Revised: 12/22/2020

Repetition

Michael C. Hackett

Assistant Professor, Computer Science

Community College of Philadelphia

Lecture Topics

- Repetitive Structures
  - Count-Controlled Loops
  - Sentinel-Controlled Loops
- Input Validation
- Branching Statements
- Infinite Loops
- Nested Loops

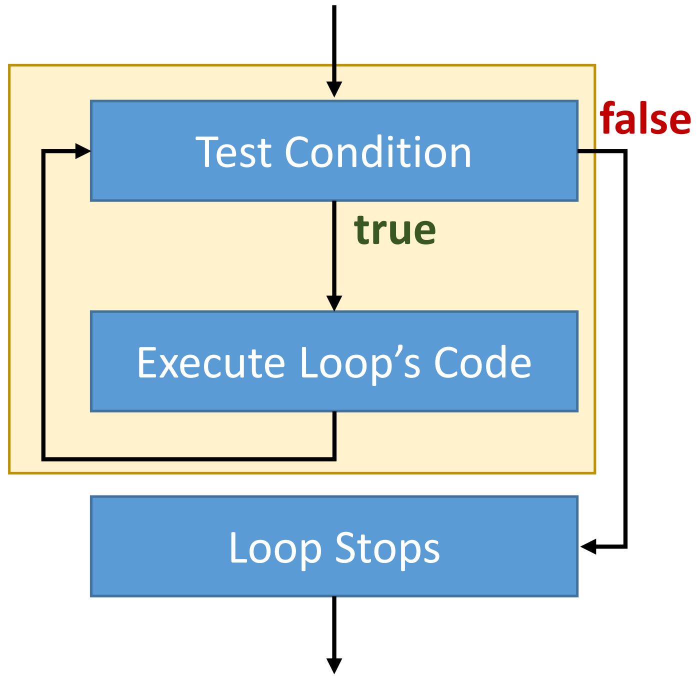
Loops

- 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.

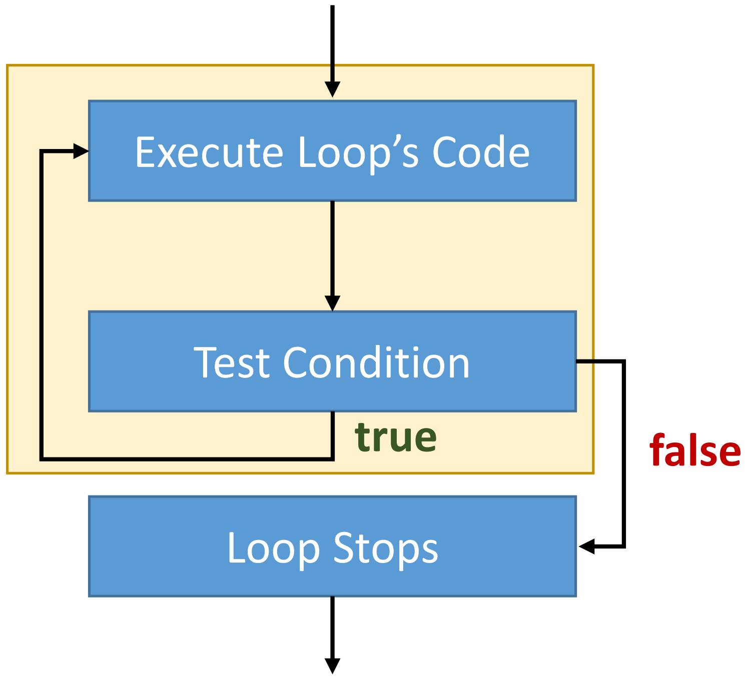
Types of Loops

- ***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.



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.

