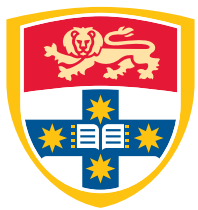
INFO1110 & COMP9001: Introduction to Programming

School of Information Technologies, University of Sydney



Copyright Warning

COMMONWEALTH OF AUSTRALIA

Copyright Regulations 1969

WARNING

This material has been reproduced and communicated to you by or on behalf of the University of Sydney pursuant to Part VB of the Copyright Act 1968 (**the Act**).

The material in this communication may be subject to copyright under the Act. Any further copying or communication of this material by you may be the subject of copyright protection under the Act.

Do not remove this notice.

Week 3: Control Flow 1: Branching and Looping

We will cover: Branching with `if` and `else`, Looping with `while` and more on handling numbers: `if`/`else`, `switch`

You should read: §§1.2 and 1.3 of Sedgewick

Lecture 5: Control Flow 1: `if`

Handling simple choices

Evaluating multiple conditions

Boolean expressions

Boolean logic

Operation	*Meaning*
x and y	true if both x and y are true, false otherwise
x or y	true if both x or y or both are true, false otherwise
not x	true if x is false
x == y	true if x and y are both true or both false, false otherwise
x != y	true if x is true and y is false, or x is false, false otherwise

Complex boolean expressions

Ask a simple question that has a yes/no answer.

Can depend on sub problems to be solve first.

Question: Ready to go out?

depends on: have keys? have phone? if it is raining, do I have umbrella?

```
1 have_keys = True
2 have_phone = True
3 is_raining = True
4 have_umbrella = False
5
6 ready_to_leave = ?
```

What are all the cases is readyToLeave `True` or `False`. Use your truth tables!

Complex boolean expressions

Question: Ready to go out?

depends on: have keys? have phone? if it is raining, do I have umbrella?

```
1 readyToLeave = ( have_keys and have_phone ) and \
2     ( ( is_raining and have_umbrella ) or ( not is_raining ) )
```

Complex boolean expressions

What is the boolean result?

```
1  # minimum needed for cake
2  # 1/2 cup butter
3  # 3/4 cup white sugar
4  # 1/2 cup cocoa powder
5  # 3 eggs
6  # 1 teaspoon vanilla extract
7  # 4 squares chopped chocolate (optional)
8
9  can_make_cake = ???
10
11 print("can_make_cake = " + str(can_make_cake))
```

Pay careful attention to the specification first, then derive the code.

We need to be careful to evaluate the expressions as intended.

Casting

Treating variables as other types

Casting primitive

Operators used in an expression can be evaluated differently based on the data types involved.

What is the result?

```
1 litres = 2
2 portion = 0.330
3 persons = litres / portion
4 print("invite " + str(persons) + " persons")
```

casting allows the programmer to explicitly tell the compiler to treat a variable, or expression, as another type.

Casting primitive types

Convert integer to floating point number

```
1 litres = 2
2 portion = 0.330
3 persons = int(litres / portion) # change the type
4 print("invite " + str(persons) + " persons")
```

The opposite is also possible. What is the result?

```
1 litres = float(2)
2 persons = float(5)
3 portion = litres / persons
4 print("each person drinks " + str(portion) + "mL")
```

Can we cast in different places, where is the best?

if

(AN UNMATCHED LEFT PARENTHESIS
CREATES AN UNRESOLVED TENSION
THAT WILL STAY WITH YOU ALL DAY.

Source: xkcd

The `if` statement

- What's it for?
  - to have different behaviour in a program based on whether something is true or false (this is a kind of *control statement*).
- What does it look like?

