Putting loops to work…

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

[26250 5250, 1050, 210, ?]

[90123241791111, 93551622, 121074, 3111, ?]

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

Putting loops to work…

[-35, -24, -13, -2, 9, 20, 31, 42]

[26250 5250, 1050, 210, 42]

[90123241791111, 93551622, 121074, 3111, 42]

[1, 11, 21, 1211, 111221, 312211]

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

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

theoretical?

Loops

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

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

Is one more *reasonable* than 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]
```

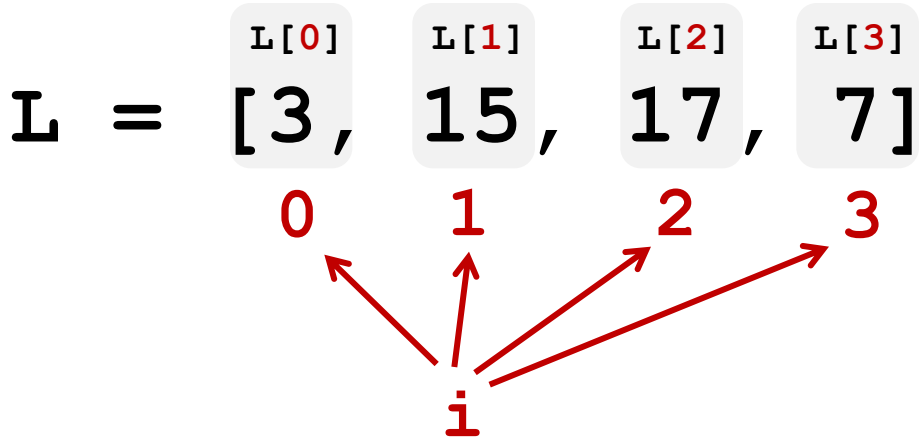


"deceptively easy"

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

element-based loops

`for`: *two types*



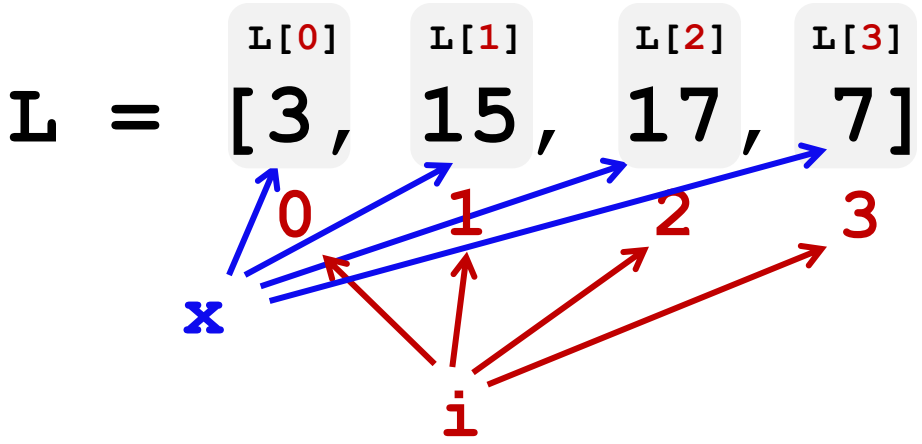

