CS 134 Lecture 5: More Conditionals

Announcements & Logistics

- **Homework 2** is due tonight at 10 pm
- **Lab 2** was released on Friday
    - Pre-lab due at the beginning of lab
    - Full Assignment due Wed/Thur 10 pm
- You can work on lab machines any time (when there's not a class)
- Make sure to keep your work consistent with what is on `evolene`
    - Always pull/clone when you start and add, commit and push to evolene when done with a work session

Do You Have Any Questions?

Last Time

- Wrapped up functions
  - Discussed return statements and variable **scope**
- Started learning about **conditionals**
  - **Boolean** data type
  - Making decisions in Python using `if else` statements

Today's Plan

- Learn more about `if else` statements
- Look at more complex decisions in Python
    - Boolean expressions with **and**, **or**, **not**
- Choosing between many different options in our code
    - `if elif else` "chained" conditionals
- Using `import` for using functions across different `.py` files
- We are going to cover a lot of material in the next 3 lectures
    - Make sure you are keeping up and getting help if needed!

Boolean Expressions and If Statement

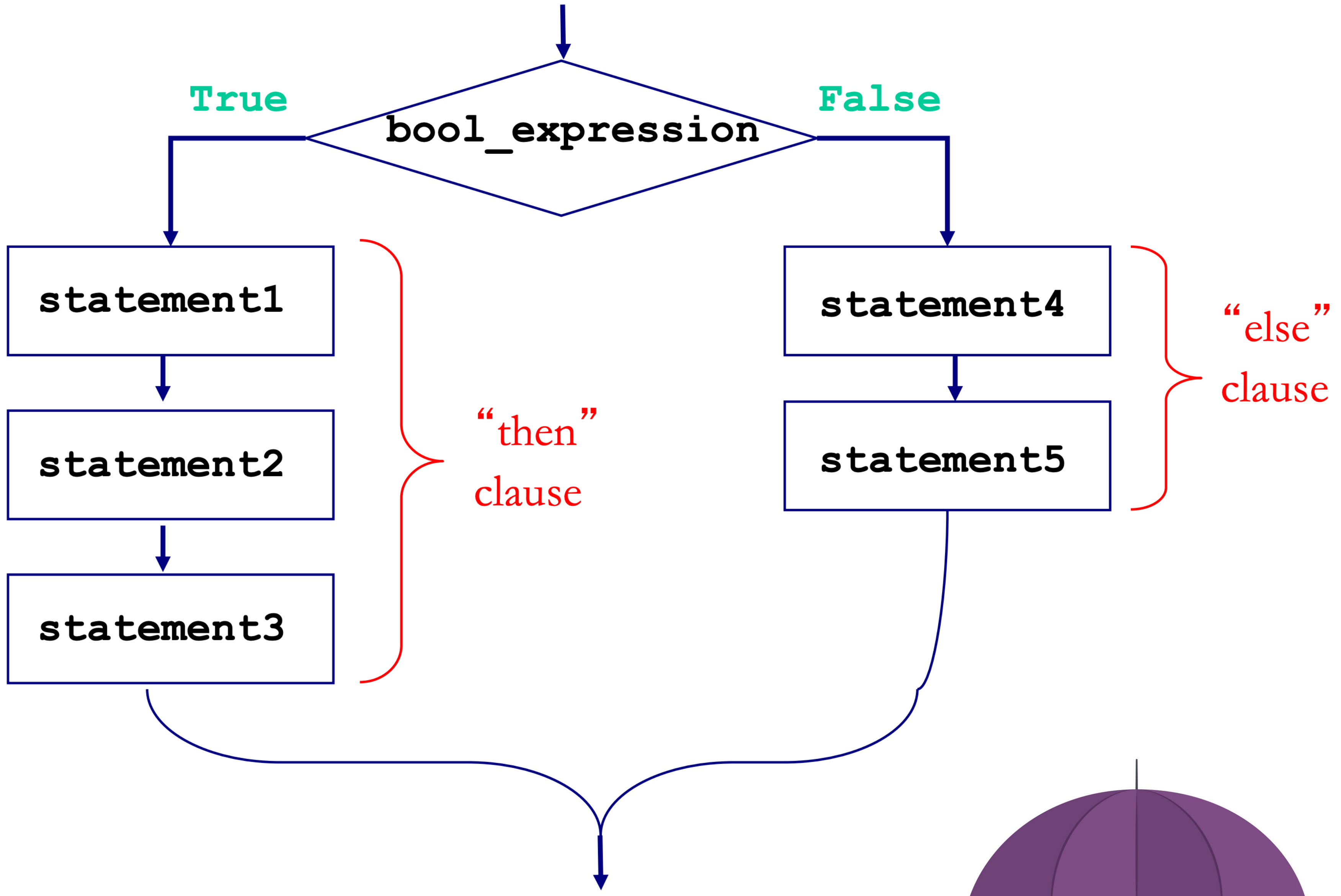
- Python expressions that result in a `True/False` output are called **boolean expressions**
  - For example, checking if a user's entered number, **`num`**, is even
- How do we do this? (What is a property of even numbers that we can use to test this condition?)
    - Even numbers are evenly divisible by 2 (remainder of zero)
    - Thus, **`num % 2`** should be zero if and only if **`num`** is even
- Now we have a Boolean expression we can test for: **`num % 2 == 0`**
- We can implement "conditional statements" in Python using Boolean expressions and an **if-else statement**

