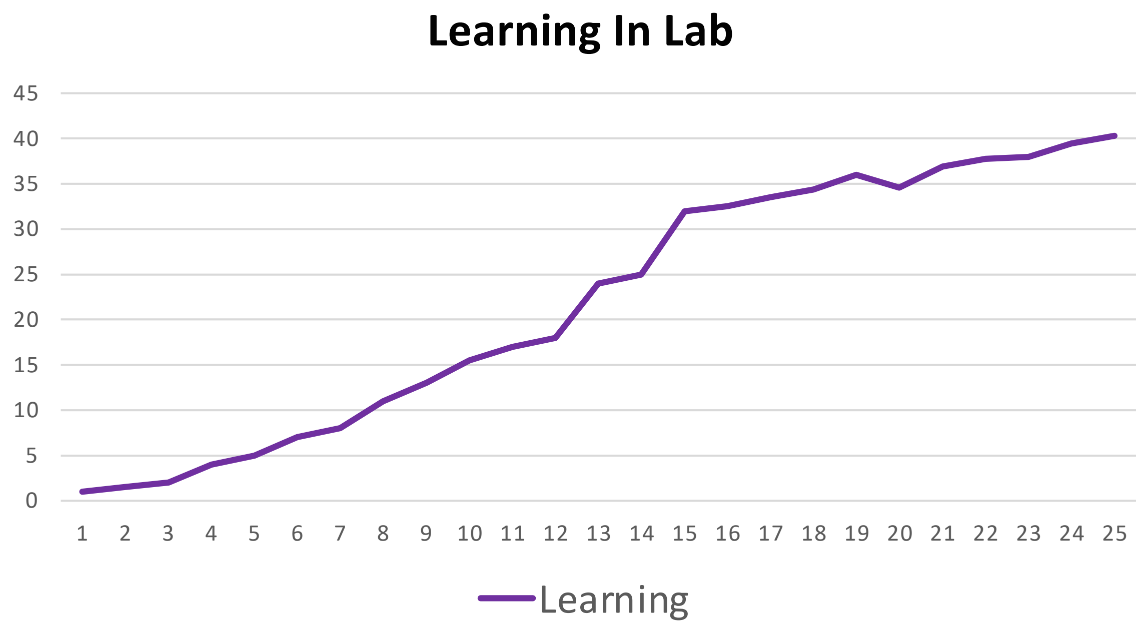
Objectives

- Conditionals
- Exam review

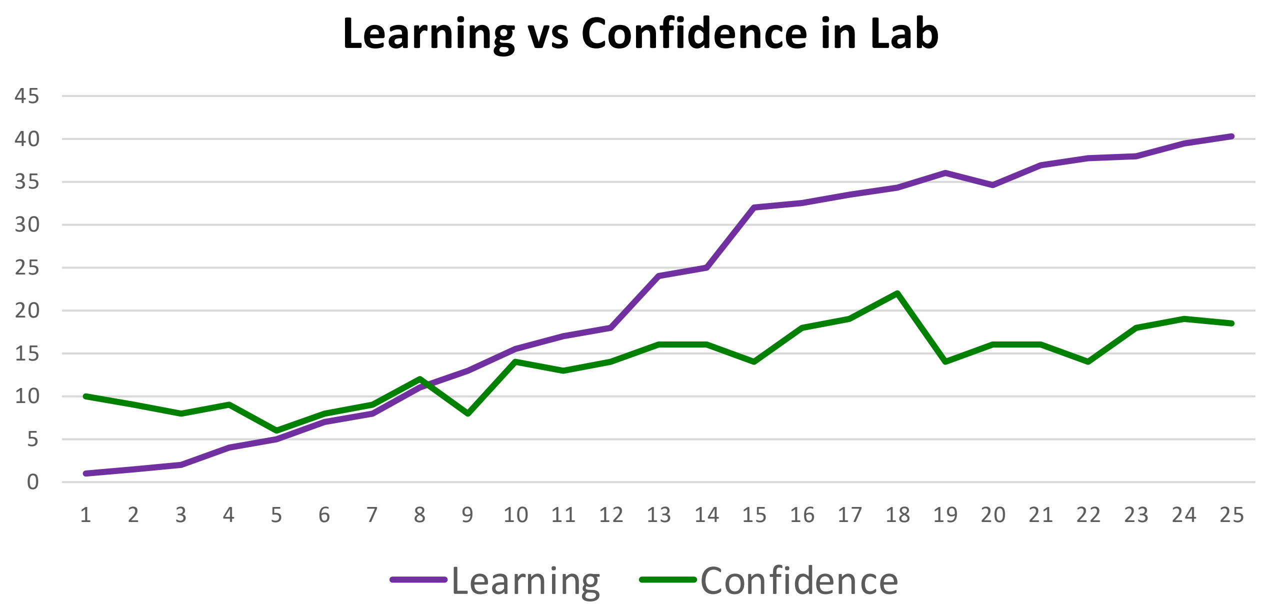
Your Learning Journey

- You're learning a lot
