

# **ALL IN ONE** **CONVERTOR**

**A Project Work Report**

*Submitted in the partial fulfilment for the award of the degree of*

**BACHELOR OF ENGINEERING**

**IN**

**COMPUTER SCIENCE  
BIG DATA ANALYTICS**

**Submitted by:**

**DIGVIJAY KUMAR**

**AVINASH RAJ**

**19BCS3878**

**19BCS3899**

**Under the Supervision of:**

**JYOTI MEHRA**



**CHANDIGARH UNIVERSITY, GHARUAN, MOHALI - 140413, PUNJAB**

**December, 2020**

Name and signature of student(s)

Name and signature of Supervisor

---

# PROJECT COMPLETION CERTIFICATE

## Project Title

This is to certify that the DIGVIJAY KUMAR AND AVINASH RAJ have successfully completed the project work titled “ ALL IN ONE CONVERTOR ”  
*Submitted in the partial fulfilment for the award of the degree of* **BACHELOR OF ENGINEERING IN COMPUTER SCIENCE BIG DATA ANALYTICS**

This project is the record of authentic work carried out during the academic year.

JYOTI MEHRA

Project Guide

**Date: 5 DEC 2020**

## **DECLARATION**

I the undersigned solemnly declare that the project report is based on my own work carried out during the course of our study under the supervision of jyoti mehra. I assert the statements made and conclusions drawn are an outcome of my work. I further certify that the work contained in the report is original and has been done by me under the general supervision of my supervisor.

II. The work has not been submitted to any other Institution for any other degree/ diploma/certificate in this university or any other University of India or abroad.

III. We have followed the guidelines provided by the university in writing the report.

IV. Whenever we have used materials (data, theoretical analysis, and text) from other sources, we have given due credit to them in the text of the report and giving their details in the references.

NAME - DIGVIJAY KUMAR  
AVINASH RAJ

UID - 19BCS3878  
19BCS3899

## **ACKNOWLEDGEMENT**

I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

I am highly indebted to (Name of your Organization Guide) for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

I would like to express my gratitude towards my parents and my department for their kind co-operation and encouragement which help me in completion of this project.

**THANKS AGAIN TO ALL WHO HELPED**

Chapter 1: Introduction to project

Chapter 2: Project Requirements (Software/Hardware requirements)

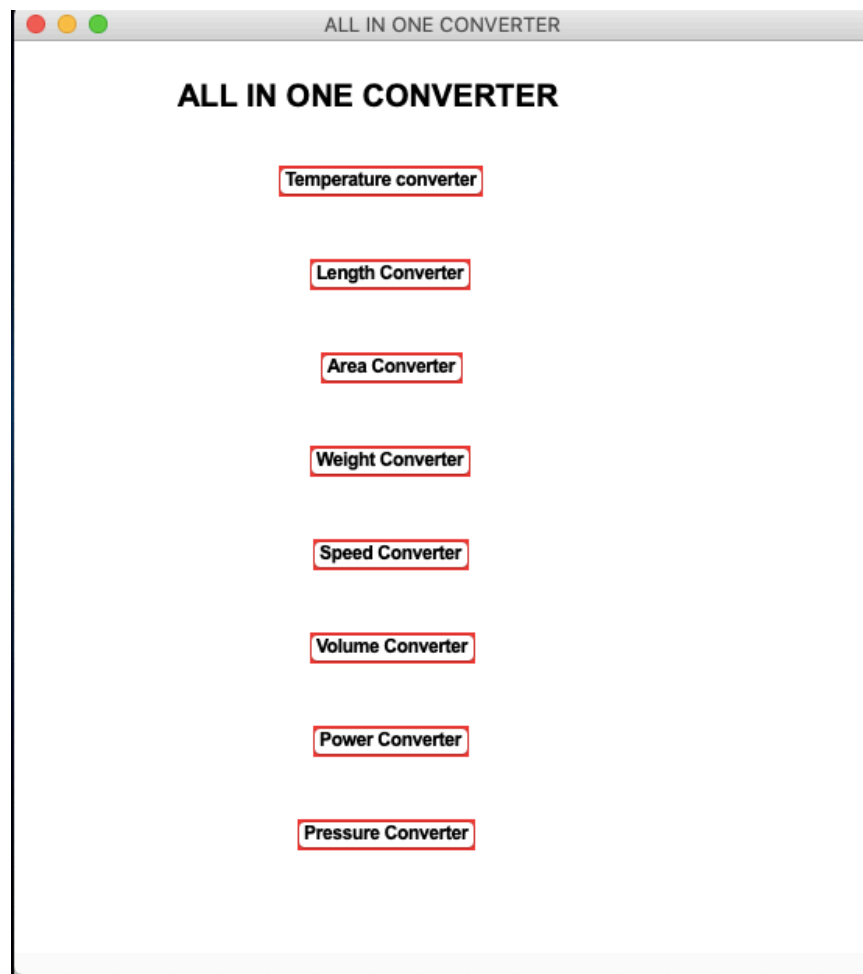
Chapter 3: Implementation Details (Algorithm, code )

Chapter 4: Output Analysis (screenshots)

# Introduction to project

This is a all in one convertor in which you can convert different different parameters like speed, power, pressure, volume, weight, area, length, temperature.

We developed this project using python language, however the project is a GUI application that convert different quantities.



# Project Requirements (Software/Hardware requirements)

## Software

1. Operating system - window 7 or above  
Mac OS X or above  
Linux
2. IDE ( working station) - VS code  
Sublime  
Pycharm  
Spyder

## Hardware

For Mac - work in all MacBook

For window - Intel pentium 4 or above

700 MB of harddisk drive

256 MB of RAM



# Implementation Details (Algorithm, code )

## ALGORITHM

**Step0 : start a function named speed convertor.**

**Step1 : set IDS for each measurement types.**

**Step2 : write a function named “CONVERT” with parameters “amount”, “from”, “to”.**

**Step3 : inside function if frm! “meps”**

**SET amt = amt\*factors[frm]**

**Return amt/factors[to]**

**[END of the function]**

**Step4 : [END of the main function]**

**Step5 : start a function named power convertor.**

**Step6 : set IDS for each measurement types.**

**Step7 : write a function named “CONVERT” with parameters “amount”, “from”, “to”.**

**Step8 : inside function if frm! = “W”**

**SET amt = amt\*factor[frm]**

**Return amt/factors[to]**

**Else return amt/factors[to]**

**[END of the function].**

**Step9 : [END of the main function].**

**Step10 : start a function named pressure convertor.**

**Step11 : set IDS for each measurement types.**

**Step12 : write a function named “CONVERT” with parameters “amount”, “from”, “to”.**

**Step13 : inside function if frm! = “W”**

**SET amt = amt\*factor[frm]**

**Return amt/factors[to]**

**Else return amt/factors[to]**

**[END of the function].**

**Step14 : [END of the main function].**

**Step15 : start a function named pressure convertor.**

**Step16 : set IDS for each measurement types.**

**Step17 : write a function named “CONVERT” with parameters “amount”, “from”, “to”.**

**Step18 : inside function if frm! = “1”**

**SET amt = amt\*factor[frm]**

**Return amt/factor[to]**

**Else return amt/factors[to]**

**[END of the function].**

**Step19 : [END of the main function].**

**STEP20 : start a function named weight convertor.**

**Step21 : set IDS for each measurement types.**

**Step22 : write a function named “CONVERT” with parameters “amount”, “from”, “to”.**

**Step23 : inside function if frm! = ‘g’**

**SET amt = amt\*factors[frm]**

**Return amt/factors[to]**

**Else return amt/factors[to]**

**[END of the function].**

**Step24 : [END of the main function].**

**STEP25 : start a function named area convertor.**

**Step26 : set IDS for each measurement types.**

**Step27 : write a function named “CONVERT” with parameters “amount”, “from”, “to”.**

**Step28 : inside function if fromvar.get() present in meter factor.keys() and to var.get() present in meter factor.keys().**

