Algorithmic Thinking: Loops and Conditionals

Announcements

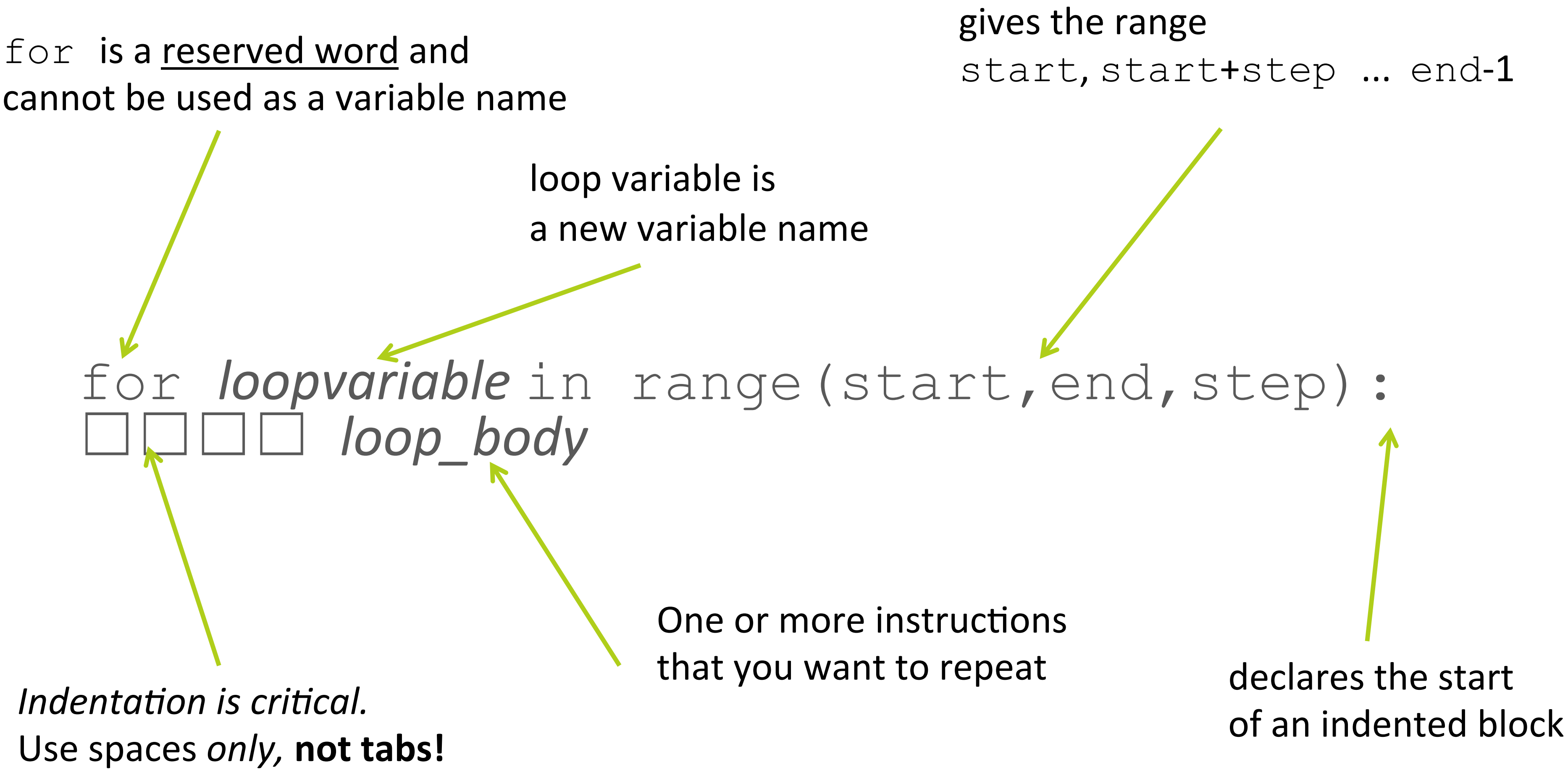
- Programming Assignment 2 due tonight (July 8th) at 11:59 via GradeScope
- Review lab today!
- Tomorrow, Wednesday:
  - OLI: Putting it Together due July 9th, 11:59PM
  - Lab 3
  - Programming Assignment 3 due July 9th, 11:59PM

Today

- Review from last time
  - A control flow structure
    - `for` loop
    - `While` loop
  - Nesting control structures
- The notion of an algorithm
- Moving from algorithm to code
- Python control structures: Conditionals

Review from last time

For Loop Syntax



`for` is a reserved word and cannot be used as a variable name

loop variable is a new variable name

gives the range `start`, `start+step` … `end`-1

```
for loopvariable in range(start,end,step):
□□□□ loop_body
```

Indentation is critical.
Use spaces *only*, **not tabs!**

One or more instructions that you want to repeat

declares the start of an indented block

Iteration with for loops

```
def test1():
    for i in range(1,6):
        print("Woof")
```

Iteration with for loops

```
def test1():
    for i in range(1,6):
        print("Woof")
```

```
>>> test1()
Woof
Woof
Woof
Woof
Woof
```

Iteration with for loops

```
def test1():
    for i in range(1,6):
        print("Woof")
```

```
>>> test1()
Woof
Woof
Woof
Woof
Woof
```

What determines how many times “Woof” is printed is **the number of elements in the range.**

Any expression that gives 5 elements in the range would give the same output.

For example, range(5), range(0,5), …

Iteration with for loops

```
def test2():
    for i in range(3,13,2):
        print(i)
```

Iteration with for loops

```
def test2():
    for i in range(3,13,2):
        print(i)
```

```
>>> test2()
3
5
7
9
11
```

Iteration with for loops

```
def test2():
    for i in range(3,13,2):
        print(i)
```

```
>>> test2()
3
5
7
9
11
```

```
Range(7)                ?
range(0, 7)             ?


range(1, 10, 2)    ?
range(2, 10, 2)    ?


range(10, 1, -1)   ?
range(10, 1, 2)   ?
```

Iteration with for loops

```
def test3():
    print("Woof" * 3)
```

Iteration with for loops

```
def test3():
    print("Woof" * 3)
```

This expression creates a string that concatenates 3 number of "Woof"s.

```
>>> test3()
WoofWoofWoof
```

Analogy:
3 * 4 is equivalent to 4+4+4
3 * "a" is equivalent to
"a" + "a" + "a"

Iteration with for loops

```
def test4():
    for i in range(1,6):
        print("Woof" * i)
```

Iteration with for loops

```
def test4():
    for i in range(1,6):
        print("Woof" * i)
```

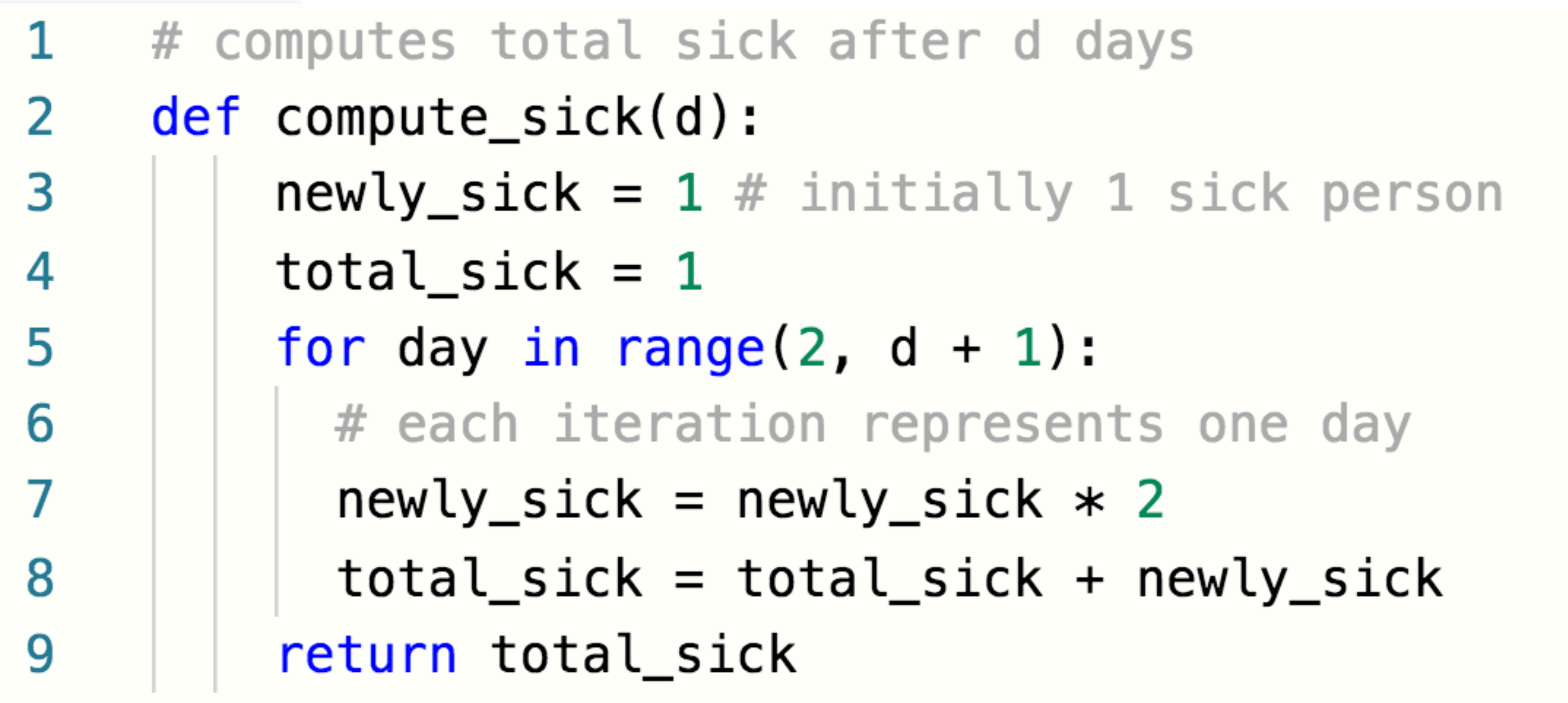
This expression creates a string that concatenates *i* number of "Woof"s.

```
>>> test4()
Woof
WoofWoof
WoofWoofWoof
WoofWoofWoofWoof
WoofWoofWoofWoofWoof
```

Analogy:
3 * 4 is equivalent to 4+4+4
3 * "a" is equivalent to
    "a" + "a" + "a"

An epidemic

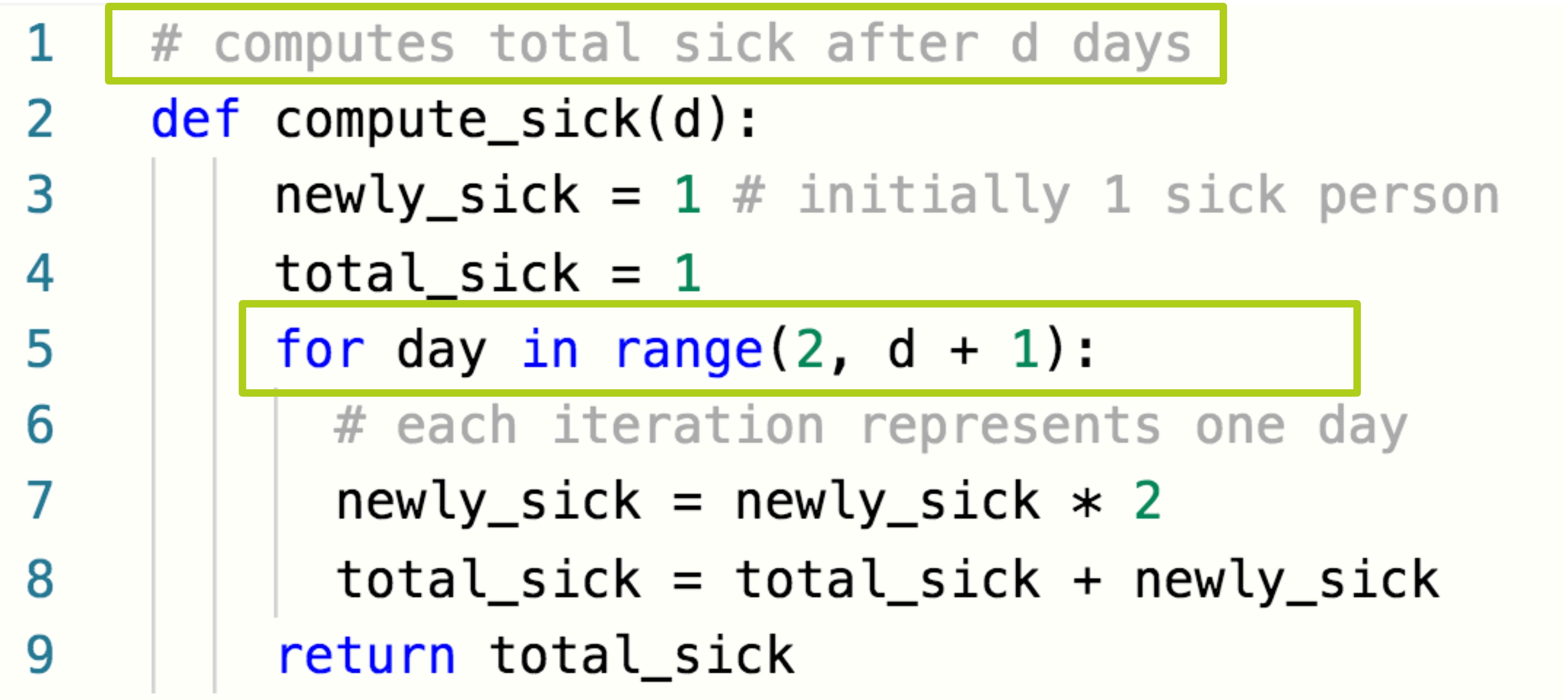
Each newly infected person infects 2 people the next day.
The function returns the number of sick people after n days.



```
# computes total sick after d days
def compute_sick(d):
    newly_sick = 1 # initially 1 sick person
    total_sick = 1
    for day in range(2, d + 1):
        # each iteration represents one day
        newly_sick = newly_sick * 2
        total_sick = total_sick + newly_sick
    return total_sick
```