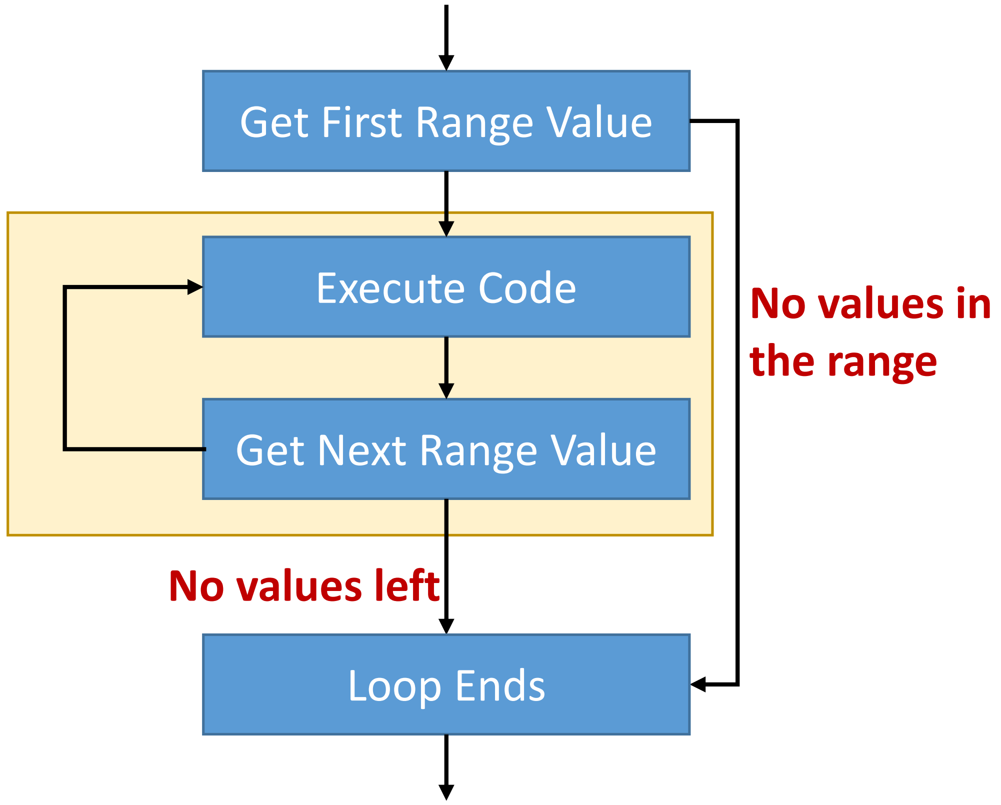
For Loops

- 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
```

$\underbrace{\quad}_{\text{Indent one tab}}$

For Loop (Flow Chart)



Range Function

- Accepts one, two or three arguments.

`range(5)` $\longrightarrow$ Values 0, 1, 2, 3, 4

`range(2, 8)` $\longrightarrow$ Values 2, 3, 4, 5, 6, 7

`range(3, 10, 2)` $\longrightarrow$ Values 3, 5, 7, 9

`range(11, 7, -1)` $\longrightarrow$ Values 11, 10, 9, 8

For Loop

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

Start (inclusive) Stop (exclusive)

```
for counter in range(1, 6) :
  print("Lap #" + str(counter))

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: " + str(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 Loops

- Python's for loop is referred to as an ***enhanced for loop*** in other languages.
    - Also called an *iterator-based*, *for-in*, or *for-each* loop.
    - It is capable of iterating through a list of items, not just numbers.

- Other languages have ***traditional*** ("C-style") ***for loops***.
    - Introduced in the C programming language.
    - Only iterates over a range of numbers.
    - Python does not have C-style for loops.
    - Java has both C-style for loops and enhanced for loops.
        - (We'll see how to use both when we move on to Java)

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.

