

Phiphat Pinyosophon

August 05 , 2024

Foundation of Programming: Python

Assignment 06

<https://github.com/pinyosophon/python110-Summer2024>

FUNCTION AND CLASSES

Introduction

Often in programming, we'll have to repeat certain process over and over again as part of a code we're writing, whether it be printing text, arithmetic, or asking for input data from user, we call these reusable block of code used to perform their related action "function". In this week assignment, we're to simplify our code from previous week by creating custom functions then replace the code we wrote with them, and organize them in classes.

FUNCTION

As mentioned before, functions are reusable block of codes we create to perform certain actions we need. While we're writing function, there are some good practices we should follow:

1. The code defining function must exist before it can be called.
2. It should be defined after imports, constants, variables.
3. function is followed by parentheses.

```
def output_message(message: str):  
    """  
    ~~~~~  
    print message  
    :param message: string message to print  
    :return: None  
    """  
    ~~~~~  
    print(message)
```

```
output_message("Program Ended")
```

Figure01: Example of function and its usage.

In Figure01, we can see an example of a function written to replace simple print() function. One of the benefits to do it this way instead of use print() function is that if there's anything we need to add onto this print function, we can do so and it will affect everything that's using this output_message function.

CLASS

In Python, "class" is used to organize functions we created. It provides a mean to bundling data and similar type of functions together. You can look at it as a template for creating objects. In Figure02, is an example of how a class was used. We created many different type of function dealing with inputting and outputting data

```
81 class IO:
82     10 usages
83     @staticmethod
84     def output_message(message: str):
85         """
86         print message
87         :param message: string message to print
88         :return: None
89         """
90         print(message)
91
92     6 usages
93     @staticmethod
94     def output_error_message(message: str, error: Exception = None):
95         """
96         print error message when there's exception
97         :param message: string message to print
98         :param error: Exception
99         :return: None
100         """
101         if Exception is not None:
102             print("--technical information--")
103             print(error, error.__doc__, type(error), sep='\n')
104
105     1 usage
106     @staticmethod
107     def input_menu_choice(menu:str)->str:
108         """
109         get user input when display menu
110         :param menu: MENU variable to display
111         :return: string 1,2,3,4 as choices user can make
112         """
113         return input(menu)
```

Figure02: example of class with functions in it.

To use function in a class, we can call on it like how you can see in Figure03 below:

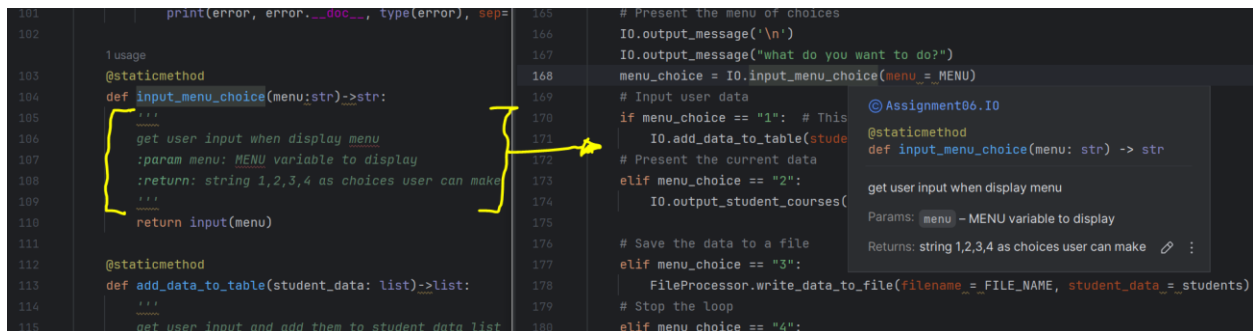
```

164
165     # Present the menu of choices
166     IO.output_message('\n')
167     IO.output_message("what do you want to do?")
168     menu_choice = IO.input_menu_choice(menu = MENU)

```

Figure03: using function organized in class.

As you can see, you put a name of a class first, followed by name of the function and then put related string/variables in the parentheses. For `IO.output_message`, you just need to type in what you want it to print, and for `IO.input_menu_choice()` you have to put in menu variable for it to work correctly. And how do you know what to put in between the parentheses? A good practice for when writing function and class would be to create comment and description for it while you create one. So when you hover over a function, it will show what the function does, what kind of data it's expecting, and what it is returning (if any).



The screenshot shows a code editor with two panels. The left panel displays the definition of the `input_menu_choice` function, which is a static method of the `IO` class. It takes a `menu` string as input and returns a string representing the user's choice. The right panel shows the same function being called in the `main` function. A yellow arrow points from the `input_menu_choice` function call in the right panel to the function definition in the left panel. A tooltip is visible over the function definition, showing the function's signature, parameters, and return value.

```

101         print(error, error.__doc__, type(error), sep='\n')
102
103     1 usage
104     @staticmethod
105     def input_menu_choice(menu:str)->str:
106         """
107         get user input when display menu
108         :param menu: MENU variable to display
109         :return: string 1,2,3,4 as choices user can make
110         """
111         return input(menu)
112
113     @staticmethod
114     def add_data_to_table(student_data: list)->list:
115         """
116         get user input and add them to student data list
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

```

Figure04: Dont forget to write comment/description for your function.