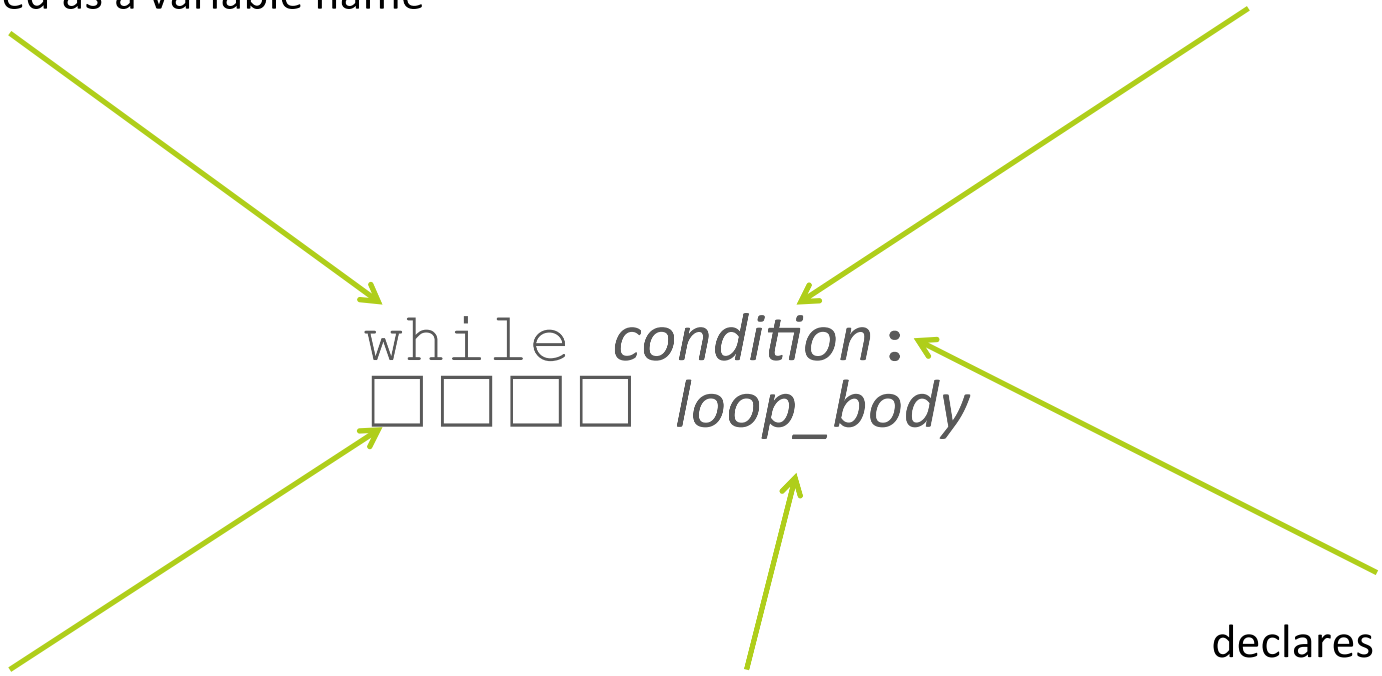
An epidemic

Each newly infected person infects 2 people the next day.
The function returns the number of sick people after n days.



```python
# computes total sick after d days
def compute_sick(d):
    newly_sick = 1 # initially 1 sick person
    total_sick = 1
    for day in range(2, d + 1):
        # each iteration represents one day
        newly_sick = newly_sick * 2
        total_sick = total_sick + newly_sick
    return total_sick
```

While Loop Syntax



`while` is a reserved word and cannot be used as a variable name

The *loop_body* executes while the *condition* holds true

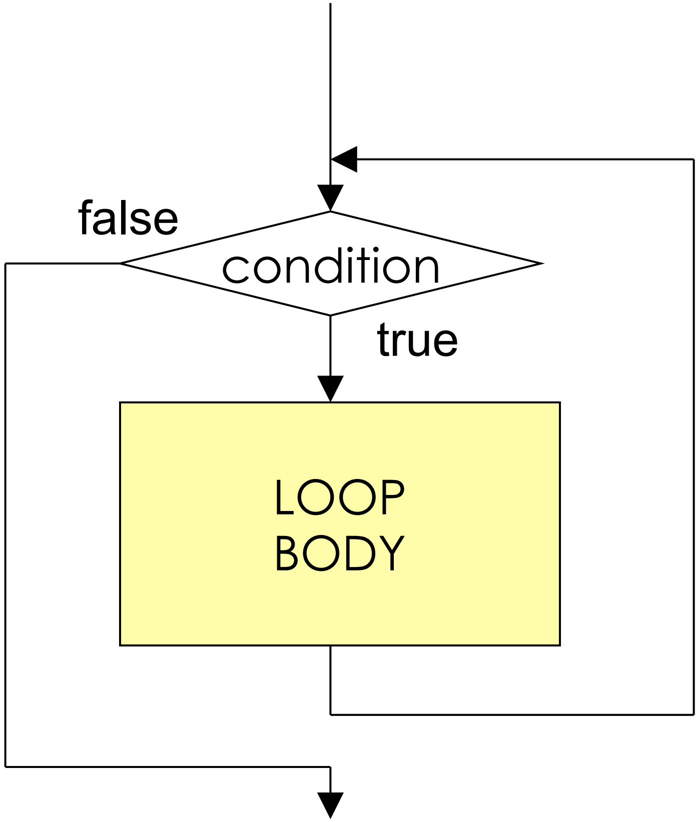
```
while condition:
□□□□ loop_body
```

Indentation is critical.
Use spaces *only,* **not tabs!**

One or more instructions that you want to repeat

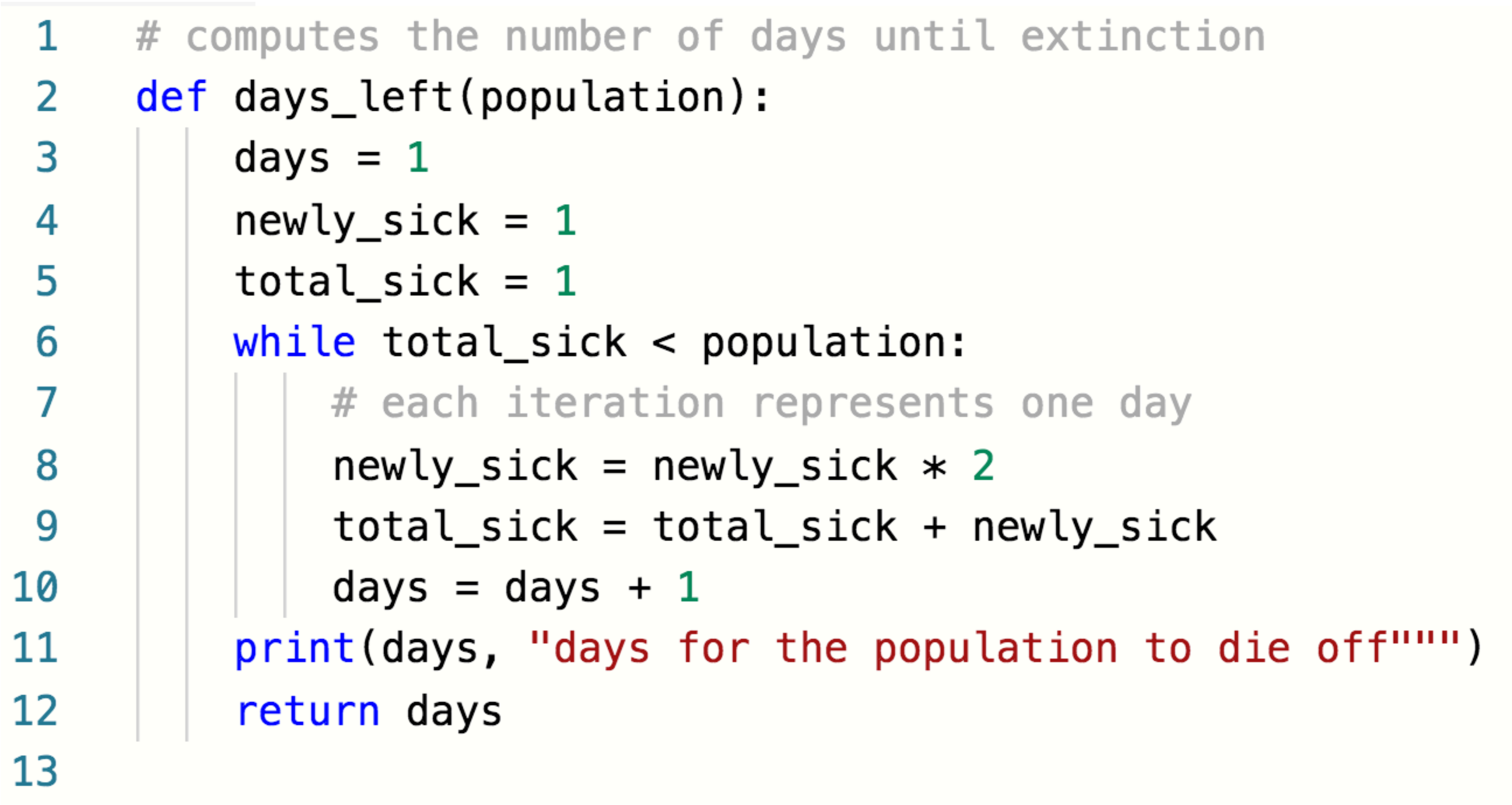
declares the start of an indented block

while loop



NOTE: If the loop condition becomes false during the loop body, the loop body still runs to completion before we exit the loop and go on with the next step.

