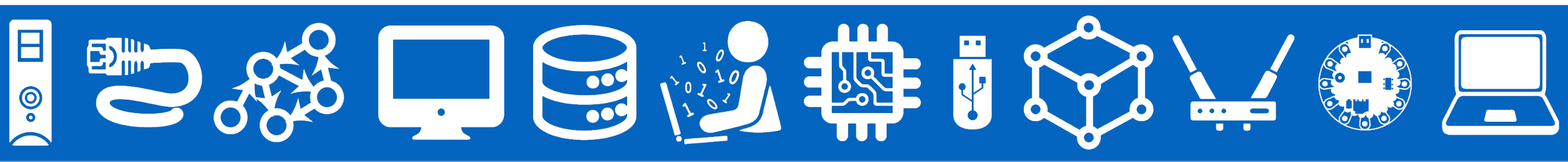
CS 134:
More Conditionals



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

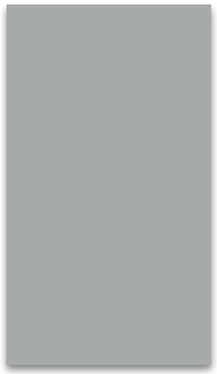
- Learn more about `if else` statements
- Look at more complex decisions in Python
    - Boolean expressions with **and**, **or**, **not**
- Choosing between many different options in our code
    - `if elif else` "chained" conditionals
- Using `import` for using functions across different `.py` files
- 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!

