

1.6. Operator



Fig. 1.6.1 Fig. 1.6.1 Photo by Charanjeet Dhiman on Unsplash



Outline

1. Introduction

2. Arithmetic Operator

- a. Example 1: All Operations
- b. Example 2: Elf Coins
- c. Example 3: Quotient and Remainder

3. Logic Operator

- a. Short Circuit Principle
- b. Example 1: and, or, not

4. Compare Operator

- a. Example 1: Comparison
- b. Example 2: Short Circuit
- c. Example 3: Prices

5. Identity Operator

- a. Example 1: is vs. ==: List
- b. Example 2: is vs. ==: String

6. Membership Operator

- a. Example 1: in
- b. Example 2: More in

7. Assignment Operator

- a. Example 1: Market

8. Bitwise Operator

- a. Example 1: Bitwise

9. Other Operator

- a. Example 1: Pi

10. Operator Precedence

11. Exercise

Roadmap

1. This topic: Operator

myMaze, myBoard

Operator							
Arithmetic Operator	Logic Operator	Comparison Operator	Assignment Operator	Identity Operator	Membership Operator	Bitwise Operator	Other Operator
Operator Precedence							

2. Course: Python 1

3. Subject: Programming

4. Field

- a. Software Engineering (SE)
- b. Computer Science and Information Engineering (CSIE)
- c. Electrical/Electronics Engineering (EE)

1.6.1. Introduction

- 1. 一般而言，大部分的程式碼都是由判斷式及運算式組成的。
- 2. 但此時先不談判斷式，我們先了解一下運算式。
- 3. 1 + 1 = 2 這種國小就有教過的式子就是運算式。
- 4. 運算式是由運算子及運算元所組成，運算子就像是這個運算的種類（如+, = 等），運算元(Operand)則是要被用來運算的資料（如1, 2, integer 等）。

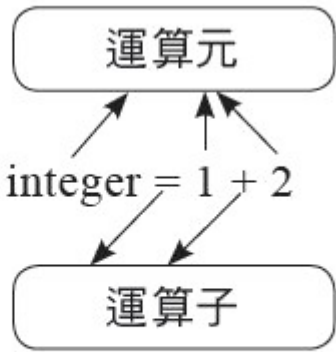


圖 3-3 運算子與運算元之可能關係

- 5. 一般來說，運算子可以分成：
- a. 算術運算子
- b. 指定運算子
- c. 比較運算子
- d. 邏輯運算子: a.k.a., 布林運算子
- e. 其他運算子

6. 以下幾個小節將對各類型的運算子作更詳細的說明。

1.6.2. Arithmetic Operator

- 1. Arithmetic Operator = 算術運算子
- 2. 算術運算子相當容易理解，就是一般數學式中常用的加減乘除，只是再加上幾個讓程式邏輯更方便的運算子，其所代表意義如下所示。

運算子	意義
+	加法運算子，執行兩運算元之加法。
-	減法運算子，執行兩運算元之減法。
*	乘法運算子，執行兩運算元之乘法。
**	指數運算子。
/	除法運算子，執行兩運算元之除法。
//	整數除法運算子，出來的結果會自動取整數，取商數
%	模數運算子，取餘數

1.6.2.1. Example 1: All Operations

1. Code+Output

p0211OpArithm.py Output

Listing 1.6.2.1.1

```
/src/Operator/p0211OpArithm.py
1 print(2 + 2)    # 加法運算子
2 print(2 - 2)    # 減法運算子
3 print(6 * 6)    # 乘法運算子
4 print(6 ** 6)   # 指數運算子
5 print(7 / 2)    # 除法運算子
6 print(7 // 2)   # 整數除法運算子，取商數
7 print(7 % 2)    # 模數運算子，取餘數
8 print(120 % 7)  # 模數運算子，取餘數
```

