CS 5: *Putting loops to work...*

[-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!

Homework 8: due Mon., 10/31 by midnight

"Office" hrs. Fri! + lots of tutoring, LAC & ...

Midterm 11/3; review on the CS5 homepage

quizzes!

Final Exam: choice of 12/16 or 17 @ 7pm

Pop tarts > candy

Official CS5 snack comparison

LIMITED EDITION
Kellogg's
FAMILY REWARDS
KelloggsFamilyRewards.com
Printed Fun
pop-tarts
toaster pastries
Frosted
Sugar Cookie
Naturally & Artificially Flavored
Good Source of 8 Vitamins & Minerals
NET WT 21.2 OZ (1 LB 5.2 OZ) (600g)
12 TOASTER PASTRIES

pop-tart recursion!

Cappuccino Pop-Tarts!

The Facts:

This doesn't exist, yet.

The Taste:

We expect it to taste real bad, but look real cool on the box. Marketed as a NEW energy food.

But this ***should*** exist!

Next Thursday will be the CS 5 in-class midterm

***Un*-warnings:**

worries? concerns? See me…

five problems, written

worth 1 hw assignment

score worries? *Extra* extra-credit in hw9 and beyond

Suggestions:

go over in-class exercises and hwk problems

create a page of notes, 2-sided is OK

consider small *variations* of the problems –
and how they would change the solutions…

only 5 minutes? Try list comprehensions & **`LoL`**!

all quizzes so far?
they're posted!

Mid-term feedback...

Don't put your name

I would love to know any thoughts you have about CS5 thus far in the term.
In particular, how you feel about the time and effort CS5 requires...

On average, how much time per week do you spend on CS5 ***outside class + lab***?

____

How does CS5's workload compare to other classes you're taking this term?

much lighter | about the same | much heavier

How would you judge the ***pace*** of CS5?

much too slow | about right | much too fast

Circle your year: First-year Sophomore Junior Senior Other

Something you'd ***keep*** about CS5 ...?

Something you'd ***change about***

Other thoughts optional, but 142% welcome:

Later today

CS 5: *Putting loops to work...*

[-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!

Homework 8: due Mon., 10/31 by midnight

"Office" hrs. Fri! + lots of tutoring, LAC & ...

Midterm 11/3; review on the CS5 homepage

quizzes!

Final Exam: choice of 12/16 or 17 @ 7pm

The ***read it and weep*** sequence

```
       1
      11
      21
    1211
  111221
  312211
13112221

     ...
```

`str` vs. **`int`**

When does the first **4** appear?

How fast do these terms grow?

Extra extra credit: in wk9!

Growth determined empirically…

```
       1
      11
      21
    1211
  111221
  312211
13112221

     ...
```

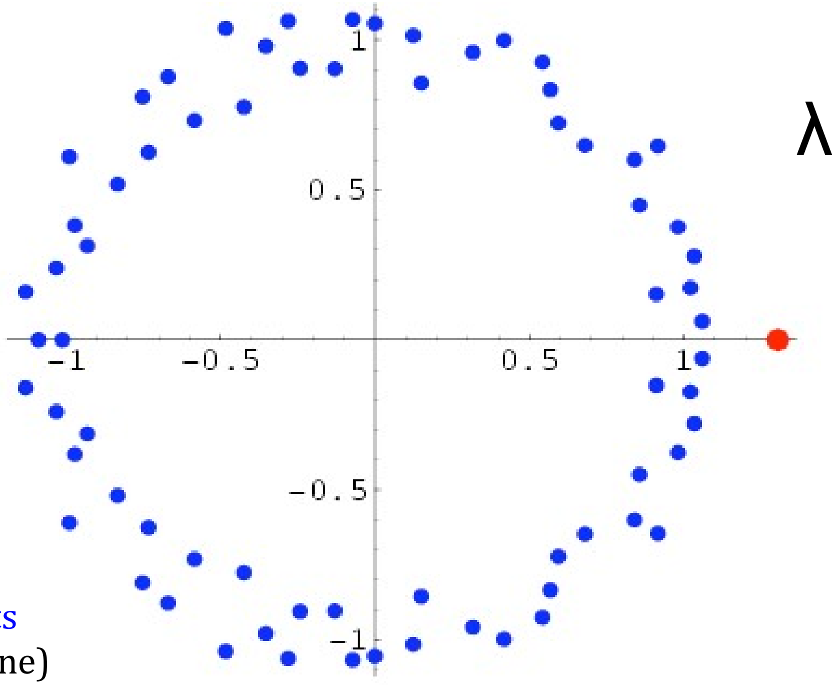
In the limit, the length of the Nth term of the read-it-and-weep sequence is

$$(1.303577...)^{N}$$

exponential growth

this base was found computationally by taking repeated ratios of term lengths…

Growth determined analytically…



the 71 roots
(complex plane)