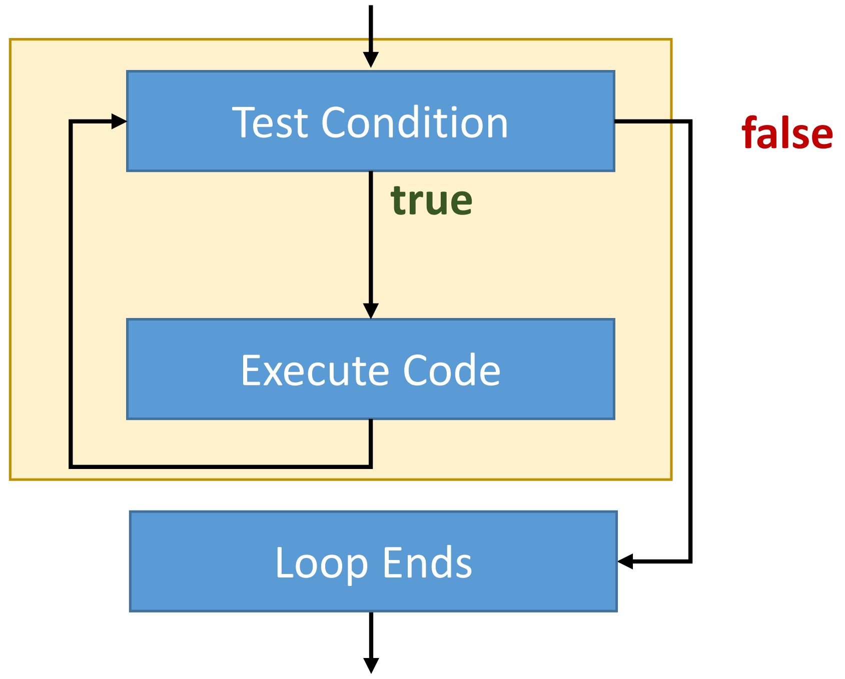
While Loop

- 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 (Flow Chart)



While Loop

```python
i = 3

while i < 8 :
  print("Number: " + str(i))
  i += 1

print("Program Complete.")
```

Sentinel Value

```
Number: 3
Number: 4
Number: 5
Number: 6
Number: 7
Program Complete.
```

While Loop

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

```
i = 12

        false
while i < 8 :
  print("Number: " + str(i))
  i += 1

print("Program Complete.")
```

```
Program Complete.
```

While Loop

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

while input != "exit" :
  #Print the input
  print("You entered: " + input)
  #Prompt for input again
  input = input("Enter word: ")

print("Goodbye!")
```

Sentinel Value

```
Enter word: cat
You entered: cat
Enter word: dog
You entered : dog
Enter word: llama
You entered : llama
Enter word: exit
Goodbye!
```

While Loop

- In the while loop below, the sentinel value that will trigger 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: " +
          str(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!
```

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.

- Both are pre-test loops.