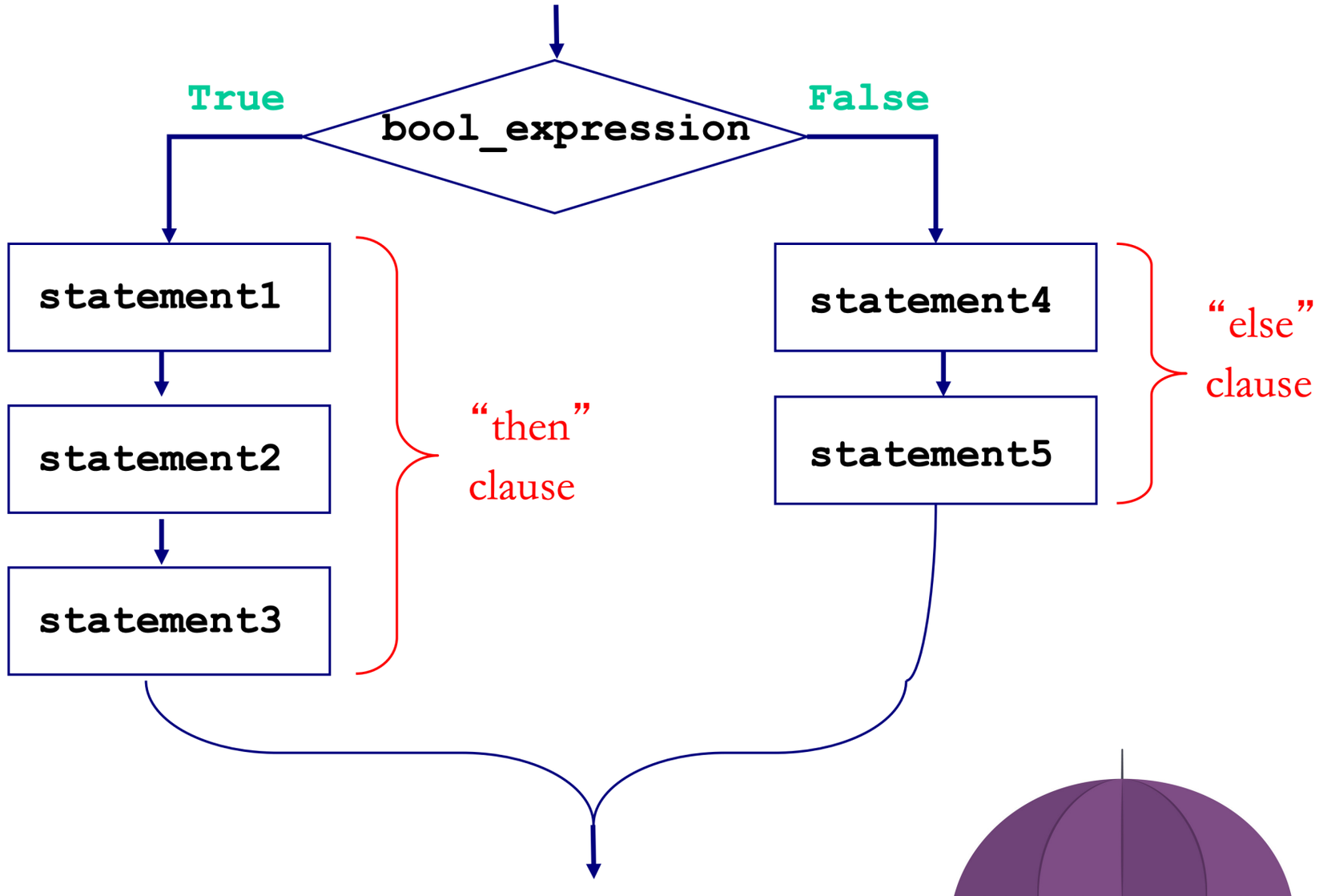
Boolean Expressions and If Statement

- 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**

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

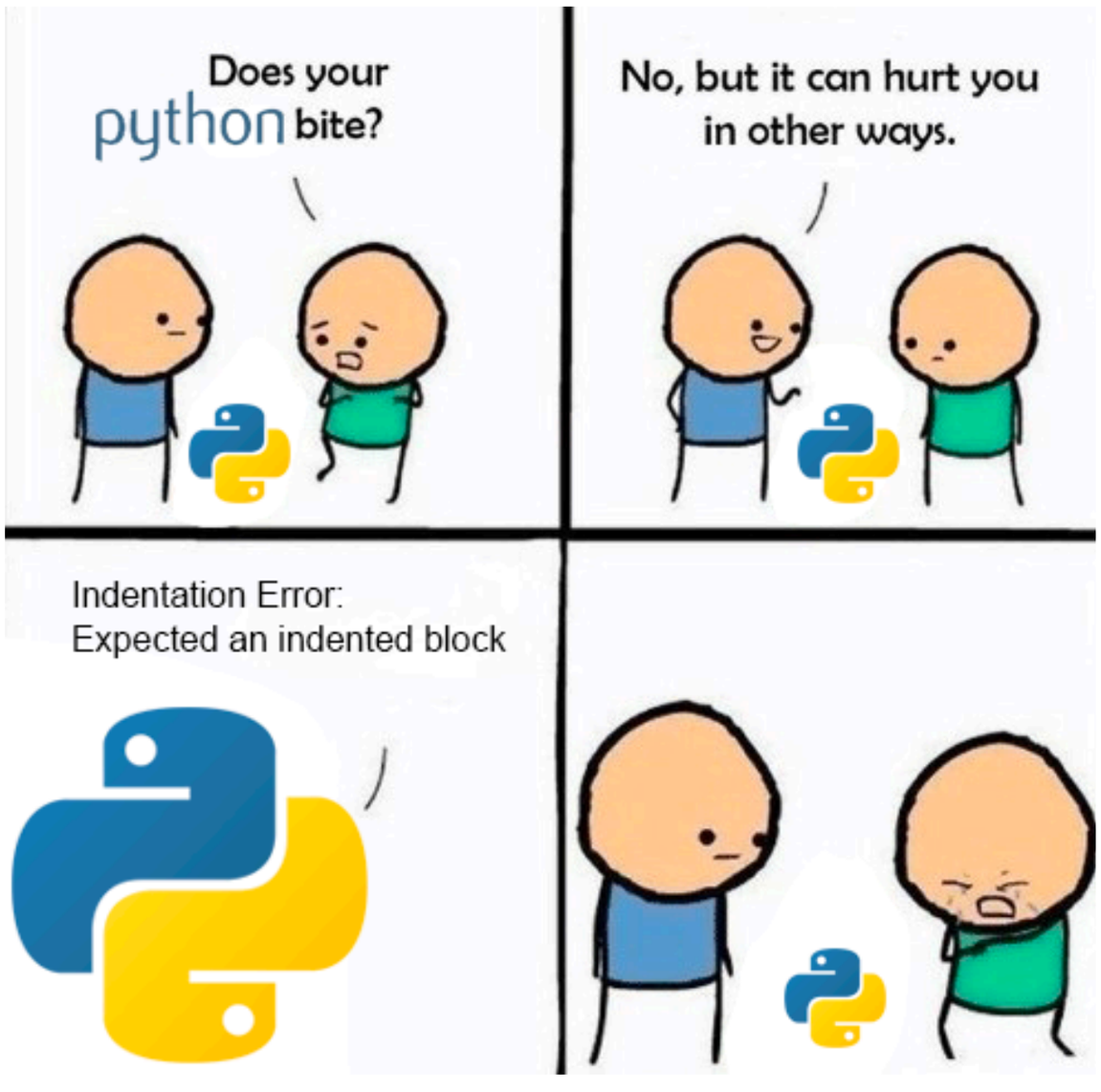
```
if num % 2 == 0:
    return True
else:
    return False
```

