

✓ Functions in Python

Estimated time needed: **40** minutes

Objectives

After completing this lab you will be able to:

- Understand functions and variables
- Work with functions and variables

Functions in Python

Welcome! This notebook will teach you about the functions in the Python Programming Language. By the end of this lab, you'll know the basic concepts about function, variables, and how to use functions.

Table of Contents

- [Functions](#)
 - [What is a function?](#)
 - [Variables](#)
 - [Functions Make Things Simple](#)
 - [Pre-defined functions](#)
 - [Using if / else Statements and Loops in Functions](#)
 - [Setting default argument values in your custom functions](#)
 - [Global variables](#)
 - [Scope of a Variable](#)
 - [Collections and Functions](#)
 - [Quiz on Loops](#)
-

Functions

A function is a reusable block of code which performs operations specified in the function. They let you break down tasks and allow you to reuse your code in different programs.

There are two types of functions :

- **Pre-defined functions**
- **User defined functions**

What is a Function?

You can define functions to provide the required functionality. Here are simple rules to define a function in Python:

- Functions blocks begin `def` followed by the function name and parentheses `()`.
- There are input parameters or arguments that should be placed within these parentheses.
- You can also define parameters inside these parentheses.
- There is a body within every function that starts with a colon `:` and is indented.
- You can also place documentation before the body.
- The statement `return` exits a function, optionally passing back a value.

An example of a function that adds on to the parameter `a` prints and returns the output as `b` :

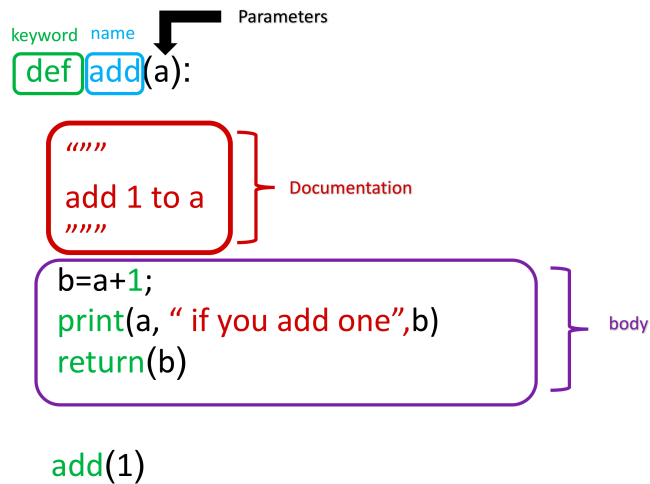
```
1 # First function example: Add 1 to a and store as b
2
```

```

3 def add(a):
4     """
5     add 1 to a
6     """
7     b = a + 1
8     print(a, "if you add one", b)
9     return(b)

```

The figure below illustrates the terminology:



We can obtain help about a function :

```

1 # Get a help on add function
2
3 help(add)

```

We can call the function:

```

1 # Call the function add()
2
3 add(1)

```

If we call the function with a new input we get a new result:

```

1 # Call the function add()
2
3 add(2)

```

We can create different functions. For example, we can create a function that multiplies two numbers. The numbers will be represented by the variables a and b :

```

1 # Define a function for multiple two numbers
2
3 def Mult(a, b):
4     c = a * b
5     return(c)
6     print('This is not printed')
7
8 result = Mult(12,2)
9 print(result)

```

The same function can be used for different data types. For example, we can multiply two integers:

```

1 # Use mult() multiply two integers
2
3 Mult(2, 3)

```

Note how the function terminates at the `return` statement, while passing back a value. This value can be further assigned to a different variable as desired.

The same function can be used for different data types. For example, we can multiply two integers:

Two Floats:

```

1 # Use mult() multiply two floats
2
3 Mult(10.0, 3.14)

```

We can even replicate a string by multiplying with an integer:

```

1 # Use mult() multiply two different type values together
2
3 Mult(2, "Michael Jackson ")

```

Variables

The input to a function is called a formal parameter.

A variable that is declared inside a function is called a local variable. The parameter only exists within the function (i.e. the point where the function starts and stops).

A variable that is declared outside a function definition is a global variable, and its value is accessible and modifiable throughout the program.

