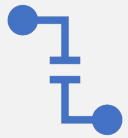
CS1011 Computer Programming in Python

Lecture 3:Loops (For and While)

Objectives



• Introduction to loops: `for` and `while` loops

• Using `range()` with `for` loops

• Infinite loops and loop control with `break`, `continue`, and `else`

Loops

A loop lets you run the same section of code over and over again until a specific condition is met

In python, we have looping statements such as:

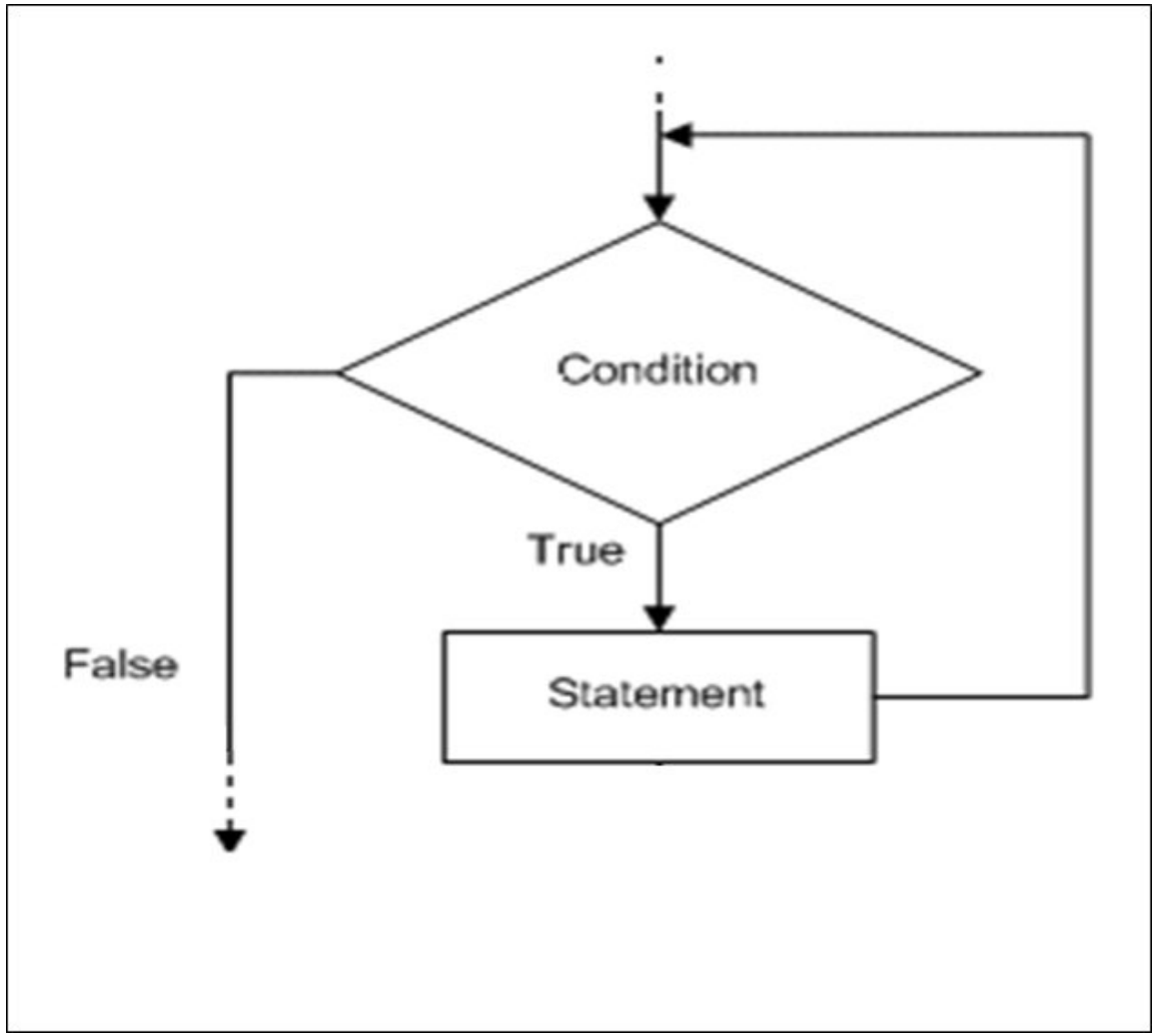
- While
- for

Loops: While

- It loops through a section of code as long as the specified condition is true.
- The indefinite or conditional loop keeps iterating until certain conditions are met.
- `while <condition>:`
  `    <body>`
- `condition` is a Boolean expression, just like in `if` statements. The 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.

