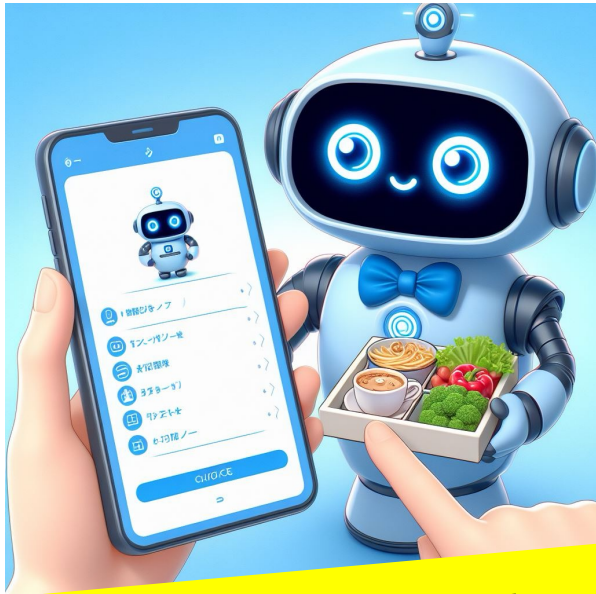


Interactive programs!



Letting the user choose...
(and more loopiness!)

Plus, if you've got a time machine...



Possible
hardware



User input...

```
meters = input('How many m? ')\n\ncm = meters * 100\n\nprint('That's', cm, 'cm.')
```

What will Python think?

I think I like these units better
than light years per year!



User input...

```
meters = input('How many m? ')
```

```
cm = meters * 100
```

```
print("That's", cm, 'cm.')
```

input ALWAYS returns a **string** –
no matter what's typed!

What will Python think?

I think I like these units better
than light years per year!



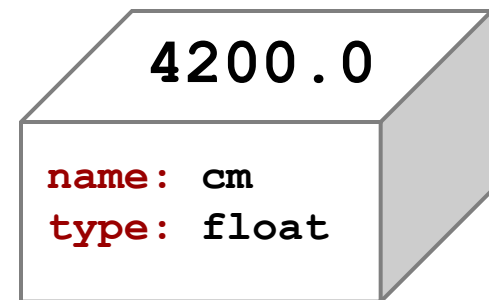
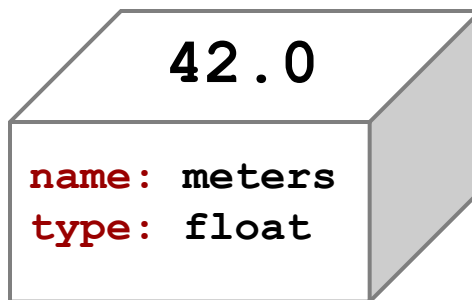
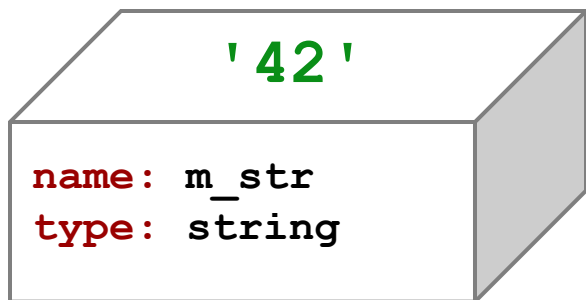
Fix #1: **convert** to the right type

```
m_str = input('How many m? ')
```

```
meters = float(m_str)
```

```
cm = meters * 100
```

```
print("That's", cm, 'cm.')
```



... but **crash**-able

Fix #2: **convert** and **check**

```
m_str = input('How many m? ')
```

```
try:
```

```
    meters = float( m_str )
```

crash-able

```
except:
```

```
    print("What? Didn't compute!")
```

```
    print("Setting meters = 42")
```

```
    meters = 42.0
```

try-except lets you try code
and – if it crashes – catch an
error and handle it

```
cm = meters * 100
```

```
print('That\'s', cm, 'cm.')
```

Ex. #2

User-errors are called *exceptions*.
This is *exception handling*.

I except!



try:

```
meters = float( m_str )
```

crash-able

except:

```
print("What? Didn't compute!")
```

```
print("Setting meters = 42")
```

```
meters = 42.0
```

try-except lets you try code
and – if it crashes – catch an
error and handle it

```
cm = meters * 100
```

```
print('That\'s', cm, 'cm.')
```

Fix #3: **eval** executes Python code!

```
m_str = input('How many m? ')
```

```
meters = eval( m_str )
```

```
cm = meters * 100
```

```
print('That is', cm, 'cm.')
```

What could go wrong here?

Fix #3: **eval** executes Python code!

```
m_str = input('How many m? ')
```

```
try:
```

```
    meters = eval( m_str )
```

```
except:
```

```
    print("What? Didn't compute!")
```

```
    print("Setting meters = 42")
```

```
    meters = 42.0
```

What could REALLY go wrong here?

```
cm = meters * 100
```

```
print('That is', cm, 'cm.')
```

Eval? More like
Evil !!



More loop control...

```
# Using return to return early from a function
def loopy0():
    for i in range(1,10):
        print(i)
        if i % 3 == 0:
            return
    print("All done!")
```

```
# Using break to exit a loop early
def loopy1():
    for i in range(1,10):
        print(i)
        if i % 3 == 0:
            break
    print("All done!")
```

I need a break!



More loop control...

Using return to return early from a function

```
def loopy0():  
    for i in range(1,10):  
        print(i)  
        if i % 3 == 0:  
            return  
    print("All done!")
```

Using continue to start a new iteration

```
def loopy2():  
    for i in range(1,10):  
        if i % 3 == 0:  
            continue  
        print(i)  
    print("All done!")
```

Using break to exit a loop early

```
def loopy1():  
    for i in range(1,10):  
        print(i)  
        if i % 3 == 0:  
            break  
    print("All done!")
```

Using pass to do nothing

```
def loopy3():  
    for i in range(1,10):  
        if i % 3 == 0:  
            pass  
        else:  
            print(i)  
    print("All done!")
```

Mystery sequences...

[-35, -24, -13, -2, 9, 20, 31, ?]

[26250, 5250, 1050, 210, ?]

[90123241791111, 93551622, 121074, 3111, ?]

[1, 11, 21, 1211, 111221, ?]

What's next?

I'm glad you asked!



