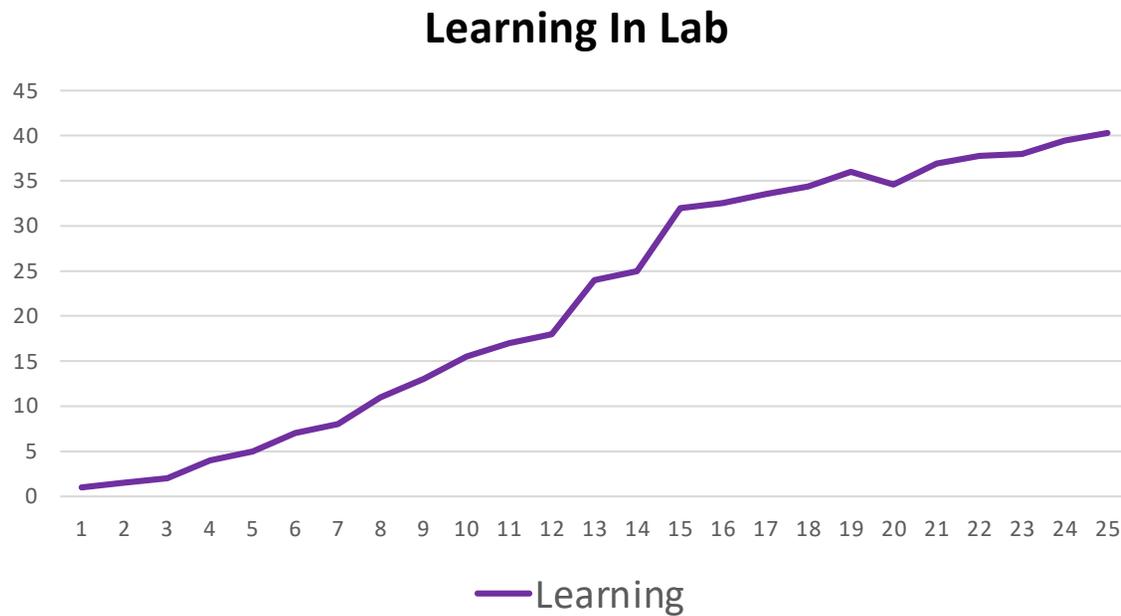


Objectives

- Conditionals
- Exam review

Your Learning Journey

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

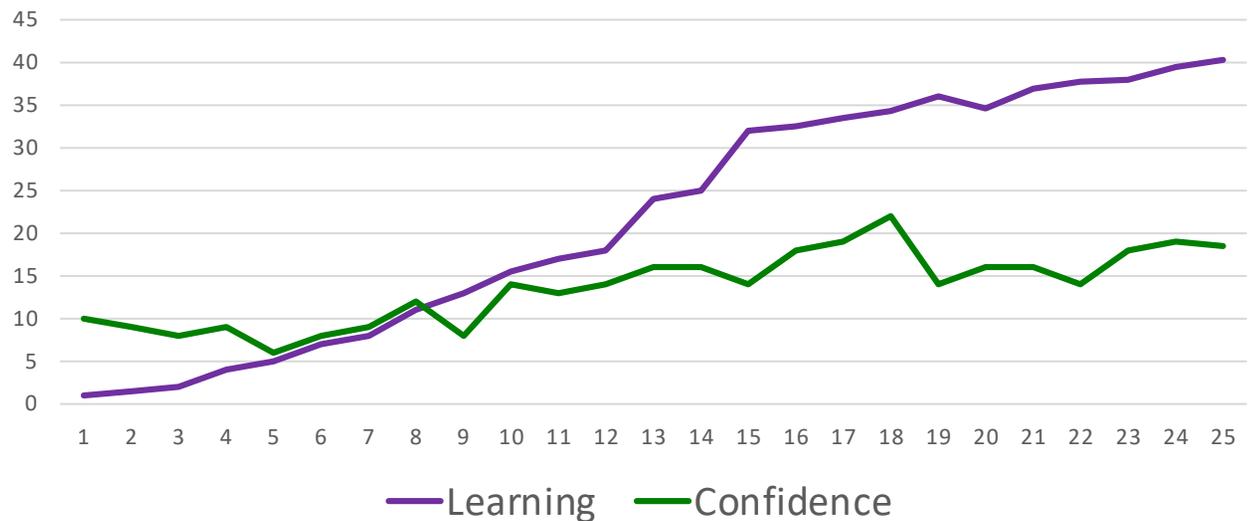


Your Learning Journey

- But struggle affects your confidence

➤ Confidence \neq Learning

Learning vs Confidence in Lab



Lab ends and ... ???

Your Learning Journey

- But struggle affects your confidence

➤ Confidence != Learning

After Lab...



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

Lab Progression

1. Functions defined for you; you call them
 - Learn how to read docstrings; more practice with Graphics API; functions vs methods
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
 - Testing floats

Python Visualizer

<https://pythontutor.com>

```
Python 3.6  
known limitations  
1 def main():  
2     x = 10  
3     sum = sumEvens( x )  
4     print("The sum of even #s up to", x, "is", sum  
5  
6 def sumEvens(limit):  
7     total = 0  
8     for x in range(0, limit, 2):  
9         total += x  
10    return total  
11  
12 main()
```

[Edit this code](#)