- 2. 在這裡要特別注意的就是`[**]`、`[/]`、`[%]`這三個運算子：`[**]`就是次方，也就是所謂的指數，因此可以看到上述程式碼第三行和第四行的差異，其實是乘法和指數的不同。`6 ** 6`就輸出6的6次方，即46656。
- 3. 而`[/]`是表示整數除法，也就是執行完除法之後的結果直接做無條件捨去小數部分，只保留整數部份。所以，`7 / 2`會輸出3.5，但是`7 // 2`則是會輸出3，而不是輸出四捨五入之後的4。
- 4. 最後一個`[%]`就更少用到了，這個百分號被稱為`[模數]`，是拿來取餘數用的。

5. 例如，上面的例子 $120 \% 7$ ，可以看成「有120顆糖果，分給七個小朋友，每個人分到的要一樣多，最後會剩幾顆？」這個题目的算法，電腦看到這個式子會自動去把 $120/7$ 所得到的餘數輸出，當然這樣的餘數應該是小於7的數值，也就是介於0到6之間的某一個數值。

1.6.2.2. Example 2: Elf Coins

1. 菲絲恩帶著普羅來到市場並拿出了5000精靈幣，「這個是我們世界的貨幣，讓我來替你裝扮一下，挑選一套可在比賽當天穿的服裝。走！讓我們到裡面去採買一番吧！」

2. Code+Output

p0211OpArithm2.py

Output

Listing 1.6.2.2.1

/src/Operator/p0211OpArithm2.py

```
1 elfCoins = 50 # 一開始有50精靈幣
2 elfCoins = elfCoins - (10 + 15) # 買了價值10精靈幣的上衣和15精靈幣的披風
3 elfCoins = elfCoins - 20 * 0.75 # 20精靈幣的帽子和鞋子組合，特價75折
4 print("還剩下", elfCoins, "精靈幣") # 印出剩餘精靈幣的數量
```

1.6.2.3. Example 3: Quotient and Remainder

1. Code+Output

p0211OpArithm-20210708.py

Output

Listing 1.6.2.3.1

/src/Operator/p0211OpArithm-20210708.py

```
1 print(7 // 2)
2 print(7 % 2)
3
4 iX = 1 + 2 * 3 / 4 ** 5 // 6 * 7
5 print(iX)
```

1.6.3. Logic Operator

1. Logic Operator = 邏輯運算子

2. 邏輯運算子又稱為布林運算子，顧名思義就是跟邏輯有關，其所代表意義如下所示。

運算子	例子	意義
and	a and b	若a為假則回傳a；若a為真則回傳b。
or	a or b	若a為假則回傳b；若a為真則回傳a。
not	not a	若a為假則回傳True；若a為真則回傳False。

1.6.3.1. Short Circuit

1. The judgment sequence can be speed-up by short circuit principle.
2. That is, we let the better judgment, the fore-most sequence location be put. [越易越快且出現越多之判斷者，放在越前面進行判斷。]
3. What is the better judgment?
 - a. The judgment with higher appearance possibility [較高出現機率的判斷句]
 - b. The judgment with quicker operations [較快速完成的判斷句]
 - c. The judgment with using less computation resource [使用較少計算資源的判斷句]

```
1  #-----
2  # a's appearance possibility of judgment is 80%
3  # b's appearance possibility of judgment is 10%
4  # c's appearance possibility of judgment is 7%
5  # d's appearance possibility of judgment is 3%
6  #-----
7  # Good codes
8  if (a and b and c and d):
9      # codes...
10     pass
11
12 # Worse codes
13 if (c and d and a and b):
14     # codes...
15     pass
16
17 # Worst codes
18 if (d and c and b and a):
19     # codes...
20     pass
```

1.6.3.2. Example 1: and, or, not

1. Code+Output

p0211OpLogic.py

Output

Listing 1.6.3.2.1

/src/Operator/p0211OpLogic.py

```
1  a, b = 0, (1, 2)    # 給予a, b 不同的值
2  print(a or b)       # 印出a or b 的結果
3  print(a and b)      # 印出a and b 的結果
4  print(not a)        # 印出not a 的結果
5  print(not a and b)  # 印出not a and b 的結果
```