A larger application ...

```
def menu():  
    """ prints our menu of options """  
    print("(0) Continue")  
    print("(1) Enter a new list")  
    print("(2) Analyze")  
    print("(9) Break (quit)")
```

```
def main():  
    """ handles user input for our menu """
```

```
    while True:  
        menu()  
        uc = input('Which option? ')
```

Calls a helper
function

```
        try:  
            uc = int(uc)           # was it an int?
```

```
        except:  
            continue             # back to the top!
```

Perhaps `uc` the
reason for this?



```
def main():  
    """ handles user input for our menu """  
    L = [30,10,20] # a starting list
```

```
    while True:  
        menu() # print menu  
        uc = input('Which option? ')
```

```
        if uc == 9:
```

```
            break
```

(9) Quit

```
        elif uc == 0:
```

```
            continue
```

(0) Continue

```
        elif uc == 1:
```

```
            ... input ... eval ...
```

(1) Get new list

```
        elif uc == 2:
```

(2) Analyze !

... and so on ...

```
def main():  
    """ handles user input for our menu """  
    L = [30,10,20] # a starting list
```

```
while True:  
    menu() # print menu  
    uc = input('Which option? ')
```

```
if uc == 9:
```

```
    break
```

(9) Quit

break breaks out of the loop...

```
elif uc == 0:
```

```
    continue
```

(0) Continue

continue jumps back to the top...

```
elif uc == 1:
```

```
    ... input ... eval ...
```

(1) Get new list

uses **eval** (+check) for a new L

```
elif uc == 2:
```

(2) Analyze !

other functions as needed...

... and so on ...

[0] Which line of code handles an input of 1 ? 227

[1] Which line of code handles an input of 5 ? 254

Big-picture view!

[4] What line of code runs after this break ? and continue ? 264 206



[2] Which line below handles an input of 7 ? 258

[3] What does input 3 print that 0 does not ?

"Looping back again"

[6a] What could you input for newL that would reach line 235?

'42'

[6b] how about reaching line 239?

main function

secret_value

while True:

input

(option from menu)

[5] Where is predict defined? 188

Full-program menu-interaction example

[EC] How could a user learn the value of `secret_value` if they guessed that variable name and could run the program -- but *didn't* have this source code?

[secret_value]

Try it!

```
175 #
176 # example looping program
177 #
178
179 def menu():
180     """ a function that simply prints the menu """
181     print("\n")
182     print("(0) Continue!")
183     print("(1) Enter a new list")
184     print("(2) Analyze! (next element)")
185     print("(9) Break (Quit)")
186     print()
187
188 def predict(L):
189     """ predict ignores its input and returns
190     what the next element _should_ have been
191     """
192     return 42
193
194 def main():
195     """ the main user-interaction """
196     print("\n")
197     print("+++++")
198     print("Welcome to the PREDICTOR!")
199     print("+++++")
200     print()
201
202     secret_value = 4.2
203
204     L = [30,10,20] # an initial list
205
206     while True: # the user-interaction loop
207         print("\nThe list is", L)
208         menu()
209         uc = input("Choose an option: ")
210
211         # "clean and check" the user's input
212         #
213         try:
214             uc = int(uc) # make into an int!
215         except:
216             print("I didn't understand your input! Continuing...")
217             continue
218
219         # run the appropriate menu option
220         #
```

```
221
222     if uc == 9: # we want to quit
223         break # leaves the while loop altogether
224
225     elif uc == 0: # we want to continue...
226         continue # goes back to the top of the while loop
227
228     elif uc == 1: # we want to enter a new list
229         newL = input("Enter a new list: ") # enter _something_
230
231         # "clean and check" the user's input
232         #
233         try:
234             newL = eval(newL) # eval runs Python's interpreter! Danger!
235             if type(newL) != list:
236                 print("That wasn't of type list. Not changing L.")
237             else:
238                 L = newL # here, things were OK, so let's set our list, L
239         except:
240             print("I didn't understand your input. Not changing L.")
241
242     elif uc == 2: # predict and add the next element
243         n = predict(L) # get the next element from the predict function
244         print("The next element is", n)
245         print("Adding it to your list...")
246         L = L + [n] # and add it to the list
247
248     elif uc == 3: # unannounced menu option!
249         pass # this is the "nop" (do-nothing) statement in Python
250
251     elif uc == 4: # unannounced menu option (slightly more interesting...)
252         m = find_min(L)
253         print("The minimum value in L is", m)
254
255     elif uc == 5: # another unannounced menu option (even more interesting...)
256         minval, minloc = find_min_loc(L)
257         print("The minimum value in L is", minval, "at day #", minloc)
258
259     else:
260         print(uc, " ? That's not on the menu!")
261
262     # last line of code while True loop
263     print("\nLooping back again... !\n")
264
265     print()
266     print("I predict... \n\n ... that you'll be back!")
```

[0] Which line of code handles an input of 1 ?

[1] Which line of code handles an input of 5 ?

227

254

Big-picture view!

[4] What line of code runs after this **break** ? and **continue** ?



[2] Which line below handles an input of 7 ?

[3] What does input 3 print that 0 does not?

[6a] What could you input for newL that would reach line 235?

[6b] how about reaching line 239?

main function

secret_value

while True:

input

(option from menu)

[5] Where is predict defined?

input
(new list)

```
175 #
176 # example looping program
177 #
178
179 def menu():
180     """ a function that simply prints the menu """
181     print("\n")
182     print("(0) Continue!")
183     print("(1) Enter a new list")
184     print("(2) Analyze! (next element)")
185     print("(9) Break (Quit)")
186     print()
187
188 def predict(L):
189     """ predict ignores its input and returns
190     |   what the next element _should_ have been
191     """
192     return 42
193
194 def main():
195     """ the main user-interaction """
196     print("\n")
197     print("+++++")
198     print("Welcome to the PREDICTOR!")
199     print("+++++")
200     print()
201
202     secret_value = 4.2
203
204     L = [30,10,20] # an initial list
205
206     while True: # the user-interaction loop
207         print("\nThe list is", L)
208         menu()
209         uc = input( "Choose an option: " )
210
211         # "clean and check" the user's input
212         #
213         try:
214             uc = int(uc) # make into an int!
215         except:
216             print("I didn't understand your input! Continuing...")
217             continue
218
219         # run the appropriate menu option
220         #
```

```
220 #
221 if uc == 9: # we want to quit
222     break # leaves the while loop altogether
223
224 elif uc == 0: # we want to continue...
225     continue # goes back to the top of the while loop
226
227 elif uc == 1: # we want to enter a new list
228     newL = input("Enter a new list: ") # enter _something_
229
230     # "clean and check" the user's input
231     #
232     try:
233         newL = eval(newL) # eval runs Python's interpreter! Danger!
234         if type(newL) != list:
235             print("That wasn't of type list. Not changing L.")
236         else:
237             L = newL # here, things were OK, so let's set our list, L
238     except:
239         print("I didn't understand your input. Not changing L.")
240
241 elif uc == 2: # predict and add the next element
242     n = predict(L) # get the next element from the predict function
243     print("The next element is", n)
244     print("Adding it to your list...")
245     L = L + [n] # and add it to the list
246
247 elif uc == 3: # unannounced menu option!
248     pass # this is the "nop" (do-nothing) statement in Python
249
250 elif uc == 4: # unannounced menu option (slightly more interesting...)
251     m = find_min(L)
252     print("The minimum value in L is", m)
253
254 elif uc == 5: # another unannounced menu option (even more interesting...)
255     minval, minloc = find_min_loc(L)
256     print("The minimum value in L is", minval, "at day #", minloc)
257
258 else:
259     print(uc, " ? That's not on the menu!")
260
261 # last line of code while True loop
262 print("\nLooping back again... !\n")
263
264 print()
265 print("I predict... \n\n ... that you'll be back!")
266
```

Full-program menu-interaction example

[EC] How could a user learn the value of **secret_value** if they guessed that variable name and could run the program -- but *didn't have this source code*?

