4. FLOW CONTROL - LOOP

VICTOR LEE

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

ITERATION

LOOP

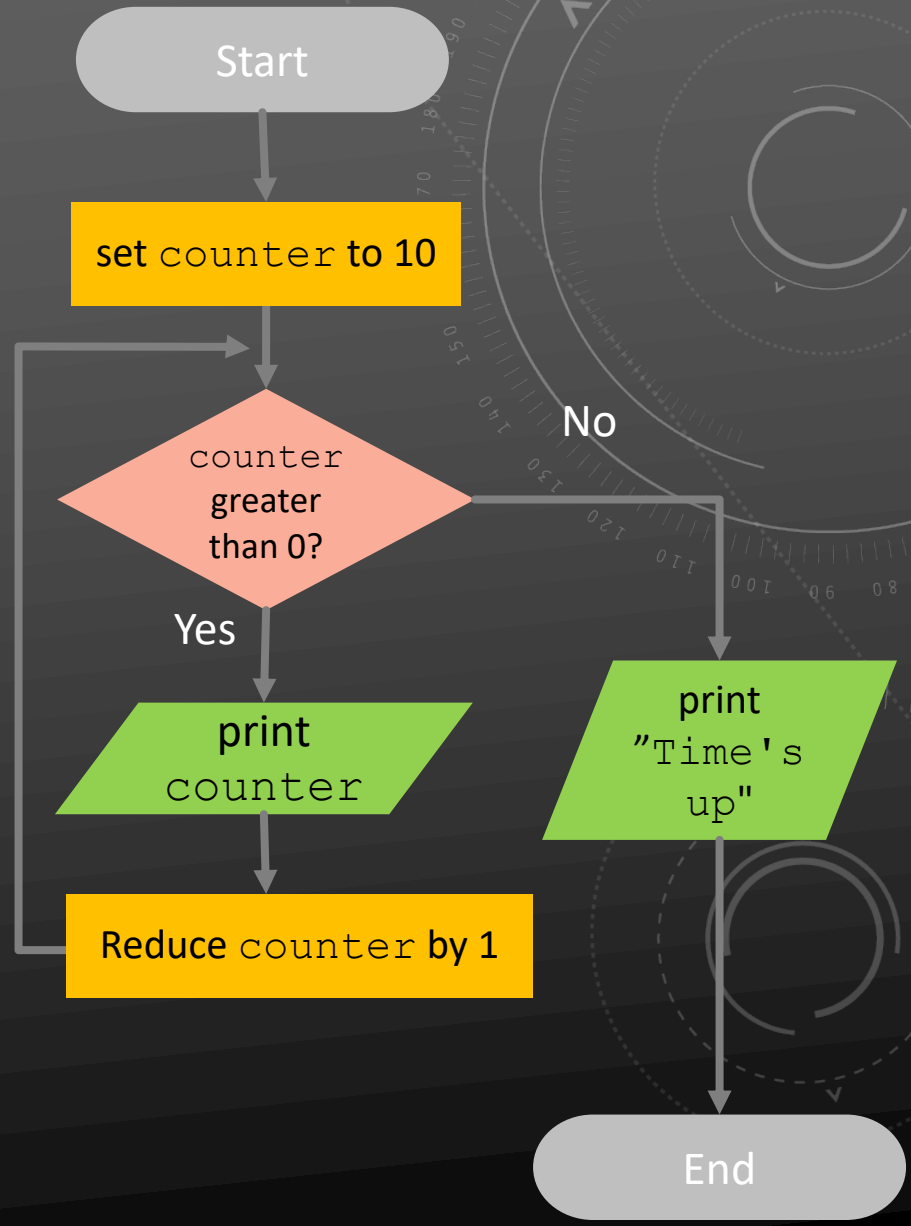
- Beside sequential execution and branch execution (`if` statement), looping is another common flow control in programming.
- When the execution enters a loop, it will execute a block of code repeatedly until the exit loop condition is met.
- Each repetition is called an iteration.