2. 邏輯運算條件式的真值表

條件式	a, b皆為真	a為假,b為真	a為真,b為假	a, b皆為假
a and b	b	a	b	a

條件式	a, b皆為真	a為假,b為真	a為真,b為假	a, b皆為假
a or b	a	b	a	b
not a	False	True	False	True
not b	False	False	True	True

Important

Exclusive OR (XOR; 互斥)

Listing 1.6.3.2.2

/src/Operator/p0211OpLogic-20210709.py

```
1  #-----
2  # Exclusive OR (XOR; 互斥)
3  #
4  #      0    1    <-- a
5  #    +---+---+
6  #    0 | 0 | 1 |
7  #    +---+---+
8  #    1 | 1 | 0 |
9  #    +---+---+
10 #    ^
11 #    |
12 #    b
13 #
14 # Formula[口訣]: 有你就沒有我, 誓不兩立
15 #
16 #-----
```

1.6.4. Compare Operator

1. Compare Operator = 比較運算子
2. 比較運算子又可稱為關係運算子，是將兩個運算元拿來相互比較，以得知兩者之間的關係。
3. 值得注意的是，[=]所代表的意義是賦值，所以原本[相等]的意義就以[==]雙等號來表示，而經過比較運算子運算後所得出的結果則是布林值。

Important

<> vs. !=

- a. <>: 是v2.x的不等於符號，已廢棄不用
- b. !=: 是v3.x的不等於符號。

1.6.4.1. Example 1: Comparison

1. Code+Output

Listing 1.6.4.1.1

```
/src/Operator/p0211OpCompare.py
```

```
1 print(2 == 2)    # 相等
2 print(2 != 2)    # 不相等
3 print(6 > 4)     # 大於
4 print(4 < 6)     # 小於
5 print(6 >= 3)    # 大於等於
6 print(6 <= 3)    # 小於等於
7 print(2 = 2)     # 這是賦值，不是比較，小心！
```

1.6.4.2. Example 2: Short Circuit

1. 比較運算子可以用在判斷變數的值是否在一個範圍之內，例如：

2. Code+Output

Listing 1.6.4.2.1

```
/src/Operator/p0211OpCompare-20220708.py
```

```
1 integer = 7
2 print(5 < integer <= 10)
3 print(5 < integer and integer <= 10)    # 等同上式
4 print(5 < integer or integer <= 10)     # 只要第一個判斷成立即為真
5
6 integer = 17
7 print(5 < integer <= 10)
8 print(5 < integer and integer <= 10)    # 等同上式
9 print(5 < integer or integer <= 10)     # 只要第一個判斷成立即為真
```

1.6.4.3. Example 3: Prices

1. 到了中午，菲絲恩與普羅到了一個巷口，兩側都是牛肉麵店。
2. 山珍麵店的老闆說：「我這邊的牛肉麵原價一碗500精靈幣，現在打55折給你。」
3. 而海味麵店的老闆則說：「我這邊的牛肉麵一碗400精靈幣，再打6折給你。」
4. 普羅認為山珍麵店比較便宜，因為它打了55折。
5. 菲絲恩說：「別急，讓我來算給你看。」

6. Code+Output

Listing 1.6.4.3.1

```
/src/Operator/p0211OpCompare2.py
```



```

1 land = 500 * 0.55 # 山珍麵店打折後的價格
2 sea = 400 * 0.6   # 海味麵店打折後的價格
3 print(land < sea) # 判斷山珍麵店打折後的價格是否低於海味麵店打折後的價格

```

1.6.5. Identity Operator

1. Identity Operator = 身份運算子

2. `is` 是判斷兩個運算元是否「本質上」相等，並回傳`bool`值。
3. 而所謂「本質上」的相等，若指的是變數，則代表是否指向同一個物件。
4. 故兩個變數可以有完全相同的值，但是卻指向不同的物件，例如：