  - Struggle is part of the learning



Your Learning Journey

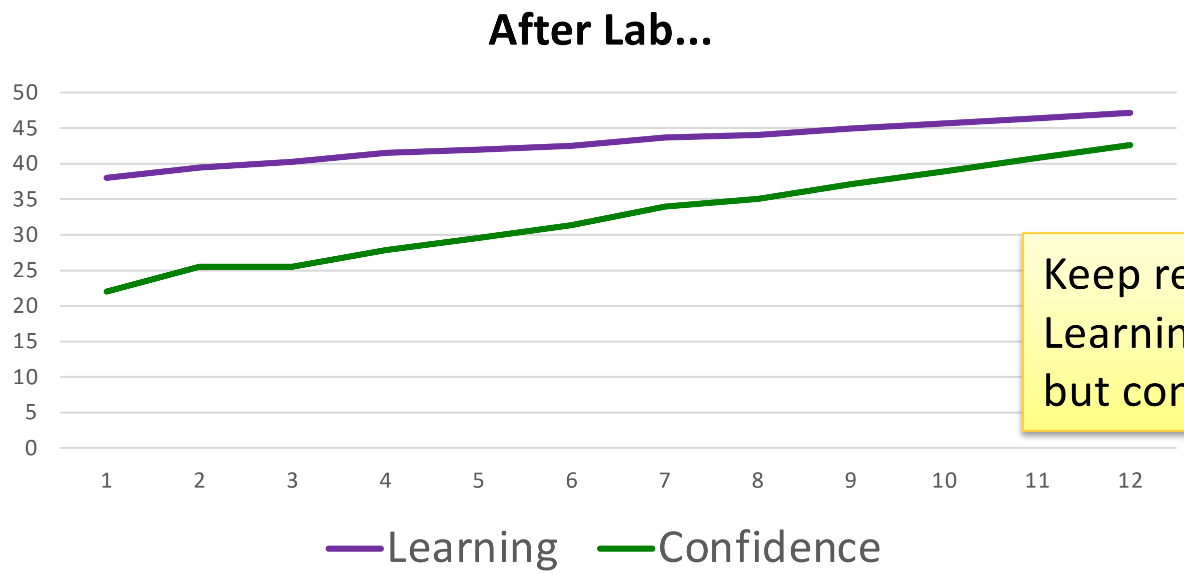
- But struggle affects your confidence
  - Confidence != Learning



Lab ends and … ???