Try it!

[0] Which line of code handles an input of 1 ?

[1] Which line of code handles an input of 5 ?

227

254

Big-picture view!

[4] What line of code runs after this **break** ? and **continue** ?



[2] Which line below handles an input of 7 ?

[3] What does input 3 print that 0 does not?

[6a] What could you input for newL that would reach line 235?

[6b] how about reaching line 239?

main function

secret_value

while True:

input

(option from menu)

[5] Where is predict defined?

input

(new list)

```
175 #
176 # example looping program
177 #
178
179 def menu():
180     """ a function that simply prints the menu """
181     print("\n")
182     print("(0) Continue!")
183     print("(1) Enter a new list")
184     print("(2) Analyze! (next element)")
185     print("(9) Break (Quit)")
186     print()
187
188 def predict(L):
189     """ predict ignores its input and returns
190     | what the next element _should_ have been
191     """
192     return 42
193
194 def main():
195     """ the main user-interaction """
196     print("\n")
197     print("+++++")
198     print("Welcome to the PREDICTOR!")
199     print("+++++")
200     print()
201
202     secret_value = 4.2
203
204     L = [30,10,20] # an initial list
205
206     while True: # the user-interaction loop
207         print("\nThe list is", L)
208         menu()
209         uc = input( "Choose an option: " )
210
211         # "clean and check" the user's input
212         #
213         try:
214             uc = int(uc) # make into an int!
215         except:
216             print("I didn't understand your input! Continuing...")
217             continue
218
219         # run the appropriate menu option
220         #
```

```
222
223
224         if uc == 9: # we want to quit
225             break # leaves the while loop altogether
226
227         elif uc == 0: # we want to continue...
228             continue # goes back to the top of the while loop
229
230         elif uc == 1: # we want to enter a new list
231             newL = input("Enter a new list: ") # enter _something_
232
233             # "clean and check" the user's input
234             #
235             try:
236                 newL = eval(newL) # eval runs Python's interpreter! Danger!
237                 if type(newL) != list:
238                     print("That wasn't of type list. Not changing L.")
239                 else:
240                     L = newL # here, things were OK, so let's set our list, L
241             except:
242                 print("I didn't understand your input. Not changing L.")
243
244         elif uc == 2: # predict and add the next element
245             n = predict(L) # get the next element from the predict function
246             print("The next element is", n)
247             print("Adding it to your list...")
248             L = L + [n] # and add it to the list
249
250         elif uc == 3: # unannounced menu option!
251             pass # this is the "nop" (do-nothing) statement in
252
253         elif uc == 4: # unannounced menu option (slightly more interesting...)
254             m = find_min(L)
255             print("The minimum value in L is", m)
256
257         elif uc == 5: # another unannounced menu option (even more interesting...)
258             minval, minloc = find_min_loc(L)
259             print("The minimum value in L is", minval, "at day #", minloc)
260
261         else:
262             print(uc, " ? That's not on the menu!")
263
264         # last line of code while True loop
265         print("\nLooping back again... !\n")
266
267     print()
268     print("I predict... \n\n ... that you'll be back!")
```

Full-program menu-interaction example

[EC] How could a user learn the value of **secret_value** if they guessed that variable name and could run the program -- but *didn't have this source code*?

Try it!

[0] Which line of code handles an input of 1 ?

[1] Which line of code handles an input of 5 ?

227

254

Big-picture view!

[4] What line of code runs after this **break** ? and **continue** ?



264

206/207

[2] Which line below handles an input of 7 ?

258

[3] What does input 3 print that 0 does not?

line 262

"42"
or
42

[6a] What could you input for newL that would reach line 235?

[6b] how about reaching line 239?

Python?
I prefer
Java

input
(new list)

[5] Where is predict defined?

188

main function

secret_value

while True:

input

(option from menu)

Full-program menu-interaction example

[EC] How could a user learn the value of **secret_value** if they guessed that variable name and could run the program -- but *didn't have this source code*?

Sols...

```
175 #
176 # example looping program
177 #
178
179 def menu():
180     """ a function that simply prints the menu """
181     print("\n")
182     print("(0) Continue!")
183     print("(1) Enter a new list")
184     print("(2) Analyze! (next element)")
185     print("(9) Break (Quit)")
186     print()
187
188 def predict(L):
189     """ predict ignores its input and returns
190     | what the next element _should_ have been
191     """
192     return 42
193
194 def main():
195     """ the main user-interaction """
196     print("\n")
197     print("+++++")
198     print("Welcome to the PREDICTOR!")
199     print("+++++")
200     print()
201
202     secret_value = 4.2
203
204     L = [30,10,20] # an initial list
205
206     while True: # the user-interaction loop
207         print("\nThe list is", L)
208         menu()
209         uc = input( "Choose an option: " )
210
211         # "clean and check" the user's input
212         #
213         try:
214             uc = int(uc) # make into an int!
215         except:
216             print("I didn't understand your input! Continuing...")
217             continue
218
219         # run the appropriate menu option
220         #
```

```
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
```

[0] Which line of code handles an input of 1 ?

[1] Which line of code handles an input of 5 ?

227

254

Big-picture view!

[4] What line of code runs after this **break** ? and **continue** ?



[2] Which line below handles an input of 7 ?

258

[3] What does input 3 print that 0 does not?

line 262

"42"
or
42

[6a] What could you input for newL that would reach line 235?

[6b] how about reaching line 239?

Python?
I prefer
Java

main function

secret_value

while True:

input

(option from menu)

[5] Where is predict defined?

188

input [0, 1, 2, secret_value]

Full-program menu-interaction example

[EC] How could a user learn the value of **secret_value** if they guessed that variable name and could run the program -- but *didn't have this source code*?