```
for i in range(len(L)):
  print L[i]
```

index-based loops

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

element-based loops

elements vs. indices




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

element-based loops

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

index-based loops

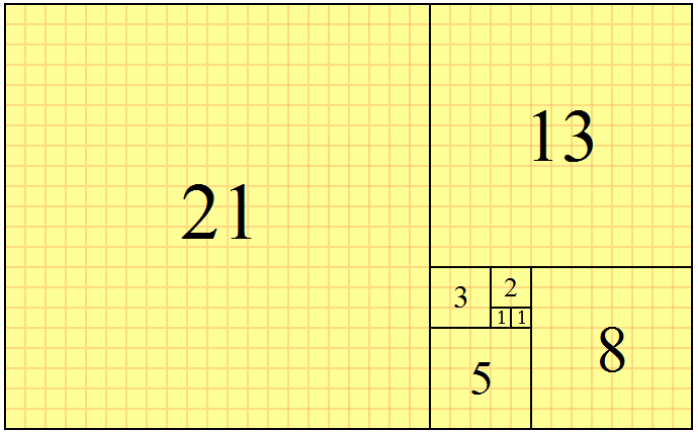
Fibonacci sequence

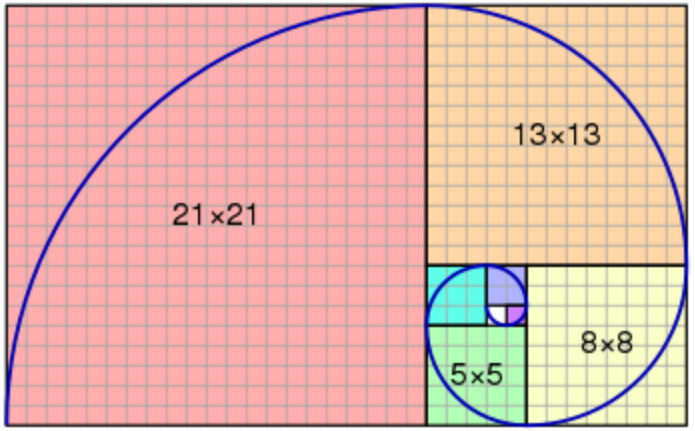
1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …

1+1 = 2, 1+2 = 3, 2+3 = 5, …

Sequence starts with the number 1,1

Next number in the sequence is the sum of the previous 2 numbers





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 min and its day
(4) Find the max and its day
(5) Your TTS investment plan
(9) Quit
Enter your choice:
```

User input...

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

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 meters? ')

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!

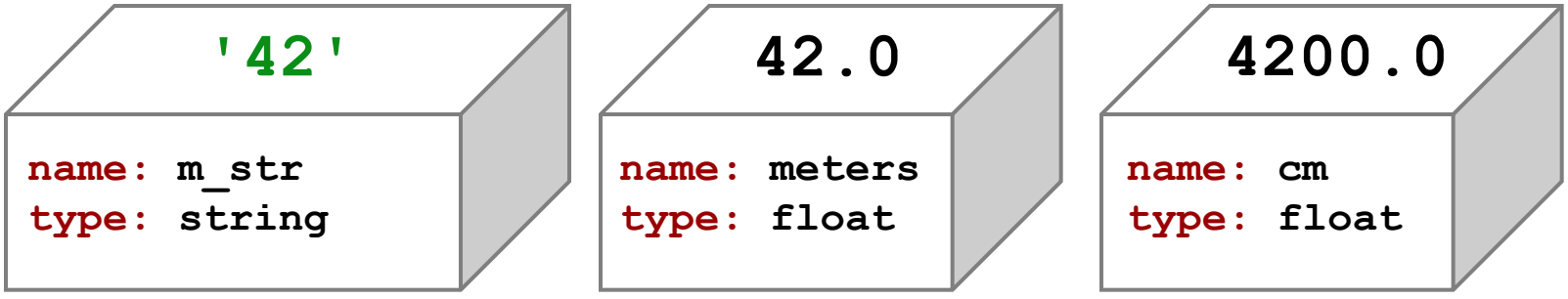


Fix #1: convert to the right type

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

meters = float(m_str)

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



... but crash-able

Fix #2: convert and check

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

try:
    meters = float(m_str)
except:
    print("What? Does not compute!")
    print("I don't get", m_str)
    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("I don't get", m_str)
   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

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:
            print("I didn't understand that")
            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

        elif uc == 0:
            continue

        elif uc == 1:
            ... split input ...

