THE UNIVERSITY OF WESTERN AUSTRALIA

Lecture 17
Nested Loops

Objectives

- To understand the use of nested loops.
- To be able to design and implement solutions to problems involving loop patterns.
  - *Post test loops*
  - *Loop and a half*
- To understand the use of `break` and `continue` statements.

Revision: Indefinite Loops

- while <condition>:
   <body>

- <condition> is a Boolean expression, just like in `if` statements. <body> is a sequence of one or more statements.

- Semantically, the body of the loop executes repeatedly as long as the condition remains true.

- When the condition is false, the loop terminates.

Revision: Interactive Loops

- A good use of the indefinite loop is to write interactive loops that allow a user to repeat certain portions of a program on demand.
- Example: At each iteration of the loop, ask the user if there is more data to process.

Revision: Sentinel Loops

- A sentinel loop continues to process data until reaching a special value that signals the end.
- This special value is called the sentinel.
- The sentinel must be distinguishable from the data since it is not processed as part of the data.

Revision: Sentinel Loops

```
#  A program to average a set of numbers
#  Using empty string as loop sentinel

def main():
  sum = 0.0
  count = 0
  xStr = input("Enter a number (<Enter> to quit) >> ")
  while xStr != "":
    sum += float(xStr)
    count += 1
    xStr = input("Enter a number (<Enter> to quit) >> ")
  print("\nThe average of the numbers is", sum / count)
```

Revision: Sentinel Loops

```
Enter a number (<Enter> to quit) >> 34
Enter a number (<Enter> to quit) >> 23
Enter a number (<Enter> to quit) >> 0
Enter a number (<Enter> to quit) >> -25
Enter a number (<Enter> to quit) >> -34.4
Enter a number (<Enter> to quit) >> 22.7
Enter a number (<Enter> to quit) >>

The average of the numbers is 3.38333333333
```

Nested Loops

- In the same way that you have an if statement within an if statement, you can have loops within loops
- For example, rather than having 1 number per input line, have multiple, comma-separated numbers per line

Nested Loops

```
# average7.py
#     Computes the average of numbers listed in a file.
#     Works with multiple numbers on a line.

def main():
    fileName = input("What file are the numbers in? ")
    infile = open(fileName,'r')
    sum = 0.0
    count = 0
    for line in infile:
        # update sum and count for values in line
        for xStr in line.split(","):
            sum += float(xStr)
            count += 1
    print("\nThe average of the numbers is", sum / count)
```

Nested Loops

- The loop that processes the numbers in each line is indented inside of the file processing loop.
- The outer while loop iterates once for each line of the file.
- For each iteration of the outer loop, the inner for loop iterates as many times as there are numbers on the line.
- When the inner loop finishes, the next line of the file is read, and this process begins again.

Nested Loops

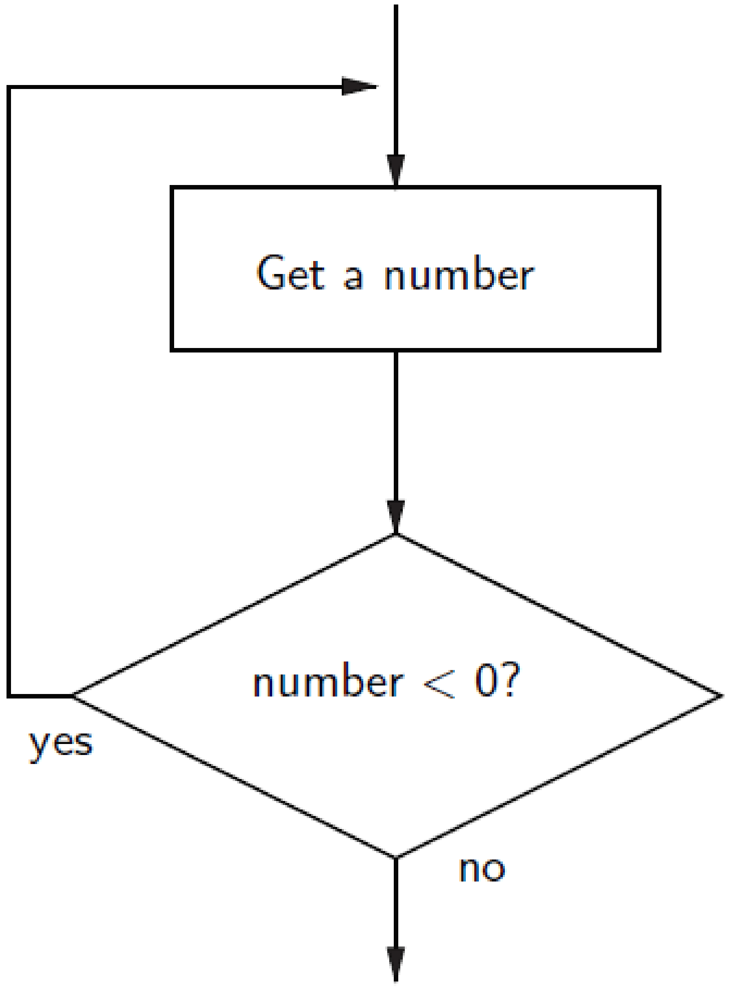
- Designing nested loops –
  - *Design the outer loop without worrying about what goes inside*
  - *Design what goes inside, ignoring the outer loop.*
  - *Put the pieces together, preserving the nesting.*

Other Loop Structures – Post-Test Loop

- Say we want to write a program that is supposed to get a nonnegative number from the user.
- If the user types an incorrect input, the program asks for another value.
- This process continues until a valid value has been entered.
- This process is *input validation*.