Sols...

```
175 #
176 # example looping program
177 #
178
179 def menu():
180     """ a function that simply prints the menu """
181     print("\n")
182     print("(0) Continue!")
183     print("(1) Enter a new list")
184     print("(2) Analyze! (next element)")
185     print("(9) Break (Quit)")
186     print()
187
188 def predict(L):
189     """ predict ignores its input and returns
190     | what the next element _should_ have been
191     """
192     return 42
193
194 def main():
195     """ the main user-interaction """
196     print("\n")
197     print("+++++")
198     print("Welcome to the PREDICTOR!")
199     print("+++++")
200     print()
201
202     secret_value = 4.2
203
204     L = [30,10,20] # an initial list
205
206     while True: # the user-interaction loop
207         print("\nThe list is", L)
208         menu()
209         uc = input( "Choose an option: " )
210
211         # "clean and check" the user's input
212         #
213         try:
214             uc = int(uc) # make into an int!
215         except:
216             print("I didn't understand your input! Continuing...")
217             continue
218
219         # run the appropriate menu option
220         #
```

```
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
```

Loops

```
def fac( N ) :  
    result = 1  
    for i in range(1, N+1):  
        result = result * i  
    return result
```

sequences!

Basic design
strategies

Is one more *reasonable*
than the other?

Recursion

```
def fac( N ) :  
    if N == 1:  
        return 1  
    else:  
        return N * fac(N-1)
```

self-similarity

Loops

Strategy: look for
repetition + use it....

Basic design
strategies

```
def fac( N ) :  
    result = 1  
    for x in range(1,N+1) :  
        result *= x  
    return result
```

Is one more *reasonable*
than the other?


Strategy: Look for
self-similarity + use it....

Recursion

```
def fac( N ) :  
    if N == 1 :  
        return 1  
    else :  
        return N*fac(N-1)
```

for: *two "loop patterns"*

`L = [3, 15, 17, 7]`



A blue variable `x` is shown below the list. Four blue arrows originate from `x` and point to each of the four elements in the list: 3, 15, 17, and 7.

"deceptively easy"

elements



```
for x in L:  
    total += x
```



element-based loops

for: two "loop patterns"

$L = [3, 15, 17, 7]$

0

1

2

3

← *indexes*

indices

("indexes")

for **i** in range(len(L))
total += L[**i**]

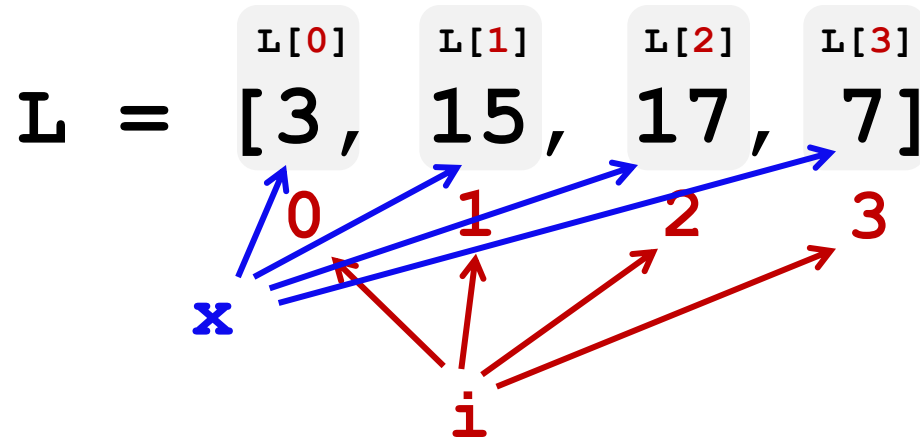
index-based loops
— access data indirectly,
(by its *index*)

elements

for **x** in L:
total += **x**

element-based loops
— access data directly

for: *two "loop patterns"*



```
for i in range(len(L))  
    total += L[i]
```

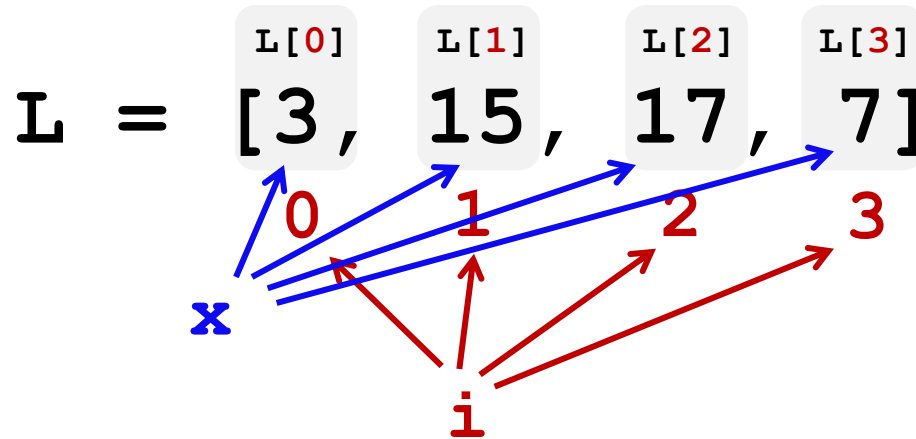
index-based loops
— access data indirectly

Elements vs **Indexes**
Indices

```
for x in L:  
    total += x
```

element-based loops
— access data directly

for: *two variables*



```
for i in range(len(L))  
    total += L[i]
```

index-based

i

Elements vs **Indexes**

Indices

```
for x in
```

```
    total
```

x

element-based loops

"Get into a rut...
and stay there!"



hw8pr4: T. T. Securities (TTS)

hw8pr4: T. T. Securities (TTS)

Analyzing a sequence of ... *anything!*

i

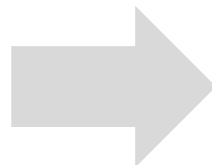
	day 0	day 1	day 2	day 3	day 4	day 5	day 6	day 7
<i>L</i> =	[40 ,	80 ,	10 ,	30 ,	27 ,	52 ,	5 ,	15]

x

indices

elements

tasks



- (0) Input a new list
- (1) Print the current list
- (2) Find the average price
- (3) Find the standard deviation
- (4) Find the min and its day
- (5) Find the max and its day
- (6) Your TTS investment plan
- (9) Quit

Enter your choice:

hw8pr4: T. T. Securities (TTS)

Analyzing a sequence of ... *stock prices?!*

i

day 0 day 1 day 2 day 3 day 4 day 5 day 6 day 7

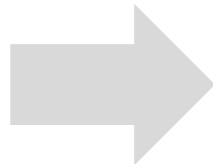
L = [40 , 80 , 10 , 30 , 27 , 52 , 5 , 15]

x

indices
("indexes")

elements

tasks



- (0) Input a new list
- (1) Print the current list
- (2) Find the average price
- (3) Find the standard deviation
- (4) Find the min and its day
- (5) Find the max and its day
- (6) Your TTS investment plan
- (9) Quit

Enter your choice:

hw8pr4: T. T. Securities (TTS)

Analyzing a sequence of ... *stock prices?!*

indices ~ days

i

day 0 day 1 day 2 day 3 day 4 day 5 day 6 day 7

L = [40 , 80 , 10 , 30 , 27 , 52 , 5 , 15]

x

elements ~ prices

Implement a
(text) menu:

- (0) Input a new list
- (1) Print the current list
- (2) Find the average price
- (3) Find the standard deviation
- (4) Find the min and its day
- (5) Find the max and its day
- (6) Your TTS investment plan
- (9) Quit

Enter your choice:



One motivation for TT securities...