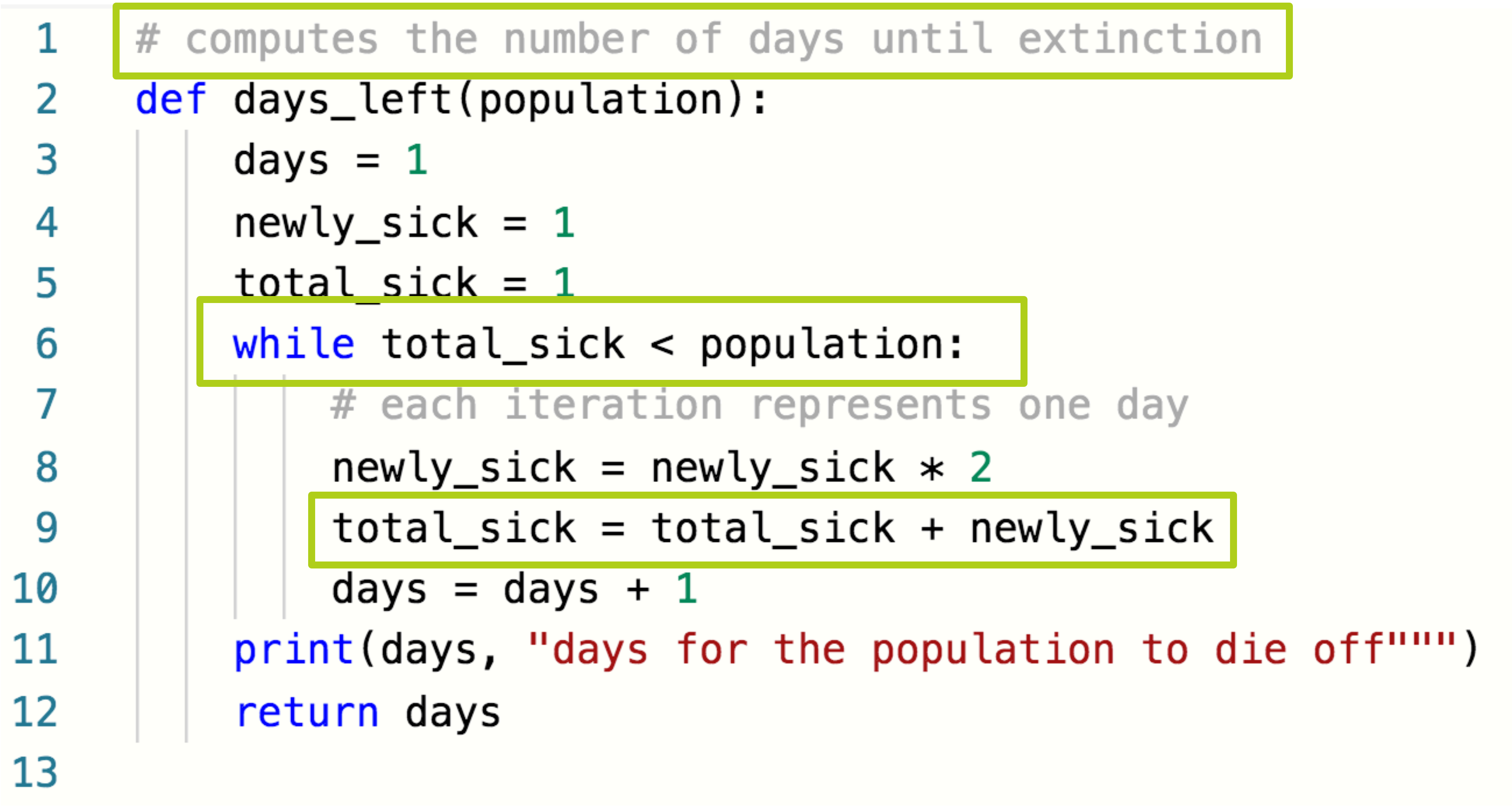
Variation on the Epidemic Example



```python
# computes the number of days until extinction
def days_left(population):
    days = 1
    newly_sick = 1
    total_sick = 1
    while total_sick < population:
        # each iteration represents one day
        newly_sick = newly_sick * 2
        total_sick = total_sick + newly_sick
        days = days + 1
    print(days, "days for the population to die off""")
    return days

```

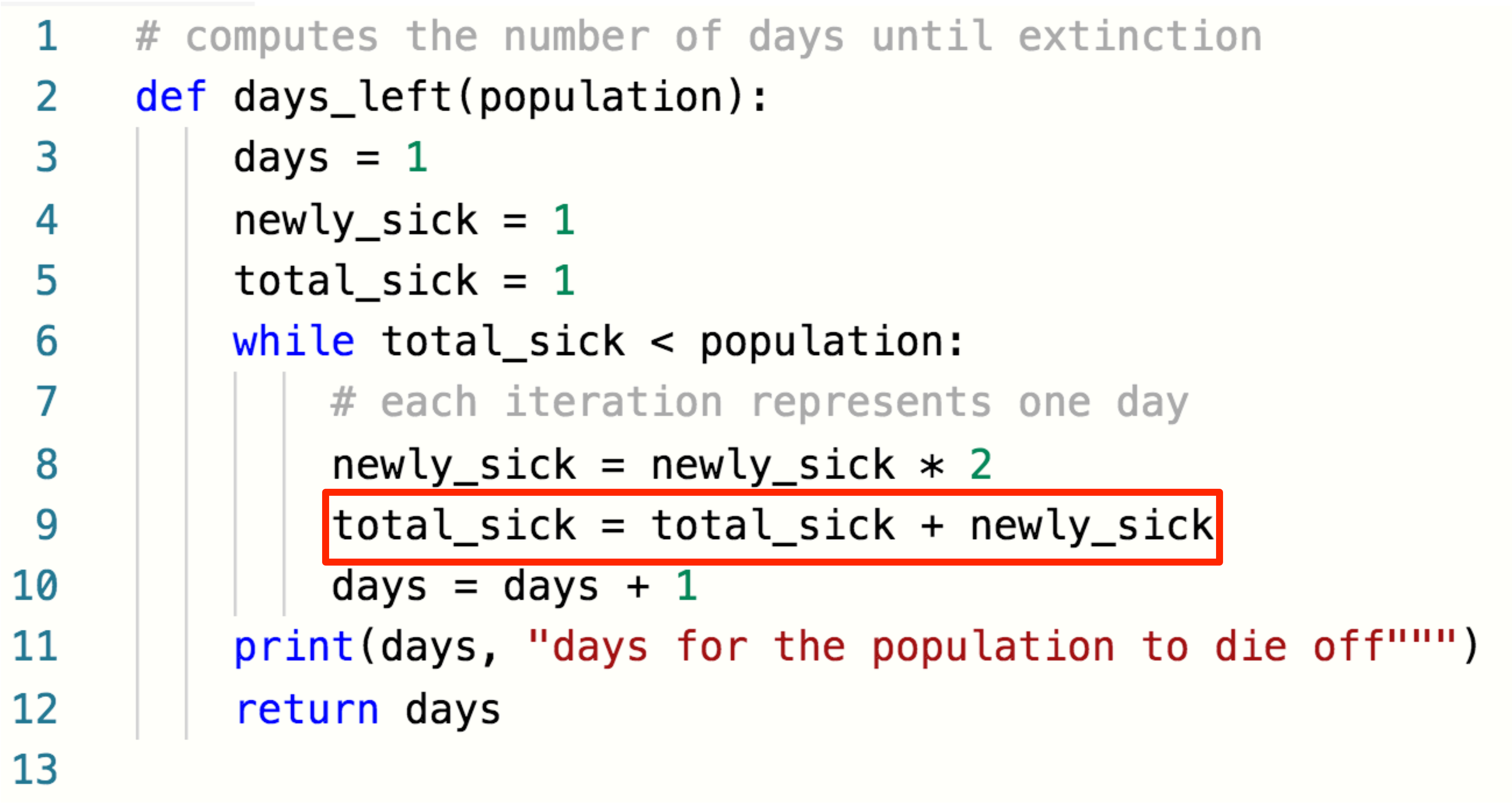
Variation on the Epidemic Example



```python
# computes the number of days until extinction
def days_left(population):
    days = 1
    newly_sick = 1
    total_sick = 1
    while total_sick < population:
        # each iteration represents one day
        newly_sick = newly_sick * 2
        total_sick = total_sick + newly_sick
        days = days + 1
    print(days, "days for the population to die off""")
    return days

```

Variation on the Epidemic Example



```python
# computes the number of days until extinction
def days_left(population):
    days = 1
    newly_sick = 1
    total_sick = 1
    while total_sick < population:
        # each iteration represents one day
        newly_sick = newly_sick * 2
        total_sick = total_sick + newly_sick
        days = days + 1
    print(days, "days for the population to die off""")
    return days
```

While vs. For Loops

```
# Prints first 10 positive integers

 i = 1
 while  i < 11:
     print(i)
     i = i + 1
```

```
# Prints first 10 positive integers

for i in range(1,11):
    print(i)
```

```
# Prints first 5 even integers

 i = 2
 while  i < 11:
     print(i)
     i = i + 2
```

```
# Prints first 5 even integers

for i in range(2,11,2):
    print(i)
```

When to use `for` or `while` loops

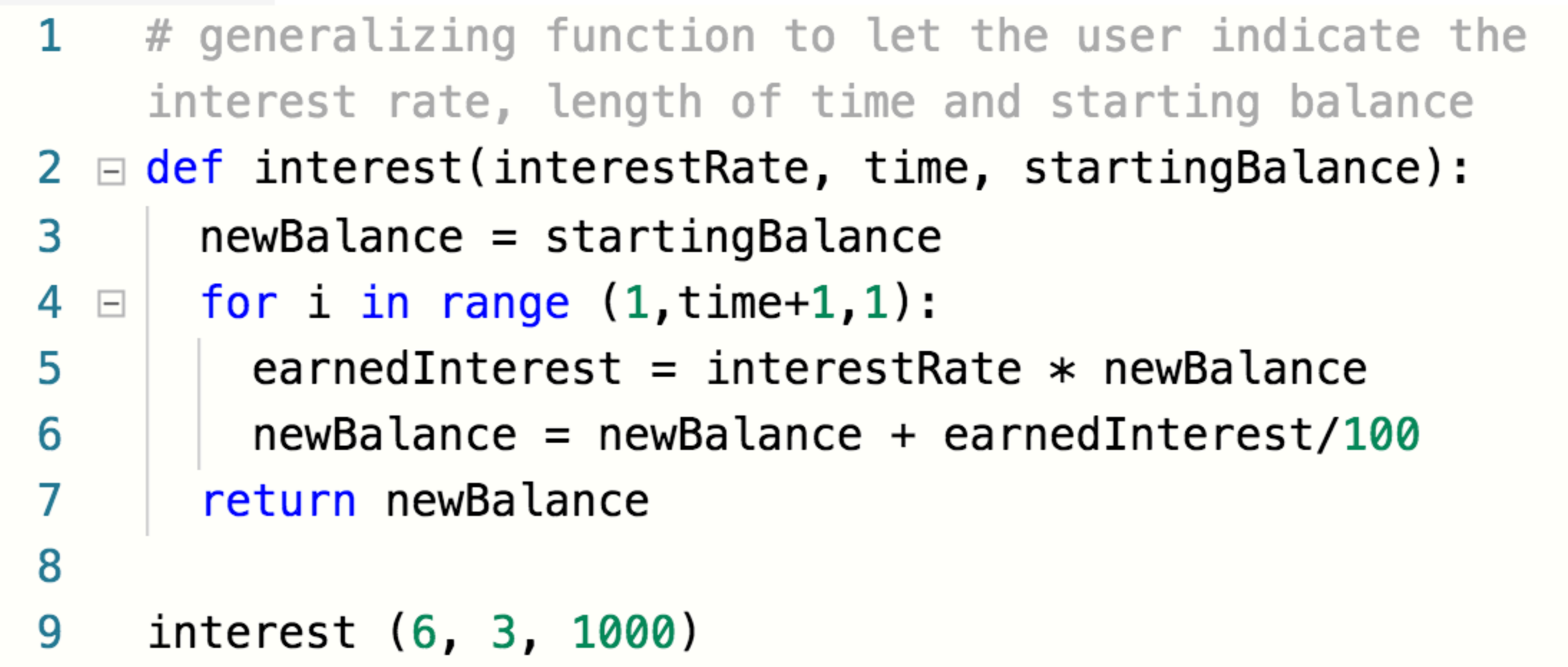
- If you know in advance **how many times** you want to run a loop use a **`for`** loop.

- When you **don’t know the number** of repetition needed, use a **`while`** loop.