**Step29 : result = (float(str(x))\*meter factor [from unit] / (meter function [to unit])**

**resulttxt.insert (0, str(result))**

**[END of the function].**

**Step30 : [END of the main function].**

**STEP31 : start a function named length convertor.**

**Step32 : set IDS for each measurement types.**

**Step33 : write a function named “CONVERT” with parameters “amount”, “from”, “to”.**

**Step34 : inside function if frm! = ‘m’**

**SET amt = amt\*factor[frm]**

**Return amt/factors[to]**

**Else return amt/factors[to]**

**[END of the function].**

**Step35 : [END of the main function].**

**STEP36 : start a function named temperature convertor.**

**Step37 : set IDS for each measurement types.**

**Step38 : write a function named “CONVERT” with parameters “amount”, “from”, “to”.**

**Step39 : inside function if celTempvar.get()! = 0.0**

**celToFah = (celTemp\*9/5 + 32)**

**fahTempvar.set(celToFah)**

**Elif fahTempvar.get()! = 0.0**

**fahToCel = ((fahTemp - 32)\*(5/9))**

**celTempvar.set(fahToCel)**

**[END of the function].**

**Step40 : [END of the main function].**

## CODE

```
import sys
import tkinter as tk
from tkinter import *
import urllib.request
import webbrowser
from functools import partial
from tkinter import Tk, StringVar, ttk

##### MAIN
#####
root = Tk()
root.title('ALL IN ONE CONVERTER')
root.geometry("550x600+200+300")
labelfont = ('ariel', 56, 'bold')
l=Label(root,text='ALL IN ONE CONVERTER',font = ("Arial", 20,
"bold"), justify = CENTER)
l.place(x=100,y=20)
#####
#####

##### SPEED CONVERTER
START
#####
#####
def SpeedConverter():
    factors = {'kmph' : 0.2777777778, 'mph' : 0.44704, 'meph' :
0.0002777778, 'mps' : 1609.344 , 'kmps' : 1000, 'meps' : 1}
    ids = {"km/hour" : 'kmph', "mile/hour" : 'mph', "meter/
hour" : 'meph', "mile/second" : 'mps', "km/second" : 'kmps'}

    # function to convert from a given unit to another
    def convert(amt, frm, to):
```

```

    if frm != 'meps':
        amt = amt * factors[frm]
        return amt / factors[to]
    else:
        return amt / factors[to]

def callback():
    try:
        amt = float(in_field.get())
    except ValueError:
        out_amt.set('Invalid input')
        return None
    if in_unit.get() == 'Select Unit' or out_unit.get() == 'Select Unit':
        out_amt.set('Input or output unit not chosen')
        return None
    else:
        frm = ids[in_unit.get()]
        to = ids[out_unit.get()]
        out_amt.set(convert(amt, frm, to))

# initiate window
root = Toplevel()
root.title("Speed Converter")

# initiate frame
mainframe = ttk.Frame(root, padding="3 3 12 12")
mainframe.pack(fill=BOTH, expand=1)
titleLabel = Label (mainframe, text = "Speed Converter", font
= ("Arial", 12, "bold"), justify = CENTER).grid(column=1,row=1)
in_amt = StringVar()
in_amt.set('0')
out_amt = StringVar()
in_unit = StringVar()
out_unit = StringVar()
in_unit.set('Select Unit')
out_unit.set('Select Unit')

# Add input field
in_field = ttk.Entry(mainframe, width=20,
textvariable=in_amt)
in_field.grid(row=1, column=2, sticky=(W, E))

# Add drop-down for input unit
in_select = OptionMenu(mainframe, in_unit, "km/hour", "mile/
hour", "meter/hour", "mile/second", "km/second") .grid(column=3,
row=1, sticky=W)

```

```

        # Add output field and drop-down
        ttk.Entry(mainframe, textvariable=out_amt,
state="readonly").grid(column=2, row=3, sticky=(W, E))
        in_select = OptionMenu(mainframe, out_unit, "km/hour", "mile/
hour", "meter/hour", "mile/second", "km/second").grid(column=3,
row=3, sticky=W)
        calc_button = ttk.Button(mainframe, text="Calculate",
command=callback).grid(column=2, row=2, sticky=E)
        for child in mainframe.winfo_children():
            child.grid_configure(padx=5, pady=5)
        in_field.focus()

##### SPEED CONVERTER
END#####
#####

```

```

23 ##### SPEED CONVERTER START #####
24 def SpeedConverter():
25     factors = {'kmph' : 0.277777778, 'mph' : 0.44704, 'meph' : 0.000277778, 'mps' : 1609.344, 'kmps' : 1000, 'meps' : 1}
26     ids = {'km/hour' : 'kmph', 'mile/hour' : 'mph', 'meter/hour' : 'meph', 'mile/second' : 'mps', 'km/second' : 'kmps'}
27
28     # function to convert from a given unit to another
29     def convert(amt, frm, to):
30         if frm != 'meps':
31             amt = amt * factors[frm]
32             return amt / factors[to]
33         else:
34             return amt / factors[to]
35
36     def callback():
37         try:
38             amt = float(in_field.get())
39         except ValueError:
40             out_amt.set('Invalid input')
41             return None
42         if in_unit.get() == 'Select Unit' or out_unit.get() == 'Select Unit':
43             out_amt.set('Input or output unit not chosen')
44             return None
45         else:
46             frm = ids[in_unit.get()]
47             to = ids[out_unit.get()]
48             out_amt.set(convert(amt, frm, to))
49
50     # initiate window
51     root = Toplevel()
52     root.title("Speed Converter")
53

```

PROBLEMS 100 OUTPUT DEBUG CONSOLE TERMINAL

1: bash

DIGVIJAYS-MacBook-Air:all in one convertor digvijaykumar\$

Python 3.9.0 64-bit 29 71 Ln 503, Col 1 Spaces: 4 UTF-8 LF Python

```

##### POWWR CONVERTER
START
#####
#####
def PowerConverter():
    factors = {'EW' : 1000000000000000000, 'PW' :
1000000000000000000, 'TW' : 10000000000000, 'GW' : 1000000000, 'MW' :
1000000, 'KW' : 1000, 'HP' : 746, 'W' : 1}
    ids = {"exawatt" : 'EW', "petawatt" : 'PW', "terawatt" :
'TW', "gigawatt" : 'GW', "megawatt" : 'MW', "kilowatt" : 'KW',
"horsepower" : 'HP', "watt" : 'W'}

    # function to convert from a given unit to another
    def convert(amt, frm, to):
        if frm != 'W':
            amt = amt * factors[frm]
            return amt / factors[to]
        else:
            return amt / factors[to]

    def callback():
        try:
            amt = float(in_field.get())
        except ValueError:
            out_amt.set('Invalid input')
            return None
        if in_unit.get() == 'Select Unit' or out_unit.get() ==
'Select Unit':
            out_amt.set('Input or output unit not chosen')
            return None
        else:
            frm = ids[in_unit.get()]
            to = ids[out_unit.get()]
            out_amt.set(convert(amt, frm, to))

    # initiate window
    root = Toplevel()
    root.title("Power Converter")

    # initiate frame
    mainframe = ttk.Frame(root, padding="3 3 12 12")
    mainframe.pack(fill=BOTH, expand=1)
    titleLabel = Label (mainframe, text = "Power Converter", font
= ("Arial", 12, "bold"), justify = CENTER).grid(column=1,row=1)
    in_amt = StringVar()
    in_amt.set('0')
    out_amt = StringVar()

