Booleans and Conditionals

Check-in and Reminders

- Reminder: **hand in Homework 1** by placing it in the folder
- Make sure to pick your **graded Homework 0** from me
- Lab 2 linked on course website—take a look before lab
- Where we have been:
- **Functions**
  - What is the purpose of writing functions?
  - What happens we call a function?
  - Where does the value returned from a function go?
  - What happens to local variables after you hit return?

Do You Have Any Questions?

Accessing Lecture Notebooks

CSCI 134 - Spring 2020

Introduction to Computer Science

Shikha's Lectures (9 am)

Links to lecture slides and files will be available after class on the date shown.

Jupyter Notebooks. In lecture, we will use Jupyter notebooks as a teaching aid. Jupyter notebooks allow us to have a rich web-based interface to run interactive python examples. The notebook for each lecture will be distributed here in the form of an html file, a pdf file, and finally the source 'ipynb' (read interactive python notebook) file.

How to read Jupyter Notebooks. Typing a command in a 'In[]' cell in a Jupyter notebook is the same as typing it in an interactive python session. The 'Out[]' cell of the notebook gives the resulting output. Thus, Jupyter notebook is essentially an enhanced way to use interactive python: it stores code examples that can be executed live and are rendered in a rich format.

Installing Jupyter Notebooks. All the lecture materials from the notebooks are available here in HTML and PDF format, and you do not need to install the application. However, if you would like to play with Jupyter notebooks and execute the code in the cells, you may download and install it by following the instructions here.

Date	Topic
February 7	Lecture 1. Hello, world.
February 10	Lecture 2. Expressions. | nameage.py | Jupyter Notebook: [html] | [pdf] | [ipynb]
February 12	Lecture 3. Functions. | dow.py | Jupyter Notebook: [html] | [pdf] [ipynb]
February 14	Winter Carnival. No Lecture.

Making Decisions

If it is raining, then bring an umbrella.

If light is yellow, slow down. If it is red, stop

True or False

Boolean Types

- Python has two values of **bool** type, written `True` and `False`
- These are called logical values or Boolean values, named after 19th century mathematician George Boole
- `True` and `False` must be capitalized !
- Boolean values naturally result when we use **relational and logical operators**

Relational Operators

< (less than)

> (greater than)

<= (less than or equal to)

> = (greater than or equal to)

! = (not equal to)

This is why the single equal sign = is "gets", which is assignment and nothing to do with mathematical equality

```
In [1]: 3 > 5
Out [1]: False
In [2]: 5 != 6
Out [2]: True
In [3]: 5 == 5
Out [3]: True
```

```
In [1]: 'bat' < 'cat'
Out [1]: True
In [2]: 'bat' < 'ant'
Out [2]: False
In [3]: 'Cat' < 'ant'
Out [3]: True
```

Logical Operators

- **`not`** `exp` evaluates to the opposite of the truth value of `exp`
- `exp1` **`and`** `exp2` evaluates to True iff both `exp1` and `exp2` evaluate to True
- `exp1` **`or`** `exp2` evaluates to True iff either `exp1` or `exp2` evaluate to 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**

Source: (http://cs111.wellesley.edu/spring19)

Membership in Strings: `in`

- We will cover strings in the coming lectures, but the `in` operator and `not in` operators are useful in predicates
- `s1 in s2` tests if string `s1` is a substring of string `s2`
- `not in` returns the opposite of in
- `s1 not in s2` is the same as `not s1 in s2`

```
In [1]: '134' in 'CS134'
Out [1]: True
In [2]: 'era' not in 'generation'
Out [2]: False
```

Membership in Lists: `in`

- `in` operator can also be used in other sequences such as a list
- A list in Python is an ordered collection of items enclosed in `[]`
- For example,

```
In [1] evenNums = [1, 2, 3, 4, 5, 6, 8, 10]
In [2] nameList = ['Anna', 'Chris', 'Zoya', 'Sherod',
'Zack']
```

