1 Python 核心数据类型详解

Python 的核心数据类型，它们是 Python 编程的基石，也是我们构建各种应用程序的基础。熟练掌握这些数据类型，将为我们后续的编程学习和实践打下坚实的基础。

1.1 数字类型（Numbers）

数字类型是 Python 中最基本的内置数据类型之一。它用于存储各种数值，包括整数、浮点数和复数。

1.1.1 整数（int）

整数是具有整数值的数字，没有小数部分。在 Python 中，整数可以是正整数、负整数或零，例如 1、-10、0。整数的精度不受限制，理论上可以存储任意大的整数值，这使得 Python 在处理大整数计算时具有很大的优势。我们可以对整数进行各种算术运算，如加（+）、减（-）、乘（*）、除（/）等，运算结果仍然是一个整数或浮点数（当除法结果不是整数时）。

1.1.2 浮点数（float）

浮点数用于表示带有小数部分的数字，例如 3.14、-0.5、2.0。浮点数在计算机中是按照一定的精度来存储的，通常遵循 IEEE 754 标准。在 Python 中，当我们进行浮点数运算时，可能会出现一些精度问题，这是因为浮点数在内存中的表示方式所导致的。例如，0.1 + 0.2 的结果可能并不是精确的 0.3，而是一个接近 0.3 的浮点数。但在大多数实际应用场景中，这种精度误差是可以接受的。

1.1.3 复数（complex）

复数是由实数部分和虚数部分组成的数，例如 3 + 4j。在 Python 中，虚数的后缀可以用“j”或者“J”来表示。复数类型主要用于科学计算和工程领域中的某些特定应用，如信号处理、量子力学等。

1.2 字符串类型（String）

字符串是由一个或多个字符组成的有序序列，是文本数据的基本表示形式。在 Python 中，字符串被定义为由单引号（’）、双引号（“）或三引号（’’’或”””）括起来的字符序列。

1.2.1 创建字符串

- 单引号和双引号在创建字符串时可以互换使用，但如果字符串中包含单引号或双引号字符，就需要使用另一种引号来避免冲突。例如，‘I’m a Python learner.’是错误的，而“I’m a Python learner.”或‘I”m a Python learner.’则是正确的。
- 三引号可以创建多行字符串，这对于包含多行文本的内容非常方便，例如文档字符串或长文本的注释。

1.2.2 字符串索引和切片

- 字符串中的每个字符都有一个索引，索引从 0 开始，依次递增。例如，对于字符串“Hello”，索引 0 对应字符“H”，索引 1 对应“e”，依此类推。

- 我们可以通过索引访问字符串中的特定字符，例如 s[0] 表示获取字符串 s 的第一个字符。同时，Python 还支持负索引，-1 表示字符串的最后一个字符，-2 表示倒数第二个字符，以此类推。
- 切片操作可以获取字符串中的一部分字符。基本语法是 s[start:end:step]，其中 start 表示起始索引（包含），end 表示结束索引（不包含），step 表示步长。例如，s[1:4] 表示获取字符串 s 中从索引 1 到索引 3 的字符。

1.2.3 字符串的常用操作和方法

- 字符串拼接：使用加号（+）可以将两个或多个字符串拼接在一起。例如，“Hello”+ ””+ “World”结果为“Hello World”。
- 字符串重复：使用乘号（）可以使字符串重复指定的次数。例如，“*Hi*” 3 结果为“HiHiHi”。
- 字符串的长度：使用 len() 函数可以获取字符串中字符的个数，包括空格和其他特殊字符。
- 字符串的查找：使用 in 关键字可以判断一个子字符串是否存在于另一个字符串中。例如，“app”in “application”结果为 True。还可以使用字符串的方法如 find()、index() 等来查找子字符串的位置。
- 字符串的替换：replace() 方法可以将字符串中的某些字符替换为其他字符。例如，“Hello World”.replace(“World”, “Python”) 结果为“Hello Python”。
- 字符串的分割：split() 方法可以按照指定的分隔符将字符串分割成多个子字符串，并返回一个列表。例如，“a,b,c”.split(“,”) 结果为 [“a”, “b”, “c”]。
- 字符串的大小写转换：upper() 方法将字符串中的所有字符转换为大写，lower() 方法转换为小写，capitalize() 方法将字符串的首字母转换为大写，其余字母转换为小写等。