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

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

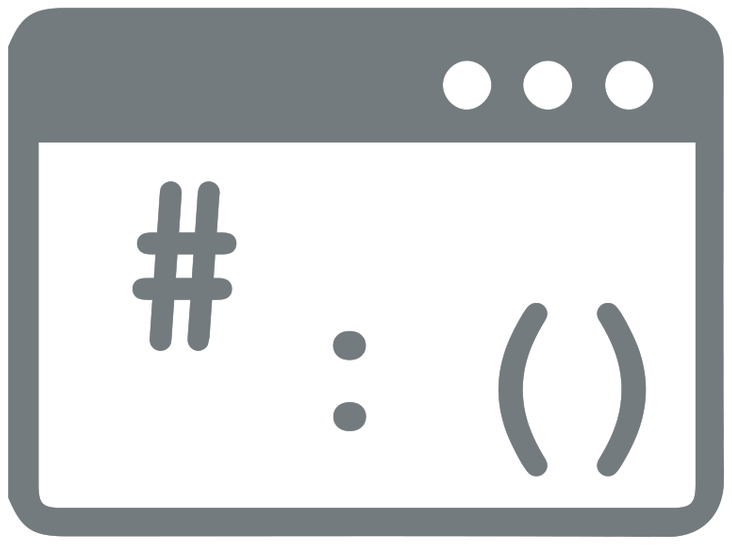
Python Conditionals (`if` Statements)

- Don't forget proper indentation!