        elif uc == 2:
```

(9) Quit

`break` jumps *out of the loop*

(0) Continue

`continue` jumps *back to the top*

(1) Get new list

use **`str.split()`** to make L

(2) Predict !

other functions as needed...

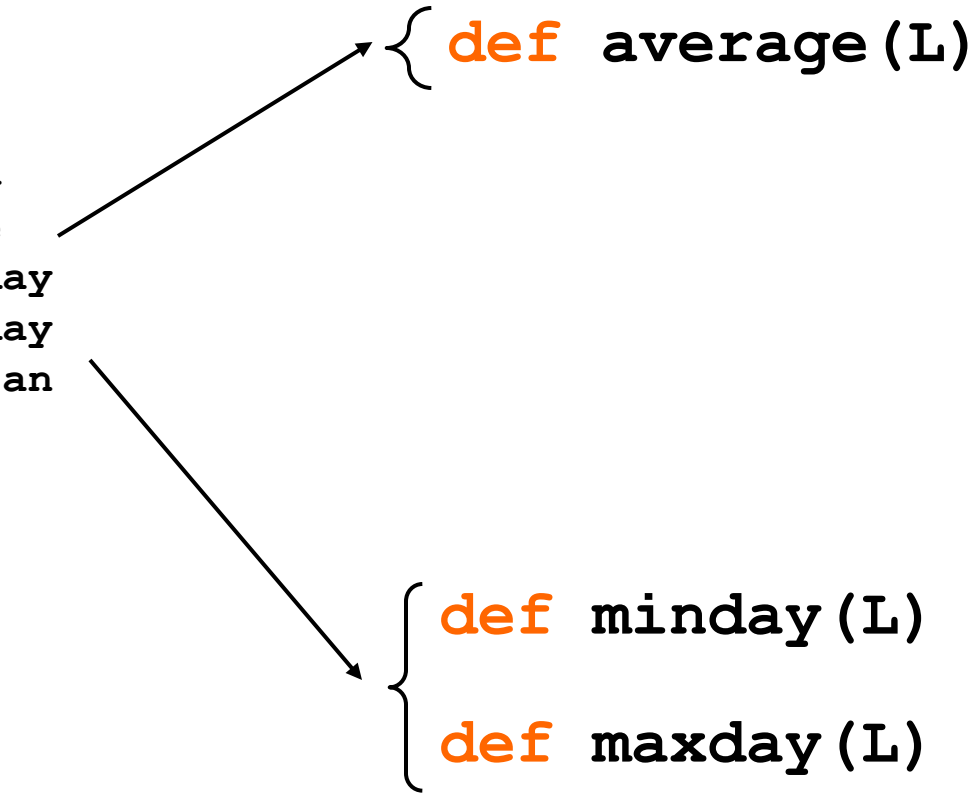
... and so on ...

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 min and its day
(4) Find the max and its day
(5) Your TTS investment plan
(9) Quit
Enter your choice:
```



Min price




```
          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]
```

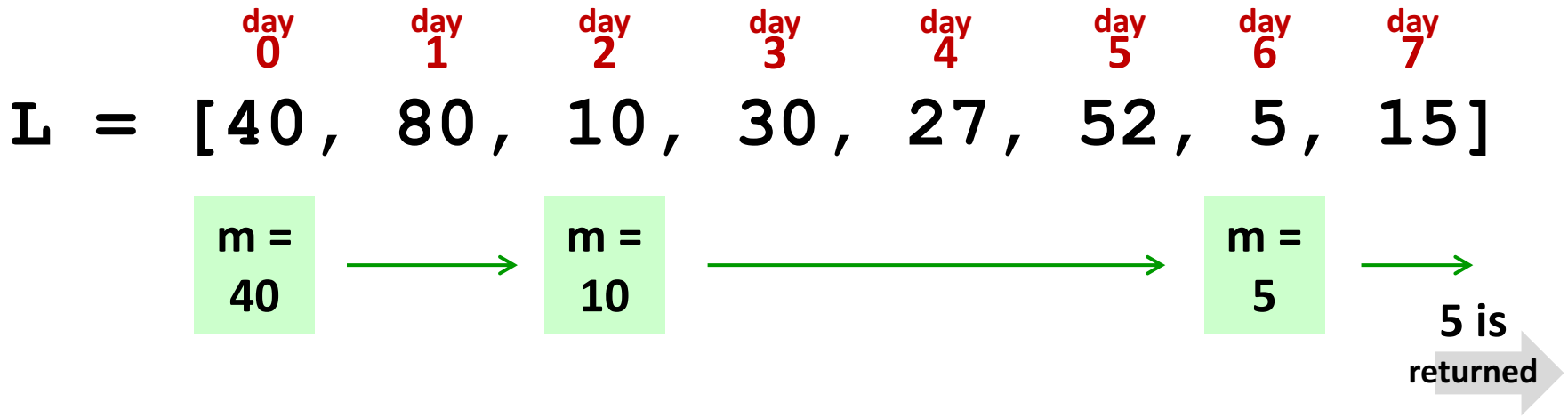
m =

m is the
"min so far"

