Revised: 2/17/2018

Repetitive Logic

Michael C. Hackett

Computer Science Department

Community College of Philadelphia

Lecture Topics

- While Loops
  - Sentinel Values
  - Else Clauses
- Input Validation
- For Loops
  - Else Clauses
- Branching Statements
- Nested For Loops
- Infinite Loops
- Loop and a half

Colors/Fonts

- Variable Names – Brown
- Literals – Blue
- Keywords – Orange
- Operators/Punctuation – Black
- Functions – Purple
- Comments – Gray
- Modules – Pink

Source Code – **Consolas**

Output – Courier New

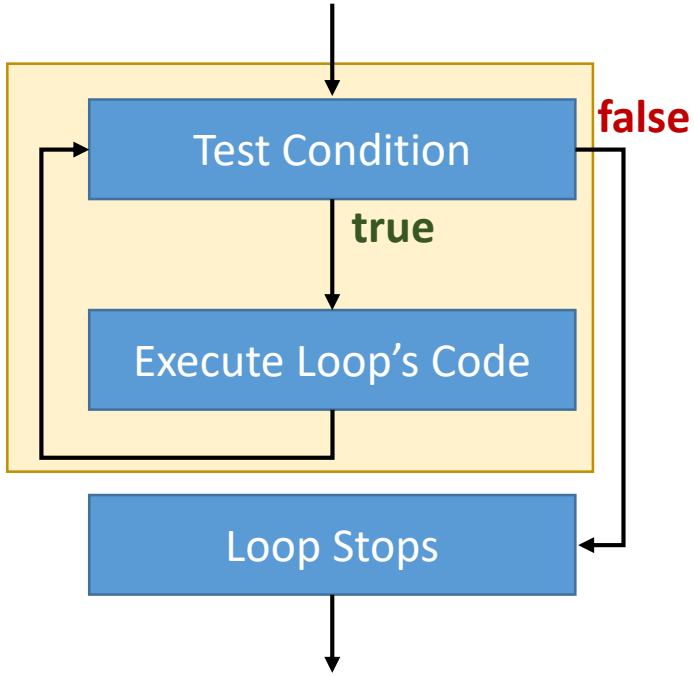
Loops

- A ***loop*** is a programming structure that allows code to be repeatedly executed, usually as long as some condition (Boolean expression) evaluates to true.
  - Each repetition of the loop's code is called an ***iteration***.

- Programming languages have a few types of loops.
  - Pre-test and Post-test Loops
  - Sentinel-Controlled and Count-Controlled.

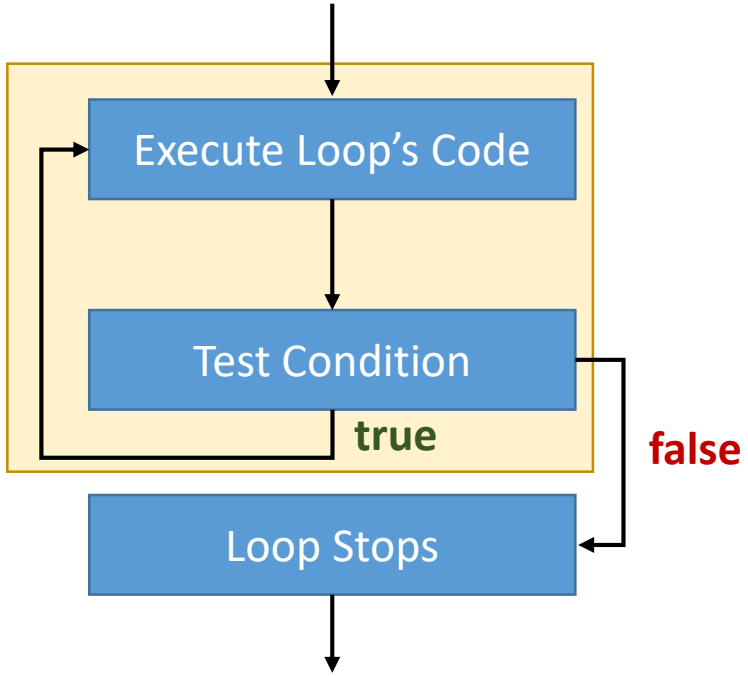
Types of Loops

- ***Pre-test loops*** test the condition *before* starting each iteration.