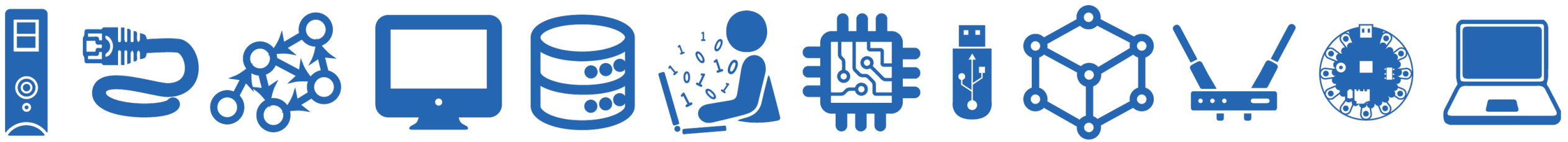


(Credit to u/ufoludek_ on r/ProgrammerHu

Some Examples



Nested Conditionals



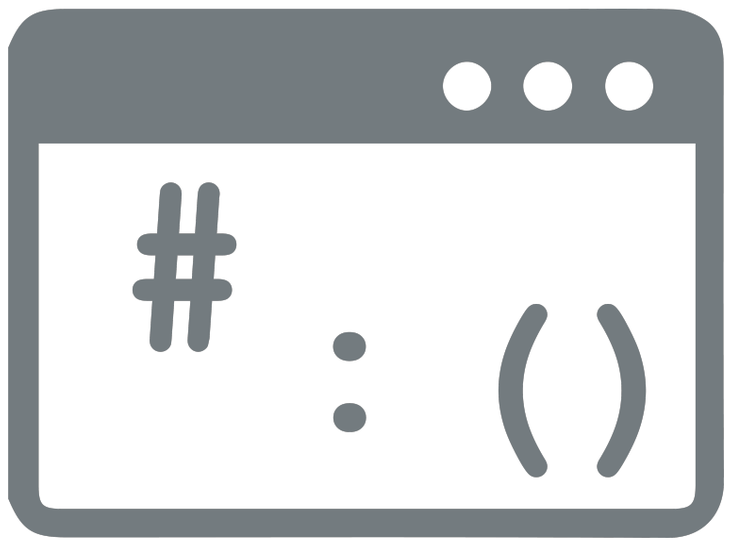
More Decisions

- 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
        ...
```

Examples:
Nested Conditionals

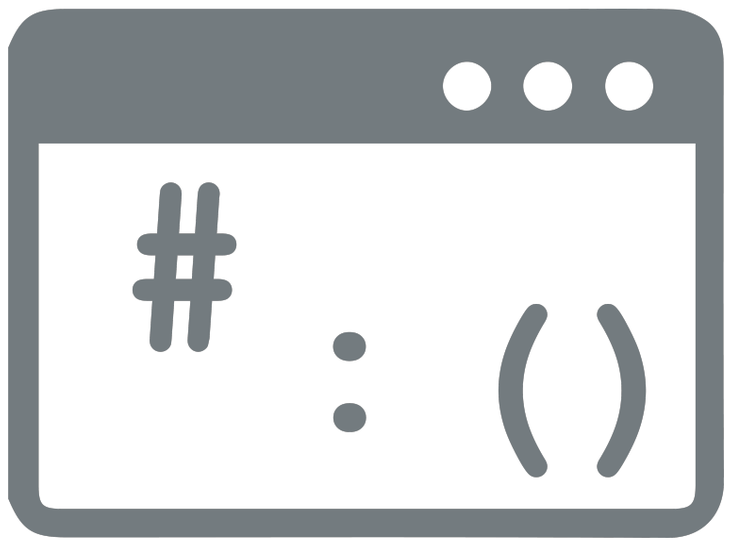


Attempt 1: Chained Conditionals

- 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!) This is an example of what NOT to do!

```
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!")
```

Examples:
Nested Conditionals



Attempt 2: Chained 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?

```
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!")
```

Attempt 2: Chained 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?
    - Adv: More readable/less complex than Attempt 1
    - Disad: Unnecessary condition checking