Try:

- Calculating interest on a savings account at 6% interest for 3 years with a starting balance of $1000.
- Generalize the above – let the user indicate the interest rate and length of time.
- Parable: grains of rice on a chessboard, (1 grain on square one, 2 grains on square 2, 4 grains on square 3 …. through square 64)

Try:



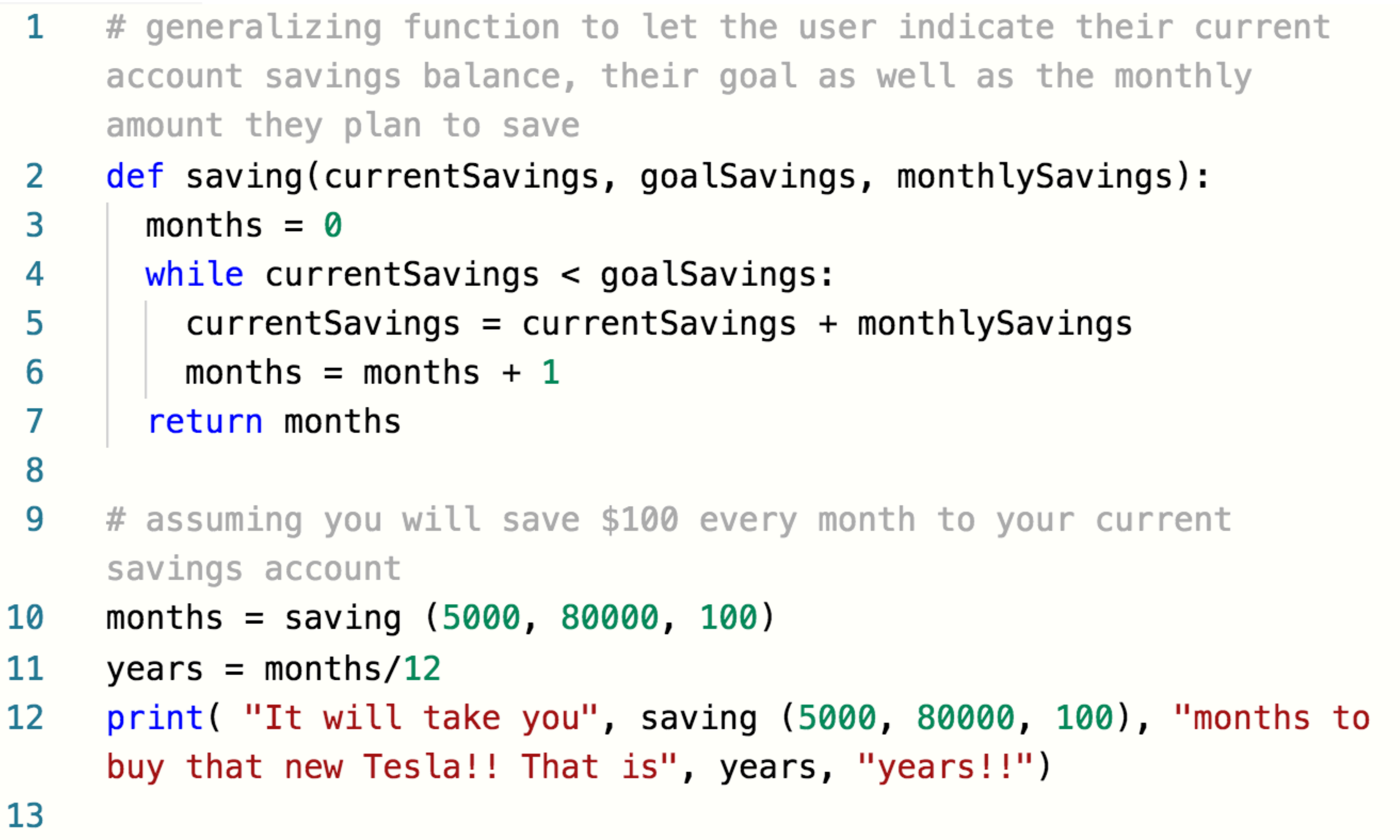
```python
# generalizing function to let the user indicate the interest rate, length of time and starting balance
def interest(interestRate, time, startingBalance):
  newBalance = startingBalance
  for i in range (1,time+1,1):
    earnedInterest = interestRate * newBalance
    newBalance = newBalance + earnedInterest/100
  return newBalance

interest (6, 3, 1000)
```

Try:

- Saving money to buy a new car – how long will it take to save for a new Tesla Model X @ $80,000. (5000.00 in a savings account)
- Saving for retirement – for different retirement targets, and calculate how long it will take to reach that target. Identify your variables and pre-assign values.
- Can you generalize the above to accommodate different user input?

Try:



```
# generalizing function to let the user indicate their current account savings balance, their goal as well as the monthly amount they plan to save
def saving(currentSavings, goalSavings, monthlySavings):
  months = 0
  while currentSavings < goalSavings:
    currentSavings = currentSavings + monthlySavings
    months = months + 1
  return months

# assuming you will save $100 every month to your current savings account
months = saving (5000, 80000, 100)
years = months/12
print( "It will take you", saving (5000, 80000, 100), "months to buy that new Tesla!! That is", years, "years!!")

```

Nesting

Nesting

```
for i in range(5):
        #body of the loop
```

Body of the loop is executed 5 times

Nesting

```
for i in range(5):
        #body of the loop
```

Body of the loop is executed 5 times

```
for j in range(3):
        #body of the loop
```

Body of the loop is executed 3 times

Nesting

```
for i in range(5):
        #body of the loop
```

Body of the loop is executed 5 times

```
for j in range(3):
        #body of the loop
```

Body of the loop is executed 3 times

```
for i in range(5):
        for j in range(3):
                #body of the loop
```

Nesting

```
for i in range(5):
        #body of the loop
```

Body of the loop is executed 5 times

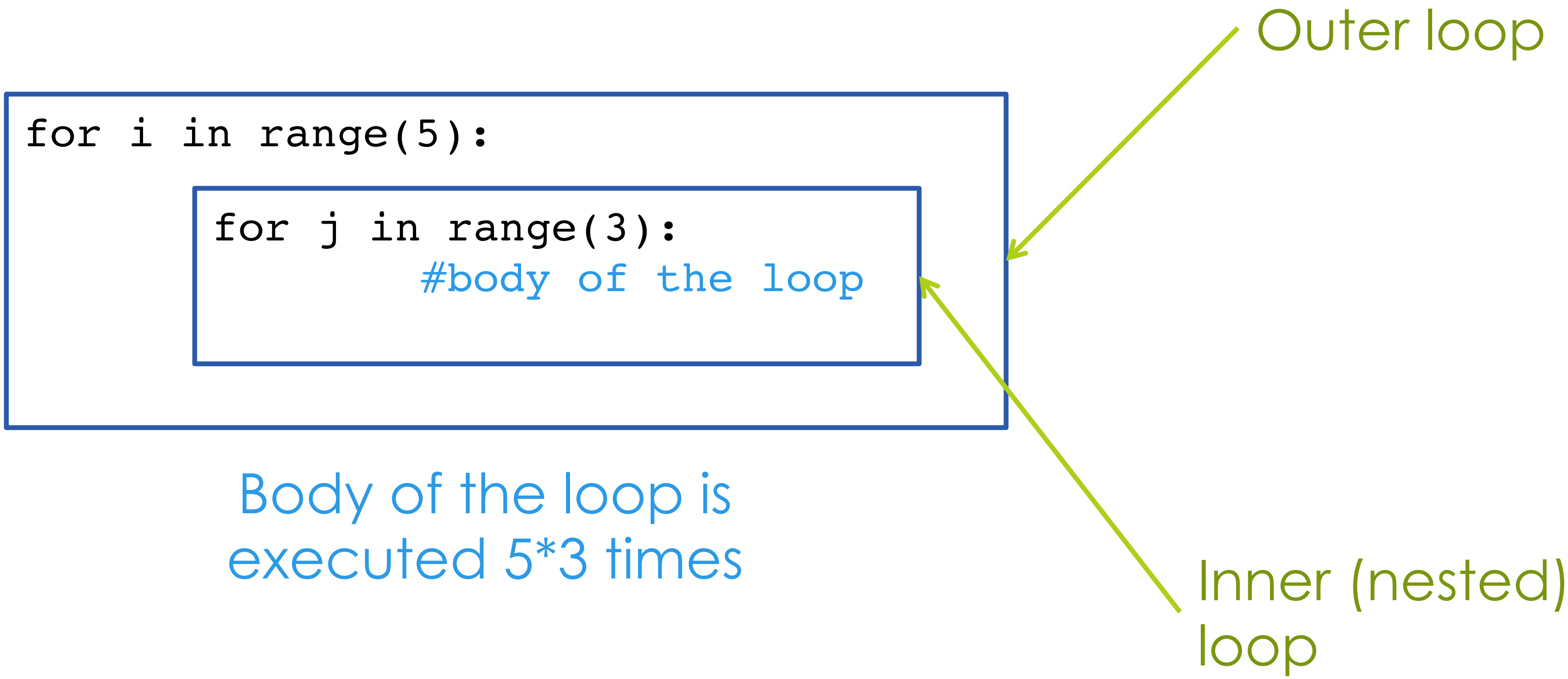
```
for j in range(3):
        #body of the loop
```

Body of the loop is executed 3 times

```
for i in range(5):
        for j in range(3):
                #body of the loop
```

Body of the loop is executed 5*3 times

Nesting



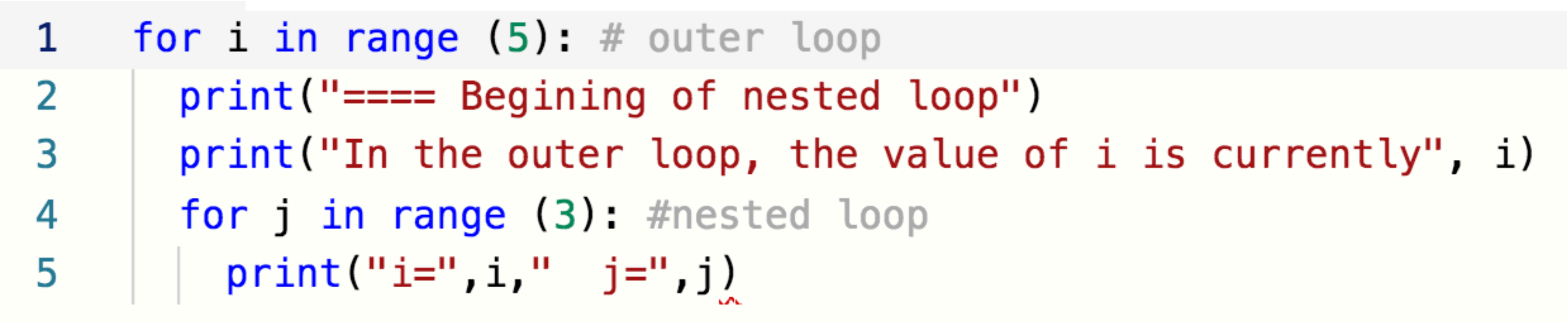
Outer loop

```
for i in range(5):
        for j in range(3):
                #body of the loop
```

Inner (nested) loop

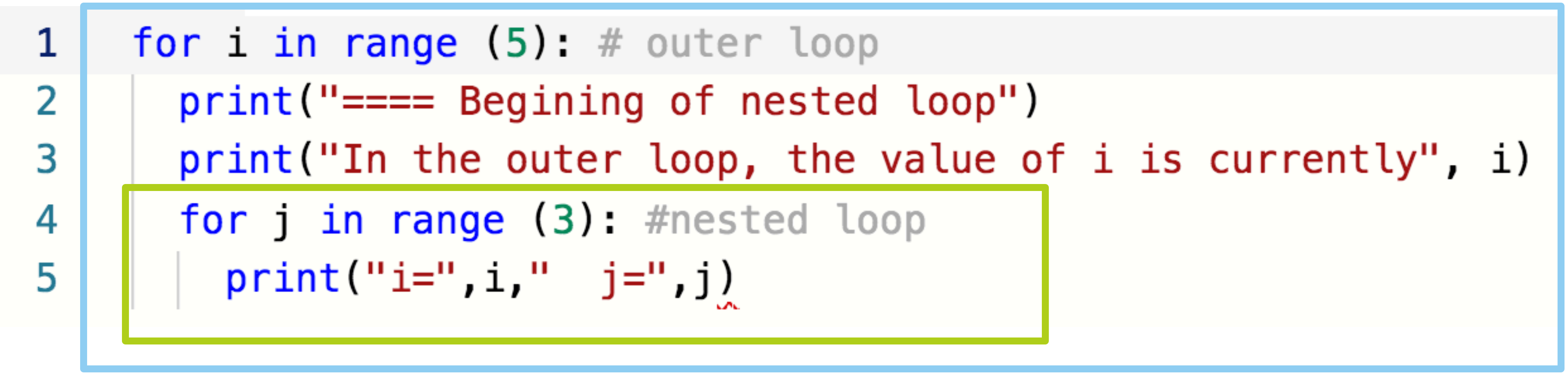
Body of the loop is executed 5*3 times

Nesting



```
for i in range (5): # outer loop
  print("==== Begining of nested loop")
  print("In the outer loop, the value of i is currently", i)
  for j in range (3): #nested loop
    print("i=",i,"  j=",j)
```

Nesting



```python
for i in range (5): # outer loop
  print("==== Begining of nested loop")
  print("In the outer loop, the value of i is currently", i)
  for j in range (3): #nested loop
    print("i=",i,"  j=",j)
```

Algorithms

Algorithms

- An algorithm is “a precise rule (or set of rules) specifying how to solve some problem.” (thefreedictionary.com)

- The study of algorithms is one of the foundations of computer science.