Post-Test Loop

```
repeat
    get a number from the user
until number is >= 0
```



Post-Test Loop

- When the condition test comes after the body of the loop it’s called a *post-test loop*.
- A post-test loop always executes the body of the code at least once.
- Python doesn’t have a built-in statement to do this, but we can do it with a slightly modified `while` loop.

Post-Test Loop

```
# A program to average a set of numbers
# Using Post-Test loop which will be execute at least once
# loop will be terminated on negative values
def main():
  sum = 0.0
  count = 0
  xStr = 1
  while xStr >=0:
      xStr = input("Enter a number (<Negative> to quit) >> ")
      if xStr >= 0:
         sum += float(xStr)
         count += 1
  print("\nThe average of the numbers is", sum / count)
```

Post-Test Loop

- Some programmers prefer to simulate a post-test loop by using the Python `break` statement.
- Executing `break` causes Python to immediately exit the enclosing loop.
- `break` is sometimes used to exit what looks like an infinite loop.

Post-Test Loop

- The same algorithm implemented with a `break`:

```
while True:
    xStr = input("Enter a number (<Negative> to quit) >> ")
    if xStr < 0:
        break # Exit loop
```

- A `while` loop continues as long as the expression evaluates to true. Since `True` *always* evaluates to true, it looks like an infinite loop!

Loop and a Half

- Stylistically, some programmers prefer the following approach:

```
while True:
    number = float(input("Enter a positive number: "))
    if number >= 0:
        break # if valid number exit loop
    print("The number you entered was not positive")
```

- Here the loop exit is in the middle of the loop body. This is what we mean by a *loop and a half*.

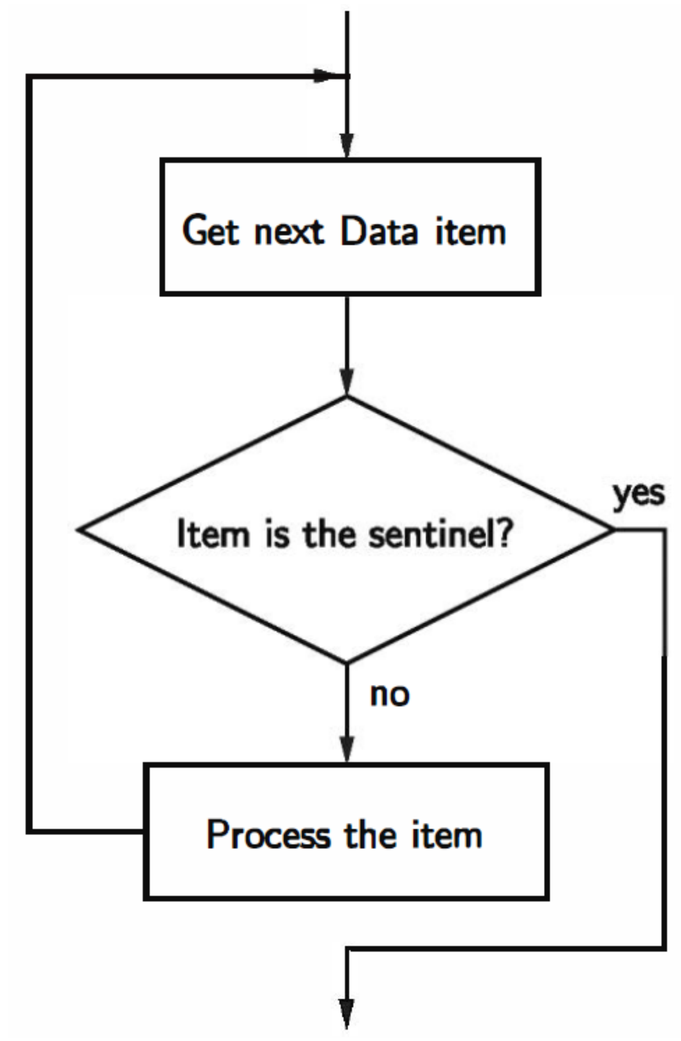
Loop and a Half

- The loop and a half is an elegant way to avoid the priming read in a sentinel loop.

```
while True:
    # get next data item
    # if the item is the sentinel: break
    # process the item
```

- This method is faithful to the idea of the sentinel loop, the sentinel value is not processed!