Types of Loops

- ***Post-test loops*** test the condition *after* completing each iteration.
  - Python does not have a post-test loop.



Sentinel-Controlled Loops

- A loop is ***sentinel-controlled*** when it uses a certain value or values to indicate that the loop should stop repeating.
  - This value is referred to as a ***sentinel value*** or ***flag value***.

- A “while loop” is a sentinel-controlled loop.

While Loop

- A ***while loop*** repeats as long as its Boolean expression is true.
  - *"While this condition is true, keep repeating…"*
  - When the condition is false, the loop stops.
- The syntax for a Python while loop is shown below.

```
while Boolean Expression:
    #code that will be
    #executed as long as the
    #Boolean Expression is true
```

Indent one tab

While Loop

```
i = 3
while i < 8 :
  print("Number:", i)
  i += 1
```

```
Number: 3
Number: 4
Number: 5
Number: 6
Number: 7
```

While Loop

- A while loop may not iterate at all, if the condition is false from the start.

```
i = 3
while i <= 2 :
  print("Number:", i)
  i += 1
```

```
(No output)
```

While Loop

```
input = input("Enter word: ")

while input.lower() != "exit" :
  #Print the input in uppercase
  print("Uppercase:", input.upper())
  #Prompt for input again
  input = input("Enter word: ")

print("Goodbye!")
```

Sentinel Value

```
Enter word: cat
Uppercase: CAT
Enter word: dog
Uppercase: DOG
Enter word: llama
Uppercase: LLAMA
Enter word: exit
Goodbye!
```

While Loop

- In the while loop below, the sentinel value that will cause the loop to terminate is when *any number* less than one is entered.

```
done = False #Controls the loop

while not done :
  number_to_square = int(input("Enter a number: "))
  if number_to_square < 1 :
    done = True
  else :
    print("Your number squared is: ",
          number_to_square ** 2)

print("Goodbye!")
```

```
Enter a number: 3
Your number squared is: 9
Enter a number: 4
Your number squared is: 16
Enter a number: 0
Goodbye!
```

Else Clauses and While Loops

- A while loop can have an else clause.
  - The else clause's code will be executed when the while loops condition evaluates to false.

```
while condition :
    #code that will be
    #executed
else :
    #code that will be executed
    #when the condition is false
```

While Loop

```
done = False #Controls the loop

while not done :
  number_to_square = int(input("Enter a number: "))
  if number_to_square < 1 :
    done = True
  else :
    print("Your number squared is:", number_to_square ** 2)
else :
  print("Goodbye!")
```

```
Enter a number: 3
Your number squared is: 9
Enter a number: 4
Your number squared is: 16
Enter a number: 0
Goodbye!
```

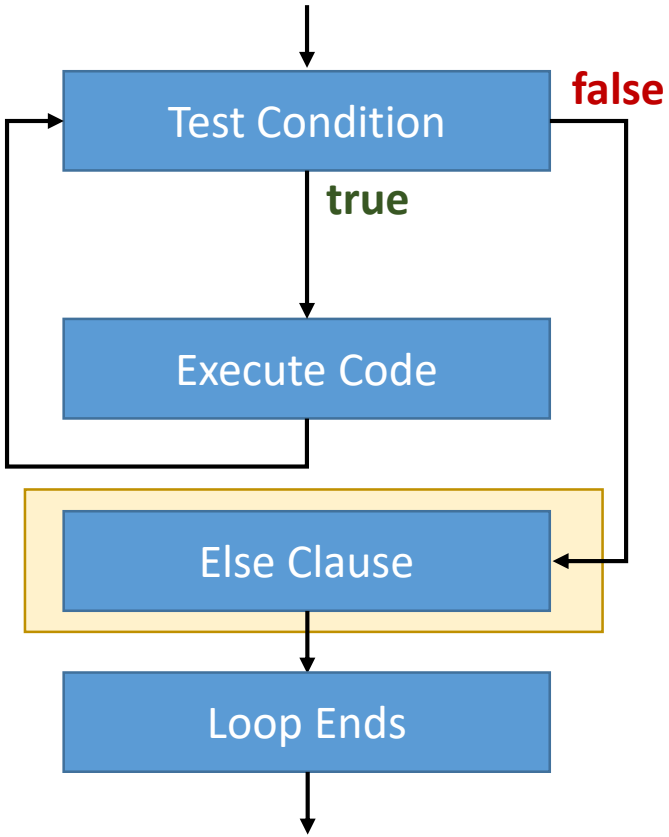
While Loop

```
done = True #Controls the loop

while not done :
  number_to_square = int(input("Enter a number: "))
  if number_to_square < 1 :
    done = True
  else :
    print("Your number squared is:", number_to_square ** 2)
else :
  print("Goodbye!")
```

```
Goodbye!
```