```



```

in_unit = StringVar()
out_unit = StringVar()
in_unit.set('Select Unit')
out_unit.set('Select Unit')

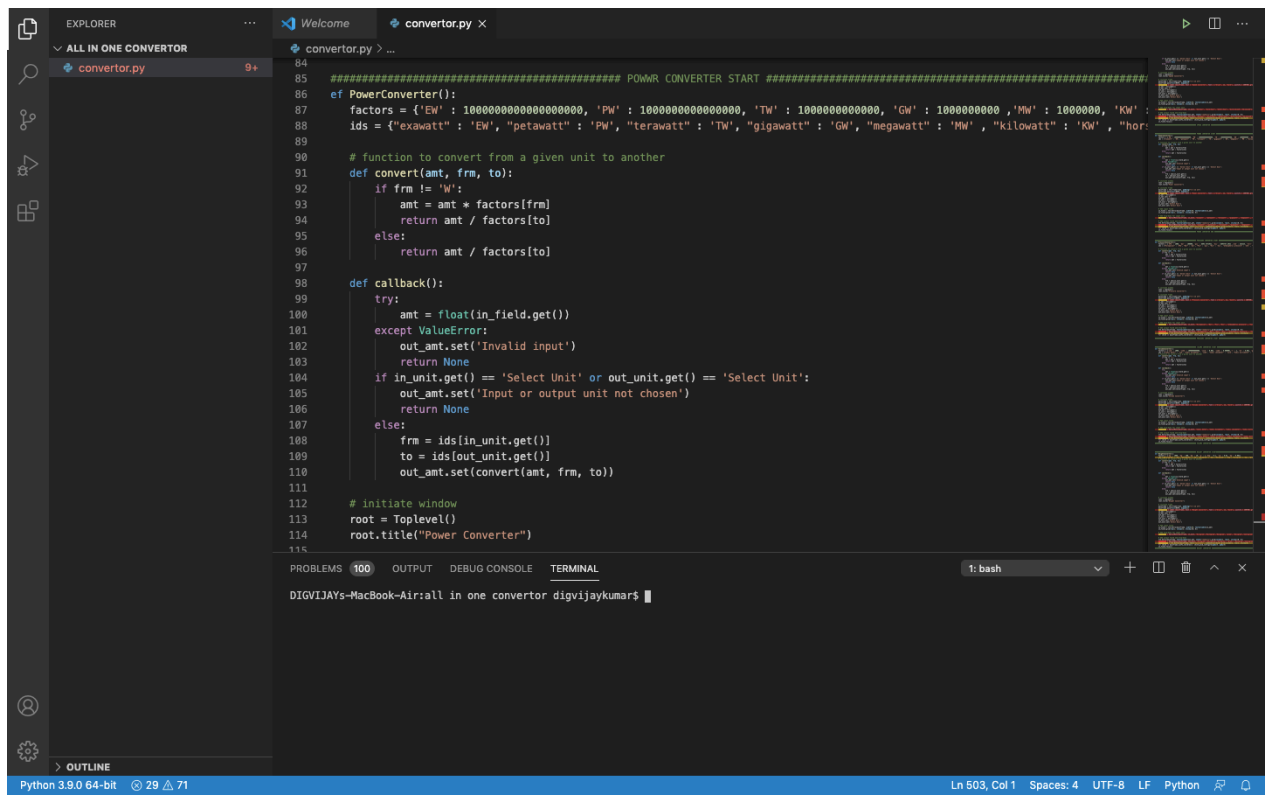
# Add input field
in_field = ttk.Entry(mainframe, width=20,
textvariable=in_amt)
in_field.grid(row=1, column=2, sticky=(W, E))

# Add drop-down for input unit
in_select = OptionMenu(mainframe, in_unit, "exawatt" ,
"petawatt" , "terawatt" , "gigawatt" , "megawatt" , "kilowatt",
"horsepower" , "watt") .grid(column=3, row=1, sticky=W)

# Add output field and drop-down
ttk.Entry(mainframe, textvariable=out_amt,
state="readonly").grid(column=2, row=3, sticky=(W, E))
in_select = OptionMenu(mainframe, out_unit, "exawatt" ,
"petawatt" , "terawatt" , "gigawatt" , "megawatt" , "kilowatt",
"horsepower" , "watt").grid(column=3, row=3, sticky=W)
calc_button = ttk.Button(mainframe, text="Calculate",
command=callback).grid(column=2, row=2, sticky=E)
for child in mainframe.winfo_children():
child.grid_configure(padx=5, pady=5)
in_field.focus()

##### POWER CONVERTER
END
#####
#####

```



```

##### PRESSURE
CONVERTER START
#####
def PressureConverter():
    factors = {'kPa' : 1000, 'bar' : 100000, 'psi' :
6894.7572932, 'ksi' : 6894757.2932, 'atm' : 101325, 'torr' :
133.32236842, 'Pa' : 1}
    ids = {"Kilopascal" : 'kPa', "Bar" : 'bar', "Psi" : 'psi',
"Ksi" : 'ksi', "atmospheric pressure" : 'atm', "Torr" :
'torr', "Pascal" : 'Pa'}

    # function to convert from a given unit to another
    def convert(amt, frm, to):
        if frm != 'W':
            amt = amt * factors[frm]
            return amt / factors[to]
        else:
            return amt / factors[to]

    def callback():
        try:
            amt = float(in_field.get())
        except ValueError:
            out_amt.set('Invalid input')
            return None
        if in_unit.get() == 'Select Unit' or out_unit.get() ==
'Select Unit':
            out_amt.set('Input or output unit not chosen')
            return None
        else:
            frm = ids[in_unit.get()]
            to = ids[out_unit.get()]
            out_amt.set(convert(amt, frm, to))

    # initiate window
    root = Toplevel()
    root.title("Pressure Converter")

    # initiate frame
    mainframe = ttk.Frame(root, padding="3 3 12 12")
    mainframe.pack(fill=BOTH, expand=1)

```

```

        titleLabel = Label (mainframe, text = "Pressure Converter",
font = ("Arial", 12, "bold"), justify =
CENTER).grid(column=1,row=1)
        in_amt = StringVar()
        in_amt.set('0')
        out_amt = StringVar()
        in_unit = StringVar()
        out_unit = StringVar()
        in_unit.set('Select Unit')
        out_unit.set('Select Unit')

```

```

        # Add input field
        in_field = ttk.Entry(mainframe, width=20,
textvariable=in_amt)
        in_field.grid(row=1, column=2, sticky=(W, E))

```

```

        # Add drop-down for input unit
        in_select = OptionMenu(mainframe, in_unit, "Kilopascal",
"Bar", "Psi", "Ksi", "atmospheric pressure",
"Torr", "Pascal").grid(column=3, row=1, sticky=W)

```

```

        # Add output field and drop-down
        ttk.Entry(mainframe, textvariable=out_amt,
state="readonly").grid(column=2, row=3, sticky=(W, E))
        in_select = OptionMenu(mainframe, out_unit, "Kilopascal",
"Bar", "Psi", "Ksi", "atmospheric pressure",
"Torr", "Pascal").grid(column=3, row=3, sticky=W)
        calc_button = ttk.Button(mainframe, text="Calculate",
command=callback).grid(column=2, row=2, sticky=E)
        for child in mainframe.winfo_children():
child.grid_configure(padx=5, pady=5)