EXAMPLE: COUNTER

- A count-down program

```
counter=10
while counter>0:
    print(counter)
    counter-=1
print("Time's up")
```

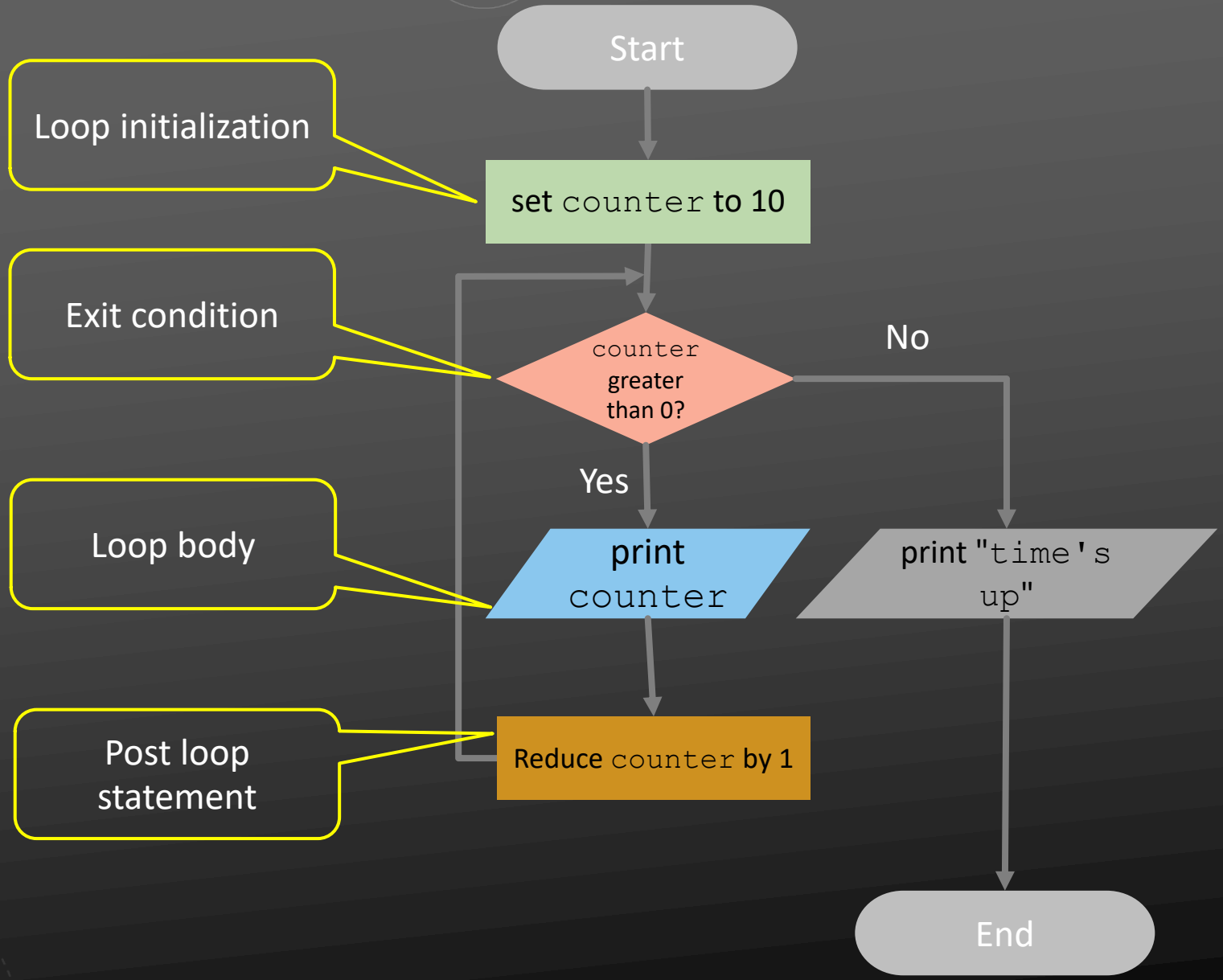
```
10
9
8
7
6
5
4
3
2
1
Time's up
```



What is the value of `counter` after the while loop?

