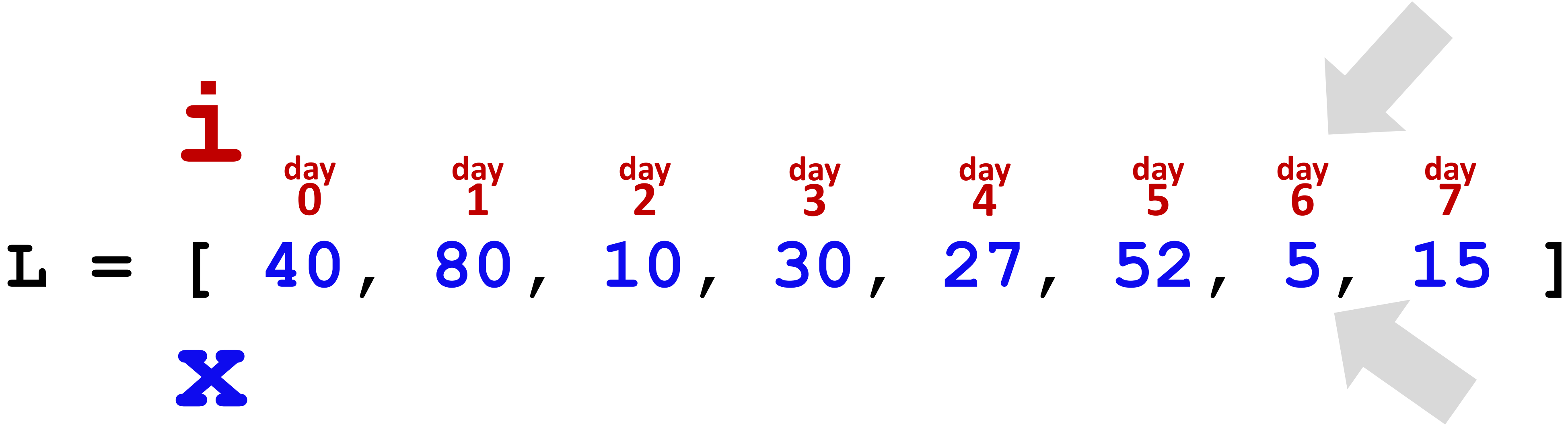
hw8pr4: T. T. Securities (TTS)

Analyzes a sequence of *"stock prices"*



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

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)`

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

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:
```

User input...

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

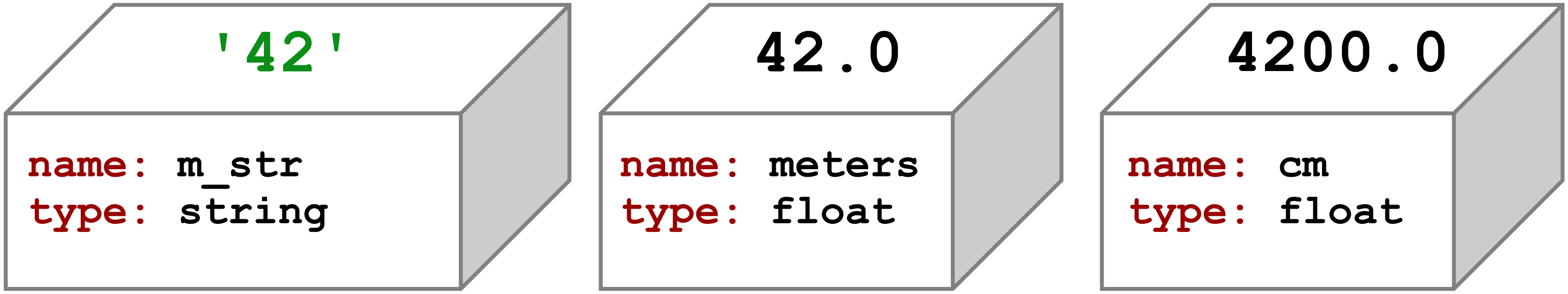
`input` **ALWAYS** returns a string – *no matter what's typed!*

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 is', cm, 'cm.')
```



'42'
name: m_str
type: string

42.0
name: meters
type: float

4200.0
name: cm
type: float

... 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-

These 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

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

What could **REALLY** go wrong here?

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:
```

(9) Quit

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

(0) Continue

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

(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` jumps *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 ...

(A) Which code below handles an input of 5 ?   of 7 ?

Try it!

Full program example of user-interactions

Name(s) ______________

(B) What does choice 3 print that 0 *does not*?

