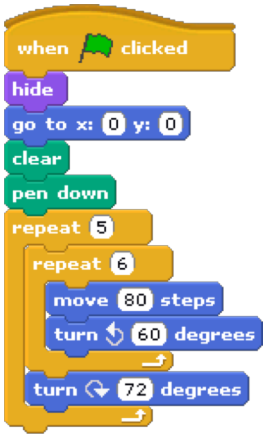
Nested Loops

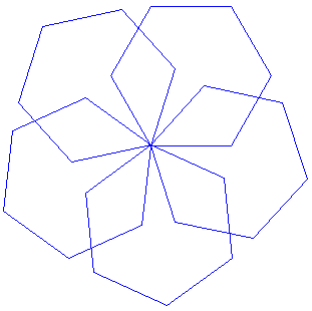
Computer Science 111
Boston University

Vahid Azadeh-Ranjbar, Ph.D.

based in part on notes from the CS-for-All curriculum developed at Harvey Mudd College

Recall: Repeating a Repetition!





- One loop inside another loop!
  - known as a *nested loop*
- How many times is the *move* statement executed above?

Repeating a Repetition!

```
for i in range(3):
    for j in range(4):
        print(i, j)
```

inner loop

outer loop

Repeating a Repetition!

```
for i in range(3):          # 0, 1, 2
    for j in range(4):      # 0, 1, 2, 3
        print(i, j)
```

```
0 0
```

Repeating a Repetition!

```
for i in range(3):          # 0, 1, 2
    for j in range(4):      # 0, 1, 2, 3
        print(i, j)
```

```
0 0
0 1
```

Repeating a Repetition!

```
for i in range(3):          # 0, 1, 2
    for j in range(4):      # 0, 1, 2, 3
        print(i, j)
```

```
0 0
0 1
0 2
```

Repeating a Repetition!

```
for i in range(3):          # 0, 1, 2
    for j in range(4):      # 0, 1, 2, 3
        print(i, j)
```

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

Repeating a Repetition!

```
for i in range(3):          # 0, 1, 2
    for j in range(4):      # 0, 1, 2, 3
        print(i, j)
```

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

Repeating a Repetition!

```
for i in range(3):          # 0, 1, 2
    for j in range(4):      # 0, 1, 2, 3
        print(i, j)
```

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

Repeating a Repetition!

```
for i in range(3):
    for j in range(4):
        print(i, j)
    print('---')
```

inner loop

outer loop

Repeating a Repetition!

```
for i in range(3):
    for j in range(4):
        print(i, j)
    print('---')
```

inner loop

outer loop

```
0 0
0 1
0 2
0 3
---
1 0
1 1
1 2
1 3
---
2 0
2 1
2 2
2 3
---
```

How many lines are printed?

```
for i in range(5):
    for j in range(7):
        print(i, j)
```

A. 4

B. 5

C. 7

D. 24

E. 35

How many lines are printed?

```
for i in range(5):
    for j in range(7):
        print(i, j)
```

A. 4

B. 5

C. 7

D. 24

E. **35**

full output:

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

Tracing a Nested `for` Loop

```
for i in range(5):      # [0,1,2,3,4]
    for j in range(i):
        print(i, j)
```

i	range(i)	j	value printed

Tracing a Nested `for` Loop

```
for i in range(5):      # [0,1,2,3,4]
    for j in range(i):
        print(i, j)
```

i	range(i)	j	value printed
0	[]	none	nothing (we exit the inner loop)
1	[0]	0	1 0
2	[0,1]	0	2 0
		1	2 1
3	[0,1,2]	0	3 0
		1	3 1
		2	3 2
4	[0,1,2,3]	0	4 0
		1	4 1
		2	4 2
		3	4 3

full output:

```
1 0
2 0
2 1
3 0
3 1
3 2
4 0
4 1
4 2
4 3
```

Second Example: Tracing a Nested `for` Loop

```
for i in range(4):
    for j in range(i, 3):
        print(i, j)
    print(j)
```

i	range(i, 3)	j	value printed

Second Example: Tracing a Nested `for` Loop

```
for i in range(4):      # [0, 1, 2, 3]
    for j in range(i, 3):
        print(i, j)
    print(j)
# would go here next
```

i	range(i, 3)	j	value printed
0	[0, 1, 2]	0	0 0
		1	0 1
		2	0 2
			2
1	[1, 2]	1	1 1
		2	1 2
			2
2	[2]	2	2 2
			2
3	[], so body of inner loop doesn't execute		
			2

full output:

```
0 0
0 1
0 2
2
1 1
1 2
2
2 2
2
2
```

PS 6: T.T. Securities (TTS)