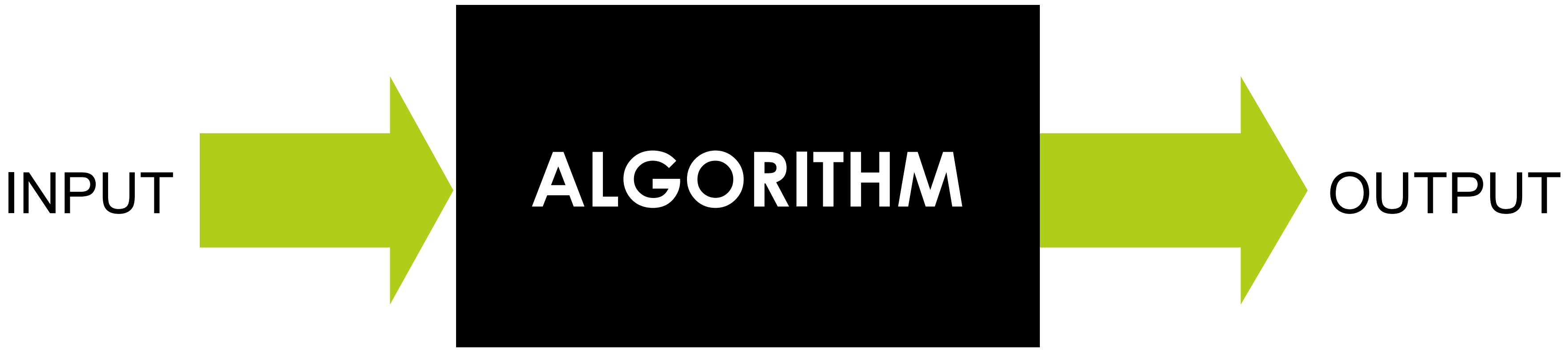
image: AL-KHWARIZMI
historyoflinearalgebra.weebly.com

Mohammed al-Khwarizmi (äl-khōwärēz´mē)

Persian mathematician of the court of Mamun in Baghdad…the word ***algorithm*** is said to have been derived from his name. Much of the mathematical knowledge of medieval Europe was derived from Latin translations of his works. (encyclopedia.com)

An algorithm is like a function

$$\boxed{F(x) \rightarrow y}$$



INPUT

ALGORITHM

OUTPUT

Input

- **Input specification**
  - Recipes: ingredients, cooking utensils, …
  - Knitting: size of garment, length of yarn, needles …
  - Tax Code: wages, interest, tax withheld, …

- Input specification for computational algorithms:
  - **What kind of data** is required?
  - **In what form** will this data be received by the algorithm?

Computation

- An algorithm requires **clear and precisely stated steps** that express how to perform the operations to yield the desired results.

- Algorithms assume a basic set of ***primitive operations*** that are assumed to be understood by the executor of the algorithm.
  - Recipes: beat, stir, blend, bake, …
  - Knitting: casting on, slip loop, draw yarn through, ...
  - Tax code: deduct, look up, check box, …
  - Computational: add, set, modulo, output, …

Output

- Output specification
  - Recipes: number of servings, how to serve
  - Knitting: final garment shape
  - Tax Code: tax due or tax refund, where to pay

- Output specification for computational algorithms:
  - **What results** are required?
  - **How** should these results be **reported**?
  - **What happens if no results can be computed** due to an error in the input? What do we output to indicate this?

Is this a “good” algorithm?

- Input: slices of bread, jar of peanut butter, jar of jam

  1. Pick up some bread.
  2. Put peanut butter on the bread.
  3. Pick up some more bread.
  4. Open the jar of jam.
  5. Spread the jam on the bread.
  6. Put the bread together to make your sandwich.

- Output?

What makes a “good” algorithm?

A good algorithm **should :**

1. produce the **correct outputs for any set of legal inputs.**

2. **execute efficiently with the fewest number of steps as possible** and should **always stop.**

3. be designed in such a way that **others will be able to understand it and modify it to specify solutions to additional problems.**

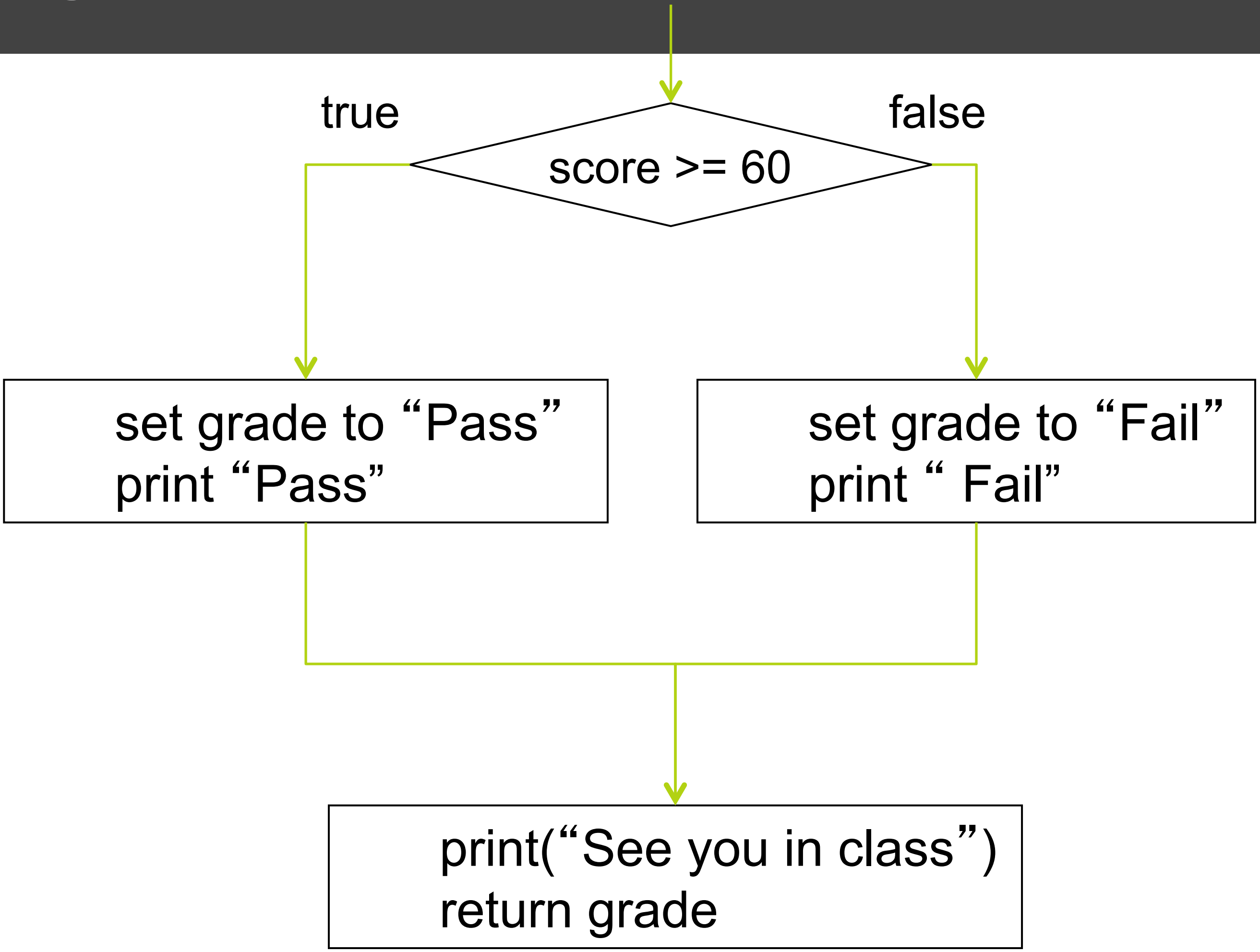
A Simple Algorithm

Input numerical *score* between 0 and 100 and
Output “Pass” or “Fail”

```
Algorithm:
1. If score >= 60
      a.  Set grade to “Pass”
      b.  Print “Pass”
2. Otherwise,
      a.  Set grade to “Fail”
      b.  Print “Fail”
3. Print “See you in class”
4. Return grade
```

Exactly **one** of step 1 or step 2 is executed, but step 3 and step 4 are **always** executed.

Control Flow



true

false

score >= 60

set grade to “Pass”
print “Pass”

set grade to “Fail”
print “ Fail”

print(“See you in class”)
return grade

Coding the Grader in Python

```
Algorithm:
1. If score >= 60
    a.  Set grade to “Pass”
    b.  Print “ Pass”
2. Otherwise,
    a.  Set grade to “Fail”
    b.  Print “Fai”
3. Print “See you in class ”
4. Return grade
```

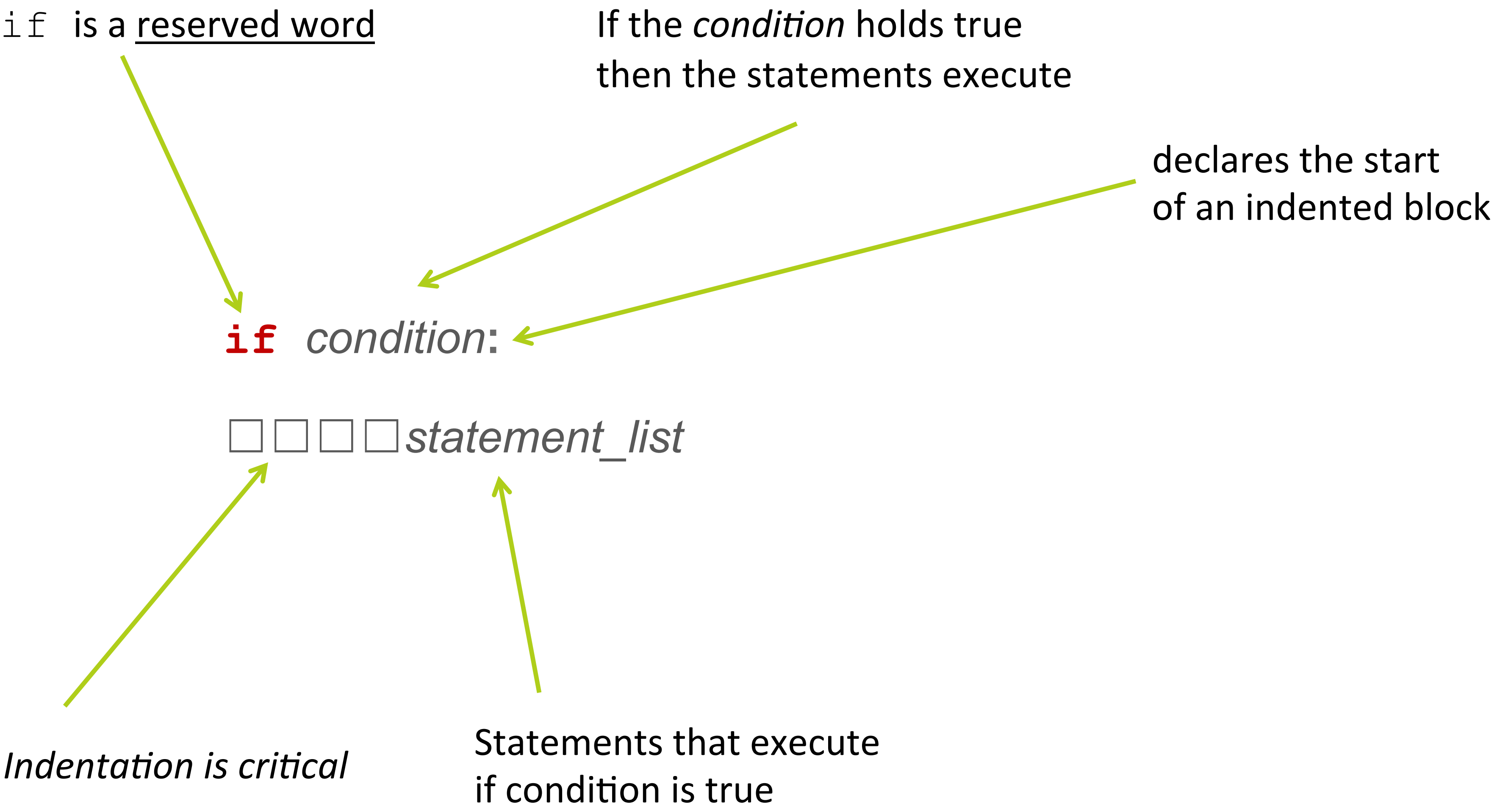
```python
def grader(score):
    if score >= 60:
        grade = "Pass"
        print("Pass")
    else:
        grade = "Fail"
        print("Fail")
    print("See you in class")
    return grade
```

If conditions

Flow chart: **`if`** statement

```
if condition:
□□□□statement_list
```

Flow chart: **`if`** statement



`if` is a <u>reserved word</u>

If the *condition* holds true
then the statements execute

declares the start
of an indented block

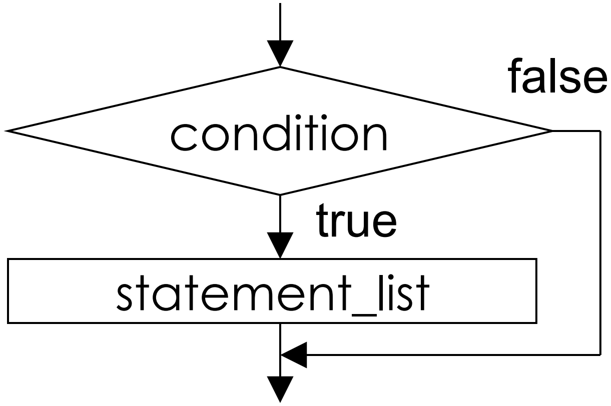
```
if condition:
□□□□statement_list
```

