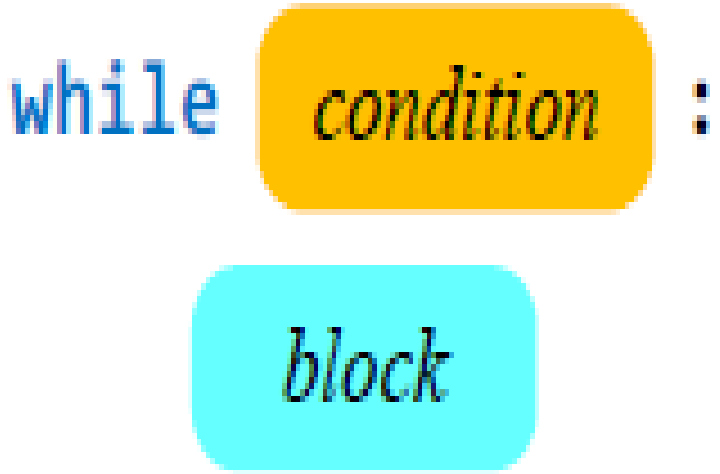
Iteration

The while Statement

The `while` statement has the general form:

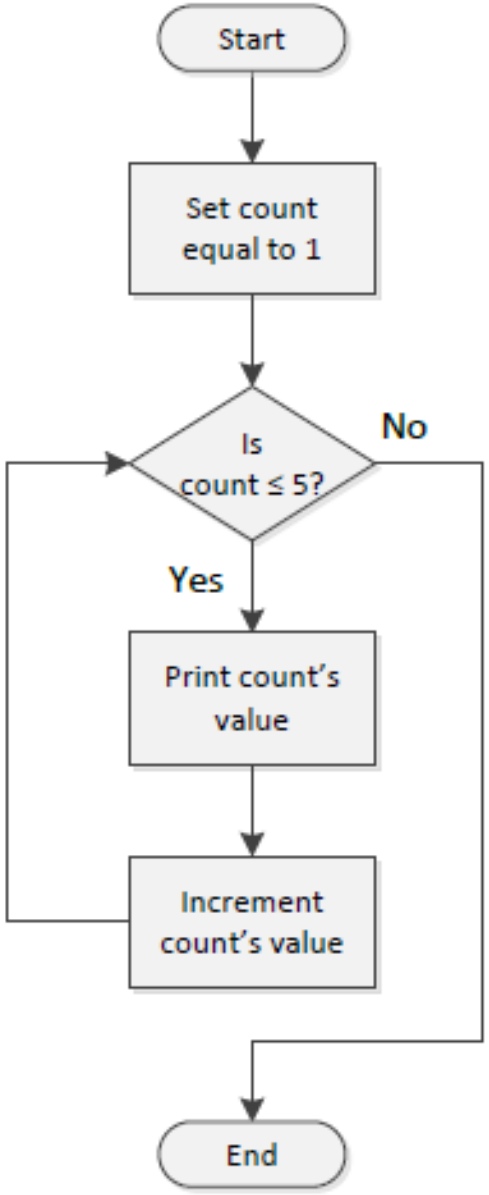


Example:

```
count = 1           # Initialize counter
while count <= 5:   # Should we continue?
    print(count)    # Display counter, then
    count += 1      # Increment counter
```

When executed, this program displays

```
1
2
3
4
5
```



Example: counts up from zero as long as the user wishes to do so

```
#  Counts up from zero.  The user continues the count by entering
#  'Y'.  The user discontinues the count by entering 'N'.

count = 0     # The current count
entry = 'Y'   # Count to begin with

while entry != 'N' and entry != 'n':
    # Print the current value of count
    print(count)
    entry = input('Please enter "Y" to continue or "N" to quit: ')
    if entry == 'Y' or entry == 'y':
        count += 1    # Keep counting
    # Check for "bad" entry
    elif entry != 'N' and entry != 'n':
        print('"' + entry + '" is not a valid choice')
    # else must be 'N' or 'n'
```

```
0
Please enter "Y" to continue or "N" to quit: y
1
Please enter "Y" to continue or "N" to quit: y
2
Please enter "Y" to continue or "N" to quit: y
3
Please enter "Y" to continue or "N" to quit: q
"q" is not a valid choice
3
Please enter "Y" to continue or "N" to quit: r
"r" is not a valid choice
3
Please enter "Y" to continue or "N" to quit: W
"W" is not a valid choice
3
Please enter "Y" to continue or "N" to quit: Y
4
Please enter "Y" to continue or "N" to quit: y
5
Please enter "Y" to continue or "N" to quit: n
```

Definite Loops vs. Indefinite Loops

Definite Loop

```
n = 1
while n <= 10:
    print(n)
    n += 1
```

```
n = 1
stop = int(input())
while n <= stop:
    print(n)
    n += 1
```