```
# example looping program

def menu():
    """ a function that simply prints the menu """
    print()
    print("(0) Continue!")
    print("(1) Enter a new list")
    print("(2) Predict the next element")
    print("(9) Break! (quit)")
    print()

def main():
    """ the main us[illegible]oop """
    print()
    print("+++++++++++++++++++++++++")
    print("Welcome to the PREDICTOR!")
    print("+++++++++++++++++++++++++")
    print()

    secret_value = 4.2

    L = [30,10,20]   # an initial list

    while True:      # the user-interaction loop
        print("\n\nThe list is", L)
        menu()
        uc = input( "Choose an option: " )

        # "clean and check" the user's input
        #
        try:
            uc = int(uc)    # make into an int!
        except:
            print("I didn't understand your input! Continuing...")
            continue

        # run the appropriate menu option
        #
        if uc == 9:     # we want to quit
            break       # leaves the while loop altogether

        elif uc == 0:   # we want to continue...
            continue    # goes back to the top of the while loop
```

main function

secret_value

while True:

(C) What line of code runs after this `break` ?

```
        elif uc == 1:   # we want to enter a new list
            newL = input("Enter a new list: ")     # enter _something_

            # "clean and check" the user's input
            #
            try:
                newL = eval(newL)    # eval runs Python's interpreter! Note: Danger
                if type(newL) != type([]):
                    print("That didn't seem like a list. Not changing L.")
                else:
                    L = newL  # here, things were OK, so let's set our list, L
            except:
                print("I didn't understand your input. Not changing L.")

        elif uc == 2:          # predict and add the next element
            n = predict(L)     # get the next element from the predict function
            print("The next element is", n)
            print("Adding it to your list...")
            L = L + [n]        # and add it to the list

        elif uc == 3:   # unannounced menu option!
            pass        # this is the "nop" (do-nothing) statement in Python

        elif uc == 4:   # unannounced menu option (slightly more interesting...)
            m = find_min(L)
            print("The minimum value in L is", m)

        elif uc == 5:   # another unannounced menu option (even more interesting...
            minval, minloc = find_min_loc(L)
            print("The minimum value in L is", minval, "at day #", minloc)

        else:           # if the input uc was anything else
            print(uc, " ?      That's not on the menu!")

        print("Running again...\n")

    print("\nI predict...  \n\n          ... that you'll be back!")
```

input

(D) What could you input for `newL` that would print this?

(E) What could you type for `newL` that would print this?

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

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 )
```

$$\sqrt{\frac{\sum_{i} (\mathtt{L[i]} - \mathtt{L}_{\mathtt{av}})^{2}}{\mathtt{len(L)}}}$$

```
def minday( L )

def maxday( L )
```

`webbrowser.open_new_tab(url)`

Min price

Just call **`min`** ?

```
      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 ]
```

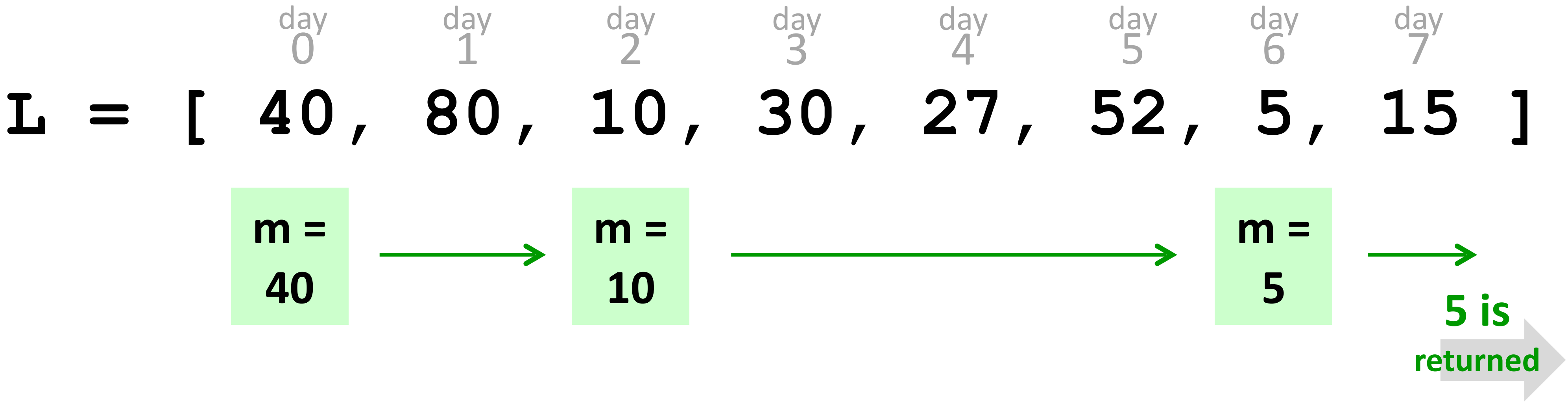
m =

m is the
"min so far"