LOOP

- In general, a program loop consists of:
  - Loop initialization (pre-loop)
  - Loop body
  - Exit condition
  - Post loop statements (stepping towards exit condition)


Start
Loop initialization
set counter to 10
Exit condition
counter greater than 0?
No
Yes
Loop body
print counter
print "time's up"
Post loop statement
Reduce counter by 1
End

TYPES OF LOOP

- `while` loop
- `for` loop

WHILE LOOP

- Loop until the given condition (logical expression) is no longer true
  - If the value of the logical expression is non-zero (true), the indented statements (code block) will be executed, otherwise, the loop terminates without executing the code block
  - After the code block is executed, the logical expression will be tested again

```
while logical_expression:
→statement
→statement
```

Code block

WHILE LOOP

- In the countdown program, the loop terminates when `counter` is zero or negative.
  - `counter` gets smaller in each iteration, so eventually it reaches 0

```
counter=10
while counter>0:
    print(counter)
    counter-=1
print("Time's up")
```

- In some other loops, it may not be obvious.
  - When `n` = 3, it prints 3, 10, 5, 16, 8, 4, 2
  - However, there is NO obvious proof that the loop terminates for **all** positive values of `n` (https://en.wikipedia.org/wiki/Collatz_conjecture)

```
n = 3
while n != 1:
    print(n)
    if n%2 == 0:        # n is even
        n = n / 2
    else:               # n is odd
        n = n*3 + 1
```

EXAMPLE: DETERMINE A NUMBER IS A PRIME OR NOT

DETERMINE A NUMBER IS A PRIME OR NOT

- Input: a number, `n`
- Output: `True` or `False`
- Process: divide `n` by 2, 3, 4, 5, ... , `n`/2

THE PROGRAM

```
n=int(input())
d=2;
isPrime=True

while d<n/2:
    if n%d==0:
        isPrime=False
    d+=1

if isPrime:
    print(str(n)+" is a prime number")
else:
    print(str(n)+" is NOT a prime number")
```

BETTER VERSION (STOP LOOPING WHEN IT IS NOT A PRIME)

```
n=int(input())
d=2;
isPrime=True

while d<n/2:
    if n%d==0:
        isPrime=False
        break;
    d+=1

if isPrime:
    print(str(n)+" is a prime number")
else:
    print(str(n)+" is NOT a prime number")
```

Do Things Right vs Do the Right Things?

