Lecture 17

Topics: *Chapter 8. Loop Structures and Booleans*

8.3 (Continues) nested loops
8.4. Computing with booleans
8.5 Other common structures: post-test, loop and half.

8.3 Common Loop Patterns

Nested Loops

Loops may contain loops. Let's take a look at the program that reads in all the numbers from a file, where numbers are separated by a white space or by a “next line” character.:

1 2 34 12 0 -12 23
3 5 456 23 09 8
1 4 -23 45 -89

8.3 Common Loop Patterns

Nested Loops

Loops may contain loops. Let's take a look at the program that reads in all the numbers from a file, where numbers are separated by a white space or by a “next line” character.:

```
1  2  34  12  0  -12  23
3  5  456  23 09  8
1  4  -23  45  -89
```

```
numbers = [] #list of all numbers from the file

for line in Source:# iterate over lines in file
  nums_in_line=line.split() #split by space

  for item in nums_in_line: # iterate over items
    numbers.append(int(item)) # in line
```

See program read_all_numbers.py

8.4 Computing with booleans

Now we have two control structures, `if` and `while`, that use conditions, which are Boolean expressions.

Boolean expression is either True (1) or False (0).

So far we used expressions that compare two values:

`>=`, `<=`, `!=`, `==`, `>`, `<`

in the last program (read_all_numbers.py) you saw: `... and ...`

8.4 Computing with booleans

Boolean operators

- are used to combine Boolean expressions to get a new Boolean expression.

AND (∧) <expr1> and <expr2> <expr1> && <expr2>
OR (∨) <expr1> or <expr2> <expr1> || <expr2>
NOT (¯ , ¬) not <expr> ! <expr>

and and or are binary operators, not is a unary operator.

The and of two expressions is **true** when both expressions are true.
The or of two expressions is **true** when at least one of the expressions is true.
The not operator computes the **opposite** of a Boolean expression.

8.4 Computing with booleans

Truth tables for boolean operators

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

Precedence rules (from high to low): `not`, `and`, `or`

Example:

a and b or not a and b *is equivalent to* (a and b) or ((not a) and b)

8.4 Computing with booleans

Properties of boolean operations

Distributive rules:

```
a or (b and c) == (a or b) and (a or c)
a and (b or c) == (a and b) or (a and c)
```

A double negation rule: `not (not a) == a`

DeMorgan's laws:

```
   not(a or b)  = (not a) and (not b)
   not(a and b) = (not a) or (not b)
```

Boolean algebra (Boolean logic) is the algebra of truth values and operations on them. It was developed by George Boole in the late 1830s.

One application of Boolean algebra is the analysis and simplification of Boolean expressions.

8.4 Computing with booleans

Properties of boolean operations

Distributive rules:

$a \vee (b \wedge c) == (a \vee b) \wedge (a \vee c)$

$a \wedge (b \vee c) == (a \wedge b) \vee (a \wedge c)$

A double negation rule: $\neg (\neg a) == a$

DeMorgan's laws:

$\neg(a \vee b) = (\neg a) \wedge (\neg b)$

$\neg(a \wedge b) = (\neg a) \vee (\neg b)$

Boolean algebra (Boolean logic) is the algebra of truth values and operations on them. It was developed by George Boole in the late 1830s.

8.4 Computing with booleans

Example:
Let's write a program that takes a temperature value as an input, and output where it is hot, warm, cold or freezing today.
Assume that if it is above 90F then it is hot;
if it is between 70F and 90F, then it is warm;
it it is between 32F and 70F, then it is cold;
and if it is bellow 32F then it is freezing.

8.4 Computing with booleans

Example:
Let's write a program that takes a temperature value as an input, and output where it is hot, warm, cold or freezing today.
Assume that if it is above 90F then it is hot;
if it is between 70F and 90F, then it is warm;
it it is between 32F and 70F, then it is cold;
and if it is bellow 32F then it is freezing.

```
if T > 90
   output HOT
if 70 <= T <= 90
   output WARM
if 32 <= T < 70
   output COLD
if  T < 32
   output FREEZING
```

8.4 Computing with booleans

Example:
Let's write a program that takes a temperature value as an input, and output where it is hot, warm, cold or freezing today.
Assume that if it is above 90F then it is hot;
if it is between 70F and 90F, then it is warm;
it it is between 32F and 70F, then it is cold;
and if it is bellow 32F then it is freezing.

```
if T > 90
   output HOT
if T <= 90 and T >= 70
   output WARM
if T < 70 and T >= 32
   output COLD
if  T < 32
   output FREEZING
```

see temperature.py

8.4 Computing with booleans

Example:
Let's write a program that takes a temperature value as an input, and output where it is hot, warm, cold or freezing today.
Assume that if it is above 90F then it is hot;
if it is between 70F and 90F, then it is warm;
it it is between 32F and 70F, then it is cold;
and if it is bellow 32F then it is freezing.

```
if T > 90
   output HOT
if T <= 90 and T >= 70
   output WARM
if T < 70 and T >= 32
   output COLD
if  T < 32
   output FREEZING
```

if we want to allow user to enter temperatures as many times as he/she wants we will use indefinite loop