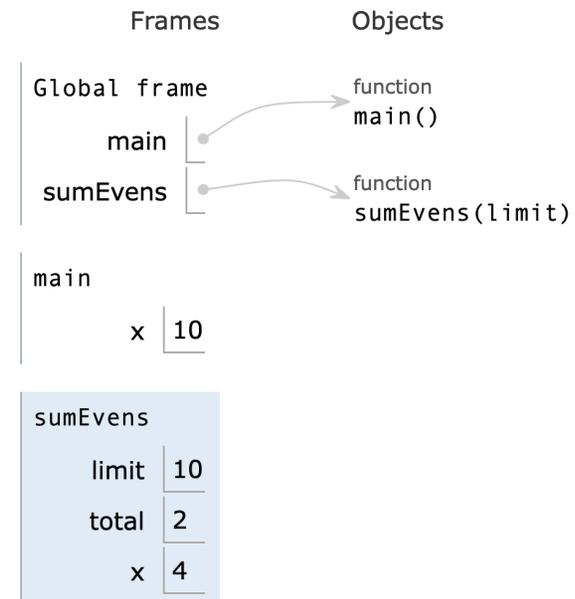
→ line that just executed
→ next line to execute



<< First < Prev Next > Last >>

Step 14 of 23

Print output (drag lower right corner to resize)

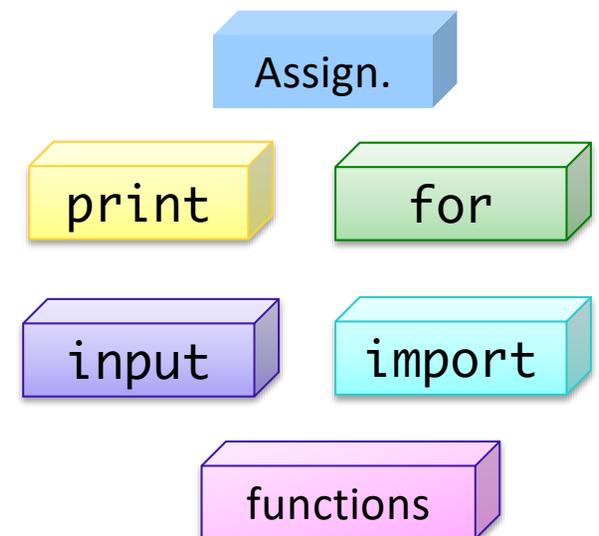


Rationale

- Why refactoring?
 - Common practice: write code, then realize it would be better (more readable, reusable, easier to test, ...) if it were in a function
 - Already worked out the logic – focus on “function-ifying” the code
- 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



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

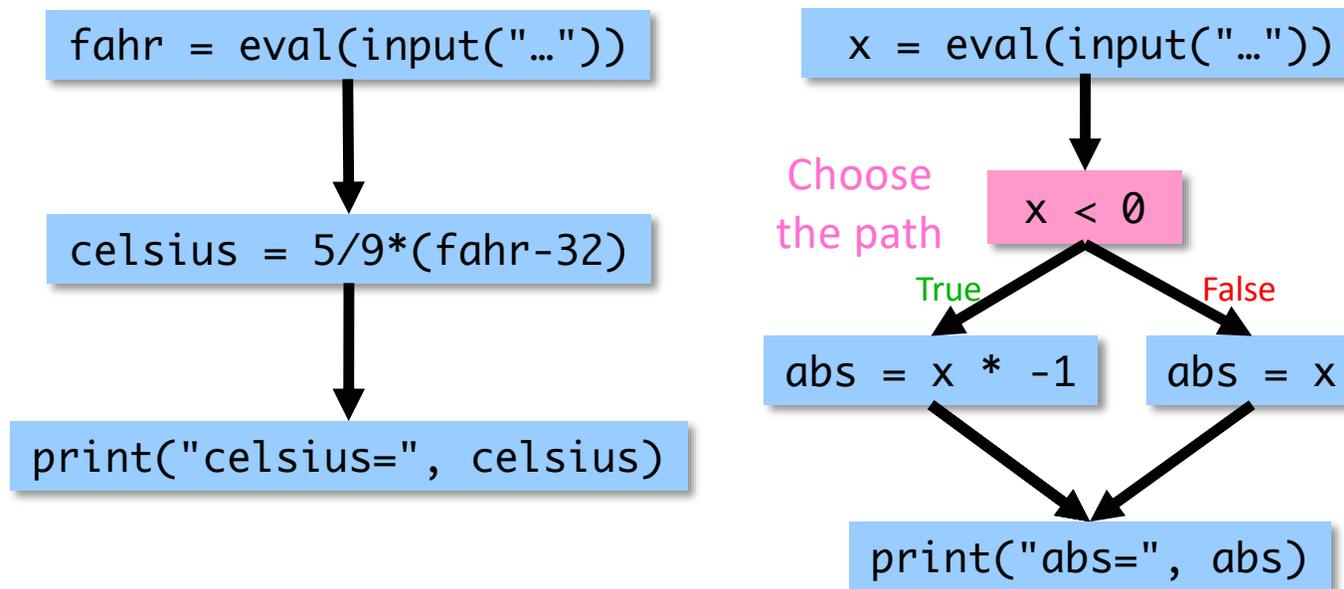
- Sometimes, we only want to execute a statement in certain cases
- Example: Finding the absolute value of a number
 - $|4| = 4$
 - $|-10| = 10$
- To get the answer, we multiply the number by -1 *only if it's a negative number*

➤ Code:

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

if Statements

- Change the *control flow* of the program

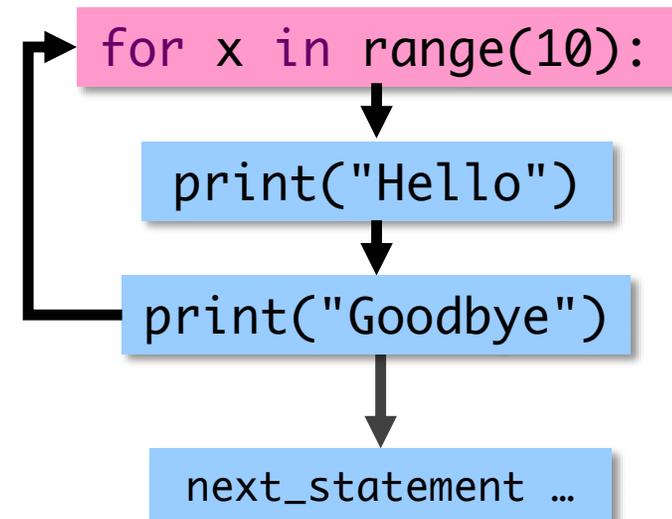


Other Constructs That Change Control Flow

- **for** loops

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

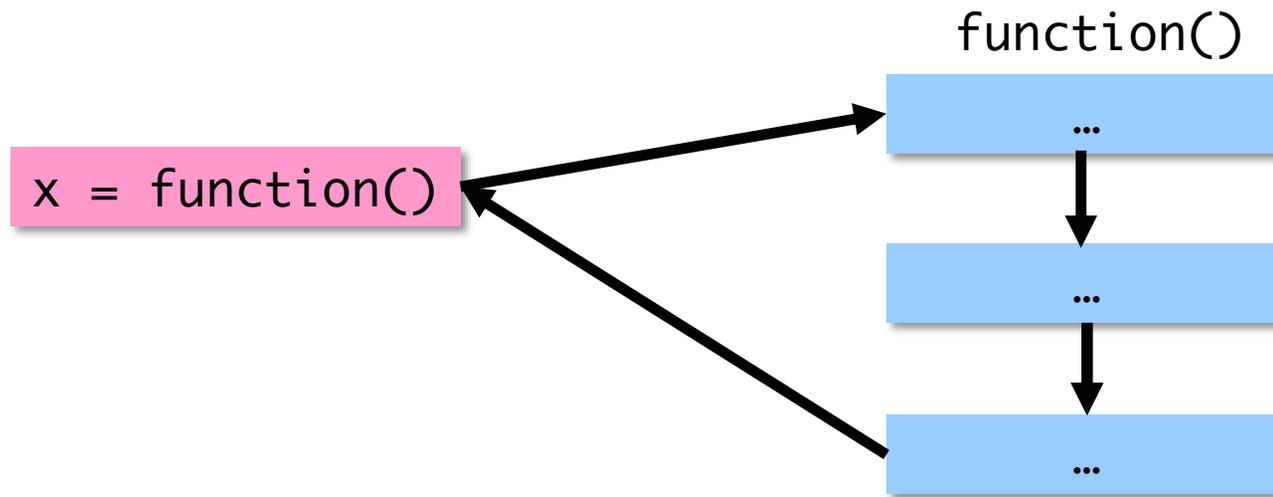
```
for x in range(10):  
    print("Hello")  
    print("Goodbye")  
next_statement ...
```



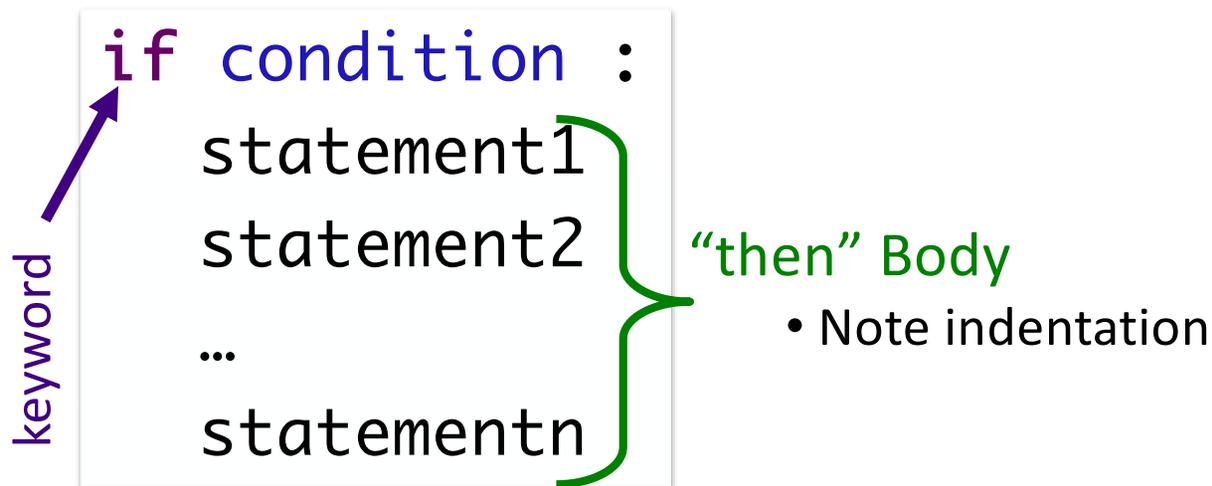
Other Constructs That Change Control Flow

- Function calls

- “Go execute some other code and then come back with the result”



Syntax of `if` statement: Simple Decision



English Examples:

`if` it is raining :
 I will wear a raincoat
`if` the PB is new :
 Remove the seal

Conditions

- Syntax (typical; variations 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