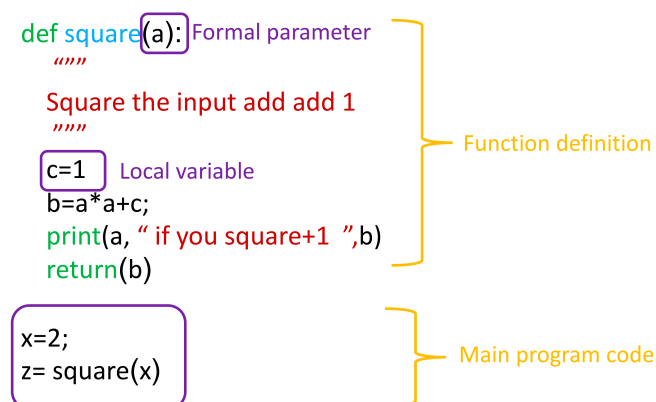
We will discuss more about global variables at the end of the lab.

```

1 # Function Definition
2
3 def square(a):
4
5     # Local variable b
6     b = 1
7     c = a * a + b
8     print(a, "if you square + 1", c)
9     return(c)

```

The labels are displayed in the figure:



We can call the function with an input of 3:

```

1 # Initializes Global variable
2
3 x = 3
4 # Makes function call and return function a y

```

```
5 y = square(x)
6 y
```

We can call the function with an input of **2** in a different manner:

```
1 # Directly enter a number as parameter
2
3 square(2)
```

If there is no `return` statement, the function returns `None`. The following two functions are equivalent:

```
1 # Define functions, one with return value None and other without return value
2
3 def MJ():
4     print('Michael Jackson')
5
6 def MJ1():
7     print('Michael Jackson')
8     return(None)
```

```
1 # See the output
2
3 MJ()
```

```
1 # See the output
2
3 MJ1()
```

Printing the function after a call reveals a **None** is the default return statement:

```
1 # See what functions returns are
2
3 print(MJ())
4 print(MJ1())
```

Create a function `con` that concatenates two strings using the addition operation:

```
1 # Define the function for combining strings
2
3 def con(a, b):
4     return(a + b)
```

```
1 # Test on the con() function
2
3 con("This ", "is")
```

[Tip] How do I learn more about the pre-defined functions in Python?

We will be introducing a variety of pre-defined functions to you as you learn more about Python. There are just too many functions, so there's no way we can teach them all in one sitting. But if you'd like to take a quick peek, here's a short reference card for some of the commonly-used pre-defined functions: [Reference](#)

Functions Make Things Simple

Consider the two lines of code in **Block 1** and **Block 2**: the procedure for each block is identical. The only thing that is different is the variable names and values.

Block 1:

```

1 # a and b calculation block1
2
3 a1 = 4
4 b1 = 5
5 c1 = a1 + b1 + 2 * a1 * b1 - 1
6 if(c1 < 0):
7     c1 = 0
8 else:
9     c1 = 5
10 c1

```

Block 2:

```

1 # a and b calculation block2
2
3 a2 = 0
4 b2 = 0
5 c2 = a2 + b2 + 2 * a2 * b2 - 1
6 if(c2 < 0):
7     c2 = 0
8 else:
9     c2 = 5
10 c2

```

We can replace the lines of code with a function. A function combines many instructions into a single line of code. Once a function is defined, it can be used repeatedly. You can invoke the same function many times in your program. You can save your function and use it in another program or use someone else's function. The lines of code in code **Block 1** and code **Block 2** can be replaced by the following function:

```

1 # Make a Function for the calculation above
2
3 def Equation(a,b):
4     c = a + b + 2 * a * b - 1
5     if(c < 0):
6         c = 0
7     else:
8         c = 5
9     return(c)

```

This function takes two inputs, a and b, then applies several operations to return c. We simply define the function, replace the instructions with the function, and input the new values of a1, b1 and a2, b2 as inputs. The entire process is demonstrated in the figure:

```

a1=5
b1=5
c1=a1+b1+2*a1*b1-1
if(c1<0):
    c1=0
else:
    c1=5

```

✱ / ✱ ✱ ✱

Code **Blocks 1** and **Block 2** can now be replaced with code **Block 3** and code **Block 4**.

