Control Constructs: Conditionals in Python

Prof. Pai H. Chou
National Tsing Hua University

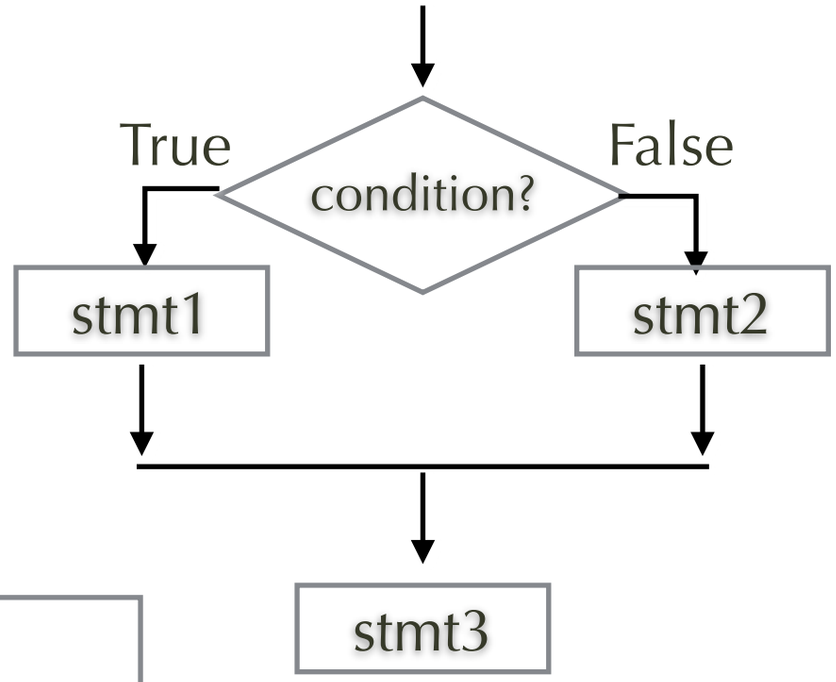
Outline

- Conditional statement
  - `if` statement
  - optional `elif`-`else` construct
- Conditional expression
  - Expr1 `if` cond `else` Expr2
- Conversion to Boolean expression

`if-else` statement

- conditional execution
  - take different actions based on condition

```
if condition:
    stmt1
else:
    stmt2
stmt3
```



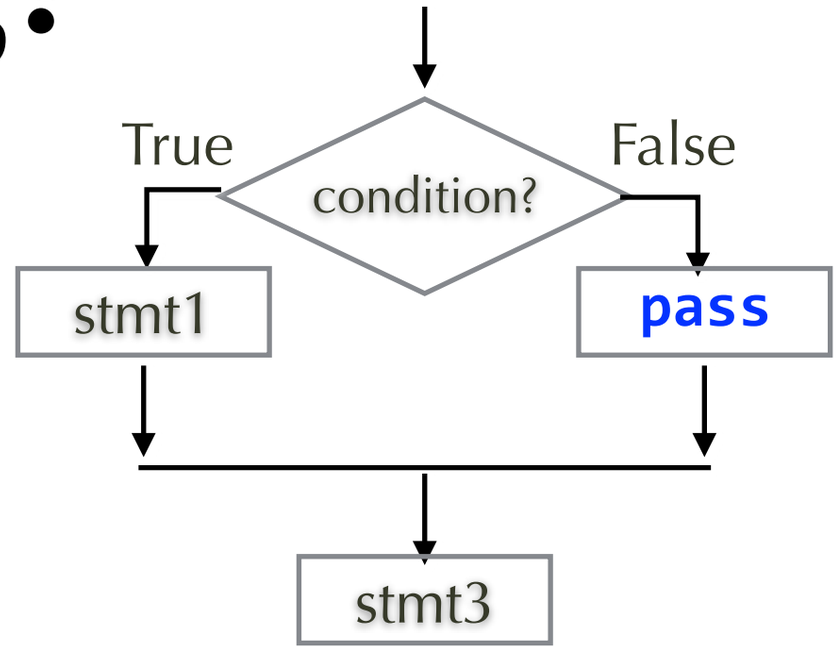
- Example: check before / 0

```
x = int(input('enter one number: '))
y = int(input('another number: '))
if y == 0:
   print('sorry, cannot divide by 0!')
else:
   print(f'{x} / {y} = {x/y}')
```

What if you want else-clause to do nothing?