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

Low precedence
After arithmetic operators

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")
```

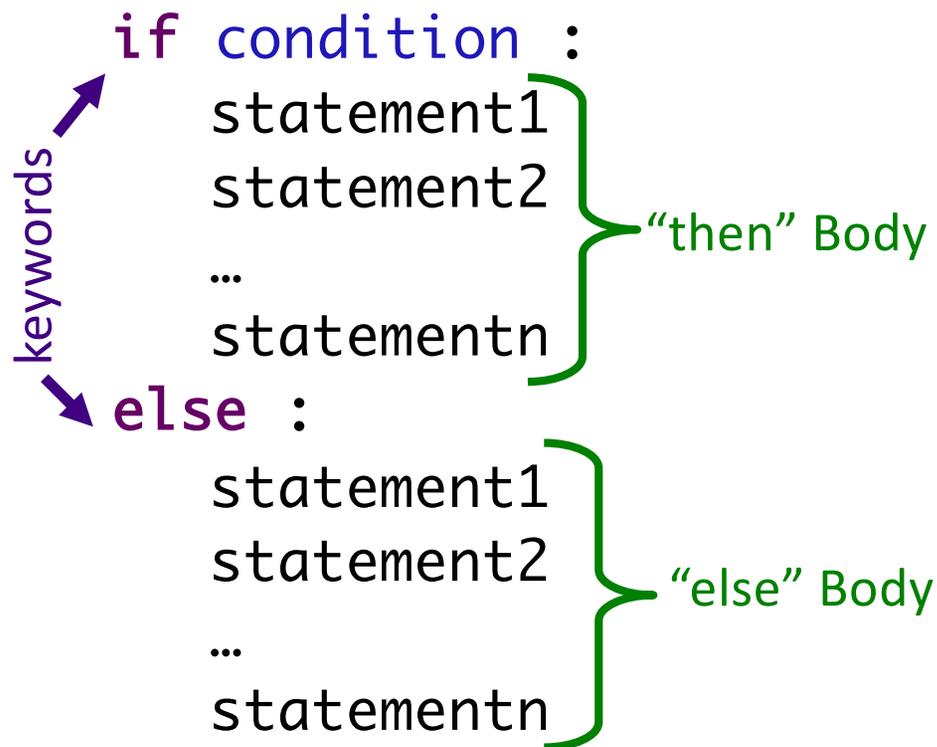
Common Mistake: Assignment Operator vs. Equality Operator

- 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



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.

Example: Two-Way Decision

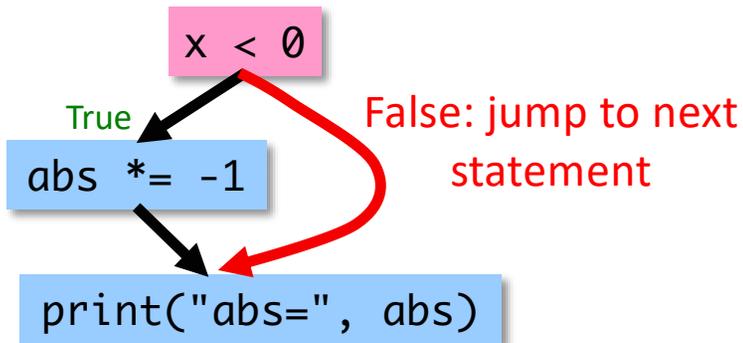
- Determine if a number is even or odd

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

Comparing Conditional Structures

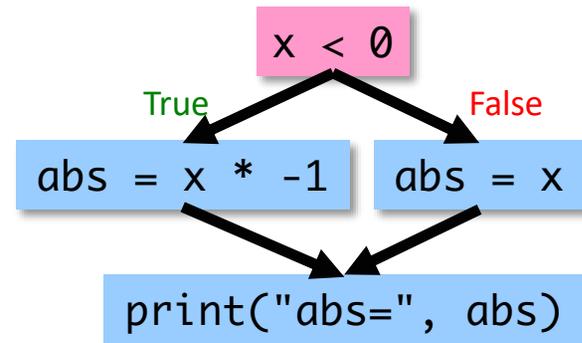
```
abs = x
if x < 0 :
    abs *= -1
print("abs=", abs)
```

If statement



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

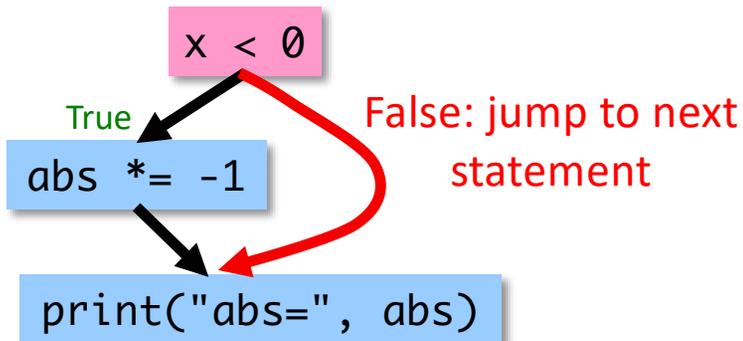
If-else statement



Comparing Conditional Structures

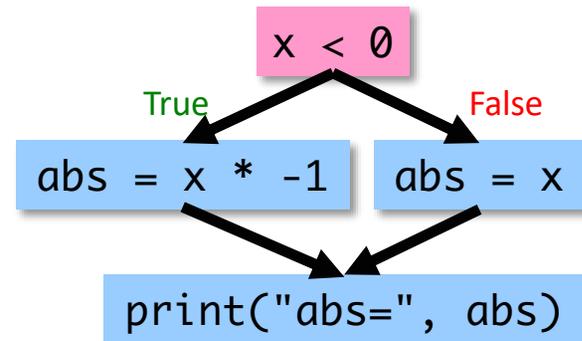
```
abs = x
if x < 0 :
    abs *= -1
print("abs=", abs)
```

If statement



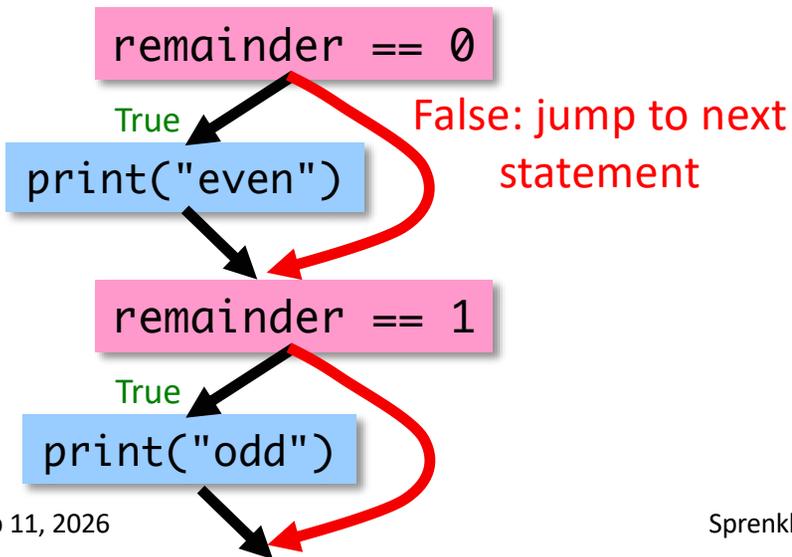
```
if x < 0 :
    abs = x * -1
else:
    abs = x
print("abs=", abs)
```

If-else statement

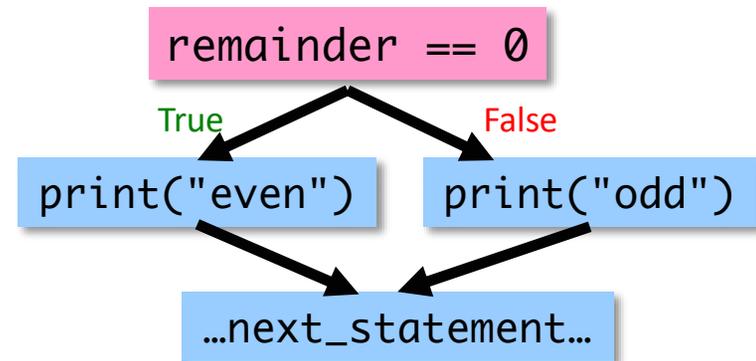


Comparing Conditional Structures

