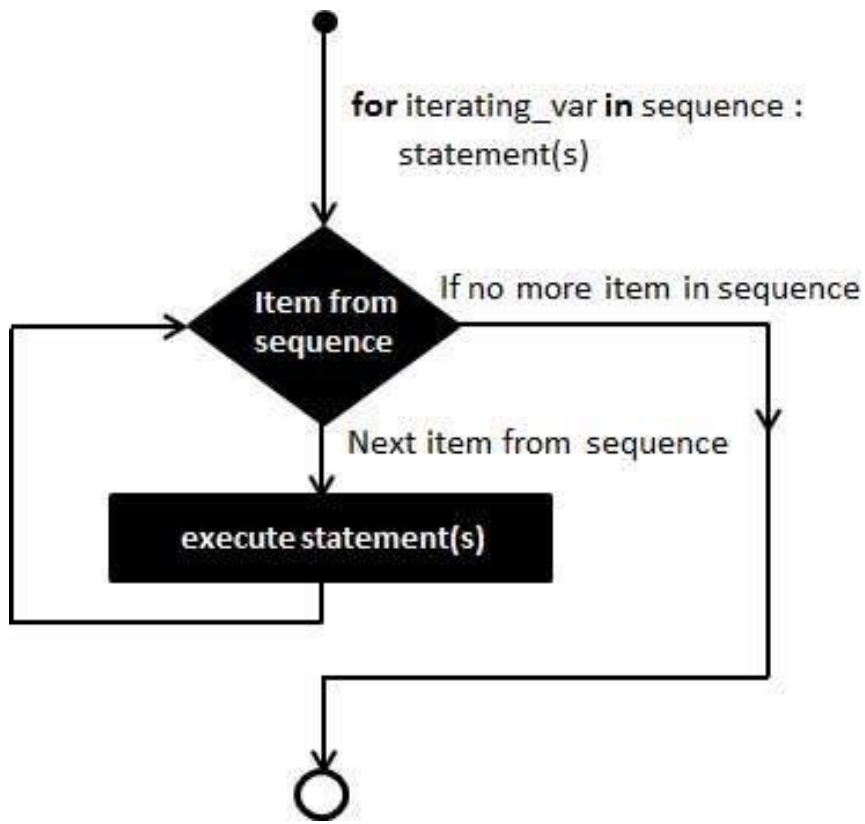
15-112
Fundamentals of Programming

Week 2 - Lecture 1:
Loops



January 19, 2016

Basic Building Blocks

Statements

Tells the computer to do something.

Data Types

Data is divided into different types.

Variables

Allows you to store data and access stored data.

Operators

Allows you to manipulate data.

Conditional Statements

Executes statements if a condition is satisfied.

Functions

Mini self-contained programs.

Basic Building Blocks

Statements
Tells the computer to do something.

Data Types
Data is divided into different types.

Variables
Allows you to store data and access stored data.

Operators
Allows you to manipulate data.

Conditional Statements
Executes statements if a condition is satisfied.

Functions
Mini self-contained programs.

Loops
Execute a block of code multiple times.

Loops give you wings !

My first ever program

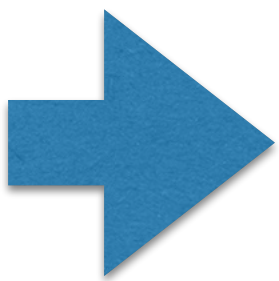
```
**********
*********
********
*******
******
*****
****
***
**
*
```

My first ever program

```
print(“**********”)
print(“**********”)
print(“**********”)
print(“**********”)
print(“*********”)
print(“********”)
print(“******”)
print(“*****”)
print(“****”)
print(“***”)
print(“**”)
print(“*”)
```

There is a better way!

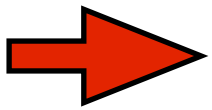
2 types of loops in Python



while loop

for loop

while loop



```
instruction1
while(expression):
    instruction2
    instruction3
instruction4
```

The code in the block should change something related to the *expression*.

iteration: a single execution of the instructions in the loop body.

while loop example

```
def getPositiveInteger():
    userInput = 0
    while(userInput <= 0):
        userInput = int(input("Enter a positive integer: "))
    return userInput
```

while loop example

```
x = 0

while(x < 5):
    print("Value of x is", x)
    x += 10
    print("This line will be printed!")

print("bye")
```

while loop

Repeating a block a certain number of times:

```
counter = 1

while(counter <= 10):
    instruction1
    instruction2
    counter += 1
```

But never use while loops to do this.
Use for loops.

while loop example

```
def countToN(n):
    counter = 1
    while(counter <= n):
        print(counter)
        counter += 1
```

while loop example

```
def sumToN(n):
    counter = 1
    total = 0
    while(counter <= n):
        total += counter
        counter += 1
    return total
```

while loop example

```
def sumFromMToN(m, n):
    counter = m
    total = 0
    while(counter <= n):
        total += counter
        counter += 1
    return total
```

Again: never use while loops to do this.
Use for loops.

Common Loop Bug 1

Off by 1 error

```
def sumToN(n):
    total = 0
    counter = 0
    while (counter <= n):
        counter += 1
        total += counter
    return total
```

Loop conditions that results in the loop body being executed either:

- 1 time too few
- 1 time too many

Manually check the first and last iterations!

Common Loop Bug 2

Infinite Loops

```
counter = 1
while (counter < 10):
    # Do some awesome complicated computation
    # ...
    # Then forget to increment counter
```

In the body you have to change something related to the *condition* being checked.

