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? ')
cm = meters * 100
print("That's", cm, 'cm.')
```

What will Python think?

User input...

```
meters = input('How many m? ')
cm = meters * 100
print("That's", cm, 'cm.')
```

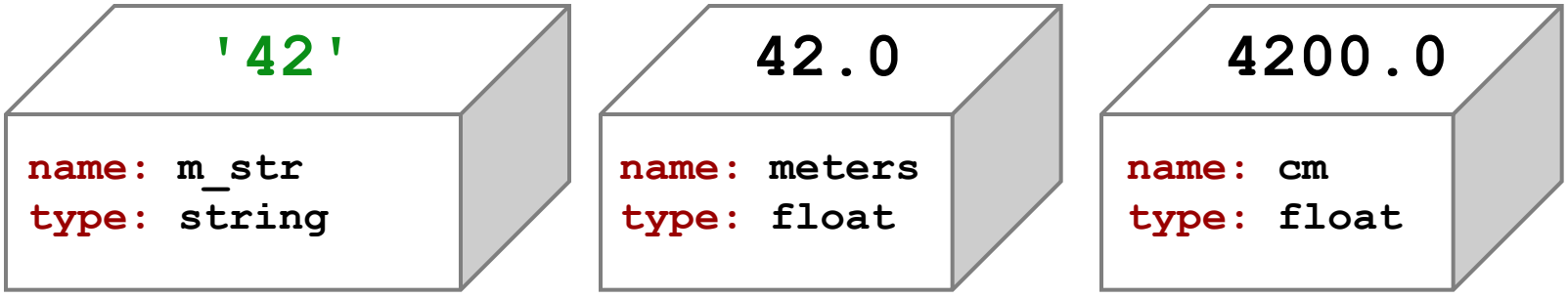
input ALWAYS returns a **string** – *no matter what's typed!*

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 )
except:
   print("What? Didn't compute!")
   print("Setting meters = 42")
   meters = 42.0

cm = meters * 100
print('That\'s', cm, 'cm.')
```

crash-able

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

Fix #2

User-errors are called ***exceptions.***

This is ***exception handling.***


```
try:
   meters = float( m_str )
except:
   print("What? Didn't compute!")
   print("Setting meters = 42")
   meters = 42.0

cm = meters * 100
print('That\'s', cm, 'cm.')
```

crash-able

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

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? ')


        try:
            uc = int(uc)      # was it an int?
        except:
            continue          # back to the top!
```

Calls a helper function

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

        elif uc == 0:
            continue

        elif uc == 1:
            ... input ... eval ...

        elif uc == 2:
```

(9) Quit

(0) Continue

(1) Get new list

(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

        elif uc == 0:
            continue

        elif uc == 1:
            ... input ... eval ...

        elif uc == 2:
```

(9) Quit

break breaks *out of the loop...*

(0) Continue

continue jumps *back to the top...*

(1) Get new list

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

(2) Analyze !

other functions as needed...

... and so on ...

Big-picture view!

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

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

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

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

"Loops back again"

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

```
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-interac
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          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!")
```

main function

secret_value

while True:

input (option from menu)

input (new list)

[5] Where is predict defined?

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

'42'

[6b] how about reaching line 239?

1 +

[secret_value]

Full-program menu-interaction example

Try it!

[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*?

[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!

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

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

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

```
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-interac
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          #
```

main function

secret_value

while True:

input (option from menu)

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

[6b] how about reaching line 239?

```
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!")
```

input (new list)

[5] Where is predict defined?

Full-program menu-interaction example

Try it!

[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*?

[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!

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

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

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

```
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-interac
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
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!")
```

main function

secret_value

while True:

input (option from menu)

input (new list)

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

[6b] how about reaching line 239?

[5] Where is predict defined?

Full-program menu-interaction example

Try it!

[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*?

Big-picture view!

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

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

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

[3] What does input **3** print that **0** *does not*? **line 262**

[4] What line of code runs after this **break** ? and **continue** ? **264** **206/207**

