Loops In Python

In this section of notes you will learn how to rerun parts of your program without having to duplicate the code.

The Need For Repetition (Loops)

Writing out a simple counting program (1 – 3).

```
print “1”

print “2”

print “3”
```

The Need For Repetition (2)

Simple program but what if changes need to be made?

- The source code must be re-edited and re-compiled each time that a change is needed.

What if you need the program to count many times?

Basic Structure Of Loops

1) Initialize the control
    a) Control – typically a variable that determines whether or not the loop executes or not.
2) Testing the control against a stopping condition
3) Executing the body of the loop (the part to be repeated)
4) Update the value of the control

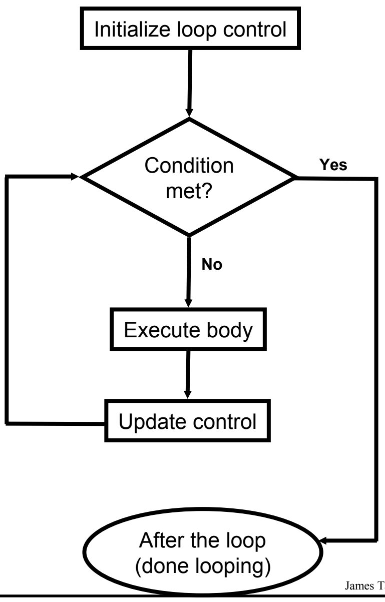
Types Of Loops

1. Pre-test loops
   - Check the stopping condition *before* executing the body of the loop.
   - The loop executes *zero or more* times.

2. Post-test loops
   - Checking the stopping condition *after* executing the body of the loop.
   - The loop executes *one or more* times.

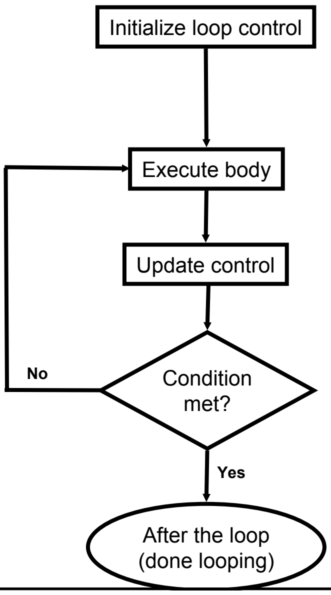
Pre-Test Loops

1. Initialize loop control
2. Check if the stopping condition has been met
   a. If it's been met then the loop ends
   b. If it hasn't been met then proceed to the next step
3. Execute the body of the loop (the part to be repeated)
4. Update the loop control
5. Go to step 2



Post-Test Loops (Not Implemented In Python)

1. Initialize loop control (sometimes not needed because initialization occurs when the control is updated)
2. Execute the body of the loop (the part to be repeated)
3. Update the loop control
4. Check if the stopping condition has been met
   a. If it's been met then the loop ends
   b. If it hasn't been met then return to step 2.



Pre-Test Loops In Python

1. While
2. For

Characteristics:

1. The stopping condition is checked *before* the body executes.
2. These types of loops execute zero or more times.

Post-Loops In Python

None: this type of looping construct has not been implemented with this language.

Characteristics:

- The stopping condition is checked *after* the body executes.
- These types of loops execute one or more times.

The While Loop

This type of loop can be used if it’s not in advance how many times that the loop will repeat (most powerful type of loop).

Format:

```
(Simple)
while (Boolean expression):
    body

(Compound)
while (Boolean expression) Boolean operator (Boolean expression):
    body
```

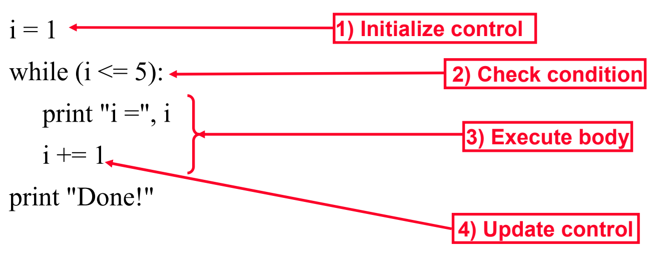
The While Loop (2)

Example: (The full program can be found in Unix under /home/courses/217/examples/loops/while1.p)

```
i = 1
while (i <= 5):
    print "i =", i
    i += 1
print "Done!"
```

The While Loop (2)

Example: (The full program can be found in Unix under /home/courses/217/examples/loops/while1.p)



```
i = 1
while (i <= 5):
    print "i =", i
    i += 1
print "Done!"
```

Tracing The While Loop

Execution	Variable
>python while1.py	i

The For Loop

Typically used when it is known in advance how many times that the loop will execute (counting loops).

Syntax:

```
for <name of loop control> in <something that can be iterated>:
    body
```

Example: (The full program can be found in Unix under /home/courses/217/examples/loops/for1.p)

```
for i in range (1, 6, 1):

    total = total + i

    print "i = ", i, "\t total = ", total

print "Done!"
```

The For Loop

Typically used when it is known in advance how many times that the loop will execute (counting loops).