1.3 列表类型（List）

列表是一种有序的、可变的数据类型，它可以包含不同类型的数据元素。列表中的元素用方括号（[ ]）括起来，元素之间用逗号分隔。

1.3.1 创建列表

- 可以直接使用方括号创建列表，例如 [1, 2, 3]、[‘apple’, ‘banana’, ‘orange’]、[True, False, None] 等。列表中的元素可以是不同数据类型，如整数、字符串、布尔值、其他列表等的混合。

1.3.2 列表的索引和切片

- 与字符串类似，列表也支持索引和切片操作。索引从 0 开始，可以通过索引访问列表中的特定元素，例如 list[0] 获取列表的第一个元素。同样，也支持负索引和切片操作，如 list[-1] 获取最后一个元素，list[1:4] 获取从索引 1 到索引 3 的元素。

1.3.3 列表的常用操作和方法

- 元素添加：使用 append() 方法可以在列表的末尾添加一个元素。例如，list.append(“new_element”)。使用 insert() 方法可以在指定位置插入一个元素，例如 list.insert(index, element)。
- 元素删除：使用 remove() 方法可以删除列表中第一个匹配的元素。如果要删除指定索引位置的元素，可以使用 pop(index) 方法，如果不指定索引，默认删除最后一个元素。
- 列表拼接：使用加号（+）可以将两个列表拼接在一起，形成一个新的列表。例如，list1 + list2。

- 列表重复：使用乘号（）可以重复列表中的元素。例如，*list* 3。
- 列表排序：sort() 方法可以对列表中的元素进行排序，可以指定排序的关键字参数，如 reverse=True 表示降序排序。
- 列表反转：reverse() 方法可以将列表中的元素顺序反转。
- 列表的长度、最大值、最小值：使用 len()、max()、min() 函数可以获取列表的长度、最大元素值和最小元素值等。

1.4 元组类型（Tuple）

元组与列表类似，也是一种有序的数据类型，但它与列表的一个重要区别是元组是不可变的。一旦元组被创建，它的元素就不能被修改。元组中的元素用圆括号（( )）括起来，元素之间用逗号分隔。

1.4.1 创建元组

- 可以直接使用圆括号创建元组，例如 (1, 2, 3)、(‘a’, ‘b’, ‘c’)、(True, False) 等。如果元组中只有一个元素，需要在元素后面加上逗号，例如 (5,)，否则 Python 会将其视为一个普通的括号表达式。

1.4.2 元组的索引和切片

- 元组同样支持索引和切片操作，与列表的用法一致。通过索引可以访问元组中的特定元素，例如 tuple[0]。切片操作可以获取元组中的一部分元素。

1.4.3 元组的常用操作和方法

- 由于元组的不可变性，它的操作方法相对较少。可以使用 len() 函数获取元组的长度，使用 max()、min() 函数获取最大值和最小值等。还可以通过索引和切片获取元组中的元素，但不能对元组中的元素进行修改，如添加、删除或修改元素的值。

1.4.4 元组的优势

- 元组的不可变性使得它在某些场景下具有优势。例如，元组可以作为字典的键，而列表不能。因为字典的键需要是不可变的类型。此外，元组在存储数据时通常比列表更节省内存，而且在某些操作中，元组的执行速度可能比列表更快。

1.5 字典类型（Dictionary）

字典是一种无序的、可变的数据类型，用于存储键 - 值对（key-value pairs）。每个键值对之间用逗号分隔，键和值之间用冒号分隔。字典中的键必须是唯一的且是不可变类型（如数字、字符串、元组等），而值可以是任意数据类型。

1.5.1 创建字典

- 可以使用花括号（{ }）创建字典，例如 {‘name’: ‘John’, ‘age’: 30, ‘city’: ‘New York’}。其中，“name”、“age”、“city”是键，对应的“John”、30、“New York”是值。

1.5.2 访问字典中的值

- 通过键来访问字典中的值，例如 dict['name'] 可以获取对应键的值。如果访问一个不存在的键，会抛出 KeyError 异常。为了安全地访问字典中的值，可以使用 get() 方法，例如 dict.get('name')，如果键不存在，get() 方法会返回 None 或者指定的默认值。