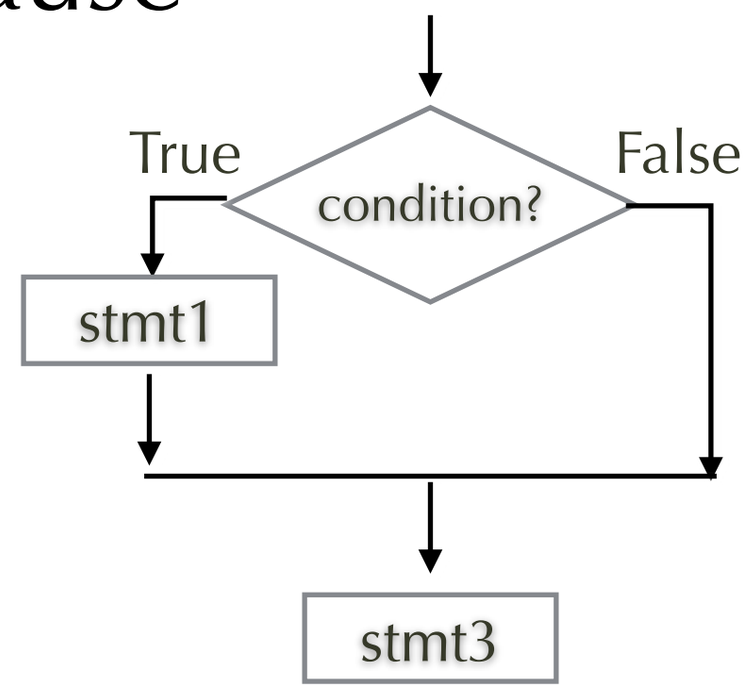
- Option 1: `pass`

```
if condition:
    stmt1
else:
    pass
stmt3
```



- Option 2: leave out else clause

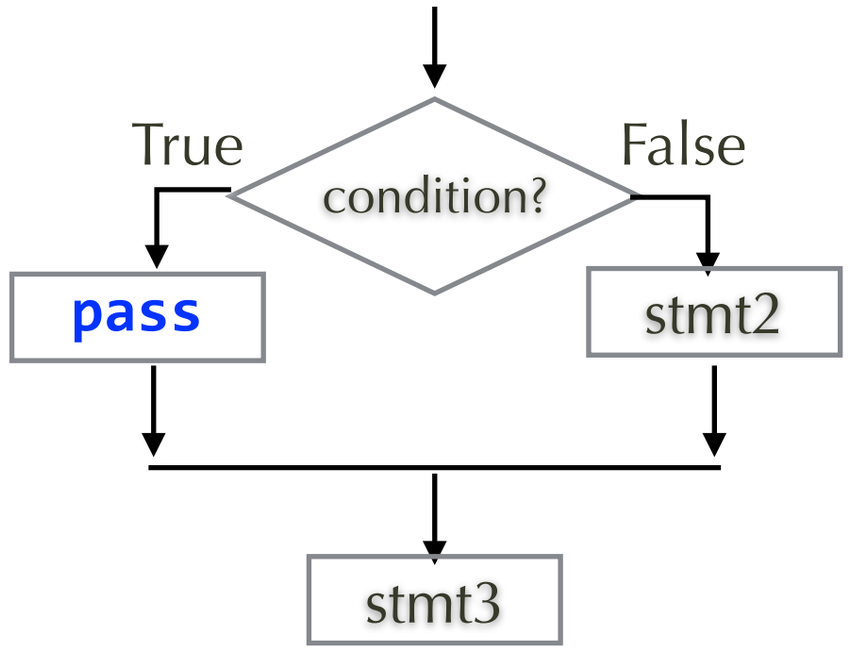
```
if condition:
    stmt1
stmt3
```



What if you want if-clause to do nothing?

