

# [220 / 319] Refactoring Conditionals

Meena Syamkumar  
Andy Kuemmel

Piazza Enrollment 693 / 701

# How to use these slides:

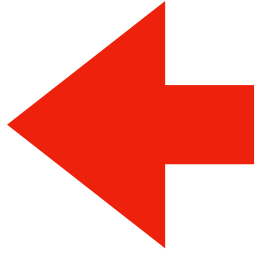
There are more examples here than we can cover in lecture.

However, you can walk through these examples along with the interactive exercises. You should do the following:

1. Think about what the answer is
2. Mentally step through the code using the example call when applicable
3. Step through the code with the Python Tutor examples we've setup for you. For the refactor examples, step through all three versions, and see which alternative (A or B) matches the output of the original version.
4. If you got something different than Python Tutor, tweak your mental model (talk to us if you don't understand something)

# Today's Outline

Review



Refactoring Conditionals

# Review 1: default for first (but not second) arg

```
def subtract(x=100, y=1):  
    return x - y
```

```
x = 200
```

```
y = 2
```

```
print(subtract(y=y))
```

**Your job:** Show what each variable (including parameters) will contain in each frame

# Review 2: arguments and conditions

```
def divide(top, bottom):  
    return top/bottom  
  
def flip_div(top=1, bottom=2, flip=False):  
    if flip:  
        return divide(top=bottom, bottom=top)  
    else:  
        return divide(top=top, bottom=bottom)  
  
x = 2  
y = 3  
print(flip_div(x, y, True))
```

**Your job:** Show what each variable (including parameters) will contain in each frame

# Review 3: globals and conditionals

```
last_b = None

def divide(t, b=None):
    global last_b
    if b == None:
        b = last_b
    last_b = b
    return t / b

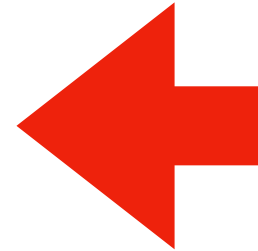
print(divide(1, 4))
print(divide(2))
```

**Your job:** what does the second print display?

# Today's Outline

Review

Refactoring Conditionals




# Refactor Exercise 1

```
def or2(cond1, cond2):  
    return cond1 or cond2
```

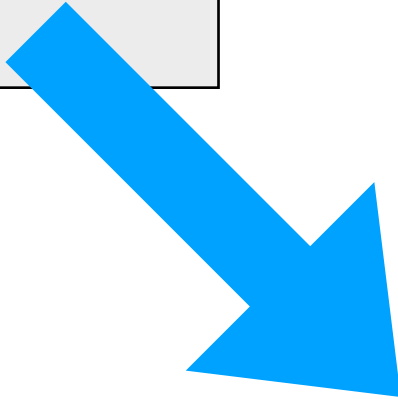
which refactor  
is correct?

hint: `or2(False, True)`



```
def or2(cond1, cond2):  
    rv = False  
    rv = rv or cond1  
    rv = rv or cond2  
    return rv
```

A



```
def or2(cond1, cond2):  
    if cond1:  
        return cond2  
    else:  
        return False
```

B



# Refactor Exercise 1

```
return b1 or b2 or b3 or ... or bN
```



```
rv = False  
rv = rv or b1  
rv = rv or b2  
rv = rv or b3  
...  
rv = rv or bN
```


Lesson: with "or", it only takes one to flip the whole thing True!

## Refactor Exercise 2

```
def and2(cond1, cond2):  
    return cond1 and cond2
```

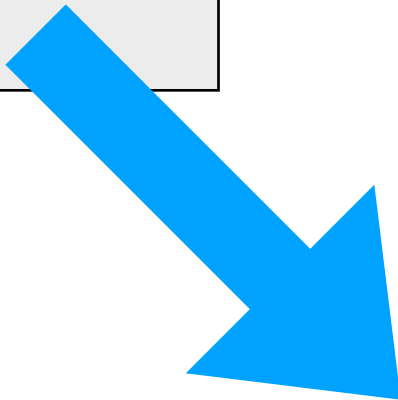
**which refactor  
is correct?**

**hint:** `and2(True, True)`



```
def and2(cond1, cond2):  
    rv = False  
    rv = rv and cond1  
    rv = rv and cond2  
    return rv
```

**A**



```
def and2(cond1, cond2):  
    if cond1:  
        return cond2  
    else:  
        return False
```

**B**

## Refactor Exercise 2

```
return b1 and b2 and b3 and ... and bN
```



equivalent

```
if b1:  
    return b2 and b3 and ... and bN  
else:  
    return False
```

Lesson: with "and", the first one can make the whole thing False!