```

```

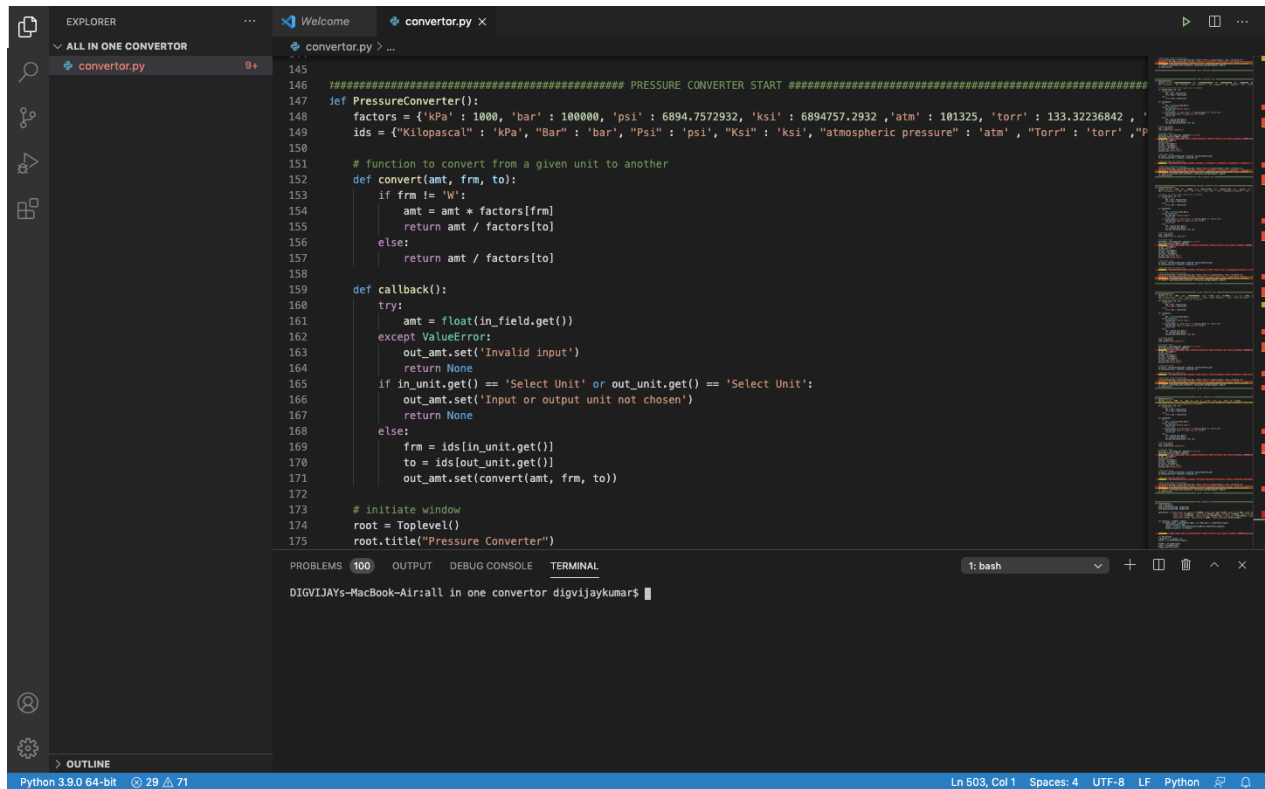
##### PRESSURE
CONVERTER START

```

```

#####
#####

```



```

##### VOLUME CONVERTER
START
#####
#####
def VolumeConverter():
    factors = {'cum' : 1000, 'cukm' : 1000000000000, 'cucm' :
0.001, 'cumm' : 0.000001, 'l' : 1, 'ml' : 0.001, 'gal' :
3.785411784}
    ids = {"Cubic meter" : 'cum', "Cubic kilometer" : 'cukm',
"Cubic cenimeter" : 'cucm', "Cubic millimeter" : 'cumm',
"Liter" : 'l', "Milliliter" : 'ml', "gallon" : 'gal'}
    # function to convert from a given unit to another
    def convert(amt, frm, to):
        if frm != 'l':
            amt = amt * factors[frm]
            return amt / factors[to]
        else:
            return amt / factors[to]

    def callback():
        try:
            amt = float(in_field.get())

```

```

        except ValueError:
            out_amt.set('Invalid input')
            return None
        if in_unit.get() == 'Select Unit' or out_unit.get() ==
'Select Unit':
            out_amt.set('Input or output unit not chosen')
            return None
        else:
            frm = ids[in_unit.get()]
            to = ids[out_unit.get()]
            out_amt.set(convert(amt, frm, to))

# initiate window
root = Toplevel()
root.title("Volume Converter")

# initiate frame
mainframe = ttk.Frame(root, padding="3 3 12 12")
mainframe.pack(fill=BOTH, expand=1)
titleLabel = Label (mainframe, text = "Volume Converter",
font = ("Arial", 12, "bold"), justify =
CENTER).grid(column=1,row=1)
in_amt = StringVar()
in_amt.set('0')
out_amt = StringVar()
in_unit = StringVar()
out_unit = StringVar()
in_unit.set('Select Unit')
out_unit.set('Select Unit')

# Add input field
in_field = ttk.Entry(mainframe, width=20,
textvariable=in_amt)
in_field.grid(row=1, column=2, sticky=(W, E))

# Add drop-down for input unit
in_select = OptionMenu(mainframe, in_unit, "Cubic meter",
"Cubic kilometer", "Cubic cenimeter", "Cubic millimeter",
"Liter", "Milliliter", "gallon") .grid(column=3, row=1, sticky=W)

# Add output field and drop-down
ttk.Entry(mainframe, textvariable=out_amt,
state="readonly").grid(column=2, row=3, sticky=(W, E))
in_select = OptionMenu(mainframe, out_unit, "Cubic meter",
"Cubic kilometer", "Cubic cenimeter", "Cubic millimeter",
"Liter", "Milliliter", "gallon").grid(column=3, row=3, sticky=W)

```

```

        calc_button = ttk.Button(mainframe, text="Calculate",
command=callback).grid(column=2, row=2, sticky=E)
        for child in mainframe.winfo_children():
            child.grid_configure(padx=5, pady=5)
        in_field.focus()
##### VOLUME CONVERTER
END#####
#####

```

```

206
207 ##### VOLUME CONVERTER START #####
208 def VolumeConverter():
209     factors = {'cum': 1000, 'cukm': 1000000000000, 'cucm': 0.001, 'cumm': 0.000001, 'l': 1, 'ml': 0.001, 'gal': 3.785411784}
210     ids = {"Cubic meter": 'cum', "Cubic kilometer": 'cukm', "Cubic centimeter": 'cucm', "Cubic millimeter": 'cumm', "Liter": 'l'}
211     # function to convert from a given unit to another
212     def convert(amt, frm, to):
213         if frm != 'l':
214             amt = amt * factors[frm]
215             return amt / factors[to]
216         else:
217             return amt / factors[to]
218
219     def callback():
220         try:
221             amt = float(in_field.get())
222         except ValueError:
223             out_amt.set('Invalid input')
224             return None
225         if in_unit.get() == 'Select Unit' or out_unit.get() == 'Select Unit':
226             out_amt.set('Input or output unit not chosen')
227             return None
228         else:
229             frm = ids[in_unit.get()]
230             to = ids[out_unit.get()]
231             out_amt.set(convert(amt, frm, to))
232
233     # initiate window
234     root = Toplevel()
235     root.title("Volume Converter")
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
1000