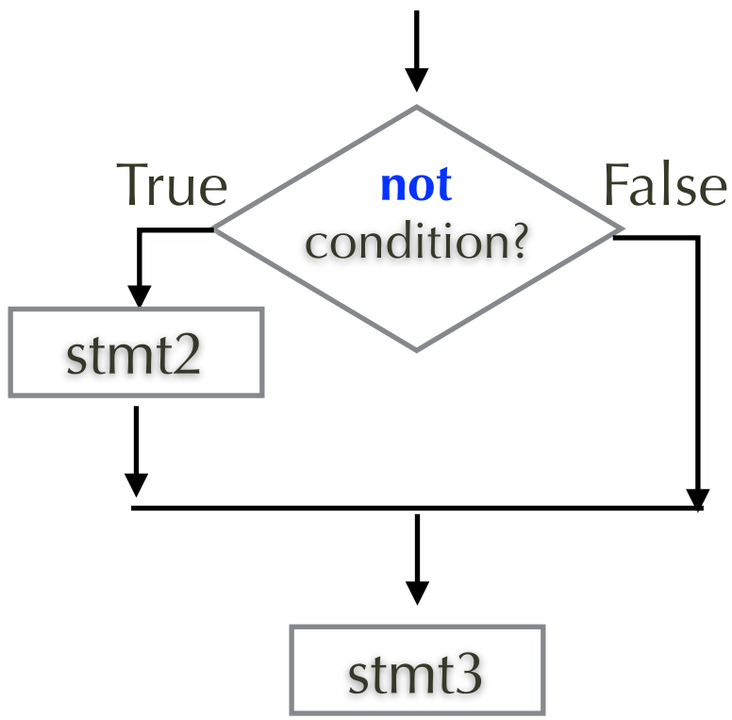
- You could do pass... but

```
if condition:
    pass
else:
    stmt2
stmt3
```



- Better to test **not** condition and leave out the else

```
if not condition:
    stmt2
stmt3
```



`pass` statement

- does nothing

```
if cond:
    pass # mandatory if you want it to do nothing
else:
    statement...
```

- Python does not allow skipping suite

```
if cond:    # no suite, followed directly by else
else:
    statement...
```

Conditional example: translation

- look up a string in multiple dictionaries
  - if found, print its definition and language
- first version: print only first found
- second version: print all versions found
- in both cases, report if not found
- Example dictionary

```
english = {'one':1, 'pan':'鍋子','cabbage':'高麗菜','pie':'派餅'}
spanish = {'uno':1, 'pan':'麵包','col':'高麗菜',    'pie':'腳'}
french =  {'une':1, 'toi':'你',  'col':'衣領',      'pie':'鵲'}
```

First version: print first word found

```
english = {'one':1, 'pan':'鍋子','cabbage':'高麗菜','pie':'派餅'}
spanish = {'uno':1, 'pan':'麵包','col':'高麗菜',    'pie':'腳'}
french =  {'une':1, 'toi':'你',  'col':'衣領',      'pie':'鵲'}
```

```
word = input('enter word to look up: ')
if word in english:
    print(f'in English: {english[word]}')
elif word in spanish:
    print(f'in Spanish: {spanish[word]}')
elif word in french:
    print(f'in French: {french[word]}')
else:
    print(f'{word} not found in dictionary')
```

- 'pan', 'pie' would match English;
- 'uno', 'col' would match Spanish
- 'une', 'toi' would match French

Second version: print all words found -- first attempt

```
english = {'one':1, 'pan':'鍋子','cabbage':'高麗菜','pie':'派餅'}
spanish = {'uno':1, 'pan':'麵包','col':'高麗菜',    'pie':'腳'}
french =  {'une':1, 'toi':'你',  'col':'衣領',      'pie':'鵲'}
```

```
word = input('enter word to look up: ')
if word in english:
    print(f'in English: {english[word]}')
if word in spanish:
    print(f'in Spanish: {spanish[word]}')
if word in french:
    print(f'in French: {french[word]}')
```

- 'one', 'cabbage' would match English only
- 'pan' would match English and Spanish
- 'col' would match Spanish and French
- 'pie' would match all three. but.... how to tell not found?

Second version: solution (a) use temporary variable to track found

- initialize temporary variable found = False

```
found = False
word = input('enter word to look up: ')
if word in english:
    print(f'in English: {english[word]}')
    found = True
if word in spanish:
    print(f'in Spanish: {spanish[word]}')
    found = True
if word in french:
    print(f'in French: {french[word]}')
    found = True
if not found:
    print(f'{word} not found in dictionary')
```

- set to True if found in any dictionary

Second version: solution (b) accumulate strings in list, print together

- initialize list to empty list

```
outList = []
```

```
word = input('enter word to look up: ')
if word in english:
    outList.append(f'in English: {english[word]}')
if word in spanish:
    outList.append(f'in Spanish: {spanish[word]}')
if word in french:
    outList.append(f'in French: {french[word]}')
```

```
if not outList:
    print(f'{word} not found in dictionary')
else:
    print('\n'.join(outList))
```

- `outList` tracks output strings; empty indicates not found. print only at the end.

