**TQC 學習筆記**

優先答題:1,2,3,4,5 ,8

閏年判斷:常備考題

**python 中 eval 的高級用法**

Python 中的 eval 函數是一種強大的工具，但它也存在一些安全風險。如果您不謹慎使用，可能會導致代碼運行錯誤、安全性漏洞甚至被惡意程式碼利用。因此，在使用 eval 之前，您需要充分瞭解它的相關知識，並採取必要的安全措施。

以下是一些 eval 的高級用法：

**1. 動態執行代碼**

eval 可以用來動態執行字串表示的 Python 代碼。這在某些情況下非常有用，例如：

* 根據使用者輸入生成代碼
* 從設定檔中載入代碼
* 在運行時修改代碼

例如，我們可以使用 eval 來根據使用者輸入生成一個計算平方根的函數：

def eval\_code(user\_input):

try:

# 使用 eval 執行用戶輸入的代碼

result = eval(user\_input)

print(result)

except Exception as e:

print(f"Error: {e}")

user\_input = input("請輸入一個計算平方根的運算式：")

eval\_code(user\_input)

**2. 訪問私有成員**

eval 可以用來訪問 Python 物件的私有成員。這在某些情況下非常有用，例如：

* 調試代碼
* 反射代碼

例如，我們可以使用 eval 來訪問 int 類的私有成員 \_bit\_length：

class MyInt(int):

pass

number = MyInt(10)

# 使用 eval 訪問私有成員

bit\_length = eval("\_bit\_length(number)")

print(bit\_length) # 輸出: 4

**3. 創建自訂類型**

eval 可以用來創建自訂類型。這在某些情況下非常有用，例如：

* 擴展 Python 的類型系統
* 創建具有特定行為的類型

例如，我們可以使用 eval 來創建一個表示複數的類型：

class Complex:

def \_\_init\_\_(self, real, imag):

self.real = real

self.imag = imag

def \_\_str\_\_(self):

return f"{self.real} + {self.imag}j"

def eval\_complex(complex\_str):

# 使用 eval 解析複數字串

real, imag = complex\_str.split("+")

real = float(real)

imag = float(imag.strip("j"))

return Complex(real, imag)

complex\_str = input("請輸入一個複數：")

complex\_object = eval\_complex(complex\_str)

print(complex\_object) # 輸出: 3.14 + 2.718j

**安全風險**

eval 的主要安全風險在於它允許執行任何 Python 代碼。這意味著，如果您不謹慎使用 eval，可能會導致以下問題：

* **代碼運行錯誤：** 如果使用者輸入或設定檔中的代碼包含語法錯誤或邏輯錯誤，則可能會導致代碼運行錯誤。
* **安全性漏洞：** 如果用戶輸入或設定檔中的代碼是惡意的，則可能會導致安全性漏洞，例如遠端代碼執行（RCE）漏洞。
* **代碼維護性差：** 使用 eval 會使代碼更加難以理解和維護，因為您需要跟蹤哪些代碼是動態執行的。

eval() 函數在 Python 中用於計算一個字串並執行其中的運算式。它主要有以下幾種高級用法:

1. **動態執行代碼**

eval() 可以用來動態執行字串形式的代碼,這在某些特殊場景下很有用,比如根據使用者輸入構建代碼字串並執行。但是要非常小心,因為直接執行未經驗證的代碼存在潛在的安全隱患。

1. **動態導入模組**

通過字串的形式指定模組名,可以用 eval() 動態導入模組。例如:

module\_name = "math"

math = eval(f"\_\_import\_\_({module\_name})")

1. **動態創建物件**

利用類的字串名稱,可以動態產生實體物件,例如:

class\_name = "MyClass"

my\_obj = eval(f"{class\_name}()")

1. **動態執行函數**

通過字串形式的函數名和參數,可以動態執行函數:

func\_name = "print"

args = ("Hello, World!",)

eval(f"{func\_name}(\*{args})")

1. **數學運算**

eval() 可以執行字串形式的算術運算式:

result = eval("1 + 2 \* 3")

print(result) *# 輸出 7*

1. **有效載入 JSON 資料**

eval() 可以用於從字串載入 JSON 資料,但要注意只在可信的資料來源上使用,避免潛在的安全風險。

json\_data = '{"name": "Alice", "age": 30}'

data = eval(json\_data)

print(data) *# {'name': 'Alice', 'age': 30}*

總的來說,eval() 是一個非常強大但也非常危險的函數,需要格外小心使用,避免執行未經驗證的代碼。

# Python 字串 format() 使用方法及示例

字串format()方法格式化指定的值，並將其插入字串的預留位置內。  
預留位置使用大括弧 {} 定義。請在下面的“預留位置”部分中瞭解有關預留位置的更多資訊。  
format() 方法返回格式化的字串。

format()方法的語法為：

template.format(p0, p1, ..., k0=v0, k1=v1, ...)