While Loop (Flow Chart)



Input Validation

- While loops are useful when validating a user's input.

- It can continue to prompt a user for input in the event the user enters invalid values.

```
Enter a positive number: -4
Error. Entered number is not positive.
Re-enter a positive number: 0
Error. Entered number is not positive.
Re-enter a positive number: 5
Thank you!
```

Input Validation

```
input = int(input("Enter a positive number: "))

while input < 1 :
  #Print error message
  print("Error. Entered number is not positive.")
  #Prompt for input again
  input = int(input("Re-enter a positive number: "))

print("Thank you!")
```

```
Enter a positive number: -4
Error. Entered number is not positive.
Re-enter a positive number: 0
Error. Entered number is not positive.
Re-enter a positive number: 5
Thank you!
```

Input Validation

```
input = int(input("Enter a number between 1 and 10: "))

while input < 1 or input > 10 :
  #Print error message
  print("Error. Entered number is outside of specified range.")
  #Prompt for input again
  input = int(input("Re-enter a number between 1 and 10: "))

print("Thank you!")
```

```
Enter a number between 1 and 10: -4
Error. Entered number is outside of specified range.
Re-enter a number between 1 and 10 : 20
Error. Entered number is outside of specified range.
Re-enter a number between 1 and 10 : 5
Thank you!
```

Count-Controlled Loops

- A loop is ***count-controlled*** when it iterates through a range of values.
- Each iteration, the loop assigns the next value in the range to a variable that can be used in the loop's code.
- The loop stops when it has exhausted the list of values in the range.
- A "for loop" is a count-controlled loop.

For Loops

- A ***for loop*** iterates over a range of values.
  - Often implemented using Python's built-in range function.

```
for variable in range :
    #code that will be
    #executed for every value
    #in the range
```

Indent one tab

Range Function

- Accepts one, two or three arguments.

```
range(5)
```
→ Values 0, 1, 2, 3, 4

```
range(2, 8)
```
→ Values 2, 3, 4, 5, 6, 7

```
range(3, 10, 2)
```
→ Values 3, 5, 7, 9

```
range(11, 7, -1)
```
→ Values 11, 10, 9, 8

For Loop

- A for loop that simulates making 5 laps around a race track.

```
for counter in range(1, 6) :
  print("Lap #", counter, sep="")

print("Finished!")
```

```
Lap #1
Lap #2
Lap #3
Lap #4
Lap #5
Finished!
```

For Loop

Start (inclusive) Stop (exclusive)

```
for i in range(3, 8) :
  print("Number:", i)
```

```
Number: 3
Number: 4
Number: 5
Number: 6
Number: 7
```

For Loop

Start (inclusive)

Stop (exclusive)

Increment/Decrement

```
for i in range(2, 9, 2) :
  print("Number:", i)
```

```
Number: 2
Number: 4
Number: 6
Number: 8
```

For Loop

Start (inclusive)