1.6.5.1. Example 1: `is` vs. `==`: List

1. Code+Output

p0211OpId.py Output

Listing 1.6.5.1.1

/src/Operator/p0211OpId.py

```

1 list1 = [1, "test"]      # 將一樣的值
2 list2 = [1, "test"]      # 賦予到兩個不同的變數上面
3
4 print(list1 == list2)    # 測試list1 跟list2 的值是否相等
5 print(list1 is list2)    # 測試list1 跟list2 的指向目標物件是否為同一個
6
7 list1 = list2             # 將list1 指向list2 所指向的物件
8
9 print(list1 == list2)    # 測試list1 跟list2 的值是否相等
10 print(list1 is list2)   # 測試list1 跟list2 的指向目標物件是否為同一個

```

1.6.5.2. Example 2: `is` vs. `==`: String

1. Code+Output

p0211OpId2.py Output

Listing 1.6.5.2.1

/src/Operator/p0211OpId2.py

```

1 string1 = "test"         # 同樣將一樣的值
2 string2 = "test"         # 賦予到兩個不同的變數上面
3
4 print(string1 == string2) # 測試string1 跟string2 的值是否相等
5 print(string1 is string2) # 測試string1 跟string2 指向目標物件是否為同一個

```

2. 這裡有個需要特別注意的地方，在將`test`這個內容賦值給`string1`跟`string2`時，照理說`string1`跟`string2`只有值一樣，所指向的物件應該是不同的。

3. 但是Python為了節省記憶體空間，因此會將使用比較簡單的資料型態且其內容相同的不同變數，在短時間內指向同一個物件。
4. 所以，這便造成上面list在測試時所指向物件雖然不同，string卻相同的情形。

1.6.6. Membership Operator

1. Membership Operator = 成員運算子
2. 而in的功能則是判斷一個元素是否為一集合的元素，適用範圍包含字串、集合、清單、序對和字典。
3. 除了字串外，其他幾個資料型態都屬於容器資料型態，類似集合的概念，所以可以使用in來判斷；但是字串也可以使用in，原因是因為字串可以視為由一連串長度為1的字串所組成的長字串。
4. 如此一來，字串也就具有類似集合的概念了，例如：

1.6.6.1. Example 1: in

1. Code+Output

p0211OpMembership.py

Output

Listing 1.6.6.1.1

/src/Operator/p0211OpMembership.py

```
1  容器 = ("白飯", "滷蛋", "排骨")    # 將元素賦值給容器
2  print("白飯" in 容器, "雞排" in 容器, "雞排" not in 容器)
```

1.6.6.2. Example 2: More in

1. Code+Output

p0211OpMembership2.py

Output

Listing 1.6.6.2.1

/src/Operator/p0211OpMembership2.py

```
1  tuple1 = (1, 2, "3", "4", [1, 2]) # 將多種不同型態的元素賦值給tuple1
2  list1 = [5, 6, "7", "8"]          # 將多種不同型態的元素賦值給list1
3  dictionary1 = {9: "b", 0: "d"}     # 將9、0當作key值，對應的值為b、d
4  set1 = {"x", "y", "z"}            # 將"x", "y", "z"當作元素賦值給set1
5  string1 = "abcdefghijklmn"        # 建立string字串開始測試in
6
7  print(1 in tuple1, 3 in tuple1, 9 not in tuple1)
8  print(5 in list1, 7 in list1, 'a' not in list1)
9  print('b' in dictionary1, 9 in dictionary1)
10 print('x' in set1, 10 in set1)
11 print('z' in string1, 'c' in string1)
```

1.6.7. Assignment Operator

- 1. Assignment Operator = 指派運算子
- 2. 指派運算子是將運算過後的結果儲存在某個變數中。
- 3. 使用指派運算子的好處是可以讓運算式變簡單，但是要看清楚其所代表的真實意義，如下表所示，以免用錯了還不自知。

運算子	例子	意義
=	a = b	將b的值賦予a
+=	a += b	a = a + b
-=	a -= b	a = a - b
*=	a *= b	a = a * b
/=	a /= b	a = a / b
//=	a //= b	a = a // b
%=	a %= b	a = a % b