Loops: while

- The condition is tested at the top of the loop. This is known as a *pre-test* loop. If the condition is initially false, the loop body will not execute at all.



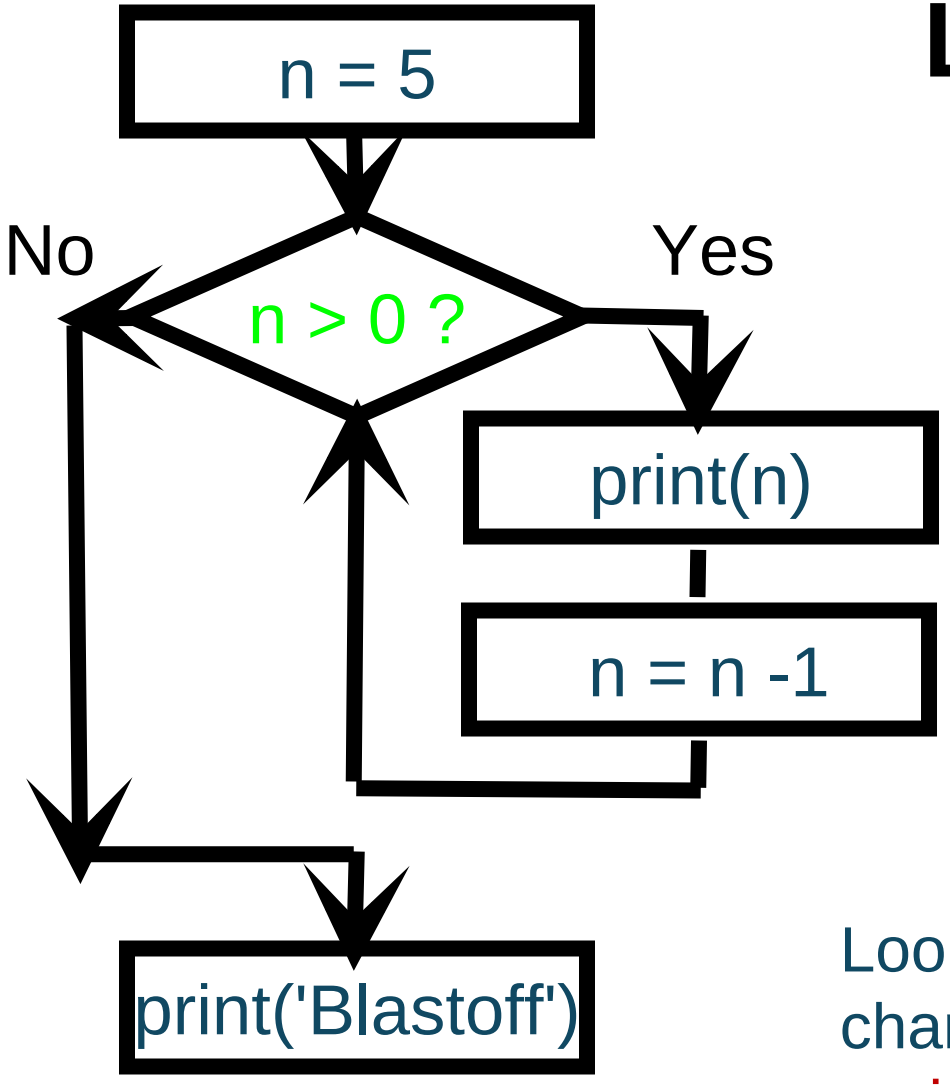
Indefinite Loop while (example)