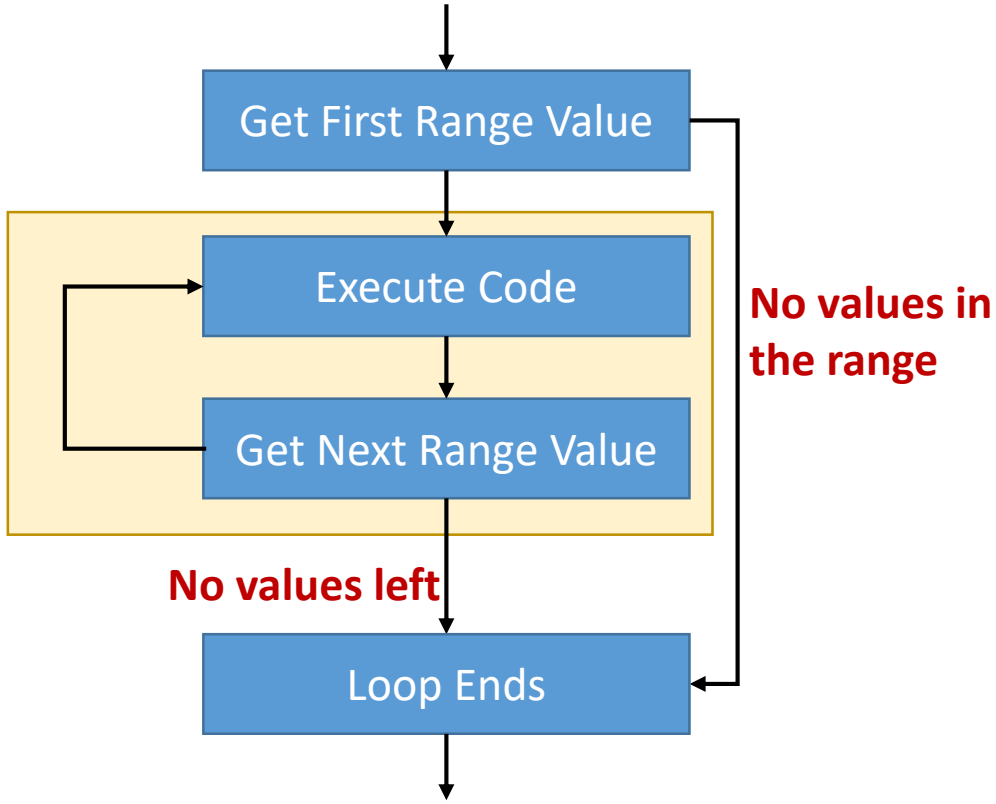
Stop (exclusive)

Increment/Decrement

```
for i in range(5, 0, -1) :
  print("Number:", i)
```

```
Number: 5
Number: 4
Number: 3
Number: 2
Number: 1
```

For Loop (Flow Chart)



Else Clauses and For Loops

- A for loop can have an else clause.
  - The else clause's code will be executed when the for loop has exhausted its ranges.

```
for variable in range :
    #code that will be
    #executed for every value
    #in the range
else :
    #code that will be executed
    #when the range is exhausted
```

For Loop

```
for i in range(0, 4) :
  print("Number:", i)
else :
  print("All Done!")
```

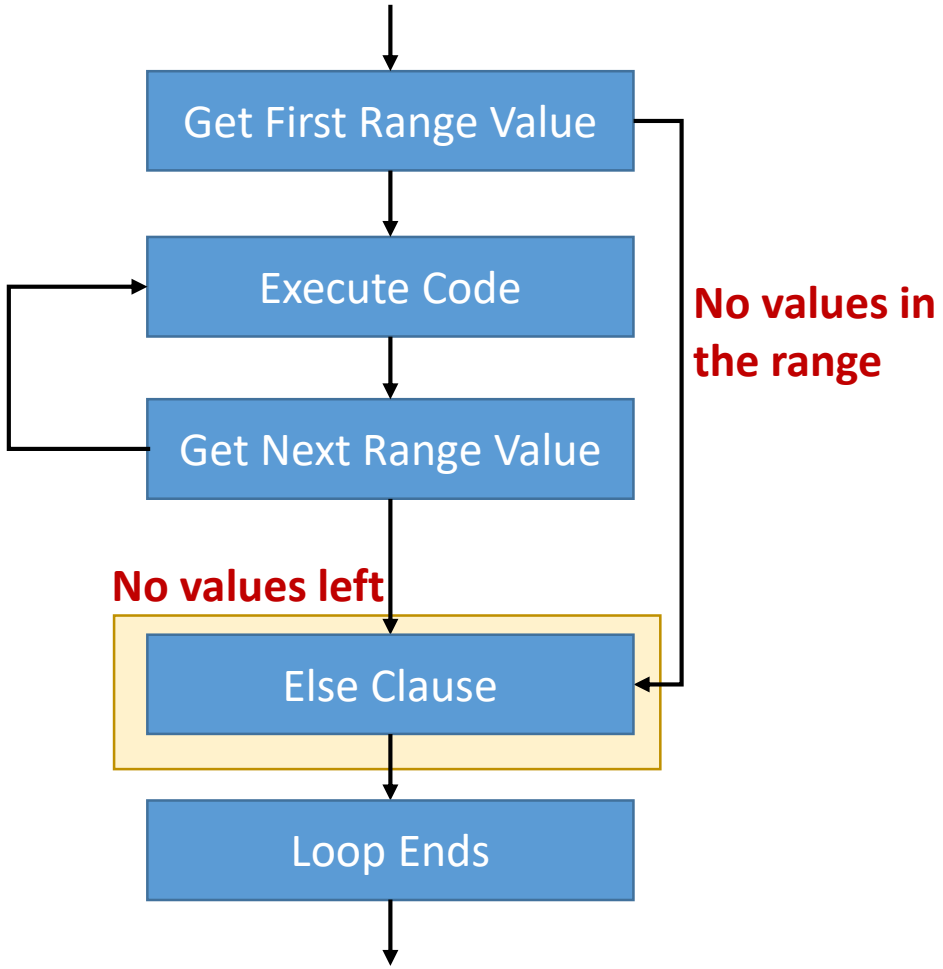
```
Number: 0
Number: 1
Number: 2
Number: 3
All Done!
```

