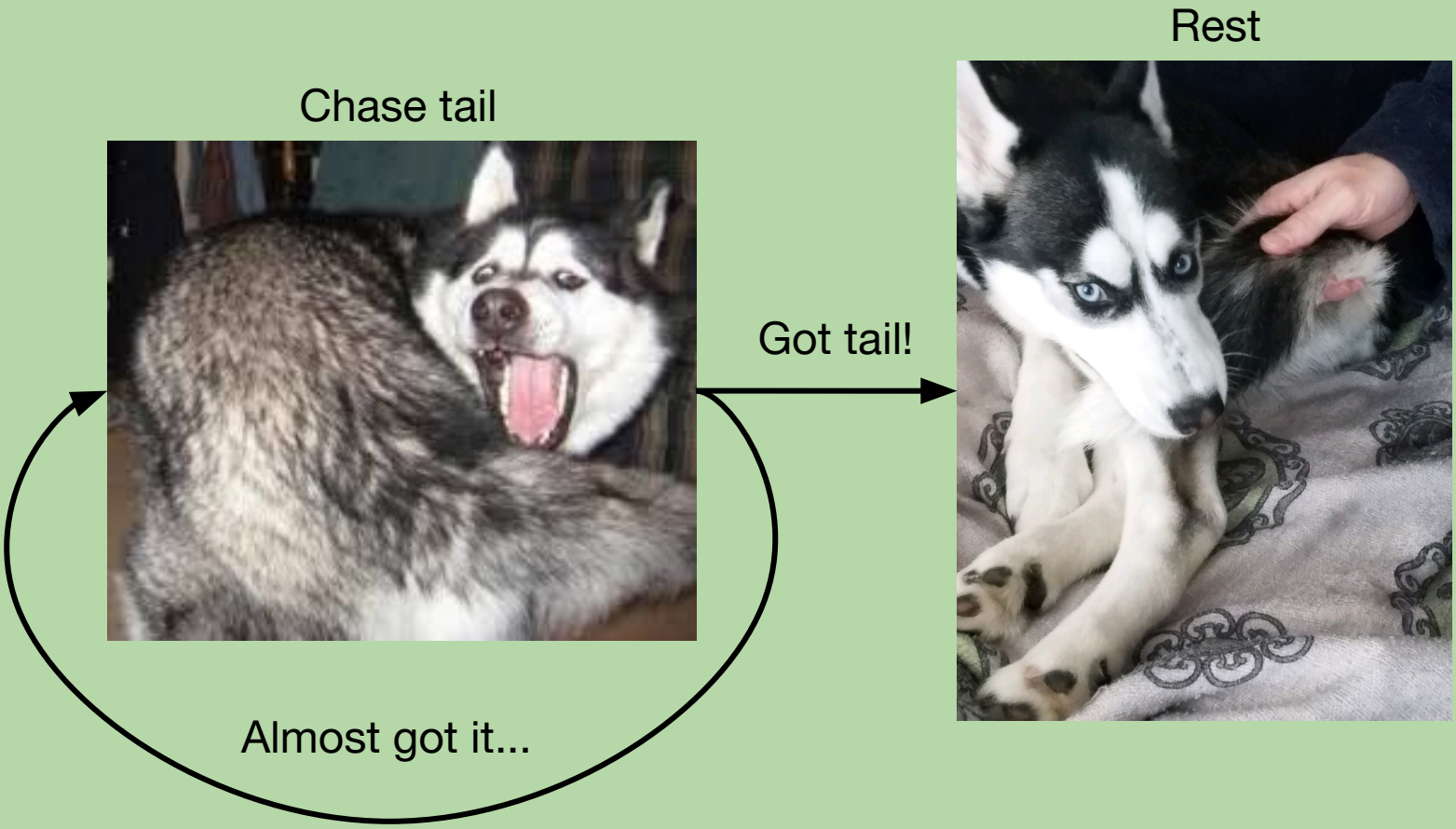
# Lecture 04
# More Iteration, Nested Loops

Meet UTA Jarrett's dog Greta, lying in her "nest"