Conversion to Function: identify code by purpose

- reorganize original code into sections

```
word = input('enter word to look up: ')
```

input

```
outList = []
if word in english:
    outList.append(f'in English: {english[word]}')
if word in spanish:
    outList.append(f'in Spanish: {spanish[word]}')
if word in french:
    outList.append(f'in French: {french[word]}')
```

compute

```
if not outList:
    print(f'{word} not found in dictionary')
else:
    print('\n'.join(outList))
```

output

Conversion to Function

- caller with I/O

```
word = input('enter word to look up: ')
outList = LookupDicts(word)
if not outList:
    print(f'{word} not found')
else:
    print('\n'.join(outList))
```

- callee (reusable code)
- assuming english, spanish, french are defined

```
def LookupDicts(word):
    outList = []
    if word in english:
        outList.append(f'in English: {english[word]}')
    if word in spanish:
        outList.append(f'in Spanish: {spanish[word]}')
    if word in french:
        outList.append(f'in French: {french[word]}')
    return outList
```

Conditional Expressions

- Syntax:
  Expr1 **`if`** *cond* **`else`** *Expr2*

- Example:

```
if not outList:
    print(f'{word} not found')
else:
    print('\n'.join(outList))
```

- can be converted into

```
print(f'{word} not found' if not outList \
      else '\n'.join(outList))
```

  - equivalently,

```
print('\n'.join(outList) if outList \
      else f'{word} not found' )
```

Cascaded Conditional Expressions

- Syntax:

  Expr1 **`if`** *cond1* **`else \`**

  Expr2 **`if`** *cond2* **`else \`**

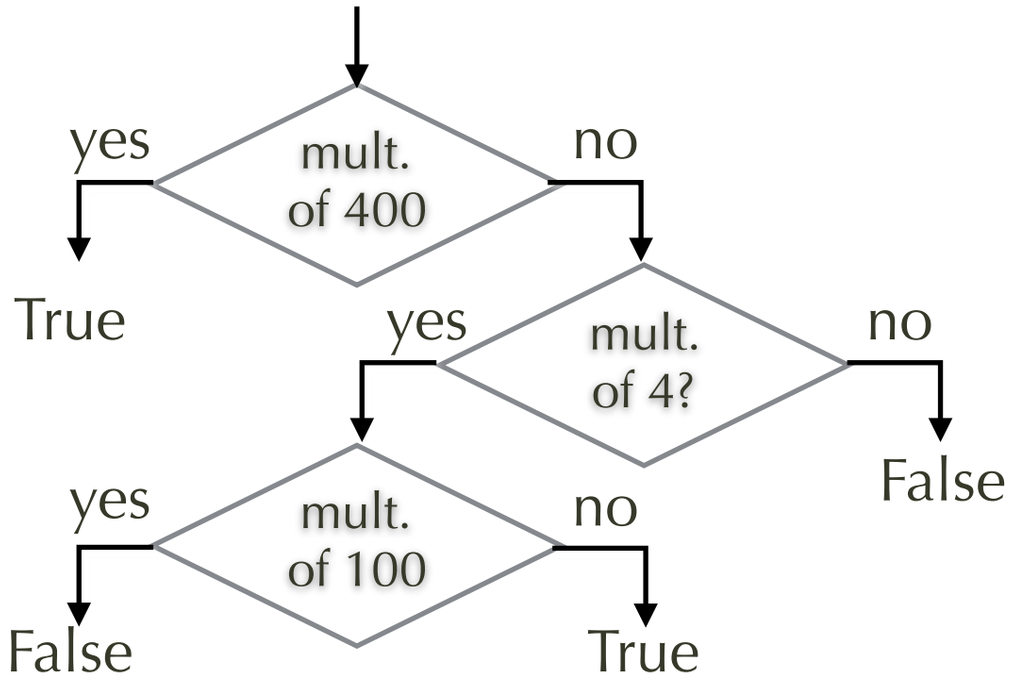
  ...*ExprN*

- Example: number of days in a month

```
def DaysInMonthYear(month, year):
    return 31 if month in {1, 3, 5, 7, 8, 10, 12} else \
           30 if month in {4, 6, 9, 11} else \
           28 if not leap(year) else \
           29
```