```
remainder = x % 2
if remainder == 0 :
    print("even")
if remainder == 1:
    print("odd")
```

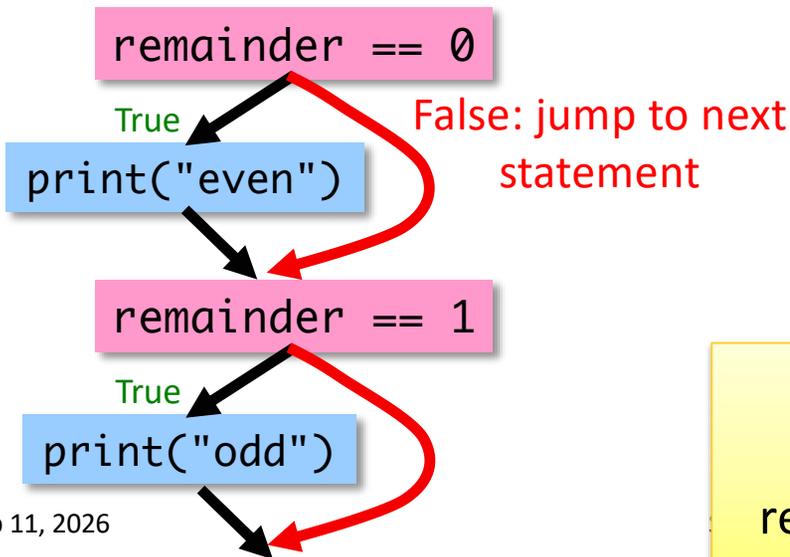


```
remainder = x % 2
if remainder == 0 :
    print("even")
else:
    print("odd")
```

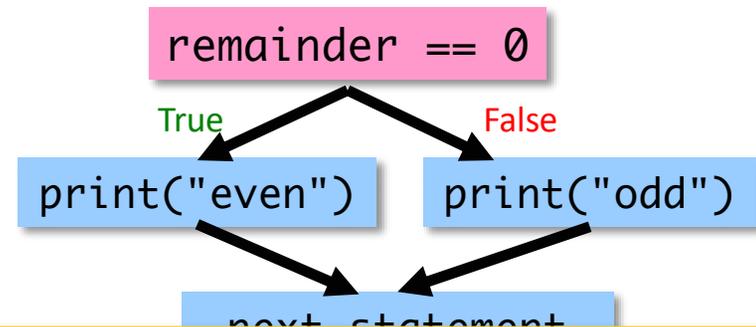


Comparing Conditional Structures

```
remainder = x % 2
if remainder == 0 :
    print("even")
if remainder == 1:
    print("odd")
```



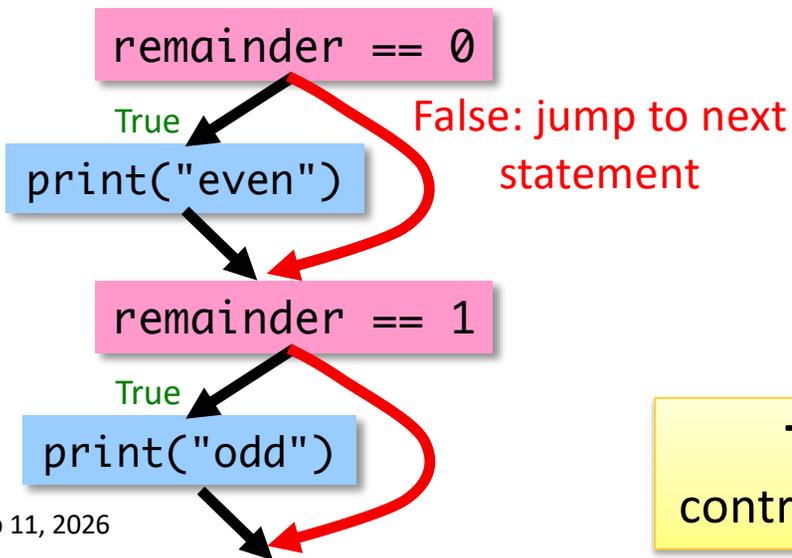
```
remainder = x % 2
if remainder == 0 :
    print("even")
else:
    print("odd")
```



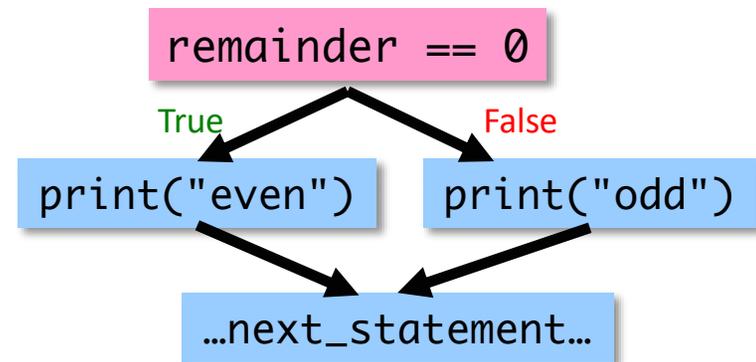
Both solutions produce the same output. But, left solution will **always** check if remainder is 1. Computer does more work!

Comparing Conditional Structures