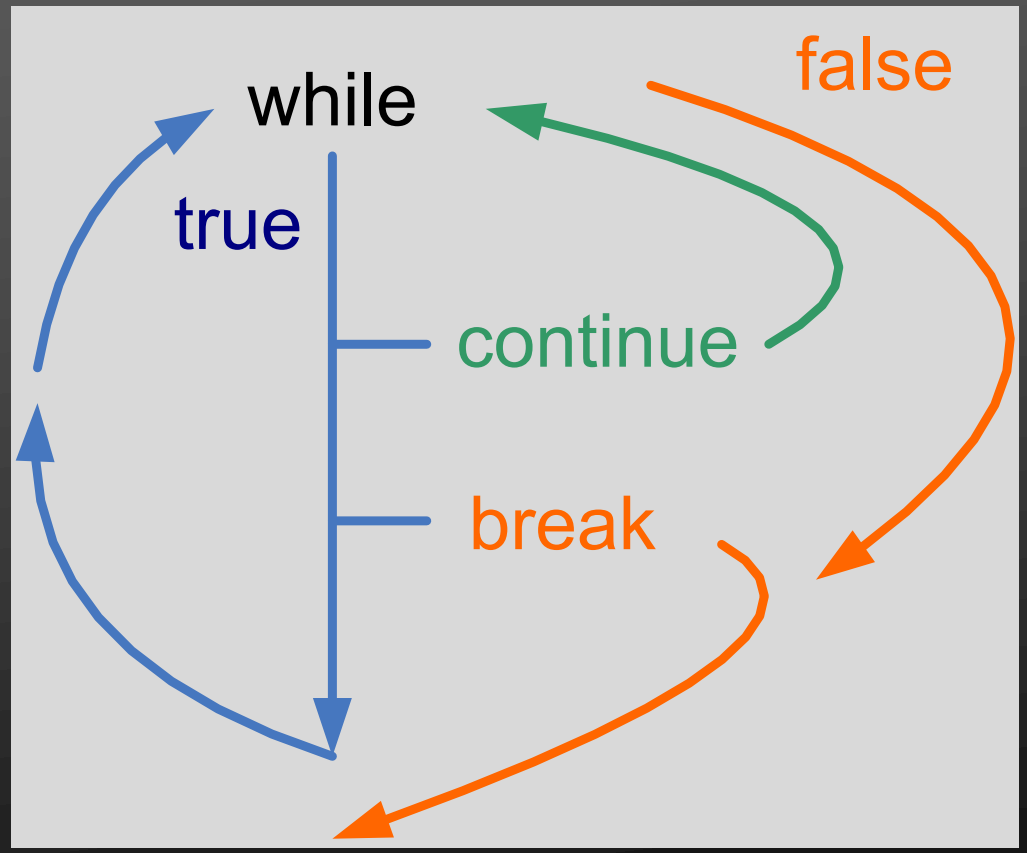
Did you see any bug  in this program?

CONTROL THE LOOP

- `break`
  - Jump out of the loop that contains the `break` statement.
  - Transfer control to the statement right after the loop.

- `continue`
  - Skip the rest of the code inside the loop that contains the `continue` statement for the **current iteration only**.

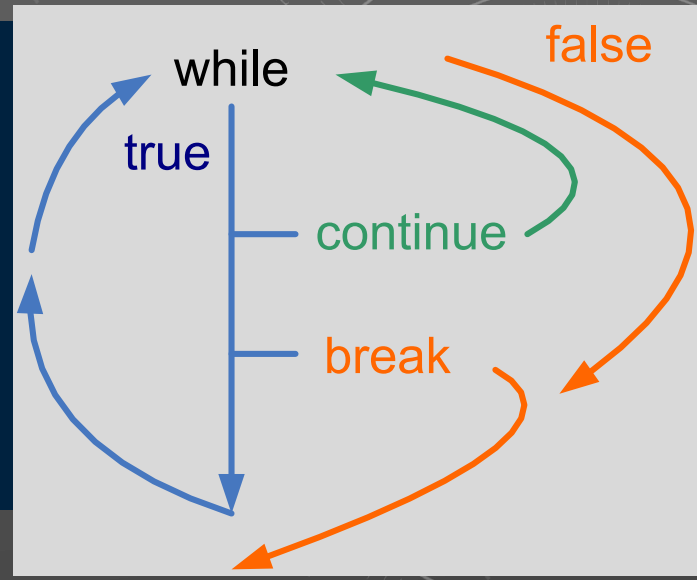
CONTINUE VS. BREAK



EXAMPLES USING THE COUNTDOWN PROGRAM

```
counter=10
while counter>0:
    print(counter)
    counter-=1
print("Time's up")
```

```
10
9
8
7
6
5
4
3
2
1
Time's up
```



- Countdown until '5' is hit versus countdown but skip '5'

```
10
9
8
7
6
Time's up
```

break_continue.py - /Users...

```
counter=10
while counter>0:
    if counter==5:
        counter-=1
        break
    print(counter)
    counter-=1
print("Time's up")
```

Ln: 9 Col: 0

break_continue.py - /Users...

```
counter=10
while counter>0:
    if counter==5:
        counter-=1
        continue
    print(counter)
    counter-=1
print("Time's up")
```