λ = 1.30357726034296…

"Conway's Constant" has an ***analytic*** definition!

It is the largest real root of this ***71st-degree*** polynomial !!

$$x^{18}(x+1)(x-1)^2\big(x^{71}-x^{69}-2x^{68}-x^{67}+2x^{66}+2x^{65}+x^{64}-x^{63}-x^{62}-x^{61}-x^{60}-x^{59}+2x^{58}+5x^{57}+3x^{56}-2x^{55}-10x^{54}-3x^{53}-2x^{52}+6x^{51}+6x^{50}+x^{49}+9x^{48}-3x^{47}-7x^{46}-8x^{45}-8x^{44}+10x^{43}+6x^{42}+8x^{41}-5x^{40}-12x^{39}+7x^{38}-7x^{37}+7x^{36}+x^{35}-3x^{34}+10x^{33}+x^{32}-6x^{31}-2x^{30}-10x^{29}-3x^{28}+2x^{27}+9x^{26}-3x^{25}+14x^{24}-8x^{23}-7x^{21}+9x^{20}+3x^{19}-4x^{18}-10x^{17}-7x^{16}+12x^{15}+7x^{14}+2x^{13}-12x^{12}-4x^{11}-2x^{10}+5x^{9}+x^{7}-7x^{6}+7x^{5}-4x^{4}+12x^{3}-6x^{2}+3x-6\big).$$

This seems frightening…!

http://www.njohnston.ca/2010/10/a-derivation-of-conways-degree-71-look-and-say-polynomial/

Happy Oct 31!

empirical?

Loops

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

Basic design strategies

Is one more *reasonable* than the other?

Recursion

theoretical?

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

Loops

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

Design strategy: look for **repetition** + describe it… .

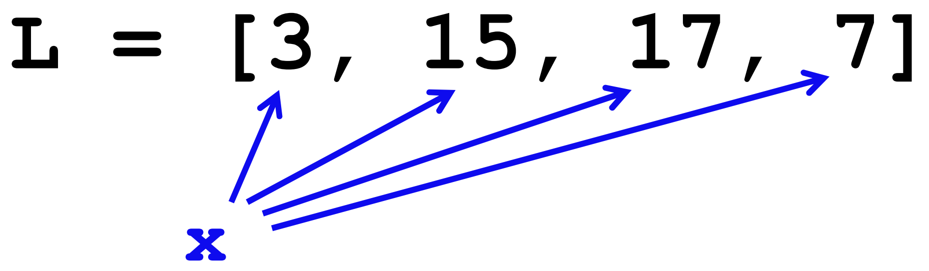
Is one more *reasonable*
n the other?

Design strategy: look for **self-similarity** + describe it… .