Indefinite Loop

```
done = False                # Enter the loop at least once
while not done:
    entry = int(input())    # Get value from user
    if entry == 999:        # Did user provide the magic number?
        done = True         # If so, get out
    else:
        print(entry)        # If not, print it and continue
```

The for Statement

- The while loop is ideal for indefinite loops.
- Python provides a more convenient way to express a definite loop.
- The for statement iterates over a sequence of values
- One way to express a sequence is via a tuple:

```
for n in 1, 2, 3, 4, 5, 6, 7, 8, 9, 10:
    print(n)
```

```
n = 1
while n <= 10:
    print(n)
    n += 1
```

range function

- A convenient way to express a sequence of integers that follow a regular pattern.
- The general form of the range function is

The general form of the `range` expression is

`range(` *begin* `,` *end* `,` *step* `)`

where

- *begin* is the first value in the range; if omitted, the default value is 0
- *end* is **one past** the last value in the range; the *end* value is always required and may **not** be omitted
- *step* is the amount to increment or decrement; if the *step* parameter is omitted, it defaults to 1 (counts up by ones)

range function

- `range(10)` → 0,1,2,3,4,5,6,7,8,9
- `range(1, 10)` → 1,2,3,4,5,6,7,8,9
- `range(1, 10, 2)` → 1,3,5,7,9
- `range(10, 0, -1)` → 10,9,8,7,6,5,4,3,2,1
- `range(10, 0, -2)` → 10,8,6,4,2
- `range(2, 11, 2)` → 2,4,6,8,10
- `range(-5, 5)` → −5,−4,−3,−2,−1,0,1,2,3,4
- `range(1, 2)` → 1
- `range(1, 1)` → (empty)
- `range(1, -1)` → (empty)
- `range(1, -1, -1)` → 1,0
- `range(0)` → (empty)

for loop using range function

```
for n in range(1, 11):
    print(n)
```

```
for n in range(21, 0, -3):
    print(n, end=' ')
```

It prints

```
21 18 15 12 9 6 3
```

More general for loop

- We initially emphasize the for loop's ability to iterate over integer sequences
- The for loop, however, can iterate over any iterable object.

```
word = input('Enter a word: ')
for letter in word:
    print(letter)
```

```
Enter a word: tree
t
r
e
e
```

More general for loop

```
for c in 'ABCDEF':
    print('[', c, ']', end='', sep='')
print()
```

```
[A][B][C][D][E][F]
```

More general for loop

```
word = input('Enter text: ')
vowel_count = 0
for c in word:
    if c == 'A' or c == 'a' or c == 'E' or c == 'e' \
       or c == 'I' or c == 'i' or c == 'O' or c == 'o':
        print(c, ', ', sep='', end='')  # Print the vowel
        vowel_count += 1                # Count the vowel
print(' (', vowel_count, ' vowels)', sep='')
```

```
Enter text: Mary had a little lamb.
a, a, a, i, e, a,  (6 vowels)
```

Nested Loops

- Just like with if statements, while and for blocks can contain arbitrary Python statements, including other loops.
- A loop can therefore be nested within another loop.

```
#  Get the number of rows and columns in the table
size = int(input("Please enter the table size: "))
#  Print a size x size multiplication table
for row in range(1, size + 1):
   for column in range(1, size + 1):
       product = row*column     # Compute product
       print(product, end=' ')  # Display product
   print()                      # Move cursor to next row
```

```
Please enter the table size: 10
1 2 3 4 5 6 7 8 9 10
2 4 6 8 10 12 14 16 18 20
3 6 9 12 15 18 21 24 27 30
4 8 12 16 20 24 28 32 36 40
5 10 15 20 25 30 35 40 45 50
6 12 18 24 30 36 42 48 54 60
7 14 21 28 35 42 49 56 63 70
8 16 24 32 40 48 56 64 72 80
9 18 27 36 45 54 63 72 81 90
10 20 30 40 50 60 70 80 90 100
```

Nested Loops : example

```
#  Get the number of rows and columns in the table
size = int(input("Please enter the table size: "))
#  Print a size x size multiplication table
for row in range(1, size + 1):
    for column in range(1, size + 1):
        product = row*column                    # Compute product
        print('{0:4}'.format(product), end='')  # Display product
    print()                                     # Move cursor to next row
```

```
Please enter the table size: 10
   1   2   3   4   5   6   7   8   9  10
   2   4   6   8  10  12  14  16  18  20
   3   6   9  12  15  18  21  24  27  30
   4   8  12  16  20  24  28  32  36  40
   5  10  15  20  25  30  35  40  45  50
   6  12  18  24  30  36  42  48  54  60
   7  14  21  28  35  42  49  56  63  70
   8  16  24  32  40  48  56  64  72  80
   9  18  27  36  45  54  63  72  81  90
  10  20  30  40  50  60  70  80  90 100
```