在這裡，p0, p1,...分別是位置參數和k0, k1,...具有值的關鍵字參數v0, v1,...。

並且，template是格式代碼與參數預留位置的混合。

## 字串format()參數

format()方法採用任意數量的參數。但是，分為兩種類型的參數：

* **位置參數** -可以使用花括弧{index}中的參數索引訪問的參數清單
* **關鍵字參數** -鍵=數值型別的參數列表，可以使用花括弧{key}中的參數鍵來訪問

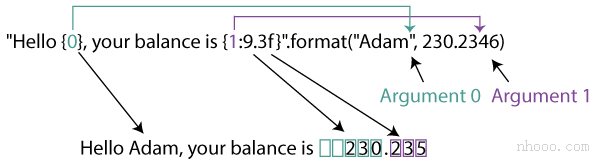
## 字串format()返回值

format()方法返回格式化的字串。

## 字串format()如何工作？

format()讀取傳遞給它的參數的類型，並根據字串中定義的格式代碼對其進行格式化。

### 對於位置參數



在這裡，參數0是字串“ Adam”，參數1是浮點數230.2346。

**注意：**在Python中，參數列表從0開始。

該字串"Hello {0}, your balance is {1:9.3f}"是範本字串。這包含用於格式化的格式代碼。

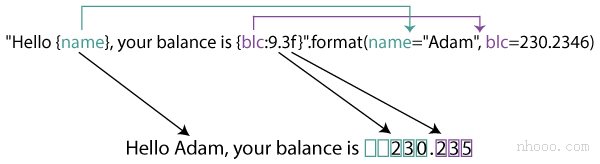
花括弧只是要放置的參數的預留位置。在上面的實例中，{0}是“Adam”的預留位置，{1:9.3f}是230.2346的預留位置。

由於範本字串引用format()參數如{0}和{1}，因此這些參數是位置參數。它們也可以在沒有數位的情況下被引用，因為{}和Python在內部將它們轉換為數位。

在內部，

* 由於"Adam"是第0 個參數，因此將其放在{0}的位置。由於{0}不包含任何其他格式代碼，因此它不執行任何其他操作。
* 然而，第一個參數230.2346不是這樣的。在這裡，{1:9.3f}將230.2346放在它的位置上，並執行9.3f操作。
* f指定格式正在處理浮點數。如果未正確指定，它將發出錯誤。
* “.”之前的部分。（9）指定數位（230.2346）可以採用的最小寬度/填充。在這種情況下，230.2346至少被分配9個位置，包括“.”。  
  如果未指定對齊選項，則將其對齊到其餘空格的右側。（對於字串，它向左對齊。）
* “.”之後的部分。（3）將小數部分（2346）截斷為給定的數字。在這種情況下，3346後將截斷2346。  
  剩餘數位（46）會四捨五入，輸出235。

### 對於關鍵字參數



我們從上面使用了相同的示例來展示關鍵字和位置參數之間的區別。

在這裡，我們不僅使用參數，還使用了鍵值作為參數。即name="Adam"和blc=230.2346。

由於這些參數是由它們的鍵作為{name}和{blc:9.3f}引用的，因此它們被稱為關鍵字或具名引數。

在內部，

* 預留位置{name}替換為name的值-“ Adam”。 由於它不包含任何其他格式代碼，因此放置了“ Adam”。
* 對於參數blc = 230.2346，預留位置{blc:9.3f}將替換為值230.2346。但是在替換它之前，像前面的示例一樣，它對其執行9.3f操作。  
  輸出230.235。小數部分在3位元後被截斷，其餘數字四捨五入。同樣，總寬度分配為9，在左側留兩個空格。

## 使用format()的基本格式

format()方法允許使用簡單的預留位置進行格式化。

### 示例1：默認，位置和關鍵字參數的基本格式

示例

# 默認參數

print("Hello {}, your balance is {}.".format("Adam", 230.2346))

# 位置參數

print("Hello {0}, your balance is {1}.".format("Adam", 230.2346))

# 關鍵字參數

print("Hello {name}, your balance is {blc}.".format(name="Adam", blc=230.2346))

# 混合參數

print("Hello {0}, your balance is {blc}.".format("Adam", blc=230.2346))

運行該程式時，所有輸出將相同：

Hello Adam, your balance is 230.2346.