Loop and a Half



Loop and a Half

- To use or not use `break`. That is the question!
- The use of `break` is mostly a matter of style and taste.
- Avoid using `break` often within loops, because the logic of a loop is hard to follow when there are multiple exits.

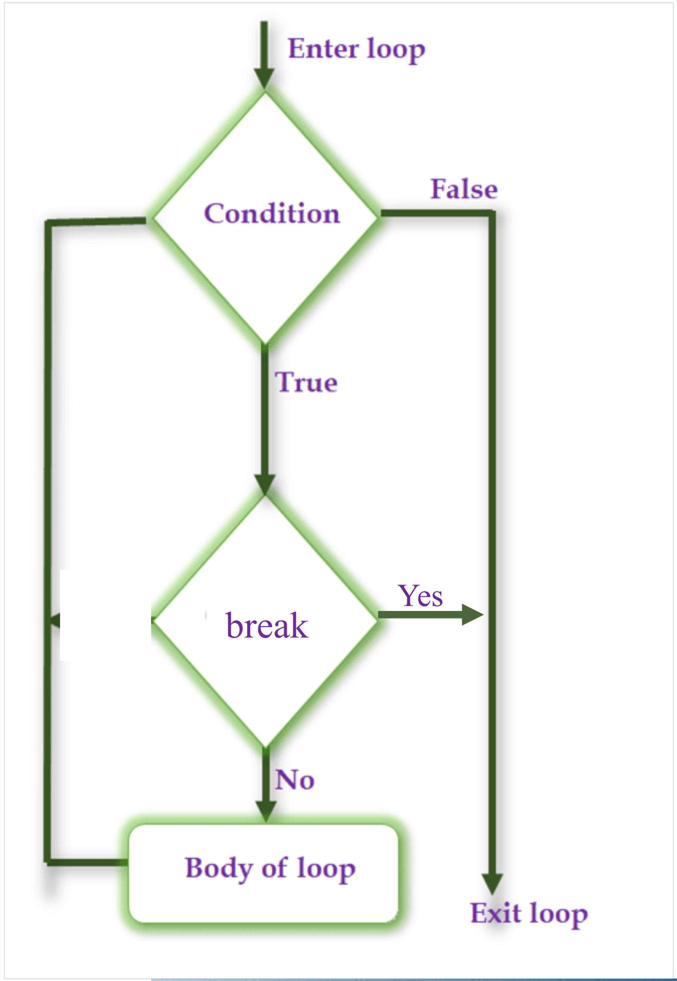
continue statement

- Continue statement returns the control to the beginning of the loop

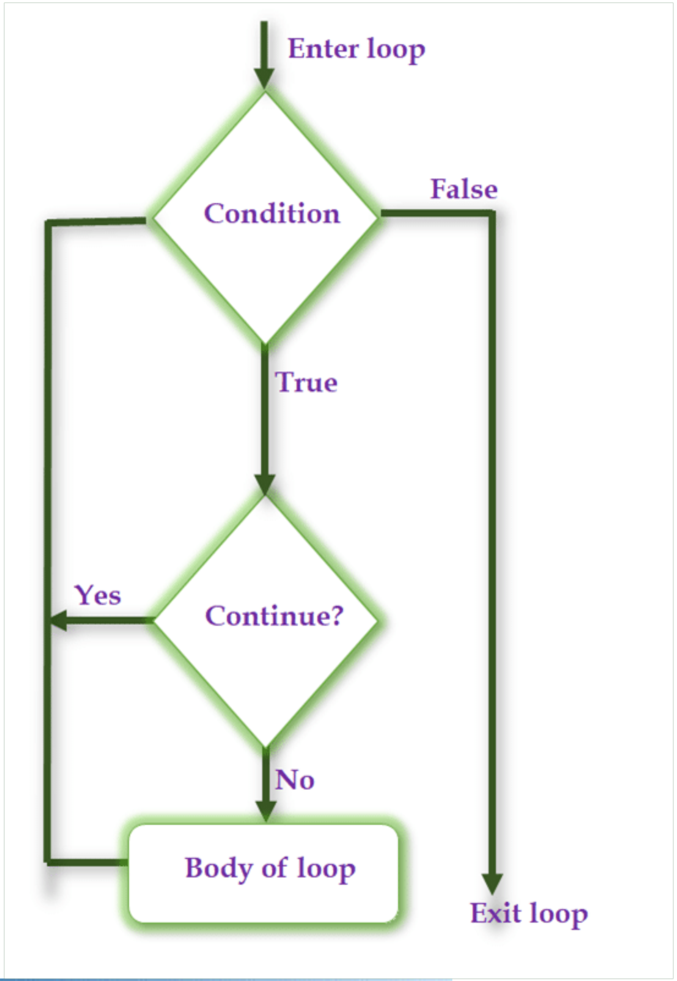
```
# print only even numbers up to 10
for i in range(11):
    if i % 2 == 1:   # % is "modulus" operator
         continue
     print(i)
```

break and continue comparison

break statement



continue statement



Summary

- Nested loops
- Post-Test loops
- `Break` statement
- Loop and a Half
- `Continue` statement