Your Learning Journey

- But struggle affects your confidence
  - Confidence != Learning



Keep reviewing, practicing
Learning may not increase as much, but confidence should

Lab Progression

1. Functions defined for you; you call them
2. Refactor code you already wrote/tested/debugged into a function that takes no parameters and doesn't return anything
3. Refactor code you already wrote/tested/debugged into a function that takes a parameter and returns something
   - Can programmatically test
4. Implement functions that return things within a module
   - Uses functionality from the random module
5. Bottom-up development of functions

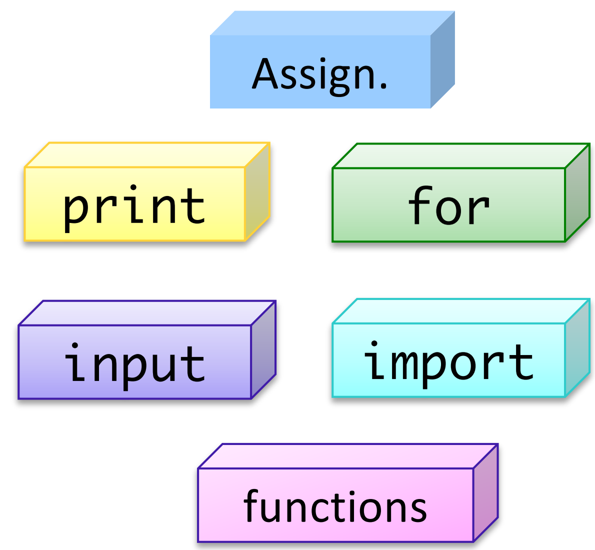
Justifications

- Why refactoring?
  - Common practice: write code, then realize it would be better (more readable, reusable, easier to test, …) if it were in a function

- Why test programmatically (when possible)?
  - Test-driven development: think about what function should do first
  - Automatically execute test cases and verify that the actual returned result is what we expected
    - No user input required!
    - Can rerun quickly/efficiently if implementation changes

Course Progression: Building Blocks

- Adding to your tool set
- We can combine them to create more complex programs
  - Solutions to problems



Evolving General Design Patterns

- General design pattern:
  1. Optionally, get user input
  2. Do some computation
  3. Display results
- General design pattern **with functions**:
  1. Optionally, get user input
  2. Do some computation by calling **functions**, get results
  3. Display results

Python Visualizer https://pythontutor.com

Python 3.6
known limitations

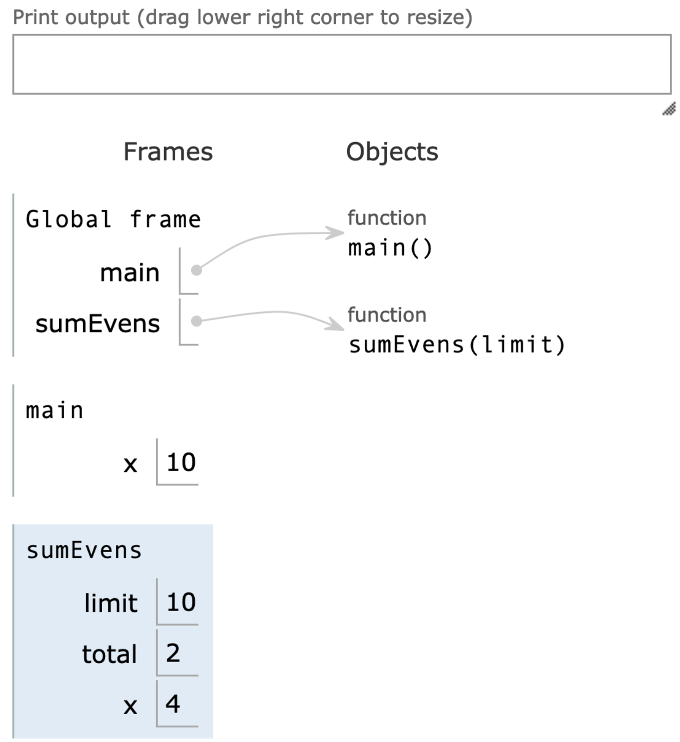
```
def main():
    x = 10
    sum = sumEvens( x )
    print("The sum of even #s up to", x, "is", sum

def sumEvens(limit):
    total = 0
    for x in range(0, limit, 2):
        total += x
    return total

main()
```