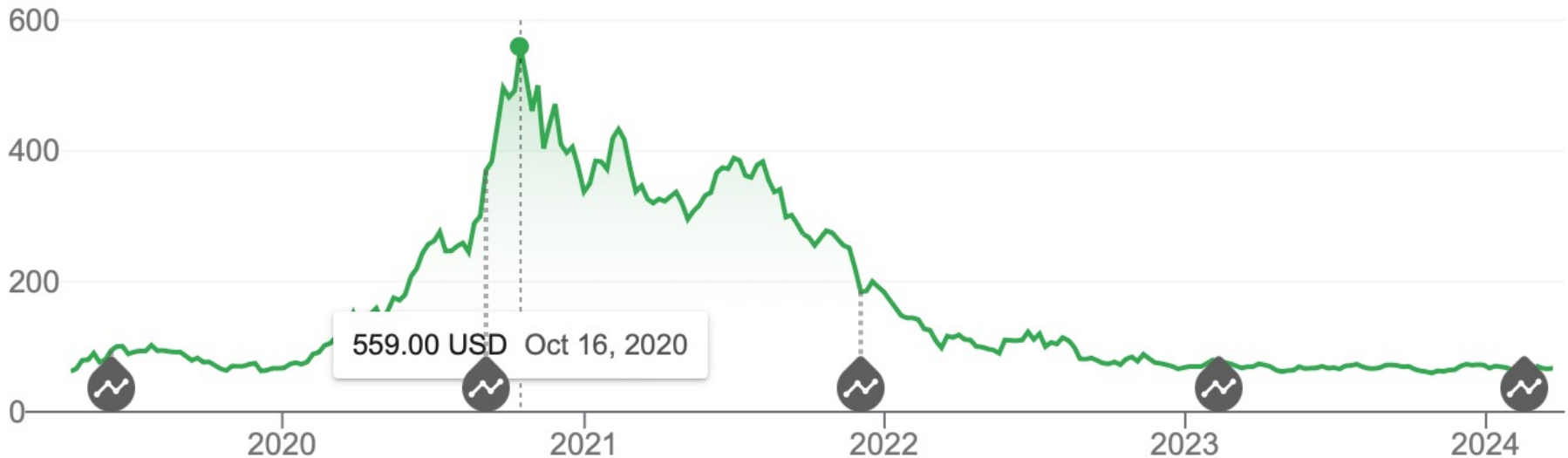
Market Summary > Zoom Video Communications Inc

66.94 USD

+4.94 (7.98%) ↑ past 5 years

Mar 21, 12:29 PM EDT • Disclaimer

1D | 5D | 1M | 6M | YTD | 1Y | 5Y | Max



Name(s) _____

The TTS-strategy:

[0] T.T. Securities's customer pledge:
"We select the day to buy and day to sell that
will maximize your price-difference..."*

Your stock's prices: $L = [40, 80, 10, 30, 27, 52, 5, 15]$

<i>index</i>	<i>element</i>
Day	Price
0	40.0
1	80.0 ✗
2	10.0
3	30.0
4	27.0
5	52.0
6	5.0 ✗
7	15.0

42 [

[1] What is the best TTS investment strategy for **this list**, L?

[1b] Which day would you _buy_ (and at what price) ?

[1c] Which day would you _sell_ (and at what price) ?

[1d] What is the per-share profit in this best case? (!!!)

42

It's NOT 75!

for each buy-day, b:

for each sell-day, s: (b+1...end)

[2] How could **nested loops** help us find the
best TTS strategy? (a "code sketch...")

Important fine print:

this all seems sketch...

*To make our business plan **realistic**, however, we only allow selling after buying.



The TTS-strategy:

[0] T.T. Securities's customer pledge:
"We select the day to buy and day to sell that
will maximize your price-difference..."*

Your stock's prices: $L = [40, 80, 10, 30, 27, 52, 5, 15]$

<i>index</i>	<i>element</i>
Day	Price
0	40.0
1	80.0
2	10.0
3	30.0
4	27.0
5	52.0
6	5.0
7	15.0

[1] What is the best TTS investment strategy for **this list**, L?

[1b] Which day would you _buy_ (and at what price)?

[1c] Which day would you _sell_ (and at what price)?

[1d] What is the per-share profit in this best case? (!!!)

It's NOT 75!

It's 42 :-)

maximum price-
difference: **42**

buy on **day 2**

sell on **day 5**

set max-so-far = 0

for each buy-day, **b**:

for each sell-day, **s**:

compute the **profit**

if **profit** is > **max-so-far**:

remember it in a variable!

return **profit**, its b-day, and s-day

Important fine print:

[2] How could **nested loops** help us find the
best TTS strategy? (a "code sketch...")

this all seems sketch...

*To make our business plan **realistic**, however, we only allow selling after buying.



The TTS-strategy:

[0] T.T. Securities's customer pledge:
"We select the day to buy and day to sell that
will maximize your price-difference..."*

Your stock's prices: $L = [40, 80, 10, 30, 27, 52, 5, 15]$

<i>index</i>	<i>element</i>
Day	Price
0	40.0
1	80.0
2	10.0
3	30.0
4	27.0
5	52.0
6	5.0
7	15.0

[1] What is the best TTS investment strategy for **this list**, L?

[1b] Which day would you _buy_ (and at what price)?

[1c] Which day would you _sell_ (and at what price)?

[1d] What is the per-share profit in this best case? (!!!)

It's NOT 75!

It's 42 :-)

maximum price-
difference: **42**

buy on **day 2**

set max-so-far = 0

for each buy-day, **b**:

for each sell-day, **s**:

if **profit** is > **max-so-far**:

remember it in a variable!

return **profit**, its b-day, and s-day

Pass these into their own future!

[2] How could **nested loops** help us find the
best TTS strategy? (a "code sketch...")

Important fine print:

this all seems sketch...