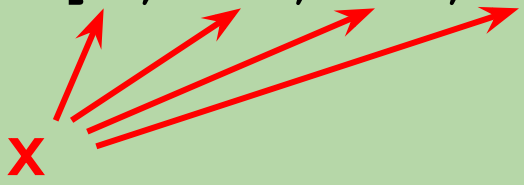
# Indefinite Loops




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

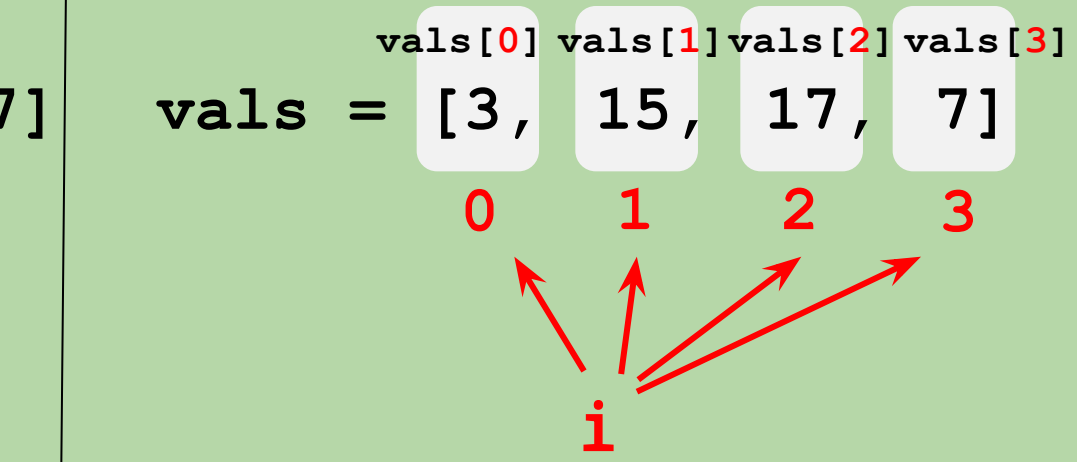
# So far: Two Types of `for` Loops

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




```
def sum(vals):
    result = 0
    for x in vals:
        result += x
    return result
```