Edit this code

line that just executed
next line to execute

<< First < Prev Next > Last >>

Step 14 of 23



Parts of an Algorithm

- Input, Output
- Primitive operations
  - What data you have, what you can do to the data
- Naming
  - Identify things we're using
- Sequence of operations
- Conditionals
  - Handle special cases
- Repetition/Loops
- Subroutines
  - Call, reuse similar techniques

Making Decisions

- Sometimes, we do things only if some condition holds (i.e., “is true”)
- Examples
  - If the PB is new (has a safety seal)
    - Then, I will take off the safety seal
  - If it is raining and it is cold
    - Then, I will wear a raincoat
  - If it is Saturday or it is Sunday
    - Then, I will wake up at 9 a.m.
    - Otherwise, I wake up at 7 a.m.
  - If the shirt is purple or the shirt is on sale and blue
    - Then, I will buy the shirt

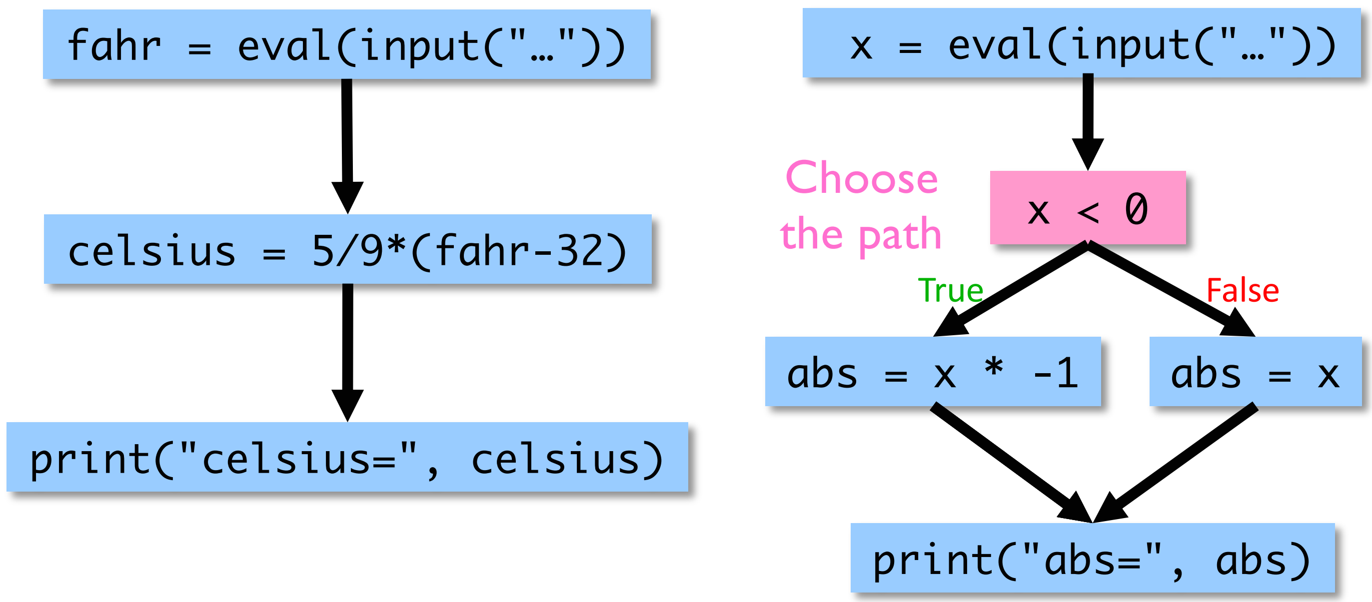
Conditionals

- Sometimes, we only want to execute a statement in certain cases
- Example: Finding the absolute value of a number
    - |4| = 4
    - |-10| = 10
  - To get the answer, we multiply the number by -1 *only if it's a negative number*
  - Code:

```
if x < 0 :
      abs = x*-1
```

