上的真和假。

布尔值的用途

布尔值在 Python 中主要用于条件语句，如 `if`、`while` 和 `for` 循环，以及逻辑运算符 `and`、`or` 和 `not`。

布尔运算

布尔运算是编程中非常常见的操作，包括：

* 与运算（`and`）：两个条件都为 `True` 时，结果才为 `True`。
* 或运算（`or`）：两个条件中至少有一个为 `True` 时，结果为 `True`。
* 非运算（`not`）：将 `True` 转换为 `False`，将 `False` 转换为 `True`。

条件语句中的布尔值

布尔值在 `if` 语句中扮演着至关重要的角色：

```
x = 10
if x > 5:
    print(”x 大于 5”)
else:
    print(”x 小于或等于 5”)
```

在这个例子中，如果 `x` 的值大于5，程序将打印 ”x 大于 5”；否则，它将打印 ”x 小于或等于 5”。

布尔值与其他操作

布尔值也常用于比较操作符的结果：

* 大于：`>`
* 小于：`<`
* 等于：`==`
* 不等于：`!=`
* 大于等于：`>=`
* 小于等于：`<=`

例如：

```
y = 20
if y == 20:
    print(”y 等于 20”)

```

注意事项

* 在 Python 中，布尔值是大写的 `True` 和 `False`，不要使用小写的 `true` 或 `false`。
* 除了布尔值，Python 中的其他值也可以在布尔上下文中被解释为 `True` 或 `False`。`None`、所有的数值零（包括 `0`、`0.0`、`0j`）、空字符串 `””“`、空列表 `[]` 和空字典 `{}` 都被解释为 `False`，其他值都被解释为 `True`。

布尔类型是编程中实现逻辑判断和控制程序流程的基础。理解布尔值和布尔运算对于编写条件逻辑和循环结构至关重要。掌握了布尔类型，你将能够更加灵活地构建程序的逻辑，使其能够根据条件做出决策。

三 字符串类型（`str`）：文本的载体

字符串是编程中用于表示文本的数据类型。在 Python 中，字符串可以用单引号（`’`）或双引号（`”`）括起来，这使得它在处理文本信息时非常灵活。

创建字符串

你可以这样创建一个字符串：

```
greeting = ”你好，世界！”
message = ’Python 是一种强大的编程语言。’

```

无论是单引号还是双引号，它们的作用都是一样的，但你可以使用一种引号来创建一个包含另一种引号的字符串：

```
example = ”她说：’你好！’”
```

字符串的特点

* **不可变性**：字符串一旦创建就不能改变。如果你需要修改字符串，Python 会创建一个新的字符串对象。
* **序列**：字符串是由字符组成的序列，可以通过索引访问每一个字符。

字符串操作

字符串支持多种操作，包括：

* **连接**：使用 `+` 符号连接字符串。
* **复制**：使用 `*` 符号进行字符串的复制。
* **切片**：通过指定索引范围获取字符串的一部分。
* **格式化**：将变量插入到字符串中。

示例

```
# 连接字符串
concatenated = ”Hello, ” + ”World!”

# 复制字符串
repeated = ”Python ” * 3

# 切片操作
first_five = ”Hello”[0:5] # 前五个字符

# 字符串格式化
name = ”Kitty”
greeting = ”Hello, ” + name
```

字符串方法

Python 的字符串类型提供了许多有用的方法，例如：

* `.upper()`：将字符串转换为大写。
* `.lower()`：将字符串转换为小写。
* `.strip()`：移除字符串两端的空白字符。
* `.split()`：按指定分隔符分割字符串。
* `.join()`：将序列中的元素连接成一个字符串。

注意事项

* 字符串的索引是从 `0` 开始的，这意味着第一个字符的索引是 `0`。
* 尝试修改字符串的某个字符会引发错误，因为字符串是不可变的。
* 当处理字符串时，要注意转义字符，如 `\n` 表示换行，`\t` 表示制表符。

字符串是编程中处理文本数据的核心。掌握字符串的创建、操作和格式化，对于处理文本信息、构建用户界面和实现文本处理功能至关重要。理解字符串的不可变性和序列特性，将帮助我们更有效地使用 Python 进行文本操作。

四 列表类型（`list`）：灵活的序列容器

列表是 Python 中一种非常灵活的数据结构，用于存储有序的元素集合。它允许元素的增加、删除和排序，这使得列表在处理一系列数据时非常有用。

创建列表

创建列表非常简单，使用方括号 `[]` 即可：

```
fruits = ['apple', 'banana', 'cherry']
numbers = [1, 2, 3, 4, 5]
```

列表的特点

* **可变性**：列表的内容可以被修改，这意味着你可以添加、删除或更改列表中的元素。
* **异构性**：列表可以包含不同类型的元素，但通常最佳实践是保持列表元素的一致性。

列表操作

列表支持多种操作，包括：

* **添加元素**：使用 `.append()` 或 `.extend()` 方法。
* **删除元素**：使用 `.pop()` 或 `.remove()` 方法。
* **排序**：使用 `.sort()` 或 `.sorted()` 方法。
* **切片**：通过指定索引范围获取列表的一部分。

示例

```
# 添加元素
fruits.append('orange')

# 删除元素
del fruits[1] # 删除指定位置的元素

# 排序列表
numbers.sort()

# 切片操作
first_two_fruit = fruits[:2] # 获取前两个水果
```

列表推导式

Python 提供了一种简洁的构建列表的方法，称为列表推导式：

```
squares = [x**2 for x in range(10)]
```

注意事项

* 列表的索引同样是从 `0` 开始的。
* 在使用 `.pop()` 方法时，如果不指定索引，默认删除并返回列表中的最后一个元素。
* `.append()` 向列表末尾添加一个元素，而 `.extend()` 可以一次性添加多个元素。