Example: leftmost digit

Write a function that
- takes an integer n as input,
- returns its leftmost digit.

e.g. 409283402013 should return 4

Idea:

Repeatedly get rid of rightmost digit until one digit is left.

```
def leftmostDigit(n):
    while (n >= 10):
        n = n // 10
    return n
```

Example: leftmost digit

Write a function that
- takes an integer n as input,
- returns its leftmost digit.

e.g. 409283402013 should return 4

Idea:

Repeatedly get rid of rightmost digit until one digit is left.

```
def leftmostDigit(n):
    n = abs(n)
    while (n >= 10):
        n = n // 10
    return n
```

Example: n'th Awesome number

A number m ≥ 0 is called "Awesome" if it is divisible by 3 or is divisible by 5.

Write a function that

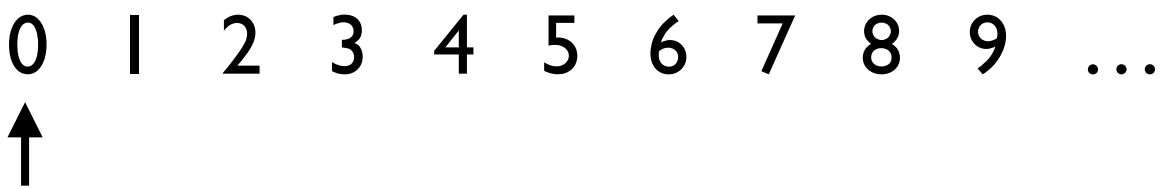
- takes a positive number n as input,
- returns the n'th Awesome number. (counting starts from 0)

Example: n'th Awesome number

A number m ≥ 0 is called "Awesome" if it is divisible by 3 or is divisible by 5.

Write a function that
- takes a positive number n as input,
- returns the n'th Awesome number. (counting starts from 0)

<u>Pictorial solution:</u>



n = 4

found = 1

is Awesome?

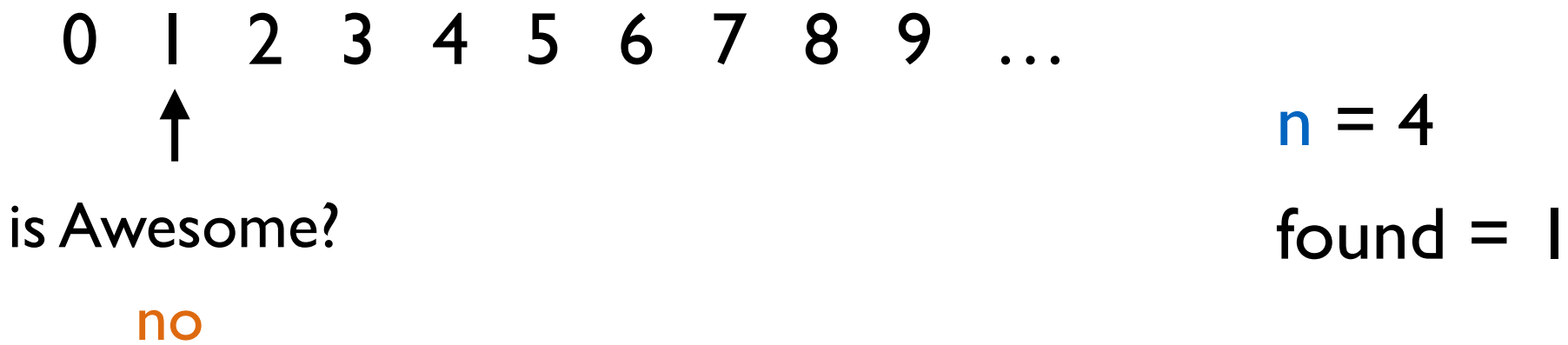
yes

Example: n'th Awesome number

A number m ≥ 0 is called "Awesome" if it is divisible by 3 or is divisible by 5.

Write a function that
- takes a positive number n as input,
- returns the n'th Awesome number. (counting starts from 0)

<u>Pictorial solution:</u>

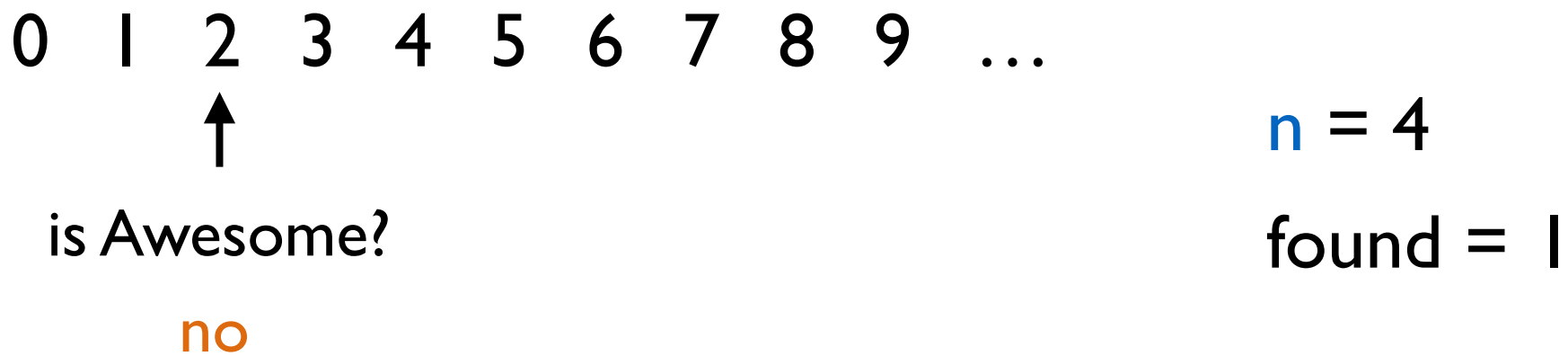


Example: n'th Awesome number

A number m ≥ 0 is called "Awesome" if it is divisible by 3 or is divisible by 5.