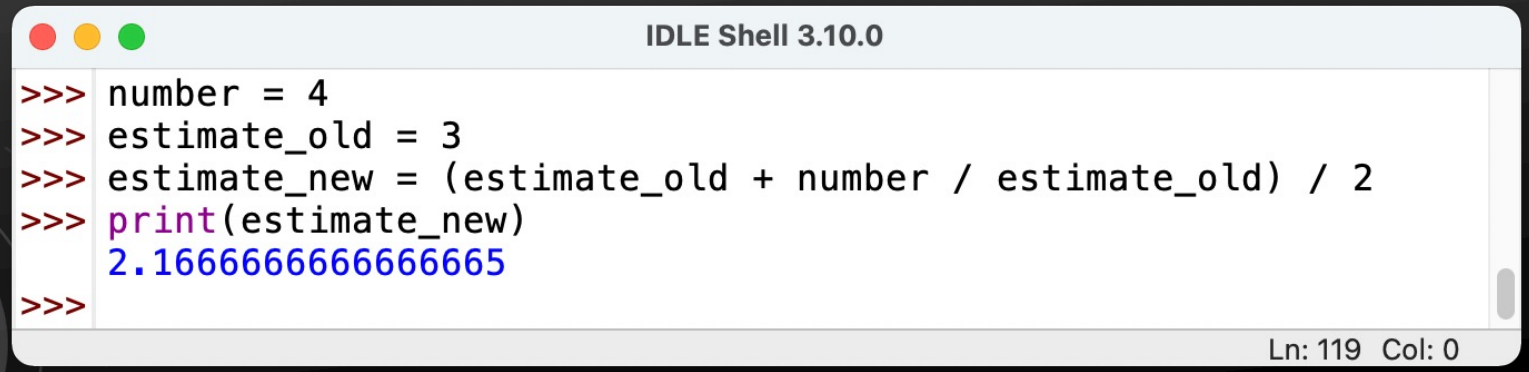
Ln: 8 Col: 19

```
10
9
8
7
6
4
3
2
1
Time's up
```

EXAMPLE: FIND THE SQUARE ROOT OF A NUMBER

- Input: a number and an estimate, e.g., number = 4, estimate = 3
- Output: (best estimate of) the square root of the number
- Process: use Newton's method to compute a better estimate with the following formula

$$estimate_{new} = \frac{estimate_{old} + {number}/{estimate_{old}}}{2}$$



EXAMPLE: FIND THE SQUARE ROOT OF A NUMBER

Use a **loop** to repeat applying the Newton's method to compute a better estimate **until** there is NO further improvement, i.e., the new estimate equals the old estimate.

THE PROGRAM

```
number = 4
estimate_old = 3
while True:
    print(estimate_old)
    estimate_new = (estimate_old + number / estimate_old) / 2
    if estimate_new == estimate_old:
        break
    estimate_old = estimate_new
```

FOR LOOP

- Designed to step through the item in **sequence object** such as a list of numbers
- Each item in the sequence object is assigned to `target` at each iteration
- The loop stops when all the items in the sequence object are visited
- Unlike while loop, which loops until the given condition is no longer true, for loop executes the code block a fixed number of times

```
for target in object:
→statement
→statement
```

Code block

BUILT-IN FUNCTION: RANGE

- Python provides a function `range` to generate a list of numbers
- `range(stop)`: generates a sequence of integers starting from 0 to (`stop` – 1),
  - `range(5) = [0, 1, 2, 3, 4]`
- `range(start, stop)`: generates a sequence of integers starting from (`start`) to (`stop` – 1)
  - `range(2, 5) = [2, 3, 4]`
- `range(start, stop, step)`: generates a sequence of integers, starting from (`start`) to (`stop` - 1), each different by `step`
  - `range (2, 8, 2) = [2, 4, 6]`