Hello Adam, your balance is 230.2346.

Hello Adam, your balance is 230.2346.

Hello Adam, your balance is 230.2346.

**注意：**對於混合參數，關鍵字參數必須始終跟隨位置參數。

## 使用format()格式化數位

您可以使用下面給出的格式說明符設置數位格式：

| 數位格式類型 | |
| --- | --- |
| **類型** | **含義** |
| d | 小數整數 |
| c | 對應的Unicode字元 |
| b | 二進位格式 |
| o | 八進制格式 |
| x | 十六進位格式（小寫） |
| X | 十六進位格式（大寫） |
| n | 與“ d”相同。除了使用數位分隔符號的當前語言環境設置 |
| e | 指數標記法。（小寫字母e） |
| E | 指數符號（大寫E） |
| f | 顯示定點編號（預設值：6） |
| F | 與“ f”相同。除了將“ inf”顯示為“ INF”和“ nan”顯示為“ NAN” |
| g | 通用格式。將數字四捨五入為p個有效數字。（默認精度：6） |
| G | 與“ g”相同。如果數量很大，則除外切換到“ E”。 |
| % | 百分比。乘以100並以％結尾。 |

### 示例2：簡單的數字格式

示例

# 整數參數

print("數字:{:d}".format(123))

# 浮點參數

print("浮點數:{:f}".format(123.4567898))

# 八進制，二進位和十六進位格式

print("bin: {0:b}, oct: {0:o}, hex: {0:x}".format(12))

當您運行該程式時，輸出將是：

數字: 123

浮點數:123.456790

bin: 1100, oct: 14, hex: c

### 示例3：用int和float填充數位格式

示例

# 最小寬度的整數

print("{:5d}".format(12))

# 寬度對於比填充更長的數字不起作用

print("{:2d}".format(1234))

# 浮點數的填充

print("{:8.3f}".format(12.2346))

# 最小寬度為0的整數

print("{:05d}".format(12))

# 填充用0填充的浮點數

print("{:08.3f}".format(12.2346))

當您運行該程式時，輸出將是：

   121234  12.235000120012.235

這裡，

* 在第一個語句中，{:5d}採用整數參數並指定最小寬度5。由於未指定對齊方式，因此它向右對齊。
* 在第二個語句中，您可以看到寬度（2）小於數位（1234），因此它不需要在左邊留任何空格，也不會截斷該數字。
* 與整數不同，浮點數具有整數和小數部分。 並且，定義為該數位的最小寬度是整個包括“.”的兩個部分。
* 在第三條語句中，{:8.3f}將小數部分截斷為3位元，四捨五入最後兩位元數字。而且，該數字現在為12.235，整個寬度為8，在左側保留2個位置。
* 如果要用零填充其餘位置，請在格式說明符之前放置零。它適用於整數和浮點數：{:05d}和{:08.3f}。

### 示例4：帶符號數位的數位格式

示例

# 顯示+號

print("{:+f} {:+f}".format(12.23, -12.23))

# 僅顯示-號

print("{:-f} {:-f}".format(12.23, -12.23))

# 顯示+號的空格

print("{: f} {: f}".format(12.23, -12.23))

當您運行該程式時，輸出將是：

+12.230000 -12.230000

12.230000 -12.230000 12.230000 -12.230000

### 對齊數位格式

將數位指定一定的寬度時，運算子<，和用於對齊。^>=

| 對齊數位格式 | |
| --- | --- |
| **類型** | **含義** |
| < | 左對齊剩餘空間 |
| ^ | 中心對齊剩餘空間 |
| > | 與剩餘空間右對齊 |
| = | 將帶符號（+）（-）強制到最左邊 |

### 示例5：左對齊，右對齊和居中對齊的數位格式

示例

# 右對齊的整數

print("{:5d}".format(12))

# 具有中心對齊的浮點數

print("{:^10.3f}".format(12.2346))

