CS 115 Lecture

Boolean logic

Taken from notes by Dr. Neil Moore

Boolean logic and logical operators

There are three **logical operators** that let us combine Boolean expressions. They have **lower** precedence than the relational operators (<, >, …)

- `not` A: True if A is False, False if A is True
  - A is any Boolean expression:

```
if not is_finished:
    do_more_work()
```

- A `and` B: True if **both** A and B are True

```
in_range = size >= 0 and size <= 100
```

- A `or` B: True if **either** A or B is True or Both!

```
if snow_inches > 6 or temperature < 0:
    print("Class is cancelled")
```

Complex Boolean expressions

- `not` has the highest precedence (but still lower than relational)
- `and` has the next highest
- `or` has the lowest of the three
- So `not A or B and C or D` means

```
(((not A) or (B and C)) or D)
```

- People often forget the order of **and** and **or** operators
  - It's not a bad idea to always use parentheses when they are both in an expression

```
not A or (B and C) or D
```

Truth tables

The **truth table** is a tool for making sense of complex Boolean expressions.

A	not A
True	False
False	True

A	B	A and B
True	True	True
True	False	False
False	True	False
False	False	False

A	B	A or B
True	True	True
True	False	True
False	True	True
False	False	False

Truth tables

- A table has one **row** for each possible combination of values of True and False
  - if there is one input, two rows (T, F)
  - two inputs, four rows (TT, TF, FT, FF)
  - three inputs, eight rows (TTT, TTF, TFT, TFF, FTT, FTF, FFT, FFF)
- A table has one **column** for each boolean expression
  - Inputs: Boolean variables or comparisons (relational expressions)
  - Intermediate results: The Boolean value of the expression for each **not**, **and, or**.
  - Output: the Boolean value of the whole expression

A more complicated example

```
not (not A or not B)
```

A	B	not A	not B	not A or not B	result
True	True				
True	False				
False	True				
False	False				

A more complicated example

```
not (not A or not B)
```

A	B	not A	not B	not A or not B	result
True	True	False	False		
True	False	False	True		
False	True	True	False		
False	False	True	True		

A more complicated example

`not (not A or not B)`

A	B	not A	not B	not A or not B	result
True	True	False	False	False	
True	False	False	True	True	
False	True	True	False	True	
False	False	True	True	True	

A more complicated example

```
not (not A or not B)
```

A	B	not A	not B	not A or not B	result
True	True	False	False	False	True
True	False	False	True	True	False
False	True	True	False	True	False
False	False	True	True	True	False

De Morgan's laws

- **Two Forms:**
  - **not (not A or not B) = A and B**
  - **not (not A and not B) = A or B**
- These can be useful for rewriting expressions to simplify them
  - Can make your code easier to understand and faster to execute
- Examples:
  - **not (x > 5 and x < 10)** is the same as not(x > 5) or not (x < 10) which is the same as **x <= 5 or x >=10**
  - **not (y == x or z != 5 and p == q)** is the same as not(y == x) and not (z != 5) or not (p == q) which is the same as **y != x and z == 5 or p != q**
- The opposite of < is >=, that is, a < b is False if a >= b is True
- The opposite of > is <= (don't forget the =)

Be careful!

- It is easy to accidentally write an expression that is *always* True or *always* False
  - **Tautolology** (always True) and **contradiction** (always False)
  - Examples:

```
if size >= 10 or size < 50:
        print(“in range”)
```

    - What happens when size is 100? 20 ? 2? (Hint: all of those values result in True!)
    - The `or` operator is True if EITHER comparison is True; the two comparisons cannot both be False at the same time!
    - So this is a tautology (always True)

```
if size < 10 and size > 100:
        print(“out of range”)
```

    - The comparisons cannot both be True at the same time! At least one condition will be False
    - So the message will never print – a contradiction (always False)

Be careful!

- Don’t trust the English language!
  - Make a truth table if you are not sure
- “I want to run this if size < 10 and if size > 100”
  - In logic, that should be an **or** operator, not an **and** operator:
    - “Run this if size < 10 or size > 100”
- “if x is equal to 4 or 5…”
  - Wrong: `if x == 4 or 5:`
  - Tests must be written out explicitly
  - Should be: “if x is equal to 4 or x is equal to 5”

```
if x == 4 or x == 5:
```

Coercing other types to bools

- Why did the last example `if x == 4 or 5:` run at all? What does Python see it as?
  - `or` is a boolean operator – it works on bools and returns a bool
  - There is a bool from the `x == 4`, but the 5 is by itself! (x == 5 is NOT implied there!)
  - Python needs a bool on the right of the or operator – how does it make the 5 a bool??
  - It forces (coerces) the 5 to be a bool according to the rules
    - For numbers, any value but 0 is turned into True, 0 is False
    - For strings, any string except the empty string is True, "" is False
    - For lists, any list except the empty list is True, the empty list [] is False
    - ALL graphics objects are True!
- So the expression above "x == 4 or 5" is ALWAYS TRUE because the 5 is coerced to True, and "anything or True" is always True. Tautology!

Coercing other types to bools

- This is NOT something you should rely on in your code – it is difficult for someone to read and understand, and it is very prone to bugs if you are not careful.
- Example: What does this condition mean? **if not name:**

  where name is a string
- not is a bool operator, so it must have a bool value to operate on
- name is a string, not a bool, so its value is coerced to a bool
- If the name is an empty string, then it's coerced to False, so **not name** is the same as **not False**, which is **True**
- That condition (not name) is equivalent to **if name == "":**
- But if name == "": is a lot easier to understand (and not get backwards!)

How Indentation Really Matters!

Are these two if structures the same?

```
if a > 12:
    if b < 50:
        print(“red”)
    else:
        print(“blue”)
```

Do they give the same output all the time?

```
if a > 12:
    if b < 50:
        print(“red”)
else:
    print(“blue”)
```