```

```
##### WEIGHT CONVERTER
START#####
#####
def WeightConverter():
    factors = {'kg' : 1000, 'hg' : 100, 'dg' : 10, 'g' :
1, 'deg' : 0.1, 'cg' : 0.01, 'mg' : 0.001}
    ids = {"Kilogram" : 'kg', "Hectagram" : 'hg', "Decagram" :
'dg', "Decigram" : 'deg', "Kilogram" : 'kg', "gram" : 'g',
"centigram" : 'cg', "milligram" : 'mg'}
    # function to convert from a given unit to another
    def convert(amt, frm, to):
        if frm != 'g':
            amt = amt * factors[frm]
            return amt / factors[to]
        else:
            return amt / factors[to]

    def callback():
        try:
            amt = float(in_field.get())
        except ValueError:
            out_amt.set('Invalid input')
            return None
        if in_unit.get() == 'Select Unit' or out_unit.get() ==
'Select Unit':
            out_amt.set('Input or output unit not chosen')
            return None
        else:
            frm = ids[in_unit.get()]
            to = ids[out_unit.get()]
            out_amt.set(convert(amt, frm, to))

    # initiate window
    root = Toplevel()
    root.title("Weight Converter")

    # initiate frame
    mainframe = ttk.Frame(root, padding="3 3 12 12")
    mainframe.pack(fill=BOTH, expand=1)
    titleLabel = Label (mainframe, text = "Weight Converter",
font = ("Arial", 12, "bold"), justify =
CENTER).grid(column=1,row=1)
    in_amt = StringVar()
    in_amt.set('0')
    out_amt = StringVar()
    in_unit = StringVar()
    out_unit = StringVar()
```

```

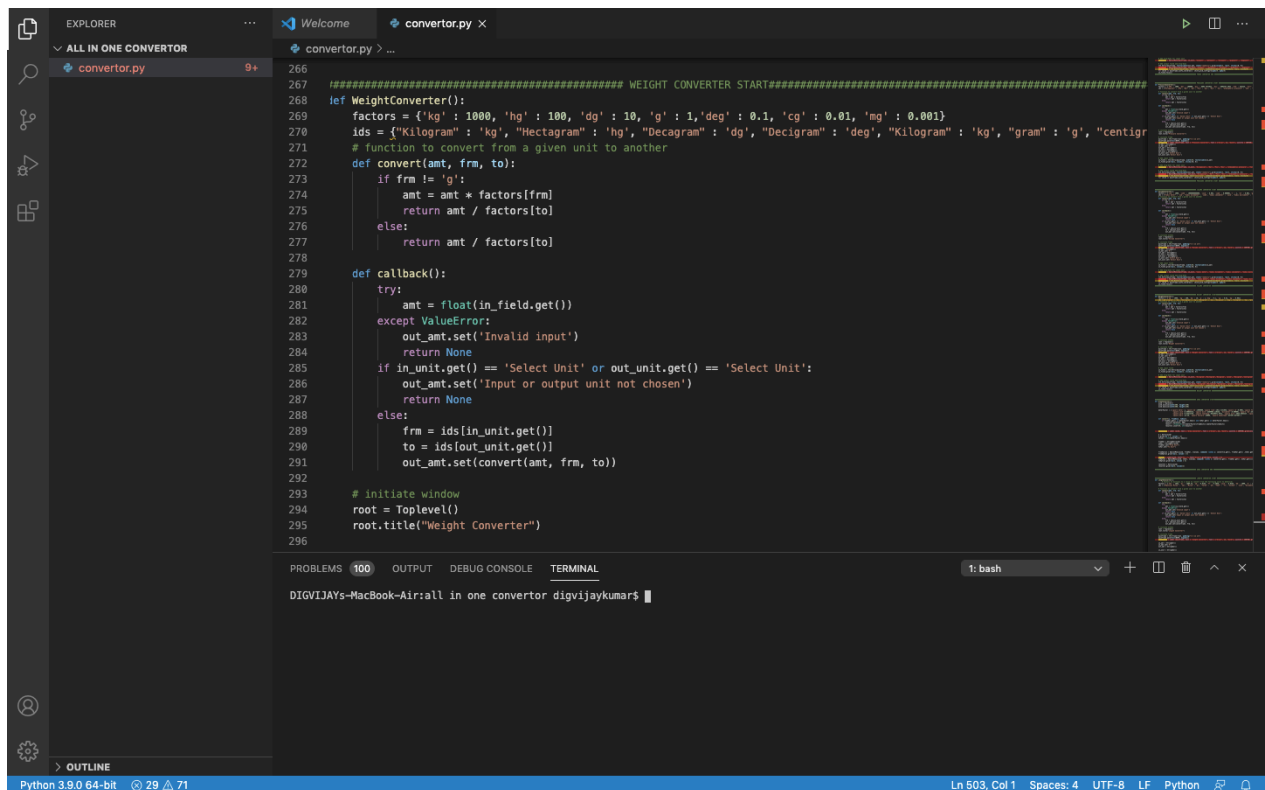
in_unit.set('Select Unit')
out_unit.set('Select Unit')

# Add input field
in_field = ttk.Entry(mainframe, width=20,
textvariable=in_amt)
in_field.grid(row=1, column=2, sticky=(W, E))

# Add drop-down for input unit
in_select = OptionMenu(mainframe, in_unit,
"Kilogram", "Hectagram", "Decagram", "gram",
"Decigram", "Centigram", "Milligram").grid(column=3, row=1,
sticky=W)

# Add output field and drop-down
ttk.Entry(mainframe, textvariable=out_amt,
state="readonly").grid(column=2, row=3, sticky=(W, E))
in_select = OptionMenu(mainframe, out_unit,
"Kilogram", "Hectagram", "Decagram", "gram",
"Decigram", "Centigram", "Milligram").grid(column=3, row=3,
sticky=W)
calc_button = ttk.Button(mainframe, text="Calculate",
command=callback).grid(column=2, row=2, sticky=E)
for child in mainframe.winfo_children():
child.grid_configure(padx=5, pady=5)
in_field.focus()
##### WEIGHT CONVERTER
END#####
#####

