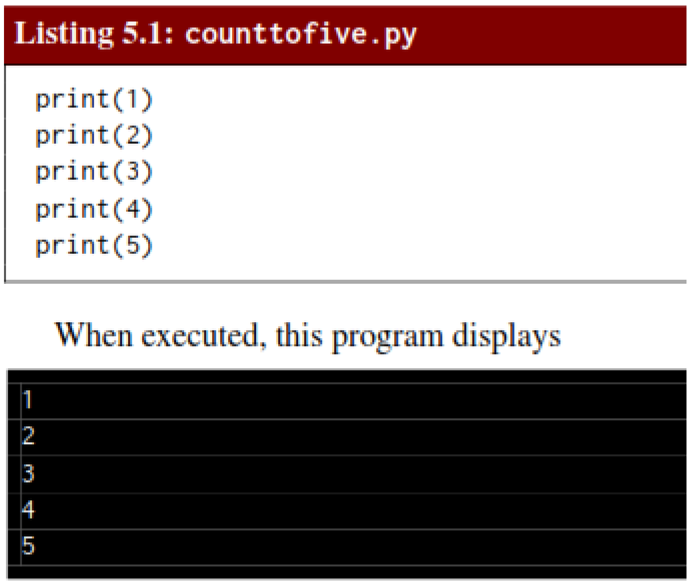
While Loop

Contents

1. Iteration

Iteration repeats the execution of a sequence of code. Iteration is useful for solving many programming problems. Iteration and conditional execution form the basis for algorithm construction.



Listing 5.1: `counttofive.py`

```
print(1)
print(2)
print(3)
print(4)
print(5)
```

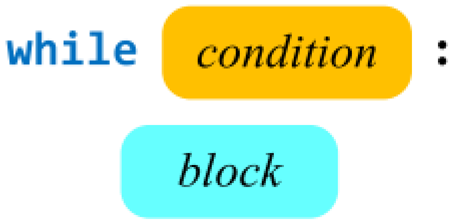
When executed, this program displays

```
1
2
3
4
5
```

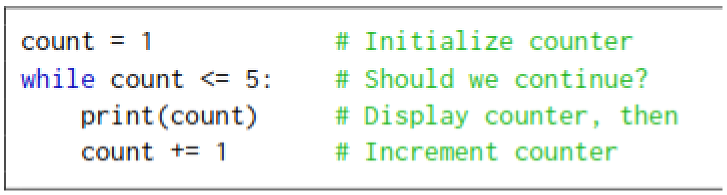
Now How would you count to 10,000??

Executing same section of code executing the same section of code over and over is known as **iteration, or looping**. Python has two different statements, **while** and **for**, that enable iteration.

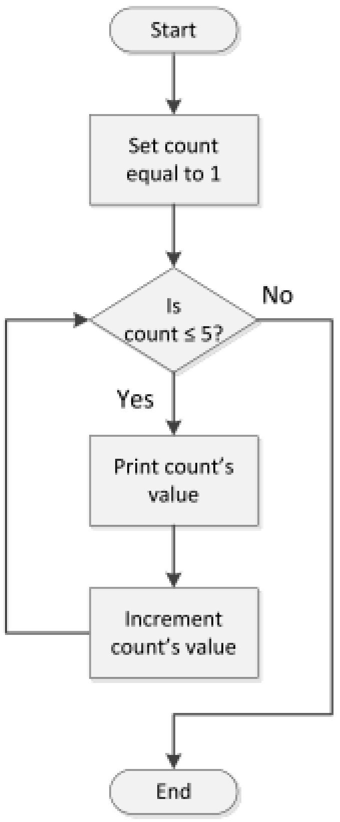
1.1 While Statement



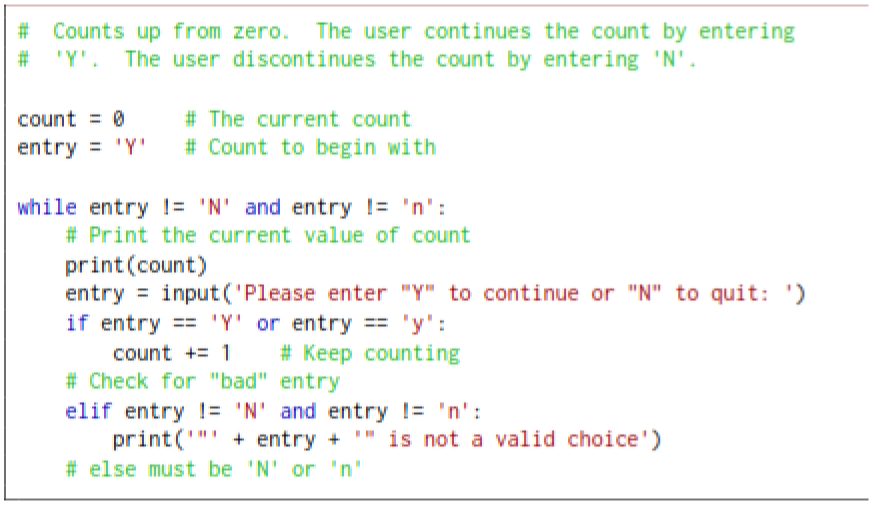
Example 1:



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



Example 2:

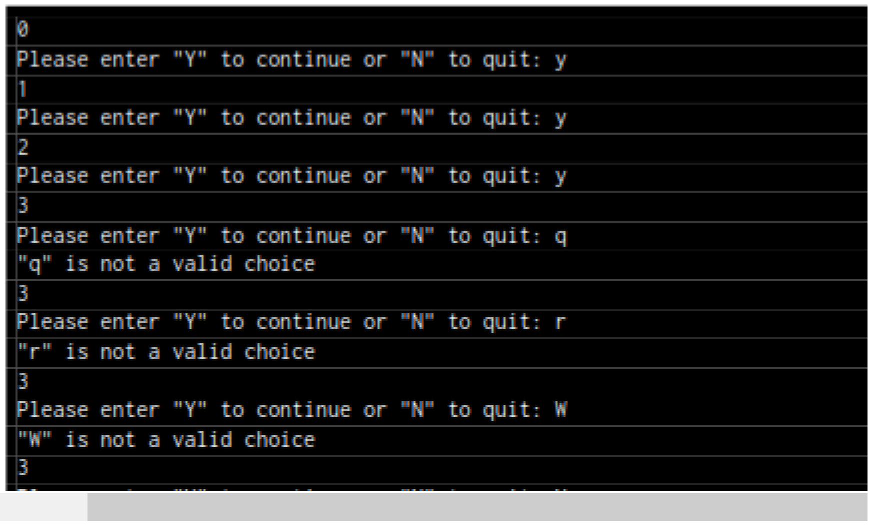


```python
#  Counts up from zero.  The user continues the count by entering
#  'Y'.  The user discontinues the count by entering 'N'.

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'
```

A sample run of Listing 5.3 (countup.py) produces



```
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
```

In Python, sometimes it is convenient to use a simple value as conditional expression in an **if** or **while** statement. Python interprets the integer value **0** and floating-point value **0.0** both as **False**. All other integer and floating-point values, both positive and negative, are considered **True**. This means the following code:

```python
x = int(input())  # Get integer from user
while x:
    print(x)    # Print x only if x is nonzero
    x -= 1      # Decrement x
```

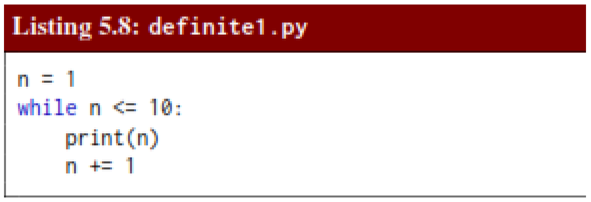
is equivalent to

```python
x = int(input())  # Get integer from user
while x != 0:
    print(x)    # Print x only if x is nonzero
    x -= 1      # Decrement x
```

1.2 Definite Loops vs. Indefinite Loop

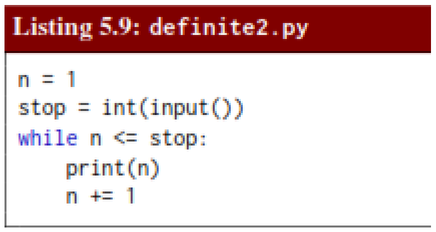
1.2.1 Definite loop

We can inspect the code and determine the exact number of iterations the loop will perform. This kind of loop is known as a **definite loop**, since we can predict exactly how many times the loop repeats.



Listing 5.8: definite1.py

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



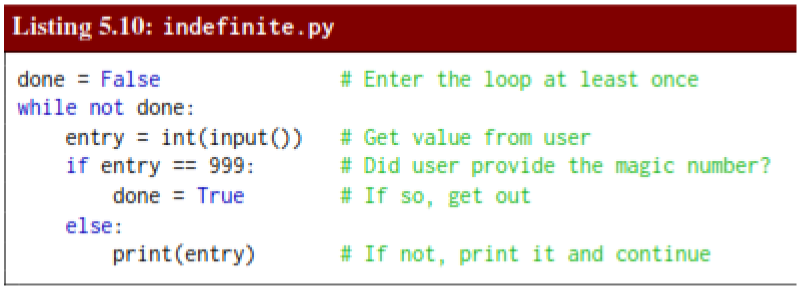
Listing 5.9: definite2.py

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

Looking at the source code of Listing 5.9 (definite2.py), we cannot predict how many times the loop will repeat. The number of iterations depends on the input provided by the user. However, at the program's point of execution after obtaining the user's input and before the start of the execution of the loop, we would be able to determine the number of iterations the while loop would perform. Because of this, the loop in Listing 5.9 (definite2.py) is considered to be a definite loop as well.

1.2.2 Indefinite loop

we cannot predict at any point during the loop's execution how many iterations the loop will perform.



Listing 5.10: indefinite.py

```python
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
```