Sit with a robot partner

- **Same partner or a new one, your choice – you will complete the robot exercise in class**

Checkout
Session10-Conditionals
from your repository

DECISION STRUCTURES, COMPUTING WITH BOOLEANS

Exam 1

- **Thursday evening, 7 to 9 p.m.**
  - But you may stay until 10 p.m. if you wish
- Section 1: Olin 257 Section 2: Olin 259
- No class Thursday morning
- **Optional review session: Wednesday, 7 p.m. to 8:30 p.m.**
  - **in Moench F-217 (CSSE lab)**
  - Also, I will be in my office Thursday morning 1st through 4th periods
- Best way to start preparing:
  - Review the Exam 1 topics (see document on course web site)

Q1

Decision, Decisions

- Normally, statements in a program execute in order, one after the other
- Sometimes we want to alter the sequential flow of a program
  - What examples have we seen of this?
- Statements that alter the flow are called

  control structures
- *Decision structures* are control structures that allow programs to "choose" between different sequences of instructions

Simple Decisions

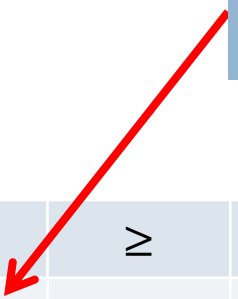
- The **if** statement
  - if <condition>:
    <body>
  - Semantics:
    "if the condition is True, run the body, otherwise skip it"
  - Example:

```
if x < 0:
    x = -x
```

- Simple conditions
  - <expr> <relop> <expr>
  - Some relational operators:

Math	<	≤	=	≥	>	≠
Python	<	<=	==	>=	>	!=

Note! Why not a single equal sign?



Q2

Class Exercise

- Checkout the **Session10-Conditionals** project from your SVN repository
  - Do the first two TODO's (#0 and #1) in the ***grade.py*** module:
    - Your name
    - Implement the *grade1* function
      - *Return 'just barely' if the given score is 60, else do nothing (and don't return anything).*

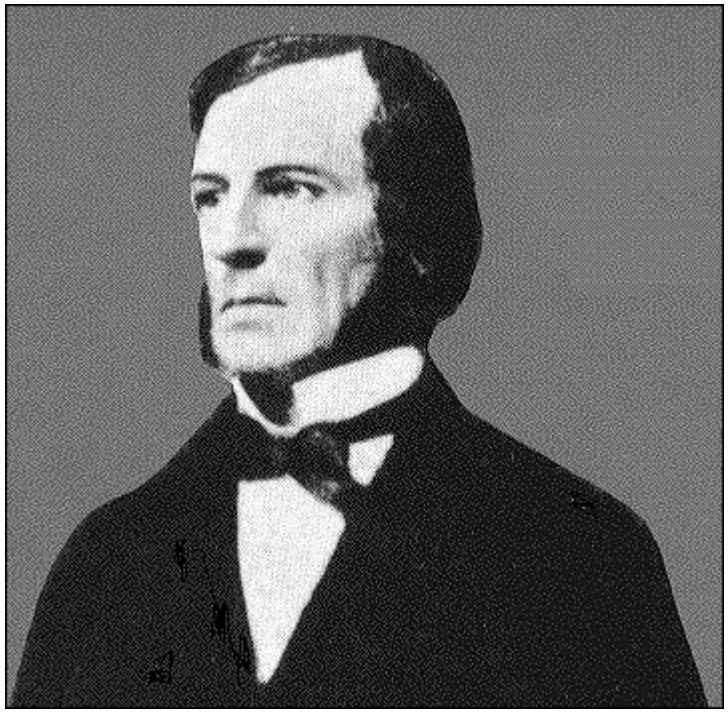
  The output should be:

```
Grade1: If the score is 60, then the grade is just barely
Grade1: If the score is 61, then the grade is None
Grade1: If the score is 59, then the grade is None
```

More on Comparisons

- Conditions are *Boolean expressions*
  - They evaluate to **True** or **False**
- Try in IDLE's Python shell:

```
3 < 4
42 > 7 ** 2
"ni" == "Ni"
"A" < "B"
"a" < "B"
```



George Boole

Q3

Boolean Variables and Operations

- Boolean constants: **True, False**
- Relational operators (<, etc.) produce Boolean values.
- Other Boolean operators:
  and, or, not

```
>>> 4 < 5
True
>>> 6 != 6
False
```

P	Q	P and Q
T	T	T
T	F	F
F	T	F
F	F	F

P	Q	P or Q
T	T	T
T	F	T
F	T	T
F	F	F

P	not P
T	F
F	T

- What does this evaluate to: **not((1 < 1) or (3 == 3))**
- Do TODO #2 in **grade.py**:
  - Implement *grade2*, which returns *'is a C'* or nothing.

Q4

Having It Both Ways: if-else

- Syntax:

```
if <condition>:
        <statementsForTrue>
else:
        <statementsForFalse>
```

- Semantics: "If the condition is true, execute the *statementsForTrue*, otherwise execute the *statementsForFalse*"
- Example:

```
if (x >= 60):
    return 'passing'
else:
    return 'failing'
```

Tip: Use *else* whenever it makes sense, because:

- Faster (don't have to repeat the test)
- Clearer

- Do TODO #3 in **grade.py**:
  - Implement *grade3* which returns '*perfect*' or '*not perfect*'.
    Note: You can do *grade3* without an *else*, because of *return*'s, but it is clearer with one.

A Mess of Nests

- Can we modify the **grade** function to return letter grades—A, B, C, D, and F?

```
def gradeNesting(score):
    if score >= 90:
        result = "A"
    else:
        if score >= 80:
            result = "B"
        else:
            if score >= 70:
                result = "C"
            else:
                if score >= 60:
                    result = "D"
                else:
                    result = "F"
    return result
```

Multi-way Decisions

- Syntax:

```
if <condition1>:
    <case 1 statements>
elif <condition2>:
    <case 2 statements>
elif <condition 3>:
    <case 3 statements>
…
else:
    <default statements>
```

reach here if condition1 is false

reach here if condition1 is false AND condition2 is true

reach here if BOTH condition1 AND condition2 are false

- Advantages of **if-elif-else** vs. nesting
  - Number of cases is clear
  - Each parallel case is at same level in code
  - Less error-prone

Do TODO #4 in **grade.py:** Implement *grade4*, which returns the letter grade for the given score

Q7

The *counting* pattern

- A special case of the accumulator pattern
- Example:

```
def count_As(scores):
    """Returns the number of A in the given list of scores"""
    count = 0
    for score in scores:
        if (score >= 90):
            count = count + 1
    return count
```

Initialize

Loop

Count conditionally

```
print count_As([87, 92, 100, 75, 93])
```

Finish the quiz,
then do the TODO's in the ***countPassFail*** module.
Commit when done.

Q8