- Print the numbers 1-5 proceeded with the sentence "the number is: " using a while loop

```
1   num = 1
2
3   while num <= 5:
4       print ("the number is: ", num)
5       num += 1
```

```
the number is:  1
the number is:  2
the number is:  3
the number is:  4
the number is:  5
>
```

Loops: while (example)




```
n = 5
while n > 0 :
    print n
    n = n – 1
print 'Blastoff!'
print n
```

5
4
3
2
1
Blastoff!
0

Loops (repeated steps) have iteration variables change each time through a loop. Often these iteration variables go through a sequence of numbers.

For Loops

- The `for` loop is a definite loop, meaning that the number of iterations is determined when the loop starts.
- We can't use a definite loop unless we know the number of iterations ahead of time.
- The `for` statement allows us to iterate through a sequence of values.
-
```
for <var> in <sequence>:
    <body>
```
- The loop index variable `var` takes on each successive value in the sequence, and the statements in the body of the loop are executed once for each value.

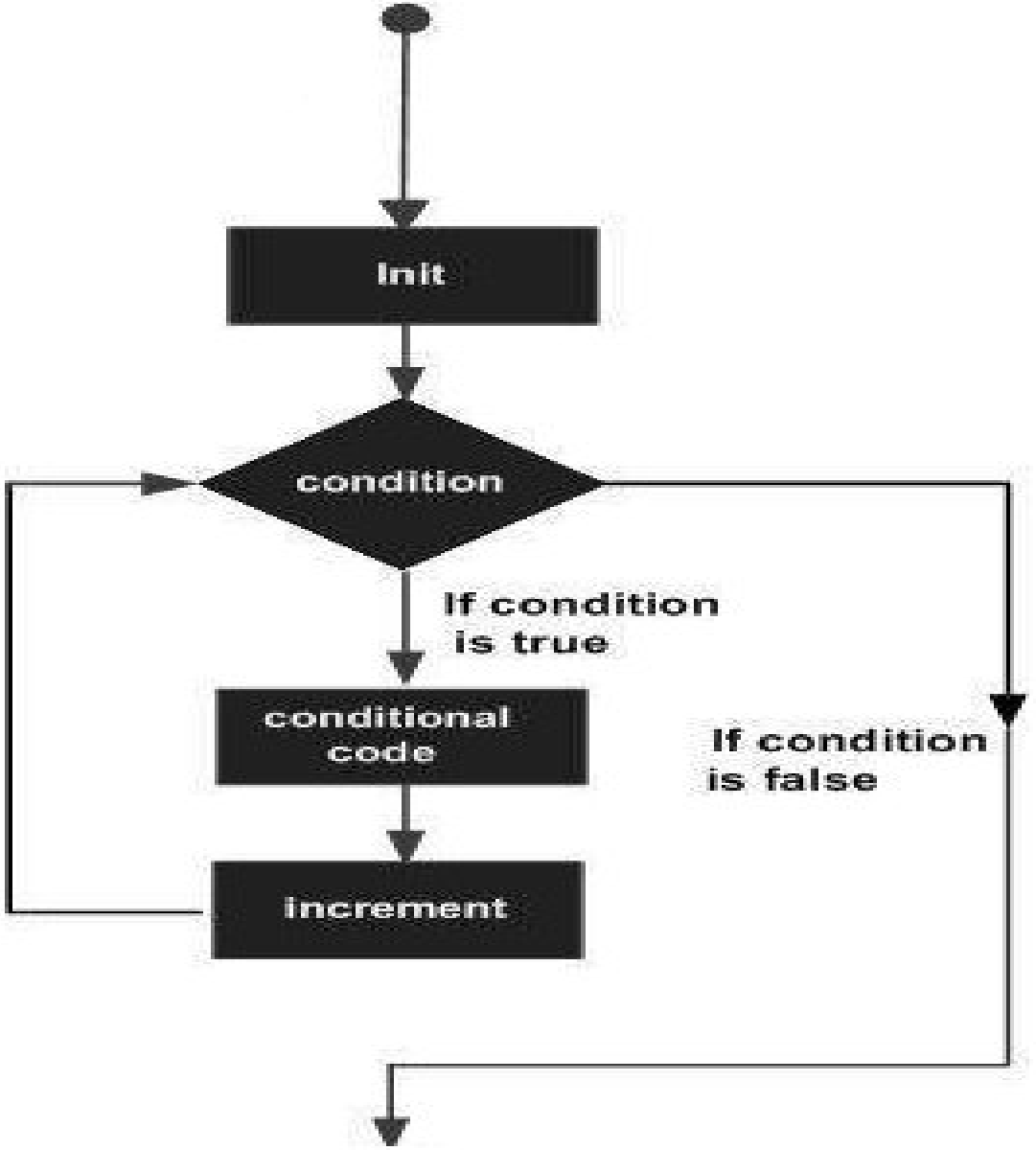
For Loops

- It is used to loop through a set of code a specified number of times
- The **range()** function determines the sequence of numbers, starting from 0 (default), and ends at a specified number
- It is possible to specify the starting value instead of 0
- **Syntax:**

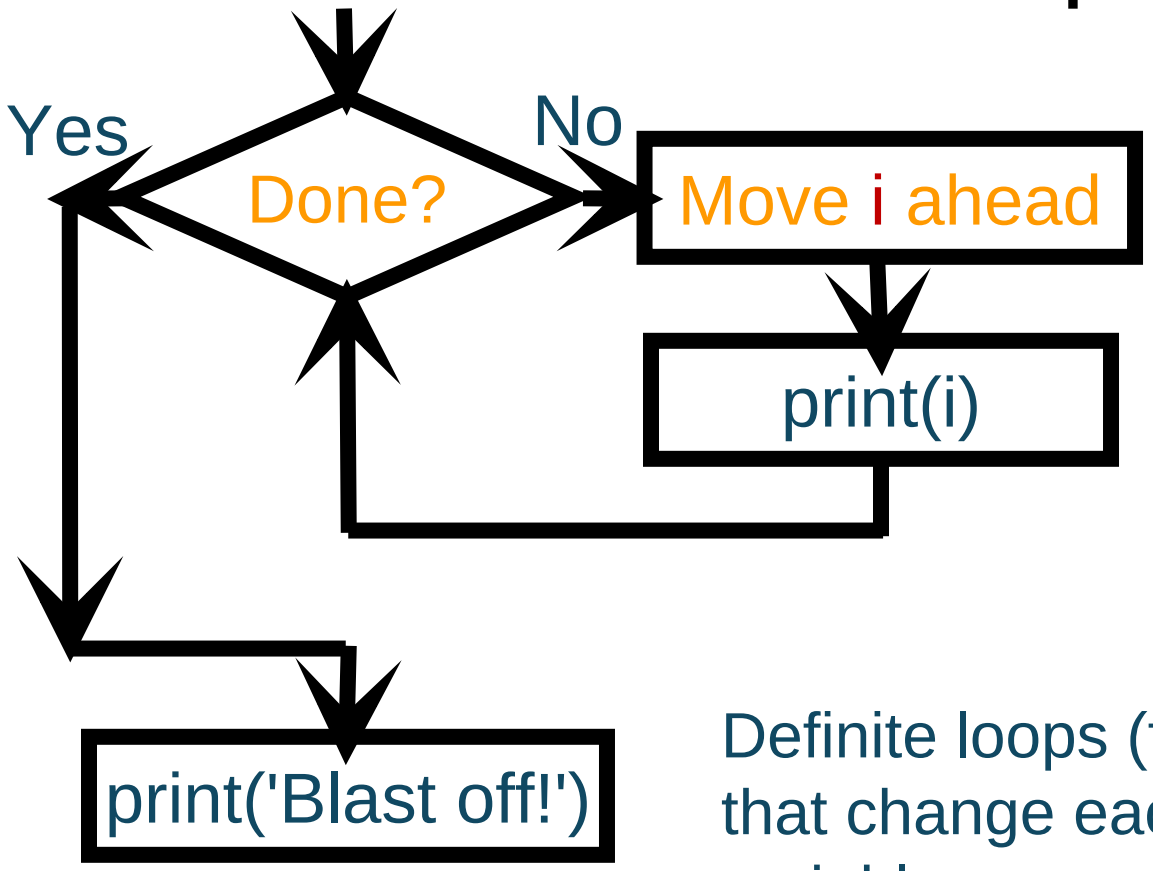
  for index in range(sequence):

      statement(s) to be executed

Loops: for


Init
condition
If condition
is true
conditional
code
increment
If condition
is false