```
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-interac
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          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
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!")
```

main function

secret_value

while True:

input (option from menu)

input (new list)

"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

[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*?

Sols...

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

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

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

[3] What does input **3** print that **0** *does not*? **line 262**

Big-picture view!

[4] What line of code runs after this **break** ? and **continue** ? **264** **206/207**

```
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-interac
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          if uc == 4:  # 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
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!")
```

"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)

input (new list)

[5] Where is predict defined? **188**

input [0, 1, 2, secret_value]

[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*?

Full-program menu-interaction example

Sols...

Loops

```
def fac( N ):
    result = 1
    for x
        re
```

sequences!

Basic design strategies

Is one more *reasonable* than the other?

Recursion

```
def fac( N ):
    if N == 1:
```

self-similarity

Loops

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

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

Basic design strategies

Is one more *reasonable* than the other?

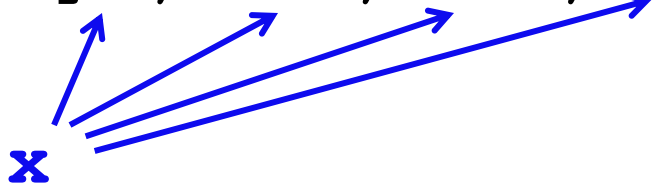
Strategy: Look for **self-similarity** + use it… .

Recursion

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

`for`: *two "loop patterns"*

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



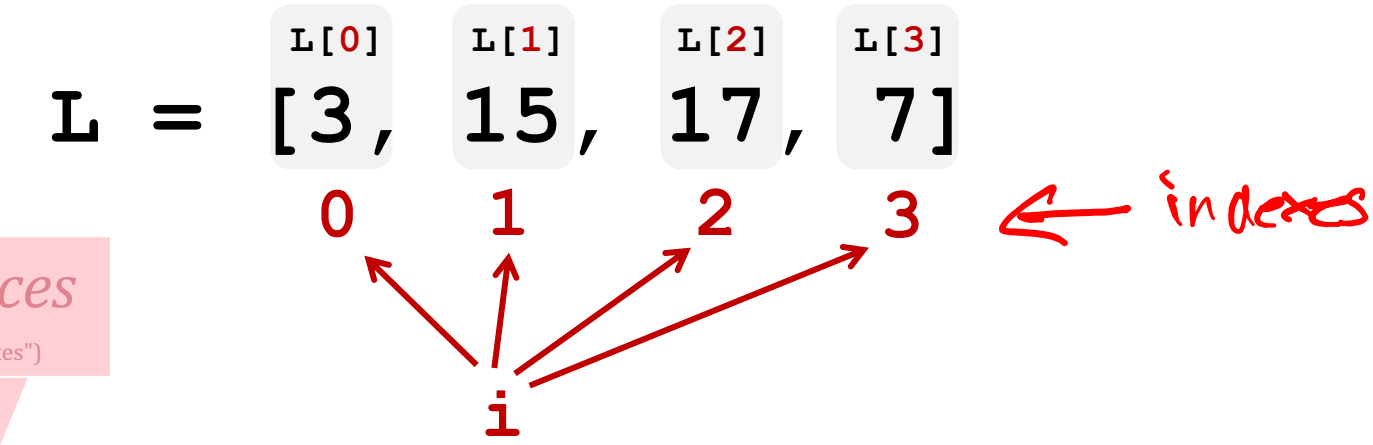
"deceptively easy"

elements

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

element-based loops

`for`: *two "loop patterns"*




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

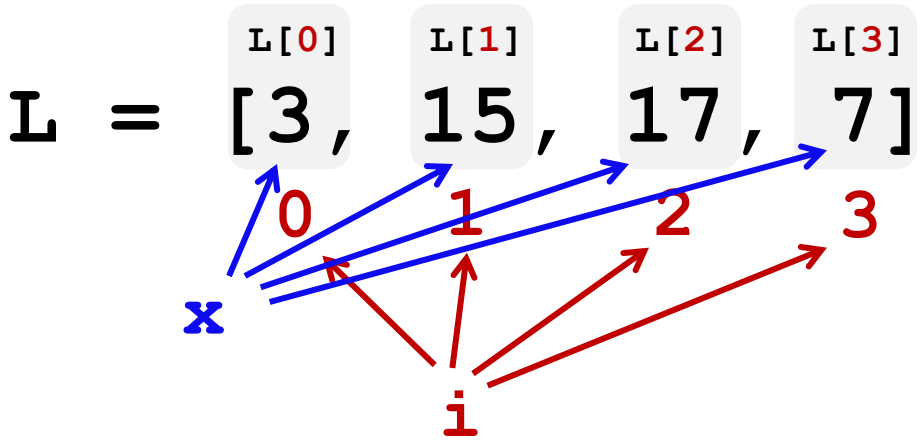
index-based loops
— access data indirectly,
(by its *index*)

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

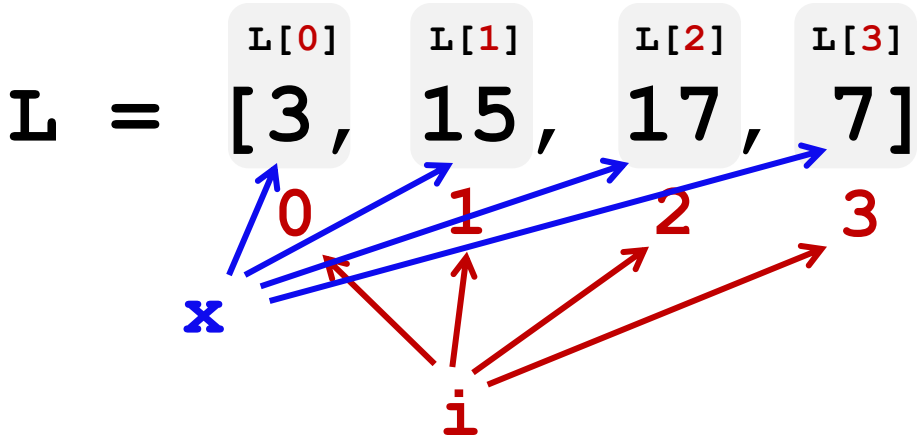
Elements vs Indexes

Indices

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

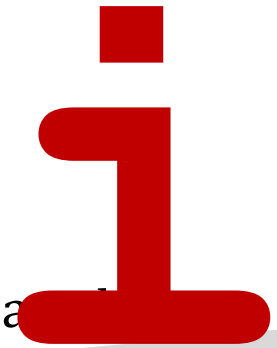
element-based loops
— access data directly

`for`: *two variables*




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

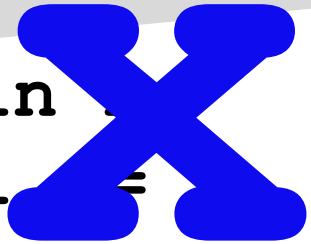
index-ba



Elements vs Indexes

Indices