`if` Statements

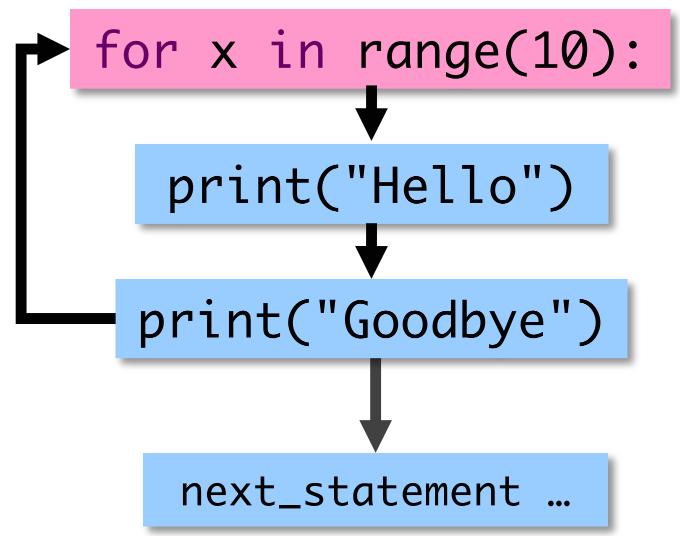
- Change the ***control flow*** of the program



Other Constructs That Change Control Flow

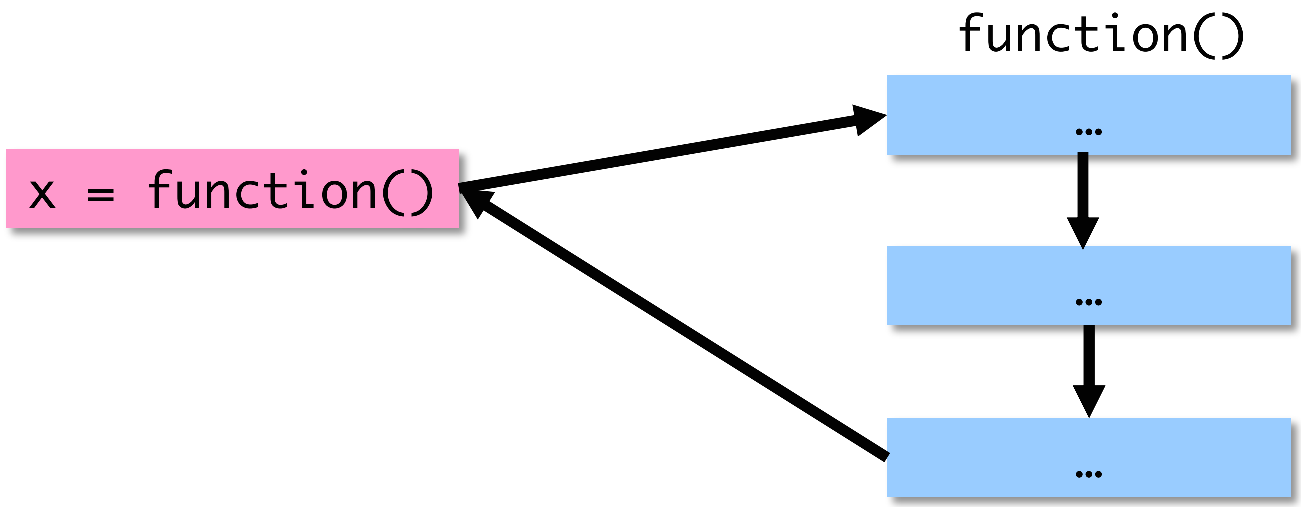
- **`for`** loops
  - Repeats a loop body a fixed number of times before going to the next statement after the **`for`** loop

```
for x in range(10):
      print("Hello")
      print("Goodbye")
next_statement …
```



Other Constructs That Change Control Flow

- Function calls
  - "Go execute some other code and then come back with the result"



Syntax of `if` statement: Simple Decision

```
if condition :
    statement1
    statement2
    …
    statementn
```

keyword

"then" Body

- Note indentation

English Examples:

`if` it is raining :
    I will wear a raincoat

`if` the PB is new :
    Remove the seal

Conditions

- Syntax (typical, others later):
  - <expr> <relational_operator> <expr>
- Evaluates to either `True` or `False`
  - Boolean type

Relational Operators

- Syntax: <expr> <relational_operator> <expr>
- Evaluates to either `True` or `False`
  - Boolean type

Low precedence
After arithmetic operators

Relational Operator	Meaning
<	Less than?
<=	Less than or equal to?
>	Greater than?
>=	Greater than or equal to?
==	Equals?
!=	Not equals?