```
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

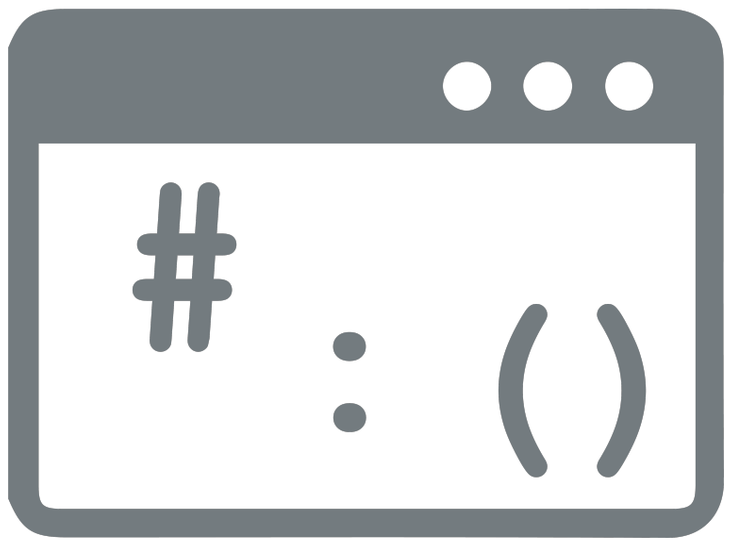
- Fortunately, there is a simpler way to specify several options by **chaining** conditionals

```
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)

Examples:
If-Elif-Else

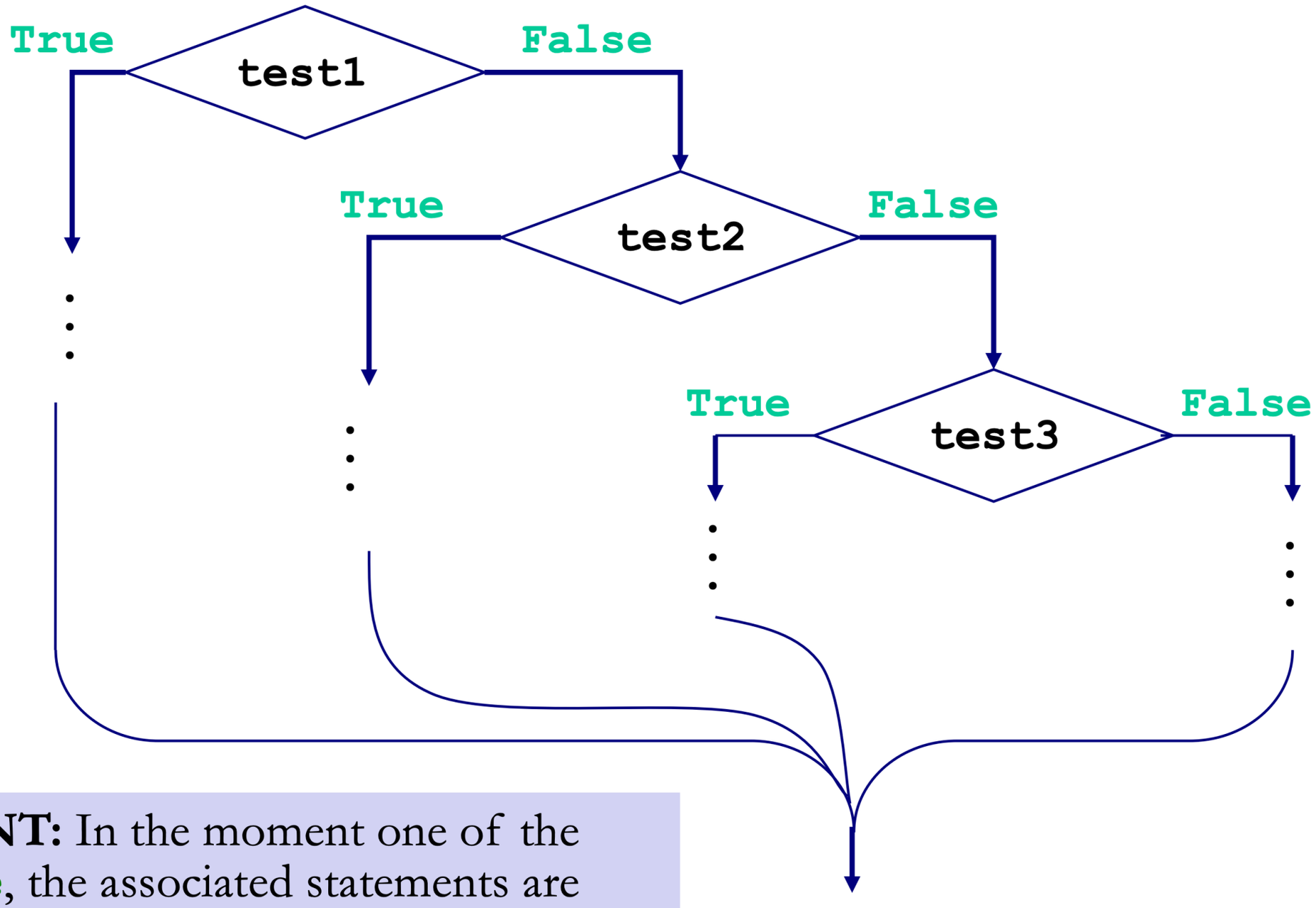


Attempt 3: Chained Conditionals

- We can chain together any number of elif blocks
- The else block is optional, but usually good to include