*To make our business plan **realistic**, however, we only allow selling after buying.



hw8pr4: T. T. Securities (TTS)

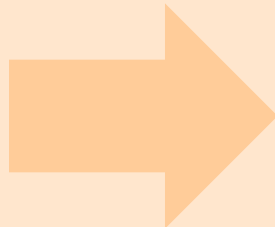
Analyzes a sequence of "stock prices"

i

	day 0	day 1	day 2	day 3	day 4	day 5	day 6	day 7									
L =	[40	,	80	,	10	,	30	,	27	,	52	,	5	,	15]

x

Implement a text *menu*:



- (0) Input a new list
 - (1) Print the current list
 - (2) Find the average price
 - (3) Find the standard deviation
 - (4) Find the min and its day
 - (5) Find the max and its day
 - (6) Your TTS investment plan
 - (9) Quit
- Enter your choice:

hw8pr4: T. T. Securities (TTS)

Analyzes a sequence of *"stock prices"*

i

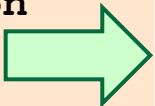
	day 0	day 1	day 2	day 3	day 4	day 5	day 6	day 7									
L =	[40	,	80	,	10	,	30	,	27	,	52	,	5	,	15]

x

Implement a (text) menu:

- (0) Input a new list
- (1) Print the current list
- (2) Find the average price
- (3) Find the standard deviation
- (4) Find the min and its day
- (5) Find the max and its day
- (6) Your TTS investment plan
- (9) Quit

Enter your choice:



Functions you'll write

All use loops...

Menu

- (0) Input a new list
- (1) Print the current list
- (2) Find the average price
- (3) Find the standard deviation
- (4) Find the min and its day
- (5) Find the max and its day
- (6) Your TTS investment plan
- (9) Quit

Enter your choice:

`def average(L)`

`def stdev(L)`

$$\sqrt{\frac{\sum_i (L[i] - L_{av})^2}{\text{len}(L)}}$$

`def minprice(L)` ➡

`def maxday(L)` ➡

also, max...



Min price

$$m = \min(L)$$

	day 0	day 1	day 2	day 3	day 4	day 5	day 6	day 7
$L =$	40	80	10	30	27	52	5	15

$m = L[0]$

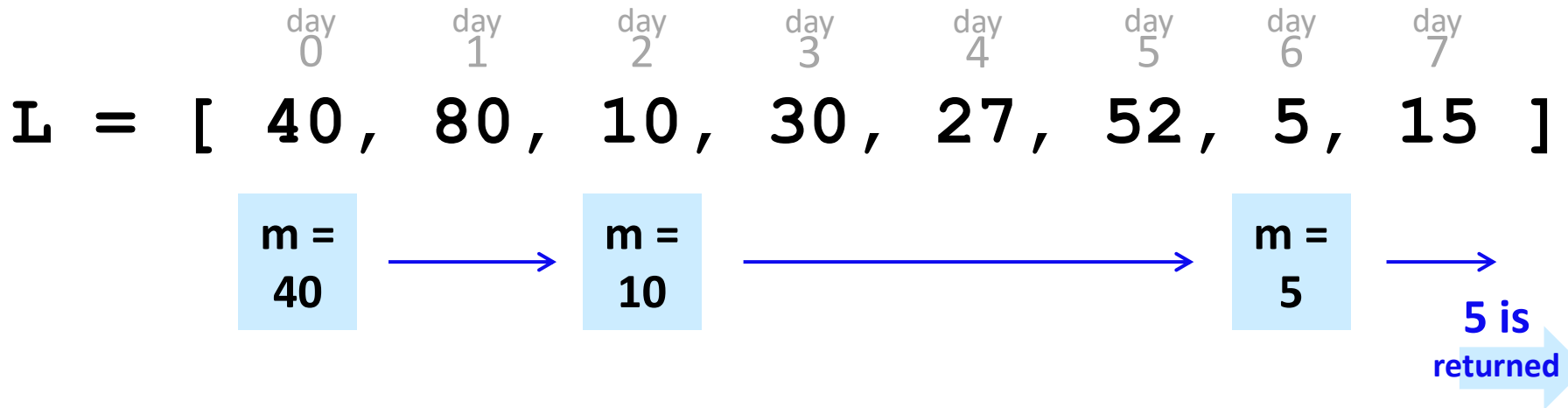
m is the
"min so far"

for x in L :
 if $x < m$
 $m = x$

What's the *idea* for finding the smallest (minimum) price?

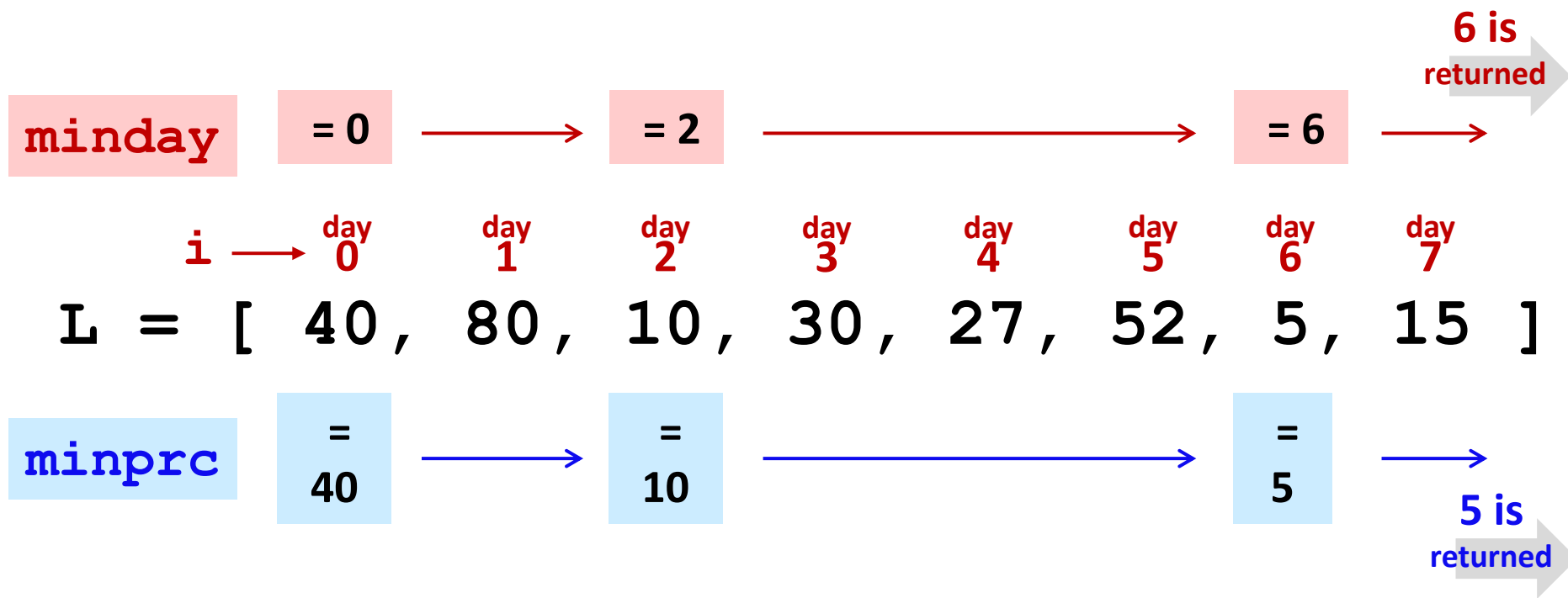
track the value of the *minimum so far* as you loop over L

Min price vs. min *day*



```
def minprice( L ):  
    m = L[0]  
    for x in L:  
        if x < m:  
            m = x  
    return m
```

What about tracking BOTH
the *day* of the minimum
price and that min **price**?



