# Roles of variables
# Flag variables

CMPT 120
Class Feb 7, 2014
By Diana Cukierman, diana@sfu.ca

## Concept: Roles of variables

- Index variables, for ex:
  - Index in a for loop
  - indexing a string
- To receive a value from the user
- To do some intermediate calculation
- Accumulator
  - Special case: counter (adding 1 each time)
- Flag

## Flag variables:

- Basic case
  - Two values (values may be : strings, numbers, Boolean, a real flag position. Examples:
    “yes”/“no”, 0/1, True/False, “up”/”down”
  - We want to detect if a condition occurred or not

  and

  - REMEMBER (and be able to check) what happened (the value of the flag reminds us!)

## Flag variables (cont.)

- A flag variable could also provide more information, some value associated to the condition happening
  - E.g. Remember (keep a value) of a position where some condition happened while searching in a string

## Flag variables: useful in..

- Detecting if some condition changes as a loop executes, sometime during the iterations and the condition happens not necessarily in every iteration!
- If we do things correctly, the flag should remember what happened before, i.e. allowing the programmer to detect that the condition indeed happened (the detection can occur inside or outside the loop)

## Combining for loops, if statements and Flag variable

- Demo with volunteers: searching /checking a condition among a sequence of people: is the person wearing glasses?

## Flags (cont.)

One possibility (no need of flags):

- Visit one person after the other…
- detect condition/situation (has glasses)
- act right away, for example, informing (print) about that specific person

## Flags (cont.)

Another possibility

- Visit one person after the other…
- detect condition/situation (has glasses)
- REMEMBER what happened
- AFTER everyone was visited: act, decide based on what was remembered

## Clarifications:

- Boolean values: True/False (with no quotes)

```
>>> 5 ==5
True
>>> x=0
>>> x >0
False
```

Boolean value, no quotes

"True" ~ is a string: "T" + "r" + "u" + "e"

## Trace by hand…

- Assume that glasses(i) is a Boolean function which is True if the person in position i has glasses
- Asking: if glasses(i) == True is the same as asking if glasses(i)

```
for i in range(num):
        if glasses(i):
            print True
        else:
            print False
```

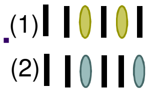
the user will see True and False printed depending on whether person in pos i has glasses or not

## Clarifications:

- The same way that there are variables holding a number or a string, variables may hold a Boolean value. The name of the variable is the programmer's choice
- Examples:

```
var = True
var2 = (x > 5)
found = False
```

## Trace by hand…



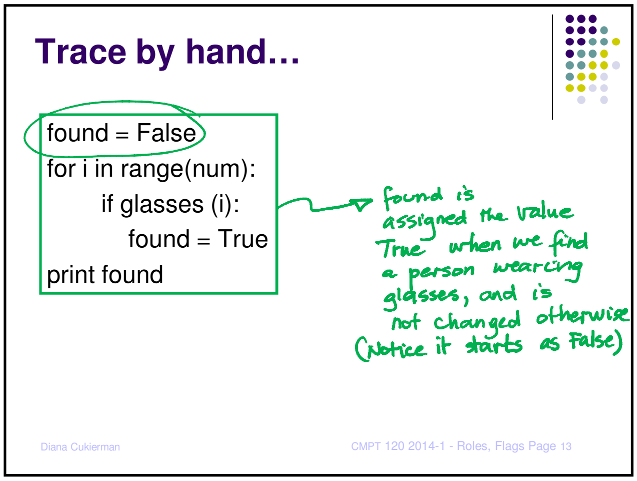
```
found = False
for i in range(num):
        if glasses(i):
            found = True
        else:
            found = False
print found
```

(1) → False ✓

(2) → True ✓

Since the value of found is changed in every position, this code causes that found is True or False depending on the LAST position only

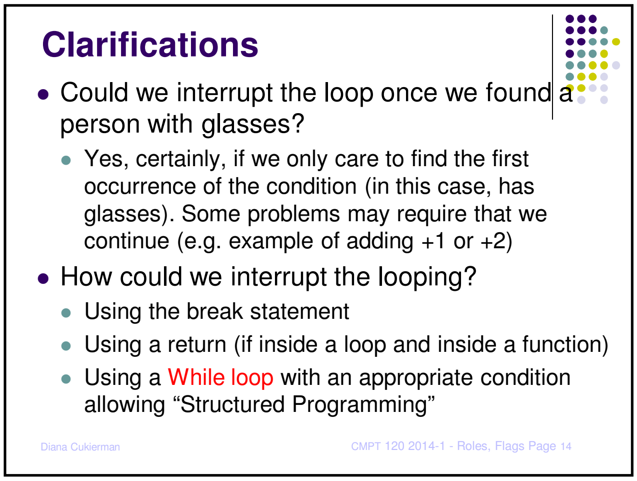
Boolean variable
In this statement it is assigned the value True