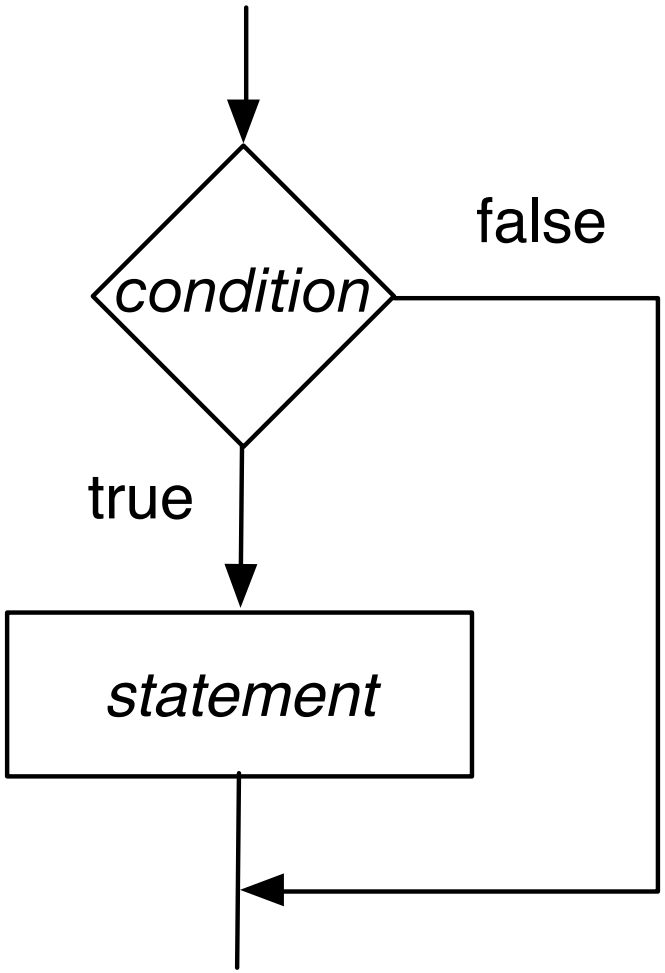
```
if condition :
    statement
```

- *condition* is a boolean expression that evaluates to `True` or `False`
- *statement* is one computer instruction or a sequence of computer instructions within the same indentation block
- In code this looks like

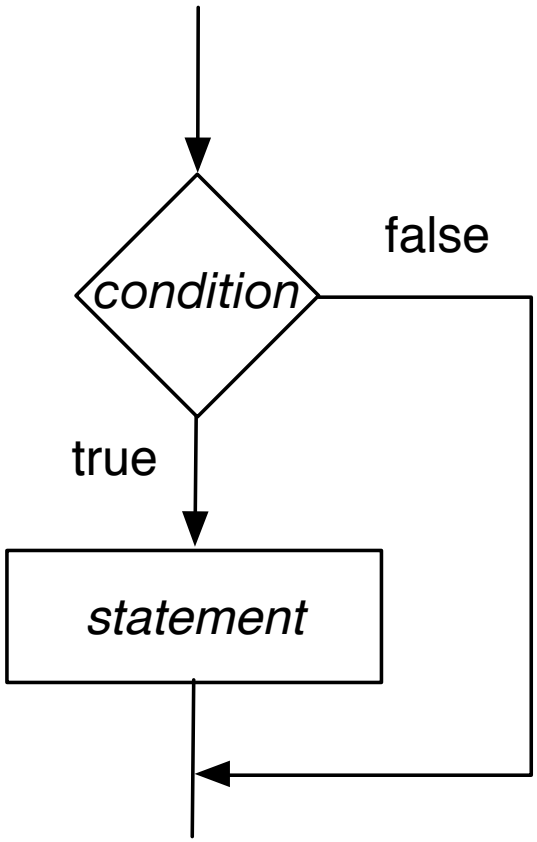
```
1    if x > 0 :
2        print("x is positive")
```

