第2章 Python語法

Python語言的語法具有幾個獨特的特點，這些特點與其他編程語言相比較為突出。以下是一些主要的特色：

1. **簡潔清晰** ：Python以其簡潔和易於閱讀的代碼而聞名。它鼓勵使用白空間和少量的括號，使得代碼更加清晰易懂。
2. **強制縮排** ：Python使用縮排來定義代碼塊，而不是使用大括號或關鍵字。這使得Python代碼具有一致的格式，並強制程序員編寫結構化的代碼。
3. **動態類型** ：Python是一種動態類型語言，這意味著變量在運行時被賦予類型，不需要像靜態類型語言那樣在代碼中顯式聲明類型。
4. **物件導向** ：Python支持物件導向編程，但它的實現方式與如Java等其他物件導向語言有所不同。例如，Python中的類成員是公開的，並且可以通過特殊方法來實現封裝。
5. **豐富的內建數據結構** ：Python內置了多種數據結構，如列表、字典、元組和集合，這些都是通過直觀的語法輕鬆使用的。
6. **列表推導式和生成器表達式** ：Python支持列表推導式和生成器表達式，這些功能提供了一種強大且簡潔的方法來創建和操作列表和其他集合類型。
7. **多重賦值和解包** ：Python允許多重賦值，以及元組和列表的自動打包和解包，這使得交換變量的值或從函數返回多個值變得簡單。
8. **lambda函數** ：Python支持匿名函數或lambda函數，這是一種簡潔的表示小函數的方式。
9. **切片操作** ：Python的切片操作讓從序列（如列表或字符串）中提取部分元素變得非常容易和直觀。

這些語法特點使Python成為一種易於學習和使用的編程語言，特別是對於初學者。它鼓勵寫出可讀性高且維護性好的代碼，同時提供了強大的功能來支持各種編程風格。

2.1 變數與基本資料型別

變數

```
x = 5
y = "Hello, World!"
```

在這個例子中，`x` 是一個整數型變數，我們將數值 5 存入其中。同樣地，`y` 是一個字串型變數，儲存了文字 "Hello, World!"。

變數命名規則：

1. 變數名稱可以包含字母、數字、底線，但不能以數字開頭。
2. 變數名稱應該有描述性；例如，使用 `name` 或 `age` 而非 `n` 或 `a`。
3. 在 Python 中，變數名稱區分大小寫，所以 `variable` 和 `Variable` 是兩個不同的變數。
4. 避免使用 Python 的關鍵字和函式名作為變數名（例如 `if`、`while`、`class` 等）。

總之，變數是存儲數據的基本方式，在學習 Python 時非常重要。通過適當地命名和使用變數，可以使你的代碼更加清晰易懂。

賦值

- 在編程中，賦值（Assignment）是將一個值存儲在變量中的過程。在Python中，賦值的語法和概念是其核心特性之一，具有以下特點：

1. **基本賦值** ：

- 使用等號 `=` 進行賦值。例如，`x = 5` 將數值 5 賦值給變量 `x`。
- Python是動態類型語言，所以不需要事先聲明變量的類型。

2. **鏈式賦值** ：

- 可以同時將多個變量賦予相同的值。例如，`a = b = c = 5` 會使得 `a`、`b` 和 `c` 三個變量都等於 5。

3. **多重賦值** ：

- Python允許在單一語句中為多個變量賦值。例如，`x, y = 1, 2` 會將 `x` 賦值為 1，將 `y` 賦值為 2。

4. **解包賦值** ：

- 可以從列表、元組或任何可迭代對象中解包值並賦給多個變量。例如，`x, y, z = [1, 2, 3]` 會將 `x`、`y`、`z` 分別賦值為 1、2、3。

5. **星號表達式** ：

- 在多重賦值中使用星號（`*`）可以捕獲多個值。例如，`first, *middle, last = [1, 2, 3, 4, 5]` 會將 `first` 賦值為 1，`last` 賦值為 5，而 `middle` 則是一個列表 `[2, 3, 4]`。

6. **增量賦值** ：

- Python支持增量賦值運算符，如 `+=`、`-=`、`*=` 等。例如，`x += 1` 會將 `x` 的值增加 1。

賦值操作在Python中是基本且頻繁使用的，並且其簡潔靈活的語法是Python易於學習和使用的原因之一。這些賦值方法提供了強大的數據操作能力，並使代碼更加簡潔和易於閱讀。

註解

在 Python 中，註解（Comment）是用於增加代碼的可讀性，幫助開發者和其他閱讀代碼的人理解代碼的用途和功能。註解不會被 Python 解釋器執行，因此可以用來提供對代碼的說明或暫時禁用某些代碼。

單行註解

- 使用 `#` 符號來開始一個單行註解。
- `#` 後的任何內容都會被 Python 解釋器忽略。

例子：

```
# 這是一個註解
print("Hello, World!")  # 這也是註解
```

多行註解

雖然 Python 沒有專門的多行註解語法，但可以使用三個連續的單引號 `'''` 或雙引號 `"""` 來創建類似多行註解的效果。

例子：

```
'''
這是一個多行註解
它包含了多行文字
'''

"""
這也是一個多行註解
使用雙引號
"""
print("Hello, World!")
```

註解的用途

1. **解釋代碼** ：說明複雜的邏輯，使其他人更容易理解。
2. **代碼調試** ：暫時停用某些代碼行，而不是刪除它們。
3. **撰寫文件** ：在函數和類別上方寫說明文字，介紹它們的功能和用法。
4. **標記** ：標記未來可能需要修改或完善的地方。

注意事項

- 過度註解可能會使代碼混亂，因此只在必要時添加註解。
- 註解應該簡潔明了，直接表達註解的目的。
- 隨著代碼的更新，記得更新相應的註解，以避免造成混淆。

合理地使用註解是良好編程習慣的一部分，可以顯著提高代碼的可維護性和可讀性。

2.2 基本資料型別

在Python中，基本資料型別是指那些構成資料核心的型別。了解這些型別對於學習Python至關重要。以下將介紹幾種常見的基本資料型別：

1. 整數（Integer）：這是最基本的數值型別，用於表示沒有小數點的數字。例如：1, 100, -50。
2. 浮點數（Float）：用於表示帶有小數的數字。例如：1.0, 3.14, -0.001。
3. 複數（Complex）：用於表示包含實部和虛部的數字，實部和虛部都是浮點數。Python中的複數可以用“j”或“J”來表示虛部。例如：1 + 2j, 3.14 - 5.6j。
4. 字串（String）：用於表示文字，必須使用引號括起來。可以使用單引號或雙引號。例如："hello", 'python'。
5. 布林值（Boolean）：用於表示真假。只有兩個值：True（真）和 False（假）。了解這些基本資料型別對於學習Python是非常重要的，因為它們是許多Python程序的基礎。每種型別都有其特定的用途和操作方法，熟悉它們將幫助你更有效地使用Python。