```
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

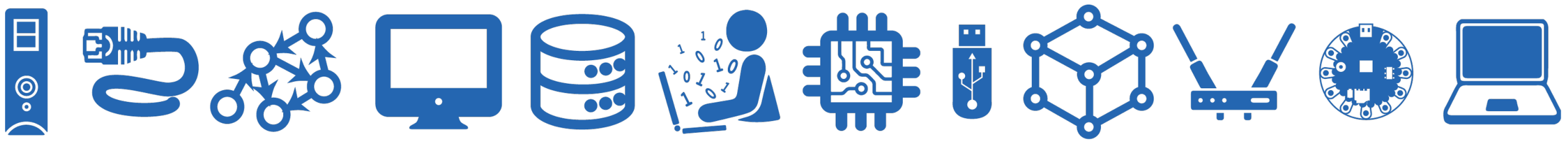


IMPORTANT: In the moment one of the tests is `True`, the associated statements are executed and the chained conditional is exited. Only in the case when tests are False, we continue checking to find a True test.

Takeaway of Conditionals

- Chained conditionals avoid messy nested conditionals
- Chaining reduces complexity and improves readability
- Since only one branches in a chained `if-elif-else` block evaluates to `True`, using them avoids unnecessary checks incurred by chaining if statements one after the other

Exercise:
Leap Year Function



Exercise: `leap_year` Function

- Let's write a function `is_leap` 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: `leap_year` Function

- Let's write a function `is_leap` 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."*

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

Exercise: `leapYear` Function

```python
def is_leap(year):
    """ Takes a year (int) as input and returns
    True if it is a leap year, else returns False"""

    # Write code here!

```

Leap years between from 1900 to 2060:

Not a leap year

1900	1904	1908	1912	1916	1920	1924	1928	1932	1936
1940	1944	1948	1952	1956	1960	1964	1968	1972	1976
1980	1984	1988	1992	1996	2000	2004	2008	2016	2020
2024	2028	2032	2036	2040	2044	2048	2052	2056	2060

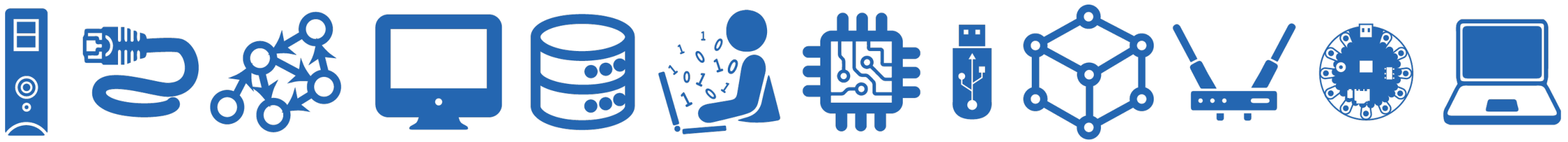
Next leap year

https://www.calendar.best/leap-years.html

Exercise: `leapYear` Function

```python
def is_leap(year):
    """Takes a year (int) as input and returns
    True if it is a leap year, else returns False"""

    # if not divisible by 4, return False
    if year % 4 != 0:
        return False

    # is divisible by 4 but not divisible by 100
    # return True
    elif year % 100 != 0:
        return True

    # is divisible by 4 and divisible by 100
    # but not divisible by 400, return False
    elif year % 400 != 0:
        return False

    # is divisible by 400 (and also 4, and 100)
    # return True
    return True
```