```
remainder = x % 2
if remainder == 0 :
    print("even")
if remainder == 1:
    print("odd")
```



```
remainder = x % 2
if remainder == 0 :
    print("even")
else:
    print("odd")
```



Takeaway: Translate written code to control flow to understand program efficiency

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.")
```

1. Draw the flow chart.
2. State the program's output

Flow of Control

- max: Given two numbers, returns the greater number

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

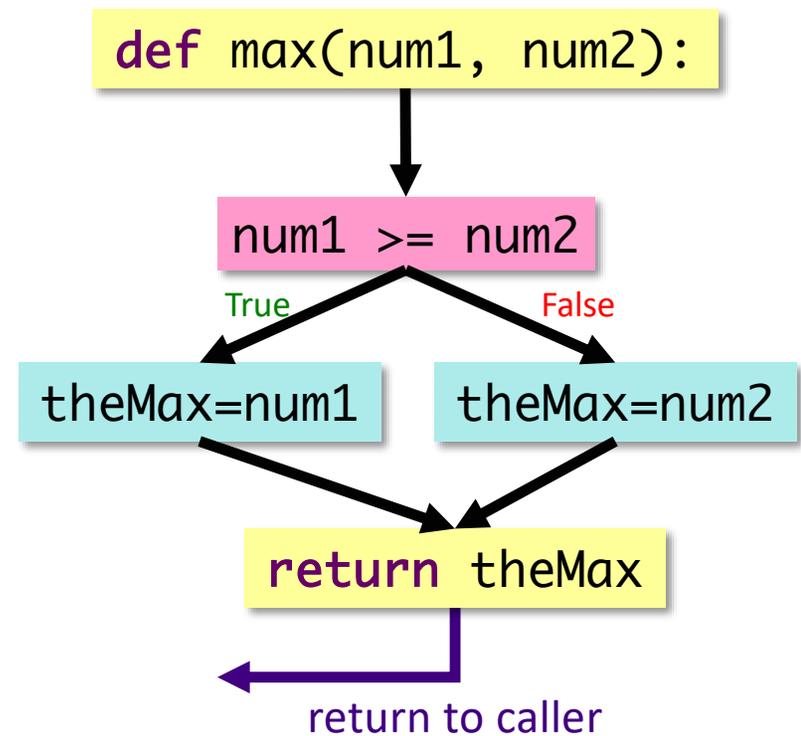
Is this implementation of the function correct?
How are you determining if it is correct?

Flow of Control

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

Check with test cases, for example, ...

Num1	Num2	Expected Output
1	2	2
2	1	2
2	2	2



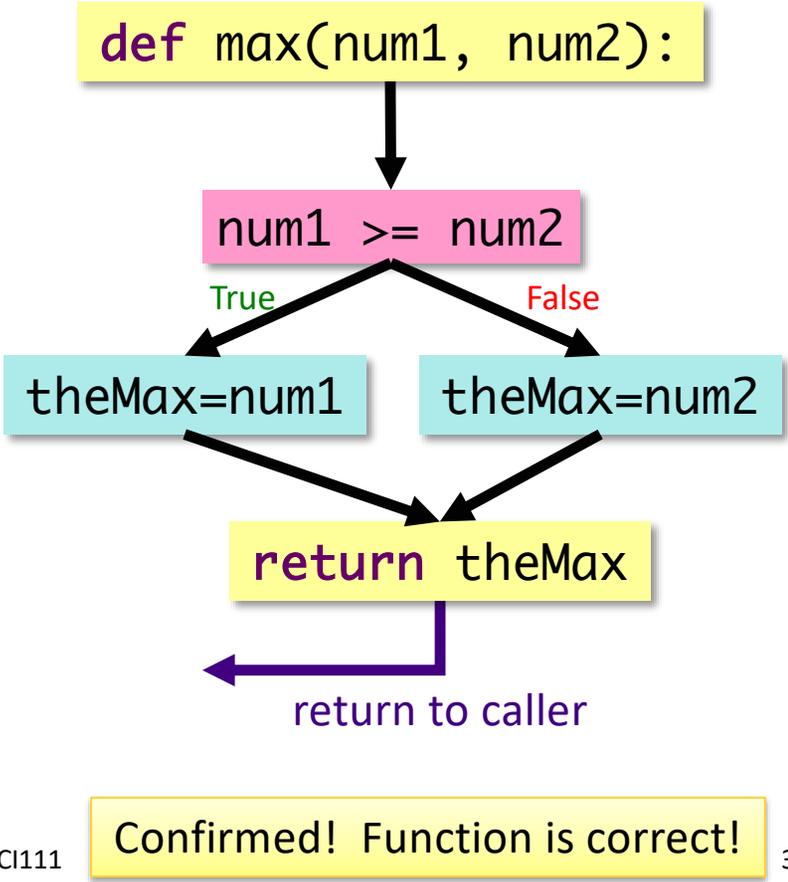
Flow of Control

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

Check with test cases, for example, ...