Block 3:

```
1 a1 = 4
2 b1 = 5
3 c1 = Equation(a1, b1)
4 c1
```

Block 4:

```
1 a2 = 0
2 b2 = 0
3 c2 = Equation(a2, b2)
4 c2
```

Pre-defined functions

There are many pre-defined functions in Python, so let's start with the simple ones.

The `print()` function:

```
1 # Build-in function print()
2
3 album_ratings = [10.0, 8.5, 9.5, 7.0, 7.0, 9.5, 9.0, 9.5]
4 print(album_ratings)
```

The `sum()` function adds all the elements in a list or tuple:

```
1 # Use sum() to add every element in a list or tuple together
2
3 sum(album_ratings)
```

The `len()` function returns the length of a list or tuple:

```
1 # Show the length of the list or tuple
2
3 len(album_ratings)
```

In-Built functions

In Python, an in-built function is a pre-defined function that is always available for use, providing common functionality without requiring any imports.

```
1 #You will see below will return an error as integer alone is not considered while using a function.It (
2
3 sum(1,2)

1 # Define a tuple
2 a = (1, 2)
3
4 # Pass the tuple to the sum function and store the result in a variable
5 c = sum(a)
6
7 # Print the result
8 print(f"The sum of the elements in the tuple {a} is {c}.")
9
```

```

1 # Define a list
2 a = [1, 2]
3
4 # Pass the list to the sum function and store the result in a variable
5 c = sum(a)
6
7 # Print the result
8 print(f"The sum of the elements in the list {a} is {c}.")
9

```

Using if/else Statements and Loops in Functions

The `return()` function is particularly useful if you have any IF statements in the function, when you want your output to be dependent on some condition:

```

1 # Function example
2
3 def type_of_album(artist, album, year_released):
4
5     print(artist, album, year_released)
6     if year_released > 1980:
7         return "Modern"
8     else:
9         return "Oldie"
10
11 x = type_of_album("Michael Jackson", "Thriller", 1980)
12 print(x)

```

We can use a loop in a function. For example, we can `print` out each element in a list:

```

1 # Print the list using for loop
2
3 def PrintList(the_list):
4     for element in the_list:
5         print(element)
6
7
8 # Implement the printlist function
9
10
11 PrintList(['1', 1, 'the man', "abc"])

```

String comparison in Functions

The relational operators compare the Unicode values of the characters of the strings from the zeroth index till the end of the string. It then returns a boolean value according to the operator used.

```

1 #Compare Two Strings Directly using in operator
2 # add string
3 string= "Michael Jackson is the best"
4
5 # Define a function
6 def check_string(text):
7
8 # Use if else statement and 'in' operator to compare the string
9     if text in string:
10         return 'String matched'
11     else:
12         return 'String not matched'
13
14 check_string("Michael Jackson is the best")

```

This program uses a user-defined function named `compareStrings()` to compare two strings.

This function receives both strings as its argument and returns 1 if both strings are equal using == operator

```
1 #Compare two strings using == operator and function
2 def compareStrings(x, y):
3 # Use if else statement to compare x and y
4     if x==y:
5         return 1
6
7 # Declare two different variables as string1 and string2 and pass string in it
8 string1 = "Michael Jackson is the best"
9 string2 = "Michael Jackson is the best"
10
11 # Declare a variable to store result after comparing both the strings
12 check = compareStrings(string1, string2)
13
14 #Use if else statement to compare the string
15 if check==1:
16     print("\nString Matched")
17 else:
18     print("\nString not Matched")
```

Count the Frequency of Words Appearing in a String Using a Dictionary.

Find the count of occurrence of any word in our string using python. This is what we are going to do in this section, count the number of word in a given string and print it.

Lets suppose we have a 'string' and the 'word' and we need to find the count of occurrence of this word in our string using python. This is what we are going to do in this section, count the number of word in a given string and print it.

The first thing, we will do is define a function and then create a list that will be empty initially.

Next, we will add a code to convert the string to a list. Python string has a split() method. It takes a string and some separator to return a list.

Now we will declare an empty dictionary.