Nested Loops : example

```
#  File permuteabc.py

#  The first letter varies from A to C
for first in 'ABC':
    for second in 'ABC': # The second varies from A to C
        if second != first:  #  No duplicate letters allowed
            for third in 'ABC':  # The third varies from A to C
                # Don't duplicate first or second letter
                if third != first and third != second:
                    print(first + second + third)
```

```
ABC
ACB
BAC
BCA
CAB
CBA
```

The break statement

```
#  Allow the user to enter a sequence of nonnegative
#  numbers.  The user ends the list with a negative
#  number.  At the end the sum of the nonnegative
#  numbers entered is displayed.  The program prints
#  zero if the user provides no nonnegative numbers.

entry = 0       # Ensure the loop is entered
sum = 0         # Initialize sum

#  Request input from the user
print("Enter numbers to sum, negative number ends list:")

while True:                 # Loop forever?  Not really
    entry = int(input())    # Get the value
    if entry < 0:           # Is number negative number?
        break               # If so, exit the loop
    sum += entry            # Add entry to running sum
print("Sum =", sum)         # Display the sum
```

The break statement

```
while Condition 1 :
    Part A
    if Condition 2 :
        Part B
        break
    Part C
```

Eliminate the break statement

```
looping = True
while looping and Condition 1 :
    Part A
    if Condition 2 :
        Part B
        looping = False
    else:
        Part C
```

The break statement

```
word = input('Enter text (no X\'s, please): ')
vowel_count = 0
for c in word:
    if c == 'A' or c == 'a' or c == 'E' or c == 'e' \
       or c == 'I' or c == 'i' or c == 'O' or c == 'o':
        print(c, ', ', sep='', end='')  # Print the vowel
        vowel_count += 1                # Count the vowel
    elif c == 'X' or c =='x':
        break
print(' (', vowel_count, ' vowels)', sep='')
```

The continue Statement

```
sum = 0
done = False
while not done:
    val = int(input("Enter positive integer (999 quits):"))
    if val < 0:
        print("Negative value",  val,  "ignored")
        continue  # Skip rest of body for this iteration
    if val != 999:
        print("Tallying", val)
        sum += val
    else:
        done = (val == 999)   # 999 entry exits loop
print("sum =", sum)
```

Infinite Loops

- An infinite loop is a loop that executes its block of statements repeatedly until the user forces the program to quit.
- Once the program flow enters the loop's body it cannot escape. Infinite loops sometimes are by design

Intentional infinite loops should be made obvious. For example,

```
while True:
    # Do something forever. . .
```

Unintentional infinite loop

```
#  List the factors of the integers 1...MAX
MAX = 20                              # MAX is 20
n = 1   # Start with 1
while n <= MAX:                       # Do not go past MAX
    factor = 1                        # 1 is a factor of any integer
    print(end=str(n) + ': ')          # Which integer are we examining?
    while factor <= n:                # Factors are <= the number
        if n % factor == 0:           # Test to see if factor is a factor of n
            print(factor, end=' ')    # If so, display it
            factor += 1               # Try the next number
    print()  # Move to next line for next n
    n += 1
```

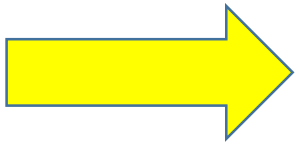
How to avoid Unintentional infinite loop

- The loop's condition must not be a tautology (a Boolean expression that can never be false). For example, the statement

```
while i >= 1 or i <= 10:
    # Block of code follows ...
```

- The condition of a `while` must be true initially to gain access to its body. The code within the body must modify the state of the program in some way so as to influence the outcome of the condition that is checked at each iteration. This usually means the body must be able to modify one of the variables used in the condition. Eventually the variable assumes a value that makes the condition false, and the loop terminates.

```
while factor <= n:
    if n % factor == 0:
        print(factor, end=' ')
        factor += 1
```



```
while factor <= n:
    if n % factor == 0:
        print(factor, end=' ')
    factor += 1
```

ITERATION EXAMPLES

```
max_value = int(input('Display primes up to what value? '))
value = 2  # Smallest prime number
while value <= max_value:
    # See if value is prime
    is_prime = True  # Provisionally, value is prime
    # Try all possible factors from 2 to value - 1
    trial_factor = 2
    while trial_factor < value:
        if value % trial_factor == 0:
            is_prime = False    # Found a factor
            break               # No need to continue; it is NOT prime
        trial_factor += 1       # Try the next potential factor
    if is_prime:
        print(value, end= ' ')  # Display the prime number
    value += 1                  # Try the next potential prime number
print()  # Move cursor down to next line
```

```
Display primes up to what value? 90
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89
```