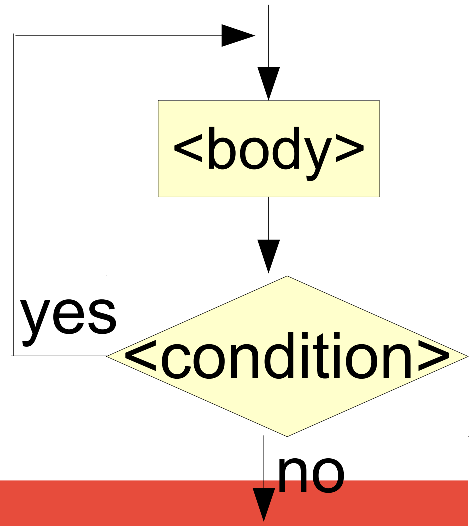
see temperature_infiniteLoop.py

8.5 Other common structures: post-test, loop and half

Post-test loop

the decision structure (`if`) along with the infinite (pre-test) loop (`while loop`) provide a complete set of control structures. Every algorithm can be expressed using just these.

Sometimes, for certain kinds of problems, alternative structures are more convenient (`for loop` and **`post-test loop`**)



8.5 Other common structures: post-test, loop and half

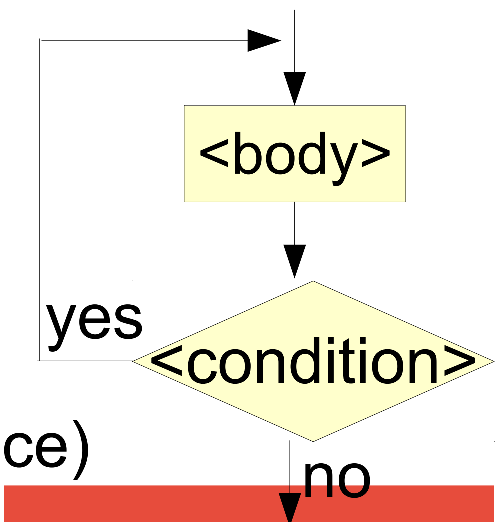
Post-test loop

the decision structure (`if`) along with the infinite (pre-test) loop (`while loop`) provide a complete set of control structures. Every algorithm can be expressed using just these.

Sometimes, for certain kinds of problems, alternative structures are more convenient (`for loop` and **`post-test loop`**)

Syntax could be:

```
repeat
   <body>
until <condition>
```



- the condition test comes <u>after</u> the loop body (the body of the loop is always executed at least once)

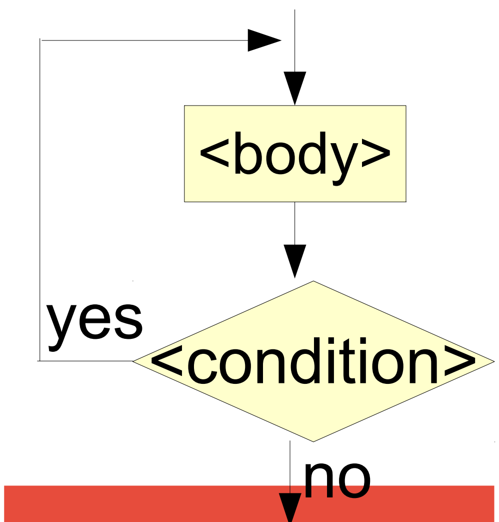
8.5 Other common structures: post-test, loop and half

Post-test loop

the decision structure (`if`) along with the infinite (pre-test) loop (`while loop`) provide a complete set of control structures. Every algorithm can be expressed using just these.

Sometimes, for certain kinds of problems, alternative structures are more convenient (`for loop` and **`post-test loop`**)

Python doesn't have a statement that directly implements a post-test loop.



8.5 Other common structures: post-test, loop and half

Post-test loop

We can simulate the post-test loop:

initialize the variable(s) that are used in the while's condition to such value(s) that the while loop is entered.

OR

use `while True` and `break` statement

`while True` is an infinite loop, condition is always true
`break` statement terminates a loop

8.5 Other common structures: post-test, loop and half

Post-test loop

We can simulate the post-test loop:

initialize the variable(s) that are used in the while's condition to such value(s) that the while loop is entered.

see programs from previous lecture:

average_i.py
```
answer = "yes"
while answer[0]=='y':
```

average_s.py
```
next_value=0
while next_value != -1000:
```

average_s_mod.py
```
ns='0'
while ns != "":
```

8.5 Other common structures: post-test, loop and half

Post-test loop

We can simulate the post-test loop:

see temperature_infiniteLoop.py

use `while True` and `break` statement

`while True` is an infinite loop, condition is always true
`break` statement terminates a loop

8.5 Other common structures: post-test, loop and half

Post-test loop

Try to **avoid** peppering the body of the loop with **multiple break statements**, because the logic of the loop might be lost.

However, there are times when this rule should be broken to provide most elegant solution.