Write a function that
- takes a positive number n as input,
- returns the n'th Awesome number. (counting starts from 0)

<u>Pictorial solution:</u>

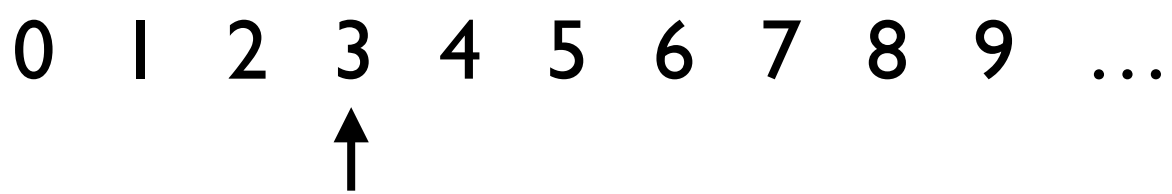


Example: n'th Awesome number

A number m ≥ 0 is called "Awesome" if it is divisible by 3 or is divisible by 5.

Write a function that
- takes a positive number n as input,
- returns the n'th Awesome number. (counting starts from 0)

<u>Pictorial solution:</u>



n = 4

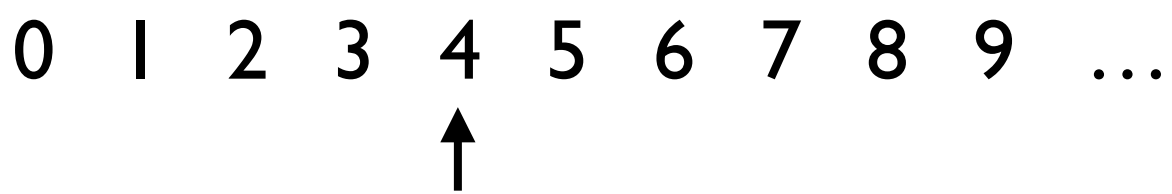
found = 2

Example: n'th Awesome number

A number m ≥ 0 is called "Awesome" if it is divisible by 3 or is divisible by 5.

Write a function that
- takes a positive number n as input,
- returns the n'th Awesome number. (counting starts from 0)

<u>Pictorial solution:</u>



is Awesome?

no

n = 4

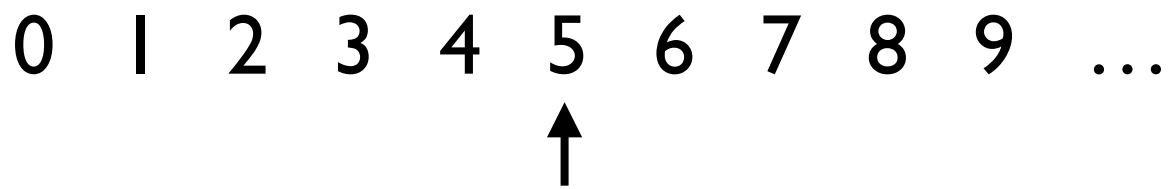
found = 2

Example: n'th Awesome number

A number m ≥ 0 is called "Awesome" if it is divisible by 3 or is divisible by 5.

Write a function that
- takes a positive number n as input,
- returns the n'th Awesome number. (counting starts from 0)

<u>Pictorial solution:</u>



is Awesome?

yes

n = 4

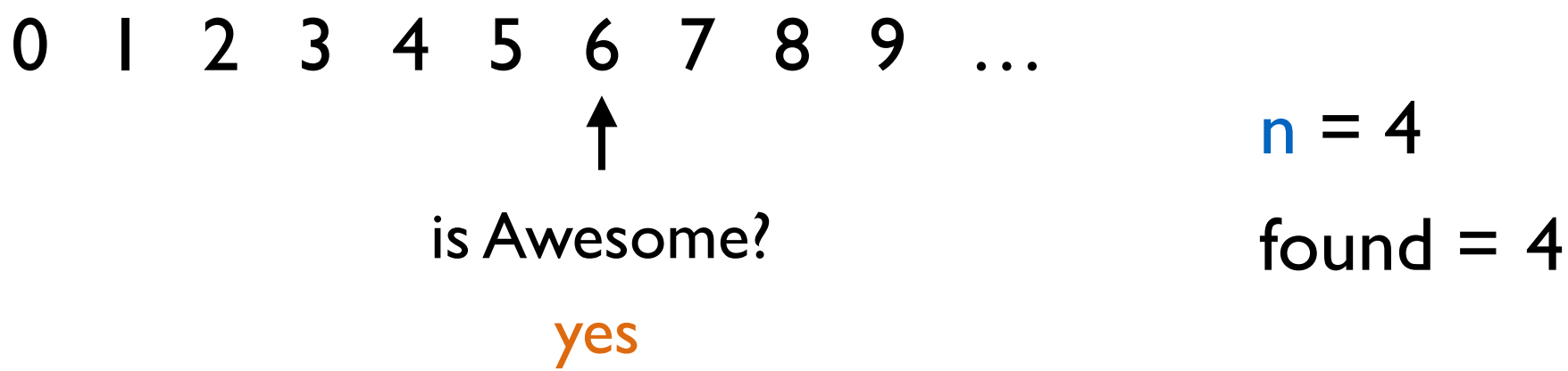
found = 3

Example: n'th Awesome number

A number m ≥ 0 is called "Awesome" if it is divisible by 3 or is divisible by 5.