For Loop

```
for i in range(4, 4) :
  print("Number:", i)
else :
  print("All Done!")
```

```
All Done!
```

For Loop (Updated Flow Chart)



For vs While Loops

- While loop
  - Use when you need to iterate as long as a condition is and remains true.
  - Sentinel-controlled.

- For loop
  - Use when you need to iterate over a range of values.
  - Count-controlled.

Branching Statements

- There are two branching statements that allow us to either:
  - Immediately exit a loop.
  - Immediately begin the next iteration.

`break`

- Once encountered, the loop will immediately stop where it is. Any code outside/after of the loop will begin to be executed.

`continue`

- Once encountered, the loop will immediately stop where it is and begin the next iteration.

break statement

```
for i in range(1, 10) :
  if i > 5 :
    break
  print("Number:", i)

print("All done!")
```

```
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
All done!
```

- This loop normally would have printed “Number: 1” through “Number: 9”
- However, once the value of i is greater than 5, the break statement will be encountered.
- The loop will exit immediately and resume the code outside of the loop.

break statement

```
for i in range(1, 10) :
  if i > 5 :
    break
  print("Number:", i)
else :
  print("All done!")
```

```
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
```

- The break statement will not only stop the loop but skip the loop's else clause, if it is present

continue statement – for loop

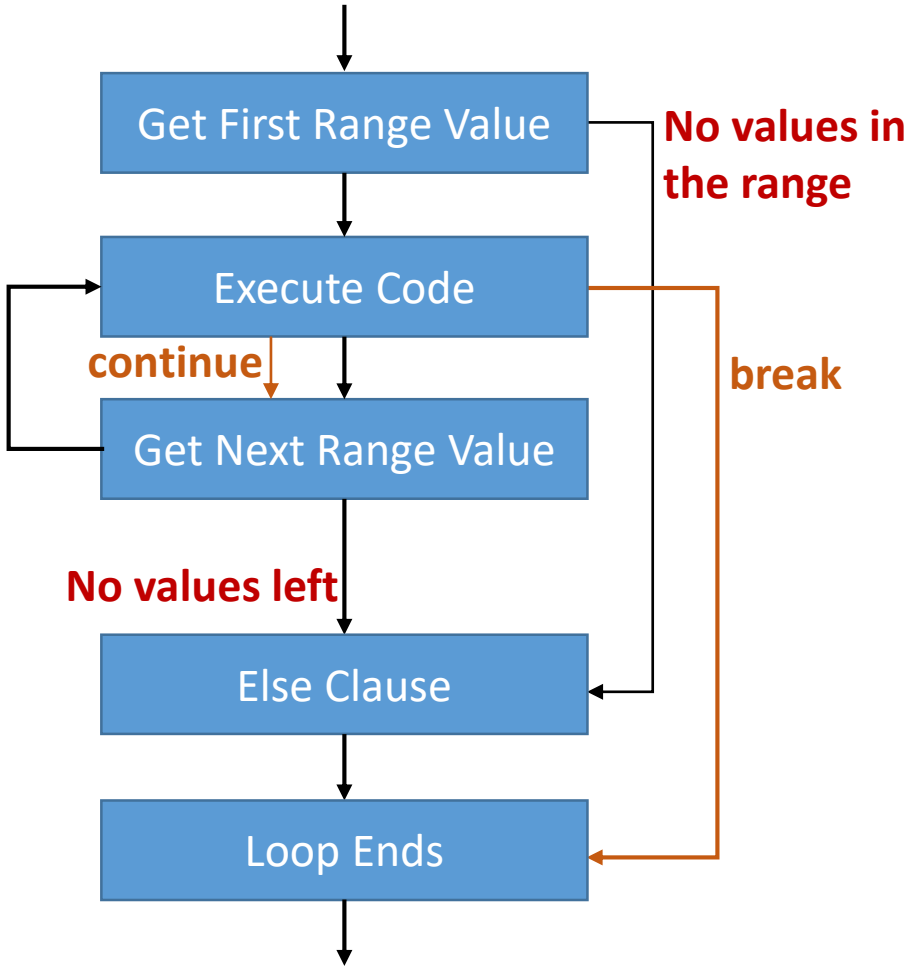
```
for i in range(1, 11) :
  if i % 2 == 1 :
    continue
  print("Number:", i)

print("All done!")
```