型別轉換

在 Python 中，「Casting」是指將變數從一種類型轉換成另一種類型的過程。Python 提供了一些內建的函數，可以用來實現基本的資料類型之間的轉換。這些轉換通常在你需要將變數的數據類型轉換為其他類型時進行，例如，從整數轉換為浮點數，或從字串轉換為整數等。

常見的 Casting 函數

1. `int()`: 將一個數字或有效的字串轉換成一個整數。
2. `float()`: 將一個整數或字串轉換成一個浮點數。
3. `str()`: 將指定的對象轉換成字串形式。

例子

下面的例子展示了如何在 Python 中使用這些 Casting 函數：

```python
# 整數轉浮點數
num_int = 7
num_float = float(num_int)
print(num_float)  # 輸出: 7.0

# 浮點數轉整數
num_float = 8.5
num_int = int(num_float)
print(num_int)  # 輸出: 8

# 數字轉字串
num_int = 10
str_num = str(num_int)
print(str_num)  # 輸出: "10"

# 字串轉整數
str_num = "15"
num_int = int(str_num)
print(num_int)  # 輸出: 15
```

注意事項

- 在進行類型轉換時，要確保轉換是有意義的和有效的。例如，將包含非數字字符的字串轉換為數字會導致錯誤。
- 轉換浮點數為整數時，Python 會進行截斷處理（捨去小數部分），而不是四捨五入。

通過這些基本的類型轉換函數，你可以在不同類型之間靈活地轉換數據，進行有效的數據處理和操作。

2.3 數值操作與運算元

在Python中，基本的數值類型主要有整數（Integers）和浮點數（Floats）。整數是沒有小數點的數字，如1、100、-20，而浮點數則是有小數點的數字，如1.23、3.14、-0.5。

基本運算元

1. 加法（`+`）：將兩個數相加。

```
print(3 + 5)  # 輸出 8
```

2. 減法（`-`）：從一個數中減去另一個數。

```
print(10 - 4)  # 輸出 6
```

3. 乘法（`*`）：將兩個數相乘。

```
print(7 * 3)  # 輸出 21
```

4. 除法（`/`）：將一個數除以另一個數。

```
print(8 / 2)  # 輸出 4.0
```

5. 整數除法（`//`）：除法後取整數部分。

```
print(8 // 3)  # 輸出 2
```

6. 取餘數（`%`）：得到除法的餘數。

```
print(8 % 3)  # 輸出 2
```

7. 指數（`**`）：計算一個數的指數。

```
print(2 ** 3)  # 輸出 8
```

操作範例

讓我們來看一個簡單的範例：

```
# 定義變數
a = 10
b = 3

# 進行運算
加法結果 = a + b
減法結果 = a - b
乘法結果 = a * b
除法結果 = a / b

# 輸出結果
print("加法結果:", 加法結果)  # 輸出 13
print("減法結果:", 減法結果)  # 輸出 7
print("乘法結果:", 乘法結果)  # 輸出 30
print("除法結果:", 除法結果)  # 輸出 3.3333333333333335
```

這些是Python中最基本的數值操作和運算元。你可以通過這些基礎的運算來執行簡單的數學計算。隨著你學習的深入，你將接觸到更多複雜的數值處理方法。

運算元種類

在 Python 中，運算元（Operators）是用來執行對變數或值的操作。根據操作的性質，Python 中的運算元可以分為以下幾種主要類別：

1. 算術運算元

用於執行基本數學運算。

- `+`：加法
- `-`：減法
- `*`：乘法

- `/` ：除法
- `%` ：模數（求餘數）
- `**` ：指數（冪運算）
- `//` ：整除（向下取整的除法）

2. 比較（關係）運算元

用於比較兩個值之間的關係。

- `==` ：等於
- `!=` ：不等於
- `>` ：大於
- `<` ：小於
- `>=` ：大於等於
- `<=` ：小於等於

3. 賦值運算元

用於將值賦給變數。

- `=` ：基本賦值
- `+=` ：加法後賦值
- `-=` ：減法後賦值
- `*=` ：乘法後賦值
- `/=` ：除法後賦值
- `%=` ：模數後賦值
- `**=` ：指數後賦值
- `//=` ：整除後賦值

4. 邏輯運算元

用於組合布林（真/假）值。

- `and` ：邏輯與
- `or` ：邏輯或
- `not` ：邏輯非

5. 身份運算元

用於比較兩個對象是否相同。

- `is` ：判斷兩個引用是否指向同一對象
- `is not`：判斷兩個引用是否指向不同對象

6. 成員運算元

用於測試序列中是否包含指定的值。

- `in` ：判斷指定值是否存在於序列中
- `not in`：判斷指定值是否不在序列中

7. 位元運算元

用於對數字進行二進位（位元）級操作。

- `&` ：位元與
- `|` ：位元或
- `^` ：位元異或
- `~` ：位元取反
- `<<`：左移
- `>>`：右移

這些運算元是 Python 程式設計中的基礎，了解和熟悉它們對於寫出有效和高效的代碼至關重要。

2.4 字串操作

在 Python 中，字串操作是一個非常重要的主題，因為它讓我們能夠處理和修改文字資料。下面將透過一些基本的例子來介紹如何在 Python 中進行字串操作。

字串創建

首先，創建一個字串很簡單，只需要將文字放入單引號或雙引號中。

```
str1 = "Hello"
str2 = 'World'
```

字串串接

你可以使用加號 `+` 來串接兩個字串。

```
greeting = str1 + " " + str2
print(greeting)  # 輸出: Hello World
```

字串長度

使用 `len()` 函數來獲得字串的長度。

```
length = len(greeting)
print(length)  # 輸出: 11
```

字串索引

字串中的每個字符都有一個索引，從 0 開始。

```
first_char = greeting[0]
print(first_char)  # 輸出: H
```

字串切片

切片讓你可以獲得字串的一部分。

```
sub_string = greeting[0:5]
print(sub_string)  # 輸出: Hello
```

字串分割

使用 `split()` 方法來分割字串。

```
words = greeting.split(" ")
print(words)  # 輸出: ['Hello', 'World']
```

字串替換

使用 `replace()` 方法來替換字串中的文字。

```
new_greeting = greeting.replace("Hello", "Hi")
print(new_greeting)  # 輸出: Hi World
```

字串大小寫轉換

使用 `upper()` 和 `lower()` 方法來轉換字串的大小寫。

```
upper_case = greeting.upper()
lower_case = greeting.lower()
print(upper_case)  # 輸出: HELLO WORLD
print(lower_case)  # 輸出: hello world
```