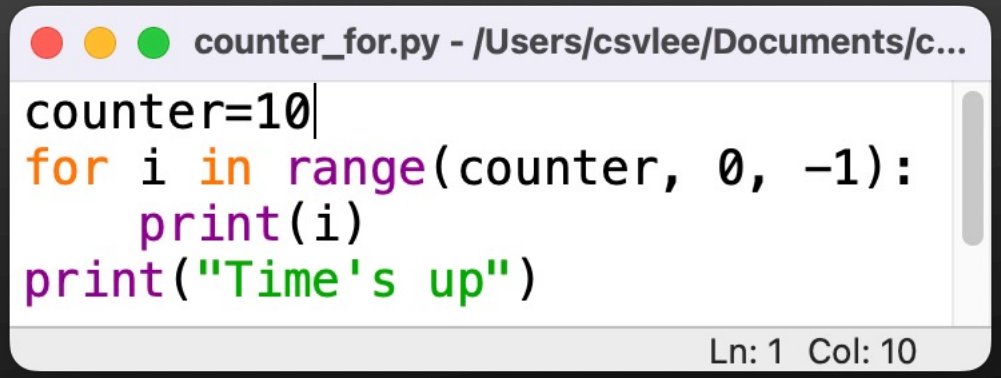
```
for i in range(8):
    print(i)
print("end of 1st loop")

for i in range(2, 8):
    print(i)
print("end of 2nd loop")

for i in range(2, 8, 2):
    print(i)
print("end of 3rd loop")
```

```
0
1
2
3
4
5
6
7
end of 1st loop
2
3
4
5
6
7
end of 2nd loop
2
4
6
end of 3rd loop
```

REWRITE THE COUNTDOWN PROGRAM USING FOR LOOP

- Hint:
  - `step` in function `range` can be negative

```
counter=10
while counter>0:
    print(counter)
    counter-=1
print("Time's up")
```




```
10
9
8
7
6
5
4
3
2
1
Time's up
```

MORE EXAMPLE

- `For` loop can be used to step through a string.
- Stretch a string using a `for` loop!

```
programming → p r o g r a m m i n g
```

stretch.py - /Users/csvlee/Documents/stre...

```
course_title='Programming'
for i in course_title:
    print(i, end=' ')
```

Ln: 4 Col: 0

Reminder: Using `print(i, end=' ')` can replace the newline character with space

ELSE STATEMENT IN LOOP

- In Python, `else` blocks may append to the `while` loop or `for` loop. The `else` blocks will **not** be executed only if the loop is terminated by `break` keyword

EXAMPLE

- Don't output "`Done`" only if the string is not completely stretched.

break_else.py - /Users/csvlee/Documents/break_else.py (3.11.1)

```
stop_letter=input("Which letter to stop? ")
course_title = 'Programming'
for i in course_title:
    if i == stop_letter:
        break;
    print(i, end=' ')
else:
    print()
    print("Done")
```

```
================== RESTART:
Which letter to stop? a
P r o g r

================== RESTART:
Which letter to stop? b
P r o g r a m m i n g
Done
```