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***



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 = 40

m = 10

m = 5

5 is returned

```
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?

Finish this code to return ***both*** the minprice and the minday of **L**!

Expand on the minprice example...

L

min_prc_day([9, 8, 5, 7, 42])

0 1 2 3 4

5, 2

```
def min_prc_day( L ):
    minprc = L[0]
    minday = 0

    for i in range(len(L)):
        if ______________________



    return minprc, minday
```

Write **mindiff** to return the **smallest** absolute difference between any two elements from **L.**

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

    for 
      for 
        if ______________________



    return mdiff
```

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

L

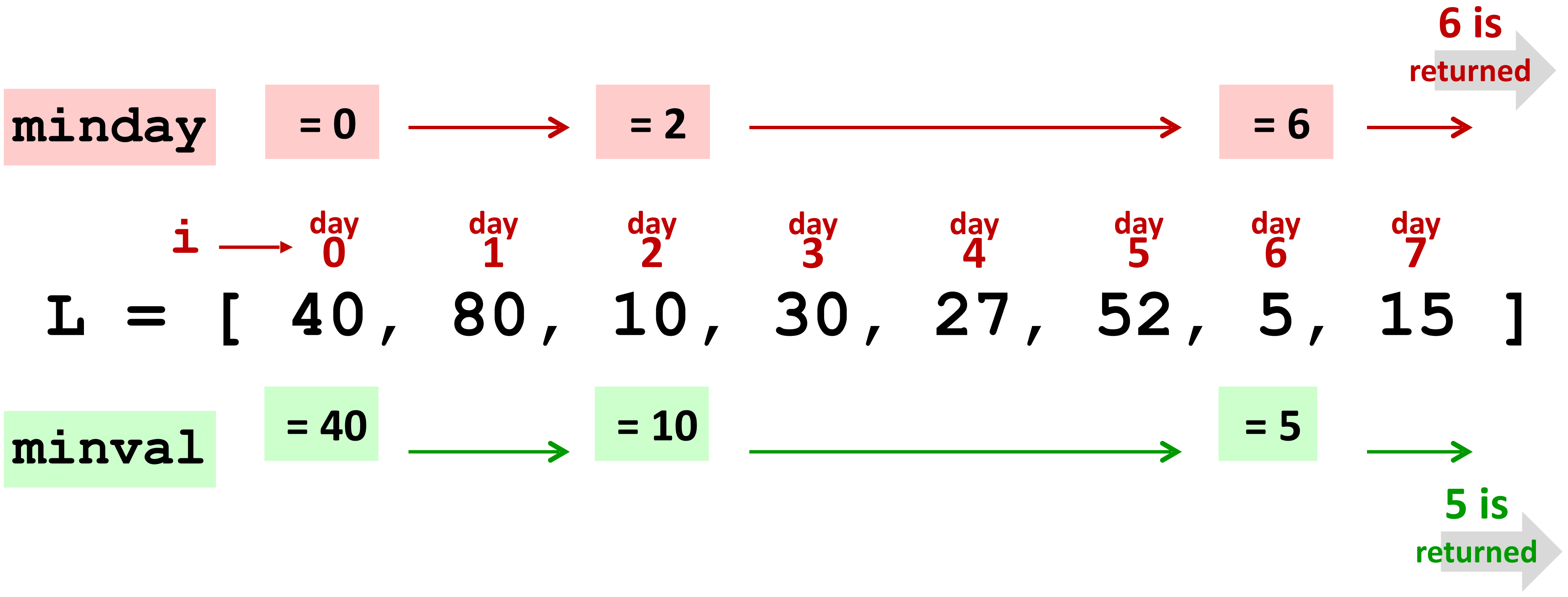
4

Try it!

Only consider **abs** differences.

L will be a list of 2 or more #s.