Recursion

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

for: *two types*



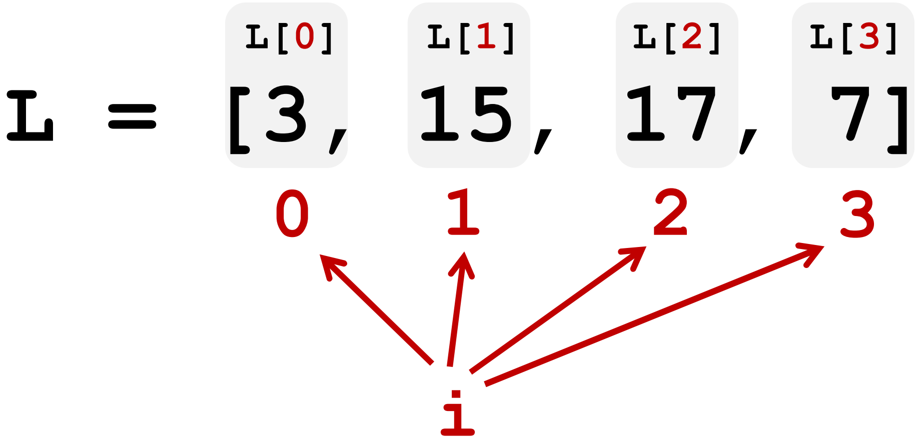
```
L = [3, 15, 17, 7]
     x
```

"deceptively easy"

```
for x in L:
    print x
```

element-based loops

`for`: *two types*



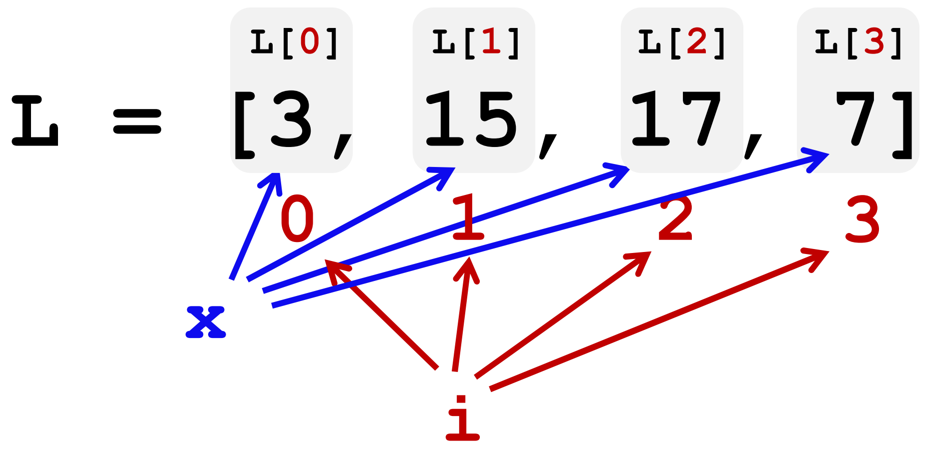
```
              list
for i in range(len(L))
  print L[i]
```

index-based loops

```
for x in L:
  print x
```

element-based loops

elements vs. indices



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

```
def sum(L):
   total = 0
   for x in L:
      total += x
   return total
```

element-based loops

```
def sum(L):
   total = 0
                  list
   for i in range(len(L))
      total += L[i]
   return total
```

index-based loops

hw8pr3: 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 is', 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('Th
```

`input` ALWAYS returns a string – no matter *what* has been typed!

What will Python think?

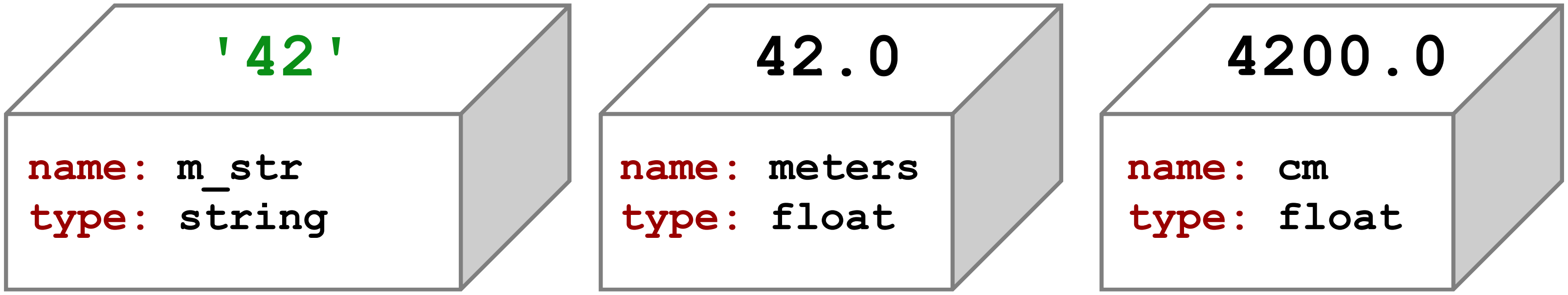
I think I like these units better than light years per year!

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? Does not compute!")
  print("Setting meters = 42")
  meters = 42.0

cm = meters * 100
print('That is', 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? Does not compute!")
   print("Setting meters = 42")
   meters = 42.0

cm = meters * 100
print('That is', 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? Does not compute!")
  print("Setting meters = 42")
  meters = 42.0

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

What could go wrong here?

A larger application

```
def menu():
    """ prints our menu of options """
    print("(0) Continue")
    print("(1) Enter a new list")
    print("(2) Predict")
    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) Predict ! ***... 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) Predict !

other functions as needed...