1.5.3 字典的常用操作和方法

- 添加和修改键值对：可以直接通过键来添加新的键值对或者修改现有键对应的值。例如，dict['new_key'] = 'new_value'添加新的键值对，dict['existing_key'] = 'new_value'修改现有键的值。
- 删除键值对：可以使用 del 语句删除指定的键值对，例如 del dict['key']。还可以使用 pop() 方法删除指定键的键值对，并返回被删除的值，例如 dict.pop('key')。另外，clear() 方法可以清空整个字典。
- 字典的遍历：可以通过 for 循环遍历字典中的键、值或键值对。例如，for key in dict 遍历键，for value in dict.values() 遍历值，for key, value in dict.items() 遍历键值对。
- 字典的合并：可以使用 update() 方法将一个字典中的键值对添加到另一个字典中，如果存在相同的键，会覆盖原来的值。

1.6 集合类型（Set）

集合是一种无序的、可变的数据类型，它包含一组唯一的元素，即集合中的元素不能重复。集合通常用于成员检测、去除重复元素、数学运算（如交集、并集、差集等）等场景。

1.6.1 创建集合

- 可以使用花括号（{ }）创建集合，例如 {1, 2, 3}。需要注意的是，如果要创建一个空集合，必须使用 set() 函数，因为 {} 会被解释为空字典。

1.6.2 集合的常用操作和方法

- 元素添加：使用 add() 方法可以向集合中添加一个元素。例如，set.add(element)。
- 元素删除：可以使用 remove() 方法删除集合中指定的元素，如果元素不存在，会抛出 KeyError 异常。使用 discard() 方法删除元素时，如果元素不存在，不会抛出异常。
- 集合运算：可以使用运算符或方法来进行集合之间的运算。例如，集合的交集运算可以用 & 或 intersection() 方法，例如 set1 & set2 或 set1.intersection(set2)。并集运算可以用 | 或 union() 方法，差集运算可以用 - 或 difference() 方法等。

1.6.3 集合的优势

- 集合的元素唯一性使得它在去除重复元素方面非常有用。同时，集合的成员检测操作通常比列表更快，因为它在内部使用哈希表来存储元素。

1.7 元组与列表的异同比较

1.7.1 相同点：

- **有序性**：列表和元组都是有序的，元素有固定的顺序，可以通过索引访问特定位置的元素，索引从 0 开始，也支持负索引。
- **支持多种数据类型**：都可以包含不同数据类型（如整数、字符串、浮点数等）的元素，并且允许混合存储不同类型的数据。
- **切片操作**：都支持切片操作，可以获取其中的一部分元素，语法相同，例如 list[start:end] 或 tuple[start:end]。

1.7.2 不同点：

对比维度	列表	元组
可变性	可变，可以对元素进行添加、修改、删除等操作，例如使用 append()、insert()、remove() 等方法。	不可变，一旦创建，元组中的元素不能被修改，也不能添加或删除元素。
内存占用	通常比元组占用更多的内存。	通常比列表占用更少的内存，因为其不可变性使得内存分配更简单和紧凑。
性能	对于包含大量元素的列表，某些操作（如添加或删除元素）可能比元组慢，尤其是在列表的开头或中间位置进行插入或删除操作时，因为需要移动大量元素。	在某些操作中，元组的执行速度可能比列表更快，因为其结构相对简单，且没有额外的可变性开销。
操作方法	提供更多操作方法，如 sort()、reverse()、pop() 等。	提供的操作方法相对较少，主要用于访问元素。
用法	适用于需要频繁修改元素的场景，例如存储动态变化的数据集合。	适用于存储固定不变的数据，确保数据的完整性，如字典的键、存储常量数据等。

通过以上对 Python 核心数据类型的详细介绍，我们了解到每种数据类型都有其独特的特点、用途和操作方法。在实际编程中，我们需要根据具体的需求和场景选择合适的数据类型来存储和处理数据。希望大家能够熟练掌握这些核心数据类型的使用，为今后的 Python 编程之路奠定坚实的基础。