## Refactor Exercise 3

which refactor  
is correct?

hint: `fix(False, False)`

```
def fix(moves, should):
    if moves:
        if should:
            return "good"
        else:
            return "duct tape"
    else:
        if should:
            return "WD-40"
        else:
            return "good"
```

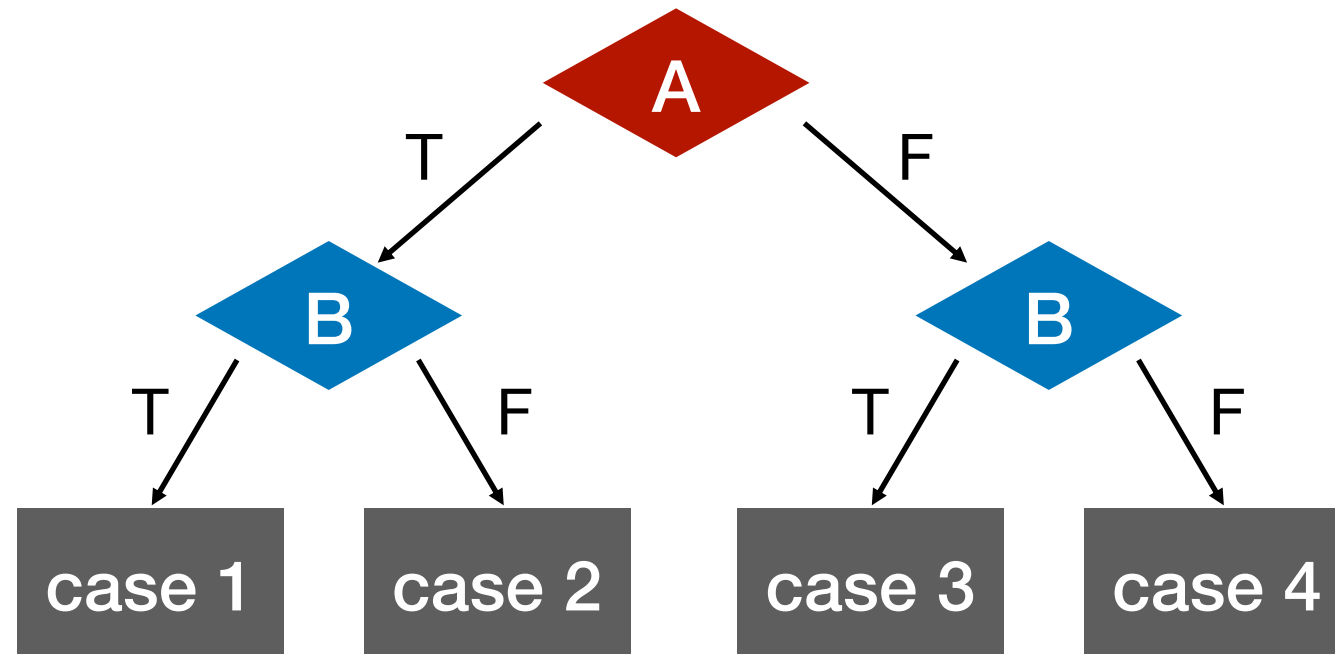
A

```
def fix(moves, should):
    if moves and not should:
        return "duct tape"
    elif not moves and should:
        return "WD-40"
    elif moves and should:
        return "good"
    elif not moves and not should:
        return "good"
```

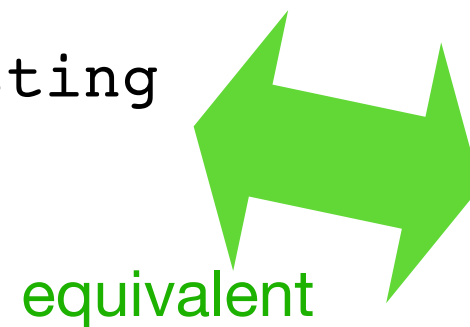
B

```
def fix(moves, should):
    if should:
        if moves:
            return "duct tape"
        else:
            return "good"
    else:
        if moves:
            return "good"
        else:
            return "duct tape"
```