Syntax:

for <*name of loop control*> in <*something that can be iterated*>:
    body

Example: (The full program can be found in Unix under /home/courses/217/examples/loops/for1.p)

1) Initialize control

2) Check condition

```
for i in range (1, 6, 1):

    total = total + i

    print "i = ", i, "\t total = ", total

print "Done!"
```

4) Update control

3) Execute body

Tracing The First For Loop Example

Execution	**Variables**	
>python for1.py	i	total

Counting Down With A For Loop

Example: (The full program can be found in Unix under /home/courses/217/examples/loops/for2.p)

```
for i in range (5, 0, -1):
    total = total + i
    print "i = ", i, "\t total = ", total
print "Done!"
```

Tracing The First For Loop Example

Execution

```
>python for1.py
```

Variables

i	total

Erroneous For Loop

The logic of the loop is such that the end condition has been reached with the start condition.

Example:

```
for i in range (5, 0, 1):
    total = total + i
    print "i = ", i, "\t total = ", total
print "Done!"
```

Loop Increments Need Not Be Limited To One

While

```
i = 0
while (i <= 100):
    print "i =", i
    i = i + 5
print "Done!"
```

For

```
for i in range (0, 105, 5):
    print "i =", i
print "Done!"
```

Sentinel Controlled Loops

The stopping condition for the loop occurs when the ‘sentinel’ value is reached. The complete program can be found in UNIX under: /home/courses/217/examples/loops/sumSeries.py

```
total = 0
temp = 0
while (temp >= 0):
    temp = input ("Enter a positive integer (negative to end series):")
    if (temp >= 0):
        total = total + temp

print "Sum total of the series:", total
```

Solving A Problem Using Loops

Problem: Write a program that will execute a game:

- The program will randomly generate a number between one and ten.
- The player will be prompted to enter their guess.
- The program will continue the game until the player indicates that they no longer want to continue.

The full program can be found in UNIX under: /home/courses/217/examples/loops/guessingGame.py

Guessing Game

```
guess = 0
answer = 0
choice = "Y"
while choice not in ("q", "Q"):
    answer = random.randrange (10) + 1
    guess = input ("Enter your guess: ")
    if (guess == answer):
        print "You guessed correctly!"
    else:
        print "You guessed incorrectly"
    print "Number was", answer, ", your guess was", guess
    print "Play again?  Enter 'q' to quit, anything else to play again"
    print "Choice:",
    choice = raw_input()
    print ""
print "Exiting game"
```

Recap: What Looping Constructs Are Available In Python/When To Use Them

Construct	When To Use
Pre-test loops	You want the stopping condition to be checked before the loop body is executed (typically used when you want a loop to execute zero or more times).
• While	• The most powerful looping construct: you can write a 'while-do' loop to mimic the behavior of any other type of loop. In general it should be used when you want a pre-test loop which can be used for most any arbitrary stopping condition e.g., execute the loop as long as the user doesn't enter a negative number.
• For	• A 'counting loop': You want a simple loop to count up or down a certain number of times.
Post-test: None in Python	You want to execute the body of the loop before checking the stopping condition (typically used to ensure that the body of the loop will execute at least once). The logic can be simulated in Python however.

Infinite Loops

Infinite loops never end (the stopping condition is never met).

They can be caused by logical errors:

- The loop control is never updated (Example 1 – below).
- The updating of the loop control never brings it closer to the stopping condition (Example 2 – next slide).

Example 1 (The full version can be found in UNIX under /home/courses/217/examples/loops/infinite1.py)

```
i = 1
while (i <=10):
    print "i = ", i
i = i + 1
```

To stop a program with an infinite loop in Unix simultaneously press the <ctrl> and the <c> keys

Infinite Loops (2)

Example 2 (The full version can be found in Unix under /home/courses/217/examples/loops/infinite2.py)

```
while (i > 0):

    print "i = ",  i

    i = i + 1
```

To stop a program with an infinite loop in Unix simultaneously press the <ctrl> and the <c> keys

Nested Loops

One loop executes inside of another loop(s).

Example structure:

```
Outer loop (runs n times)
    Inner loop (runs m times)
        Body of inner loop (runs n x m times)
```

Example program (the full program can be found in UNIX under: /home/courses/217/examples/loops/nested.py)

```
for i in range (1, 3, 1):
    for j in range (1, 4, 1):
        print "i = ", i, " j = ", j
print "Done!"
```

Testing Loops

Make sure that the loop executes the proper number of times.

Test conditions:

1) Loop does not run
2) Loop runs exactly once
3) Loop runs exactly ‘n’ times

Testing Loops: An Example

```
sum = 0
i = 1
last = 0

last = input ("Enter the last number in the sequence to sum : ")
while (i <= last):
    sum = sum + i
    print "i = ", i
    i = i + 1

print "sum =", sum
```

You Should Now Know

When and why are loops used in computer programs

What is the difference between pre-test loops and post-test loops

How to trace the execution of pre-test loops

How to properly write the code for a loop in a program

What are nested loops and how do you trace their execution

How to test the execution of loop