***element***-based loop




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

***index***-based loop

**Both are examples of definite loops (i.e., fixed number of iterations)**

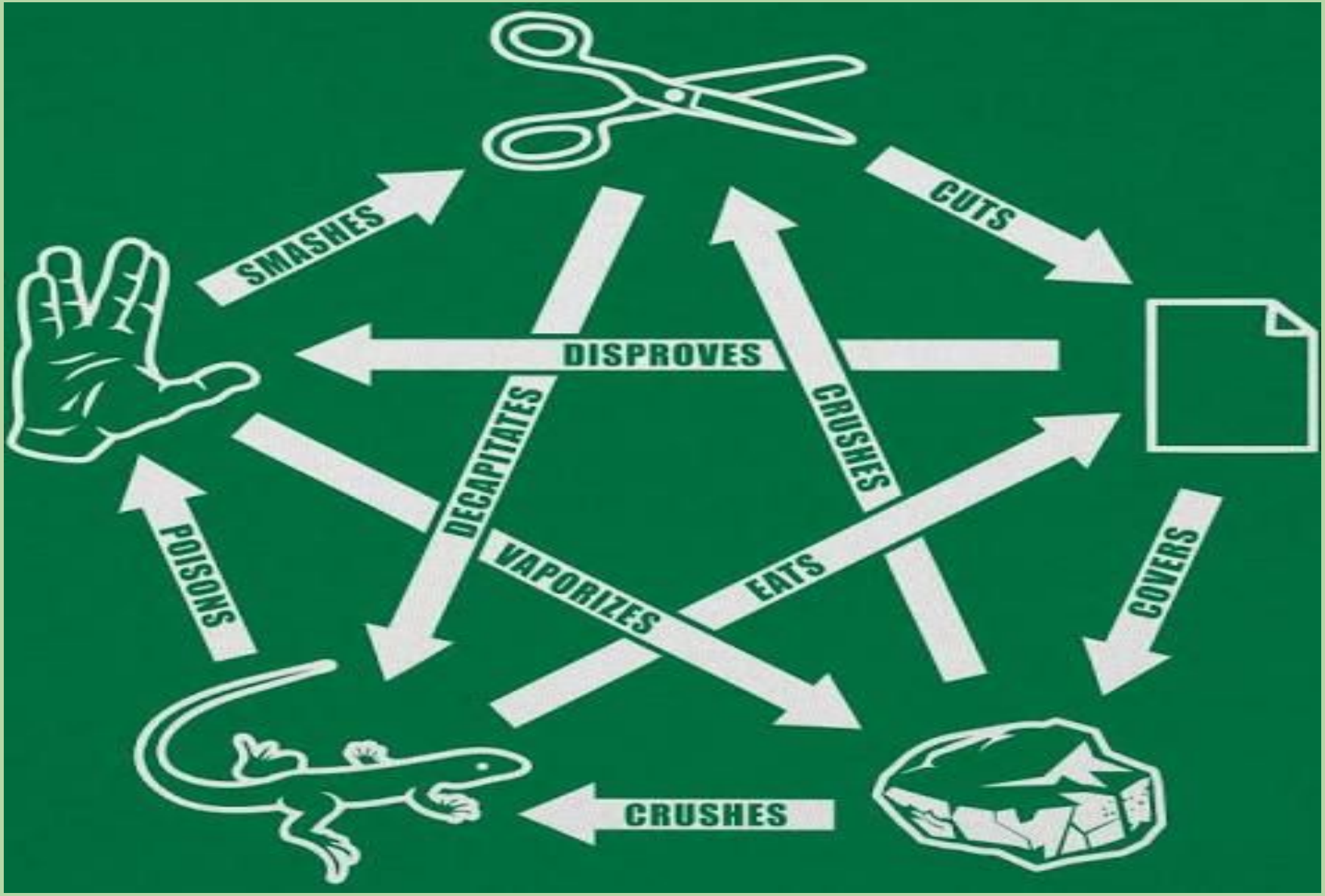
# *Indefinite* Loops

- Use an *indefinite loop* when the # of repetitions you need is:
  - not obvious or known
  - impossible to determine before the loop begins, e.g.,
    - `Finding an element`
    - `Computing an estimate up to some error bound`
    - `Playing a game of rock, paper, scissors (as opposed to one round)`

- Toy problem: `print_multiples(n, bound)`
  - should print all multiples of `n` that are less than `bound`
  - output for `print_multiples(9, 100)`:

```
9 18 27 36 45 54 63 72 81 90 99
```

# Rock, Paper, Scissors, Lizard, Spock



# Indefinite Loop for Printing Multiples

**`while`** loops are how you code indefinite loops in Python:

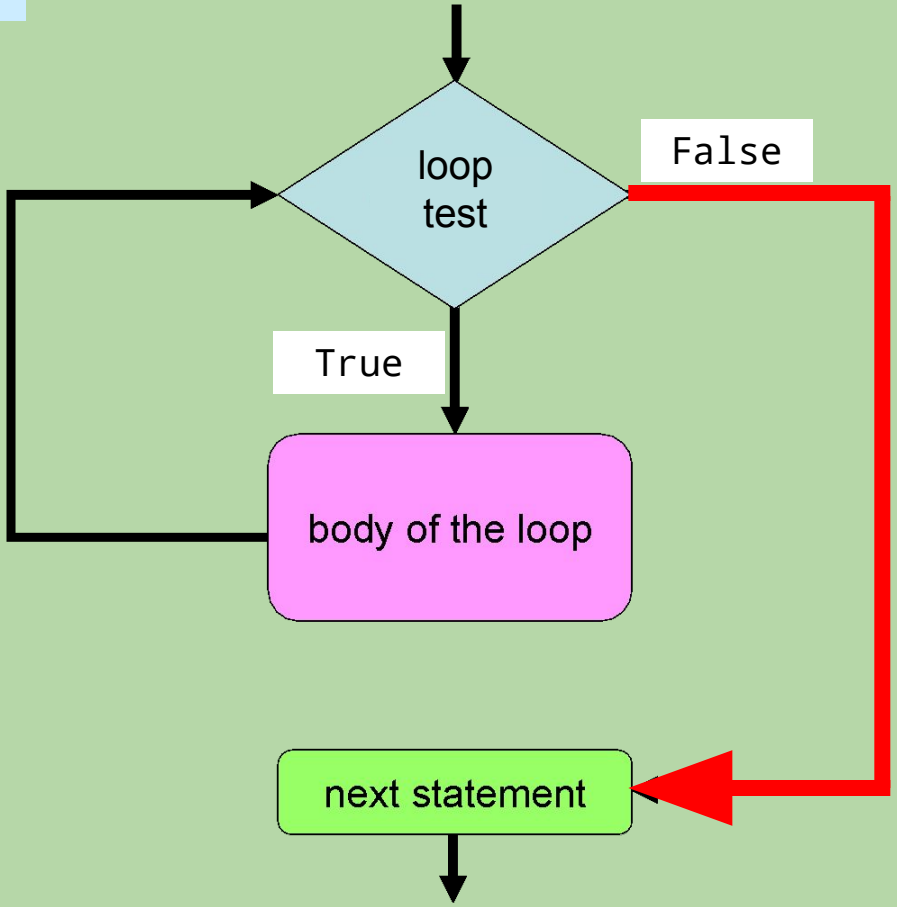
```
def print_multiples(n, bound):
    mult = n
    while mult < bound:
        print(mult, end=" ")
        mult = mult + n
    print()
```