Use Python interpreter

Example: Using Conditionals

- Determine if a number is even or odd

```
x = eval(input("Enter a number: "))
remainder = x % 2
if remainder == 0 :
    print(x, "is even")
if remainder == 1:
    print(x, "is odd")
```

 evenorodd.py

Common Mistake:
Assignment Operator vs. Equality Operator

- Assignment operator: =
- Equality operator: ==

```
x = eval(input("Enter a number: "))
remainder = x%2
if remainder = 0 :
        print(x, "is even.")
```

Syntax error

Syntax of `if` statement: Two-Way Decision

keywords

```
if condition :
    statement1
    statement2
    …
    statementn
else :
    statement1
    statement2
    …
    statementn
```

“then” Body

“else” Body

English Example:

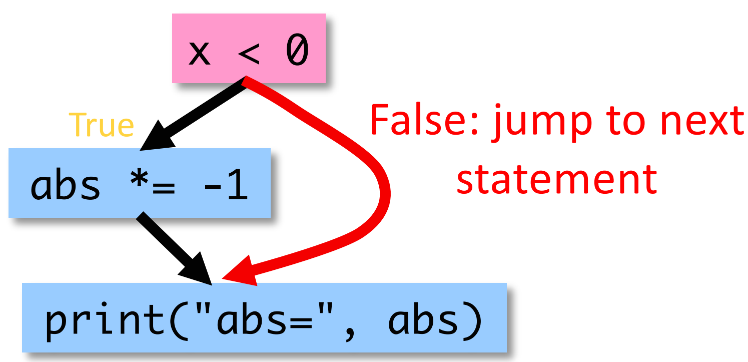
if it is Saturday or it is Sunday :
    I wake up at 9 a.m.
else :
    I wake up at 7 a.m.

If-Else statements (absolute values)

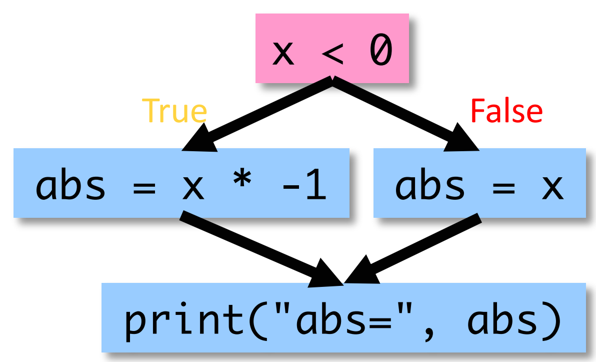
```
abs = x
if x < 0 :
    abs *= -1
print("abs=", abs)
```

```
if x < 0 :
    abs = x * -1
else:
    abs = x
print("abs=", abs)
```

If statement



If-else statement



Example: Using Conditionals

- Determine if a number is even or odd
- More efficient implementation
  - Don’t need to check if remainder is 1 because if it’s not 0, it *must* be 1

```
x = eval(input("Enter a number: "))
remainder = x % 2
if remainder == 0:
      print(x, "is even")
else:
      print(x, "is odd")
```

Practice: Draw the Flow Chart

```
print("This program determines your birth year")
print("given your age and current year")
print()
age = eval(input("Enter your age: "))

if age > 120:
    print("Don't be ridiculous, you can't be that old.")
else:
    currentYear = eval(input("Enter the current year: "))
    birthyear = currentYear - age
    print()
    print("You were either born in", birthyear, end=' ')
    print("or", birthyear-1)
print("Thank you.  Come again.")
```

What does this code do?