- `item in mylist` tests if `item` is present in the list `myList`

```
In [1]: '4' in 'evenNums'
Out [1]: True
In [2]: 'Shikha' in 'nameList'
Out [2]: False
```

Predicates

- A predicate is any function that returns a Boolean value

```
def isDivisible(num, factor)
    ```determines whether the number is divisible by factor```
    return (num % factor) == 0
```

Can return a Boolean expression directly

```
def isEven(n)
    ```determines whether the number is even```
    return isDivisible(n, 2)
```

Notice: A function call in return

Predicates

- Simple predicate to check if a letter is a vowel

```
def isVowel1(char)
    ```determines whether a character is a vowel```
    c = char.lower()  # returns char as lowercase
    return (c == 'a' or c == 'e' or c == 'i' or c == 'e' or c
    == 'o' or c == 'u')
```

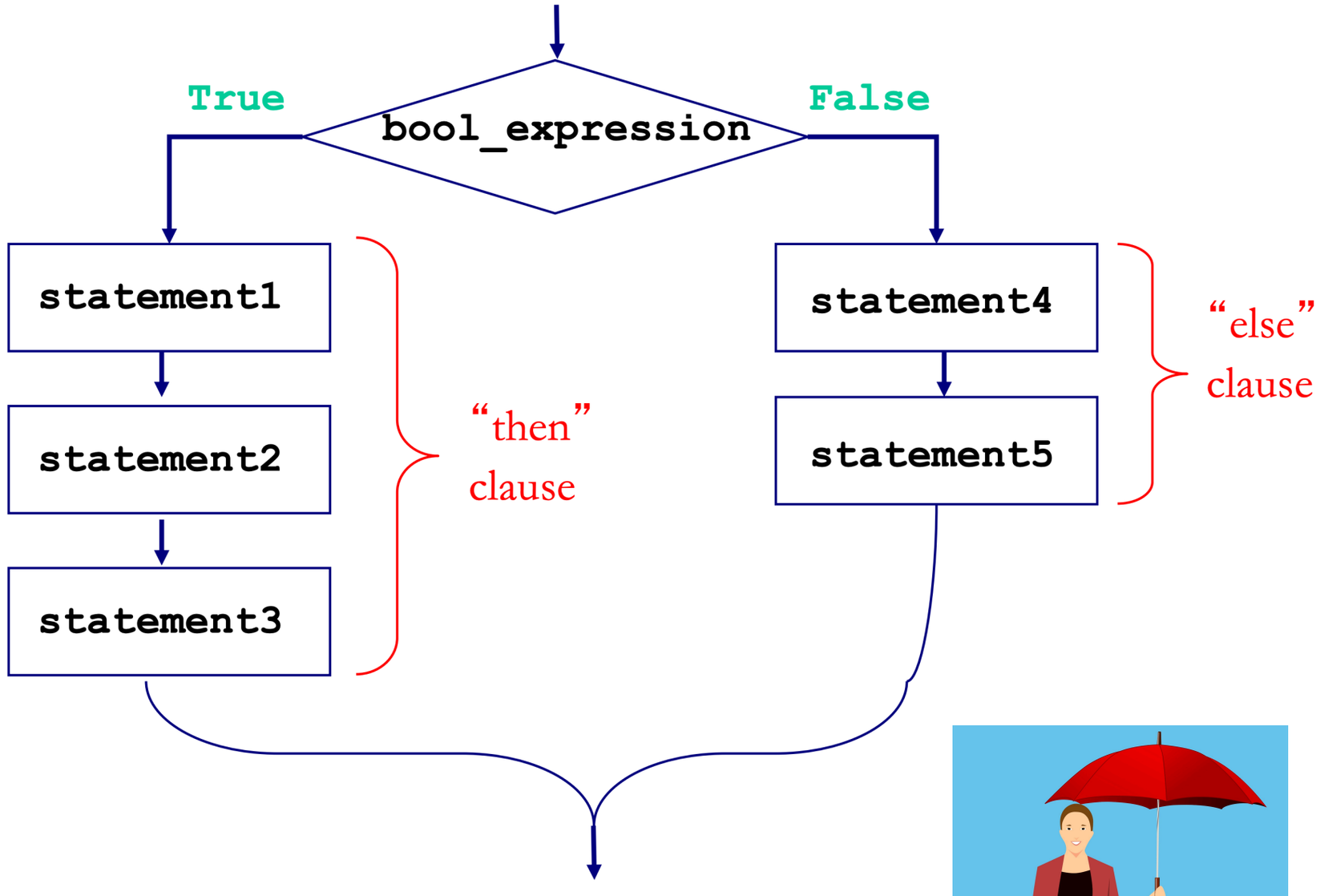
Can we chain and say `c == 'a' or 'e' or 'i' or 'e' or 'u'`?

```
def isVowel2(char)
    ```determines whether a character is a vowel```
    c = char.lower() # returns char as lowercase
    return c in 'aeiou'
```