1.6.7.1. Example 1: Market

- 1. 「我好久沒有來逛市集了，今天還真是大豐收呢。啊！我來教你一個能把計算剩餘精靈幣的魔法加以簡化的方法吧。」
- 2. 菲絲恩話一說完，普羅便驚訝地看著菲絲恩說：「你教我的那個咒語已經夠簡單的了，竟然還有更簡化的？！」
- 3. Code+Output

p0211OpAssignment.py Output

Listing 1.6.7.1.1

```
/src/Operator/p0211OpAssignment.py

1  elfCoins = 5000      # 一開始有5000 精靈幣
2  cost = 0             # 預設總支出為0
3  cost += 1000 + 1500  # 買了價值1000 精靈幣的上衣和1500 精靈幣的披風
4  cost += 2000 * 0.75  # 2000 精靈幣的帽子和鞋子組合，特價75 折
5  sea = 400            # 海味麵店的牛肉麵原價
6  sea *= 0.6           # 牛肉麵打折後的價格
7  cost += sea * 2       # 買了兩碗海味麵店的牛肉麵
8  elfCoins -= cost      # 剩餘的精靈幣 = 原來的精靈幣數量 - 總支出
9  print("還剩下", elfCoins, "精靈幣") # 印出剩餘精靈幣的數量
```

1.6.8. Bitwise Operator

- 1. Bitwise Operator = 位元運算子
- 2. 顧名思義，就是以位元為運算單元的運算子。

運算子	例子	意義
&	a & b	Binary AND.
	a b	Binary OR.
^	a ^ b	Binary XOR.
~	~ a	Binary 1's complement.
<<	a << n	Binary left shift.
>>	a >> n	Binary right shift.

1.6.8.1. Example 1: Bitwise


1. Code+Output

p0211OpBitwise.py

Output

Listing 1.6.8.1.1

```
/src/Operator/p0211OpBitwise.py
1  a = 13          # 12    = 1101
2  b = 7           # 7     = 0111
3  print(a & b)    # and   = 0101 = 5
4  print(a | b)    # or    = 1111 = 15
5  print(a ^ b)    # xor   = 1010 = 10
6
7  # 1's Complement; add 1 bit at left side for sign bit
8  print(~a)       # not a = 10010 = -16 + 2 = -14
9  print(~b)       # not b = 11000 = -16 + 8 = -8
10
11 # Shift left operation adds n bits at left side for storing shifted-bits
12 print(a << 2)    # sft l = 0011 0100 = 32+16+4 = 52
13 print(b << 2)    # sft l = 0001 1100 = 16+8+4 = 28
14
15 # Shift right operation removes n bits at right side
16 print(a >> 2)    # sft r = 0011 = 3
17 print(b >> 2)    # sft r = 0001 = 1
```

 **Important**

Complement [補數]

1. 1's Complement [1補數] [Web, Wikipedia]

2. 2's Complement [2補數] [Web, Wikipedia]

1.6.9. Other Operator

1. 除了上述所提到的四大類運算子之外，還有一些比較少用，且無法歸類到上述四大類運算子之中，故另外以下表為各位介紹。

運算子名稱	運算子	意義
逗號運算子	,	分隔變數、資料集裡的元素等等
分號運算子	;	分隔運算式
點號運算子	.	存取類別、模組的方法或屬性
小括弧運算子	()	定義tuple、函式/方法呼叫
中括弧運算子	[]	定義list、序列(Sequence)形態的索引符號
大括弧運算子	{}	定義dict, set
冒號運算子	:	控制條件後的分隔符號或辭典元素之配對

1.6.9.1. Example 1: Pi

- 1. 「我好久沒有來逛市集了，今天還真是大豐收呢。啊！我來教你一個能把計算剩餘精靈幣的魔法加以簡化的方法吧。」
- 2. 菲絲恩話一說完，普羅便驚訝地看著菲絲恩說：「你教我的那個咒語已經夠簡單的了，竟然還有更簡化的？！」