Example: Leap year

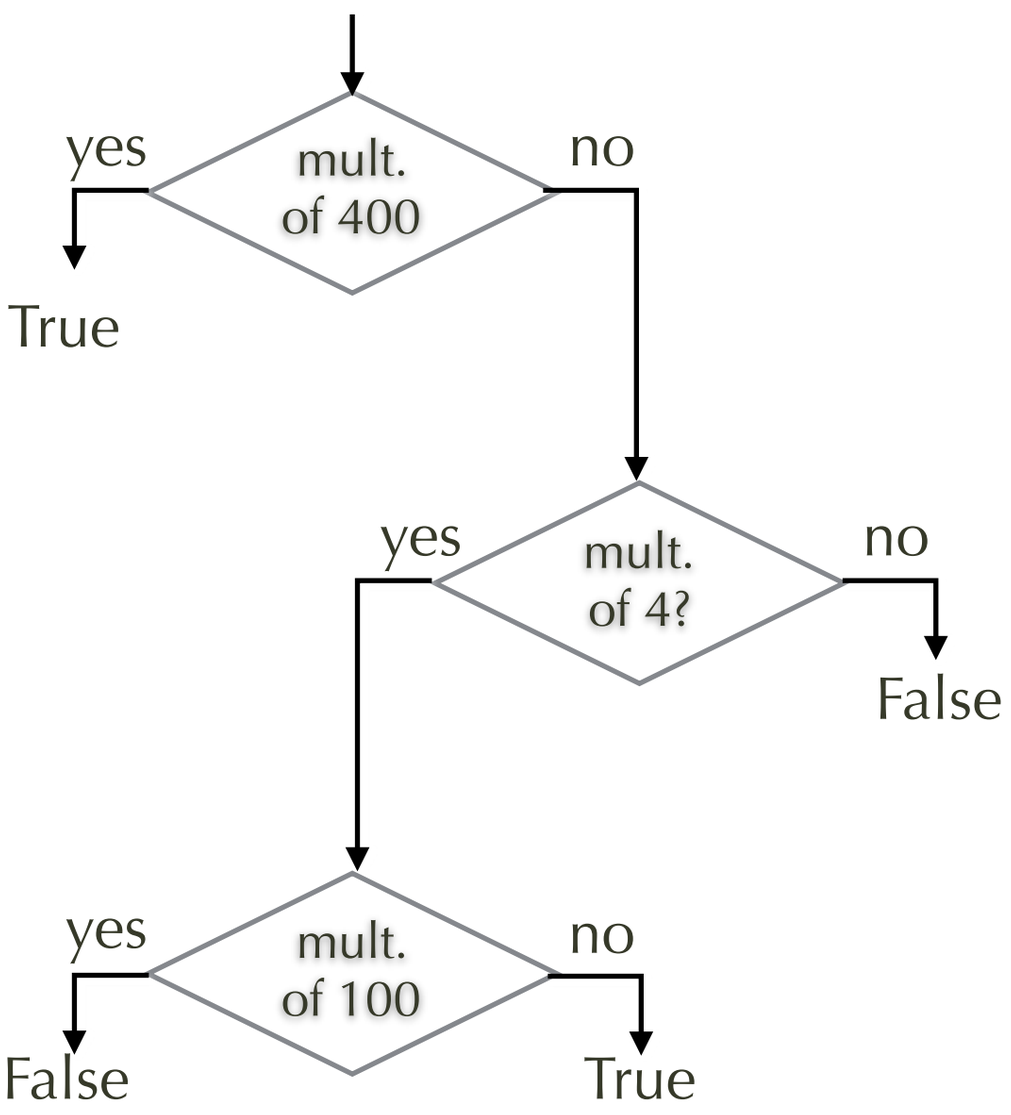
- multiple of 400; otherwise,
  multiple of 4 but not multiple of 100
  - e.g., 1600, 2000, 1984, but not 1700, 1800, ..