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

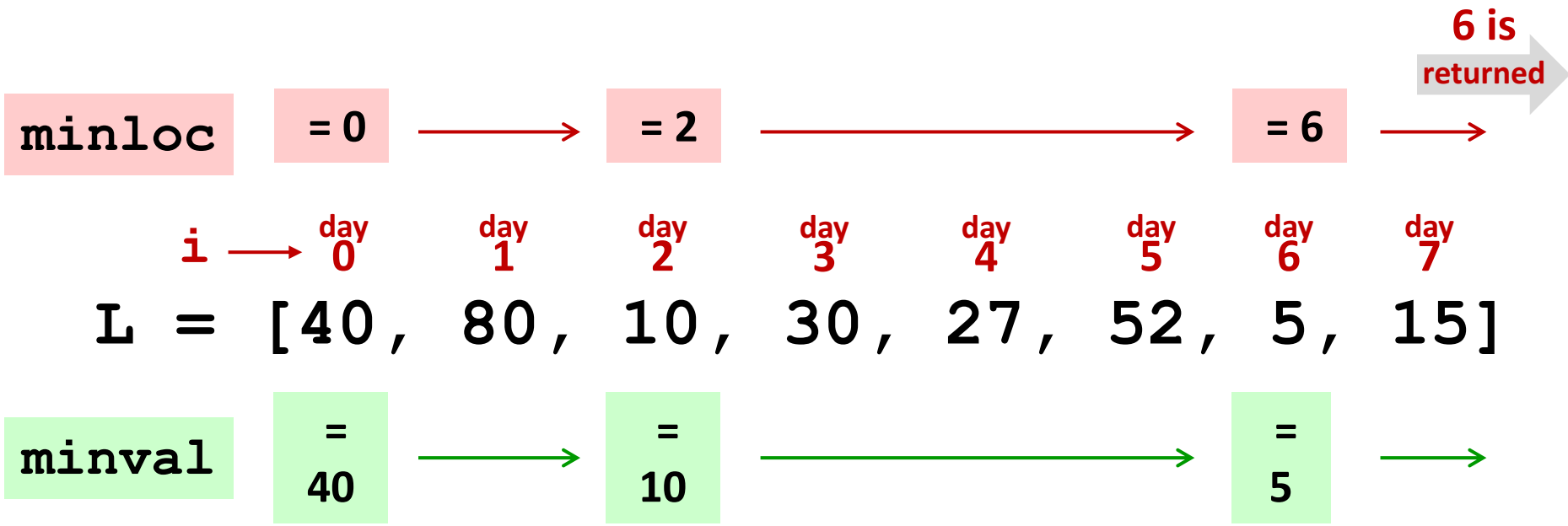
Track the value of the ***minimum so far*** as you loop over L

Min price vs. min ***day***




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

What about the ***day*** of the minimum price?