Analyzes a sequence of stock prices

	day 0	day 1	day 2	day 3	day 4	day 5	day 6	day 7
prices =	[45,	80,	10,	30,	27,	50,	5,	15]

You will implement a menu of options:

```
(0) Input a new list of prices
(1) Print the current list
(2) Find the latest price
(3) Find the average price
...
(8) Quit
Enter your choice:
```

Our starter code

```
def display_menu():
    """ prints a menu of options
    """
    print()
    print('(0) Input a new list of prices')
    print('(1) Print the current prices')
    print('(2) Find the latest price')
    ## Add the new menu options here.

    print('(8) Quit')
    print()

...
```

Our starter code

```
def tts():
    prices = []
    while True:
        display_menu()
        choice = int(input('Enter your choice: '))
        print()
        if choice == 0:
            prices = get_new_prices()
        elif choice == 8:
            break
        elif choice == 1:
            print_prices(prices)
        elif choice == 2:
            latest = latest_price(prices)
            print('The latest price is', latest)
        ## add code to process the other choices here
        ...
    print('See you yesterday!')
```

The remainder of the program

- Each menu option should have its own helper function.
- Each function will use one or more loops.
  - ***most of them will <u>not</u> be nested!***
- You may *not* use the built-in `sum`, `min`, or `max` functions.
  - use your own loops instead!

T.T. Securities
==
Time Travel
Securities!

```
(0) Input a new list of prices
(1) Print the current list
(2) Find the latest price
(3) Find the average price
...
(7) Your TTS investment plan
(8) Quit
Enter your choice:
```

The TTS Advantage!

```
prices = [45, 80, 10, 30, 27, 50, 5, 15]
```

Day	Price
0	45.00
1	80.00
2	10.00
3	30.00
4	27.00
5	50.00
6	5.00
7	15.00

What is the best TTS investment strategy here?

To be realistic, however, you may only sell ***after*** you buy.

The TTS Advantage!

```
prices = [45, 80, 10, 30, 27, 50, 5, 15]
```

Day	Price
0	45.00
1	80.00
2	10.00
3	30.00
4	27.00
5	50.00
6	5.00
7	15.00

What is the best TTS investment strategy here?

To be realistic, however, you may only sell ***after*** you buy.

Finding a minimum difference

`diff` should return the **smallest** absolute diff. between any value from `l1` and any value from `l2`.

```
                 l1        l2
>>> diff([12,3,7], [6,0,5])
1
```

- How can we try all possible pairs of values?

Finding a minimum difference

`diff` should return the **smallest** absolute diff. between any value from `l1` and any value from `l2`.

```
                 l1        l2
>>> diff([12,3,7], [6,0,5])
1
```

- How can we try all possible pairs of values?
  use nested loops!

- As we try pairs, we keep track of the min diff thus far:

```
def diff(l1, l2):
    mindiff = abs(l1[0]-l2[0])
    for x in l1:
        for y in l2:
            d = abs(x - y)
            if d < mindiff:
                mindiff = d
    return mindiff
```

What if we want the *indices* of the min-diff values?

```
>>> diff_indices([12,3,7], [6,0,5])
[2, 0]
```

l1 l2

index of value in l2

index of value in l1

```
def diff_indices(l1, l2):   # what needs to change?
    mindiff = abs(l1[0] - l2[0])

    for x in l1:
        for y in l2:
            d = abs(x - y)
            if d < mindiff:
                mindiff = d

    return mindiff
```