Flow of Control

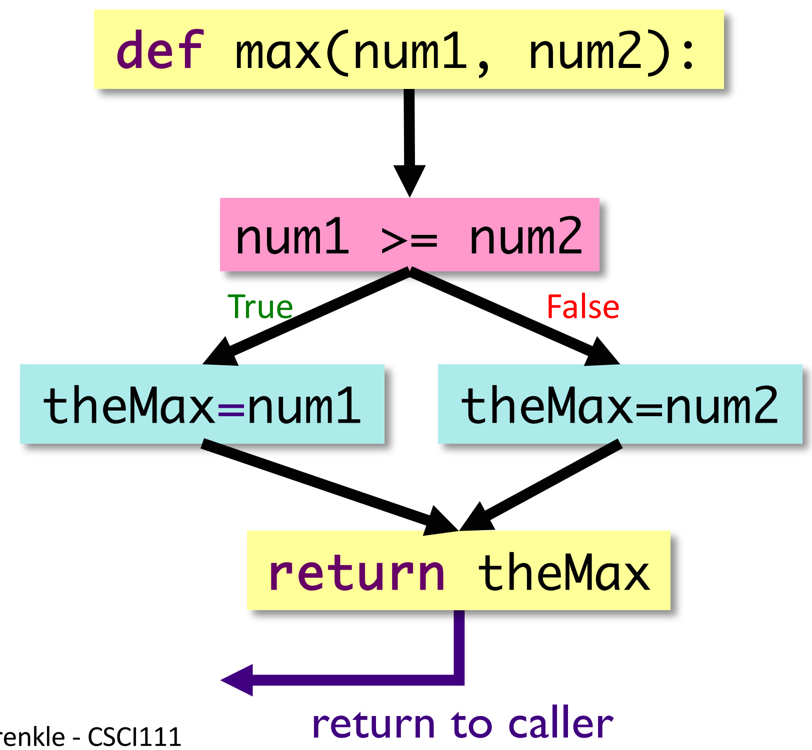
- `max`: Given two numbers, returns the greater number

Is this implementation of the function correct?

```
def max(num1, num2):
    if num1 >= num2:
          theMax = num1
    else:
          theMax = num2
    return theMax
```

Flow of Control

```
def max(num1, num2):
    if num1 >= num2:
        theMax = num1
    else:
        theMax = num2
    return theMax
```



Flow of Control: Using `return`

- `max`: Given two numbers, returns the greater number

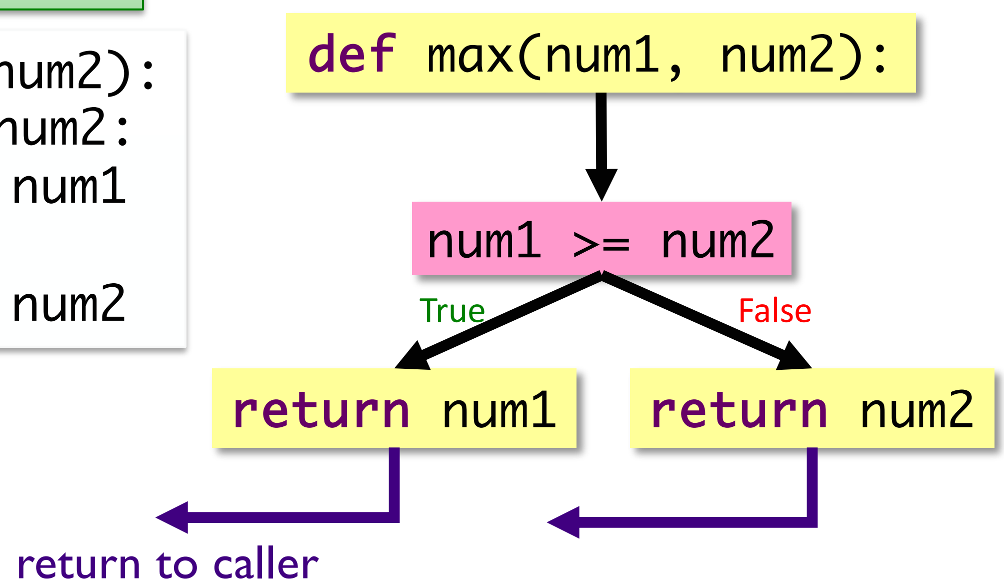
Is this implementation of the function correct?

```
def max(num1, num2):
    if num1 >= num2:
        return num1
    else:
        return num2
```