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!
```

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.

continue statement – for loop

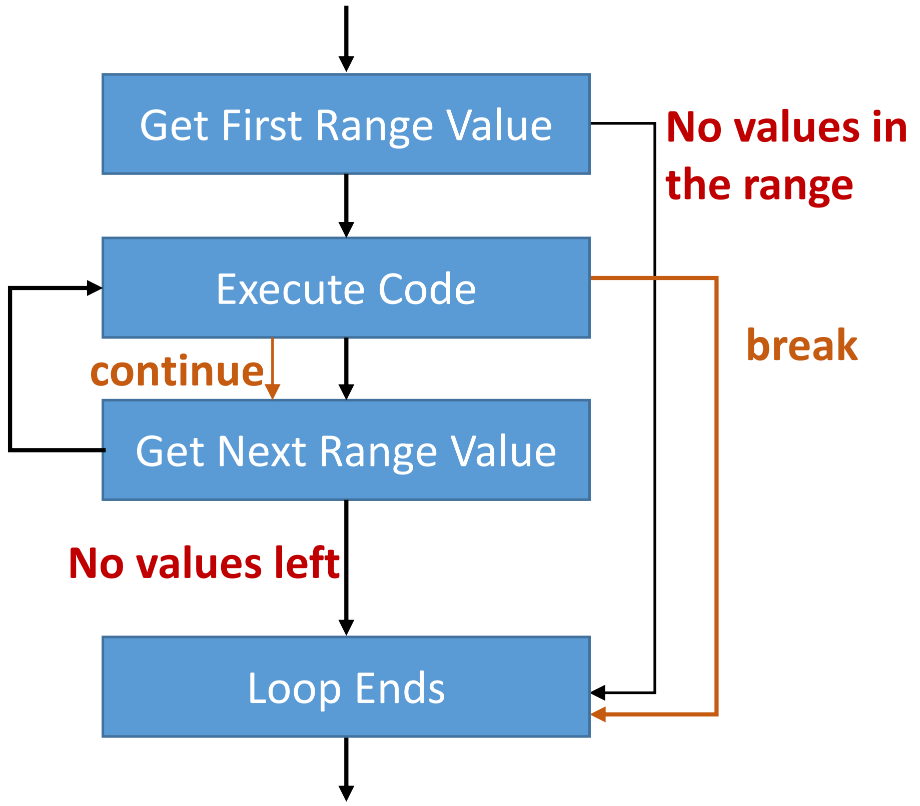
```
for i in range(1, 11) :
  if i % 2 == 1 :
    continue
  print("Number:", i)

print("All done!")
```

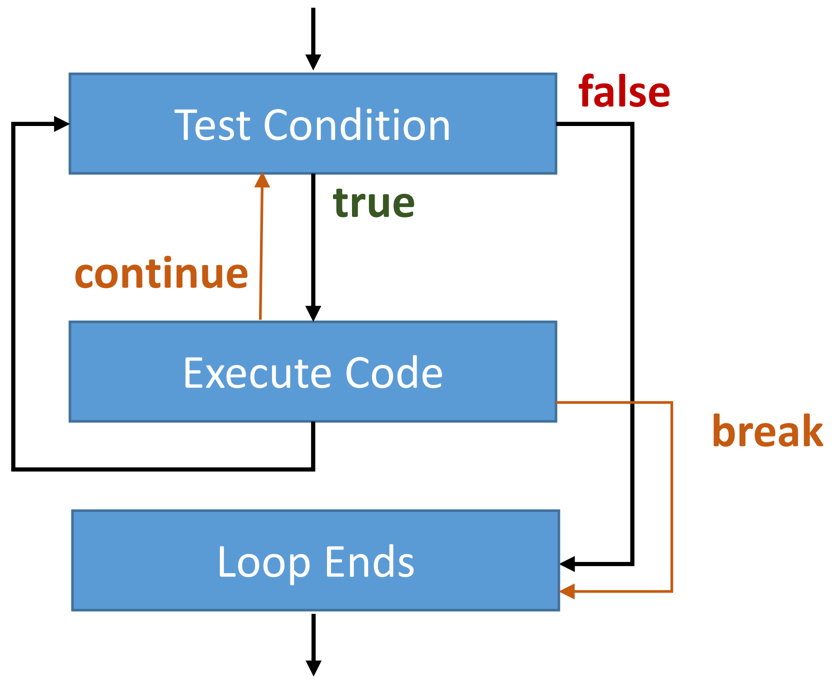
```
Number: 2
Number: 4
Number: 6
Number: 8
Number: 10
All done!
```

- 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


Get First Range Value
No values in the range
Execute Code
break
continue
Get Next Range Value
No values left
Loop Ends

While Loop


Test Condition
false
true
continue
Execute Code
break
Loop Ends

Infinite Loops

- An ***infinite loop*** is a loop that never stops.
  - In the example below, the loop's condition will never be false.

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

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

- Enhanced for loops will never be infinite, because the range of values is always finite.
- We'll look more closely at infinite loops when we move on to looping in Java.

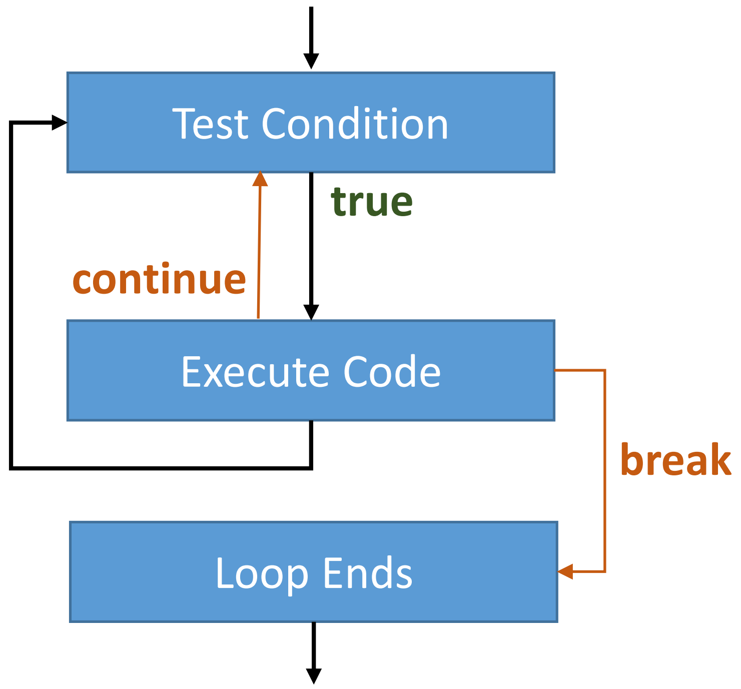
Infinite Loops