In Figure04, it shows that this `input_menu_choice` expect to have a variable for menu, and it will return string. Since our menu consist of 4 choices, only those 4 choices are expected to work, and anything other than that will give error or the program will prompt user to type in new input.

Global Variables vs. Local Variables

Now that we started writing our own function, we also started to deal with variables in a way we previously did not in weeks before. We now are dealing with global variables and local variables. Local variables, as the name suggests, are variables we created to use locally within a function. You will not be able to access these variables from outside their own function. For example, please take a look at example below in Figure05:

```

113     def add_data_to_table(student_data: list)->list:
114         """
115         get user input and add them to student data list
116         :param student_data: a list created to store and adding data from user input
117         :return: return a list of student data
118         """
119         student_first_name: str = ''
120         student_last_name: str = ''
121         course_name: str = ''
122         student_row: dict = {}
123         try:
124             student_first_name = input("Enter the student's first name: ")
125             if not student_first_name.isalpha():
126                 raise ValueError("The First name should not contain numbers.")
127             student_last_name = input("Enter the student's last name: ")
128             if not student_last_name.isalpha():
129                 raise ValueError("The last name should not contain numbers.")
130             course_name = input("Please enter the name of the course: ")
131             student_row = {"FirstName": student_first_name,
132                           "LastName": student_last_name,
133                           "CourseName": course_name}
134             student_data.append(student_row)
135             IO.output_message(f"You have registered {student_first_name} {student_last_name} for {course_name}.")

```

Figure05: local variables.

As you can see that this function has 4 local variables, `student_first_name`, `student_last_name`, `course_name`, and `student_row`. These functions will not be available to use anywhere else except from within this `add_data_to_table()` function. So if these variables only exist locally, we have to use other ways to access data created from within this function. We can use “return” to pass on, or return data from within a function outside of it. For example:

```

27     class FileProcessor:
28         @staticmethod
29         def read_data_from_file(filename: str, student_data: list) -> list:
30             """
31             get data from json file and store it in a list called student data
32             :param filename: refer to json file
33             :param student_data: a list created to store data from json file
34             :return: will return a list
35             """
36             try:
37                 file = open(filename, "r")
38                 student_data = json.load(file)
39                 file.close()
40             except FileNotFoundError as e:
41                 IO.output_error_message( message: "Text file must exist before running this script!\n", e)
42                 file = open(filename, "w")
43                 json.dump(student_data, file)
44             except JSONDecodeError as e:
45                 IO.output_error_message( message: "--Technical Information--", e)
46                 file = open(filename, "w")
47                 json.dump(student_data, file)
48             except Exception as e:
49                 IO.output_error_message( message: "Unhandle Exception", e)
50             finally:
51                 if file.closed == False:
52                     file.close()
53             return student_data
54

```

Figure06: “Returning” data created from within a function.

In Figure06, we created `read_data_from_file()` function to read and extract list from a json file. The data we get from json file is stored in a list called `student_data`. Then we have to call it to access it from outside the function, this way, we can access this data and manipulate it to use in other functions in this script, as you can see in Figure07:

```
158  > #get data returned from read_data_from_file so it can be used in other function, as Python is executed from top-down
159  #this has to be done here so other function can have access to it
160  students = FileProcessor.read_data_from_file(filename = FILE_NAME, student_data = students)
161
```

Figure07: Accessing data returned from a function.

In Figure07, `students` is a global variable we created so that we can use to transfer list data we got from executing functions. Also note that this line of code must exist before other lines that need data from `students` list, this is because Python is executing from top to bottom, as you can see in Figure08:

```
157
158  #get data returned from read_data_from_file so it can be used in other function, as Python is executed from top-down
159  #this has to be done here so other function can have access to it
160  students = FileProcessor.read_data_from_file(filename = FILE_NAME, student_data = students)
161
162  # Present and Process the data
163  while (True):
164
165      # Present the menu of choices
166      IO.output_message('\n')
167      IO.output_message("what do you want to do?")
168      menu_choice = IO.input_menu_choice(menu=__MENU)
169      # Input user data
170      if menu_choice == "1": # This will not work if it is an integer!
171          IO.add_data_to_table(student_data=__students)
172      # Present the current data
173      elif menu_choice == "2":
174          IO.output_student_courses(student_data=__students)
175
176      # Save the data to a file
177      elif menu_choice == "3":
178          FileProcessor.write_data_to_file(filename=__FILE_NAME, student_data=__students)
179      # Stop the loop
180      elif menu_choice == "4":
181          break # out of the loop
182      else:
183          IO.output_message("Please only choose option 1, 2, 3, or 4")
184
185  IO.output_message("Program Ended")
186
```

Figure08: accessing needed data BEFORE executing everything else that need it.

Summary

Function is a convenient way to simplify our coding, but it should be noted that, while writing function, it should exist before it can be called, and it should be defined after all the imports, constants, and variable. Also take note to be careful where you define your

variables. Variables which are defined in a global level can be used anywhere in your code, while the ones you define locally within a function can only be accessed in that function, and nowhere else. Always be extra careful as local and global can have the same name, so it's best to avoid it. You can group any related functions together in class and use it later. Take extra caution to create a line of code to access data in a function to be accessed globally before it's needed as Python executes from top to bottom.