What if we want the *indices* of the min-diff values?

```
>>> diff_indices([12,3,7], [6,0,5])
[2, 0]
```

l1 l2

index of value in l2

index of value in l1

```
def diff_indices(l1, l2):
    mindiff = abs(l1[0] - l2[0])
    pos1 = 0
    pos2 = 0

    for i in range(len(l1)):
        for j in range(len(l2)):
            d = abs(l1[i] - l2[j])
            if d < mindiff:
                mindiff = d
                pos1 = i
                pos2 = j
    return [pos1, pos2]
```

Printing Patterns

```
for row in range(3):
    for col in range(4):
        print('#', end=' ')
    print()  # go to next line
```

col

row	0	1	2	3
0	#	#	#	#
1	#	#	#	#
2	#	#	#	#

Fill in the Blank #1

```
for row in range(3):
    for col in range(6):
        print(_____, end=' ')
    print()  # go to next line
```

col

row

```
0 1 2 3 4 5
0 1 2 3 4 5
0 1 2 3 4 5
```

Fill in the Blank #1

```
for row in range(3):
    for col in range(6):
        print(col, end=' ')
    print()  # go to next line
```

col

row

```
0 1 2 3 4 5
0 1 2 3 4 5
0 1 2 3 4 5
```

Fill in the Blank #2

```
for row in range(3):
    for col in range(6):
        print(_____, end=' ')
    print()  # go to next line
```

col

row

```
0 0 0 0 0 0
1 1 1 1 1 1
2 2 2 2 2 2
```

Fill in the Blank #2

```
for row in range(3):
    for col in range(6):
        print(row, end=' ')
    print()  # go to next line
```

col

row

```
0 0 0 0 0 0
1 1 1 1 1 1
2 2 2 2 2 2
```

What is needed in the blanks to get this pattern?

```
for row in range(5):
    for col in ___________:
        print(_____, end=' ')
    print()  # go to next line
```

```
0 0 0 0 0
1 1 1 1
2 2 2
3 3
4
```

	first blank	second blank
A.	`range(row)`	`row`
B.	`range(row)`	`col`
C.	`range(5 - row)`	`row`
D.	`range(5 - row)`	`col`
E.	none of the above	

What is needed in the blanks to get this pattern?

```
for row in range(5):
    for col in ___________:
        print(_____, end=' ')
    print()  # go to next line
```

```
0 0 0 0 0
1 1 1 1
2 2 2
3 3
4
```

	first blank	second blank
A.	`range(row)`	`row`
B.	`range(row)`	`col`
C.	`range(5 - row)`	`row`
D.	`range(5 - row)`	`col`
E.	none of the above	

What is needed in the blank to get this pattern?

```
for row in range(3):
    for col in range(6):
        print(_____, end=' ')
    print()  # go to next line
```

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

A. row + col

B. row – col

C. row + col + 1

D. row – col + 1

E. row % col

if you have time...

```
0 1 0 1 0 1
1 0 1 0 1 0
0 1 0 1 0 1
```

What is needed in the blank to get this pattern?

```
for row in range(3):
    for col in range(6):
        print(_____, end=' ')
    print()  # go to next line
```

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

A. **row + col**

B. row – col

C. row + col + 1

D. row – col + 1

E. row % col

if you have time...

```
0 1 0 1 0 1
1 0 1 0 1 0
0 1 0 1 0 1
```

`(row + col) % 2`

Nested loops: Rows vs. columns

For these loops, let **side=5**:

What does this print?

```
for row in range(side):
  for col in range(side):
    if row >= col:
        print(row, end=' ')
    else:
        print(' ', end=' ')
  print()
```

columns

rows	0	1	2	3	4
0					
1					
2					
3					
4					

What `if` tests print these?

```
for row in range(side):
  for col in range(side):
    if
        print(row, end=' ')
    else:
        print(' ', end=' ')
  print()
```

	0	1	2	3	4
0	0	0	0	0	0
1	1	1	1	1	
2	2	2	2		
3	3	3			
4	4				

```
for row in range(side):
  for col in range(side):
    if
        print(row, end=' ')
    else:
        print(' ', end=' ')
  print()
```

	0	1	2	3	4
0	0	0	0	0	0
1	1	1	1	1	
2	2	2	2		
3	3	3	3	3	
4	4	4	4	4	4

Extra! how about *this hourglass*?

```
0 0 0 0 0
  1 1 1
    2
  3 3 3
4 4 4 4 4
```