`is_even.py`

Python Conditionals (`if` Statements)

```
if <boolean expression>:
        statement1
        statement2
        statement3
else:
        statement4
        statement5
```

Note: (syntax) Indentation and colon after if and else



If it is raining, then bring an umbrella. Else, bring your sunglasses.

Optional Else & Simplifying Conditionals

- The else block is **optional**: not a requirement (not always needed!)
- Sometimes we can simplify conditionals
    - For example, all three below are equivalent inside the body of a function that returns `True` if num is even, and `False` otherwise

```
if num % 2 == 0:
    return True
else:
    return False
```

$\rightarrow$

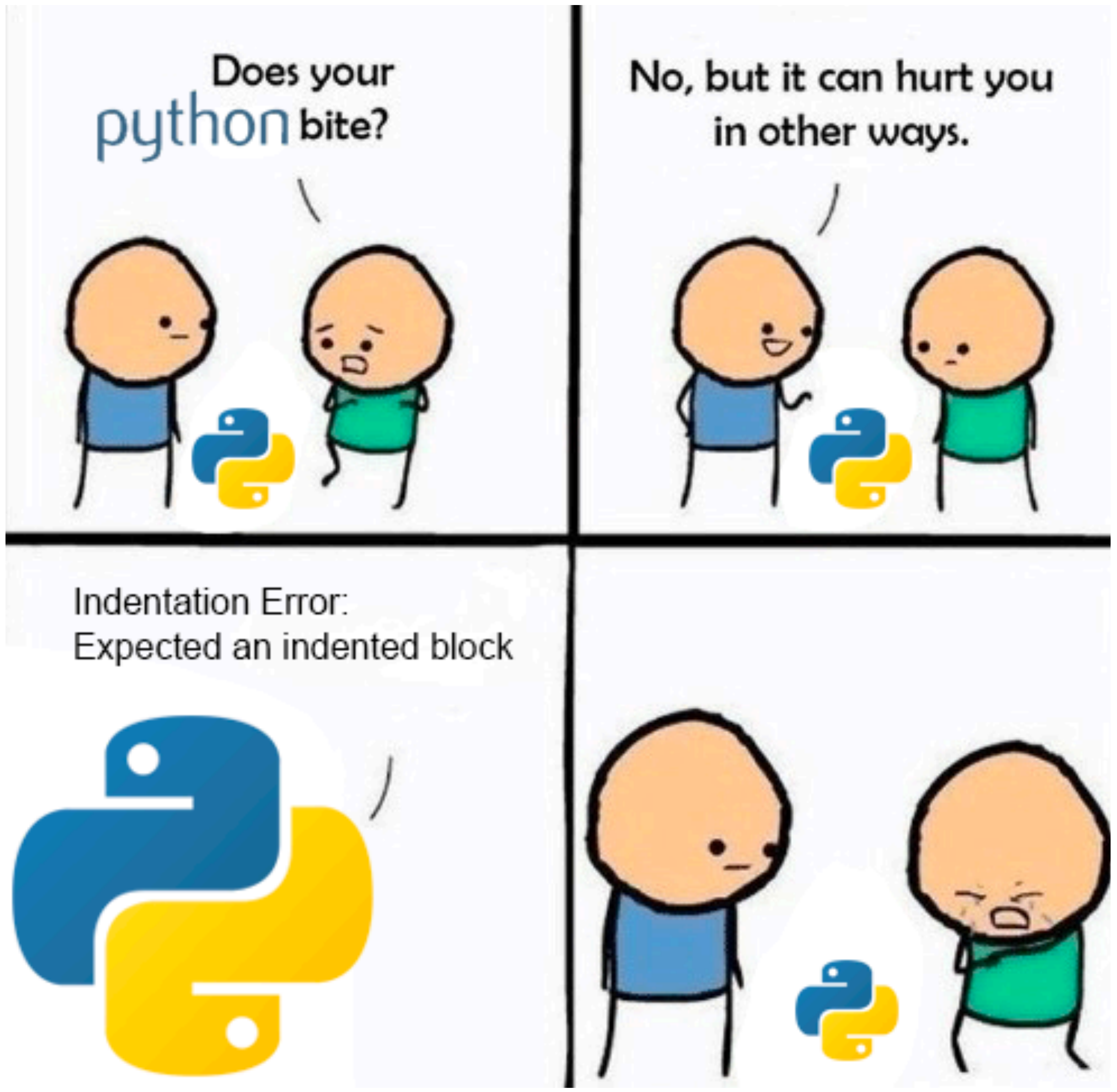
```
if num % 2 == 0:
    return True
return False
```

$\rightarrow$

```
return num % 2 == 0
```

Python Conditionals (`if` Statements)

- Don't forget proper indentation!


Does your python bite?
No, but it can hurt you in other ways.
Indentation Error:
Expected an indented block

(Credit to u/ufoludek_ on r/ProgrammerHumor)

Let's See Some Examples

More Decisions

- Sometimes, we need a more complicated conditional structure with more than 2 options but exactly one option is possible
- Example: Write a function that takes a temp value in Fahrenheit
    - If temp is above 80, print "It is a hot one out there."
    - If temp is between 60 and 80, print "Nice day out, enjoy!"
    - If temp is below 60, print "Chilly day, don't forget a jacket."
- Notice that temp **can only be in one of those** ranges
    - If we find that temp is greater than 80, no need to check the rest!

Nested Conditionals

```
if booleanExpression1:
    statement 1
    ...
else:
    if booleanExpression2:
        statement 2
        ...
    else:
        statement 3
        ...
```

Attempt 1: Chained Conditionals

- We can **nest** if-else statements (using indentation to distinguish between matching if-else blocks)
- Works, but this can quickly become unnecessarily complex (and hard to read!) The code below is an example of what **NOT** to do!

```python
def weather1(temp):
    if temp > 80:
        print("It is a hot one out there.")
    else:
        if temp >= 60:
            print("Nice day out, enjoy!")
        else:
            if temp >= 40:
                print("Chilly day, wear a sweater.")
            else:
                print("Its freezing out, bring a winter jacket!")
```