# `while` Loops

```
while <loop test>:
    <body of the loop>
```

Steps:

1. evaluate the loop test (a boolean expression)
2. if it's `True`, execute the statements in the body, and go back to step 1
3. if it's `False`, skip the statements in the body and go to the statement after the loop



# Tracing a `while` Loop

- Let's trace the loop for `print_multiples(15, 70)`:

```
mult = n
while mult < bound:
    print(mult, end=' ')
    mult = mult + n
print()
```

n bound

Prints everything on the same line with spaces in between! Neat!

| mult < bound | output thus far | mult |
|---|---|---|

# Tracing a `while` Loop

- Let's trace the loop for `print_multiples(15, 70)`:


```
mult = n
while mult < bound:
    print(mult, end=' ')
    mult = mult + n
print()
```

| mult < bound | output thus far | mult |
|---|---|---|
| | | 15 |
| 15 < 70 (True) | 15 | 30 |
| 30 < 70 (True) | 15 30 | 45 |
| 45 < 70 (True) | 15 30 45 | 60 |
| 60 < 70 (True) | 15 30 45 60 | 75 |
| 75 < 70 (False) | | |

**so we exit the loop and `print()`**

# Important!

- In general, a `while` loop's test includes a key "loop variable".
- We need to update that loop variable in the body of the loop.
- Failing to update it can produce an *infinite loop*!

- Recall the loop in `print_multiples`:

```
mult = n
  while mult < bound:
      print(mult, end=' ')
      mult = mult + n
```

**What is the loop variable?**
**Where is it updated?**

# Important!

- In general, a `while` loop's test includes a key "loop variable".
- We need to update that loop variable in the body of the loop.
- Failing to update it can produce an *infinite loop*!

- Recall the loop in `print_multiples`:

```
mult = n
  while mult < bound:
      print(mult, end=' ')
      mult = mult + n
```

**What is the loop variable? mult**
**Where is it updated? In the body of the loop**

# Important!

- In general, a `while` loop's test includes a key "loop variable".
- We need to update that loop variable in the body of the loop.
- Failing to update it can produce an *infinite loop*!
- Showing every iteration makes progress towards making the while loop condition false is one way to show a while loop will terminate

# Factorial Using a `while` Loop

- We don't need an indefinite loop, but we can still use `while`!

```
def fac(n):
        result = 1
        while n > 0:
            result *= n
            _____________   # what do we need here?
        return result
```

- Let's trace `fac(4)`:

| n | n > 0 | result |
|---|---|---|

# Factorial Using a `while` Loop

- We don't need an indefinite loop, but we can still use `while`!

```
def fac(n):
        result = 1
        while n > 0:
            result *= n
            n = n - 1
        return result
```

- Let's trace `fac(4)`:

| n | n > 0 | result |
|---|---|---|

# Factorial Using a `while` Loop

- We don't need an indefinite loop, but we can still use `while`!

```
def fac(n):
        result = 1
        while n > 0:
            result *= n
            n = n - 1
        return result
```

- Let's trace `fac(4)`:

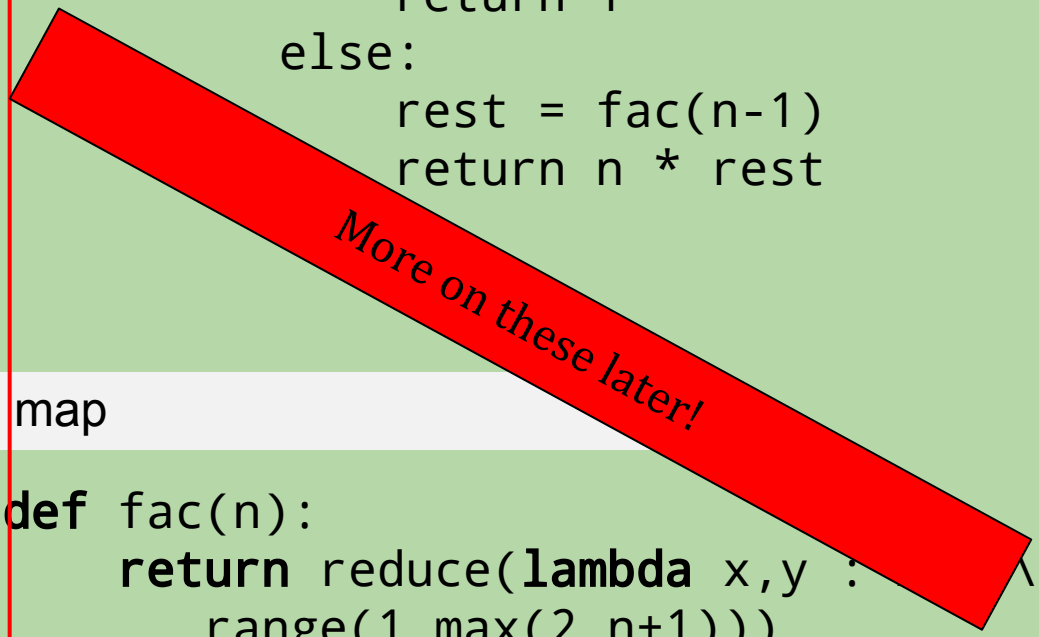
| n | n > 0 | result |
|---|---|---|
| 4 | | 1 |
| 4 | 4 > 0 (True) | 1*4 = 4 |
| 3 | 3 > 0 (True) | 4*3 = 12 |
| 2 | 2 > 0 (True) | 12*2 = 24 |
| 1 | 1 > 0 (True) | 24*1 = 24 |
| 0 | 0 > 0 (False) | |

so we exit the loop and `return 24`

# Factorial Four Ways!

recursion

```
def fac(n):
    if n == 0:
        return 1
    else:
        rest = fac(n-1)
        return n * rest
```




map

```
def fac(n):
    return reduce(lambda x,y : x*y,\
       range(1,max(2,n+1)))
```

for loop

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

while loop

```
def fac(n):
    result = 1
    while n > 0:
        result *= n
        n = n - 1
    return result
```

# Extreme Looping!

- What does this code do?

```
print('It keeps')

while True:
    print('going and')

print('Phew! Done!')
```

# Extreme Looping!

- What does this code do?

```
print('It keeps')
while True:
    print('going and')
print('Phew! Done!')      # never gets here!
```

- An infinite loop!

Use **Ctrl-C** to stop a program inside python

Use **W-F2** to stop a program in PyCharm

# Breaking Out of A Loop

```
import random

while True:
    print('Help!')
    if random.choice(range(10000)) == 111:
        break
    print('Let me out!')

print('At last!')
```

- What are the final two lines that are printed?

# Breaking Out of A Loop

```
import random

while True:
    print('Help!')
    if random.choice(range(10000)) == 111:
        break
    print('Let me out!')

print('At last!')
```

- What are the final two lines that are printed?

```
Help!
At last!
```

- How could we count the number of repetitions?

# Counting the Number of Repetitions

```
import random

count = 1
while True:
    print('Help!')
    if random.choice(range(10000)) == 111:
        break
    print('Let me out!')
    count += 1

print('At last! It took', count, 'tries to escape!')
```

# Important!

- In general, a `while` loop's test includes a key "loop variable".
- We need to update that loop variable in the body of the loop.
- Failing to update it can produce an *infinite loop*!
- Can rely on a statistical argument (e.g., rock, paper, scissors)
- Counting the number of iterations and exiting after a maximum has been reached is a safer way to loop indefinitely

# Counting the Number of Repetitions

```
import random

count = 1
while count<=5000:
    print('Help!')
    if random.choice(range(10000)) == 111:
        break
    print('Let me out!')
    count += 1

print('At last! It took', count, 'tries to escape!')
```