```





```
##### AREA CONVERTER
START#####
#####
```

```
def AreaConverter():
    wind = Toplevel()
    wind.minsize(width=400, height=150)
    wind.maxsize(width=400, height=150)

    meterFactor = {'square meter':1, 'square km':1000000, 'square
rood':1011.7141056, 'square cm':0.0001, 'square foot':0.09290304 ,
                    'square inch':0.00064516, 'square mile':
2589988.110336, 'milimeter':0.000001, 'square rod':25.29285264,
                    'square yard':0.83612736, 'square township':
93239571.9721, 'square acre':4046.8564224 , 'square are': 100,
                    'square barn':1e-28, 'square hectare':10000,
'square homestead':647497.027584 }

    def convert(x, fromUnit, toUnit):
        if fromVar.get() in meterFactor.keys() and toVar.get() in
meterFactor.keys():
            resulttxt.delete(0, END)
            result = (float(str(x))*meterFactor[fromUnit])/
(meterFactor[toUnit])
            resulttxt.insert(0, str(result))

    titleLabel = Label (wind, text = "Area Converter", font =
("Arial", 12, "bold"), justify = CENTER).grid(column=1,row=1)

    e = Entry(wind)
    e.grid(row = 1, column = 2)
    values = list(meterFactor.keys())

    fromVar = StringVar(wind)
    toVar = StringVar(wind)
    fromVar.set("From Unit")
    toVar.set("To Unit")

    fromOption = OptionMenu(wind, fromVar, *values, command=
lambda y: convert(e.get(), fromVar.get(), toVar.get()))
    fromOption.grid(row=1, column = 3)

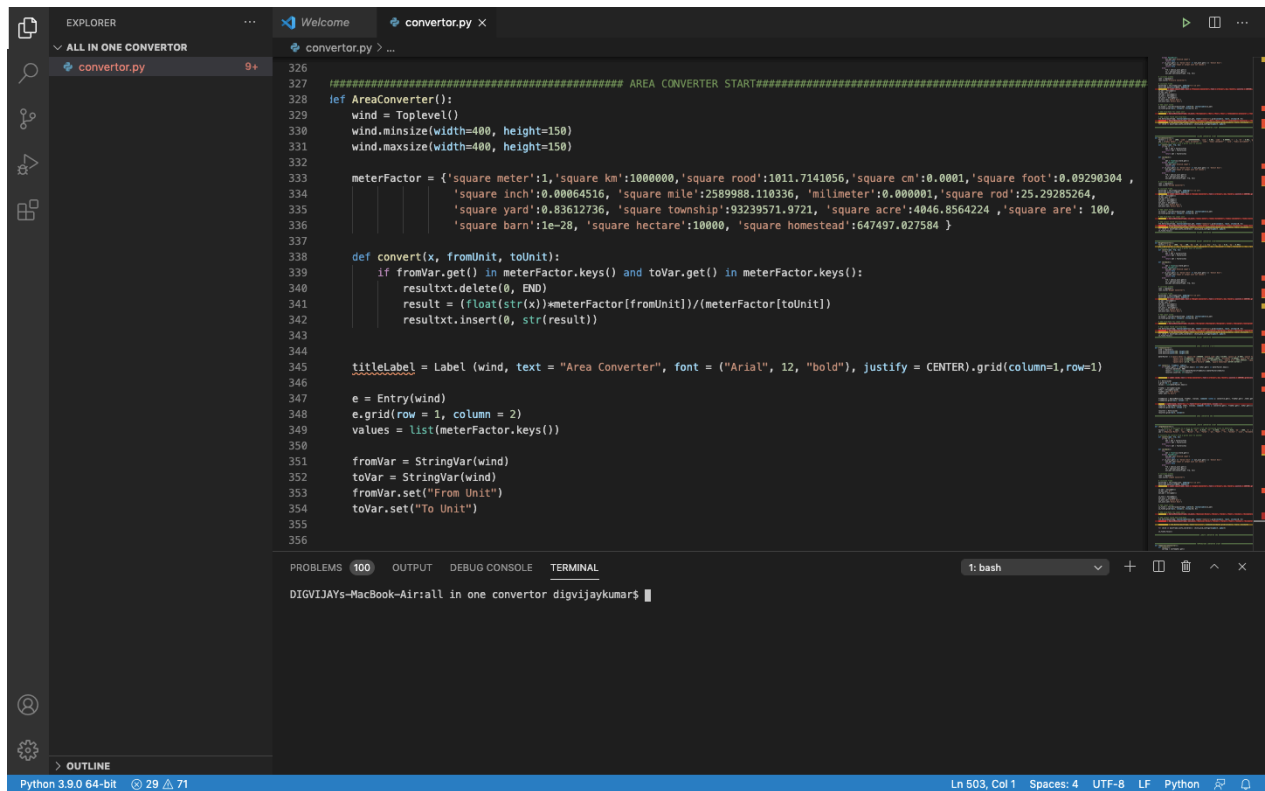
    toLabel = Label(wind, text="To : ", font="Arial").grid(row=2,
column = 2)
    toOption = OptionMenu(wind, toVar, *values, command= lambda
x: convert(e.get(), fromVar.get(), toVar.get()))
```

```
toOption.grid(row=3, column = 3)
```

```
resulttxt = Entry(wind)
resulttxt.grid(row=3, column=2)
```

```
##### AREA CONVERTER
END
```

```
#####
#####
```



```
326 ##### AREA CONVERTER START#####
327
328 def AreaConverter():
329     wind = Tk()
330     wind.minsize(width=400, height=150)
331     wind.maxsize(width=400, height=150)
332
333     meterFactor = {'square meter':1,'square km':1000000,'square rood':1011.7141056,'square cm':0.0001,'square foot':0.00298384 ,
334                   'square inch':0.00064516, 'square mile':2589988.110336, 'millimeter':0.000001,'square rod':25.29285264,
335                   'square yard':0.83612736, 'square township':932239571.9721, 'square acre':4846.8564224, 'square are': 100,
336                   'square barn':1e-28, 'square hectare':10000, 'square homestead':647497.027584 }
337
338     def convert(x, fromUnit, toUnit):
339         if fromVar.get() in meterFactor.keys() and toVar.get() in meterFactor.keys():
340             resulttxt.delete(0, END)
341             result = (float(str(x))*meterFactor[fromUnit])/(meterFactor[toUnit])
342             resulttxt.insert(0, str(result))
343
344     titleLabel = Label (wind, text = "Area Converter", font = ("Arial", 12, "bold"), justify = CENTER).grid(column=1,row=1)
345
346     e = Entry(wind)
347     e.grid(row = 1, column = 2)
348     values = list(meterFactor.keys())
349
350     fromVar = StringVar(wind)
351     toVar = StringVar(wind)
352     fromVar.set("From Unit")
353     toVar.set("To Unit")
354
355
356
```

```
##### LENGTH CONVERTER
START
#####
#####
def LengthConverter():
    # factors to multiply to a value to convert from the
    # following units to meters(m)
    factors = {'nmi' : 1852, 'mi' : 1609.34, 'yd' : 0.9144,
               'ft' : 0.3048, 'inch' : 0.0254, 'km' : 1000, 'm' : 1, 'cm' :
               0.01, 'mm' : 0.001}
    ids = {"Nautical Miles" : 'nmi', "Miles" : 'mi', "Yards" :
           'yd', "Feet" : 'ft', "Inches" : 'inch', "Kilometers" : 'km',
           "meters" : 'm', "centimeters" : 'cm', "millileters" : 'mm'}

    # function to convert from a given unit to another
    def convert(amt, frm, to):
        if frm != 'm':
            amt = amt * factors[frm]
            return amt / factors[to]
        else:
            return amt / factors[to]

    def callback():
        try:
            amt = float(in_field.get())
        except ValueError:
            out_amt.set('Invalid input')
            return None
        if in_unit.get() == 'Select Unit' or out_unit.get() ==
'Select Unit':
            out_amt.set('Input or output unit not chosen')
            return None
        else:
            frm = ids[in_unit.get()]
            to = ids[out_unit.get()]
            out_amt.set(convert(amt, frm, to))

    # initiate window
    root = Toplevel()
    root.title("Length Converter")

    # initiate frame
    mainframe = ttk.Frame(root, padding="3 3 12 12")
    mainframe.pack(fill=BOTH, expand=1)
    titleLabel = Label(mainframe, text = "Length Converter",
font = ("Arial", 12, "bold"), justify =
CENTER).grid(column=1, row=1)
```

```

in_amt = StringVar()
in_amt.set('0')
out_amt = StringVar()

in_unit = StringVar()
out_unit = StringVar()
in_unit.set('Select Unit')
out_unit.set('Select Unit')

# Add input field
in_field = ttk.Entry(mainframe, width=20,
textvariable=in_amt)
in_field.grid(row=1, column=2, sticky=(W, E))

# Add drop-down for input unit
in_select = OptionMenu(mainframe, in_unit, "Nautical Miles",
"Miles", "Yards", "Feet", "Inches", "Kilometers", "meters",
"centimeters", "millileters").grid(column=3, row=1, sticky=W)

# Add output field and drop-down
ttk.Entry(mainframe, textvariable=out_amt,
state="readonly").grid(column=2, row=3, sticky=(W, E))
in_select = OptionMenu(mainframe, out_unit, "Nautical Miles",
"Miles", "Yards", "Feet", "Inches", "Kilometers", "meters",
"centimeters", "millileters").grid(column=3, row=3, sticky=W)

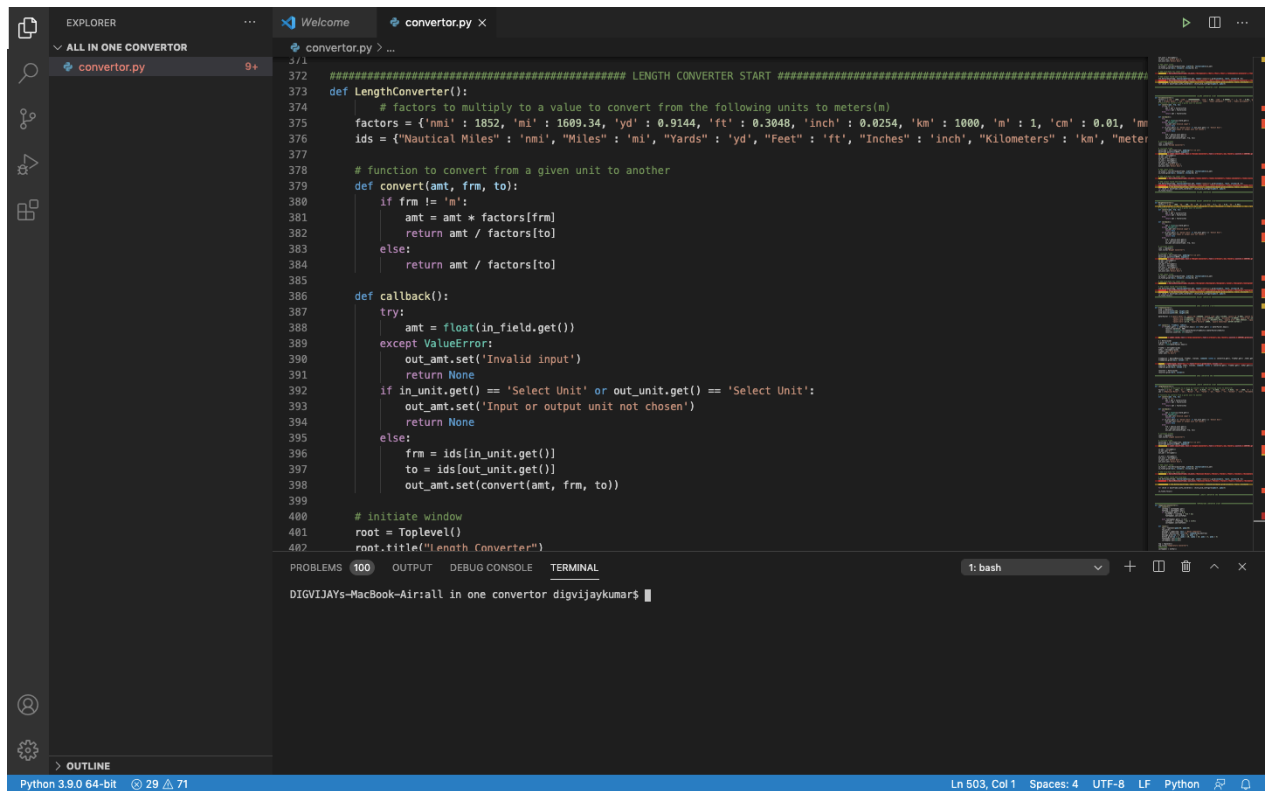
calc_button = ttk.Button(mainframe, text="Calculate",
command=callback).grid(column=2, row=2, sticky=E)

for child in mainframe.winfo_children():
child.grid_configure(padx=5, pady=5)

in_field.focus()

##### LENGTH
CONVERTER END
#####
#####