Indentation is critical

Statements that execute
if condition is true

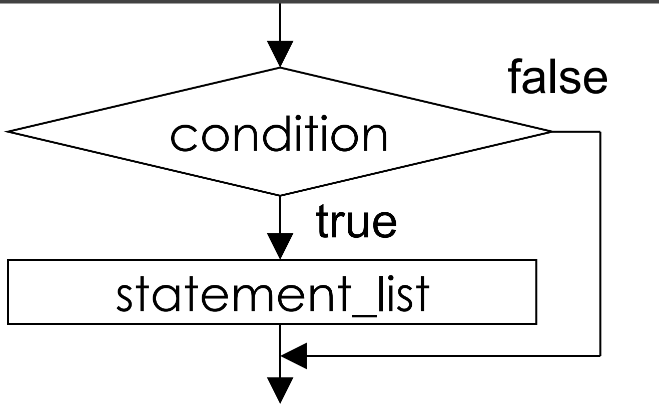
Flow chart: `if` statement

```
if condition:
□□□□statement_list
```



Flow chart: `if` statement

```
if condition:
□□□□statement_list
```



```
# Print statement gets
  executed

 grade = 75
 if grade > 60:
     print(“You pass!”)
```

```
# Print statement does not get
  executed

 grade = 50
 if grade > 60:
     print(“You pass!”)
```

Flow chart:
if/else statement

```
if condition:
□□□□ statement_list1
else:
□□□□ statement_list2
```

Flow chart: **if/else** statement

```
if  condition:
□□□□  statement_list1
else:
□□□□  statement_list2
```

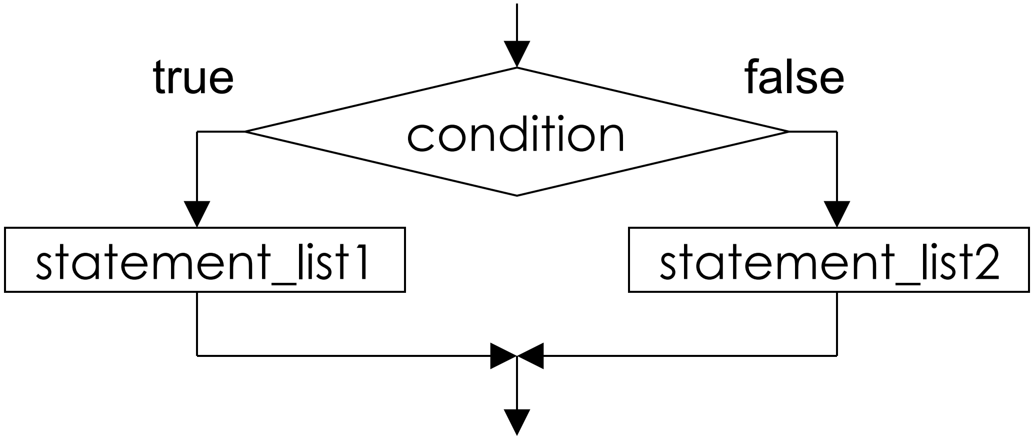
If the *condition* holds **true** then these statements execute

If the *condition* is **false** then these statements execute

`else` is a reserved word

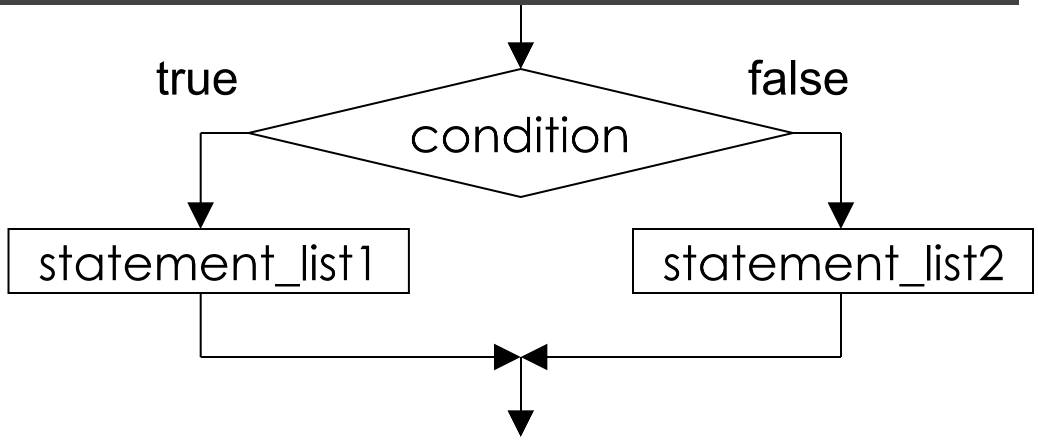
Flow chart:
if/else statement

```
if condition:
□□□□ statement_list1
else:
□□□□ statement_list2
```



Flow chart:
if/else statement

```
if condition:
□□□□ statement_list1
else:
□□□□ statement_list2
```



```
# You pass gets executed

  grade = 75
  if grade > 60:
      print(“You pass!”)

  else:
      print(“You fail :(”)
```

```
# You fail gets executed

  grade = 50
  if grade > 60:
      print(“You pass!”)

  else:
      print(“You fail :(”)
```

Flow chart: `if/elif/else` statement

```
if condition1:
□□□□ statement_list1
elif condition2:
□□□□ statement_list2
else:
□□□□ statement_list3
```

Flow chart:
if/elif/else statement

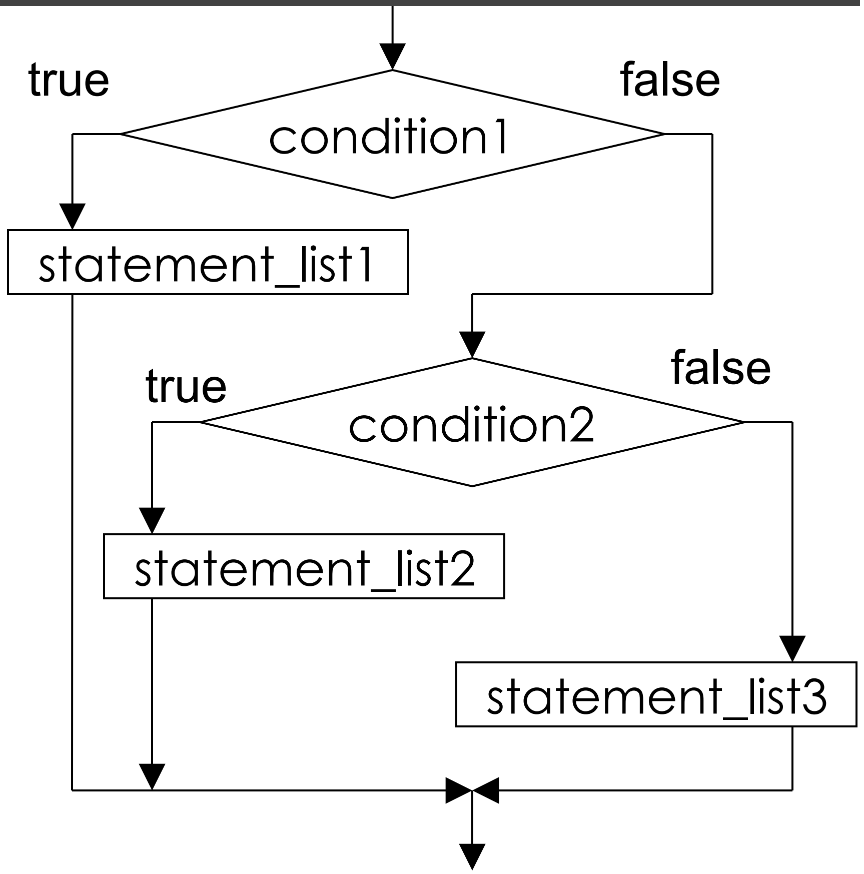
```
if condition1:
□□□□ statement_list1
elif condition2:
□□□□ statement_list2
else:
□□□□ statement_list3
```

You can have as many as you need!

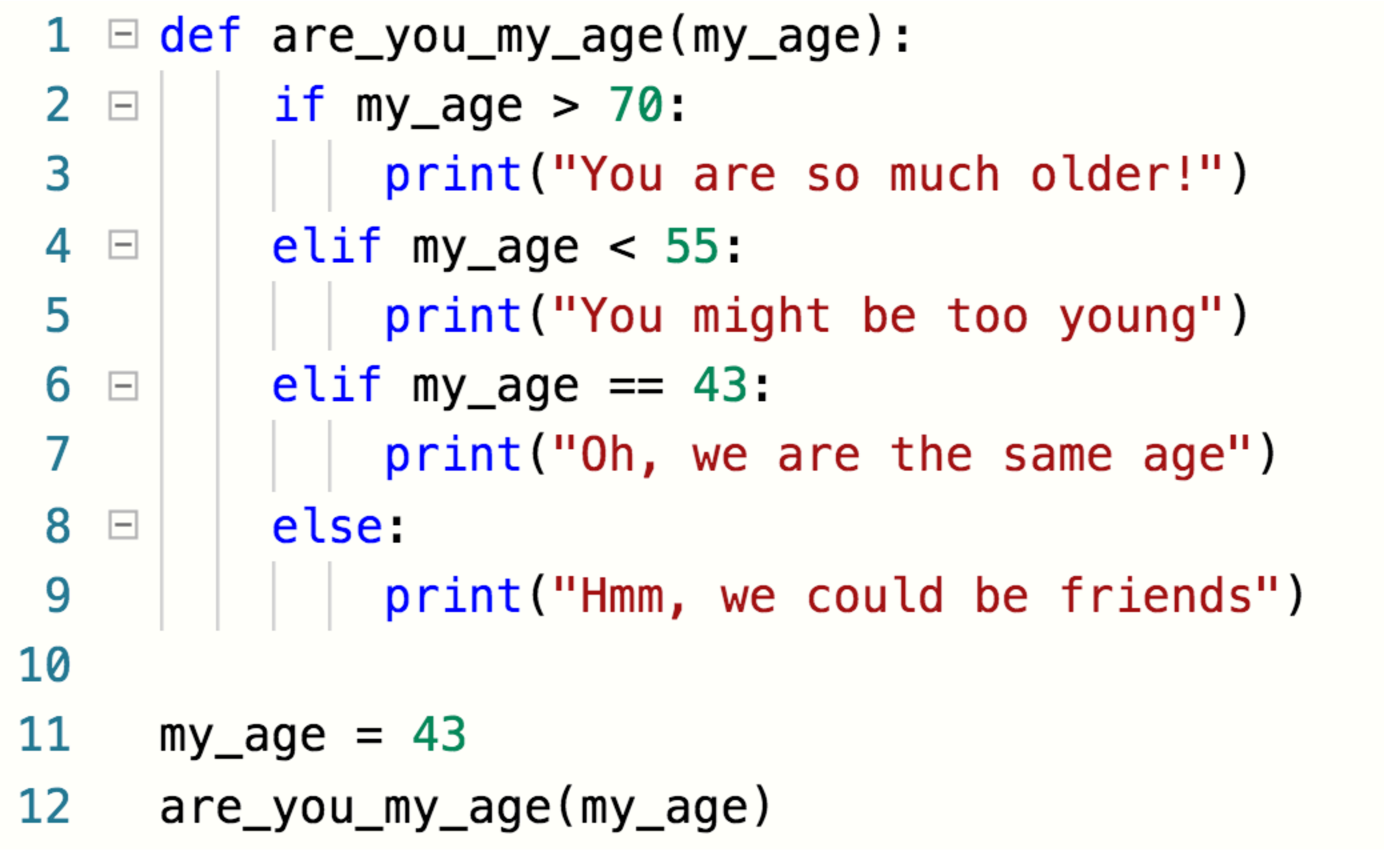
`elif` is a reserved word

Flow chart:
if/elif/else statement

```
if condition1:
□□□□ statement_list1
elif condition2:
□□□□ statement_list2
else:
□□□□ statement_list3
```