Write a function that
- takes a positive number n as input,
- returns the n'th Awesome number. (counting starts from 0)

<u>Pictorial solution:</u>

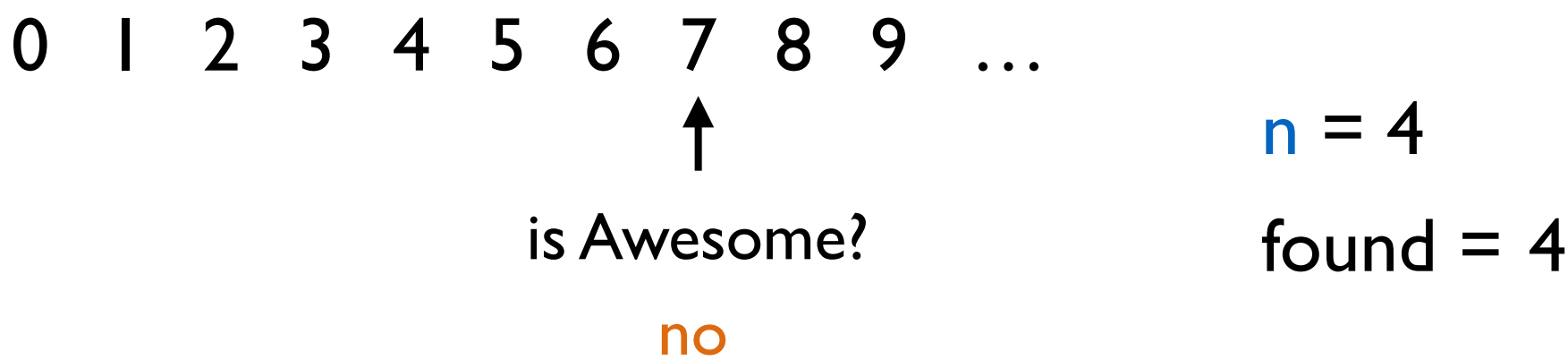


Example: n'th Awesome number

A number m ≥ 0 is called "Awesome" if it is divisible by 3 or is divisible by 5.

Write a function that
- takes a positive number n as input,
- returns the n'th Awesome number. (counting starts from 0)

<u>Pictorial solution:</u>

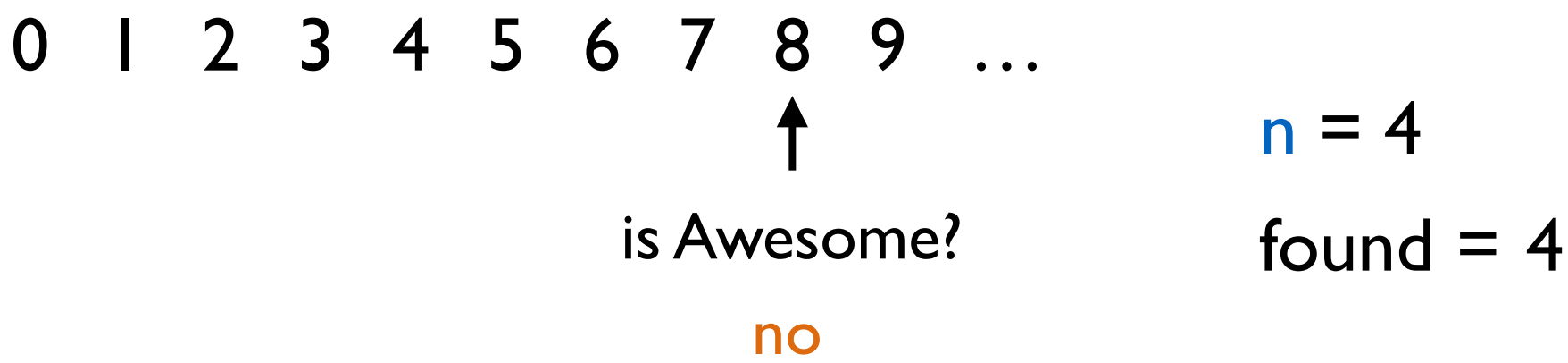


Example: n'th Awesome number

A number m ≥ 0 is called "Awesome" if it is divisible by 3 or is divisible by 5.

Write a function that
- takes a positive number n as input,
- returns the n'th Awesome number. (counting starts from 0)

<u>Pictorial solution:</u>

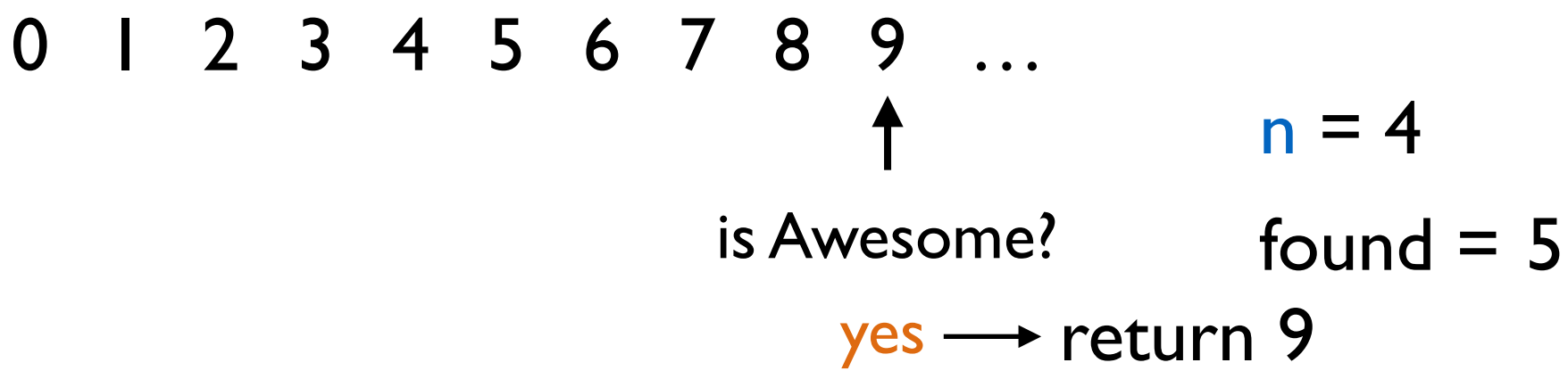


Example: n'th Awesome number

A number m ≥ 0 is called "Awesome" if it is divisible by 3 or is divisible by 5.

