IF ELSE AND DEBUGGING

9.19.2018

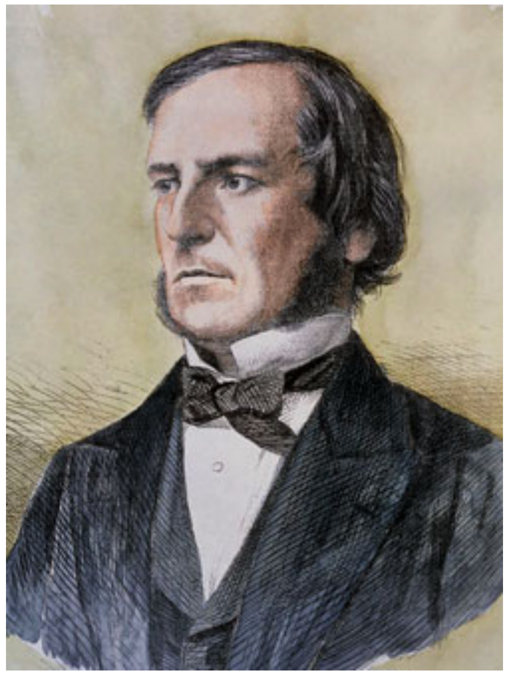
PROBLEM SET 1

* Due Friday (9/21) at 10:59AM

* **THIS WEEK ONLY:** Manu (the TA) moved his office hour to TODAY (Wednesday) from 5:00-6:00PM in the 3rd floor NHB atrium

* **THIS WEEK ONLY:** I’m adding an extra office hour Thursday 11:00AM-12:30PM (in my office, NHB 3.134)

BOOLEANS


George Boole

* The **bool** data type has only two possible values: **True** and **False**

* Whenever you use a *comparison operator*, the result is a bool

BOOLEANS

* Important comparison operators:
  * ==, !=
  * <, <=, >, >=
  * in, not in
  * and, or *- these combine two bools*

BOOLEANS

- e.g.
  - a == b
  - a and b
  - a in b

IF

* bools are used by if statements to control the execution of code

* e.g.

```
if some_bool:
    do_something()
```

* do_something() will execute only if some_bool is True. If some_bool is False, the code inside is skipped

IF-ELSE

* an else block is executed when the condition is false, e.g.

*
```
if some_bool:
    do_one_thing()
else:
    do_another_thing()
```

* do_another_thing() is only executed when some_bool is False

IF-ELIF-ELSE

* if statements can be chained together

*
  ```
  if some_bool:
      do_something()
  elif some_other_bool:
      do_another_thing()
  else:
      do_a_third_thing()
  ```

* when do you think the else block (do_a_third_thing) executes?

IF-ELIF-ELSE

* what’s the difference between the last slide and this?

*
```
if some_bool:
    do_something()
if some_other_bool:
    do_another_thing()
else:
    do_a_third_thing()
```

IF-ELIF-ELSE

* any number of elif blocks can be chained together! (but that’s kind of ugly)

IF WITHIN FOR

* if statements can, of course, appear within for loops

* for thing in collection:

```
for thing in collection:
    if thing == ‘a special thing’:
        print(“wow neat”)
    else:
        print(“not that special”)
```

DEBUGGING

* <u>Situation 1:</u> you run some code, it throws out an error. What do you do next?

  * Look at the stack trace

  * Use the debugger

THE DEBUGGER

* accessing the debugger
* from jupyter notebook:
    * type debug in an empty cell, hit shift-enter
* from ipython:
    * type debug after an error, hit enter

THE DEBUGGER

* when in the debugger:
    * you can run any bit of code you want by typing it and hitting enter
    * (mostly I use it to print the values of variables)
    * you can move around your code, but that’s pretty advanced
    * when done, you MUST “exit”

DEBUGGING

* Situation 2: you run some code, it DOESN'T throw an error, but gives you the wrong answer
  * Manually trace your code:
    * **print** the value of each intermediate variable that you create!
      * (this can be overwhelming: maybe run on a cut down version of the data)
    * CREATE errors, then use debug!

THAT’S IT