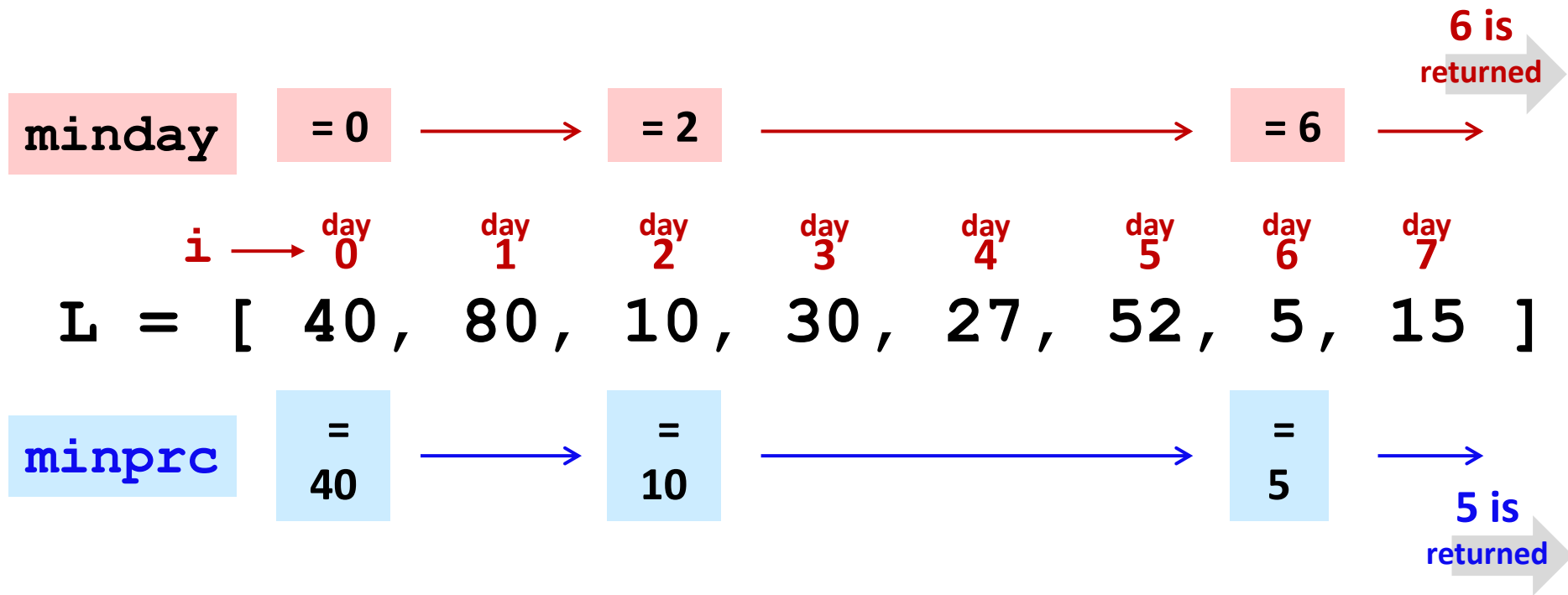
```
def min_prc_day( L ) :  
    minprc = L[0]  
    minday = 0  
    for i in range( len(L) ) :  
        if L[i] < minprc :  
            minprc = L[i]  
            minday = i  
    return minprc, minday
```

track price *and* day

loop over locs (i)

check one and
update both
(as needed)

return *both*!



```
def min_prc_day( L ):
```

```
    minprc = L[0]
```

```
    minday = 0
```

```
    for i in range(len(L)):
```

```
        if L[i] < minprc:
```

```
            minprc = L[i]
```

```
            minday = i
```

```
    return minprc, minday
```

track price *and* day

loop over locs (i)

check one and
update both
(as needed)

return *both*!

T. T. Securities

*"Taking the broke
out of brokerage."*

Software side ...

- (0) Input a new list
- (1) Print the current list
- (2) Find the average price
- (3) Find the standard deviation
- (4) Find the min and its day
- (5) Find the max and its day
- (6) Your TTS investment plan
- (9) Quit

Enter your choice:



*Hardware
side...*



Investment analysis for the 21st century ... *and beyond*

The TTS advantage!

What is the best
TTS investment
strategy here?

Your stock's prices: $L = [40, 80, 10, 30, 27, 52, 5, 15]$

Day	Price
-----	-------

0	40.0
---	------

1	80.0
---	------

2	10.0
---	------

3	30.0
---	------

4	27.0
---	------

5	52.0
---	------

6	5.0
---	-----

7	15.0
---	------

- (0) Input a new list
- (1) Print the current list
- (2) Find the average price
- (3) Find the standard deviation
- (4) Find the min and its day
- (5) Find the max and its day
- (6) Your TTS investment plan
- (9) Quit

Enter your choice:

Important fine print:

To make our business plan realistic, however, we only allow selling after buying.

The TTS advantage!

What is the best
TTS investment
strategy here?

Your stock's prices: $L = [40, 80, 10, 30, 27, 52, 5, 15]$

Day	Price
-----	-------

0	40.0
---	------

1	80.0
---	------

2	10.0
---	------

3	30.0
---	------

4	27.0
---	------

5	52.0
---	------

6	5.0
---	-----

7	15.0
---	------

set max-so-far = 0

for each buy-day, **b**:

for each sell-day, **s**:

compute the *profit*

if *profit* is > max-so-far:

remember it in a variable!

return *profit*, its b-day, and s-day

Important fine print:

To make our business plan **realistic**, however, we only allow selling after buying.

The TTS advantage!

What is the best
TTS investment
strategy here?

Your stock's prices: $L = [40, 80, 10, 30, 27, 52, 5, 15]$

Day	Price
-----	-------

0	40.0
---	------

1	80.0
---	------

2	10.0
---	------

3	30.0
---	------

4	27.0
---	------

5	52.0
---	------

6	5.0
---	-----

7	15.0
---	------

set max-so-far = 0

for each buy-day, **b**:

for each sell-day, **s**:

compute the

"All-Pairs"
Algorithm!!!