```
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



Nested For Loops

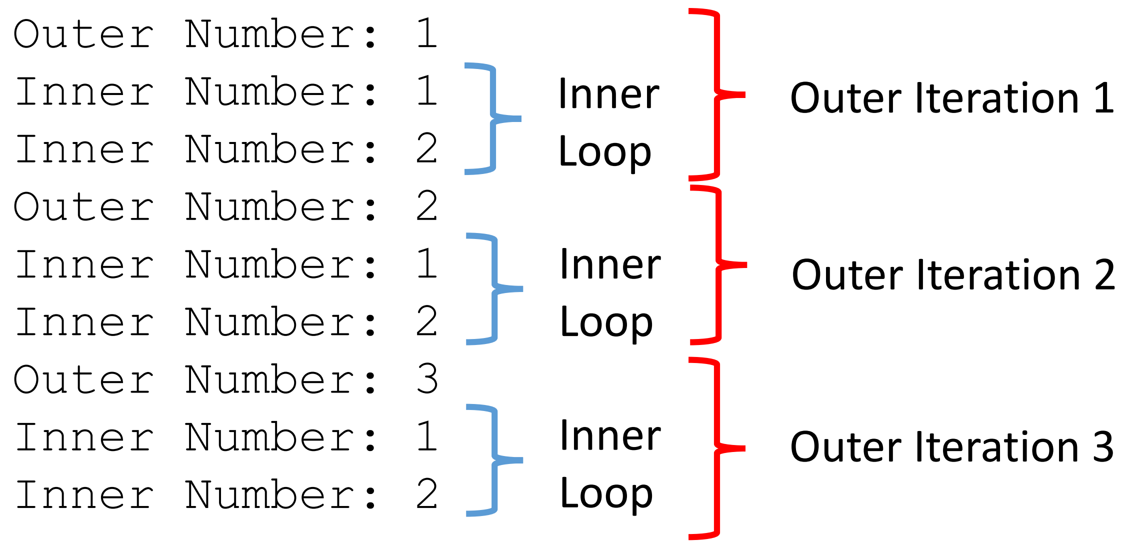
- A nested loop is a loop within a loop.
- For every iteration of the outer loop, the inner loop will be iterated to completion.

```
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
```