importing functions



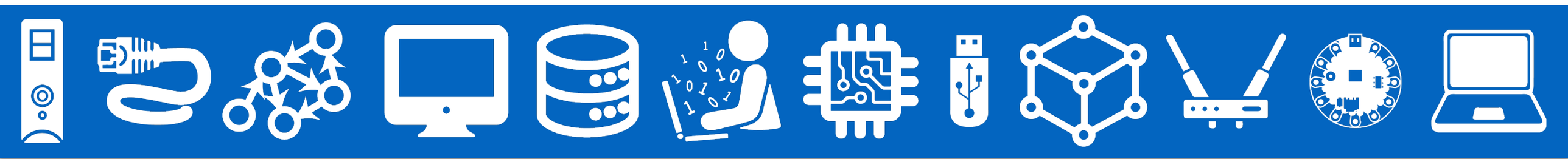
Using functions in different files

- Especially when we're using larger amounts of code, we'll want to use functions in different files than the one it's defined in
- To do this, we use the following syntax:

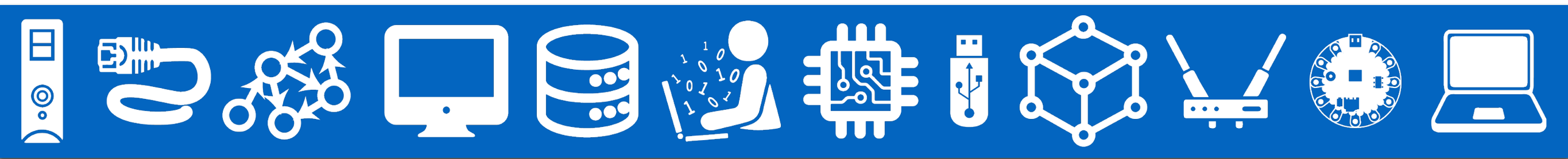
  `from` <name of file without extension> `import` <name of function w/o arguments>

- ex: `from leap import is_leap`

The end!



CS134:
Lab 02



Lab 2: Goals

- In this lab, you will be writing a **non-trivial Python** script to compute the current day of the week in Williamstown
- High-level learning goals:
    - Defining and calling **functions**.
    - Using **arithmetic operators** in Python.
    - Testing your code in **interactive** Python.
    - Writing **conditional (if else) statements** to make decisions in your code

How Computers Keep Track of Time

- On Unix machines time is represented by the **number of seconds,** starting from the beginning of Thursday, January 1, 1970
    - The date is arbitrary, but is called the Unix "epoch"
- In Python we can access this value using the time module()
- The time value is in UTC (current time in England)
- While the value is a float, we only need the integer part for this lab

```
$ python3
>>> from time import time
>>> time()
1612800680.9091752
```

Figuring Out the Day of the Week

- The `time` module gives us the total number of seconds since the Epoch
- *Our goal:* Use this value to figure out what the current day of the week is in England (for now, later we will deal with timezones)
- Approach (break down the problem):
    - How many minutes have elapsed since the Epoch?
    - How many hours? Days?
    - Suppose the number of days divide evenly by 7. What day of the week is it? What if they do not divide evenly?
- How do we do this using arithmetic operations?
    - *Hint*: Think about our example involving `num_coins` from last week

`utc_day(time_value)`

- **You will write this function first!**
- This function takes a floating point number as a parameter, `time_value`
    - `time_value` represents the UTC time in England
    - `time_value` is the total number of seconds since the Epoch
- This function should return:
    - A number between 0-6, which is the day of the week corresponding to `time_value`
    - Where 0 is Sunday, 1 is Monday, ..., 6 is Saturday

The end!