```



```

##### TEMPERATURE
CONVERTER START
#####
def TemperatureConverter():
    def convert():
        celTemp = celTempVar.get()
        fahTemp = fahTempVar.get()
        if celTempVar.get() != 0.0:
            celToFah = (celTemp * 9/5 + 32)
            fahTempVar.set(celToFah)

        elif fahTempVar.get() != 0.0:
            fahToCel = ((fahTemp - 32) * (5/9))
            celTempVar.set(fahToCel)

    def reset():
        top = Toplevel(padx=50, pady=50)
        top.grid()
        message = Label(top, text = "Reset Complete")

```

```

        button = Button(top, text="OK", command=top.destroy)
        message.grid(row = 0, padx = 5, pady = 5)
        button.grid(row = 1, padx = 10, pady = 10, padx = 5,
pady = 5)
        fahTempVar.set(int(0))
        celTempVar.set(int(0))

    top = Toplevel()
    top.title("Temperature Converter")
    ###MAIN###
    celTempVar = IntVar()
    celTempVar.set(int(0))
    fahTempVar = IntVar()
    fahTempVar.set(int(0))
    titleLabel = Label (top, text = "Temperature Converter", font
= ("Arial", 12, "bold"), justify = CENTER).grid(column=1,row=1)

    cellLabel = Label (top, text = "Celcius: ", font = ("Arial",
14), fg = "black")
    cellLabel.grid(row = 2, column = 1, pady = 10, sticky = NW)

    fahLabel = Label (top, text = "Fahrenheit: ", font =
("Arial", 14), fg = "black")
    fahLabel.grid(row = 3, column = 1, pady = 10, sticky = NW)

    celEntry = Entry (top, width = 10, bd = 5, textvariable =
celTempVar)
    celEntry.grid(row = 2, column = 1, pady = 10, sticky = NW,
padx = 125 )

    fahEntry = Entry (top, width = 10, bd = 5, textvariable =
fahTempVar)
    fahEntry.grid(row = 3, column = 1, pady = 10, sticky = NW,
padx = 125 )

    convertButton =Button (top, text = "Convert", font =
("Arial", 8, "bold"), relief = RAISED, bd=5, justify = CENTER,
highlightbackground = "red", overrelief = GROOVE,
activebackground = "green", activeforeground="blue", command =
convert)
    convertButton.grid(row = 4, column = 1, pady = 8, padx =
12, pady = 5, sticky = NW, padx = 55)

    resetButton = Button (top, text = "Reset", font = ("Arial",
8, "bold"), relief = RAISED, bd=5, justify = CENTER,
highlightbackground = "red", overrelief = GROOVE,

```

```

activebackground = "green", activeforeground="blue", command =
reset)
    resetButton.grid(row = 4, column = 2, ipady = 8, ipadx = 12,
pady = 5, sticky = NW)

```

```

##### TEMPERATURE
CONVERTER END
#####
#####

```

```

439 ##### TEMPERATURE CONVERTER START #####
440
441 def TemperatureConverter():
442     def convert():
443         celTemp = celTempVar.get()
444         fahTemp = fahTempVar.get()
445         if celTempVar.get() != 0.0:
446             celToFah = (celTemp * 9/5 + 32)
447             fahTempVar.set(celToFah)
448
449         elif fahTempVar.get() != 0.0:
450             fahToCel = ((fahTemp - 32) * (5/9))
451             celTempVar.set(fahToCel)
452
453     def reset():
454         top = Toplevel(padx=50, pady=50)
455         top.grid()
456         message = Label(top, text="Reset Complete")
457         button = Button(top, text="OK", command=top.destroy)
458         message.grid(row = 0, padx = 5, pady = 5)
459         button.grid(row = 1, ipadx = 10, ipady = 10, padx = 5, pady = 5)
460         fahTempVar.set(int(0))
461         celTempVar.set(int(0))
462
463     top = Toplevel()
464     top.title("Temperature Converter")
465     #####
466     celTempVar = IntVar()
467     celTempVar.set(int(0))
468     fahTempVar = IntVar()
469     fahTempVar.set(int(0))
470
471     top.mainloop()
472
473 if __name__ == '__main__':
474     TemperatureConverter()
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
1000

```

```

widget = Button(root, text="Temperature converter", bg="white" ,
fg="black",font = ("Arial", 12, "bold"), relief = RAISED, bd=5,
justify = CENTER, highlightbackground = "red", overrelief =
GROOVE, activebackground = "green", activeforeground="blue",
command=TemperatureConverter).place(x=170,y=80)
widget = Button(root, text="Length Converter", bg="white" ,
fg="black",font = ("Arial", 12, "bold"), relief = RAISED, bd=5,
justify = CENTER, highlightbackground = "red", overrelief =
GROOVE, activebackground = "green", activeforeground="blue",
command=LengthConverter).place(x=190,y=140)