Logical Operators

- Logical operators **`and`**, **`or`**, **`not`** are used to combine Boolean values
- For two Boolean expressions **`exp1`** and **`exp2`**
  - **`not exp1`** (! in other languages) returns the opposite of `exp1`
  - **`exp1 and exp2`** (&& in other languages) is **`True`** iff **`exp1`** and **`exp2`** are **`True`**
  - **`exp1 or exp2`** (|| in other languages) is **`True`** iff either **`exp1`** or **`exp2`** are **`True`**

Truth Table for or

exp1	exp2	exp1 or exp2
True	True	**True**
True	False	**True**
False	True	**True**
False	False	**False**

Truth Table for and

exp1	exp2	exp1 and exp2
True	True	**True**
True	False	**False**
False	True	**False**
False	False	**False**

Attempt 2: Sequence of Ifs

- What if we use a bunch of if statements (w/o else) one after the other to solve this problem?
- What are the advantages/disadvantages of this approach?

```python
def weather2(temp):
    if temp > 80:
        print("It is a hot one out there.")
    if temp >= 60 and temp <= 80:
        print("Nice day out, enjoy!")
    if temp <60 and temp >= 40:
        print("Chilly day, wear a sweater")
    if temp < 40:
        print("Its freezing out, bring a winter jacket!")
```

If Elif Else Statements

- Fortunately, there is a simpler way to specify several options by **chaining** conditionals

```
if booleanExpression1:
    statement 1
    ...
elif booleanExpression2:
    statement 2
    ...
else:
    statement 3
    ...
```

A better approach that avoids too many indented blocks and improves code readability

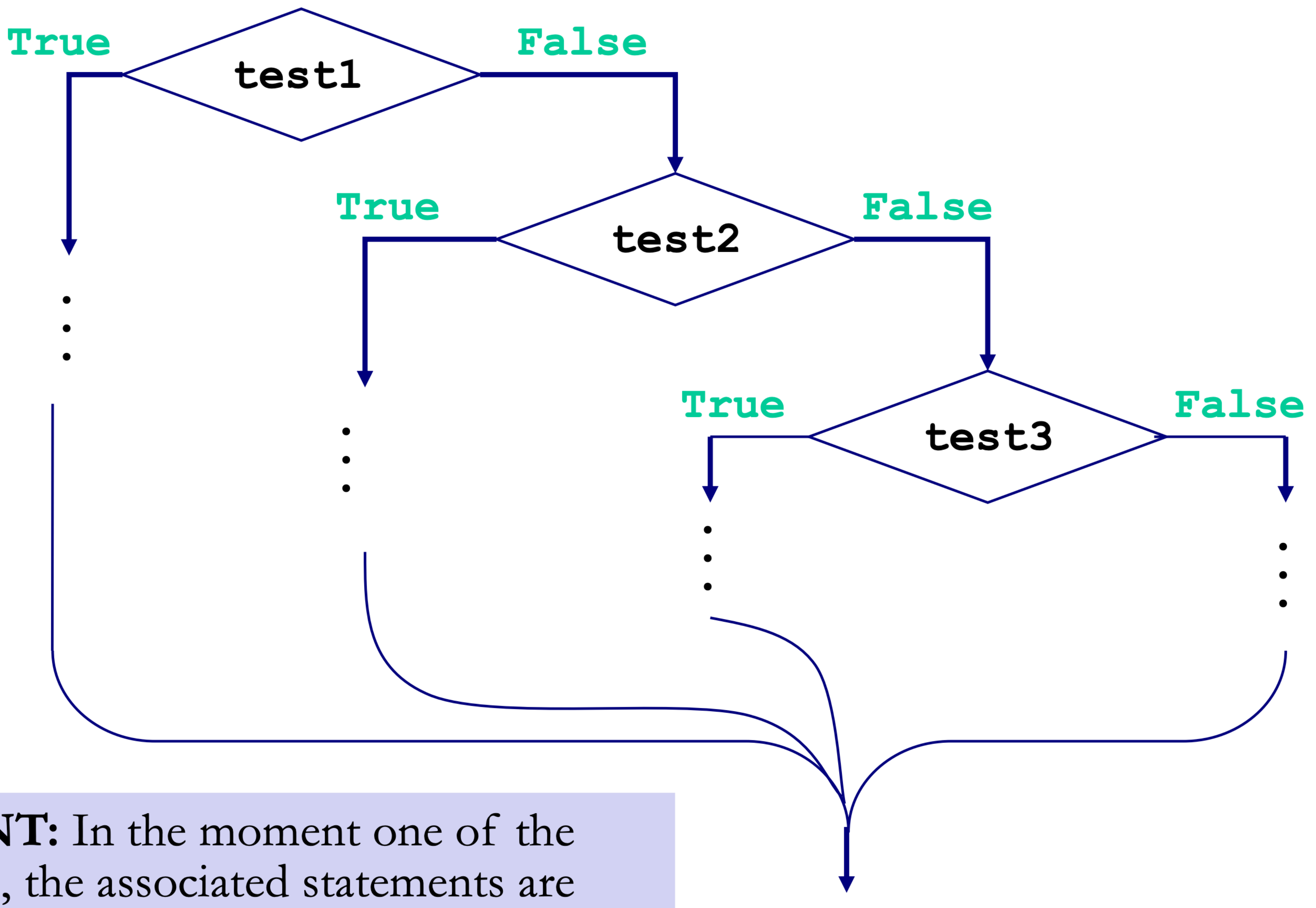
Can have any number of **`elif`** conditions, but only one (optional) **`else`** (at the end)