字串格式化

在 Python 中，字串格式化是一種非常有用的功能，它允許你創建一個格式化的字串，可以包含來自變數或表達式的資料。有幾種常見的方法可以實現字串格式化：

1. 使用 `str.format()`

這是 Python 2.6 以上版本引入的一種新的格式化方法，比 `%` 運算符更加靈活。

```
name = "Bob"
age = 30
msg = "Hello, {}. You are {} years old.".format(name, age)
print(msg)  # 輸出: Hello, Bob. You are 30 years old.
```

2. 使用 f-string (格式化字串字面值)

這是 Python 3.6 以上版本引入的最新格式化方法，簡潔且易於閱讀。

```
name = "Carol"
age = 28
msg = f"Hello, {name}. You are {age} years old."
print(msg)  # 輸出: Hello, Carol. You are 28 years old.
```

字串格式化的應用

1. **插入變數** ：可以將變數的值插入到字串中。
2. **格式化數字** ：可以指定數字的格式，如小數位數、千分位等。
3. **對齊和填充** ：可以指定字串的對齊方式和用於填充的字符。

例子：格式化數字

```
pi = 3.1415926
formatted_pi = f"Pi is approximately {pi:.2f}"
print(formatted_pi)  # 輸出: Pi is approximately 3.14
```

在這裡，`{pi:.2f}` 表示將變數 `pi` 格式化為帶有兩位小數的浮點數。

字串格式化是 Python 中一個非常強大的功能，能夠幫助你輕鬆地組織和呈現數據。使用適當的格式化方法可以使你的代碼更加清晰和易於維護。

2.5 容器資料型別

容器資料型別指的是可以包含其他對象的資料結構。這些型別允許我們將多個元素組織在一起，通常用於存儲、訪問和操作數據集合。主要的容器型別包括列表（List）、元組（Tuple）、字典（Dictionary）和集合（Set）。下面我將為您詳細介紹每種容器型別的特點和用途：

1. **列表（List）**：

- 可變的（Mutable），意味著列表創建後可以修改。
- 可以包含不同類型的元素，例如數字、字符串、甚至其他容器。
- 支持索引和切片操作，可以用來訪問和修改列表中的元素。
- 例子：`[1, 2, 3, 'hello', [4.5, 'world']]`

2. **元組（Tuple）**：

- 不可變的（Immutable），一旦創建就不能更改。
- 也可以包含不同類型的元素。
- 常用於函數返回多個值和將數據作為「只讀」格式存儲。
- 例子：`(1, 2, 3, 'hello', (4.5, 'world'))`

3. **字典（Dictionary）**：

- 由鍵（Key）和值（Value）對組成的集合。
- 鍵是唯一的，而值則可以是任何資料型別。
- 可變的，可以在運行時添加或刪除鍵值對。
- 常用於需要快速訪問數據的場景，如數據庫查詢。
- 例子：`{'name': 'Alice', 'age': 25, 'email': 'alice@example.com'}`

4. **集合（Set）**：

- 一組無序且唯一的元素。
- 可變的，可以在運行時添加或刪除元素。
- 常用於去除重複元素和執行集合運算（如聯集、交集、差集等）。
- 例子：`{1, 2, 3, 'apple', 'banana'}`

了解和熟練使用這些容器型別對於有效進行數據處理和實現複雜的數據結構至關重要。每種容器型別都有其特定的用途和特點，熟悉這些將幫助您在Python編程中更加得心應手。

2.6 列表與列表操作

在 Python 中，列表（List）是一種非常靈活且廣泛使用的數據結構。它可以儲存不同類型的元素，包括數字、字串、甚至其他列表。列表是可變的，這意味著我們可以在程序運行時添加、移除或修改其內容。

創建列表

列表通常用方括號 `[]` 表示，元素之間用逗號分隔。

```
my_list = [1, 2, 3, "Python", "台灣"]
```

存取元素

列表中的元素可以透過索引來存取，索引從 0 開始。

```
first_element = my_list[0]  # 1
last_element = my_list[-1]  # "台灣" (使用負數索引可以從列表末尾開始存取)
```

添加元素

可以使用 `append()` 方法在列表末尾添加新元素。

```
my_list.append("新元素")
```

插入元素

使用 `insert()` 方法在指定位置插入元素。

```
my_list.insert(1, "插入元素")  # 在索引 1 的位置插入
```

移除元素

有幾種方法可以從列表中移除元素：

- 使用 `remove()` 移除第一個匹配的元素。
- 使用 `pop()` 移除並返回指定位置的元素（預設為最後一個）。
- 使用 `del` 關鍵字來移除特定索引的元素或切片。

```
my_list.remove("Python")  # 移除 "Python"
popped_element = my_list.pop()  # 移除並返回最後一個元素
del my_list[0]  # 刪除第一個元素
```

列表切片

切片用於獲取列表的一部分。

```
sub_list = my_list[1:3]  # 獲取從索引 1 到 2 的元素
```

列表遍歷

遍歷列表的元素，可以使用 for 循環。

```
for element in my_list:
    print(element)
```

列表長度

使用 `len()` 函數來獲取列表的長度。

```
length = len(my_list)
```

列表合併

可以使用加號 `+` 或 `extend()` 方法來合併兩個列表。

```
another_list = [4, 5, 6]
combined_list = my_list + another_list  # 使用加號合併
my_list.extend(another_list)  # 使用 extend 方法合併
```

列表解析

在 Python 中，列表解析（List Comprehension）是一種簡潔且高效的方法，用於創建列表。這種方法不僅可以讓代碼更加簡潔，還能提高執行效率。列表解析通常用於從其他列表或可迭代對象中生成新的列表。

基本語法

列表解析的基本語法如下：

```
[表達式 for 項目 in 可迭代對象]
```

簡單的例子

例如，創建一個包含前 10 個整數平方的列表：

```
squares = [x**2 for x in range(10)]
print(squares)  # 輸出: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

添加條件語句

你還可以在列表解析中加入條件語句，以過濾元素或修改輸出結果。

```
# 創建一個只包含偶數的平方列表
even_squares = [x**2 for x in range(10) if x % 2 == 0]
print(even_squares)  # 輸出: [0, 4, 16, 36, 64]
```

嵌套列表解析

列表解析也可以嵌套，以處理更複雜的數據結構。

```
# 生成一個 3x3 的二維矩陣
matrix = [[j for j in range(3)] for i in range(3)]
print(matrix)  # 輸出: [[0, 1, 2], [0, 1, 2], [0, 1, 2]]
```

列表解析與傳統迴圈的比較

使用列表解析通常比使用傳統的迴圈方法更為簡潔。比如，以下兩段代碼完成相同的功能，但列表解析的代碼更加簡潔：

```
# 使用傳統迴圈
squares = []
for x in range(10):
    squares.append(x**2)