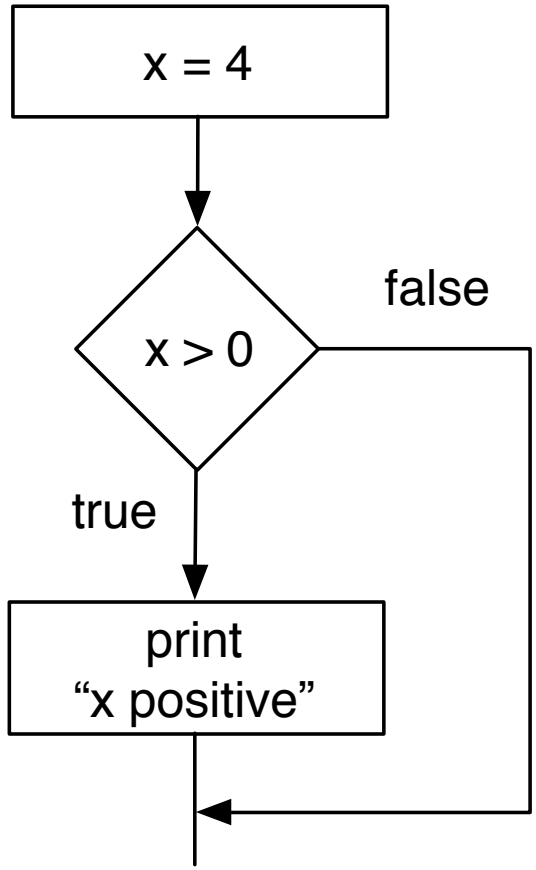
How `if` works

The “if” statement is a simple control flow structure: it is used to test the value of an expression, to see whether it’s true, and if true, then to do something else.