Attempt 3: Chained Conditionals

- We can chain together any number of `elif` blocks
- The `else` block is **optional** (not a required part of the syntax)

```python
def weather3(temp):
    if temp > 80:
        print("It is a hot one out there.")
    elif temp >= 60:
        print("Nice day out, enjoy!")
    elif temp >= 40:
        print("Chilly day, wear a sweater.")
    else:
        print("Its freezing out, bring a winter jacket!")
```

Flow Diagram: Chained Conditionals



IMPORTANT: In the moment one of the tests is **`True`**, the associated statements are executed and the chained conditional is exited. Only in the case when tests are False, we continue checking to find a True test.

Takeaways

- Chained conditionals avoid messy nested conditionals
- Chaining reduces complexity and improves readability
- Since at most one branch in a chained `if-elif-else` block can be executed (the first condition that evaluates to `True`, or the **`else`** if all conditions are false) using them avoids unnecessary checks incurred by chaining if statements one after the other

Exercise: `leapYear` Function

- Let's write a function **`leapYear`** that takes a **`year`** (int) as input, and returns **`True`** if **`year`** is a leap year, else returns **`False`**
- When is a given year a leap year?
  - *"Every year that is exactly divisible by four is a leap year, except for years that are exactly divisible by 100, but these centurial years are leap years, if they are exactly divisible by 400."*

How do we structure this logic using booleans and conditionals?

Exercise: `leapYear` Function

- Let's write a function **`leapYear`** that takes a **`year`** (int) as input, and returns **`True`** if **`year`** is a leap year, else returns **`False`**
- When is a given year a leap year? (wikipedia)
  - *"Every year that is exactly divisible by four is a leap year, except for years that are exactly divisible by 100, but these centurial years are leap years, if they are exactly divisible by 400."*

Decomposition!

  - If year is **not** divisible by 4: year is not a leap year
  - Else (divisible by 4) and if **not** divisible by 100: is a leap year
  - Else (divisible by 4 and by 100) and **not** divisible by 400: not a leap year
  - Else (if we make it to here must be divisible by 400): is a leap year

`leap.py`

Importing functions

Using functions in different files

Suppose you define a function `is_leap()` in the file `leap.py`

- If you want to use this function in a *different* file (e.g, `main.py`)
  - You need to tell python about it using an **`import`** statement
- ex: `from leap import is_leap`

```
from <filename w/o extension> import <function name>
```

main.py