3. Code+Output

p0211OpOther.py

Output

Listing 1.6.9.1.1

```
/src/Operator/p0211OpOther.py

1  import math                # import math 模組
2
3  a, b = (10, 20)             # 定義a為10，而b為20
4  x = 3; y = 4; z = 5        # 定義x為3，y為4，z為5。 可以同時定義多個變數
5  print(math.pi)            # 呼叫math的pi這個屬性，也就是圓周率
6  tuple1 = (1, 2, 3, 4)      # 定義tuple1為(1, 2, 3, 4)
7  list1 = [5, 6, 7, 8]       # 定義list1為[5, 6, 7, 8]
8  dict1 = {'a': 1, 'b': 2}   # 定義dict1中'a'為1，'b'為2
9  print(list1[:3])           # 印出list1裡面第一個元素到第三個元素
```

- 4. 此外，還有一種特殊運算子稱為位元運算子，其主要功能在於處理位元資料的運算，但較少被使用，故僅在附錄A.5 介紹。
- 5. 但是，由於位元運算子在使用上會和二進位制有絕對的關係，為了避免普羅因對二進位制的不了解，而無法理解位元運算子的使用方式，故菲絲恩也將二進位制的觀念一併放進附錄A.4 中介紹，請普羅自行參閱。

1.6.10. Operator Precedence

- 1. 是指[運算子優先順序]
- 2. 因為可以把運算式拿來當運算子，因此常常出現一個運算式裡面有好幾個運算元。

3. 既然會出現這種情形，那就必須先釐清各種運算子之間的優先順序。
4. 在同一個運算式中優先權越高的，電腦便會優先執行該子運算式。
5. 也是大家小學學算術時，那種類似先加減後乘除的觀念。
6. 下表列出各運算子之優先權，由高至低排序：

運算子	說明
() , [] , {}	tuple 、 list 、 dict
a[i] , a[i:j] , a.b , a() , a.b()	內含中小括號之呼叫
+k , -k , ~k	正負數及補數
a*b , a/b , a%b , a//b	乘法、除法、取餘數、整數除法
a+b , a-b	加法、減法
a<<b , a>>b	位移運算
a&b	AND位元運算
a^b	XOR位元運算
a b	OR位元運算
< , > , <= , >= , == , <> , !=	比較運算 [其中<>是v2.x的不等於符號，已廢棄不用]
is , is not , in , not in	本體測試、成員關係
not a	not邏輯運算
a and b	and邏輯運算
a or b	or邏輯運算

1.6.11. Exercise

1.6.11.1. Ex. 1

1. Question: 若執行下列程式碼，最終result值應為多少？

2. Code+Output

Code

Output

```
1 num1 = 30
2 num2 = 9
3 result = num1 % num2
```

1.6.11.2. Ex. 2

1. Question: 若執行下列程式碼，最終result值應為多少？

2. Code+Output

Code Output

```
1 num1 = 4
2 num2 = 18
3 result = num2 // num1
4 result *= 2
```

1.6.11.3. Ex. 3

1. Code+Output

Question Code

- 1 在精靈銀行存錢每半年複利1次，半年利息是0.5%。
- 2 假設帕森現在開了一個新帳戶並存入1000元，則十年後帳戶裡應有多少錢？
- 3 (請撰寫一程式計算該結果並加以輸出)

1.6.11.4. Ex. 4

1. Code+Output

Question Code

- 1 1精靈幣與2.5新台幣等值，請問2000新台幣可以兌換多少精靈幣？
- 2 (請撰寫一程式計算該結果並加以輸出)

1.6.11.5. Ex. 5

1. Code+Output

Question Code

- 1 精靈國的計程車由500精靈幣開始跳表，
- 2 自上車後開始計算公里數，每開1公里加收1000精靈幣，
- 3 下車前再收5000精靈幣作為清潔費。

- 4 請問普羅搭乘計程車移動28公里應付多少錢？
- 5 (請撰寫一程式計算該結果並加以輸出)