Nested loops: Rows vs. columns

For these loops, let **side=5**:

What does this print?

```
for row in range(side):
  for col in range(side):
    if row >= col:
        print(row, end=' ')
    else:
        print(' ', end=' ')
  print()
```

columns

rows	0	1	2	3	4
0	0				
1	1	1			
2	2	2	2		
3	3	3	3	3	
4	4	4	4	4	4

What `if` tests print these?

```
for row in range(side):
  for col in range(side):
    if
        print(row, end=' ')
    else:
        print(' ', end=' ')
  print()
```

	0	1	2	3	4
0	0	0	0	0	0
1	1	1	1	1	
2	2	2	2		
3	3	3			
4	4				

```
for row in range(side):
  for col in range(side):
    if
        print(row, end=' ')
    else:
        print(' ', end=' ')
  print()
```

	0	1	2	3	4
0	0	0	0	0	0
1	1	1	1	1	
2	2	2	2		
3	3	3	3	3	
4	4	4	4	4	4

Extra! how about *this hourglass*?

```
0 0 0 0 0
  1 1 1
    2
  3 3 3
4 4 4 4 4
```

Nested loops: Rows vs. columns

For these loops, let **side=5**:

What does this print?

```
for row in range(side):
  for col in range(side):
    if row >= col:
        print(row, end=' ')
    else:
        print(' ', end=' ')
  print()
```

columns

rows	0	1	2	3	4
0	0				
1	1	1			
2	2	2	2		
3	3	3	3	3	
4	4	4	4	4	4

What `if` tests print these?

```
for row in range(side):
  for col in range(side):
    if row + col <= 4:
        print(row, end=' ')
    else:
        print(' ', end=' ')
  print()
```

	0	1	2	3	4
0	0	0	0	0	0
1	1	1	1	1	
2	2	2	2		
3	3	3			
4	4				

more generally:

`row + col < side`

```
for row in range(side):
  for col in range(side):
    if
        print(row, end=' ')
    else:
        print(' ', end=' ')
  print()
```

	0	1	2	3	4
0	0	0	0	0	0
1	1	1	1	1	
2	2	2	2		
3	3	3	3	3	
4	4	4	4	4	4

Extra! how about *this hourglass*?

```
0 0 0 0 0
  1 1 1
    2
  3 3 3
4 4 4 4 4
```

```
for row in range(side):
  for col in range(side):
    if row >= col:
        print(row, end=' ')
    else:
        print(' ', end=' ')
  print()
```

A

columns

rows	0	1	2	3	4
0	0				
1	1	1			
2	2	2	2		
3	3	3	3	3	
4	4	4	4	4	4

```
for row in range(side):
  for col in range(side):
    if row + col <= 4:
        print(row, end=' ')
    else:
        print(' ', end=' ')
  print()
```