```
for x in
    total
```



element-based loops

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

hw8pr4: T. T. Securities (TTS)

hw8pr4: T. T. Securities (TTS)

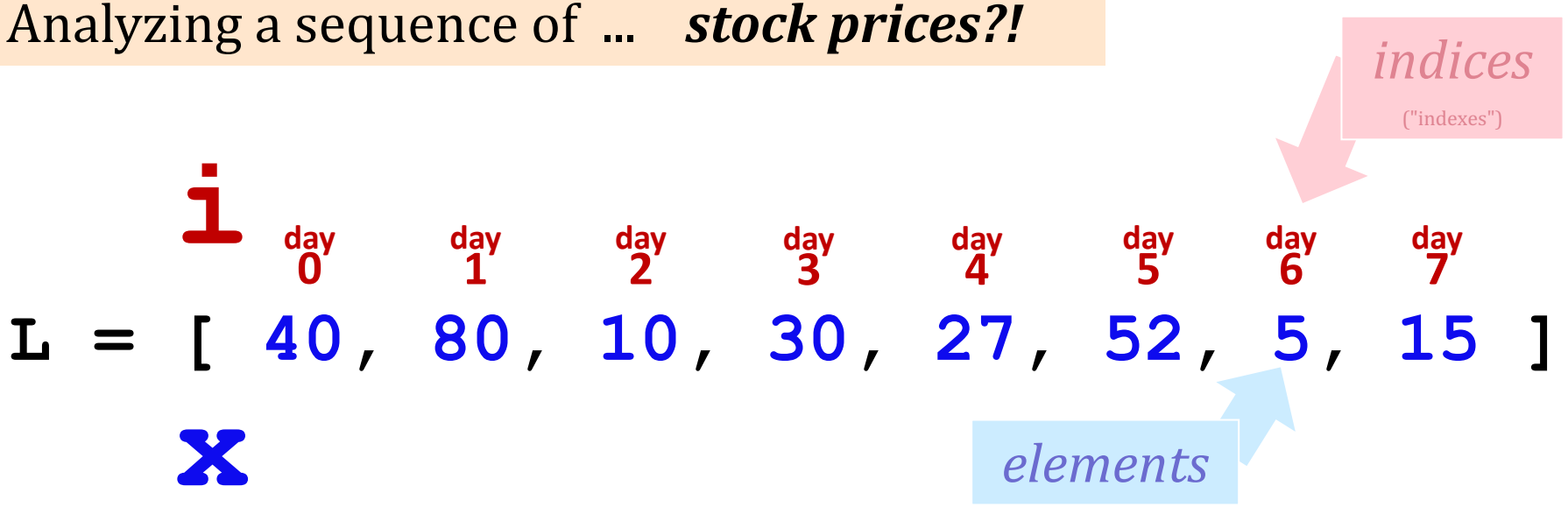
Analyzing a sequence of ... ***anything!***

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?!***



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  day  day  day  day  day  day  day
       0    1    2    3    4    5    6    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:
```