# 使用列表解析
squares = [x**2 for x in range(10)]
```

> 列表解析是 Python 中一個非常強大且有用的特性。它能夠讓你以更簡潔、更高效的方式生成列表。當你需要從一個列表或其他可迭代對象中創建一個新列表時，考慮使用列表解析。不過，也要注意，過於複雜的列表解析可能會降低代碼的可讀性，因此在這些情況下使用傳統的迴圈可能是一個更好的選擇。

2.7 字典

字典（Dictionary）是一種非常靈活且強大的容器型別，主要用於存儲和管理成對的鍵值對（key-value pairs）。以下是字典的一些關鍵特點和常用操作：

字典的特點

1. **鍵值對結構**：

- 字典中的每個元素都是一個鍵值對，鍵（key）用於唯一標識元素，值（value）則是與鍵相關聯的數據。

2. **鍵的唯一性**：

- 每個鍵在字典中必須是唯一的。如果添加了重複的鍵，則新的值將覆蓋舊的值。

3. **可變性**：

- 字典是可變的，可以在運行時增加、刪除或修改鍵值對。

4. **無序性**：

- 字典是無序的，這意味著不保證元素的存儲順序。

5. **動態性**：

- 字典可以根據需要動態地擴展或收縮。

常見操作

1. **創建字典**：

```
my_dict = {'name': 'John', 'age': 30, 'city': 'New York'}
```

2. **訪問元素**：

- 通過鍵來訪問對應的值。

```
name = my_dict['name']  # 返回 'John'
```

3. **修改元素**：

- 通過鍵來修改值。

```
my_dict['age'] = 31
```

4. **添加新的鍵值對**：

```
my_dict['email'] = 'john@example.com'
```

5. **刪除元素**：

- 可以使用 `del` 關鍵字或 `pop` 方法。

```
del my_dict['city']  # 刪除鍵 'city'
age = my_dict.pop('age')  # 刪除鍵 'age' 並返回其值
```

6. **檢查鍵是否存在**：

```
'name' in my_dict  # 返回 True 或 False
```

7. **遍歷字典**：

- 可以遍歷鍵、值或鍵值對。

```
for key in my_dict:
    print(key, my_dict[key])
```

8. **字典的方法**：

- 如 `keys()`, `values()`, `items()` 等方法提供了更多操作字典的方式。

字典在許多情況下都非常有用，特別是當需要建立和管理大量成對數據時。由於其高效的鍵值對存儲方式，字典在Python中被廣泛應用於數據結構和算法的實現中。

2.8 縮排

在Python程式語言中，縮排是一種重要的結構元素。它被用來定義程式碼的結構，特別是用來劃分程式碼塊。這與許多其他編程語言不同，這些語言可能使用大括號（{}）來定義程式碼塊。

Python中的縮排規則如下：

1. **一致性**：所有同一程式碼塊內的語句必須具有相同的縮排水平。
2. **空格數量**：標準縮排通常是四個空格。不過，重要的是選擇一種縮排風格（空格數）並在整個程式中一致使用。
3. **程式碼塊**：條件語句（如if）、迴圈（如for和while）以及函式和類定義等都使用縮排來劃分程式碼塊。
4. **縮排錯誤**：如果縮排不一致，Python解釋器會拋出`IndentationError`，指出程式無法正確執行。

正確使用縮排是學習Python的關鍵部分，因為它不僅影響程式的可讀性，還直接影響程式的邏輯和功能。

使用縮排場合

在Python中，以下這些語句會用到縮排來定義程式碼塊：

1. **條件語句**：`if`、`elif`、`else`。這些用於執行基於條件的不同程式碼塊。

```
if 條件:
    # 條件為真時執行的程式碼
elif 另一條件:
    # 另一條件為真時執行的程式碼
else:
    # 所有條件都不滿足時執行的程式碼
```

2. **迴圈**：`for` 和 `while`。用於重複執行程式碼塊直到滿足特定條件。

```
for 變數 in 序列:
    # 對序列中的每個元素執行的程式碼

while 條件:
    # 當條件為真時重複執行的程式碼
```

3. **函式定義**：使用 `def` 關鍵字。

```
def 函式名稱(參數):
    # 函式內部的程式碼
```

4. **類別定義**：使用 `class` 關鍵字。

```
class 類別名稱:
    # 類別內部的程式碼，包括方法定義
```

5. **上下文管理器**：使用 `with` 關鍵字，常見於檔案操作。

```
with 某個上下文管理器 as 變數:
    # 在此上下文中執行的程式碼
```

6. **例外處理**：使用 `try`、`except`、`finally`、`else`。

```
try:
    # 嘗試執行的程式碼
except 錯誤類型:
    # 發生特定錯誤時執行的程式碼
else:
    # 沒有錯誤發生時執行的程式碼
finally:
    # 無論是否發生錯誤都會執行的程式碼
```

這些語句中，隨著關鍵字後的冒號，緊接著的新行就需要縮排。這樣Python解釋器才能正確地識別出哪些程式碼是屬於哪個程式碼塊的。正確的縮排對於保持程式碼的清

晰結構和正確執行非常關鍵。

2.9 迴圈

在 Python 中，迴圈是一種基本的程式結構，用於重複執行一段代碼。Python 提供了兩種主要的迴圈類型：`for` 迴圈和 `while` 迴圈。此外，`break` 和 `continue` 是用於控制迴圈執行的兩個重要語句。

`for` 迴圈

`for` 迴圈用於遍歷序列（如列表、元組、字典、集合、字串）中的每個元素。

基本語法：

```
for 變數 in 序列:
    執行的代碼
```

例子：

```
for i in range(5):
    print(i)  # 輸出 0, 1, 2, 3, 4
```

`while` 迴圈

`while` 迴圈會在條件為真時重複執行。

基本語法：

```
while 條件:
    執行的代碼
```

例子：

```
i = 0
while i < 5:
    print(i)
    i += 1  # 輸出 0, 1, 2, 3, 4
```

迴圈加上邏輯判斷

當我們在 Python 中使用迴圈加上邏輯判斷時，這使得我們可以在特定條件下重複執行某些操作。這是自動化和處理重複任務的強大工具。讓我們通過一個範例來了解它的工作原理。

範例：找出一個範圍內的所有偶數
假設我們想要找出 1 到 10 之間的所有偶數。我們可以使用 for 迴圈來遍歷這個範圍，並使用 if 語句來檢查每個數字是否為偶數。

```
for i in range(1, 11):
    if i % 2 == 0:
        print(f"{i} 是偶數")
```

在這個例子中，range(1, 11) 生成一個從 1 到 10 的數字序列。for 迴圈遍歷這個序列中的每個數字（用變量 i 表示）。對於迴圈中的每個數字，我們使用 if 語句來檢查 i % 2 == 0 是否為真。% 是模數運算符，用於找出兩個數相除的餘數。因此，如果 i 被 2 整除，那麼 i % 2 的結果將是 0，表示 i 是偶數。

這個範例展示了如何結合迴圈和邏輯判斷來處理一些常見的編程任務。您可以嘗試修改這段代碼來尋找奇數，或者更改範圍來實踐這些概念。

`break` 語句

`break` 用於立即終止迴圈的執行，無論迴圈條件是否仍為真。

例子：