```

```

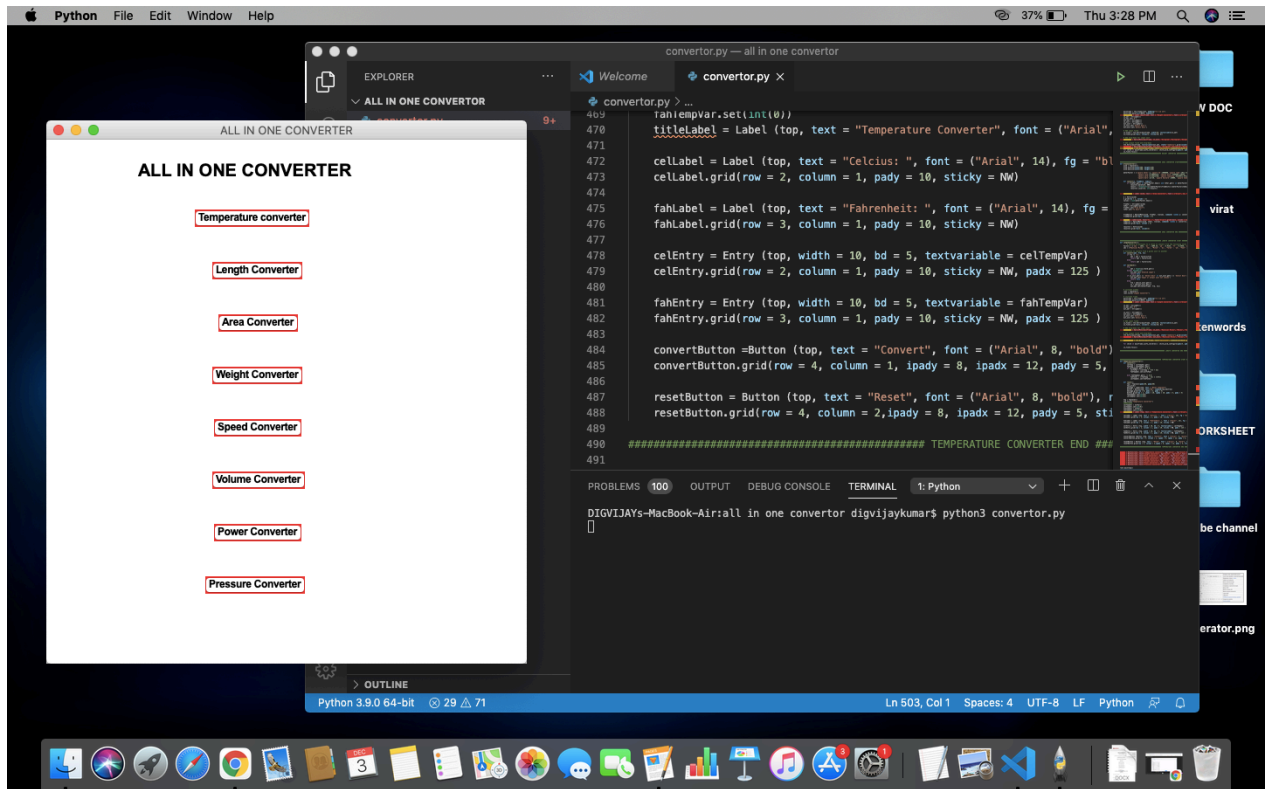
widget = Button(root, text="Area Converter", bg="white" ,
fg="black",font = ("Arial", 12, "bold"), relief = RAISED, bd=5,
justify = CENTER, highlightbackground = "red", overrelief =
GROOVE, activebackground = "green", activeforeground="blue",
command=AreaConverter).place(x=197,y=200)
widget = Button(root, text="Weight Converter", bg="white" ,
fg="black",font = ("Arial", 12, "bold"), relief = RAISED, bd=5,
justify = CENTER, highlightbackground = "red", overrelief =
GROOVE, activebackground = "green", activeforeground="blue",
command=WeightConverter).place(x=190,y=260)
widget = Button(root, text="Speed Converter", bg="white" ,
fg="black",font = ("Arial", 12, "bold"), relief = RAISED, bd=5,
justify = CENTER, highlightbackground = "red", overrelief =
GROOVE, activebackground = "green", activeforeground="blue",
command=SpeedConverter).place(x=192,y=320)
widget = Button(root, text="Volume Converter", bg="white" ,
fg="black",font = ("Arial", 12, "bold"), relief = RAISED, bd=5,
justify = CENTER, highlightbackground = "red", overrelief =
GROOVE, activebackground = "green", activeforeground="blue",
command=VolumeConverter).place(x=190,y=380)
widget = Button(root, text="Power Converter", bg="white" ,
fg="black",font = ("Arial", 12, "bold"), relief = RAISED, bd=5,
justify = CENTER, highlightbackground = "red", overrelief =
GROOVE, activebackground = "green", activeforeground="blue",
command=PowerConverter).place(x=192,y=440)
widget = Button(root, text="Pressure Converter", bg="white" ,
fg="black",font = ("Arial", 12, "bold"), relief = RAISED, bd=5,
justify = CENTER, highlightbackground = "red", overrelief =
GROOVE, activebackground = "green", activeforeground="blue",
command=PressureConverter).place(x=182,y=500)

root.mainloop()

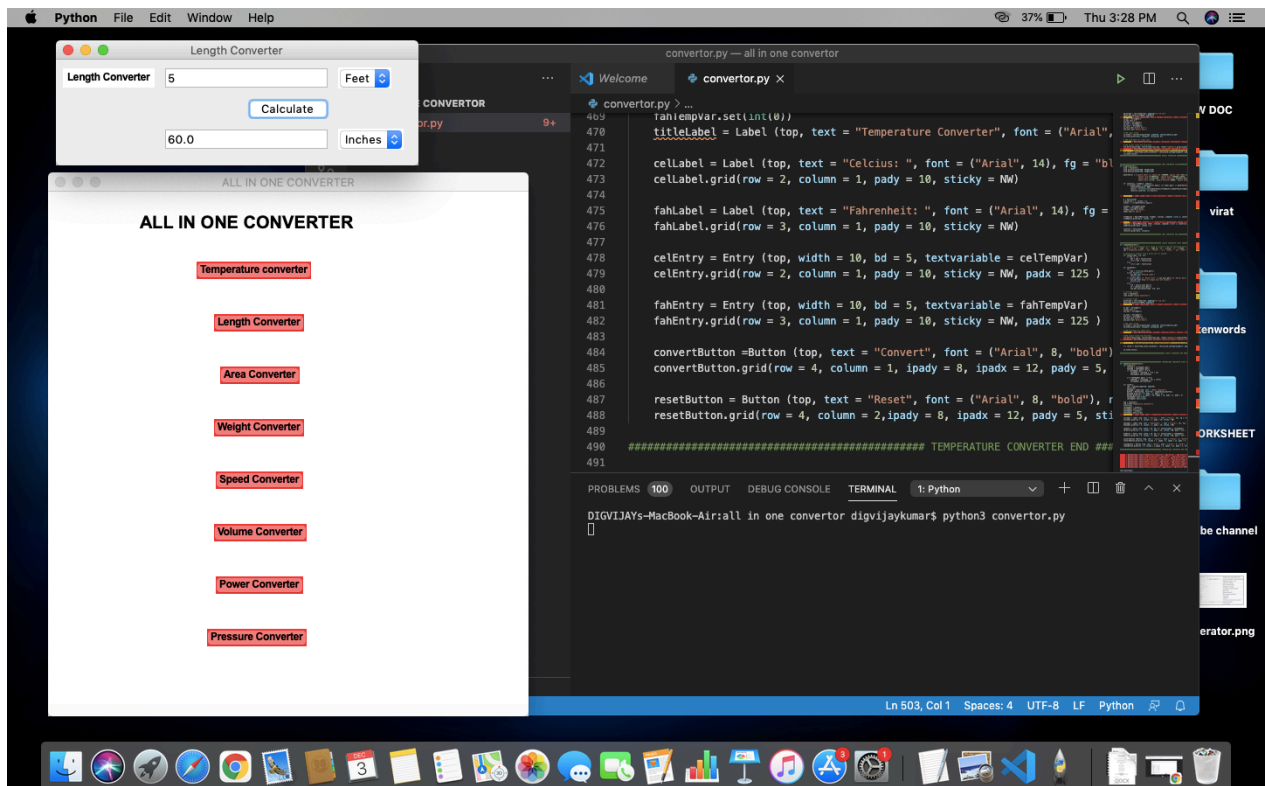
```

## Output Analysis (screenshots)





Here we convert 5 feet into 60 inches through length convertor. Similarly we can convert other quantities.



**THANK YOU**