... and so on ...

Full program example of user-interactions

```
# example looping program with user-input

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 user-interaction loop """
    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

while True:

(3) What line of code runs after this break ?

(1) Which block below handles an input of 7 ?

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

```
        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!")
```

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

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

(6) predict is a function defined elsewhere (off this page) Find the two other functions called here, but defined elsewhere: they both include *find* in their names!

(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 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 )
```

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

```
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"

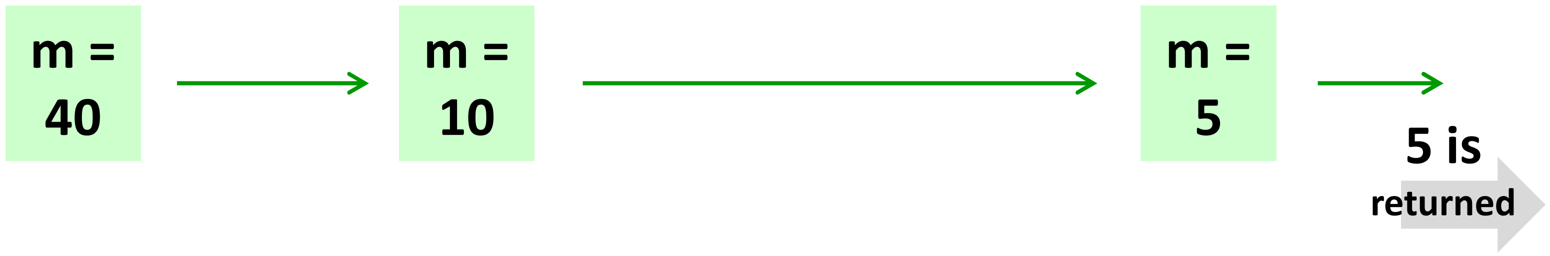
What's the ***idea*** for finding the smallest (minimum) price?

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 the ***day*** of the minimum price?

Mid-term feedback …

Don't put your name

I would love to know any thoughts you have about CS5 thus far in the term.
In particular, how you feel about the time and effort CS5 requires…

On average, how much time per week do you spend on CS5 ***outside class + lab***?

____

How does CS5's workload compare to other classes you're taking this term?

much lighter — about the same — much heavier

How would you judge the ***pace*** of CS5?

much too slow — about right — much too fast

Circle your year: First-year Sophomore Junior Senior Other

Something you'd ***keep*** about CS5 …?

Something you'd ***change about / get rid of / add to*** CS5 …?

Other thoughts optional, but 142% welcome:

Finish this code to return the **index** (location) of L's min.

```
                 L
>>> i_min( [9, 8, 5, 7, 42] )
             0  1  2  3   4
2
```

```
def i_min( L ):

    minval = L[0]
    minloc = 0

               list
    for i in range(len(L)):
        if ___________ :
            minval = ____
            minloc = ____

    return minloc
```

Hints:
track of the minimum value in minval
track the location of the min inside minloc

What does this print?