## Trace by hand…

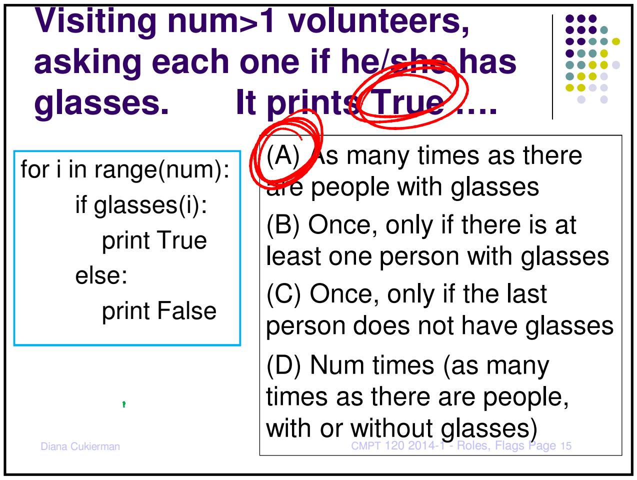
```
found = False
for i in range(num):
        if glasses (i):
            found = True
print found
```

found is assigned the value True when we find a person wearing glasses, and is not changed otherwise (Notice it starts as False)



## Clarifications

- Could we interrupt the loop once we found a person with glasses?
  - Yes, certainly, if we only care to find the first occurrence of the condition (in this case, has glasses). Some problems may require that we continue (e.g. example of adding +1 or +2)
- How could we interrupt the looping?
  - Using the break statement
  - Using a return (if inside a loop and inside a function)
  - Using a While loop with an appropriate condition allowing “Structured Programming”



## Visiting num>1 volunteers, asking each one if he/she has glasses. It prints True ….

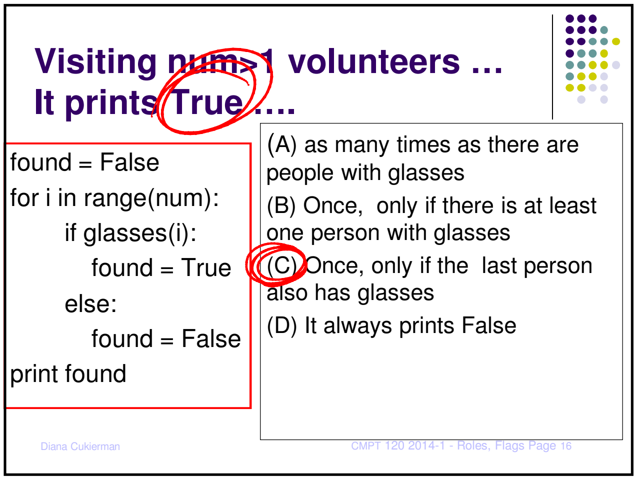
```
for i in range(num):
        if glasses(i):
            print True
        else:
            print False
```

[✓ (A)] As many times as there are people with glasses

(B) Once, only if there is at least one person with glasses

(C) Once, only if the last person does not have glasses

(D) Num times (as many times as there are people, with or without glasses)



## Visiting num>1 volunteers … It prints True ….

```
found = False
for i in range(num):
        if glasses(i):
            found = True
        else:
            found = False
print found
```

(A) as many times as there are people with glasses

(B) Once, only if there is at least one person with glasses

[✓ (C)] Once, only if the last person also has glasses

(D) It always prints False

## Visiting num>1 volunteers … It prints False ….

```
found = False
for i in range(num):
        if glasses (i):
            found = True
print found
```

(A) as many times as there are people without glasses

(B) Once, only if there is at least one person without glasses

(C) Once, only if the last person does not have glasses

[✓ (D)] Once, only if no-one has glasses

(E) It always prints False

## More examples, clarifications…

- The next example was only seen in class as a demo with volunteers.

Note:

- This is not the only way to solve this problem,

and

- There are more possible ways and typical examples or patterns when flag variables may be used

## For loop with a flag and also an accumulator

- This example asked that different actions are done (adding + 1 or +2) depending on …
- being in a position BEFORE the condition is detected (before finding a person with glasses) (in the example, adding +1)

OR

- being in a position AFTER the condition is detected (after finding the person with glasses) (in the example, adding +2).

## Previous example applied to strings:

Define a function that receives a string *st* and a character *ch* and returns a number calculated:

- Search to see if *ch* is in the string *st*
- FOR EVERY CHARACTER IN *st*
- If before finding *ch,* add 1
- After finding *ch* (it may or not happen) and including when ch was found, add 2

difActionFlag("abcXdefg", "X") should return

1+1+1 + 2+2+2+2+2 = 13