# How many values does this loop print?

```
a = 40
while a > 2:
    a = a // 2
    print(a - 1)
```

| a > 2 | a | prints |
|---|---|---|

A. 2

B. 3

C. 4

D. 5

E. none of these

# How many values does this loop print?

```
a = 40
while a > 2:
    a = a // 2
    print(a - 1)
```

| a > 2 | a | prints |
|---|---|---|
| | 40 | |
| True | 20 | 19 |
| True | 10 | 9 |
| True | 5 | 4 |
| True | 2 | 1 |
| False | | |

A. 2

B. 3

C. **4**

D. 5

E. none of these

# For what inputs does this function return True?

```
def mystery(n):
    while n != 1:
        if n % 2 != 0:
            return False
        n = n // 2
    return True
```

Try tracing these two cases:

| mystery(12) | mystery(8) |
|---|---|
| n | n |
| 12 | 8 |

A. odd numbers

B. even numbers

C. multiples of 4

D. powers of 2

E. none of these

# For what inputs does this function return True?

```
def mystery(n):
    while n != 1:
        if n % 2 != 0:
            return False
        n = n // 2
    return True
```

Try tracing these two cases:

| mystery(12) | mystery(8) |
|---|---|
| n | n |
| 12 | 8 |
| 6 | 4 |
| 3 | 2 |
| False | 1 |
| | True |

A. odd numbers

B. even numbers

C. multiples of 4

D. **powers of 2**

E. none of these

Wesley says it’s break time so it’s break time

# *Nested* Loops!


```
for y in range(84):
  for m in range(12):
    for d in range(f(m,y)):
      for h in range(24):
        for mn in range(60):
          for s in range(60):
            tick()
```

# Nested Loops!

- Nested Loops are loops where a loop appears inside the body of another loop.
  - The loop inside the body is called the inner loop. The other is called the outer loop.
- The inner loop completes all passes for a single pass of the outer loop
  - This is very useful for many types of algorithms, especially with data that has more than one dimension.

# 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 = 5*7 executions of inner code block**

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) |
|---|---|
| 0 | [] |
| 1 | [0] |
| 2 | [0,1] |
| 3 | [0,1,2] |
| 4 | [0,1,2,3] |

| j | value printed |
|---|---|
| none | nothing (we exit the inner loop) |
| 0 | 1 0 |
| 0 | 2 0 |
| 1 | 2 1 |
| 0 | 3 0 |
| 1 | 3 1 |
| 2 | 3 2 |
| 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
```

# Side Note: Staying on the Same Line When Printing

- By default, `print` puts an invisible *newline* character at the end of whatever it prints.
  - causes separate `prints` to print on different lines

- Example: What does this output?

```
for i in range(7):
    print(i * 5)
```

```
0
5
10
15
20
25
30
```

# Staying on the Same Line When Printing (cont.)

- To get separate `prints` to print on the same line, we can replace the newline with something else.

- Examples:

```
for i in range(7):
    print(i * 5, end=' ')
```

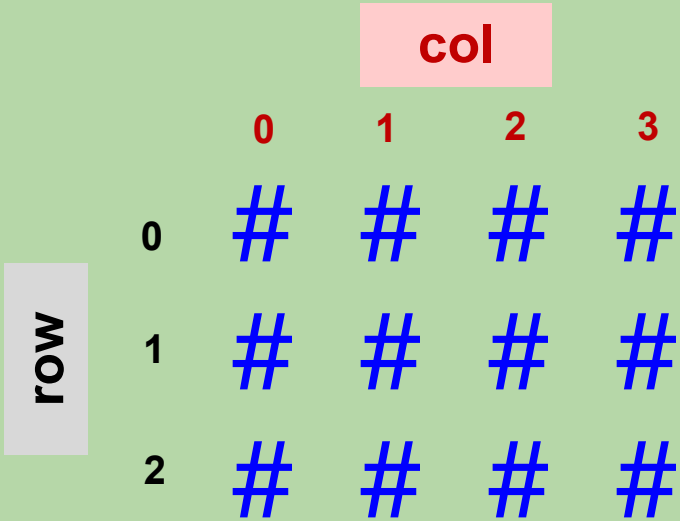
```
0 5 10 15 20 25 30
```

```
for i in range(7):
    print(i * 5, end=',')
```

```
0,5,10,15,20,25,30,
```

# Printing Patterns

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






48

# 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
```



49

# 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
```



50

# 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
```



51

# 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
```



52

# 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 | |



53

# 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 | |



54