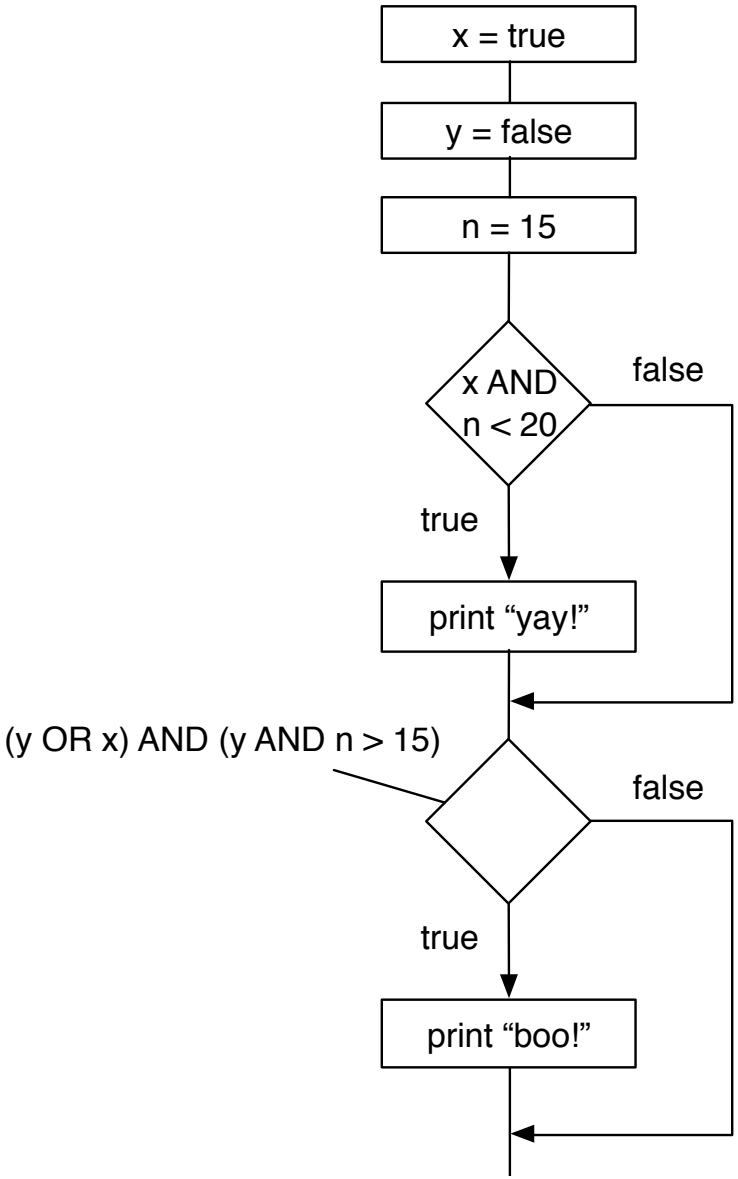
How `if` works


condition
false
true
statement

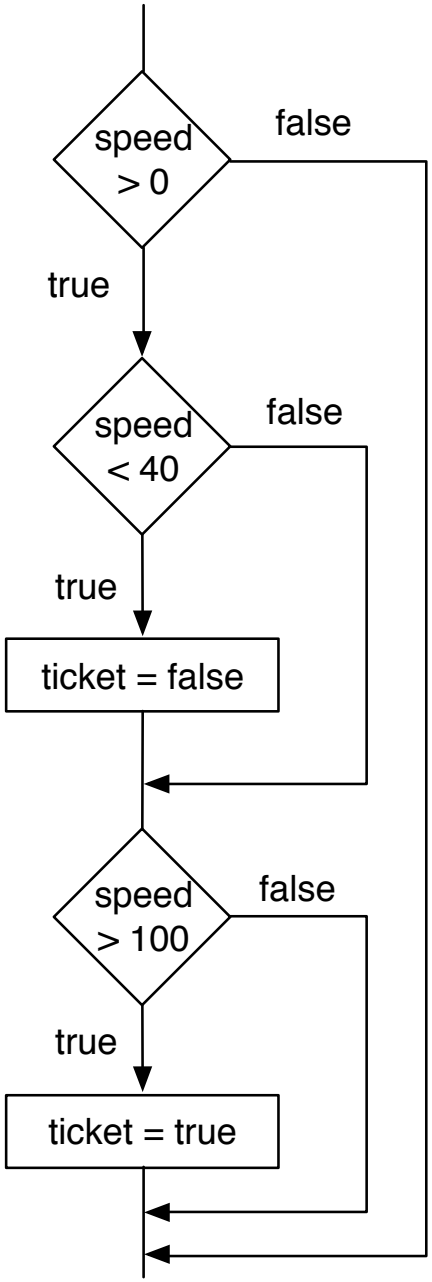

x = 4
x > 0
false
true
print
“x positive”

More logic with `if`

The *condition* can be a complex boolean expression. What is the code?



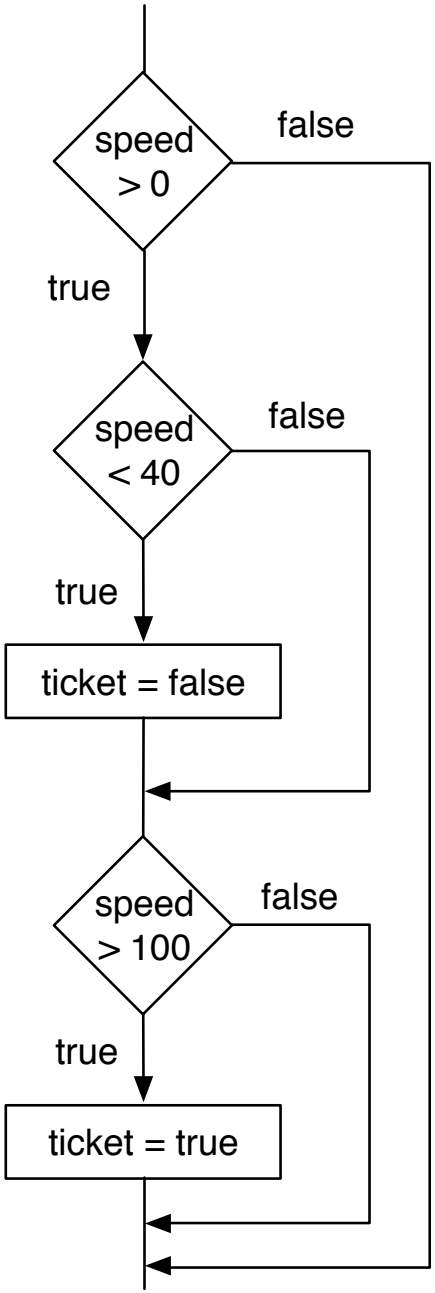
Nested `if`



You can put `if`s inside `if`s, like this:

```
1 ticket = False
2 if speed > 0:
3   if speed < 40:
4     ticket = False
5   if speed > 100:
6     ticket = True
```

Deskcheck



Control statements like `if` can lead to different statements being executed, code paths.