```
             list
for i in range(4):
             list
    for j in range(4):
        print(abs(i-j),end='')
    print()
```

j	0	1	2	3	
					0
					1
					2
					3
					i

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

Only consider **abs** differences.
L will be a list of numbers.
Hint: Use a nested loop!

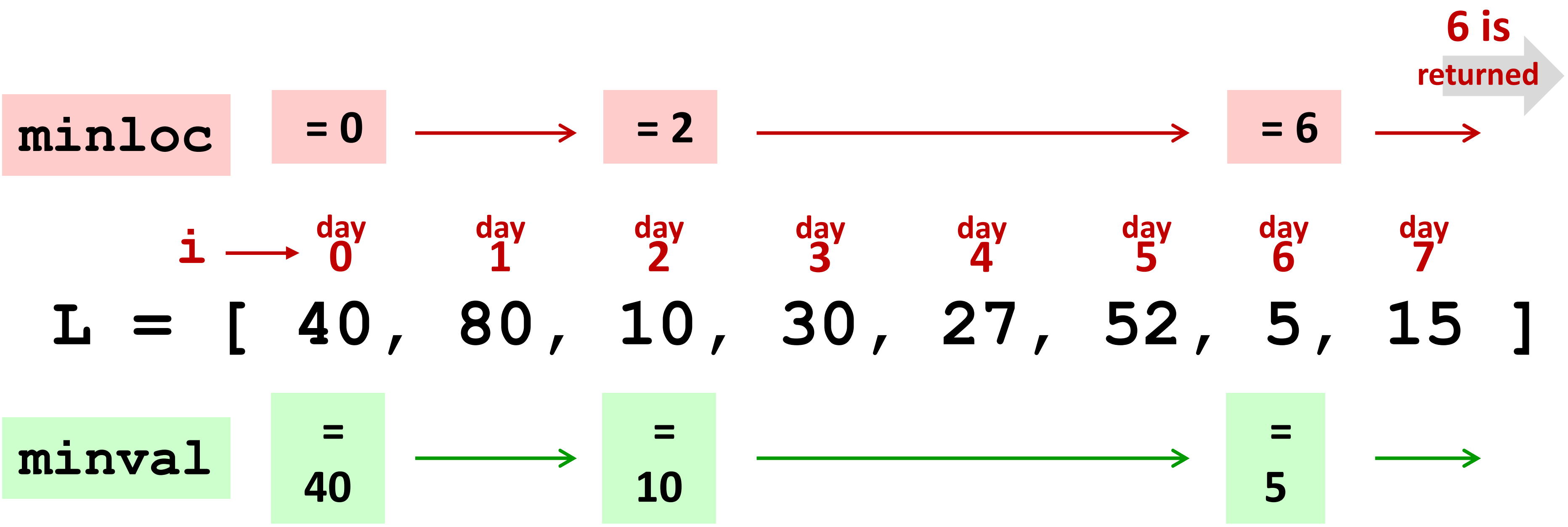
```
>>> mindiff( [42,3,47,100,-9] ) ⟶ 5

def mindiff( L ):
```

Quiz, p.2

Neel and Chaitanya

Brots… !



minloc = 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

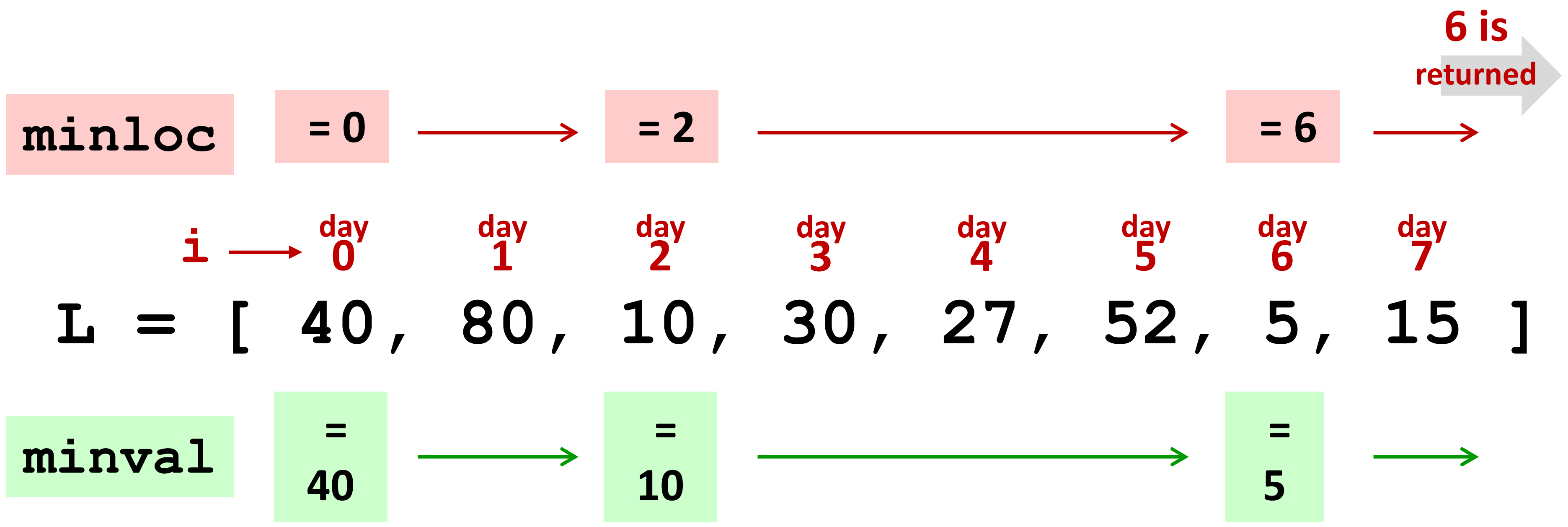
```
def i_min( L ):
  minval = L[0]
  minloc = 0
  for i in range(len(L)):
    if