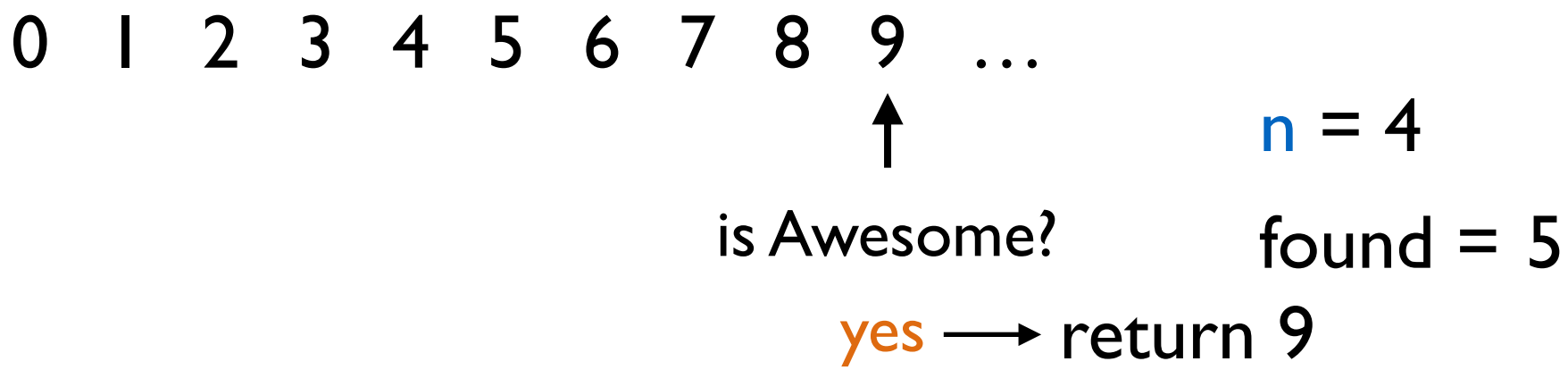
Write a function that
- takes a positive number n as input,
- returns the n'th Awesome number. (counting starts from 0)

<u>Pictorial solution:</u>



Example: n'th Awesome number

Pictorial solution:



Algorithm:

- Let found = 0
- Go through every number 0, 1, 2, 3, ... :
  - if the number is Awesome, increment found
- When found > n, return corresponding number

Example: n'th Awesome number

- Let found = 0
- Go through every number 0, 1, 2, 3, ... :
  - if the number is Awesome, increment found
- When found > n, return corresponding number

```
def nthAwesome(n):
```

Example: n'th Awesome number

- Let found = 0
- Go through every number 0, 1, 2, 3, ... :
  - if the number is Awesome, increment found
- When found > n, return corresponding number

```
def nthAwesome(n):
    found = 0
```

Example: n'th Awesome number

- Let found = 0
- Go through every number 0, 1, 2, 3, ... :
  - if the number is Awesome, increment found
- When found > n, return corresponding number

```
def nthAwesome(n):
    found = 0
    guess = 0
```

Example: n'th Awesome number

- Let found = 0
- Go through every number 0, 1, 2, 3, ... :
  - if the number is Awesome, increment found
- When found > n, return corresponding number

```
def nthAwesome(n):
    found = 0
    guess = -1
```

Example: n'th Awesome number

- Let found = 0
- Go through every number 0, 1, 2, 3, ... :
  - if the number is Awesome, increment found
- When found > n, return corresponding number

```
def nthAwesome(n):
    found = 0
    guess = -1
    while (found <= n):
```

Example: n'th Awesome number

- Let found = 0
- Go through every number 0, 1, 2, 3, ... :
  - if the number is Awesome, increment found
- When found > n, return corresponding number

```
def nthAwesome(n):
    found = 0
    guess = -1
    while (found <= n):
        guess += 1
```

Example: n'th Awesome number

- Let found = 0
- Go through every number 0, 1, 2, 3, ... :
  - if the number is Awesome, increment found
- When found > n, return corresponding number

```
def nthAwesome(n):
    found = 0
    guess = -1
    while (found <= n):
        guess += 1
        if (isAwesome(guess)):
```

Example: n'th Awesome number

- Let found = 0
- Go through every number 0, 1, 2, 3, ... :
  - if the number is Awesome, increment found
- When found > n, return corresponding number

```
def nthAwesome(n):
    found = 0
    guess = -1
    while (found <= n):
        guess += 1
        if (isAwesome(guess)):
            found += 1
```

Example: n'th Awesome number

- Let found = 0
- Go through every number 0, 1, 2, 3, ... :
  - if the number is Awesome, increment found
- When found > n, return corresponding number

```
def nthAwesome(n):
    found = 0
    guess = -1
    while (found <= n):
        guess += 1
        if (isAwesome(guess)):
            found += 1
    return guess
```

Example: n'th Awesome number

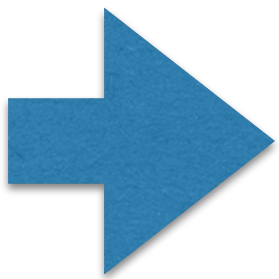
```
def nthAwesome(n):
    found = 0
    guess = -1
    while (found <= n):
        guess += 1
        if (isAwesome(guess)):
            found += 1
    return guess
```

```
def nthAwesome(n):
    found = 0
    guess = 0
    while (found <= n):
        if (isAwesome(guess)):
            found += 1
        guess += 1
    return guess - 1
```

```
def isAwesome(m):
    return ((m % 3) == 0) or ((m % 5) == 0)
```