The variables will have different values depending on the current conditions

	speed	ticket	
	0	F	

Extending `if`

`if` you had to write an "`if`" for each possible outcome you can easily miss a case:

```
        if x > 0 :
            # do something

        if x <= 0 :
            # do something else
```

what is the opposite case?

```
if (i == 7 and j < i) or \
    ( (not ((j - i) < 5)) and (j != 0) ):
    # do something

if (i != 7 or j >= i))) and \
    ( (not ((j - i) >= 5)) or (j == 0) ) ):
    # do something else
```

Extending `if` (cont.)

`if` you had to write an "`if`" for each possible outcome you can easily miss a case:

```
        if x > 0 :
            # do something

        if not (x < 0) :
            # do something else
```

```
if (i == 7 and j < i) or \
    ( (not ((j - i) < 5)) and (j != 0) ):
    # do something

if not ((i == 7 and j < i) or \
    ( (not ((j - i) < 5)) and (j != 0) ) ):
    # do something else
```

Extending `if` (cont.)

The keyword `pass` can help make an *empty* statement. It does NOTHING, but it is a syntactically correct statement

```
        if x > 0 :
            pass
```

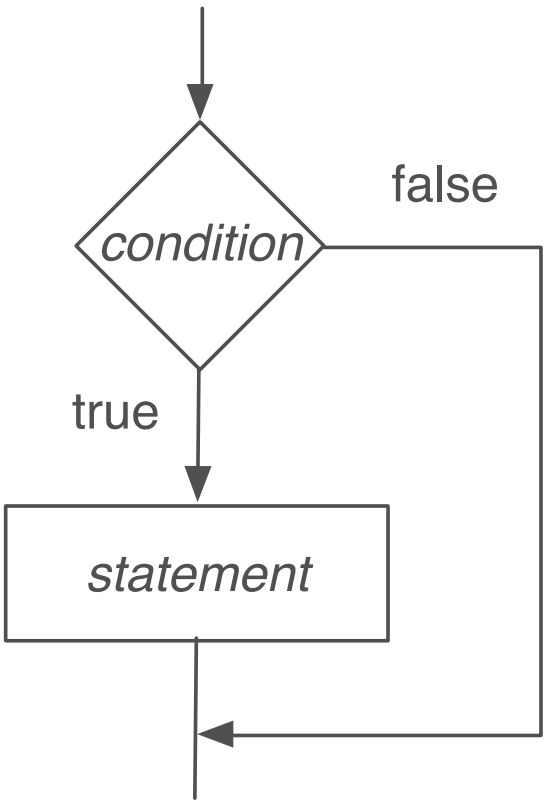
why? Simplicity and readability.

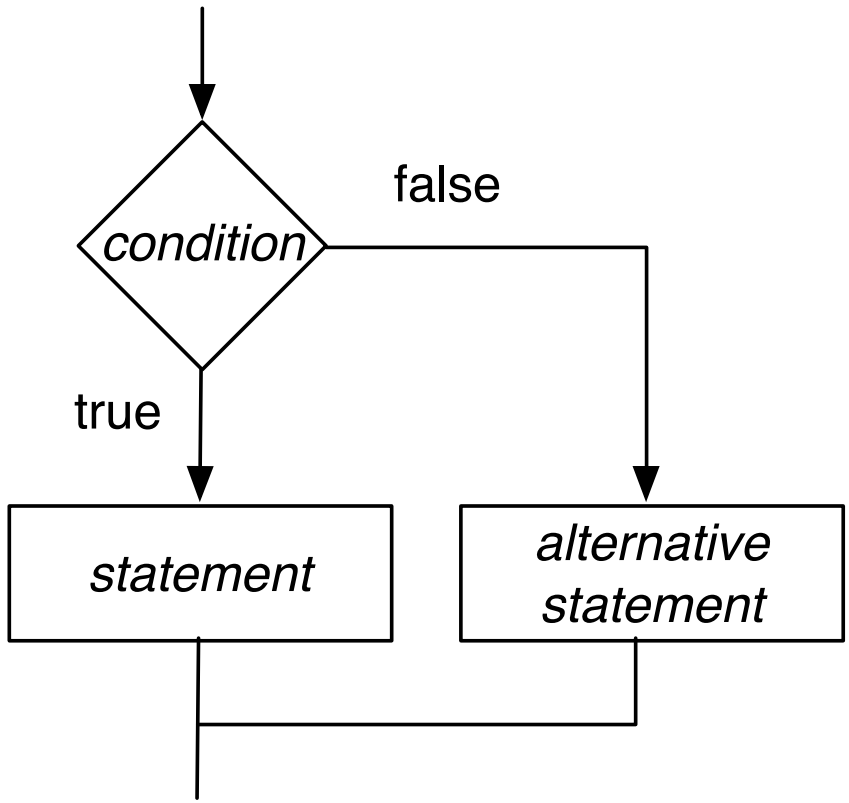
```
if (i == 7 and j < i) or \
    ( (not ((j - i) < 5)) and (j != 0) ):
    # do nothing when condition is True
    pass
else:
    # do something when condition is False
```