Flow of Control: Using **return**

Is this implementation of the function correct?

```
def max(num1, num2):
    if num1 >= num2:
        return num1
    else:
        return num2
```



Flow of Control: Using `return`

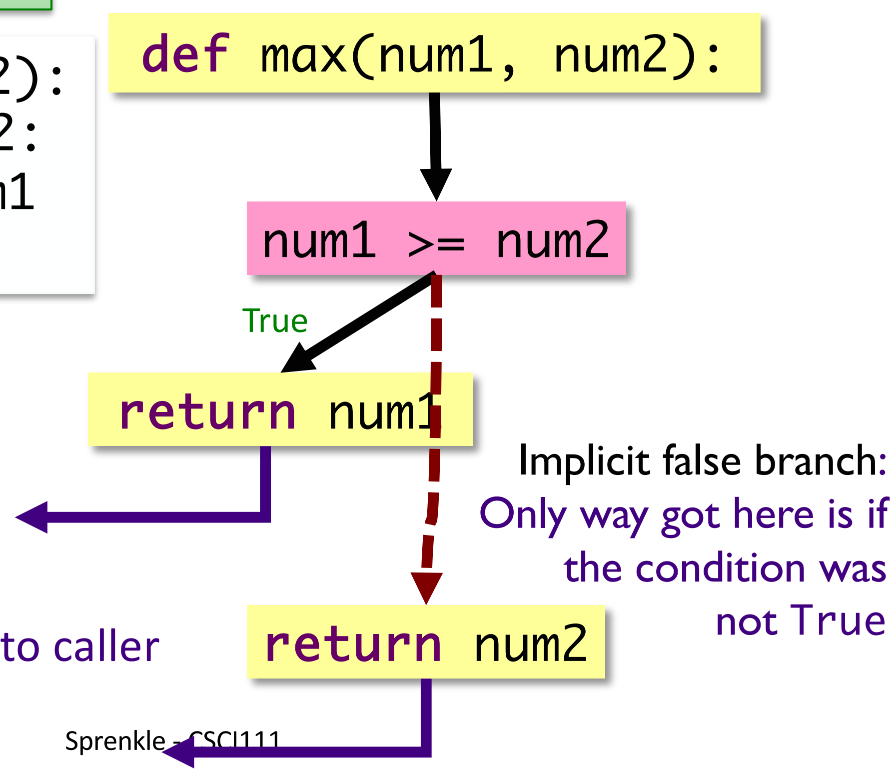
Is this implementation of the function correct?

```
def max(num1, num2):
    if num1 >= num2:
        return num1
    return num2
```

Flow of Control: Using `return`

Is this implementation of the function correct?

```
def max(num1, num2):
    if num1 >= num2:
        return num1
    return num2
```



Practice: Speeding Ticket Fines

- Any speed clocked over the limit results in a fine of at least $50, plus $5 for each mph over the limit, plus a penalty of $200 for any speed over 90mph.
- Our function
  - Input: speed limit and the clocked speed
  - Output: the appropriate fine
    - What should the appropriate fine be if the user is not speeding?
- Write test cases first!

`speedingticket.py`

Exam Friday

Change in today's office hours: 11:30-1:30
Friday: I will be observing another class

- In-class, on paper
  - Emphasis on critical thinking
- Exam Preparation Document is on course web page
- Similar problems to class and lab
  - Review questions
  - Worksheets
  - Problems
- Content: up through Lab 4
- No broader issue this week

Looking Ahead

- Lab 4
  - Practicing ***functions***
  - Due Friday
- Exam Friday
- No broader issue this week