```
for i in range(5):
    if i == 3:
        break
    print(i)  # 輸出 0, 1, 2
```

`continue` 語句

`continue` 用於跳過當前迴圈的剩餘代碼，並進行下一次迴圈的迭代。

例子：

```
for i in range(5):
    if i == 3:
        continue
    print(i)  # 輸出 0, 1, 2, 4
```

總結

- `for` 迴圈用於遍歷序列中的元素。
- `while` 迴圈在滿足一定條件時重複執行。
- `break` 用於提前終止迴圈。
- `continue` 用於跳過當前迴圈的剩餘部分，直接進入下一次迭代。

適當地使用這些迴圈和控制語句可以讓你的程式代碼更有效率和易於理解。不過，要注意避免使用過多的 `break` 和 `continue`，因為它們有時會使代碼的流程變得難以追蹤和維護。

2.10 邏輯判斷

當您在 Python 中使用 `if-else` 語句時，您實際上在告訴電腦：「如果某件事情是真的，那麼做這件事；否則，做另一件事。」這是編程中非常基本且強大的一個概念。

讓我們透過一個簡單的例子來說明 `if-else` 的用法：

```
number = 10

if number > 5:
    print("數字大於 5")
else:
    print("數字不大於 5")
```

在這個例子中，我們首先設置一個變數 `number`，並給它賦值 10。接著，我們使用 `if-else` 語句來檢查 `number` 是否大於 5。如果 `number > 5`（這是「如果」的條件），則執行冒號後面的代碼塊（在這個例子中是 `print("數字大於 5")`）。如果條件不滿足（即 `number <= 5`），則執行 `else` 之後的代碼塊（即 `print("數字不大於 5")`）。

您可以透過更改 `number` 變數的值來看看不同情況下會發生什麼。這是學習條件語句的一個很好的練習。

布林與布林操作

在 Python 中，布林（Boolean）是一種數據類型，用於表示真（True）或假（False）的值。布林值通常用於條件判斷和控制程式流程。布林操作則涉及使用布林運算元來處理布林值。

布林值

- `True`：代表真，是一個布林值。
- `False`：代表假，也是一個布林值。

> 布林值 `True` 和 `False` 的首字母必須大寫。

布林操作

Python 提供了幾種基本的布林操作：

1. **邏輯運算元**

- `and`：邏輯與（AND），當所有條件都為真時，結果為真。
- `or`：邏輯或（OR），當至少一個條件為真時，結果為真。
- `not`：邏輯非（NOT），用來反轉布林值，`True` 變 `False`，`False` 變 `True`。

例子：

```
a = True
b = False
print(a and b)  # 輸出: False
print(a or b)   # 輸出: True
print(not a)    # 輸出: False
```

2. **比較運算元**

- 用於比較數值或變數，結果為布林值。
- 包括 `==`（等於）、`!=`（不等於）、`>`（大於）、`<`（小於）、`>=`（大於等於）、`<=`（小於等於）。

例子：

```
print(5 > 3)   # 輸出: True
print(2 == 3)  # 輸出: False
```

3. **身份運算元**

- 用於比較兩個物件是否為同一個實例。
- `is` 和 `is not` 是身份運算元。

例子：

```
x = [1, 2, 3]
y = [1, 2, 3]
print(x is y)      # 輸出: False，因為 x 和 y 是不同的對象
print(x is not y)  # 輸出: True
```

4. **成員運算元**

- 用於檢測一個值是否存在於序列中（如列表、元組、字串）。
- `in` 和 `not in` 是成員運算元。

例子：

```
list = [1, 2, 3]
print(1 in list)       # 輸出: True
print(5 not in list)   # 輸出: True
```

使用場景

- **控制流程**：在 `if`、`while` 等語句中用於控制程式的流程。
- **條件表達式**：在決定執行哪個分支或迴圈是否繼續時使用。

布林條件

在 Python 中，某些值或條件被評估為 `True`，這在布林上下文（如 `if` 語句中的條件）中非常重要。以下是在 Python 中被認為是 `True` 的情況：

1. **非零數值**：

- 任何非零的整數或浮點數。例如，`1`, `-1`, `3.14` 等。

2. **非空容器**：

- 包含至少一個元素的字符串、列表、元組、集合或字典。例如，`"hello"`, `[1, 2, 3]`, `(1,)`。

3. **特殊物件**：

- 大多數物件，包括自定義類的實例，除非該類定義了 `__bool__()` 或 `__len__()` 方法並返回 `False` 或 `0`。

4. **特定常數** ：

- `True` 本身。

5. **具有非零長度的自定義物件** ：

- 如果一個自定義物件的類定義了 `__len__()` 方法，並且該方法返回非零值，則該物件在布林上下文中被認為是 `True`。

相對地，以下值或條件在 Python 中被認為是 `False`：

- 數值 `0`（包括 `0`, `0.0`）。
- 空容器，如 `""`（空字符串）、`[]`（空列表）、`()`（空元組）、`{}`（空字典）和 `set()`（空集合）。
- 常數 `None`。
- 自定義物件，如果其類定義了 `__bool__()` 方法並返回 `False`，或者定義了 `__len__()` 方法並返回 `0`。

這些規則是 Python 中處理布林值和條件判斷的基礎，對於編寫條件語句和進行邏輯判斷非常關鍵。

2.11 異常處理

- 在 Python 中，`try` 和 `except` 語句用於異常處理。當 Python 程式碼中發生錯誤時，它會引發異常。異常處理是一種處理 Python 程式中錯誤的方法，它可以讓程式在面對預期外的情況時，仍然能夠正常運行或優雅地終止。

基本結構

`try` 塊讓你測試一塊代碼以查看是否有錯誤。`except` 塊讓你處理錯誤。

語法：

```
try:
    # 嘗試執行的代碼
except 發生異常的類型:
    # 出現異常時，執行的代碼
```

範例

以下是一個基本的 `try` 和 `except` 使用例子：

```
try:
    # 嘗試除以零
    result = 10 / 0
except ZeroDivisionError:
    print("不能除以零！")
```

在這個例子中，如果 `10 / 0` 的運算導致 `ZeroDivisionError` 異常，則 `except` 塊的代碼會被執行。

處理多種異常

你可以定義多個 `except` 塊來處理不同類型的異常：

```
try:
    # 可能引發多種異常的代碼
except ZeroDivisionError:
    # 處理除以零的錯誤
except ValueError:
    # 處理值錯誤
```

通用異常處理

使用 `except` 而不指定任何異常類型可以捕獲所有類型的異常：