2 types of loops in Python

while loop



for loop

for loop

for *var-name* **in** *sequence*:
    loop-body

```
for x in [1, 2, 3, 4, 5]:
    print(x)
```

list (a data type in Python)

1st iteration: x = 1
2nd iteration: x = 2
3rd iteration: x = 3
4th iteration: x = 4
5th iteration: x = 5

for loop

for *var-name* **in** *sequence*:
    loop-body

```
for x in "Hello":
    print(x)
```

A string is a sequence too

1st iteration: x = "H"
2nd iteration: x = "e"
3rd iteration: x = "l"
4th iteration: x = "l"
5th iteration: x = "o"

for loop

for *var-name* **in** *sequence*:
    loop-body

range(n) ≈ [0, 1, 2, …, n-1]

```
for x in [0, 1, 2, 3, 4]:
    print(x)
```

```
for x in range(5):
    print(x)
```

for loop

for *var-name* **in** *sequence*:
    loop-body

```
def sumToN(n):
    total = 0
    for x in range(n+1):
        total += x
    return total
```

```
def sumToN(n):
    total = 0
    x = 0
    while (x <= n):
        total += x
        x += 1
    return total
```

For loop is the right choice here!

for loop

for *var-name* **in** *sequence*:
    loop-body

range(m, n) ≈ [m, m+1, m+2, …, n-1]

```
def sumFromMToN(m, n):
    total = 0
    for x in range(m, n+1):
        total += x
    return total
```

for loop

for *var-name* **in** *sequence*:
    loop-body

range(m, n, k) ≈ [m, m+k, m+2k, …]

```
def sumEveryKthFromMToN(m, n, k):
    total = 0
    for x in range(m, n+1, k):
        total += x
    return total
```

for loop

```
def sumOfOddsFromMToN(m, n):
    total = 0
    for x in range(m, n+1):
        if (x % 2 == 1):
            total += x
    return total
```

for loop

```
def sumOfOddsFromMToN(m, n):
    total = 0
    for x in range(m, n+1):
        if (x % 2 == 1):
            total += x
    return total
```

```
def sumOfOddsFromMToN(m, n):
    if (m % 2 == 0): m += 1
    total = 0
    for x in range(m, n+1, 2):
        total += x
    return total
```

for loop

```
def sumOfOddsFromMToN(m, n):
    if (n % 2 == 0): n -= 1
    total = 0
    for x in range(n, m-1, -2):
        total += x
    return total
```

Unclear code!!!

for loop vs while loop

```
for x in range(10):
    print(x)
```

```
x = 0
while(x < 10):
    print(x)
    x += 1
```

Use while loop when the number of iterations is *indefinite*.

e.g. continue to do something until a certain event

Exercise: Testing primality

Write a function that:

- Gets a positive integer input
- Returns True if the integer is prime
- Returns False otherwise

prime:

- greater than 1,
- is only divisible by 1 and itself

Exercise: Testing primality

Algorithm:

- Let n denote the input number.
- Go through every number from 2 to n-1.
- If one of these numbers divides n, then n is not prime.
- Otherwise, n is prime.

Exercise: Testing primality

Algorithm:

- Let n denote the input number.
- Go through every number from 2 to n-1.
- If one of these numbers divides n, then n is not prime.
- Otherwise, n is prime.

Exercise: Testing primality

- Let n denote the input number.
- Go through every number from 2 to n-1.
- If one of these numbers divides n, then n is not prime.
- Otherwise, n is prime.

```
def isPrime(n):
    for possibleFactor in range(2, n):
        # Check if possibleFactor divides n
```

Exercise: Testing primality

- Let n denote the input number.
- Go through every number from 2 to n-1.
- If one of these numbers divides n, then n is not prime.
- Otherwise, n is prime.

```
def isPrime(n):
    for possibleFactor in range(2, n):
        if (n % possibleFactor == 0): return False
```

Exercise: Testing primality

- Let n denote the input number.
- Go through every number from 2 to n-1.
- If one of these numbers divides n, then n is not prime.
- Otherwise, n is prime.

```
def isPrime(n):
    for possibleFactor in range(2, n):
        if (n % possibleFactor == 0): return False
    return True
```

Exercise: Testing primality

- Let n denote the input number.
- Go through every number from 2 to n-1.
- If one of these numbers divides n, then n is not prime.
- Otherwise, n is prime.

```
def isPrime(n):
    if (n < 2): return False
    for possibleFactor in range(2, n):
        if (n % possibleFactor == 0): return False
    return True
```

Start thinking about running time

```
def isPrime(n):
    if (n < 2): return False
    for x in range(2, n):
        if(n % x == 0): return False
    return True
```

How many iterations?

In the worst case?
(worst possible n)

~ n

What if the input is

```
203703597633448608626844568840937816105146839366593625063614044935438129976333670618339737 1
```

(length of the input = number of digits = 90 ~ log n)

Start thinking about running time

```
def fasterIsPrime(n):
    if (n < 2): return False
    maxFactor = round(n**0.5)
    for x in range(2, maxFactor + 1):
        if(n % x == 0): return False
    return True
```

How many iterations?

In the worst case?
(worst possible n)

~ n**0.5

Start thinking about running time

```
def fasterIsPrime(n):
    if (n < 2): return False
    if (n == 2): return True
    if (n % 2 == 0): return False
    maxFactor = round(n**0.5)
    for x in range(3, maxFactor + 1, 2):
        if (n % x == 0): return False
    return True
```

How many iterations?

In the worst case?
(worst possible n)

~ n**0.5 / 2

Example: Find the n'th prime

Write a program that:

- Gets a positive integer n as input
- Returns the n'th prime number

- Let found = 0
- Go through every number 2, 3, 4, 5, ... :
  - if the number is prime, increment found
- When found > n, return the corresponding prime

Remember: We start counting from 0.