Simplified check using in!

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.

Image Source: (http://cs111.wellesley.edu/spring19)

Conditionals and Returns

- Are these two functions logically equivalent?
- Do they return the same answer for all inputs?

```
def abs1(n)
    ```returns the absolute value of a number```
    if n < 0:
        return -n
    else:
        return n
def abs2(n)
    ```returns the absolute value of a number```
    return char in 'aeiou'
    if n < 0:
        return -n
    return n
```

Notice the missing else

Nested Conditionals

```
if boolean_expression1:
    statement1
    statement2
else:
    if boolean_expression2:
        statement3
        statement4
    else:
        statement5
        statement6`
```

```
def movieAge(age):
    if age < 8:
        return 'G'
    else:
        if age < 13:
            return 'PG'
        else:
            if age < 18:
                return 'PG-13'
            else:
                return 'R'
```

Image Source: (http://cs111.wellesley.edu/spring19)

A Better Approach: Chaining

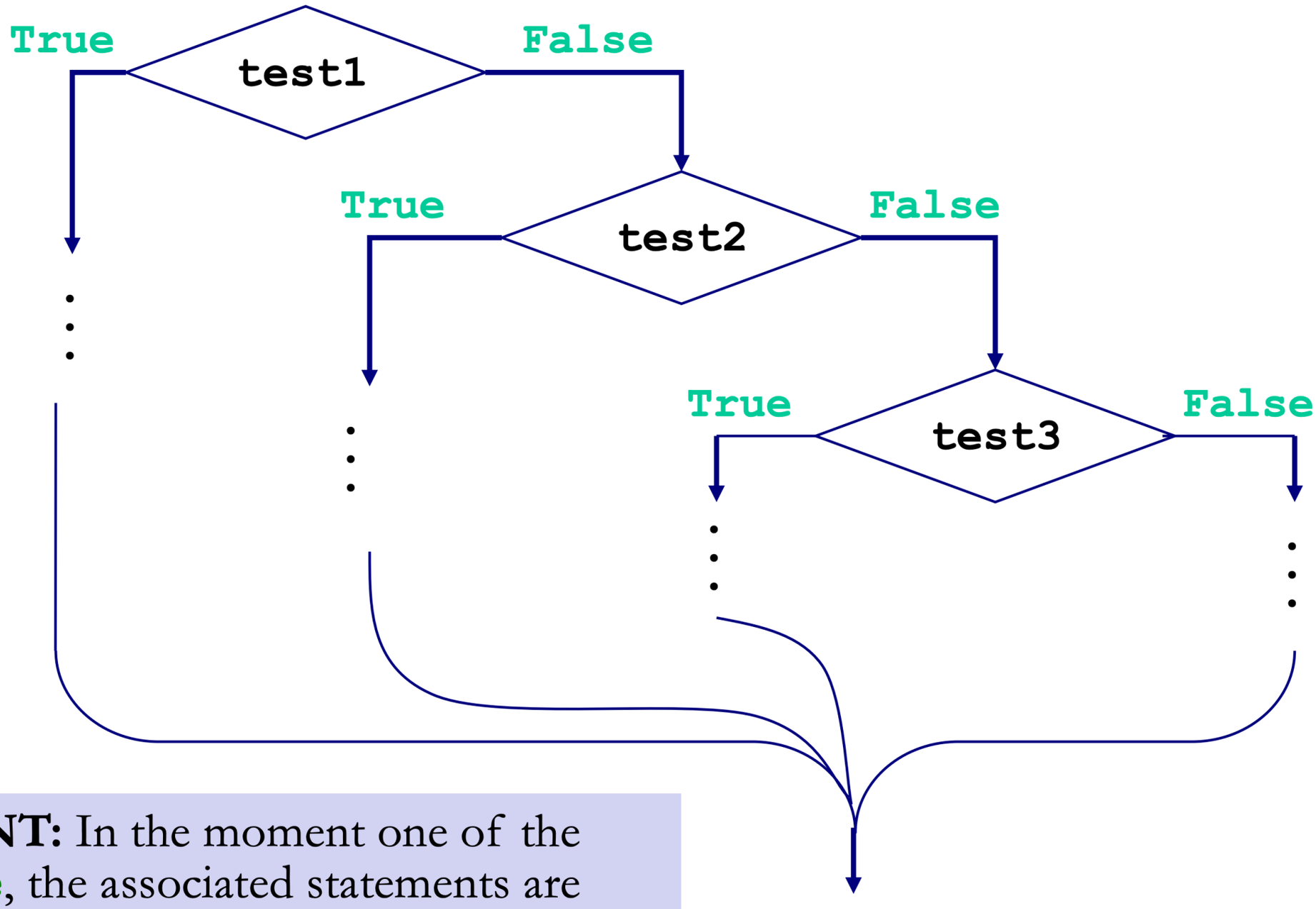
```
if boolean_expression1:
    statement1
    statement2
elif boolean_expression2:
    statement3
    statement4
elif boolean_expression3:
    statement5
    statement6
else:
    statement7
    statement8
```

```
def movieAge(age):
    if age < 8:
        return 'G'
    elif age < 13:
        return 'PG'
    elif age < 18:
        return 'PG-13'
    else:
        return 'R'
```