```
try:
    # 嘗試執行的代碼
except:
    # 處理所有類型的異常
```

不過，這種做法並不推薦，因為它會掩蓋所有錯誤，使得偵錯變得困難。

`else` 和 `finally` 語句

- `else` 塊：如果沒有異常發生，則執行 `else` 塊。
- `finally` 塊：無論是否發生異常，`finally` 塊都會被執行。

語法：

```
try:
    # 嘗試執行的代碼
except 異常類型:
    # 異常處理代碼
else:
    # 沒有異常時執行的代碼
finally:
    # 無論是否異常都會執行的代碼
```

總結

`try` 和 `except` 語句在 Python 中是異常處理的核心。合理使用異常處理可以使你的程式在面對錯誤時更加穩健，並提供更友好的用戶體驗。然而，過度依賴異常處理來控制程序流程是不好的做法，應當在必要時才使用異常處理。

2.12 檔案讀寫

在 Python 中，讀取和寫入文件是一個常見且重要的任務。我將為您提供一個基本的指南，來說明如何完成這些操作。

讀取文件

要讀取文件的內容，您可以使用 `open()` 函數與 `with` 語句。這樣可以確保文件在讀取後會被正確地關閉。下面是一個簡單的例子：

```
with open('example.txt', 'r', encoding='utf-8') as file:
    content = file.read()
    print(content)
```

這裡，`'example.txt'` 是要讀取的文件名。`'r'` 模式表示「讀取模式」。`encoding='utf-8'` 確保文件以正確的編碼被讀取，這對於處理非英文字符很重要。

寫入文件

寫入文件與讀取類似，但您會使用 `'w'` 模式（寫入模式）。如果文件不存在，Python 會創建它。如果文件已存在，它會被覆蓋。

```
with open('example.txt', 'w', encoding='utf-8') as file:
    file.write('這是一些要寫入的文字。')
```

使用 `'w'` 模式會覆蓋原有的內容。如果您想要在文件的末尾添加內容，應該使用 `'a'` 模式（附加模式）。

> 注意事項
> - 確保使用 `with` 語句來自動管理文件的開關。這樣做可以預防很多常見的錯誤。
> - 適當地處理文件路徑和錯誤。在處理文件時，總是有可能遇到各種問題，例如文件不存在或權限不足等。

> - 在處理文本文件時，正確的字符編碼非常重要，特別是當文件包含非英文字符時。

檔案路徑

在Python中，路徑表示法是指如何以字符串的形式表達檔案系統中的檔案或目錄的位置。Python處理檔案路徑時有幾個重要的原則和概念：

1. **絕對路徑與相對路徑** ：

- **絕對路徑** ：指從檔案系統的根目錄（例如，在Windows上是 `C:\` ，在Unix/Linux系統上是 `/` ）開始的完整路徑。
- **相對路徑** ：相對於當前工作目錄的路徑。例如，如果當前目錄是 `/home/user` ，則相對路徑 `documents/report.txt` 實際上指的是 `/home/user/documents/report.txt` 。

2. **路徑分隔符號** ：

- 在Windows系統中，路徑分隔符通常是反斜杠 `\` 。
- 在Unix/Linux和Mac OS系統中，則使用斜杠 `/` 作為分隔符。
- Python的 `os` 模塊中的 `os.path` 可以智慧地處理不同操作系統間的路徑差異。

3. **通用字元和轉義** ：

- 路徑中可能包含需要特殊處理的字元，如空格。在這些情況下，路徑應當使用引號括起來，或者使用轉義字元（如在空格前使用 `\` ）。
- 在字符串中直接使用反斜杠（如在Windows路徑中）時，應該使用原始字符串（在字符串前加 `r` ，例如 `r"C:\Users\Name"` ），以避免Python將反斜杠作為轉義字元處理。

4. **路徑操作** ：

- Python的 `os.path` 模塊提供了許多實用的函式來處理路徑，如 `join` 、 `split` 、 `exists` 、 `isfile` 和 `isdir` 等。
- `os.path.join` 函式可以智慧地構建路徑，自動處理不同操作系統間的分隔符差異。

5. **路徑規範化** ：

- 使用 `os.path.normpath` 函式可以將路徑規範化，以解決不一致的分隔符號問題，並簡化路徑。

理解這些基本概念對於在Python中有效地處理檔案和目錄路徑非常重要。

2.13 函數定義與調用

在Python中，函數是一種將程式碼組織成可重複使用的塊的方式。學習函數對於成為一名有效率的Python程式設計師非常重要。以下是關於Python函數的基本概念：

函數定義

- **函數定義** 開始於關鍵字 `def` ，後跟函數名稱和括號。
- 括號內可以包含參數，參數用於從函數外部傳遞數據到函數內部。
- 函數體是縮排的塊，包含要執行的語句。

基本範例

```
def greet(name):
    print("Hello, " + name + "!")
```

這個函數名稱為 `greet` ，接受一個名為 `name` 的參數，並打印一個問候語。

調用函數

要使用函數，你需要**調用** 它，並根據需要提供必要的參數。

```
greet("Alice")
```

返回值

- 函數可以通過 `return` 語句返回值。
- 如果未指定 `return` ，函數將默認返回 `None` 。

返回值範例

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

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

函數參數

在 Python 中，函數參數是函數定義的重要組成部分，它們為函數提供了必要的輸入信息。理解不同類型的函數參數對於寫出靈活且強大的函數至關重要。

基本函數參數

1. **位置參數（Positional Arguments）**：

- 這些參數根據它們的位置被傳遞給函數。
- 它們必須以正確的順序傳入。

```
def function(a, b):
    # a 和 b 是位置參數
    return a + b

function(2, 3)  # 正確的調用
```

2. **關鍵字參數（Keyword Arguments）**：

- 這些參數是根據參數名稱傳遞給函數的。
- 可以不按順序傳入，因為 Python 解釋器可以用名稱識別它們。

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

function(a=2, b=3)  # 關鍵字參數的調用
function(b=3, a=2)  # 順序可以不同
```

預設參數值

- 可以為函數參數提供預設值。
- 當調用函數時，如果未傳遞該參數，則使用預設值。

```
def function(a, b=4):
    return a + b

function(2)    # 使用預設值 b=4，輸出: 6
function(2, 3) # 覆蓋預設值，輸出: 5
```

可變參數

1. **不定長度參數（Arbitrary Arguments）**：

- 使用 `*args` 接受不定長度的位置參數。
- `args` 是一個包含所有未命名位置參數的元組。

```
def function(*args):
    return sum(args)

function(1, 2, 3)  # 輸出: 6
```

2. **不定長度關鍵字參數（Arbitrary Keyword Arguments）**：

- 使用 `**kwargs` 接受不定長度的關鍵字參數。
- `kwargs` 是一個包含所有未命名關鍵字參數的字典。

```
def function(**kwargs):
    return kwargs

function(a=1, b=2)  # 輸出: {'a': 1, 'b': 2}
```