1.6.11.6. Ex. 6

1. Code+Output

Question

Code

- 1 網路傳輸速度為 $1\text{M} = 128\text{kB/s}$ ，而檔案大小為 $1\text{MB} = 1024\text{kB}$ ，
- 2 在理想狀態下，以速度 100M 傳輸 256MB 的檔案需要花多少秒？
- 3 (請撰寫一程式計算該結果並加以輸出)

1.6.11.7. Ex. 7

1. Code+Output

Question

Code

- 1 請將海龍公式撰寫成Python程式碼，
- 2 並用其計算出三邊長分別為3，4，5之三角形的面積後輸出。

1.6.11.8. Ex. 8

1. Code+Output

Question

Code

- 1 修改Ex. 7的程式，讓使用者可利用系統參數的方式代入三角形的三邊長，並輸出三角形面積。

1.6.11.9. Ex. 9

1. Code+Output

Question

Code

請修正以下程式碼的運算子，使其能正確輸出 $1^3 + 3^3 = 28$ 。乘法公式: $a^3 + b^3 = (a + b)(a^2 - a * b + b^2)$

- 1 `a = 1`
- 2 `b = 3`
- 3 `Ans = (a + b)*(a^2 - a*b + b^2)`
- 4 `print("1^3 + 3^3 = ", Ans)`

1.6.11.10. Ex. 10

1. Code+Output

Question

Code

- 1 預設變數a = True且b = False。
- 2 請寫一個程式輸出a xor b的結果。
- 3 xor為互斥運算。
- 4 當兩個運算元值不同時輸出true。
- 5 其餘則輸出false。



Fig. 1.6.11.10.1 Photo by Randolfo Santos on Google Map, March 2022.

1. Start: 20120311

2. System Environment:

Listing 1.6.11.10.1

requirements.txt

```
1 sphinx>=6.1.3 # Sphinx
2 graphviz>=0.20.1 # Graphviz
3 sphinxbootstrap4theme>=0.6.0 # Theme: Bootstrap
4 sphinx-material>=0.0.35 # Theme: Material
5 sphinxcontrib-plantuml>=0.25 # PlantUML
6 sphinxcontrib.bibtex>=2.5.0 # Bibliography
7 sphinx-autorun>=1.1.1 # ExecCode: pycon
8 sphinx-execute-code-python3>=0.3 # ExecCode
9 btd.sphinx.inheritance-diagram>=2.3.1 # Diagram
10 sphinx-copybutton>=0.5.1 # Copy button
11 sphinx_code_tabs>=0.5.3 # Tabs
12 sphinx-immaterial>=0.11.3 # Tabs
13
14 #-----
15 #-- Minor Extension
16 #-----
17 sphinxcontrib.httpdomain>=1.8.1 # HTTP API
18
19 #sphinxcontrib-blockdiag>=3.0.0 # Diagram: block
20 #sphinxcontrib-actdiag>=3.0.0 # Diagram: activity
21 #sphinxcontrib-nwdiag>=2.0.0 # Diagram: network
22 #sphinxcontrib-seqdiag>=3.0.0 # Diagram: sequence
23
24 #-----
25 #-- Still Wait For Upgrading Version
26 #-----
27
28 #-----
29 #-- Still Under Testing
30 #-----
31 #numpy>=1.24.2 # Figure: numpy
32
33 #-----
34 #-- NOT Workable
35 #-----
36 #sphinxcontrib.jsdemo==0.1.4 # ExecCode: Need replace add_js_file()
37 #jupyter-sphinx==0.4.0 # ExecCode: Need gcc compiler
38 #sphinxcontrib.slide==1.0.0 # Slide: Slideshow
39 #hieroglyph==2.1.0 # Slide: make slides
40 #matplotlib>=3.7.1 # Plot: Need Python >= v3.8
41 #manim==0.17.2 # Diagram: scipy, numpy need gcc
42 #sphinx_diagrams==0.4.0 # Diagram: Need GKE access
43 #sphinx-tabs>=3.4.1 # Tabs: Conflict w/ sphinx-material
```