2023/02/25 开始重新学习python

[二维数组的创建](# 二维数组的创建)

[函数](# 函数)

[string](# string)

[for loop](# for loop)

[self](# self)

[init](# init)

[一维数组创建](# 一维数组)

heap

二维数组的创建

在 Python 中，初始化一个二维数组可以使用以下两种方法：

方法一：使用嵌套列表

使用嵌套列表可以方便地创建一个二维数组，如下所示：

```
n = 3  # 定义数组的行数
m = 4  # 定义数组的列数
arr = [[0] * m for _ in range(n)]  # 创建一个 n 行 m 列的二维数组
```

这个代码片段创建了一个 n 行 m 列的二维数组，并将其中的每个元素都初始化为 0。其中 _ 是一个特殊的变量名，表示我们不关心这个循环变量的值。

方法二：使用 NumPy 库

如果需要使用大规模的二维数组或进行高性能的数组运算，可以使用 NumPy 库。创建一个二维数组可以使用以下代码：

```
import numpy as np

n = 3  # 定义数组的行数
m = 4  # 定义数组的列数
arr = np.zeros((n, m))  # 创建一个 n 行 m 列的二维数组
```

这个代码片段创建了一个 n 行 m 列的二维数组，并将其中的每个元素都初始化为 0。可以使用 np.ones 函数将数组初始化为 1，或者使用 np.random.rand 函数将数组初始化为随机数。

函数

Python中定义函数的一般格式为：

```
def function_name(parameters):
    """Docstring"""
    # Function body
    return expression
```

其中，`function_name` 是函数的名称，`parameters` 是函数的参数，用于接收调用函数时传递的数据。`Docstring` 是对函数功能的描述，通常使用三引号 `"""` 包裹字符串。

定义函数后，可以通过函数名进行调用，例如：

```
result = function_name(argument1, argument2)
```

其中，`argument1` 和 `argument2` 是传递给函数的实际参数。

函数的参数可以分为以下几类：

1. 位置参数（positional arguments）：按照位置顺序传递的参数，例如 `def function(x, y, z)` 中的 `x`、`y` 和 `z` 就是位置参数。
2. 关键字参数（keyword arguments）：使用参数名和值进行传递的参数，例如 `function(y=2, x=1, z=3)` 中的 `y=2`、`x=1` 和 `z=3` 就是关键字参数。
3. 默认参数（default arguments）：在函数定义时指定的默认值，如果调用函数时没有传递该参数，则使用默认值。例如 `def function(x=0, y=0)` 中的 `x=0` 和 `y=0` 就是默认参数。
4. 可变参数（variable-length arguments）：可以接受任意个数的参数，例如 `def function(*args)` 中的 `*args` 表示可以接受任意个数的位置参数。
5. 关键字可变参数（keyword variable-length arguments）：可以接受任意个数的关键字参数，例如 `def function(**kwargs)` 中的 `**kwargs` 表示可以接受任意个数的关键字参数。

函数的参数可以按照顺序依次传递，也可以使用关键字进行指定。例如：

```
def function(x, y, z):
    print(x, y, z)

function(1, 2, 3) # 按照顺序传递参数，输出 1 2 3
function(z=3, y=2, x=1) # 使用关键字传递参数，输出 1 2 3
```

在函数定义时，可以将默认参数设置为特定的值。例如：

```
def function(x=0, y=0):
    print(x, y)
```

```
function() # 输出 0 0
function(x=1) # 输出 1 0
```

可变参数和关键字可变参数使用时需要注意参数的格式。例如：

```
def function(*args):
    print(args)

function(1, 2, 3) # 输出 (1, 2, 3)

def function(**kwargs):
    print(kwargs)

function(x=1, y=2, z=3) # 输出 {'x': 1, 'y': 2, 'z': 3}
```

最后需要注意，函数的参数可以在函数内部进行修改，但是不会影响函数外部的变量。如果需要在函数内部修改外部变量，需要使用 `

```
class Solution:
    def longestCommonSubsequence(self, text1: str, text2: str) -> int:
```

这是 Python 中定义类和函数的语法。`class Solution:` 定义了一个名为 `Solution` 的类，`def longestCommonSubsequence(self, text1: str, text2: str) -> int:` 定义了一个名为 `longestCommonSubsequence` 的方法，该方法有三个参数：`self`，`text1` 和 `text2`，其中 `self` 表示该方法是一个对象的方法，而 `text1` 和 `text2` 则是该方法所需要的输入参数。`: str` 和 `: int` 表示这两个参数的类型，分别为字符串类型和整数类型。`-> int` 表示该方法的返回值类型为整数类型。

```
def LCS(self,text1,text2) -> int:
```

注意这种 -> int只存在python 3 刷题的时候记得调一下

string

paramater中 text1: str

text1[i] 访问下标

ord（text1【i】）返回i位置的ascii值

for loop

```
        for i in range (m-2,-1,-1): # 从m-2开始，不包括-1，每个i --（一定要写这个）
            for j in range (i+1,m):# 从i+1开始，不包括m
```

self

```
result = 0
    def findTargetSumWays(self, nums: List[int], target: int) -> int:
        # revise backtrack
        self.backtrack(nums, 0,target)
        return self.result

    def backtrack(self,nums:List[int], i : int, remain: int):
      # result 和 backtrack前面都要加上self.
```

init

```
    def __init__(self):
        self.used = None
        self.track = None
        self.res = None

    def permute(self, nums: List[int]) -> List[List[int]]:
        #initialize variables
        self.track = []
        self.res = []
        self.used = [False] * len(nums)
```

一维数组

```
diff = [0 for i in range(length)]
```