`webbrowser.open_new_tab(url)`

let's see...

One motivation for TT securities…

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]

index	*element*
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?

It's **NOT** 75!

[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

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...")*

this *all* seems sketch...

Important fine print:

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

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]

index	*element*
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

maximum price-difference: **42**

buy on **day 2**

sell on ***day 5***

[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 :-)

```
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
```

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

this *all* seems sketch...

Important fine print:

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

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]

index	*element*
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**

Pass these into their own future!

```
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
```

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

this *all* seems sketch...

Important fine print:

*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  day  day  day  day  day  day  day
        0    1    2    3    4    5    6    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:
```

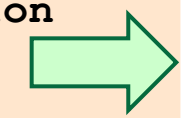
hw8pr4: T. T. Securities (TTS)

Analyzes a sequence of *"stock prices"*

```
        i
        day  day  day  day  day  day  day  day
        0    1    2    3    4    5    6    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 )
```

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

```
def minprice( L )
def maxday( L )
```

also, max...

```
webbrowser.open_new_tab(url)
```

Min price

m = min(L)

Just call **min** ?

	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]

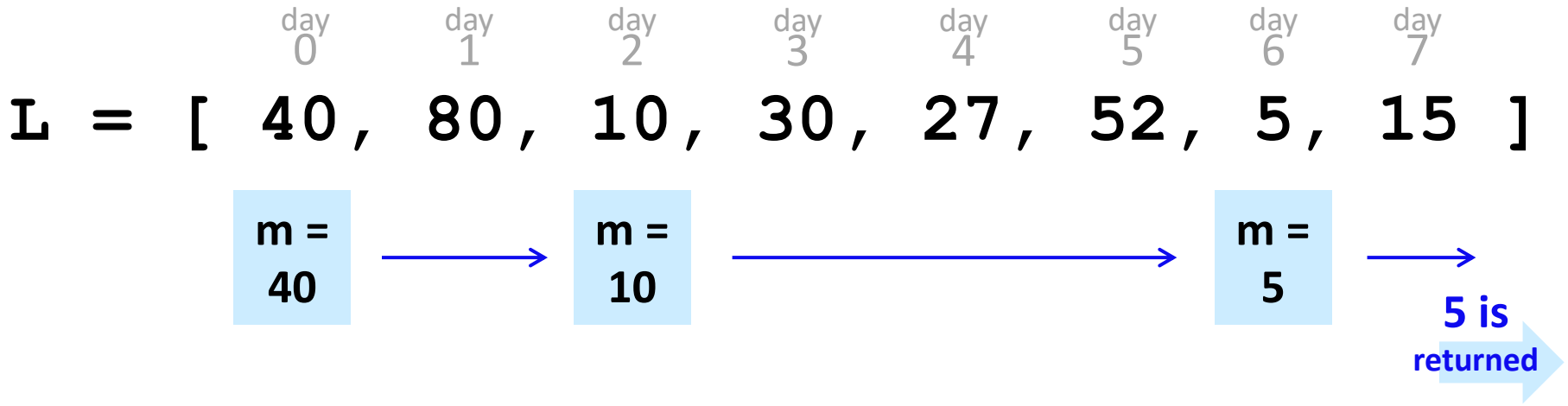
m is the "min so far"

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

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

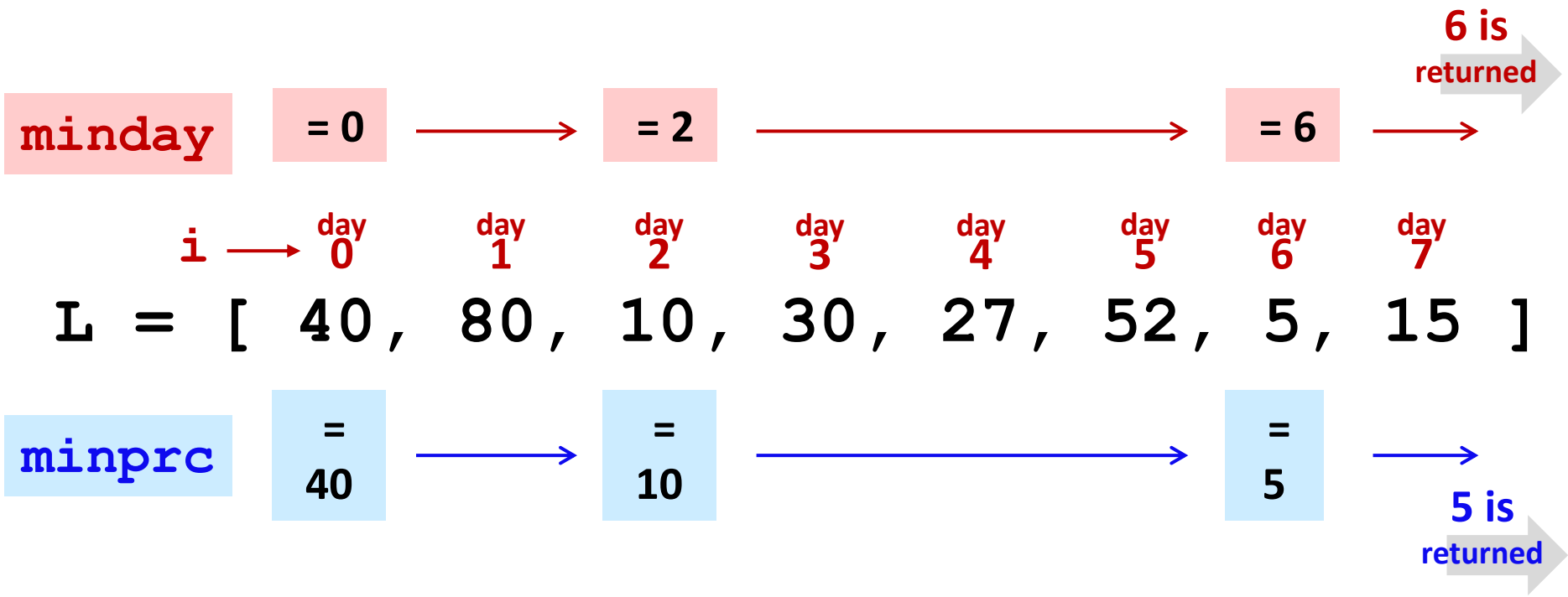
track the value of the ***minimum so far*** as you loop over L

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 <u>BOTH</u> the ***day*** of the minimum price ***<u>and</u>*** that min **price**?



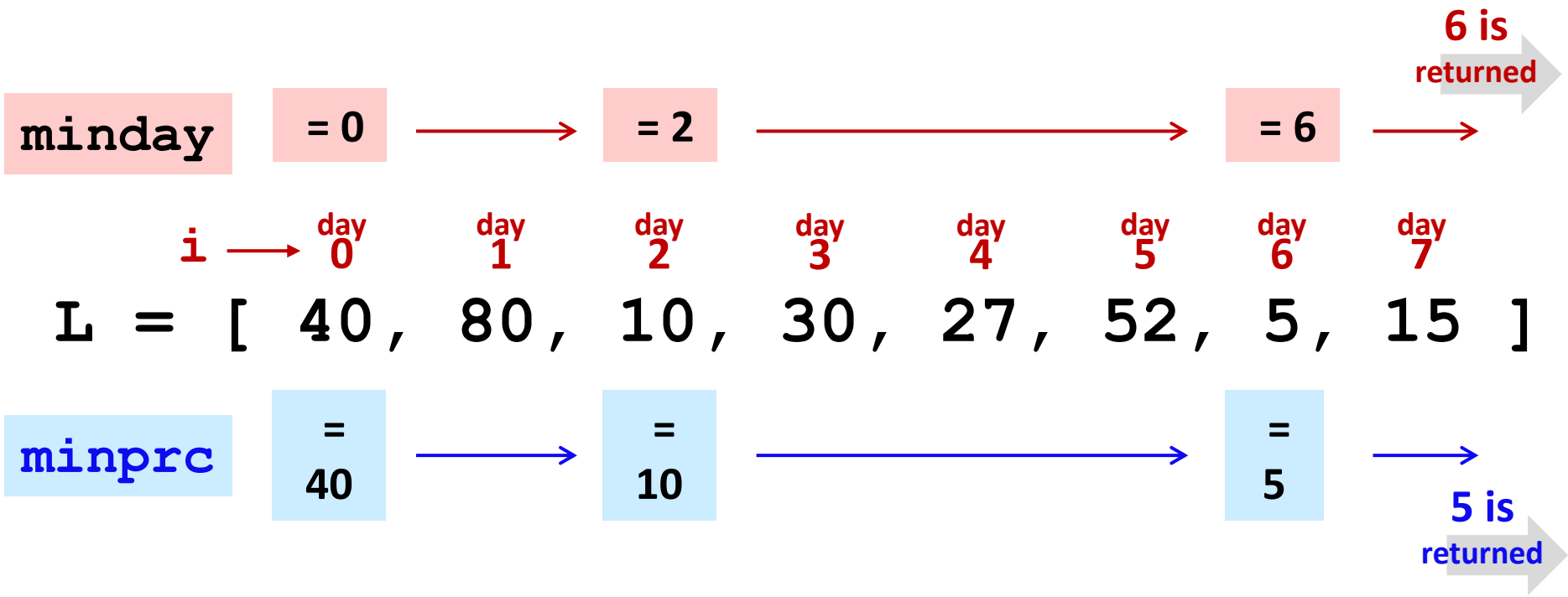