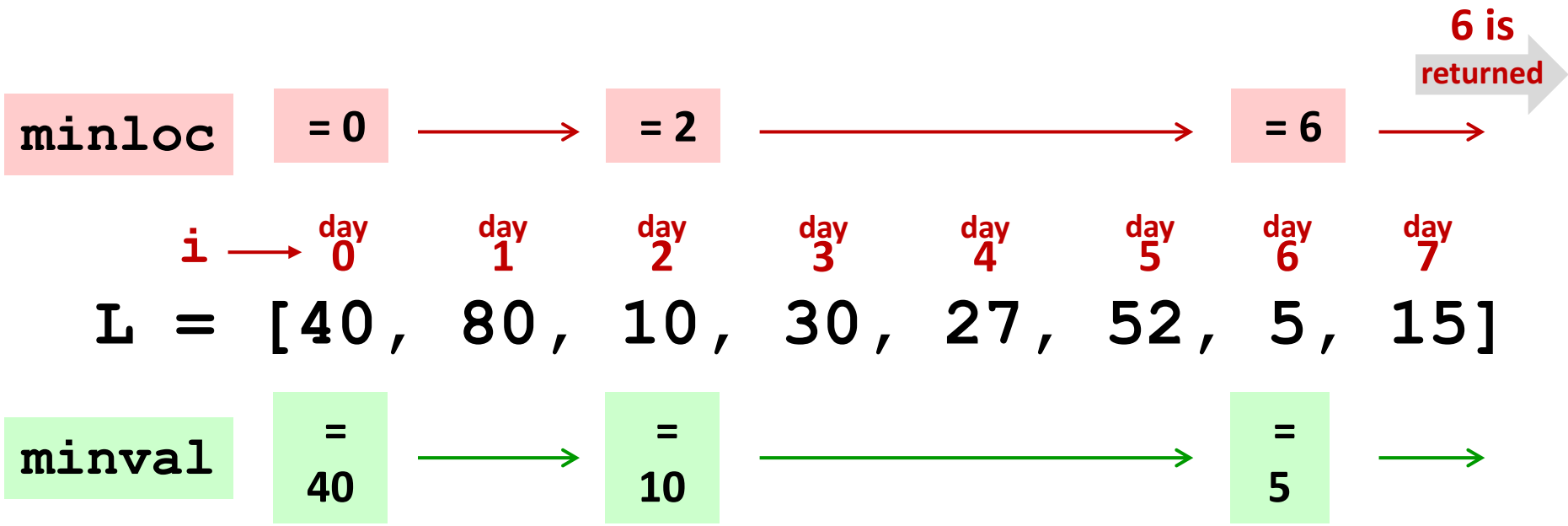



```
def i_min(L):
    minval = L[0]
    minloc = 0
    for i in range(len(L)):
        if
            
    return minloc
```

track ***both*** day and price

loop!

update when needed




```
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

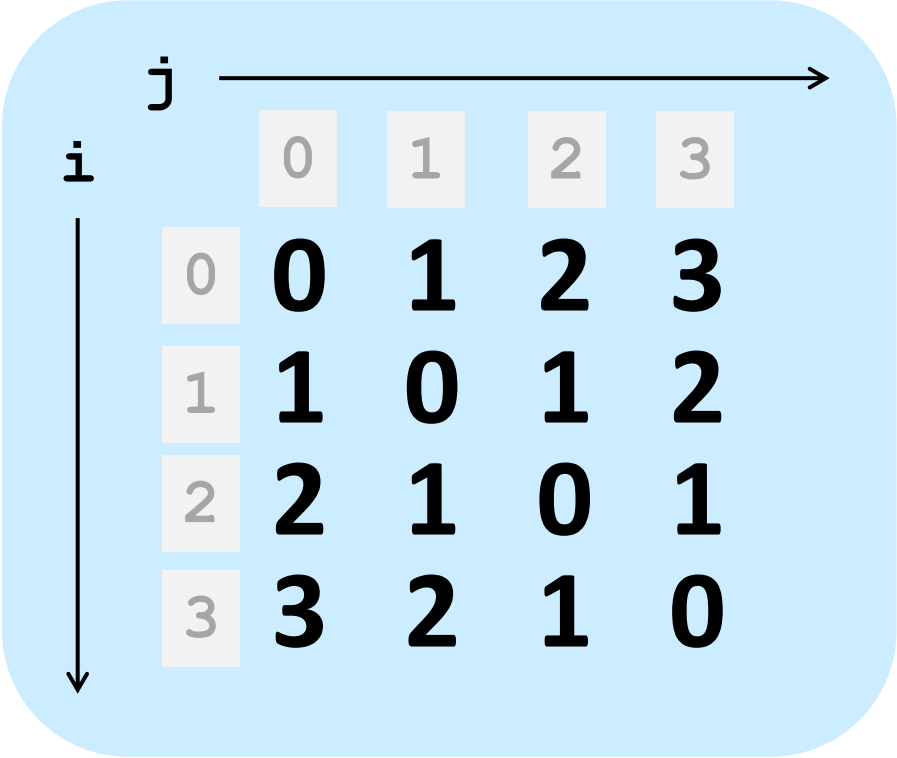
loop!

update when needed

Nested loops...

[0,1,2,3]

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



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

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

4

L

```
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 loop over **L twice**...

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

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

4

L

```
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 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 min and its day
(4) Find the max and its day
(5) 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.

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.