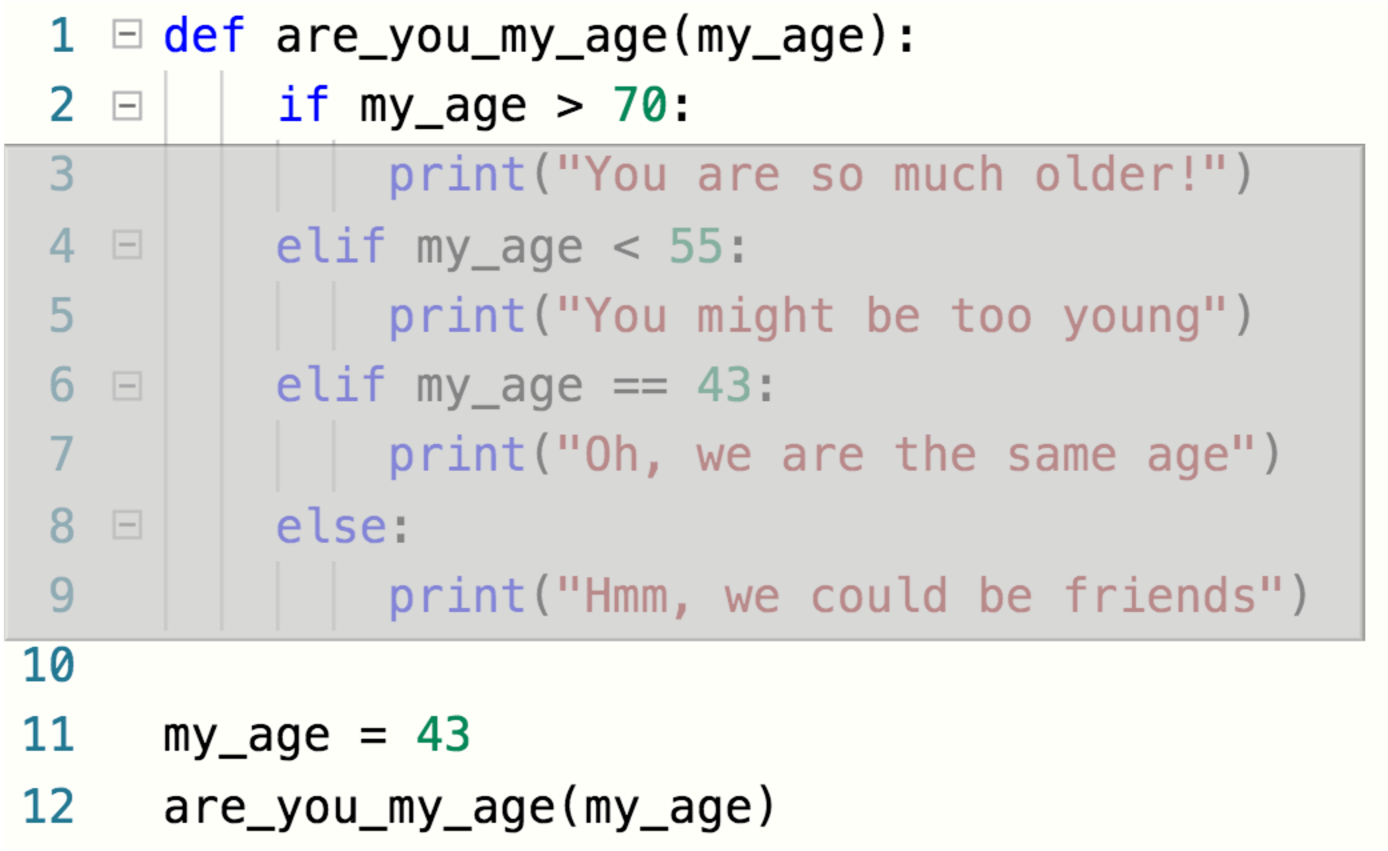
What is going to get printed?



```python
def are_you_my_age(my_age):
    if my_age > 70:
        print("You are so much older!")
    elif my_age < 55:
        print("You might be too young")
    elif my_age == 43:
        print("Oh, we are the same age")
    else:
        print("Hmm, we could be friends")

my_age = 43
are_you_my_age(my_age)
```

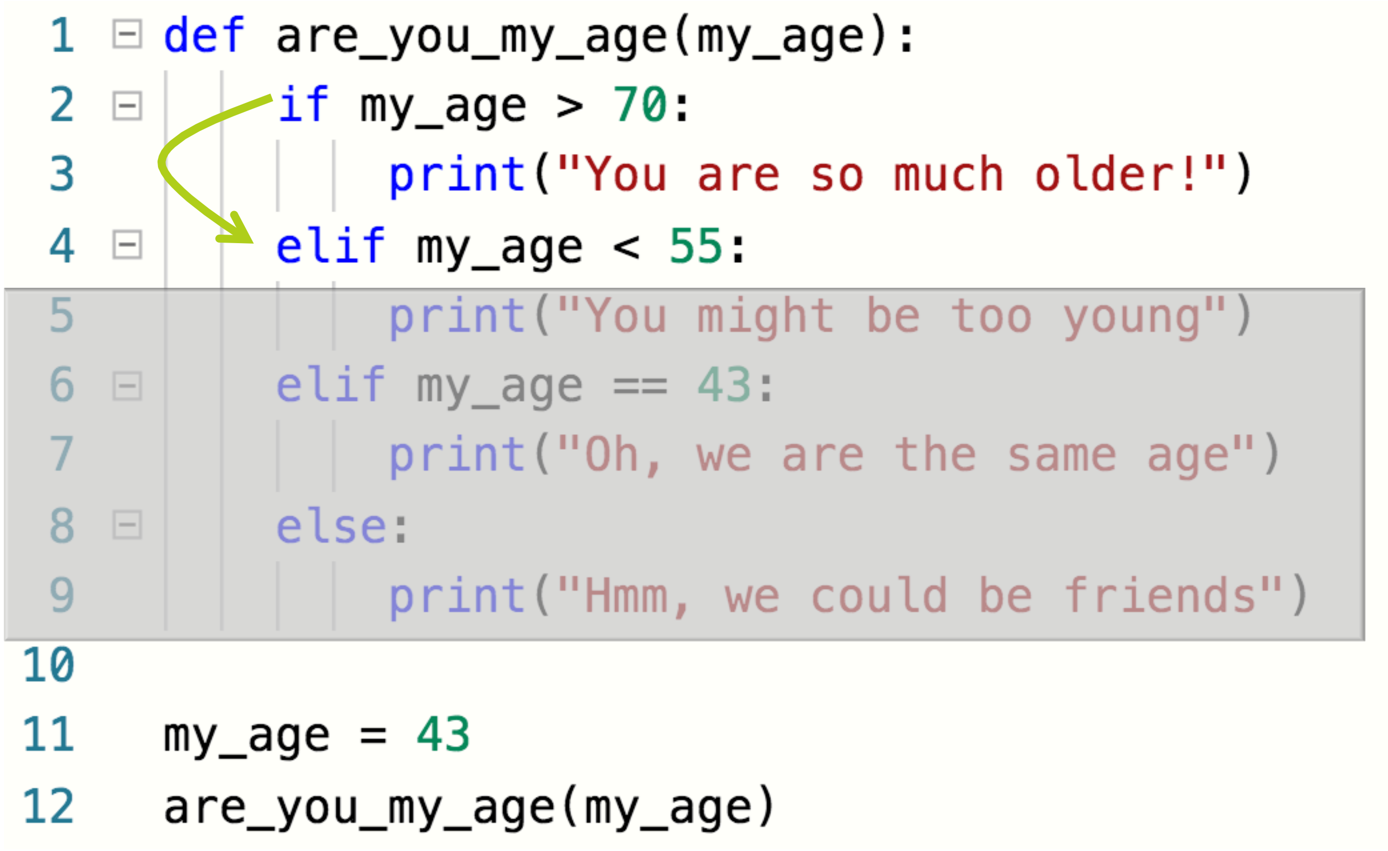
What is going to get printed?



```
def are_you_my_age(my_age):
    if my_age > 70:
        print("You are so much older!")
    elif my_age < 55:
        print("You might be too young")
    elif my_age == 43:
        print("Oh, we are the same age")
    else:
        print("Hmm, we could be friends")

my_age = 43
are_you_my_age(my_age)
```

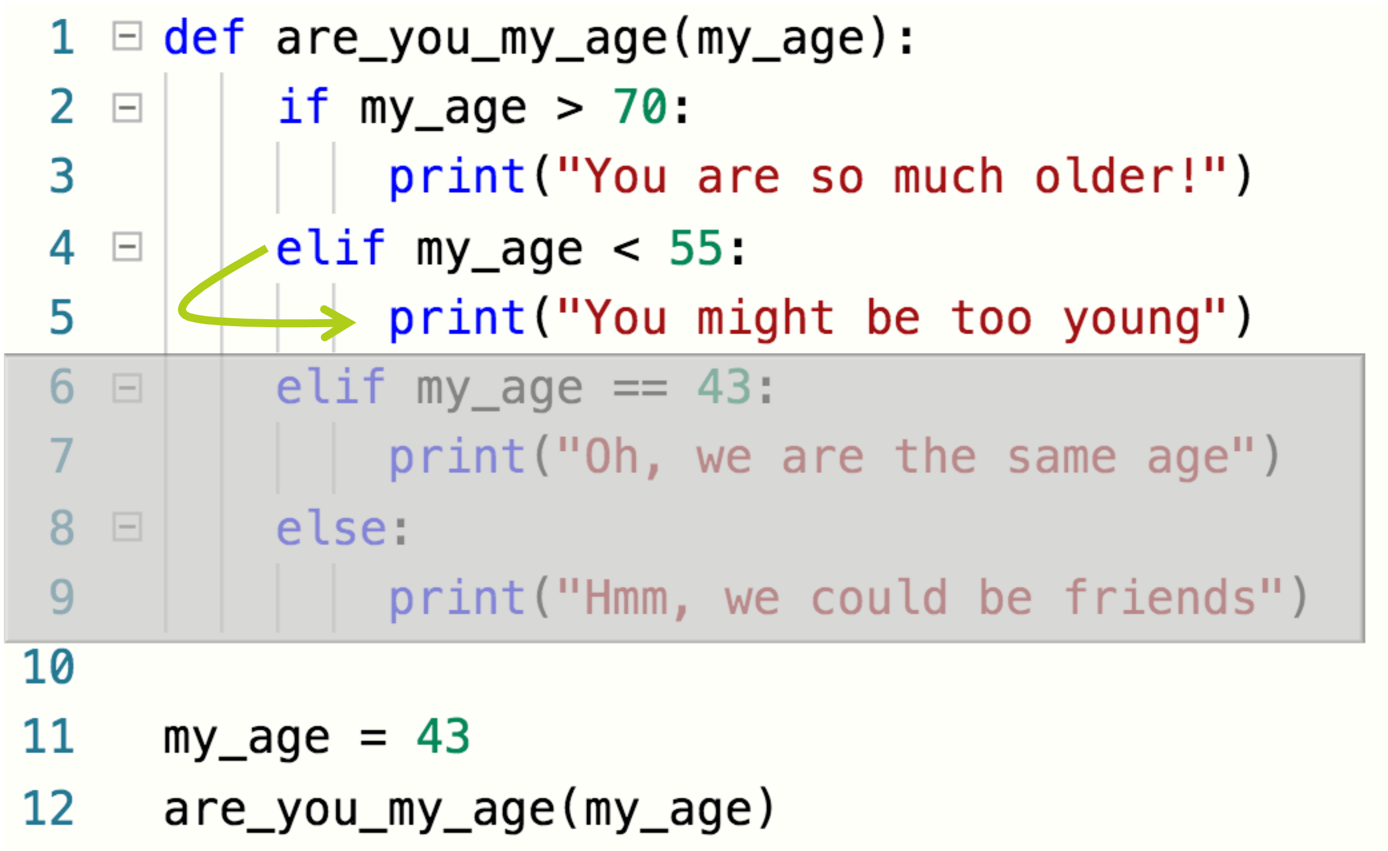
What is going to get printed?



```python
def are_you_my_age(my_age):
    if my_age > 70:
        print("You are so much older!")
    elif my_age < 55:
        print("You might be too young")
    elif my_age == 43:
        print("Oh, we are the same age")
    else:
        print("Hmm, we could be friends")

my_age = 43
are_you_my_age(my_age)
```