# 整數左對齊，用0填充

print("{:<05d}".format(12))

# 中心對齊的浮點數

print("{:=8.3f}".format(-12.2346))

當您運行該程式時，輸出將是：

   12  12.235  12000- 12.235

**注意：**對於第三個示例，左對齊用零填充整數可能會導致問題，該示例返回12000而不是12。

## 使用format()進行字串格式化

作為數位，可以使用format()以類似的方式格式化字串。

### 示例6：帶填充和對齊的字串格式

示例

# 左對齊的字串填充

print("{:5}".format("cat"))

# 右對齊的字串填充

print("{:>5}".format("cat"))

#中心對齊的字串填充

print("{:^5}".format("cat"))

# 中心對齊的字串填充

# 和 '\*' 填補字元

print("{:\*^5}".format("cat"))

當您運行該程式時，輸出將是：

cat    cat cat *\*cat\**

### 示例7：使用format()截斷字串

示例

# 將字串截斷為3個字母

print("{:.3}".format("caterpillar"))

# 將字串截斷為3個字母

# 和 填充

print("{:5.3}".format("caterpillar"))

# 將字串截斷為3個字母,

# 填充和中心對齊

print("{:^5.3}".format("caterpillar"))

當您運行該程式時，輸出將是：

catcat   cat

## 使用format()格式化類和字典成員

Python內部getattr()以“ .age”形式用於類成員。並且，它使用\_\_getitem\_\_()查找“[index]”形式的字典成員。

### 示例8：使用format()格式化類成員

示例

# define Person class

class Person:

    age = 23

    name = "Adam"

# 格式化age

print("{p.name}'年齡是 : {p.age}".format(p=Person()))

當您運行該程式時，輸出將是：

Adam's 年齡是 : 23

在這裡，Person對象作為關鍵字參數p傳遞。

在範本字串中，分別使用.name和.age訪問Person的name和age。

### 示例9：使用format()格式化字典成員

示例

# 定義Person字典

person = {'age': 23, 'name': 'Adam'}

# 格式化age

print("{p[name]}'s age is: {p[age]}".format(p=person))

當您運行該程式時，輸出將是：

Adam's age is: 23

與class類似，person字典作為關鍵字參數p傳遞。

在範本字串中，分別使用[name]和[age]訪問人員的name和age。

有一種更簡單的方法可以使用Python格式化字典str.format(\*\*mapping)。

示例

# 定義Person字典

person = {'age': 23, 'name': 'Adam'}

# 格式化age

print("{name}'s age is: {age}".format(\*\*person))

\*\* 是格式參數（最小欄位寬度）。

## 使用format()作為格式代碼的參數

您還可以動態傳遞格式代碼，例如精度，對齊，填補字元作為位置或關鍵字參數。

### 示例10：使用format()的動態格式化

示例

# 動態字串格式範本

string = "{:{fill}{align}{width}}"

# 將格式代碼作為參數傳遞

print(string.format('cat', fill='\*', align='^', width=5))

# 動態浮點格式範本

num = "{:{align}{width}.{precision}f}"

# 將格式代碼作為參數傳遞

print(num.format(123.236, align='<', width=8, precision=2))

運行該程式時，輸出為：

\*\*cat\*\*123.24

這裡，

* 在第一個示例中，" cat"是要格式化的位置參數。同樣，fill='\*'，align='^'和width=5是關鍵字參數。
* 在範本字串中，這些關鍵字參數不是作為要列印的普通字串而是作為實際格式代碼檢索的fill, align 和 width。  
  參數將替換相應的已命名預留位置，並且字串“ cat”將相應地設置格式。
* 同樣，在第二個示例中，123.236是位置參數，並且align，width和precision作為格式代碼傳遞到範本字串。

## 帶有format()的其他格式選項

format()還支援特定於類型的格式設置選項，例如日期時間格式和複數格式。

format()內部調用\_\_format\_\_()從datetime，而format()訪問複數的屬性。

您可以輕鬆地重寫任何物件的\_\_format \_\_()方法以進行自訂格式設置。

### 示例11：使用format()和重寫\_\_format \_\_()方法的特定於類型的格式化

示例

import datetime

# 日期時間格式

date = datetime.datetime.now()

print("當前時間: {:%Y/%m/%d %H:%M:%S}".format(date))