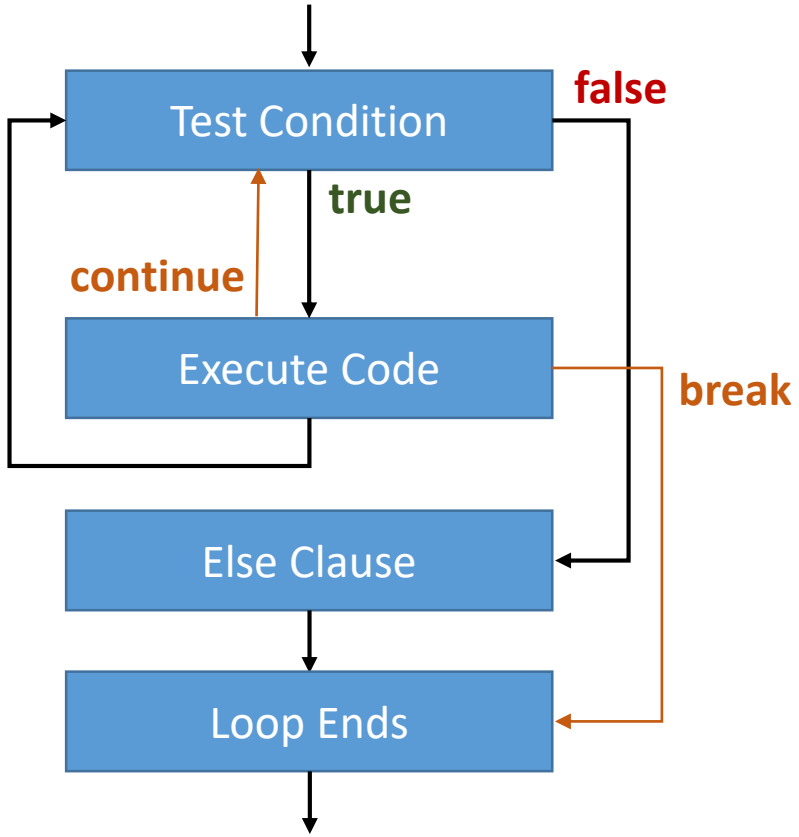
```
Number: 2
Number: 4
Number: 6
Number: 8
Number: 10
All done!
```

- If i's value is odd, the continue statement will be encountered.
- Instead of finishing the iteration and printing out the number, the loop stops there and begins the next iteration.

For Loop (Complete Flow Chart)



While Loop (Complete Flow Chart)



Nested For Loops

- A nested loop is a loop within a loop.
- For every iteration of the outer loop, the inner loop will be iterated to completion.

```
for i in range(1, 4) :
  print("Number:", i)
  for j in range(1, 3) :
    print("Number:", j)
```

Be sure to use different names for your counters. Any variables declared in outer loops will be accessible by inner loops, including the counter.