```
def min_prc_day( L ):
    minprc = L[0]
    minday = 0
    for i in range(len(L)):
        if L[i] < minprc:
            minprc = L[i]
            minday = i
    return minprc, minday
```

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."

```
(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:
```

Software side ...

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!!!

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

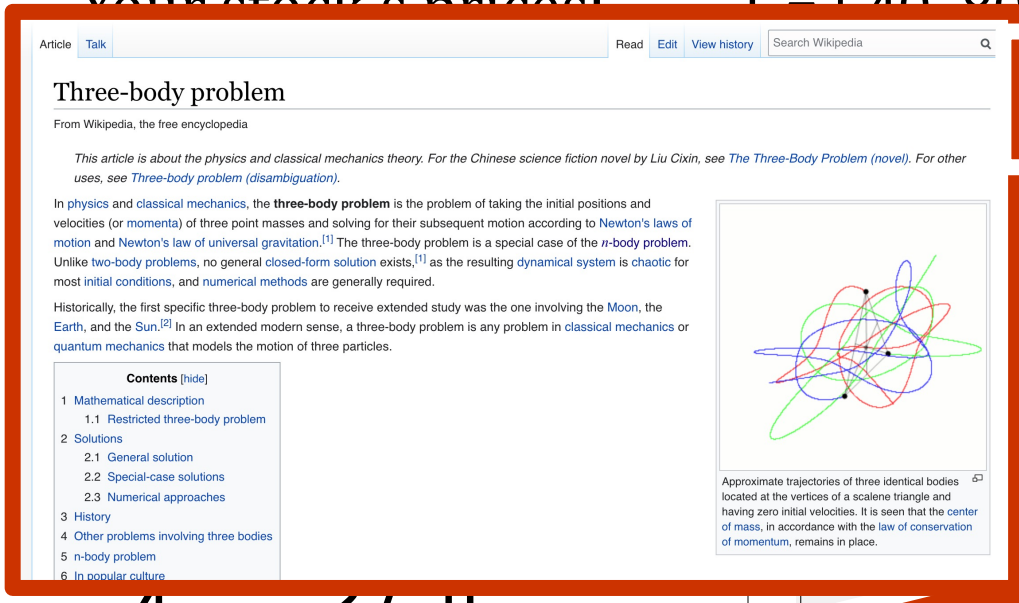
Important fine print:

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

The TTS advantage!

What is the best TTS investment strategy here?

Your stock's prices: L = [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!)

y-day, **b**:

sell-day, **s**:

te the

4	27.0
5	52.0
6	5.0
7	15.0

"All-Pairs" Algorithms!!!

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

Important fine print:

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

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

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

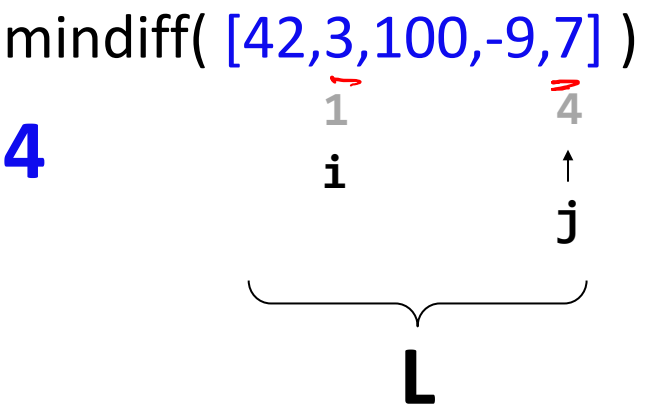
mindiff([42,3,100,-9,7])

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



Hint: This uses nested loops!

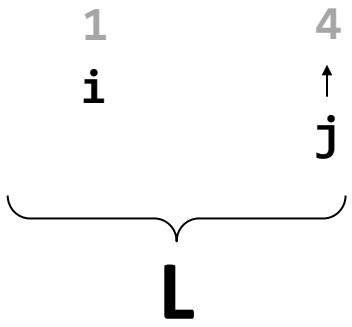
```
for i in range(4):
    for j in range(4):
```

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.**

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

4




```
def mindiff( L ):

    mdiff = 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 < mdiff
                mdiff = diff

    return mdiff
```

Hint: This uses nested loops!

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

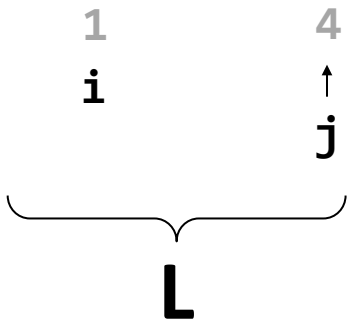
Track the value of the *minimum so far* as you loop over **L twice**...

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[illegible], 10, 30, 27, 52, 5, 15]

very similar to mindiff

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

...ch 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

See "*future you*" in lab!

... hw8 is ready to help!

Important fine print:

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