Hint: Use a nested loop!



minday = 0 → = 2 → = 6 → 6 is returned

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]

minval = 40 → = 10 → = 5 → 5 is returned

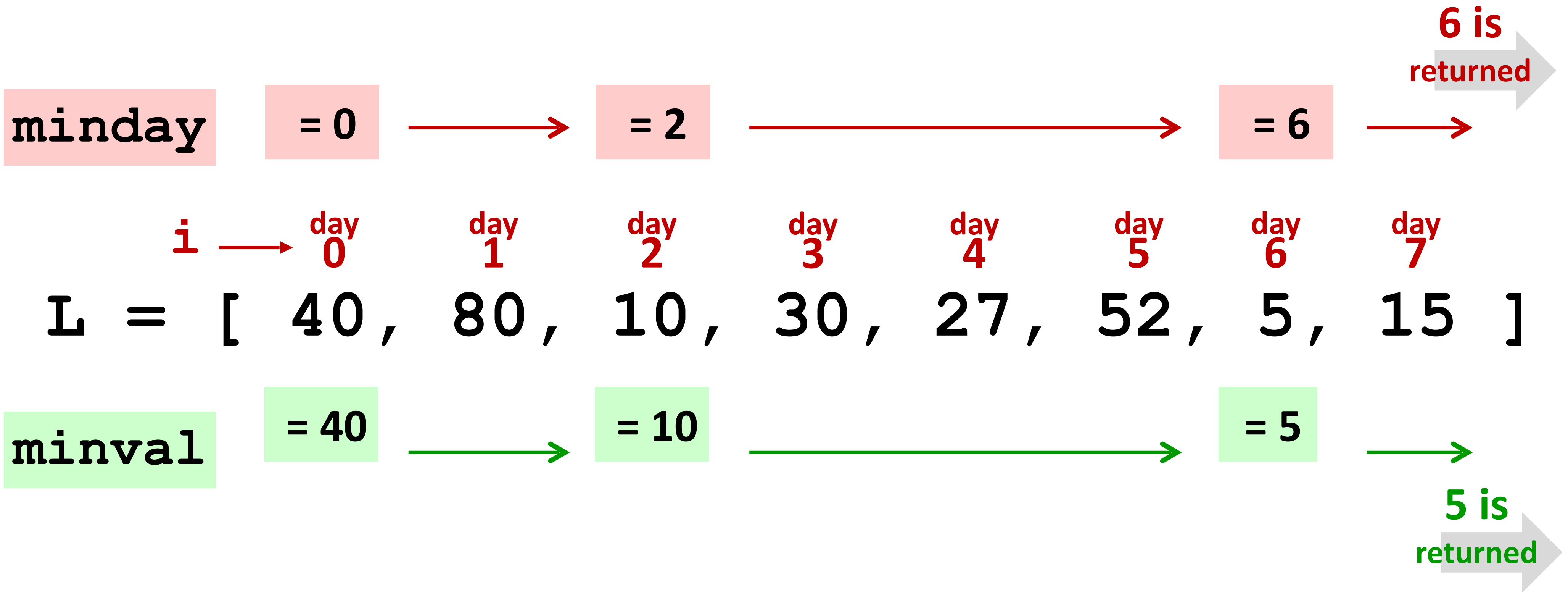
```
def min_prc_day( L ):
  minprc = L[0]
  minday = 0
  for i in range(len(L)):
    if


  return minprc, minday
```

track *both* price and day

loop over

update both
(as needed)

return *both*!



minday = 0 = 2 = 6

6 is returned

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 ]
```

minval = 40 = 10 = 5

5 is returned

```
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 ***both*** price and day

loop over

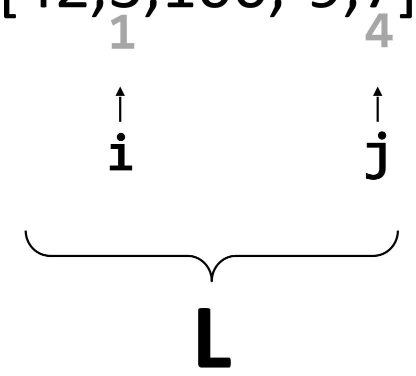
update both
(as needed)

return *both*!

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(     ,len(L)):

            if

    return mdiff
```

Hint: Use nested loops:

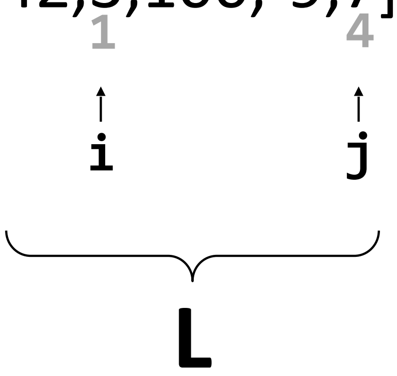
```
for i in range(4):
   for j in range(4):
```

Track the value of the *minimum so far* as you <u>loop over **L twice**</u>...

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)):

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

    return mdiff
```

Hint: Use 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**...

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

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.