列表是 Python 中用于存储序列数据的主力军。它们不仅可以用来存储数据，还可以通过各种方法进行数据操作和管理。掌握列表的使用，对于进行有效的数据处理和程序设计至关重要。

五 元组类型（`tuple`）：不可变的序列

元组是 Python 中的一种数据结构，它与列表类似，但元组一旦创建就不能被修改，这使得元组在某些需要数据不可变的场景下非常有用。

创建元组

创建元组非常简单，使用圆括号 `()` 并用逗号分隔元素：

```
coordinates = (4.0, 5.0)
colors = ’red’, ’green’, ’blue’  # 圆括号可以省略
```

元组的特点

* **不可变性**：元组一旦创建，就不能添加、删除或更改元素。
* **性能**：由于元组的不可变性，它通常比列表有更好的性能，特别是在作为字典的键时。

元组操作

虽然元组不可变，但你仍然可以：

* **访问元素**：使用索引来获取元组中的元素。
* **切片**：获取元组的一部分。
* **遍历**：使用循环遍历元组中的所有元素。

示例

```
# 访问元素
x, y = coordinates
print(x)  # 输出: 4.0

# 切片操作
first_two_colors = colors[:2]  # 获取前两个颜色

# 遍历元组
for color in colors:
    print(color)
```

元组与列表的转换

你可以将列表转换为元组，反之亦然：

```
# 将列表转换为元组
list_to_tuple = tuple([1, 2, 3])

# 将元组转换为列表
tuple_to_list = list((1, 2, 3))
```

注意事项

* 元组默认是不可变的，但元组内的可变类型对象（如列表）可以被修改。
* 单个元素的元组在定义时需要在元素后面加上逗号，例如 `(single_element,)`。

元组以其不可变性在 Python 中扮演着特殊的角色，特别是在需要确保数据不被更改的情况下。理解元组的使用，对于编写安全、高效的程序代码非常重要。

六 集合类型（`set`）：独特的无序集合

在 Python 中，集合（`set`）是一种无序且不包含重复元素的容器。它类似于数学中的集合概念，支持多种集合运算，如并集、交集和差集。

创建集合

创建一个集合非常简单，使用花括号 `{}` 或 `set()` 函数：

```
odd_numbers = {1, 2, 3, 3}  # 注意：集合中不会有重复的3
prime_numbers = set((2, 3, 5, 7, 11))
```

集合的特点

* **无序性**：集合中的元素没有特定的顺序。
* **不重复性**：集合会自动去除重复的元素。

集合操作

集合支持的基本操作包括：

* **添加元素**：使用 `.add()` 方法。
* **移除元素**：使用 `.remove()` 或 `discard()` 方法，后者在元素不存在时不会引发错误。
* **集合运算**：包括并集（`union`）、交集（`intersection`）、差集（`difference`）和对称差集（`symmetric_difference`）。

示例

```
# 添加元素
odd_numbers.add(4)

# 移除元素
odd_numbers.discard(2) # 即使2不在集合中，也不会报错

# 集合运算
even_numbers = {2, 4, 6}
union = odd_numbers.union(even_numbers) # 并集
intersection = odd_numbers.intersection(even_numbers) # 交集
difference = odd_numbers.difference(even_numbers) # 差集
```

注意事项

* 集合中的元素必须是不可变类型，因为集合需要能够明确地比较元素是否相同。
* 集合的元素不按特定的顺序排列，所以不应该依赖元素的顺序。

集合类型在处理唯一性数据和执行集合运算时非常有用。掌握集合的创建和操作，可以帮助我们更高效地处理数据集合，特别是在需要执行数学意义上的集合运算时。

七 字典类型（`dict`）：键值对的集合

字典（`dict`）是 Python 中一种非常有用的数据结构，它存储了键值对（key-value pairs），其中键（key）是唯一的，而值（value）可以是任何数据类型。

创建字典

创建字典使用花括号 `{}`，并用冒号 `:` 分隔键和值：

```
person = {'name': 'Alice', 'age': 25, 'city': 'New York'}
```

字典的特点

* **通过键访问**：字典中的每个键都是唯一的，可以通过键来快速访问对应的值。
* **可变性**：字典的内容可以被修改，可以添加新的键值对，也可以更改或删除已有的键值对。

字典操作

字典支持多种操作，包括：

* **添加键值对**：直接指定新的键和值。
* **修改键值对**：通过键来修改对应的值。
* **删除键值对**：使用 `del` 或 `pop()` 方法。
* **遍历**：通过循环遍历字典中的所有键值对。

示例

```
# 添加键值对
person['email'] = 'alice@example.com'

# 修改键值对
person['age'] = 26

# 删除键值对
del person['city']
email = person.pop('email')  # 返回并删除键'email'对应的值

# 遍历字典
for name, detail in person.items():
    print(f"{name}: {detail}")
```

注意事项

* 键必须是不可变类型，通常是字符串或数字。
* 键是区分大小写的，因此 `'Name'` 和 `'name'` 会被视为两个不同的键。
* 尝试访问不存在的键会导致错误，使用 `get()` 方法可以避免这个错误，它在键不存在时返回 `None` 或指定的默认值。

字典类型以其通过键访问值的特性，在数据存储和检索方面提供了极大的灵活性。掌握字典的使用，对于处理复杂的数据结构和实现高效的数据管理非常重要。

综上，我们一起探索了 Python 中的 7 种基础数据类型：数值、布尔值、字符串、列表、元组、集合和字典。下回见 ~

原文链接: https://juejin.cn/post/7388316163577741324