if ... else

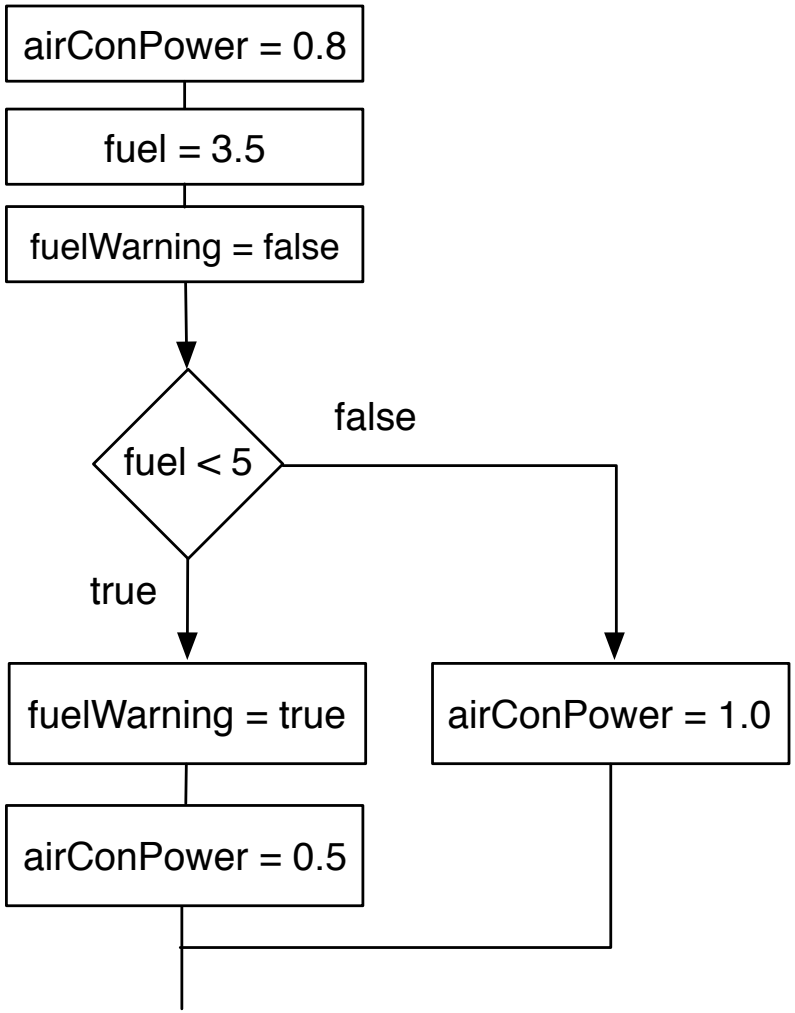
The alternative to writing both cases out explicitly is to use `else`.
Here's the syntax:

```
1   if (condition) then
2   ·   statement
3   else
4   ·   alternative statement
```