Num1	Num2	Expected Output
1	2	2
2	1	2
2	2	2

Feb 11, 2026



Sprenkle - CSCI111

Flow of Control

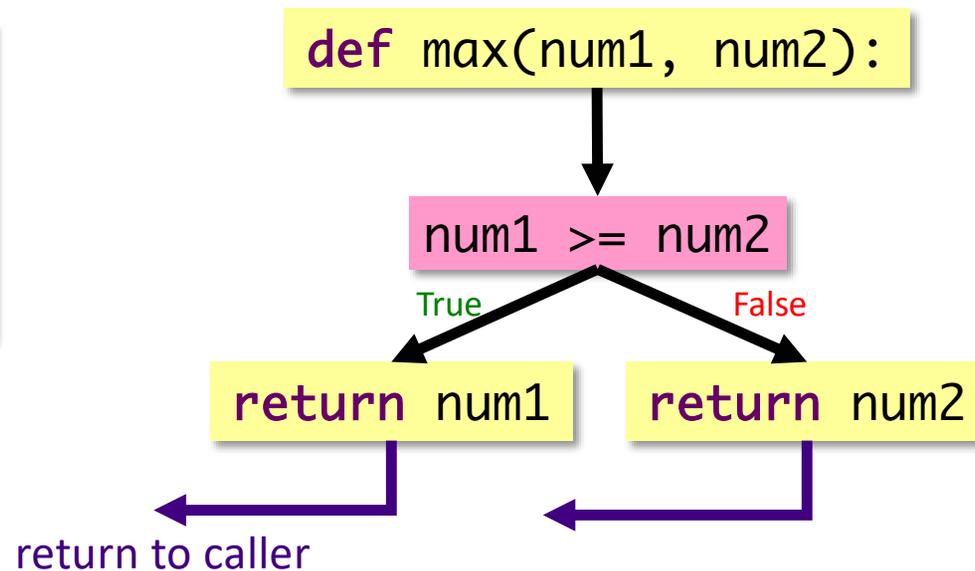
- max: Given two numbers, returns the greater number

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

Is this implementation of the function correct?