Compare this implementation of **movieAge** with that of the previous slide. For chained conditionals, we write less code, which is also easier to read because of fewer indentations.

Image Source: (http://cs111.wellesley.edu/spring19)

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.

Image Source: (http://cs111.wellesley.edu/spring19)

Exercise: Days in Month

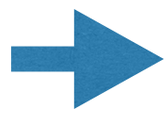
- Define a function named **`daysInMonth`** that takes a month (as an integer between 1-12) as the argument, and returns the number of days in it, assuming the year is not a leap year.
- If month is not between 1 and 12, return an error message.

```
def daysInMonth(month)
    '''Given a month between 1-12, returns the number of days in
it,  assuming the year is not a leap year'''
if month < 1 or month > 12:
    return 'Error: Month does not fall between 1-12'
elif month == 2:
    return 28
elif month == 4 or month == 6 == month == 9 or month == 11:
    return 30
return 31
```

Simplifying Boolean Expressions

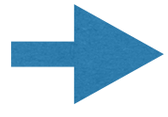
- There are several code patterns involving booleans and conditionals that can be simplified as good coding style

```
if BE:
    return True
else:
    return False
```



```
return BE
```

```
if BE1:
    return BE2
else:
    return False
```



```
return BE1 and BE2
```

Acknowledgments

These slides have been adapted from:

- http://cs111.wellesley.edu/spring19 and
- https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-0001-introduction-to-computer-science-and-programming-in-python-fall-2016/