# 複數格式

complexNumber = 1+2j

print("Real part: {0.real} and Imaginary part: {0.imag}".format(complexNumber))

# 自訂  \_\_format\_\_()方法

class Person:

    def \_\_format\_\_(self, format):

        if(format == 'age'):

            return '23'

        return 'None'

print("Adam's 年齡是: {:age}".format(Person()))

運行該程式時，輸出為：

當前時間: 2020/04/17 14:33:02

Real part: 1.0 and Imaginary part: 2.0

Adam's 年齡是: 23

這裡，

* **對於datetime：**  
  當前日期時間作為位置參數傳遞給format()方法。  
  並且，在內部使用\_\_format \_\_()方法，format()可訪問年，月，日，時，分和秒。
* **對於複數：**  
  1 + 2j在內部轉換為ComplexNumber物件。  
  然後訪問其屬性real和imag，數位被格式化。
* **重寫\_\_format \_\_()：**  
  與datetime一樣，您可以重寫自己的\_\_format \_\_()方法以進行自訂格式設置，當以{：age}訪問時，該格式將返回年齡

您也可以使用物件 \_\_str\_\_() 和 \_\_repr\_\_() 功能，通過使用format()的簡寫標記法。

像\_\_format \_\_()一樣，您可以輕鬆重寫物件的\_\_str \_\_()和\_\_repr\_()方法。

### 示例12：使用format()的\_\_str()\_\_和\_\_repr()\_\_簡寫分別為！r和！s

示例

# \_\_str\_\_() 和 \_\_repr\_\_() 簡寫為 !r 和 !s

print("Quotes: {0!r}, Without Quotes: {0!s}".format("cat"))

# \_\_str\_\_() 和 \_\_repr\_\_() 實現類

class Person:

    def \_\_str\_\_(self):

        return "STR"

    def \_\_repr\_\_(self):

        return "REPR"

print("repr: {p!r}, str: {p!s}".format(p=Person()))

運行該程式時，輸出為：

Quotes: 'cat', Without Quotes: cat

repr: REPR, str: STR

# 正確使用Python f-string格式化字串的7個層級

f-string是Python 3.6推出的一種簡單而不同的字串格式技術，可以優雅地表達Python字串。除非您仍在使用舊的 Python 版本，否則在格式化字串時，f 字串絕對應該是您的首選。因為它可以通過一個迷你語法滿足您的所有要求，甚至運行字串的運算式。本文將深入探討這項技術從初級到深度的7個層次。在瞭解它們之後，您可能會成為字串格式化大師。

## 1. 輕鬆從變數顯示值

使用 f 字串只需要做兩件事：

* 在字串前添加一個小寫的f;
* 使用f字串中以{variable\_name}插值變數.

name = 'Yang'

title = 'full stack hacker'

print(f'{name} is a {title}.')

# Yang is a full stack hacker.

如上所示，在 f 字串機制的説明下，我們可以編寫簡單且更少的代碼，以便在字串中顯示更多代碼。它完美地呼應了 Python 的禪宗。

"簡單總比複雜好。

## 2. 數位格式化

有時僅僅顯示原始值可能無法滿足我們的需求，但是直接修改原始變數通常不是個好主意，因為變數可能在其他地方使用。不用擔心，Python f字串還支援"格式規範迷你語言"，它使我們能夠根據自己喜歡的方式在 f 字串中格式化值，尤其是數字。

pi = 3.1415926

print(f'Pi is approximately equal to {pi:.2f}')

# Pi is approximately equal to 3.14

id = 1 # need to print a 3-digit number

print(f"The id is {id:03d}")

# The id is 001

N = 1000000000 # need to add separator

print(f'His networth is ${N:,d}')

# His networth is $1,000,000,000

以上示例僅顯示了冰山一角。對於格式規格語法的完整清單，相應的官方文檔是您最好的朋友。

## 3. 正確列印特殊字元

我們可以通過 f 字串列印這些字元或其他特殊字元嗎？比如''和{}。是的，當然。但語法有點棘手。讓我們來看看。

## 3.1 列印引號

正如我們所知，反斜線\是常用的轉義字元，用於調用對其以下字元的替代解釋。對於 f 字串，我們需要注意一條規則：\在 f 字串運算式的括弧{}中不起作用。

name = 'Yang'

# Correct Way:

print(f'\'{name}\' is a full stack hacker.')