Nested For Loops

```
for i in range(1, 4) :
  print("Outer Number:", i)
  for j in range(1, 3) :
    print("Inner Number:", j)
```

```
Outer Number: 1
Inner Number: 1
Inner Number: 2
Outer Number: 2
Inner Number: 1
Inner Number: 2
Outer Number: 3
Inner Number: 1
Inner Number: 2
```

Inner Loop

Inner Loop

Inner Loop

Outer Iteration 1

Outer Iteration 2

Outer Iteration 3

Infinite While Loops

- An infinite loop is a loop that does not stop or exit.
- In many cases, an infinite loop is the result of poor programming.

```
done = False
my_number = 0
while not done :
  my_number += 1
  print("Number:", my_number)
```

```
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
…
```

Infinite While Loops

- Sometimes, infinite loops can be useful.
  - For example, perpetually getting user input until they enter a command to exit/stop.

- However, when we intentionally create an infinite loop, we will want to provide a way for the loop to exit.
  - Use a break statement to stop the loop.

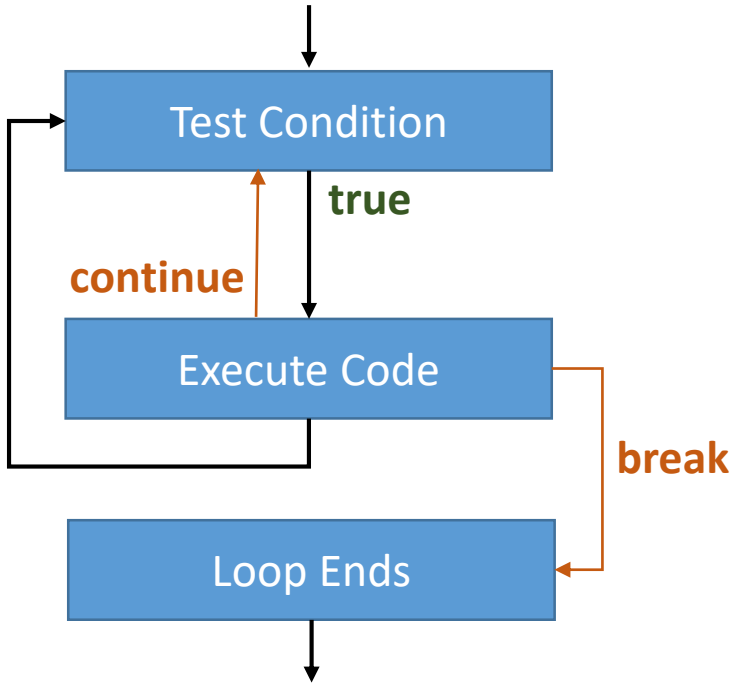
Infinite While Loops

```
while True :
  number_to_square = int(input("Enter a number (0 to exit): "))
  if number_to_square == 0 :
    break
  else :
    print("Your number squared is:", number_to_square ** 2)

print("Goodbye!")
```

```
Enter a number (0 to exit): 3
Your number squared is: 9
Enter a number (0 to exit): 4
Your number squared is: 16
Enter a number (0 to exit): 0
Goodbye!
```

Infinite While Loop (Flow Chart)



Loop and a half

- Python does not have a “do-while loop”, unlike many other programming languages.

- A do-while loop is similar to a while loop, but the condition is tested at the end of an iteration.
  - It is a *post-test* loop.
  - <u>It guarantees the loop will always iterate at least once</u>.

- We can use a while loop to simulate the behavior of a do-while loop.
  - In Python, this is called a ***loop and a half***.

Loop and a half

- A loop and a half is created using an infinite while loop.
- The last statement in the body of the loop is an if statement
  - This if statement's condition determines when to exit the loop.

```
while True :
  #code that will be
  #executed
  if condition :
    break
```

Loop and a half

- This loop and a half verifies that the user's input was non-negative.

```
while True :
  store_sales = int(input("Enter total sales for the store: "))
  if store_sales >= 0 :
    break
print("Thank you.")
```

```
Enter the total sales for the store: -100
Enter the total sales for the store: -5
Enter the total sales for the store: 2500
Thank you.
```

Loop and a half (Flow Chart)