Definite Loop: For Loop — Iterating Through a Sequence




```
for i in [5, 4, 3, 2, 1]
    print i
print 'Blastoff!'
```

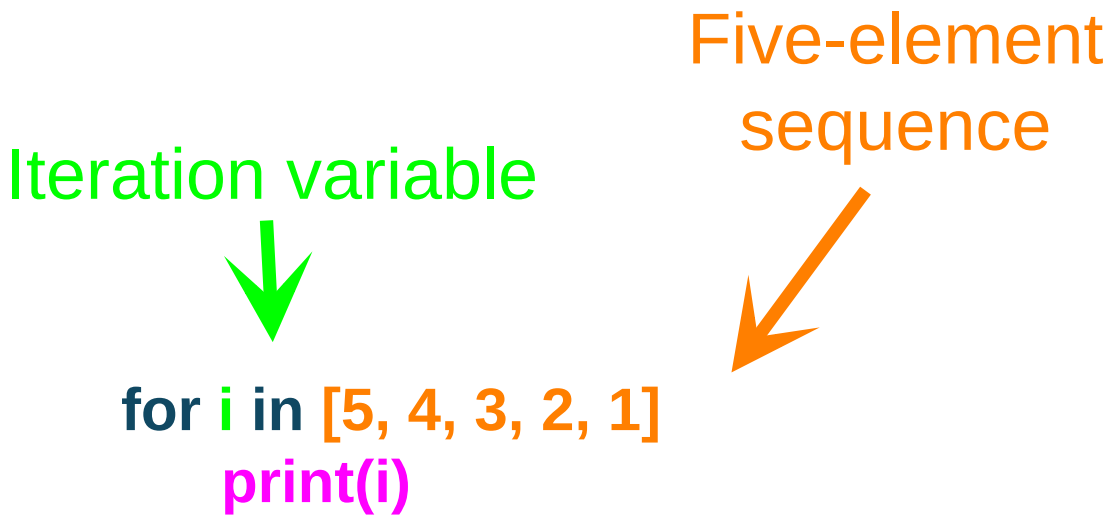
```
5
4
3
2
1
Blastoff!
```

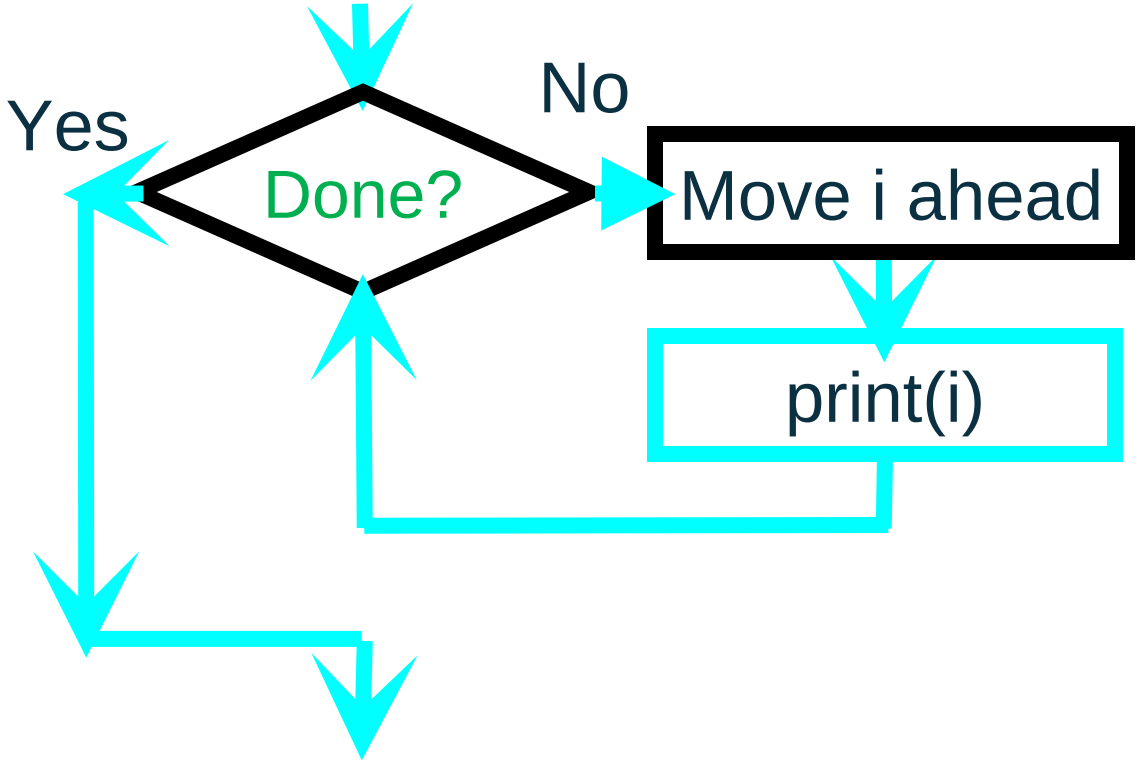
Definite loops (for loops) have explicit iteration variables that change each time through a loop. These iteration variables move through the sequence or set.

Looking at in...

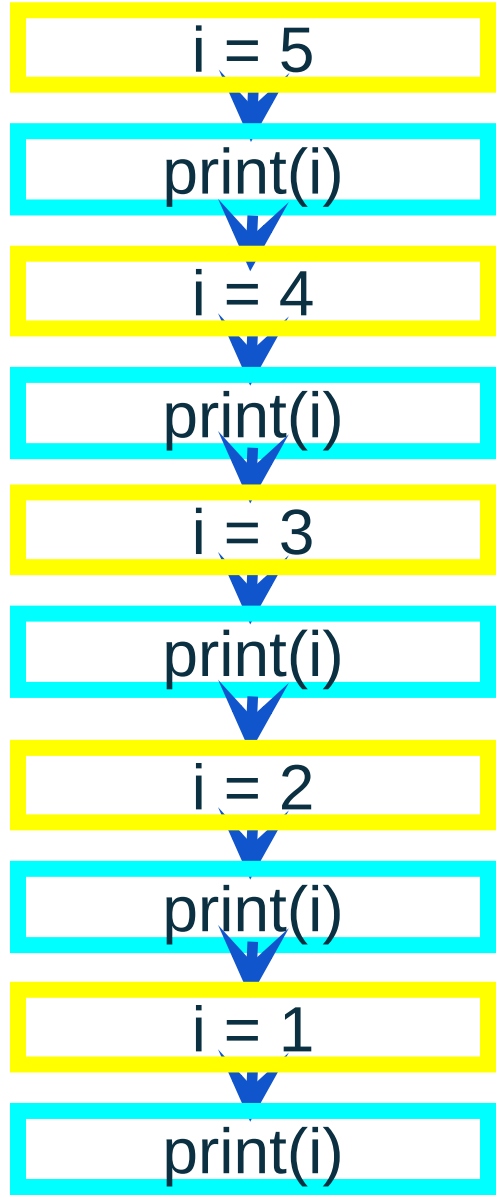
The iteration variable "iterates" through all of the values in the sequence

The block (body) of code is executed once for each value in the sequence






```
for i in [5, 4, 3, 2, 1] :
    print(i)
```



Definite Loop Example: For Loop with range()

- Print the numbers 0-5 proceeded with the sentence "the number is: " using a for loop

```
1  for num in range(6):
2      print("the number is: ",num)
3
4
5
```

```
the number is:  0
the number is:  1
the number is:  2
the number is:  3
the number is:  4
the number is:  5
```