Example: Find the n'th prime

- Let found = 0
- Go through every number 2, 3, 4, 5, ... :
  - if the number is prime, increment found
- When found > n, return the corresponding prime

```
def nthPrime(n):
    found = 0
    guess = 0
    while (found <= n):
        guess += 1
        if (isPrime(guess)):
            found += 1
    return guess
```

Need to use while loop

Example: The factoring problem

Write a function that:

- gets a positive integer as input
- returns the smallest *factor* ≠ 1

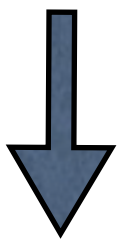
factor: divides the integer with no remainder.

Exercise

Example: The factoring problem

Why you should care about this problem:

If there is an efficient program to solve
the factoring problem



can break most cryptographic systems
used on the internet!

break

continue

break

Break out of the loop

```
def countToN(n):
    counter = 1
    while(True):
        print(counter)
        if(counter == n):
            break
        counter += 1
```

once this is executed, you leave the loop body

break

In a while loop, condition is checked at the beginning

```
while(expression):
    …
```

Using a break statement, can check condition anywhere

```
while(True):
    …
    if(expression):
        break
    …
```

break

Break out of the loop

```
def sumGivenNumbers():
    total = 0
    while(True):
        x = input("Enter number (or 'done' to quit): ")
        if(x == "done"):
            break
        else:
            total += int(x)
    return total

print(sumGivenNumbers())
```

continue

Break out of the current iteration

```
def sumOfOddsToN(n):
    total = 0
    for x in range(1, n+1):
        if(x % 2 == 0):
            continue
        total += x
    return total
```

once this is executed,
you skip to the next iteration

continue

Break out of the current iteration

```
def multiplyGivenNumbers():
    # if 0 is given as input, we ignore it
    product = 1
    while(True):
        x = input("Enter number (or 'done' to quit): ")
        if(x == "done"):
            break
        elif(int(x) == 0):
            continue
        product *= int(x)
    return product

print(multiplyGivenNumbers())
```

Nested Loops

Nested loops

Many situations require one loop inside another loop.

```
for y in range(10):
    for x in range(8):
        # Body of the nested loop
```

Nested loops

Many situations require one loop inside another loop.

```
for y in range(10):
    for x in range(8):
        print("Hello")
```

How many times will "Hello" get printed?

Nested loops

Many situations require one loop inside another loop.

```
for y in range(4):
    for x in range(y):
        print("Hello")
```

y	# iterations of inner loop
0	0
1	1
2	2
3	3

How many times will "Hello" get printed?

Example: Draw a rectangle

Write a function that:

- Gets two integers, height and width as input
- Prints a rectangle with those dimensions

height = 4, width = 3

```
* * *
* * *
* * *
* * *
```

Repeat 4 times:

- Print a row (3 stars)

Example: Draw a rectangle

Write a function that:

- Gets two integers, height and width as input
- Prints a rectangle with those dimensions

height = 4, width = 3

```
* * *
* * *
* * *
* * *
```

Repeat 4 times:
    Repeat 3 times:
        - Print a star
    Skip a line

Example: Draw a rectangle

Write a function that:

- Gets two integers, height and width as input
- Prints a rectangle with those dimensions

height = 4, width = 3

```
* * *
* * *
* * *
* * *
```

```
for row in range(4):
    for col in range(3):
        print("*", end=" ")
    print()
```

Example: Draw a rectangle

Write a function that:

- Gets two integers, height and width as input
- Prints a rectangle with those dimensions

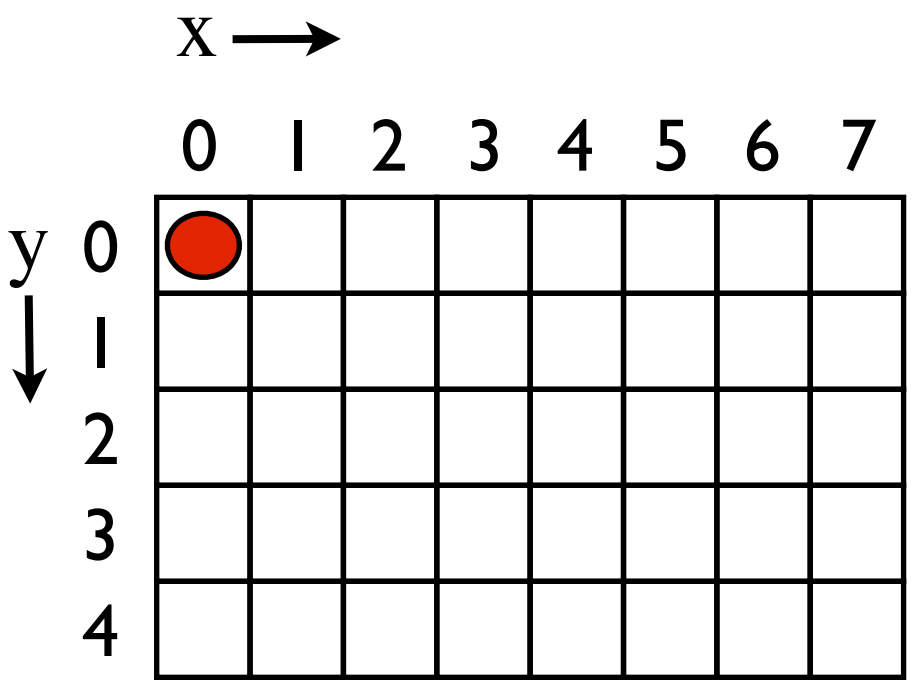
height = 4, width = 3

```
* * *
* * *
* * *
* * *
```

```
def printRectangle(height, width):
    for row in range(height):
        for col in range(width):
            print("*", end= " ")
        print()
```