Important fine print:

To make our business plan realistic, however, we only allow

"N-body" problems (***)
shortest paths...
closest-pair

The TTS advantage!

What is the best TTS investment strategy here?

Your stock's price: 1 = [40.80. 10. 30. 27. 52. 5. 15]

Article Talk Read Edit View history Search Wikipedia

Three-body problem

From Wikipedia, the free encyclopedia

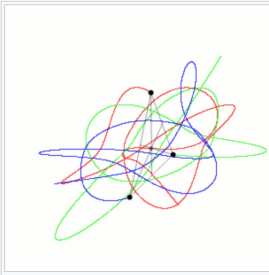
This article is about the physics and classical mechanics theory. For the Chinese science fiction novel by Liu Cixin, see [The Three-Body Problem \(novel\)](#). For other uses, see [Three-body problem \(disambiguation\)](#).

In physics and classical mechanics, the **three-body problem** is the problem of taking the initial positions and velocities (or momenta) of three point masses and solving for their subsequent motion according to Newton's laws of motion and Newton's law of universal gravitation.^[1] The three-body problem is a special case of the *n*-body problem. Unlike two-body problems, no general closed-form solution exists,^[1] as the resulting dynamical system is chaotic for most initial conditions, and numerical methods are generally required.

Historically, the first specific three-body problem to receive extended study was the one involving the Moon, the Earth, and the Sun.^[2] In an extended modern sense, a three-body problem is any problem in classical mechanics or quantum mechanics that models the motion of three particles.

Contents [hide]

- 1 Mathematical description
 - 1.1 Restricted three-body problem
- 2 Solutions
 - 2.1 General solution
 - 2.2 Special-case solutions
 - 2.3 Numerical approaches
- 3 History
- 4 Other problems involving three bodies
- 5 n-body problem
- 6 In popular culture



Approximate trajectories of three identical bodies located at the vertices of a scalene triangle and having zero initial velocities. It is seen that the center of mass, in accordance with the law of conservation of momentum, remains in place.

"N-body" problems (***)
for N == 3 (or more!)

ay-day, b:

sell-day, s:

ite the

"All-Pairs"
Algorithms!!!

"N-body" problems (***)
shortest paths...
closest-pair

Important fine print:

To make our business plan **realistic**, however, we only allow

Write **mindiff** to return the **smallest** abs. diff.
between any two elements from **L**.

mindiff([42,3,100,-9,7])

mindiff([42,3,100,-9,7])

 **Hint:** This uses nested loops!

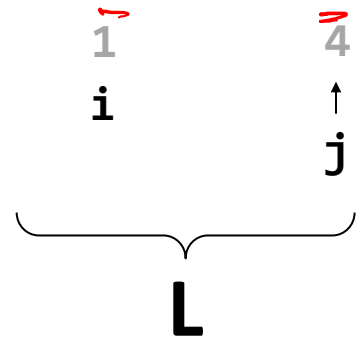
Write **mindiff** to return the **smallest** abs. diff.
between any two elements from **L**.

mindiff([42,3,100,-9,7])

4

mindiff([42,3,100,-9,7])

4



Hint: This uses nested loops!

```
for i in range(4):  
    for j in range(4):
```

Track the value of the
minimum so far as you
loop over **L** twice...

Write **mindiff** to return the **smallest** abs. diff.
between any two elements from **L**.

```
def mindiff( L ):
```

```
    mindiff = abs(L[1]-L[0])
```

```
    for i in range(len(L)):
```

```
        for j in range(i+1, len(L)):
```

```
            diff = abs(L[i]-L[j])
```

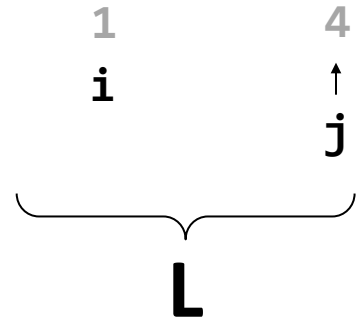
```
            if diff < mindiff:
```

```
                mindiff = diff
```

```
    return mindiff
```

mindiff([42,3,100,-9,7])

4



Hint: This uses nested loops!

```
for i in range(4):
```

```
    for j in range(4):
```

Track the value of the
minimum so far as you
loop over **L** twice...

Write **mindiff** to return the **smallest** abs. diff. between any two elements from **L**.

```
def mindiff( L ):
```

```
    mdiff = abs(L[1]-L[0])
```

```
    for i in range(len(L)):
```

```
        for j in range(i+1, len(L)):
```

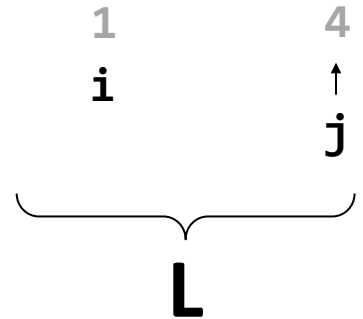
```
            if abs(L[j]-L[i]) < mdiff:
```

```
                mdiff = abs(L[j]-L[i])
```

```
    return mdiff
```

mindiff([42,3,100,-9,7])

4



Hint: This uses nested loops!

```
for i in range(4):  
    for j in range(4):
```

Track the value of the *minimum so far* as you loop over **L** twice...

The TTS advantage!

What is the best
TTS investment
strategy here?

You

very similar to mindiff

[40, 80, 10, 30, 27, 52, 5, 15]

Day	Price
-----	-------

0	40.0
---	------

1	80.0
---	------

2	10.0
---	------

3	30.0
---	------

4	27.0
---	------

5	52.0
---	------

6	5.0
---	-----

7	15.0
---	------

set max-so-far = 0

for each buy-day, **b**:

for each sell-day, **s**:

compute the *profit*

if *profit* is > max-so-far:

remember it in a variable!

return *profit*, its b-day, and s-day

Important fine print:

To make our business plan **realistic**, however, we only allow selling after buying.

The TTS advantage!

What is the best
TTS investment
strategy here?

Your stock's prices: $L = [40, 80, 10, 30, 27, 52, 5, 15]$

Day	Price
0	40.0
1	80.0
2	10.0
3	30.0
4	27.0
5	52.0
6	5.0
7	15.0

set max-so-far = 0

See "future you" in lab!

when sell-day, s:

compute the *profit*

if *profit* is > *max-so-far*:

remember it in a variable!

return *profit*, its b-day, and s-day

... hw8 is ready to help!

Important fine print:

To make our business plan realistic, however, we only allow selling after buying.