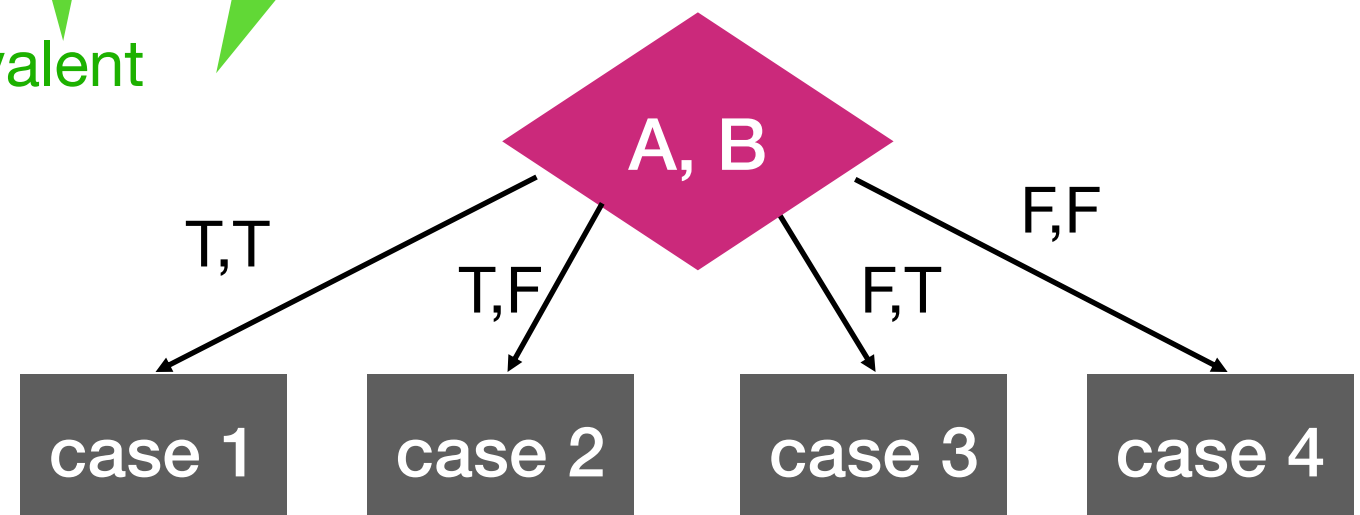
# Refactor Exercise 3



**Option 1:** Nesting



**Option 2:** Chaining

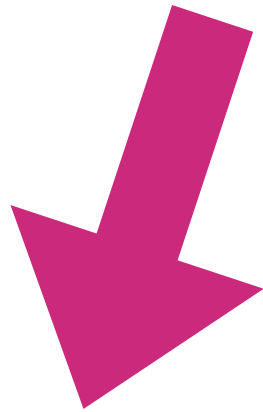


**Lesson:** when handling combinations of booleans, you can either do either (a) nesting or (b) chaining with and

## Refactor Exercise 4

which refactor  
is correct?

hint: `is_220(2, 2, 0)`



```
def is_220(a, b, c):  
    if a==2:  
        if c==0:  
            if b==2:  
                return True  
    return False
```

A



```
def is_220(a, b, c):  
    if a==2 or b==2 or c==0:  
        return False  
    return True
```

B

## Refactor Exercise 4

`return b1 and b2 and b3 and ... and bN`



equivalent

```
if b1:
    if b2:
        if b3:
            ...
            if bN:
                return True
return False
```

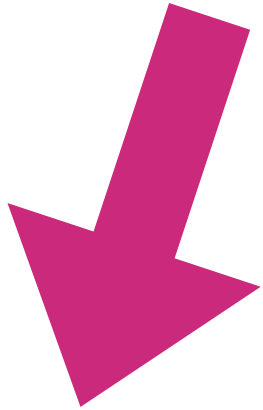
Lesson: nesting a lot of if's inside each other is equivalent to and'ing all the conditions

```
def is_220(a, b, c):  
    return a==2 and b==2 and c==0
```

## Refactor Exercise 5

which refactor  
is correct?

hint: `is_220(2, 2, 1)`



```
def is_220(a, b, c):  
    if a==2:  
        return True  
    if b==2:  
        return True  
    if c==0:  
        return True  
    return False
```

A



```
def is_220(a, b, c):  
    if a!=2:  
        return False  
    if b!=2:  
        return False  
    if c!=0:  
        return False  
    return True
```

B



## Refactor Exercise 5

return **b1** and **b2** and **b3** and ... and **bN**



```
if not b1:  
    return False  
if not b2:  
    return False  
if not b3:  
    return False  
...  
if not bN:  
    return False  
return True
```

**Lesson: checking if everything is True can be translated to seeing if we can find anything False**