總結

- **位置參數** 依賴於參數的位置。

- **關鍵字參數** 依賴於參數的名稱。
- 可以為參數指定**預設值** 。
- 使用 `*args` 和 `**kwargs` 接受**不定數量的參數** 。

了解這些函數參數的使用方式可以幫助你寫出更靈活和強大的函數，從而使你的 Python 程式更加有效和易於維護。

2.14 模組

模組（Module）在 Python 中是一種重要的功能，它允許你將代碼組織成可重用的部分。讓我們逐步了解模組的概念和用法。

模組的基本概念

1. **什麼是模組？** ：模組其實就是一個包含 Python 定義和聲明的文件。文件名就是模組的名字加上 `.py` 擴展名。
2. **模組的用途** ：模組讓你能夠邏輯地組織你的 Python 代碼段。把相關的代碼分組到一個模組中，使得代碼更加容易理解和使用。
3. **重用代碼** ：模組可以被其他部分的代碼導入，以使用該模組中的函數、類別等。

模組的使用

1. **導入模組** ：使用 `import` 關鍵字可以導入模組。例如：`import math` 導入了 Python 的標準數學模組。
2. **使用模組** ：當模組被導入後，你可以使用 `模組名.函數名` 的方式調用其中的函數。

假設我們有一個名為 `my_module.py` 的文件，內容如下：

```
def say_hello(name):
    print(f"Hello, {name}!")
```

你可以在另一個文件中導入並使用這個模組：

```
import my_module

my_module.say_hello("Alice")  # 輸出: Hello, Alice!
```

在 Python 中，`import`，`from` 和 `as` 是用於模組（modules）和套件（packages）導入的關鍵字。它們使得你可以在你的程式碼中使用其他模組或套件中的功能。下面分別對它們進行簡單說明：

1. **import** ：

- 用途：導入整個模組或套件。
- 語法：`import module_name`。
- 例子：`import math`。這樣做之後，你可以使用 `math.sqrt(4)` 這樣的語法來呼叫 `math` 模組中的函數。

2. **from ... import ...** ：

- 用途：從一個模組中導入特定的部分（比如函數、類或變數）。
- 語法：`from module_name import some_function`。
- 例子：`from math import sqrt`。這樣做之後，你可以直接使用 `sqrt(4)` 而不需要 `math.` 前綴。

3. **as** ：

- 用途：給導入的模組或者其部分提供一個別名。
- 語法：`import module_name as alias` 或者 `from module_name import some_function as alias`。
- 例子：`import math as m`。使用這種方式後，你可以用 `m.sqrt(4)` 來呼叫 `sqrt` 函數。

創建自己的模組

1. **創建模組** ：任何 Python 文件都可以作為一個模組，你只需將你的函數和定義放在一個 `.py` 文件中。
2. **導入模組** ：使用 `import` 語句加上你的文件名（不含 `.py`）來導入這個模組。

模組是 Python 中用於組織和重用代碼的一個強大工具。它們幫助你將代碼分割成小的部分，使得代碼更加模組化、易於維護和重用。无论是使用 Python 的標準庫中的模組，還是創建你自己的模組，都是提高代碼質量和開發效率的好方法。

常用模組

1. `math` 模組,

這個模組提供了一組廣泛的數學函數，適用於浮點數計算。

- `math.sqrt(x)`：計算 `x` 的平方根。
- `math.sin(x)`，`math.cos(x)`，`math.tan(x)`：分別計算正弦、餘弦和正切。
- `math.log(x[, base])`：計算 `x` 的自然對數，可選擇底數 `base`。
- `math.factorial(x)`：計算 `x` 的階乘。
- `math.pow(x, y)`：計算 `x` 的 `y` 次方。

2. `cmath` 模組

這個模組提供了一些用於復數計算的數學函數。

- `cmath.sqrt(x)`：計算復數 `x` 的平方根。
- `cmath.exp(x)`：計算 `e` 的 `x` 次方。
- `cmath.polar(x)`：將復數轉換為其極坐標形式。
- `cmath.rect(r, phi)`：將極坐標形式轉換為復數形式。
- `cmath.phase(x)`：返回復數的相位角。

3. `os` 模組

這個模組提供了與操作系統交互的功能。

- `os.listdir(path)`：列出指定目錄下的文件和目錄名。
- `os.getcwd()`：獲取當前工作目錄。
- `os.mkdir(path)`：創建新目錄。
- `os.remove(path)`：刪除文件。
- `os.path.join(path1[, path2[, ...]])`：將多個路徑組合後返回。

4. `matplotlib` 模組

主要用於數據可視化，繪製圖表。

- `matplotlib.pyplot.plot(x, y)`：繪製 x 和 y 的線圖。
- `matplotlib.pyplot.scatter(x, y)`：繪製 x 和 y 的散點圖。
- `matplotlib.pyplot.hist(x)`：繪製 x 的直方圖。
- `matplotlib.pyplot.xlabel(s)`，`matplotlib.pyplot.ylabel(s)`：分別設置 x 軸和 y 軸的標籤。
- `matplotlib.pyplot.title(s)`：設置圖表的標題。

2.15 類別與物件

在 Python 中，類別（Class）和物件（Object）是面向對象編程（Object-Oriented Programming, OOP）的核心概念。

以下是一些 Python 類命名的基本規則和最佳實踐：

1. **首字母大寫** ：類名通常使用駝峰式大小寫（CamelCase），即每個單詞的首字母大寫，例如 `MyClass`。
2. **避免使用內置名稱** ：不要使用像 `list`、`str` 這樣的 Python 內置名稱作為類名。
3. **清晰易懂** ：類名應該簡潔且描述性強，能夠清楚地表達類的功能或用途。
4. **底線的使用** ：底線可以用於改善可讀性，例如 `My_Class`，或者在某些特殊情況下，例如避免與 Python 的關鍵字衝突（雖然這種情況很少見）。

類別（Class）

1. **定義** ：類別是一種定義了特定屬性和方法的數據結構。你可以將類別視為創建物件的模板或藍圖。
2. **組件** ：

- **屬性（Attributes）** ：屬性是類別內部定義的變數，用於存儲數據。
- **方法（Methods）** ：方法是類別內部定義的函數，用於描述類別的行為或操作數據。

基本語法：

```
class ClassName:
    def __init__(self, parameters):
        # 初始化屬性
    def method1(self, parameters):
        # 方法的實現
```

物件（Object）

1. **定義** ：物件是根據類別創建的實例。每個物件都有特定的屬性和方法，與其他物件相互獨立。
2. **創建** ：物件是通過調用類別並傳遞初始化參數來創建的。

創建物件的語法：

```
objectName = ClassName(parameters)
```

例子

考慮一個簡單的類別 `Person` ，它有姓名（name）和年齡（age）兩個屬性，以及一個介紹自己的方法：

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def introduce(self):
        print(f"Hello, my name is {self.name} and I am {self.age} years old.")