Nested loops

```
for y in range(5):
    for x in range(8):
        # Body of the nested loop
```



Example

```
for y in range(4):
    for x in range(5):
        print("(", x , ",", y, ")", end=" ")
    print()
```

x →

y ↓

```
(0,0) (1,0) (2,0) (3,0) (4,0)
(0,1) (1,1) (2,1) (3,1) (4,1)
(0,2) (1,2) (2,2) (3,2) (4,2)
(0,3) (1,3) (2,3) (3,3) (4,3)
```

Example

```
for y in range(4):
    for x in range(y):
        print("(", x , ",", y, ")", end=" ")
    print()
```

```
\n
( 0 , 1 )
( 0 , 2 ) ( 1 , 2 )
( 0 , 3 ) ( 1 , 3 ) ( 2 , 3 )
```

Example

```
for y in range(1, 10):
    for x in range(1, 10):
        print(y*x, end=“ ”)
    print()
```

Multiplication table

```
for y in range(1, 10):
    for x in range(1, 10):
        print(y*x, end=" ")
    print()
```

```
1  2  3  4  5  6  7  8  9
2  4  6  8  10  12  14  16  18
3  6  9  12  15  18  21  24  27
4  8  12  16  20  24  28  32  36
5  10  15  20  25  30  35  40  45
6  12  18  24  30  36  42  48  54
7  14  21  28  35  42  49  56  63
8  16  24  32  40  48  56  64  72
9  18  27  36  45  54  63  72  81
```

A trick to get rid of nested loops

Write a function for the inner loop.

Example: Write a function that:

- Gets an integer height as input
- Prints a right-angled triangle of that height

height = 5

```
*****
****
***
**
*
```

```
def printStars(n):
    for x in range(n):
        print("*", end="")

def printTriangle(height):
    for x in range(height):
        printStars(  ?  )
    print()
```

A trick to get rid of nested loops

Write a function for the inner loop.

Example: Write a function that:

- Gets an integer height as input
- Prints a right-angled triangle of that height

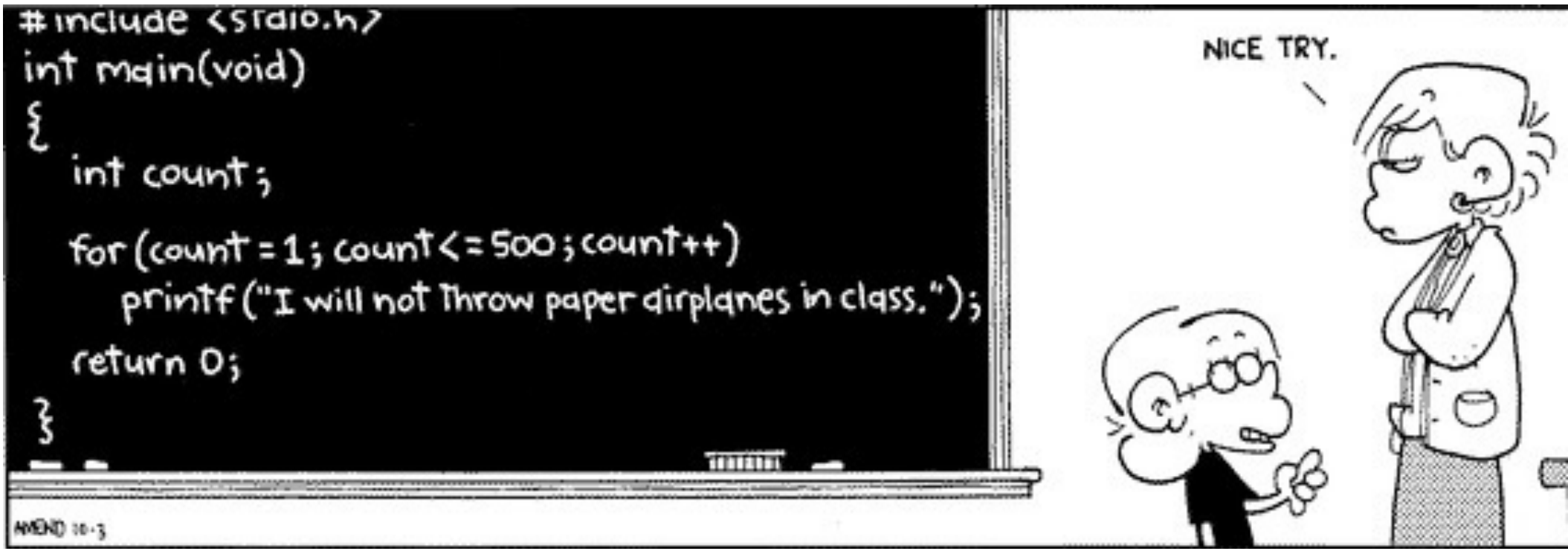
height = 5

```
*****
****
***
**
*
```

```
def printStars(n):
    for x in range(n):
        print("*", end="")

def printTriangle(height):
    for x in range(height):
        printStars(height - x)
    print()
```


WILL NOT SELL MIRACLE CURES
WILL NOT SELL MIRACLE CURES
WILL NOT SELL MIRACLE CURES
WILL NOT SELL MIRACLE CURES
WILL NOT SELL MIRACLE CUR
WILL NOT SELL MIRACLE CUR
WILL NOT SELL MIRACLE CU
WILL NOT SELL MIRACLE C
WILL NOT SELL MIRACLE C
WILL NOT SELL MIR


#include <stdio.h>
int main(void)
{
int count;
for (count = 1; count <= 500; count++)
printf ("I will not throw paper airplanes in class.");
return 0;
}
NICE TRY.
AMEND 10-3