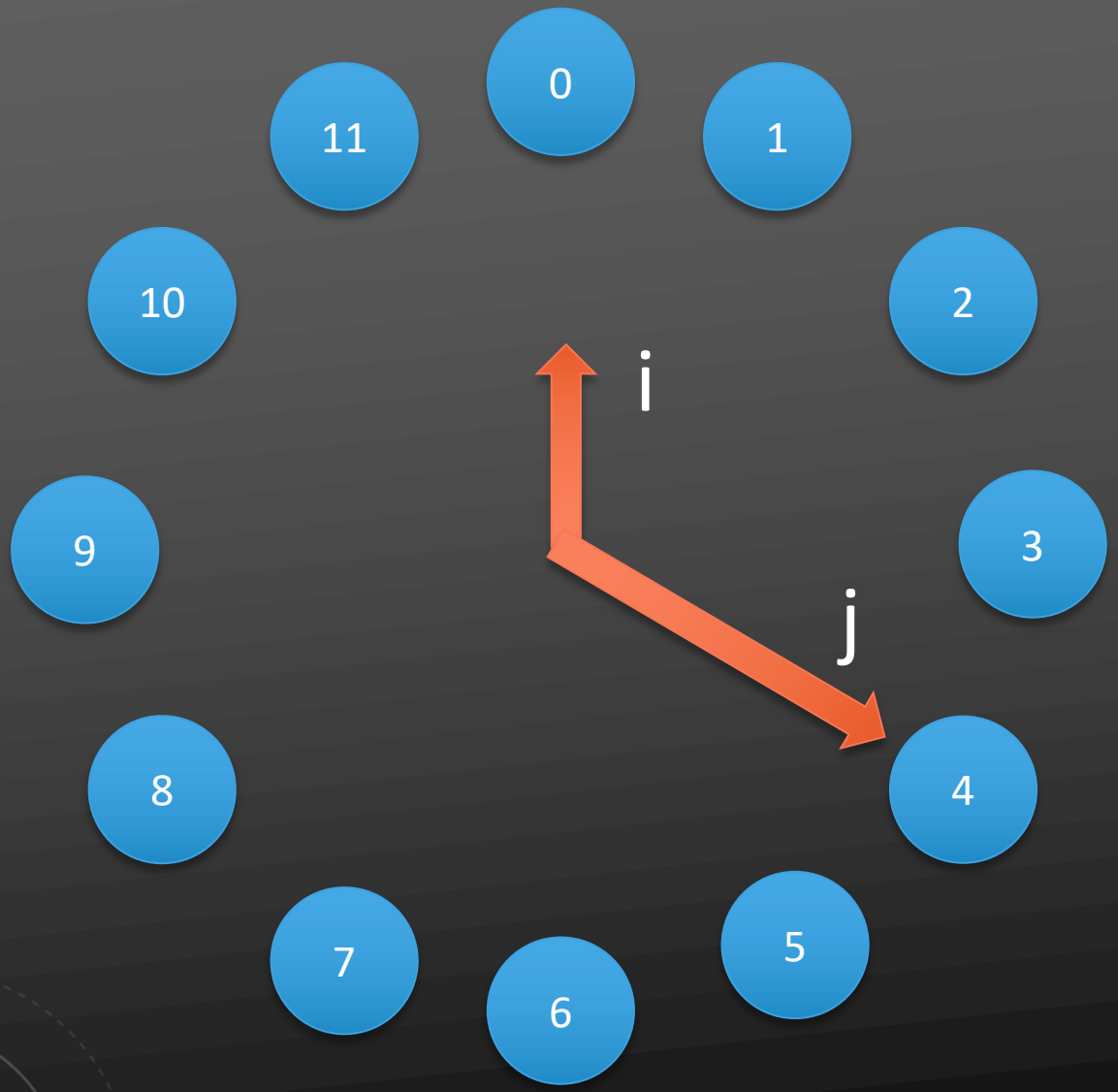
NESTED LOOP

- In some problems, we need more than one layer of looping to solve a problem
- Example: find all the prime numbers between 1 to 100