B

	0	1	2	3	4
0	0	0	0	0	0
1	1	1	1	1	
2	2	2	2		
3	3	3			
4	4				

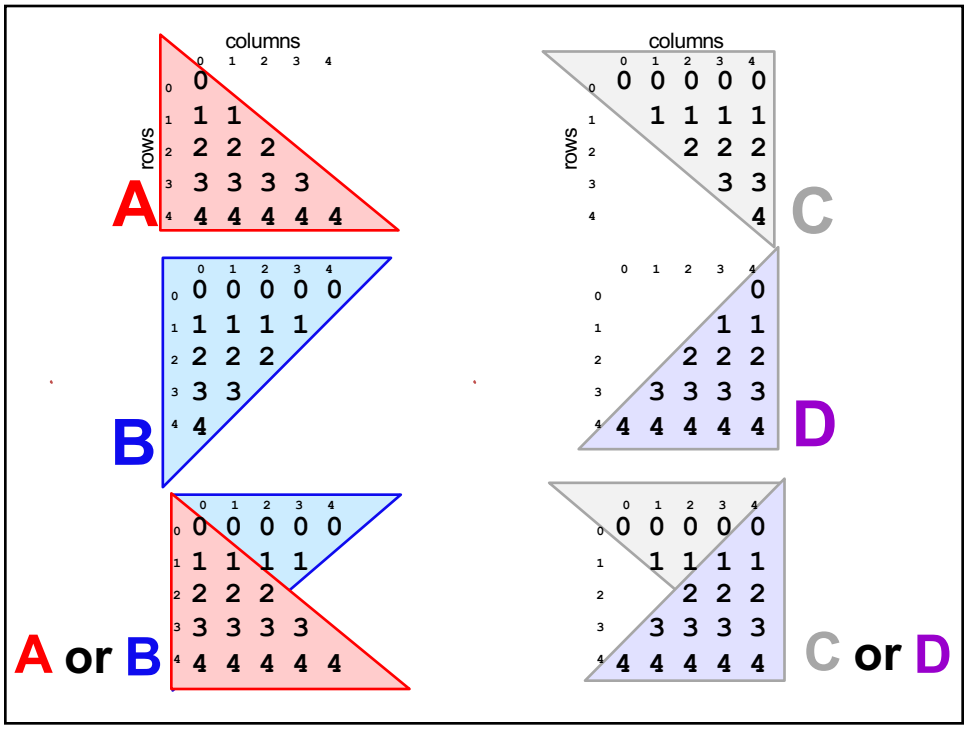
```
for row in range(side):
  for col in range(side):
    if row >= col or row+col<=4
        print(row, end=' ')
    else:
        print(' ', end=' ')
  print()
```

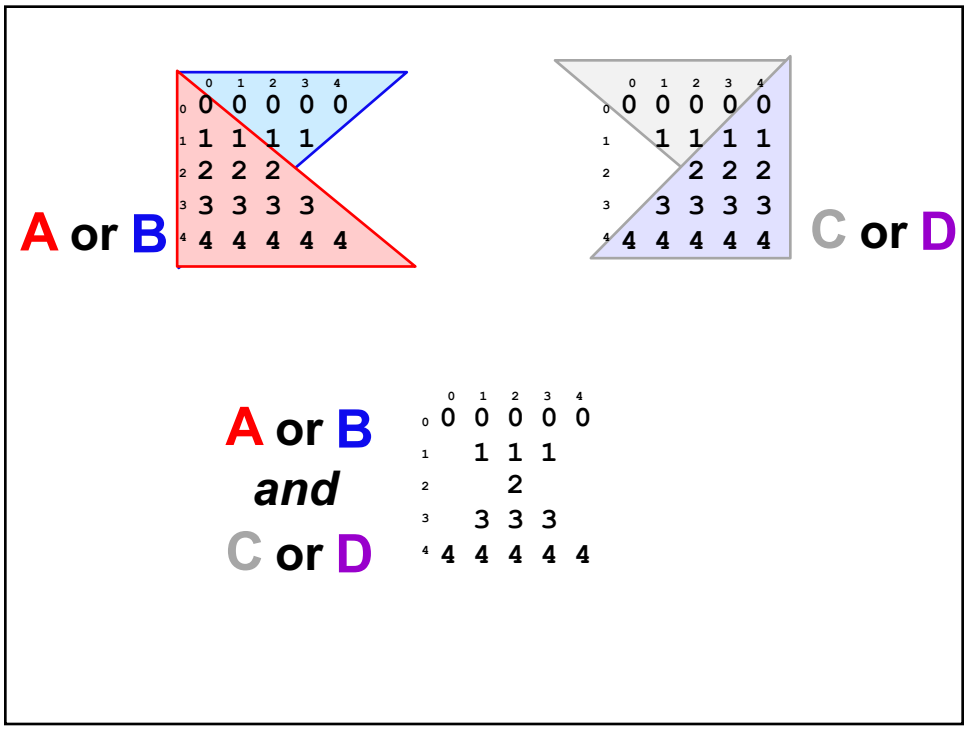
A or B

	0	1	2	3	4
0	0	0	0	0	0
1	1	1	1	1	
2	2	2	2		
3	3	3	3	3	
4	4	4	4	4	4

Extra! how about *this hourglass*?

```
0 0 0 0 0
  1 1 1
    2
  3 3 3
4 4 4 4 4
```


columns
rows
A
B
C
D
A or B
C or D


A or B
C or D
A or B
and
C or D