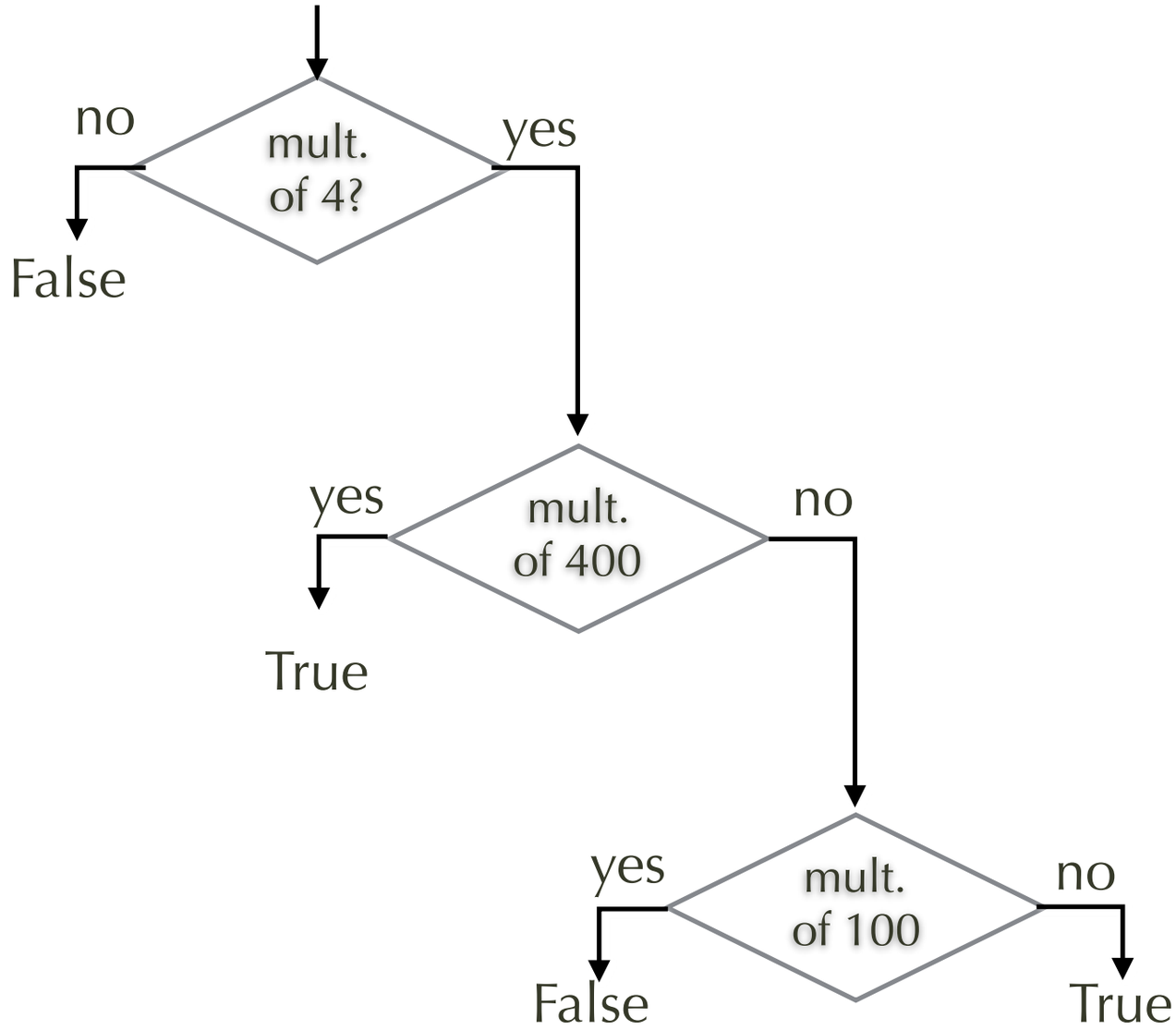



```
>>> leap(1600)
True
>>> leap(1700)
False
>>> leap(1800)
False
>>> leap(1900)
False
>>> leap(2000)
True
```