Next we will add code using for loop to iterate the words and value will count the frequency of each words in the string and store them to the dictionary.

Finally we will print the dictionary.

```
1 # Python Program to Count words in a String using Dictionary
2 def freq(string):
3
4     #step1: A list variable is declared and initialized to an empty list.
5     words = []
6
7     #step2: Break the string into list of words
8     words = string.split() # or string.lower().split()
9
10    #step3: Declare a dictionary
11    Dict = {}
12
13    #step4: Use for loop to iterate words and values to the dictionary
14    for key in words:
15        Dict[key] = words.count(key)
16
17    #step5: Print the dictionary
18    print("The Frequency of words is:",Dict)
19
20 #step6: Call function and pass string in it
21 freq("Mary had a little lamb Little lamb, little lamb Mary had a little lamb.Its fleece was white as s
22 Everywhere that Mary went The lamb was sure to go")
```

Setting default argument values in your custom functions

You can set a default value for arguments in your function. For example, in the `isGoodRating()` function, what if we wanted to create a threshold for what we consider to be a good rating? Perhaps by default, we should have a default rating of 4:

```
1 # Example for setting param with default value
2
3 def isGoodRating(rating=4):
4     if(rating < 7):
5         print("this album sucks it's rating is",rating)
6
7     else:
8         print("this album is good its rating is",rating)
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

Global variables

So far, we've been creating variables within functions, but we have not discussed variables outside the function. These are called global variables.

Let's try to see what `printer1` returns:

```
1 # Example of global variable
2
3 artist = "Michael Jackson"
4 def printer1(artist):
5     internal_var1 = artist
6     print(artist, "is an artist")
7
8 printer1(artist)
9 # try running the following code
10 #printer1(internal_var1)
```

We got a Name Error: name 'internal var' is not defined. Why?

It's because all the variables we create in the function is a **local variable**, meaning that the variable assignment does not persist outside the function.

But there is a way to create **global variables** from within a function as follows:

```
1 artist = "Michael Jackson"
2
3 def printer(artist):
4     global internal_var
5     internal_var= "Whitney Houston"
6     print(artist,"is an artist")
7
8 printer(artist)
9 printer(internal_var)
```

Scope of a Variable

The scope of a variable is the part of that program where that variable is accessible. Variables that are declared outside of all function definitions, such as the `myFavouriteBand` variable in the code shown here, are accessible from anywhere within the program. As a result, such variables are said to have global scope, and are known as global variables. `myFavouriteBand` is a global variable, so it is accessible from within the `getBandRating` function, and we can use it to determine a band's rating. We can also use it outside of the function, such as when we pass it to the `print` function to display it:

```

1 # Example of global variable
2
3 myFavouriteBand = "AC/DC"
4
5 def getBandRating(bandname):
6     if bandname == myFavouriteBand:
7         return 10.0
8     else:
9         return 0.0
10
11 print("AC/DC's rating is:", getBandRating("AC/DC"))
12 print("Deep Purple's rating is:",getBandRating("Deep Purple"))
13 print("My favourite band is:", myFavouriteBand)

```

Take a look at this modified version of our code. Now the `myFavouriteBand` variable is defined within the `getBandRating` function. A variable that is defined within a function is said to be a local variable of that function. That means that it is only accessible from within the function in which it is defined. Our `getBandRating` function will still work, because `myFavouriteBand` is still defined within the function. However, we can no longer print `myFavouriteBand` outside our function, because it is a local variable of our `getBandRating` function; it is only defined within the `getBandRating` function:

```

1 # Deleting the variable "myFavouriteBand" from the previous example to demonstrate an example of a local variable
2
3 del myFavouriteBand
4
5 # Example of local variable
6
7 def getBandRating(bandname):
8     myFavouriteBand = "AC/DC"
9     if bandname == myFavouriteBand:
10         return 10.0
11     else:
12         return 0.0
13
14 print("AC/DC's rating is: ", getBandRating("AC/DC"))
15 print("Deep Purple's rating is: ", getBandRating("Deep Purple"))
16 print("My favourite band is", myFavouriteBand)

```

Finally, take a look at this example. We now have two `myFavouriteBand` variable definitions. The first one of these has a global scope, and the second of them is a local variable within the `getBandRating` function. Within the `getBandRating` function, the local variable takes precedence. **Deep Purple** will receive a rating of 10.0 when passed to the `getBandRating` function. However, outside of the `getBandRating` function, the `getBandRating` s local variable is not defined, so the `myFavouriteBand` variable we print is the global variable, which has a value of **AC/DC**:

```

1 # Example of global variable and local variable with the same name
2
3 myFavouriteBand = "AC/DC"
4
5 def getBandRating(bandname):
6     myFavouriteBand = "Deep Purple"
7     if bandname == myFavouriteBand:
8         return 10.0
9     else:
10         return 0.0
11
12 print("AC/DC's rating is:",getBandRating("AC/DC"))
13 print("Deep Purple's rating is: ",getBandRating("Deep Purple"))
14 print("My favourite band is:",myFavouriteBand)

```

Collections and Functions

When the number of arguments are unknown for a function, They can all be packed into a tuple as shown:

```

1 def printAll(*args): # All the arguments are 'packed' into args which can be treated like a tuple
2     print("No of arguments:", len(args))
3     for argument in args:
4         print(argument)
5 #printAll with 3 arguments
6 printAll('Horsefeather', 'Adonis', 'Bone')
7 #printAll with 4 arguments
8 printAll('Sidecar', 'Long Island', 'Mudslide', 'Carriage')

```

Similarly, The arguments can also be packed into a dictionary as shown:

```

1 def printDictionary(**args):
2     for key in args:
3         print(key + " : " + args[key])
4
5 printDictionary(Country='Canada', Province='Ontario', City='Toronto')
6

```

Functions can be incredibly powerful and versatile. They can accept (and return) data types, objects and even other functions as arguments. Consider the example below:

```

1 def addItem(list):
2     list.append("Three")
3     list.append("Four")
4
5 myList = ["One", "Two"]
6
7 addItem(myList)
8
9 myList
10

```

Note how the changes made to the list are not limited to the functions scope. This occurs as it is the lists **reference** that is passed to the function - Any changes made are on the original instance of the list. Therefore, one should be cautious when passing mutable objects into functions.

Quiz on Functions Lab-3

Come up with a function that divides the first input by the second input:

```

1 # Write your code below and press Shift+Enter to execute
2 def div(a, b):
3     return(a/b)
4 div(100, 5)

```

 20.0

► [Click here for the solution](#)

Use the function `con` for the following question.

```

1 # Use the con function for the following question
2 def con(a, b):
3     return(a + b)

```

Can the `con` function we defined before be used to add two integers or strings?

```

1 # Write your code below and press Shift+Enter to execute
2 # Yes, for example:
3 con(2, 2)

```

```
3 con(2, 2)
```

 4

► [Click here for the solution](#)

Can the `con` function we defined before be used to concatenate lists or tuples?

```
1 # Write your code below and press Shift+Enter to execute
2 # Yes, for example:
3 con(['a', 1], ['b', 1])
```


 ['a', 1, 'b', 1]

► [Click here for the solution](#)

Write a function code to find total count of word `little` in the given string: "Mary had a little lamb Little lamb, little lamb Mary had a little lamb.Its fleece was white as snow And everywhere that Mary went Mary went, Mary went Everywhere that Mary went The lamb was sure to go"

```
1 # Write your code below and press Shift+Enter to execute
2 # Python Program to Count words in a String using Dictionary
3 def freq(string,passedkey):
4     words = []
5     words = string.split() # or string.lower().split()
6     Dict = {}
7     for key in words:
8         if(key == passedkey):
9             Dict[key] = words.count(key)
10    print("Total Count:",Dict)
11 freq("Mary had a little lamb Little lamb, little lamb Mary had a little lamb.\
12 Its fleece was white as snow And everywhere that Mary went Mary went, Mary went \
13 Everywhere that Mary went The lamb was sure to go","little")
```

--INSERT--

 Total Count: {'little': 3}

► [Click here for the solution](#)

The last exercise!

Congratulations, you have completed your first lesson and hands-on lab in Python.

Author

[Joseph Santarcangelo](#)

Other contributors

[Mavis Zhou](#)



Skills Network