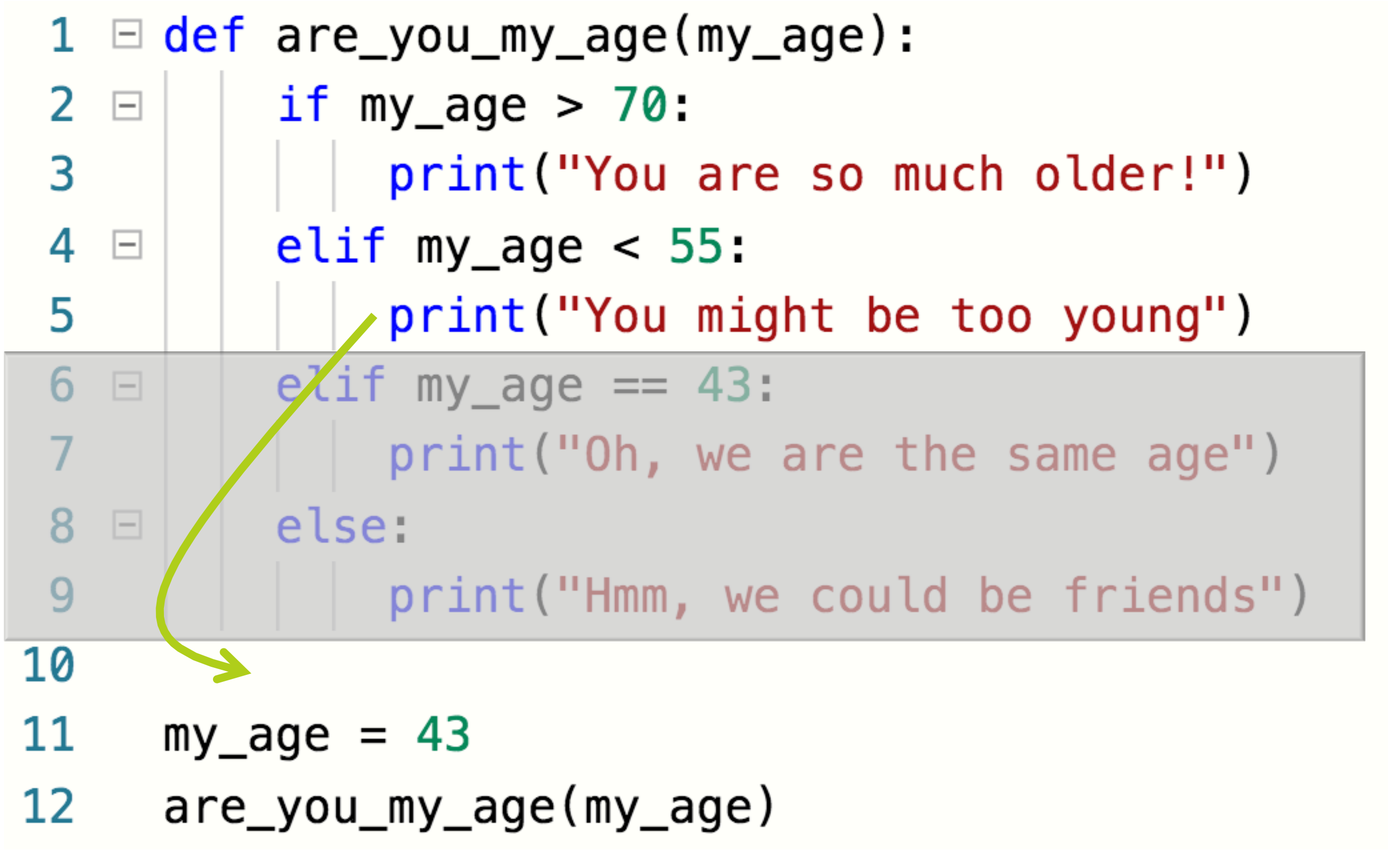
What is going to get printed?



```python
def are_you_my_age(my_age):
    if my_age > 70:
        print("You are so much older!")
    elif my_age < 55:
        print("You might be too young")
    elif my_age == 43:
        print("Oh, we are the same age")
    else:
        print("Hmm, we could be friends")

my_age = 43
are_you_my_age(my_age)
```

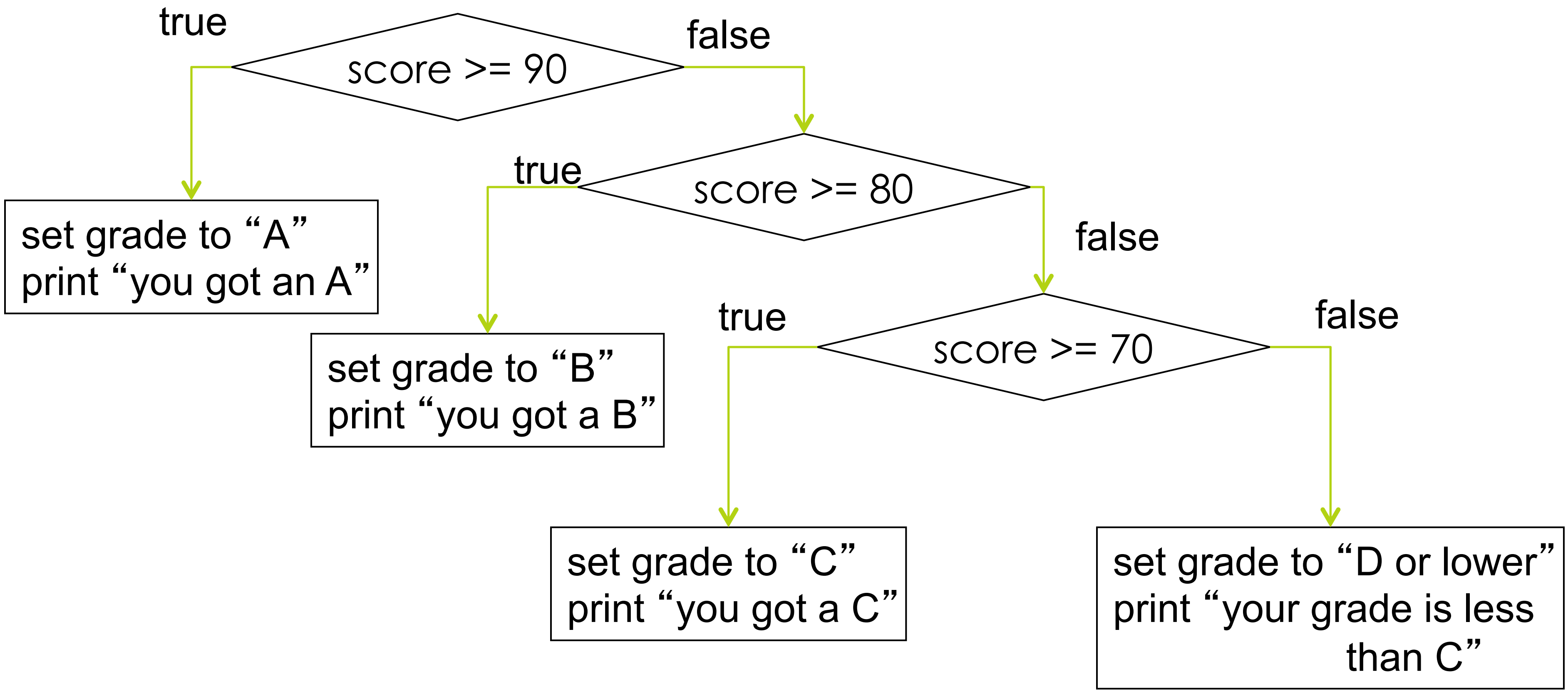
What is going to get printed?



```python
def are_you_my_age(my_age):
    if my_age > 70:
        print("You are so much older!")
    elif my_age < 55:
        print("You might be too young")
    elif my_age == 43:
        print("Oh, we are the same age")
    else:
        print("Hmm, we could be friends")

my_age = 43
are_you_my_age(my_age)
```

Grader for Letter Grades



Letter grade program in Python

```python
def grader3(score):
    if score >= 90:
        grade = "A"
        print("You got an A")
    elif score >= 80:
        grade = "B"
        print("You got a B")
    elif score >= 70:
        grade = "C"
        print("You got a C")
    else:
        grade = "D or lower"
        print("Your grade is less than C")
    return grade
```

Nested if statements

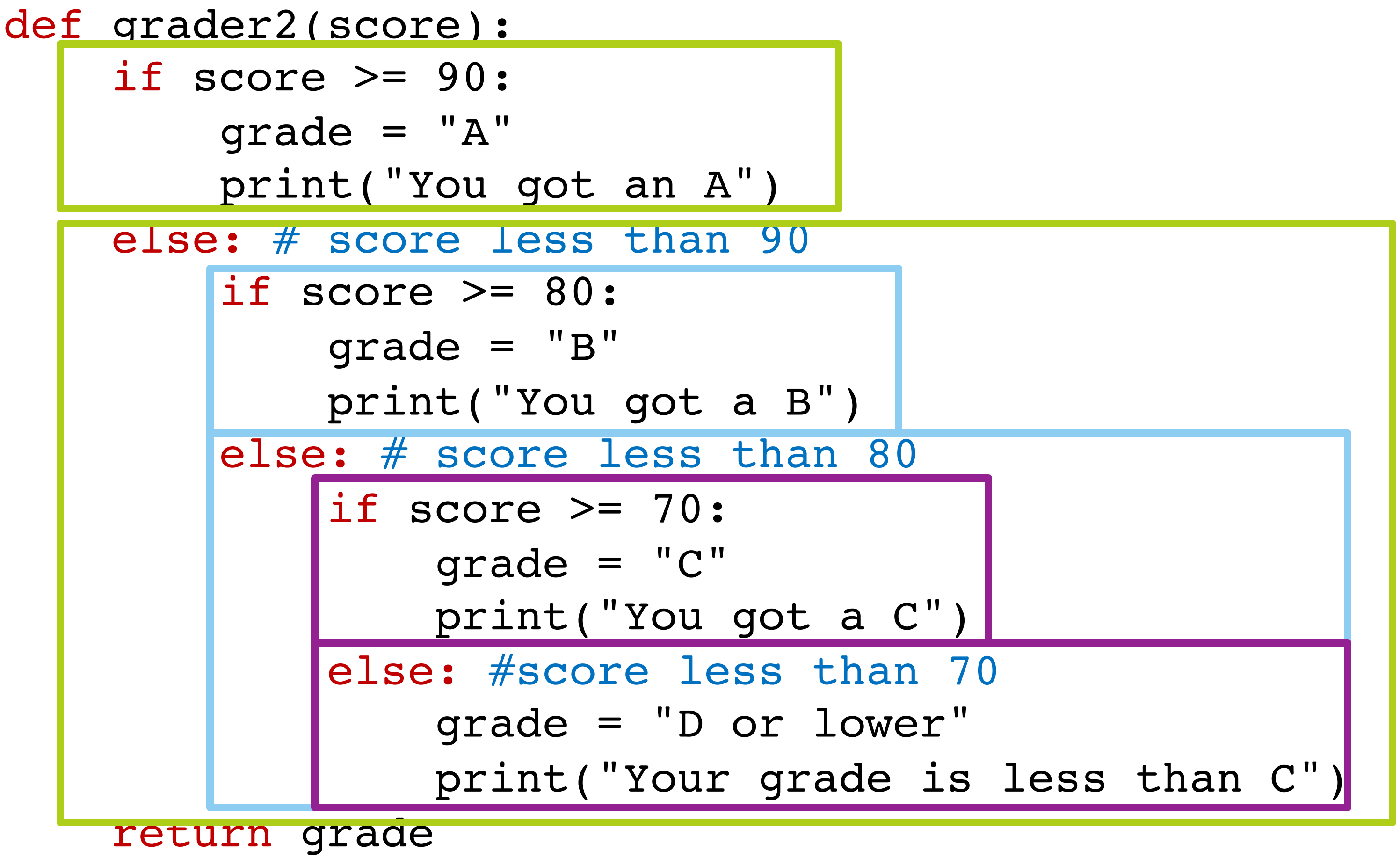
Nested if statements

```
def grader2(score):
    if score >= 90:
        grade = "A"
        print("You got an A")
    else: # score less than 90
        if score >= 80:
            grade = "B"
            print("You got a B")
        else: # score less than 80
            if score >= 70:
                grade = "C"
                print("You got a C")
            else: #score less than 70
                grade = "D or lower"
                print("Your grade is less than C")
    return grade
```

Nested if statements

```
def grader2(score):
    if score >= 90:
        grade = "A"
        print("You got an A")
    else: # score less than 90
        if score >= 80:
            grade = "B"
            print("You got a B")
        else: # score less than 80
            if score >= 70:
                grade = "C"
                print("You got a C")
            else: #score less than 70
                grade = "D or lower"
                print("Your grade is less than C")
    return grade
```

Nested if statements



```
def grader2(score):
    if score >= 90:
        grade = "A"
        print("You got an A")
    else: # score less than 90
        if score >= 80:
            grade = "B"
            print("You got a B")
        else: # score less than 80
            if score >= 70:
                grade = "C"
                print("You got a C")
            else: #score less than 70
                grade = "D or lower"
                print("Your grade is less than C")
    return grade
```

Summary

- Notion of an algorithm:
  - Kinds of instructions needed to express algorithms
  - What makes an algorithm a good one
- Instructions for specifying control flow (for loop, while loop, if/then/else)
  - Flow charts to express control flow in a language-independent way
  - Coding these control flow structures in Python

Exercise

Write a function that prints whether **`die1`** and **`die2`** are doubles, cat's eyes (two 1's) or neither of these.

```
def print_doubles(die1, die2):
```

Exercise

Write a function that returns how many of the three integers `n1`, `n2`, and `n3` are odd:

```
def num_odd(n1, n2, n3):
```