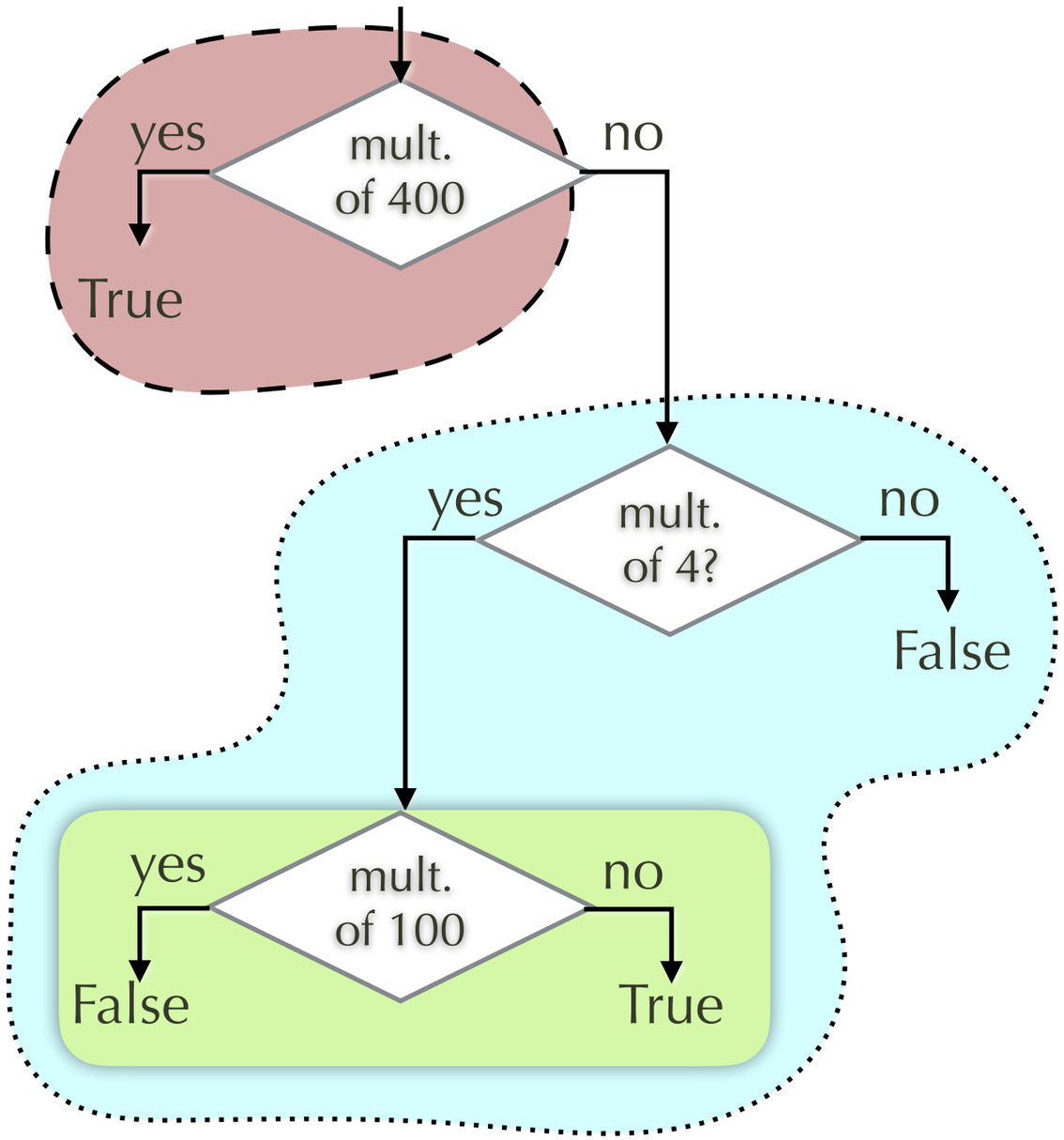
```
>>> leap(2001)
False
>>> leap(2002)
False
>>> leap(2003)
False
>>> leap(2004)
True
>>> leap(2005)
False
```

Equivalent flowcharts





Translation from flowchart




```
def leap(year):
    if (year % 400 == 0):
        return True
    else:
        if (year % 4 == 0):
            if (year % 100 == 0):
                return False
            else:
                return True
        else:
            return False
```

`else if` vs. `elif`

- `else:`

  `if` *condition*...

  can be changed into `elif` *condition*...

- why? one less level of indentation!

```
def leap(year):
    if (year % 400 == 0):
        return True
    else:
        if (year % 4 == 0):
            if (year % 100 == 0):
                return False
            else:
                return True
        else:
            return False
```

