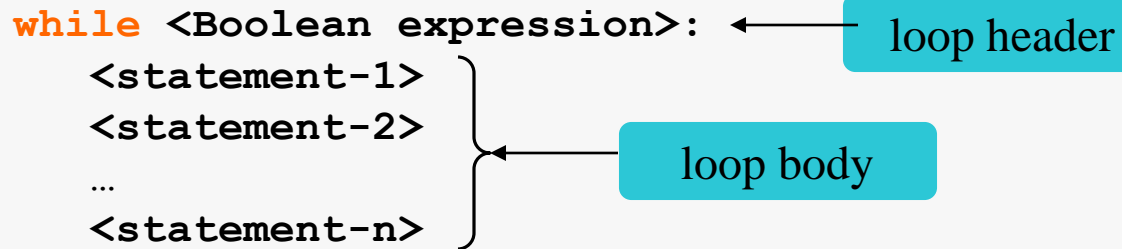


Special Topics in Computer Science- CSC 4992

The **while** Loop and Indefinite Loops

Syntax of the **while** Loop

```
while <Boolean expression>: ← loop header  
    <statement-1>  
    <statement-2>  
    ...  
    <statement-n> } ← loop body
```

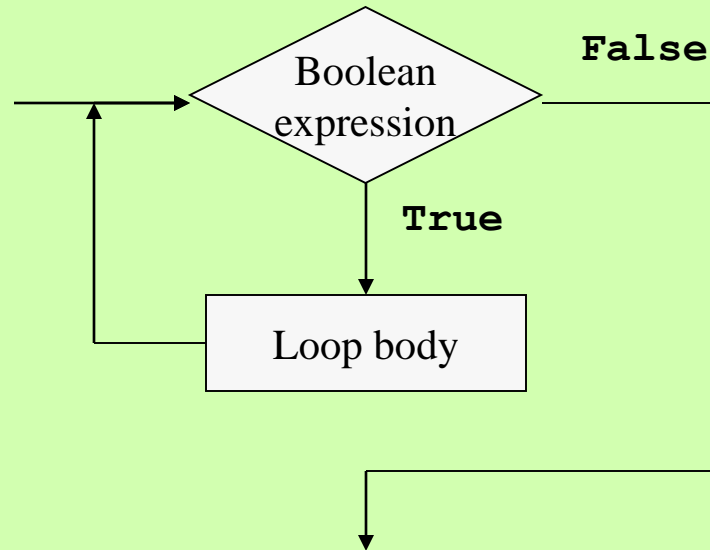


A *Boolean expression* can be a comparison of two numbers or strings, using ==, !=, <, >, <=, >=

These operations return the Boolean values **True** or **False**

Note that == means *equal to*, whereas = means *assignment*

Behavior of the **while** Loop




```
while <Boolean expression>:  
    <sequence of statements>
```

```
# The condition  
# The loop body
```

Logic of the **while** Loop

```
i = 0  
while i < count:  
    guess = (guess + x / guess) / 2  
    i += 1
```



A light blue rounded rectangular callout box with the text "loop control variable" is positioned to the right of the code. A black arrow points from the box to the variable 'i' in the first line of code, 'i = 0'.

Before the loop header, the *loop control variable* must be set to the appropriate initial value

Somewhere in the loop body, a statement must reset the loop control variable so that the Boolean expression eventually becomes **False**, to terminate the loop

These steps were *automatic* in the **for** loop

Which Loop Should I Use?

- A **for** loop works well when the number of iterations is predictable (usually count-controlled)
- A **while** loop works well when the number of iterations is not predictable
- How could the # of iterations be unpredictable?

Which Loop

Example (1)

```
sum = 0
anum = 1
while anum <= 10:
    sum = sum + anum
print(sum)
```

```
sum = 0
anum = 1                                #initialization
while anum <= 10:                        #condition
    sum = sum + anum
    anum = anum + 1                      #change of state
print(sum)
```

Example (2)

$$y = \sqrt{x}$$

Sir Isaac Newton developed a method to approximate the value of the square root of a given number. Suppose **x** is the number and **guess** is an approximation of the square root of **x**. Then a closer approximation to the actual square root of **x** is equal to

$$(\text{guess} + x / \text{guess}) / 2$$

The initial value of **guess** is $x / 2$

Try Some Guesses in the Shell

```
>>> from math import sqrt
>>> x = 2
>>> sqrt(x)
1.4142135623730951
>>> guess = x / 2
>>> guess
1.0
>>> guess = (guess + x / guess) / 2
>>> guess
1.5
>>> guess = (guess + x / guess) / 2
>>> guess
1.4166666666666665
>>>
```


Use a Loop to Repeatedly Guess

The approximation algorithm repeatedly replaces a guess with a better approximation. Write a program that allows the user to specify the value of **x** and the number of iterations used in this approximation. The program computes and displays the resulting estimate of the square root of **x**, as well as Python's own estimate using Python's **math.sqrt** function.