Definite Loop Example: For Loop with range()

- Print the numbers 1-5 proceeded with the sentence "the number is: " using a for loop

main.py saved

```
1   for num in range(1,6):
2       print("the number is: ",num)
3
4
```

```
>
the number is:  1
the number is:  2
the number is:  3
the number is:  4
the number is:  5
```

Infinite Loop

- An **infinite loop** (also called endless loop) in Python is a sequence of statements in a program which executes endlessly.
- It occurs when the loop condition always evaluates to true. In other words, a loop becomes an infinite loop if a loop condition never becomes false.

PYTHON Copy code

```python
# An example of an infinite loop.
count = 0
while count < 100:
    print('Count: ',count)
    count = count * 1 # OOPS, does not change the count value.
```

```
Output:
      Count:  0
      Count:  0
      . . . .
    --never stops--
```

loop control : The break & continue Statement

The **break** statement can stop the loop even if the condition is true

The **continue** statement can stop the current iteration, and continue with the next

Breaking Out of a Loop

The **break** statement ends the current loop and jumps to the statement immediately following the loop

It is like a loop test that can happen anywhere in the body of the loop

```
1     sum = 0
2     number = 0
3
4     while number < 20:
5         number += 1
6         sum += number
7         if sum >= 100:
8             break
9
10    print("The number is", number)
11    print("The sum is", sum)
```

break out of the loop

```
The number is 14
The sum is 105
```

The program adds integers from 1 to 20 in this order to sum until sum is greater than or equal to 100. Without lines 7–8, this program would calculate the sum of the numbers from 1 to 20. But with lines 7–8, the loop terminates when sum becomes greater than or equal to 100. Without lines 7–8, the output would be:

```
The number is 20
The sum is 210
```

While Loop with Break

- Exit the loop after printing number 3

```
1   num = 1
2
3   while num <= 5:
4       print ("the number is: ", num)
5
6       if num == 3:
7           break
8
9       num += 1
```

```
the number is:  1
the number is:  2
the number is:  3
```

Finishing an Iteration with continue

- The **continue** statement ends the current iteration and jumps to the top of the loop and starts the next iteration.

jump to the end of the iteration

```
1  sum = 0
2  number = 0
3
4  while number < 20:
5      number += 1
6      if number == 10 or number == 11:
7          continue
8      sum += number
9
10 print("The sum is", sum)
```

```
The sum is 189
```

The program adds all the integers from 1 to 20 except 10 and 11 to sum. The continue statement is executed when number becomes 10 or 11. The continue statement ends the current iteration so that the rest of the statement in the loop body is not executed; therefore, number is not added to sum when it is 10 or 11.

Without lines 6 and 7, the output would be as follows:

```
The sum is 210
```

In this case, all the numbers are added to sum, even when number is 10 or 11. Therefore, the result is 210.

While Loop with Continue

- Skip printing number 3

```
1   num = 1
2
3   while num <= 5:
4       if num == 3:
5           num += 1
6           continue
7
8       print ("the number is: ", num)
```

```
the number is:  1
the number is:  2
the number is:  4
the number is:  5
```

else Clauses on Loops

- In a for or while loop the break statement may be paired with an else clause. If the loop finishes without executing the break, the else clause executes.
- In a for loop, the else clause is executed after the loop finishes its final iteration, that is, if no break occurred.
- In a while loop, it's executed after the loop's condition becomes false.
- In either kind of loop, the else clause is **not** executed if the loop was terminated by a break.

else Clauses on Loops

This is exemplified in the following `for` loop, which searches for prime numbers:

```
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print(n, 'equals', x, '*', n//x)
...             break
...     else:
...         # loop fell through without finding a factor
...         print(n, 'is a prime number')
...
2 is a prime number
3 is a prime number
4 equals 2 * 2
5 is a prime number
6 equals 2 * 3
7 is a prime number
8 equals 2 * 4
9 equals 3 * 3
```