# 創建一個 Person 物件
person1 = Person("Alice", 30)

# 使用物件的方法
person1.introduce()  # 輸出: Hello, my name is Alice and I am 30 years old.
```

在這個例子中：

- `Person` 是一個類別。
- `__init__` 是一個特殊的方法，稱為構造函數，用於初始化物件的屬性。
- `person1` 是 `Person` 類別的一個實例（或物件）。
- `introduce` 是一個定義在 `Person` 類別中的方法，用於顯示個人資訊。

一切皆物件

在 Python 中，的確「一切皆物件」（Everything is an Object）。這是 Python 語言的一個核心哲學，意味著所有 Python 中的數據類型都是以物件（Object）的形式存在。這包括了基本的數據類型（如整數、浮點數、字符串），以及函數、類、模組等。

物件的特性

1. **身份** ：每個物件都有一個唯一的身份（ID），可以用 `id()` 函數來查看。
2. **類型** ：物件的類型決定了它可以進行哪些操作，或者它擁有哪些方法。類型可以用 `type()` 函數來查看。
3. **值** ：物件所代表的數據或信息。對於某些類型的物件（如列表），它們的值可以變更（mutable），而對於其他類型（如整數、字符串）則不可以（immutable）。

例子

- 整數物件：

```
x = 100
print(type(x))  # 輸出: <class 'int'>
print(id(x))    # 輸出: (x 的唯一身份)
```

- 字符串物件：

```
y = "Hello"
print(type(y))  # 輸出: <class 'str'>
print(id(y))    # 輸出: (y 的唯一身份)
```

- 函數也是物件：

```python
def my_function():
    print("Hello World")

print(type(my_function))  # 輸出: <class 'function'>
print(id(my_function))    # 輸出: (my_function 的唯一身份)
```

- 類自定義的物件：

```python
class MyClass:
    pass

my_obj = MyClass()
print(type(my_obj))  # 輸出: <class '__main__.MyClass'>
print(id(my_obj))    # 輸出: (my_obj 的唯一身份)
```

> Python 中的「一切皆物件」概念是指一切都是物件實例，無論是數據、函數還是類。這種設計使得 Python 變得非常靈活和動態，因為它允許函數和類等實體具有與基本數據類型相同的性質。這也意味著在 Python 中，你可以對函數進行像對待普通數據一樣的操作，比如將函數作為參數傳遞，或者將它們賦值給變數等。

2.16 關鍵字與內建函數

在 Python 中，關鍵字（keywords）是一些被語言預先定義的、具有特殊意義的單詞。這些關鍵字不能用作變量名、函數名或任何其他標識符名稱。截至 2023 年，Python 3 的關鍵字如下：

1. `False`：布林值假。
2. `None`：表示空值或無。
3. `True`：布林值真。
4. `and`：邏輯運算符「與」。
5. `as`：用於建立別名。
6. `assert`：用於調試目的的斷言語句。
7. `async`：定義異步函數的關鍵字。
8. `await`：在異步函數中等待一個異步操作完成。
9. `break`：跳出當前循環。
10. `class`：用於定義類。
11. `continue`：跳到當前循環的下一次迭代。
12. `def`：用於定義函數。
13. `del`：用於刪除對象。
14. `elif`：用於 if 語句中，表示「否則如果」。
15. `else`：用於 if 或循環語句中，表示「否則」。
16. `except`：用於捕獲異常。
17. `finally`：無論是否有異常發生，都會執行的代碼塊。
18. `for`：用於 for 循環。
19. `from`：用於導入模塊的特定部分。
20. `global`：聲明全局變量。
21. `if`：用於 if 語句。
22. `import`：用於導入模塊。
23. `in`：檢查某元素是否在某序列中。
24. `is`：檢查兩個引用是否指向同一對象。
25. `lambda`：用於創建匿名函數。
26. `nonlocal`：在函數中聲明非局部變量。
27. `not`：邏輯運算符「非」。

28. `or`：邏輯運算符「或」。
29. `pass`：一個空操作/佔位語句。
30. `raise`：引發異常。
31. `return`：從函數返回值。
32. `try`：用於捕獲和處理異常。
33. `while`：用於 while 循環。
34. `with`：用於簡化異常處理。
35. `yield`：從函數返回一個生成器。

要查看你目前 Python 環境中的所有關鍵字，可以使用以下代碼：

```
import keyword
print(keyword.kwlist)
```

這將列出當前 Python 版本中所有的關鍵字。這些關鍵字構成了 Python 語言的基本結構，並且在編寫 Python 程式時扮演著關鍵角色。了解並熟悉這些關鍵字對於學習 Python 非常重要。

內建函數

在 Python 中，除了關鍵字（keywords）之外，還有一些內建函數（built-in functions）和內建常量（built-in constants），這些都是 Python 標準庫的一部分。這些函數和常量可以在不導入任何模塊的情況下直接使用，它們對於日常的 Python 編程非常有用。截至 2023 年，Python 的一些主要內建函數和常量如下：

常用內建函數

1. `print()`：打印輸出。
2. `len()`：返回對象（如字符串、列表、字典）的長度。
3. `type()`：返回對象的類型。
4. `int()`：將一個值轉換為整數。
5. `float()`：將一個值轉換為浮點數。
6. `str()`：將一個值轉換為字符串。
7. `list()`：創建一個新的列表。
8. `dict()`：創建一個新的字典。

9. `set()`：創建一個新的集合。
10. `input()`：獲取用戶輸入。
11. `range()`：生成一個數值範圍。
12. `sum()`：計算數字序列的總和。
13. `max()`：返回序列中的最大值。
14. `min()`：返回序列中的最小值。
15. `sorted()`：返回已排序的序列。
16. `open()`：打開一個文件並返回對應的文件對象。
17. `abs()`：返回數字的絕對值。
18. `round()`：四捨五入數字。
19. `zip()`：將多個序列的元素配對。
20. `enumerate()`：同時返回索引和元素的序列。
21. `filter()`：過濾序列。
22. `map()`：對序列的每個元素應用函數。
23. `globals()`：返回當前全局符號表。
24. `locals()`：返回當前局部符號表。
25. `dir()`：列出對象的屬性和方法。
26. `help()`：調用內置的幫助系統。

內建常量

1. `True`、`False`：布林值。
2. `None`：表示空值。
3. `Ellipsis` 或 `...`：用於切片語法。
4. `NotImplemented`：表示某個操作沒有實現。

這些內建函數和常量為 Python 編程提供了基本的功能，無需額外導入模塊即可使用。對於初學者來說，熟悉這些內建函數和常量將有助於提高編碼效率和理解 Python 程序。