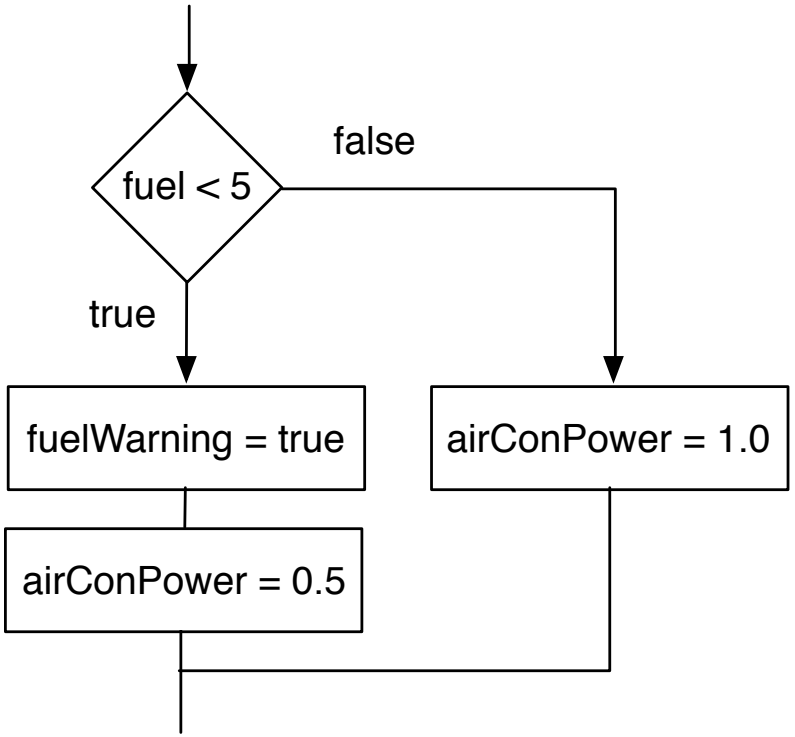




Example


airConPower = 0.8
fuel = 3.5
fuelWarning = false
fuel < 5
false
true
fuelWarning = true
airConPower = 1.0
airConPower = 0.5

Deskcheck 2



Don't always know values ahead of time

Use desk check to track each variable value and how it changes

As part of testing, you should pick values that are normal, abnormal, on the boundary etc.

fuel	airConPower	fuelWarning
0		
5		

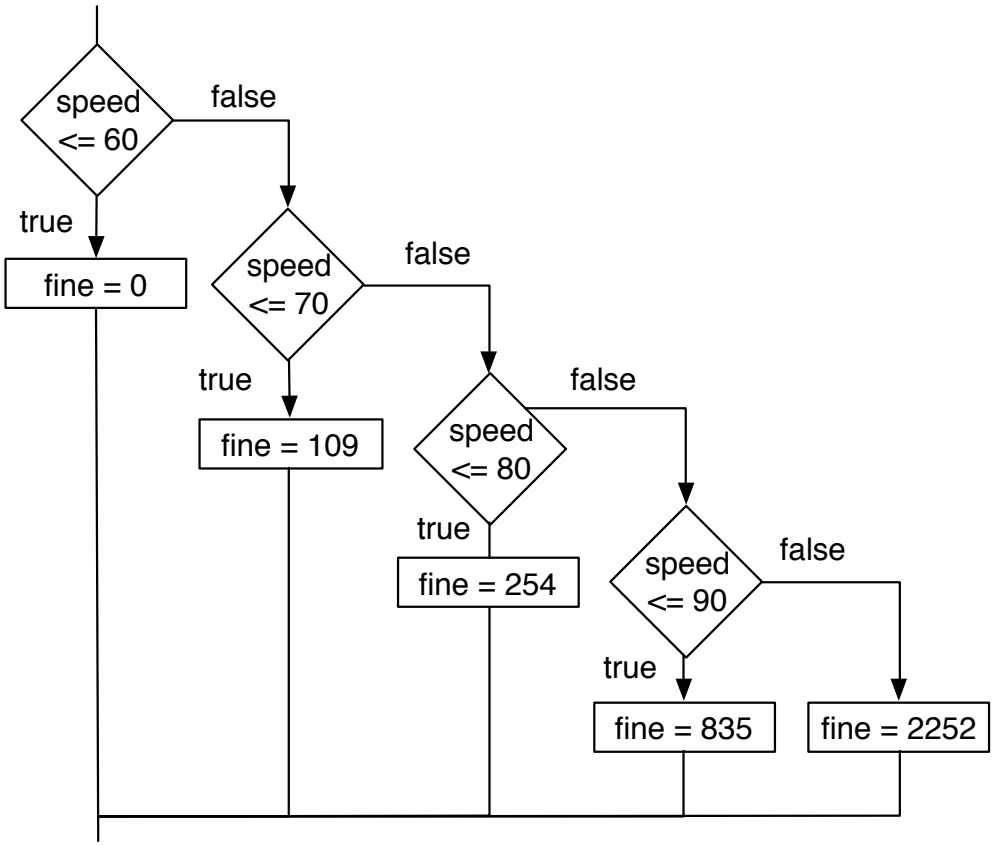
if ... elif ... elif ...