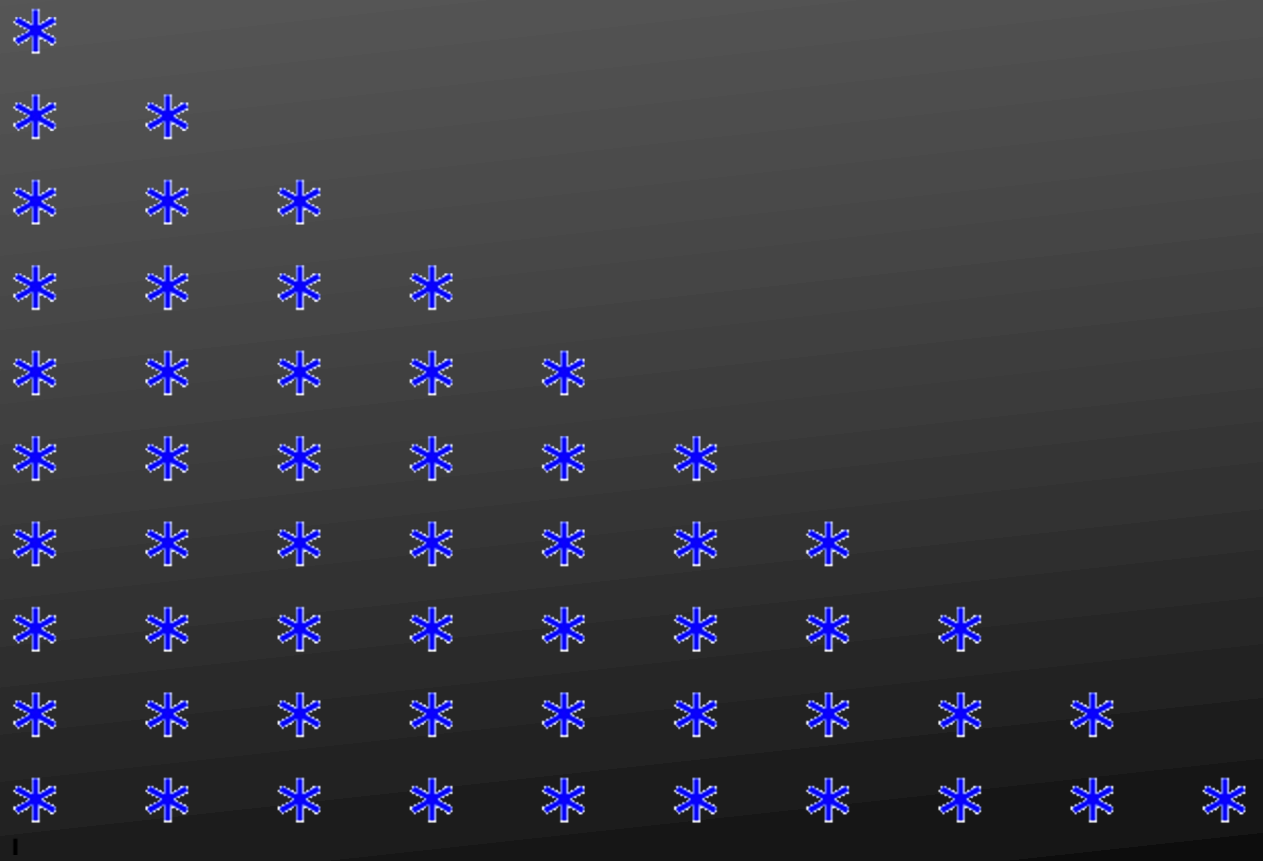
NESTED FOR-LOOP

```
for i in range(3):
    print("Outer for:")
    for j in range(2):
        print("Inner for:", end=" ")
        print("i="+str(i)+", j="+str(j))

# The outer loop is executed 3 times
# For each iteration of the outer loop,
# the inner loop is executed 2 times
```

```
Outer for:
Inner for: i=0, j=0
Inner for: i=0, j=1

Outer for:
Inner for: i=1, j=0
Inner for: i=1, j=1

Outer for:
Inner for: i=2, j=0
Inner for: i=2, j=1
```


0
1
2
3
4
5
6
7
8
9
10
11
i
j

NOW, YOU SHOULD KNOW HOW TO PRINT A RIGHT-ANGLED TRIANGLE



SUMMARY

- In Python, repeating task could be expressed with
  - `while`
  - `for`
- `break`, `continue` and `else` can be applied to both `while` and `for` loop
- A complete looping structure may consist of
  - Loop initialization
  - Loop body
  - Exit condition
  - Post loop statements
- Nested loop is a loop with more than one layer