same as

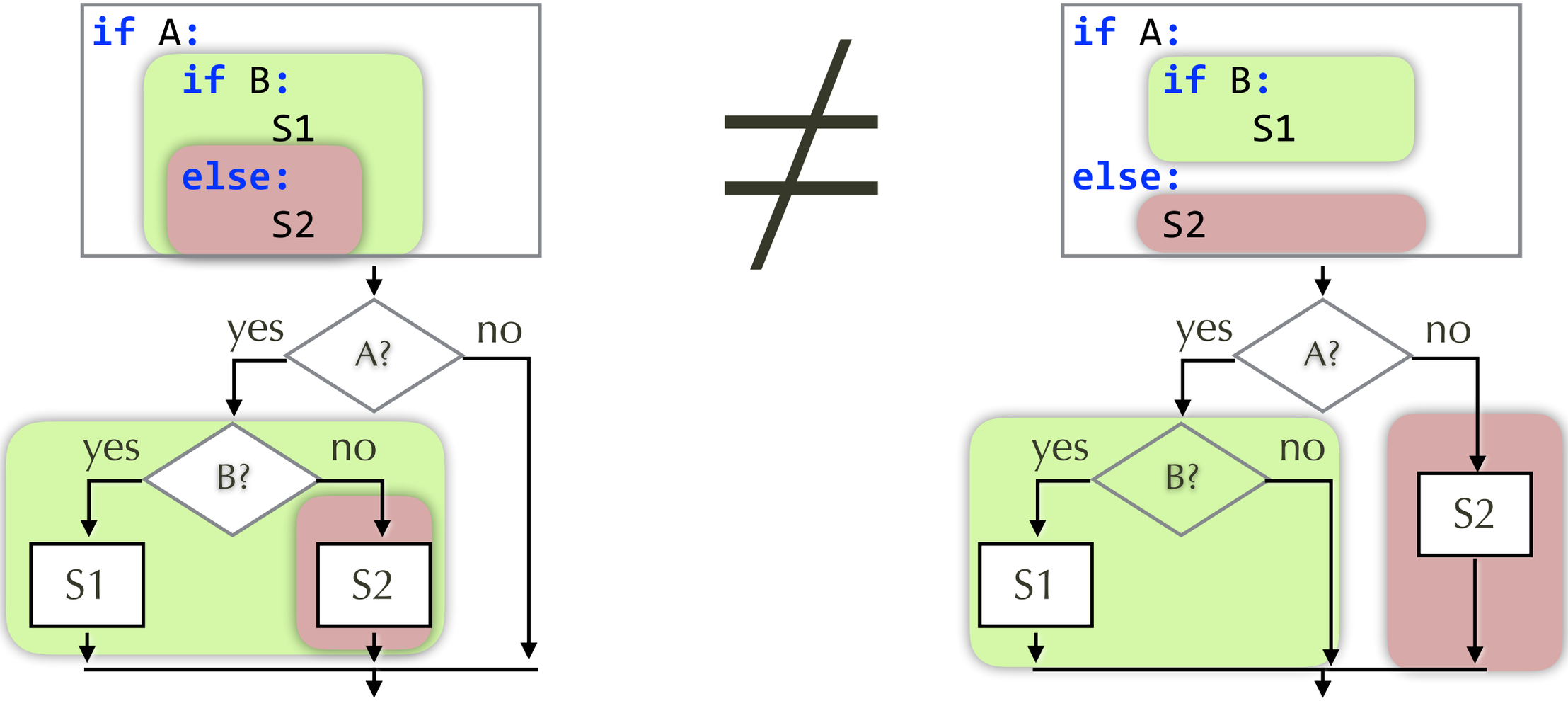
```
def leap(year):
    if (year % 400 == 0):
        return True
    elif (year % 4 == 0):
        if (year % 100 == 0):
            return False
        else:
            return True
    else:
        return False
```

`else if` vs. `elif`

- Invalid syntax to say on the same line `else if` *condition*...`:`
  - `else:` (including colon) must be on its own line
- Must use `elif` *condition*...
  - this is a Python language rule!

`if-else` matching

- `else`, `elif` are optional
- `else` or `elif` must be matched up with an `if` or another `elif` by indentation



Functions returning boolean

- Could eliminate the if-else statement by just returning the boolean expression!


```
return (year % 100 != 0)
```

```
def leap(year):
    if (year % 400 == 0):
        return True
    else:
        if (year % 4 == 0):
            if (year % 100 == 0):
                return False
            else:
                return True
        else:
            return False
```

Eliminating if-else when returning Boolean: converting to `and`

- **`if`** *cond1*:
  `return` *cond2*
  `else:`
  `return` *False*

- same as
  `return` *cond1* **`and`** *cond2*

c1	c2	res
T	T	T
	F	F
F	T	F
	F	F

same truth tables!

c1	c2	and
T	T	T
T	F	F
F	T	F
F	F	F

Conversion to `and`

- convert to

  `return` *cond1* **`and`** *cond2*

```
def leap(year):
    if (year % 400 == 0):
        return True
    else:
        if (year % 4 == 0):
            return (year % 100 != 0)
        else:
            return False
```

```
return (year % 4) == 0) and (year % 100 != 0)
```

Eliminating if-else when returning Boolean: converting to `or`

- **`if`** *cond1*:
  `return True`
  `else:`
  `return` *cond2*

c1	c2	res
T	T	T
	F	T
F	T	T
	F	F

- is the same as
  `return` *cond1* **`or`** *cond2*

c1	c2	or
T	T	T
	F	T
F	T	T
	F	F

Conversion to `or`

- convert to

  `return` *cond1* **`or`** *cond2*

```
def leap(year):
    if (year % 400 == 0):
        return True
    else:
        return (year % 4) == 0) and (year % 100 != 0)
```



```
return (year % 400 == 0) or \
       (year % 4) == 0) and (year % 100 != 0)
```

- => no **`if`** statement in this case!