Use a **for** Loop

```
import math

x = float(input("Enter a number: "))
count = int(input("Enter the number of iterations: "))

guess = x / 2
for i in range(count):
    . . .
print("Guess:  %0.15f" % guess)
print("Actual: %0.15f" % math.sqrt(x))
```

We perform a definite number of approximations

Or Use a **while** Loop

```
import math

x = float(input("Enter a number: "))
count = int(input("Enter the number of iterations: "))

guess = x / 2
i = 0
while i < count:
    . . .
    . . .
print("Guess:  %0.15f" % guess)
print("Actual: %0.15f" % math.sqrt(x))
```

This loop is also count-controlled, and is equivalent in meaning to the **for** loop

Another Way to Look at Newton's Method

We repeatedly improve the **guess** until the difference between the square of the **guess** and **x** reaches a given tolerance value.

```
tolerance = 0.00001
while ...
    ...
```

We cannot predict how many times this process will be repeated

This is called an *indefinite loop* or a *conditional iteration*

The while Loop

```
import math

x = float(input("Enter a number: "))

tolerance = 0.00001
guess = x / 2
While ...:
    ...
print("Guess: %0.15f" % guess)
print("Actual: %0.15f" % math.sqrt(x))
```

Add a tolerance value and quit the loop when the condition becomes false

Which Loop

- The while loop continues to perform the statements in the body of the loop as long as the condition remains **true**.
- Once the condition evaluates to **false** the while loop **stops** performing the statements in the body.
- A condition can be any Python Boolean expression.

Break

- A **break** statement in a loop, if executed, exits the loop
- It exists immediately, skipping whatever remains of the loop as well as the else statement (if it exists) of the loop

Continue

- A **continue** statement, if executed in a loop, means to immediately jump back to the top of the loop and re-evaluate the conditional
- Any remaining parts of the loop are skipped for the one iteration when the continue was executed

Sentinel-Controlled Loops

- An indefinite loop can test for the presence of a **special value** called a *sentinel*
- When the sentinel is reached, the loop should **stop**

An Example Problem

Input test scores from the user and print the average score.

The user signals the end of input by simply pressing the enter or return key.

This value shows up as an empty string, which is the sentinel value.

Sentinel-Controlled Loops

```
total = 0
count = 0
data = input("Enter a score or just return to quit: ")

while data != "":
    total += int(data)
    count += 1
    data = input("Enter a score or just return to quit: ")

if count == 0:
    print("No scores were entered.")
else:
    print("The average is", total / count)
```

The input statement must be written twice

Simplify with a **break** Statement

```
total = 0
count = 0

while True:
    data = input("Enter a score or just return to quit: ")
    if data == "":
        break
    total += int(data)
    count += 1

if count == 0:
    print("No scores were entered.")
else:
    print("The average is", total / count)
```

break causes an immediate exit from the enclosing loop

When Should I Use a **break**?

- When the loop body executes at least once but the number of iterations is unpredictable
- The condition of the enclosing **while** loop is just **True**
- The condition for loop exit is placed in an enclosing **if** statement
- A **break** may also be helpful within a **for** loop

Input Checking *redux*

```
rate = int(input('Enter the interest rate[0-100]: '))
if rate < 0 or rate > 100:
    print('ERROR: Rate must be between 0 and 100!')
else:
    interest = principal * rate / 100
    print('Your interest is', interest)
```

We need to continue this process until the user enters a legitimate number

Input Checking *redux*

```
while True:
    rate = int(input('Enter the interest rate[0-100]: '))
    if rate < 0 or rate > 100:
        print('ERROR: Rate must be between 0 and 100!')
    else:
        break
interest = principal * rate / 100
print('Your interest is', interest)
```

The loop controls the input process

The computation process occurs only after the loop has finished

Guess

```
import random

smaller = int(input("Enter the smaller number: "))
larger = int(input("Enter the larger number: "))
myNumber = random.randint(smaller, larger)
count = 0
while True:
    count += 1
    userNumber = int(input("Enter your guess: "))
    if userNumber < myNumber:
        print("Too small")
    elif userNumber > myNumber:
        print("Too large")
    else:
        print("You've got it in", count, "tries!")
        break
```


Product

```
prod= 1.0
data = input("Enter a number: ")
while data != "":
    number = float(data)
    prod*= number
    data = input("Enter the next number: ")
print("The product is", prod)
```