# 'Yang' is a full stack hacker.

# 錯誤方式:

print(f'{\'name\'} is a full stack hacker.')

# SyntaxError: f-string expression part cannot include a backslash

## 3.2 列印雙括弧{}

用 f字串列印{}的方法是不同的, 非常容易出bug。這次我們不能使用反斜線。

name = 'Yang'

# 1

print(f'{name} is a full stack hacker.')

# 'Yang' is a full stack hacker.

# 2

print(f'{{name}} is a full stack hacker.')

# {name} is a full stack hacker.

# 3

print(f'{{{name}}} is a full stack hacker.')

# {Yang} is a full stack hacker.

# 4

print(f'{{{{name}}}} is a full stack hacker.')

# {{name}} is a full stack hacker.

# 5

print(f'{{{{{name}}}}} is a full stack hacker.')

# {{Yang}} is a full stack hacker.

如上例所示，該變數是作為f-字串的括弧還是變數處理取決於其周圍的括弧數。如果您不知道這種奇怪的機制，則容易出現錯誤。

## 3.3 列印反斜線\

列印反斜線\很簡單：只需使用雙反斜線列印。但是不要將它們添加到 f 字串運算式括弧當中。

name = 'Yang'

print(f'\\{name}\\ is a full \\stack hacker.')

# \Yang\ is a full \stack hacker.

#錯誤的

print(f'{\\name\\} is a full \\stack hacker.')

# SyntaxError: f-string expression part cannot include a backslash

## 4. 小心列印字典值

將字典的值應用到 f 字串中也容易出現錯誤。我們必須使用不同的引號來描述字典鍵和 f 字串，如下所示。如果f字串用雙引號表示，那麼變數裡的字典鍵必須用單引號。

Hacker = {'name': 'Yang'}

print(f"{Hacker['name']} is a hacker")

# Yang is a hacker

print(f'{Hacker["name"]} is a hacker')

# Yang is a hacker

print(f'{Hacker['name']} is a hacker')

# 語法錯誤 SyntaxError: invalid syntax

print(f"{Hacker["name"]} is a hacker")

# 語法錯誤 SyntaxError: invalid syntax

如上所示，如果我們對鍵名和 f 字串都使用相同的單引號或雙引號, Python 會對我們的代碼感到困惑, 從而報錯。

## 5. 正確處理多行 F 字串

為了使我們的代碼更易讀，有必要使用多行書寫一長串字元。但如果是 f 字串，不要忘記在每行之前添加f。

name = 'Yang'

# 錯誤方式

print(f"{name} is a hacker."

"{name} is also a top writer."

"{name} writes on Medium."

)

# Yang is a hacker.{name} is also a top writer.{name} writes on Medium.

# 正確方式

print(f"{name} is a hacker."

f"{name} is also a top writer."

f"{name} writes on Medium."

)

# Yang is a hacker.Yang is also a top writer.Yang writes on Medium.

**6. 像大師一樣顯示日期和時間**

如果我們需要列印日期或時間，f 字串將再次告訴我們它是多麼方便:

from datetime import datetime

today = datetime.today()

print(f"Today is {today}")

# Today is 2021-07-31 18:20:48.956829

print(f"Today is {today:%B %d, %Y}")

# Today is July 31, 2021

print(f"Today is {today:%m-%d-%Y}")

# Today is 07-31-2021

如上例所示，在 f 字串的説明下，我們可以使用我們任何喜歡的格式列印日期或時間。

**7. 評估 F 字串內的運算式**

當我第一次知道 f 字串時， 我簡直不敢相信：我們可以在 f 字串內運行 Python 運算式。如果是真的，還算是字串嗎？

我仔細閱讀了 PEP 498，終於明白了：

F 字串提供了一種將運算式嵌入字串字面的方法。需要注意的是，f 字串實際上是在執行時間評估的表達方式，而不是恒定的值。

因此，f 字串與普通字串不同，此功能賦予它更大的能力。例如，我們可以在它裡面運行一個顯示時間的功能。

from datetime import datetime

print(f"Today is {datetime.today()}")

# Today is 2021-07-31 18:20:48.956829

## 結論

Python 中的 f-string是一個偉大的字串格式化技術， 顯示了 Python 是多麼優雅。它非常強大，因為它不是一個普通的字串，而是在運行時可以表達的字串。對於一些特殊情況，它有特殊的規則，我們需要謹慎使用它。