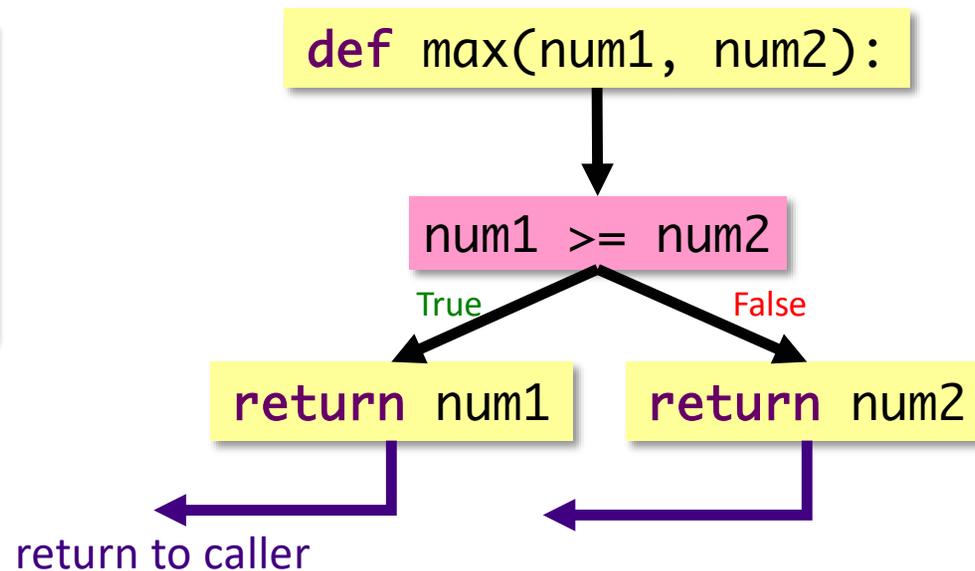
Flow of Control: Using return

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



Flow of Control: Using `return`

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



Is this implementation of the function correct? **Yes!**

Flow of Control: Using **return**

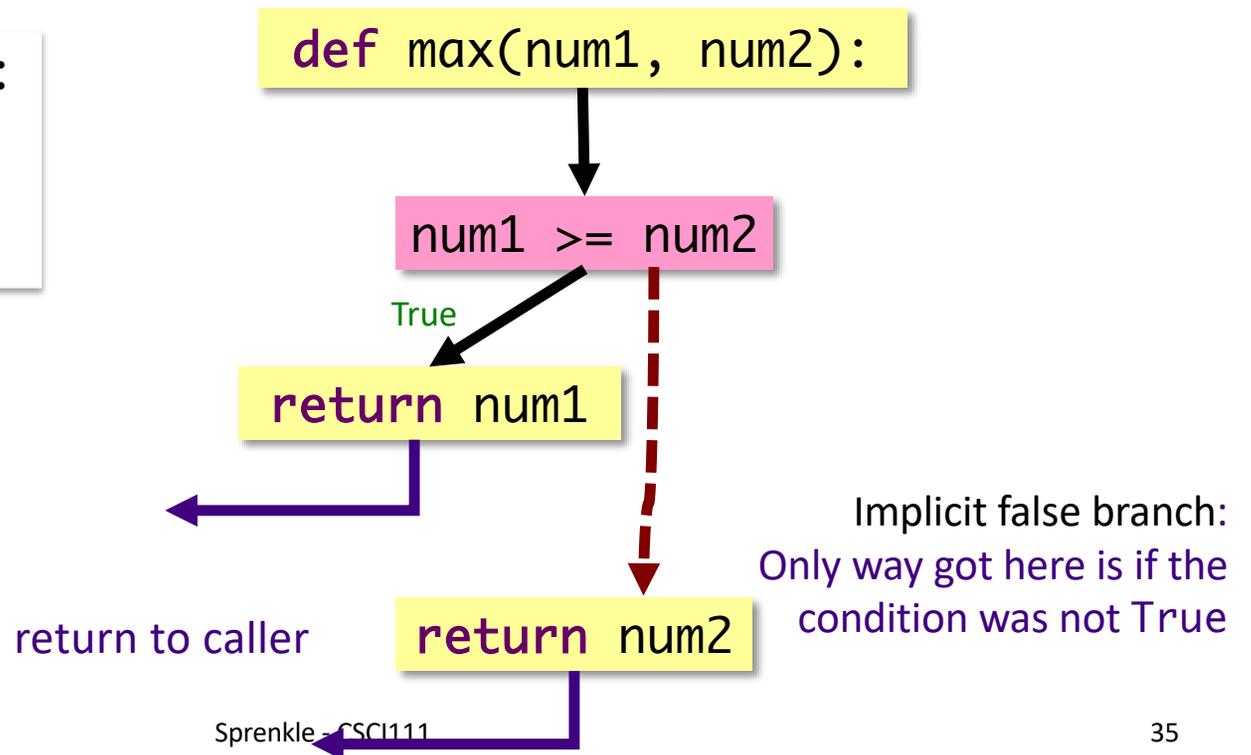
- **max**: Given two numbers, returns the greater number

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

Is this implementation of the function correct?

Flow of Control: Using `return`

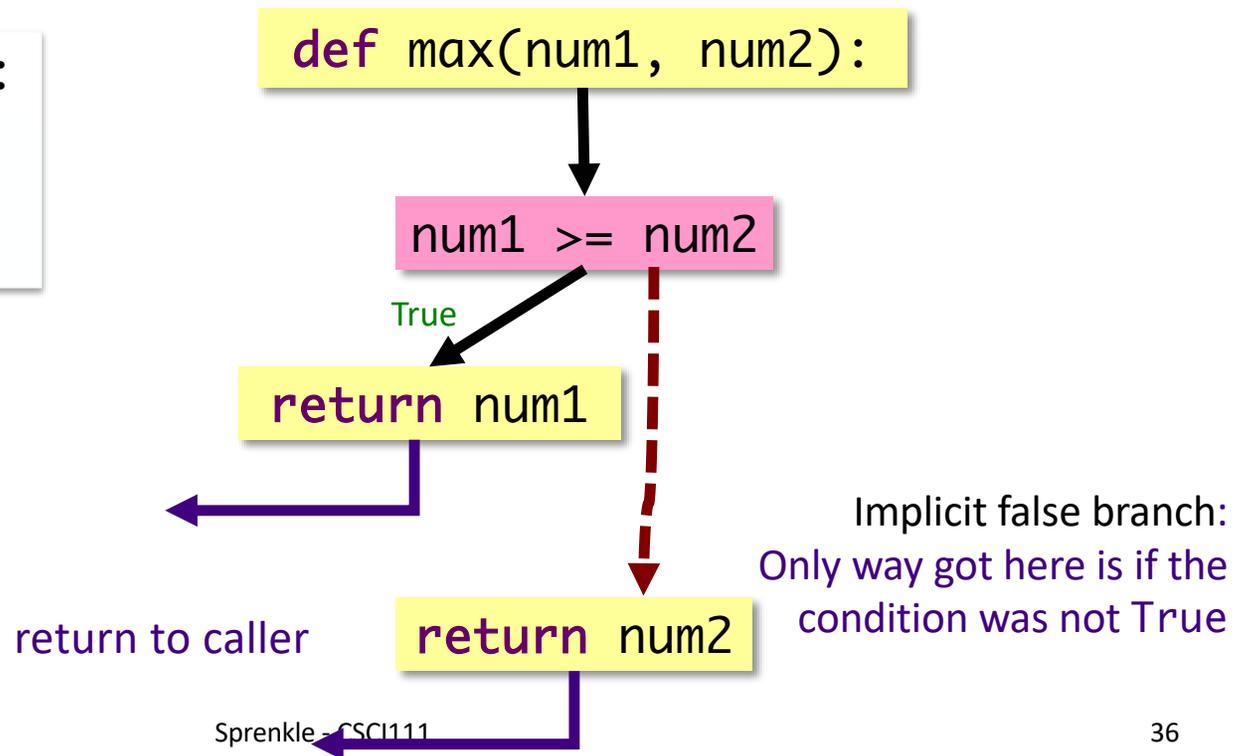
```
def max(num1, num2):  
    if num1 >= num2:  
        return num1  
    return num2
```



Flow of Control: Using **return**

Is this implementation of the function correct? **Yes!**

```
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 clocked speed
 - Output: the appropriate fine
 - What should the appropriate fine be if the user is not speeding?

Our process

1. Write test cases
2. Implement function
3. Test function

Test Cases for Speeding Ticket Function

Speed limit	Clocked Speed	Fine
50	50	0
50	51	55

Create test cases for the function's different expected behaviors

Exam Friday

- In-class, on paper
 - Emphasis on critical thinking
 - Practice writing with a pencil (preferably) or pen!
- Exam Preparation Document is on course web page
- Similar problems to class and lab
 - Review questions
 - Worksheets
 - Problems
- Content: up through Lab 4
 - (not today's content)

Looking Ahead

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