You can have separate conditions checked in sequence. `if` statements with the alternative statement as `elif` (short for else if):

```
1 if ch == 'a':
2   # do thing 1
3 elif ch == 'b':
4   # do thing 2
5 elif ch == 'c':
6   # do thing 3
```

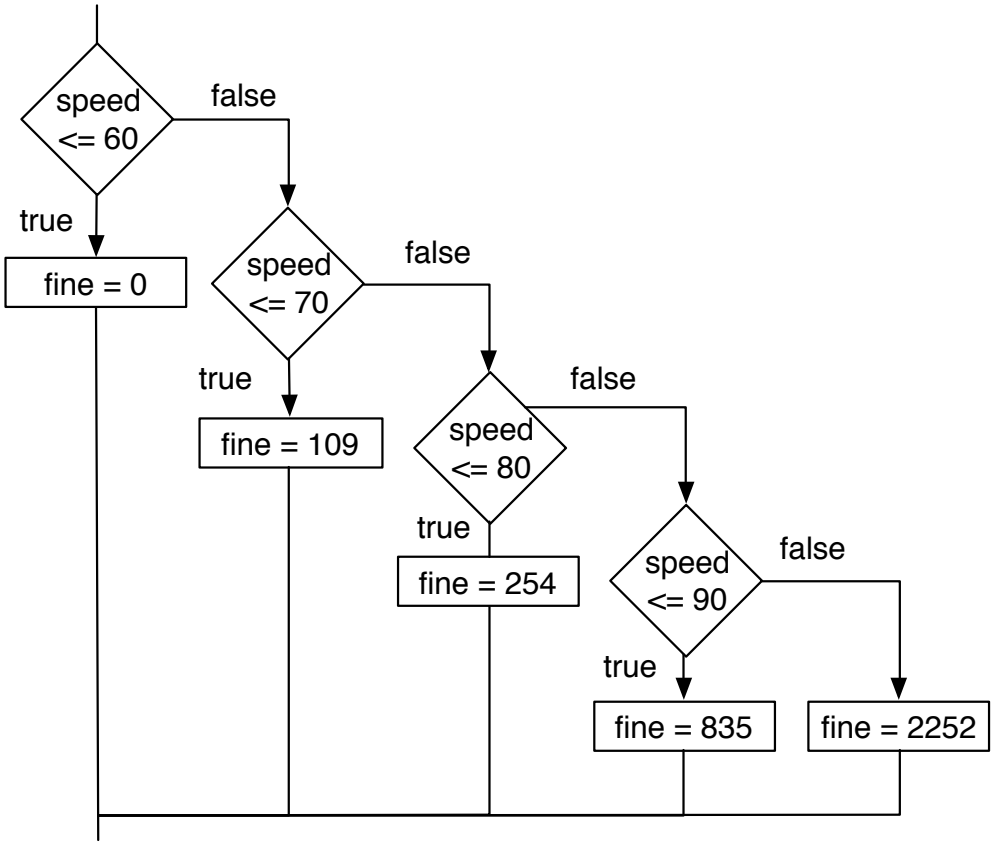
It is equivalent to putting the expressions after the `else`s into their own separate blocks.

An example of checking conditions in a specific order




```
1  if speed <= 60:
2    fine = 0
3  elif speed <= 70:
4    fine = 109
5  elif speed <= 80:
6    fine = 254
7  elif speed <= 90:
8    fine = 835
9  else:
10   fine = 2252
```

Deskcheck 3



speed	fine
-789	