  return minloc
```

track *both* day and price

loop!

update when needed



minloc = 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 →

```
def i_min( L ):
   minval = L[0]
   minloc = 0
   for i in range(len(L)):
    if  L[i] < minval:
       minval = L[i]
       minloc = i
   return minloc
```

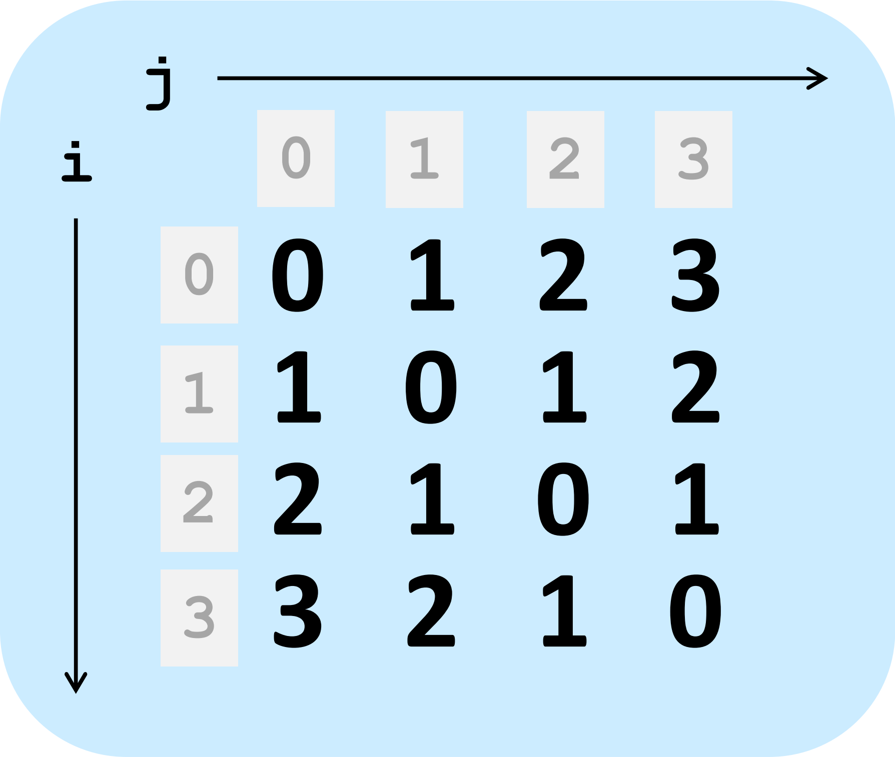
track ***both*** day and price

loop!

update when needed

Nested loops...

```
                   [0,1,2,3]
            list
for i in range(4):
                 list
   for j in range(4):
      print(abs(i-j),end='')
   print()
```



	0	1	2	3
0	0	1	2	3
1	1	0	1	2
2	2	1	0	1
3	3	2	1	0

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

$$\text{mindiff}(\ \underbrace{[42,\underset{\uparrow}{3},\underset{\uparrow}{7},100,-9]}_{\mathbf{L}})$$

4

```
def mindiff( L ):

  m = abs(L[1]-L[0])

  for i in range(len(L)):
    for j in range(    ,len(L)):

      if

  return m
```

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.**

$$\text{mindiff}(\ \underbrace{[42,\underset{\uparrow}{3},\underset{\uparrow}{7},100,-9]}_{\mathbf{L}}\ )$$

4

```
def mindiff( L ):

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

   return m
```

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>...

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

```
for each buy-day, b:
    for each sell-day, s:
        compute the profit
        if it's the max-so-far:
            remember it in a variable!
```

Important fine print:

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