ECE-122
Chapter-2:
Elements of Programming
2.6 Iterations: **while** loops

for loops: review

- The **for** loop. It is a ***definite*** loop that executes some instructions a finite number of times.
  - using the **range** function

```
for i in range(6):
    print("Enter loop",i)
```

```
for i in range(3,6):
    print("Enter loop",i)
```

```
for i in range(0,10,2):
    print("Enter loop",i)
```

```
for i in range(6):
    if i%2==0:
        print("Enter loop",i)
```

```
for i in range(6):
    for j in range(6):
        print("Enter loop",i,j)
```

```
animals=["lion","giraffe","elephant","zebra","monkey","tiger"]
for i in range(len(animals)):
    print(animals[i])
```

for loops: review

- using the sequence of value/items defined in a **list**

```
for i in [0,1,2,3,4,5]:
    print("Enter loop",i)
```

```
for i in [0,2,1,1,4,3]:
    print("Enter loop",i)
```

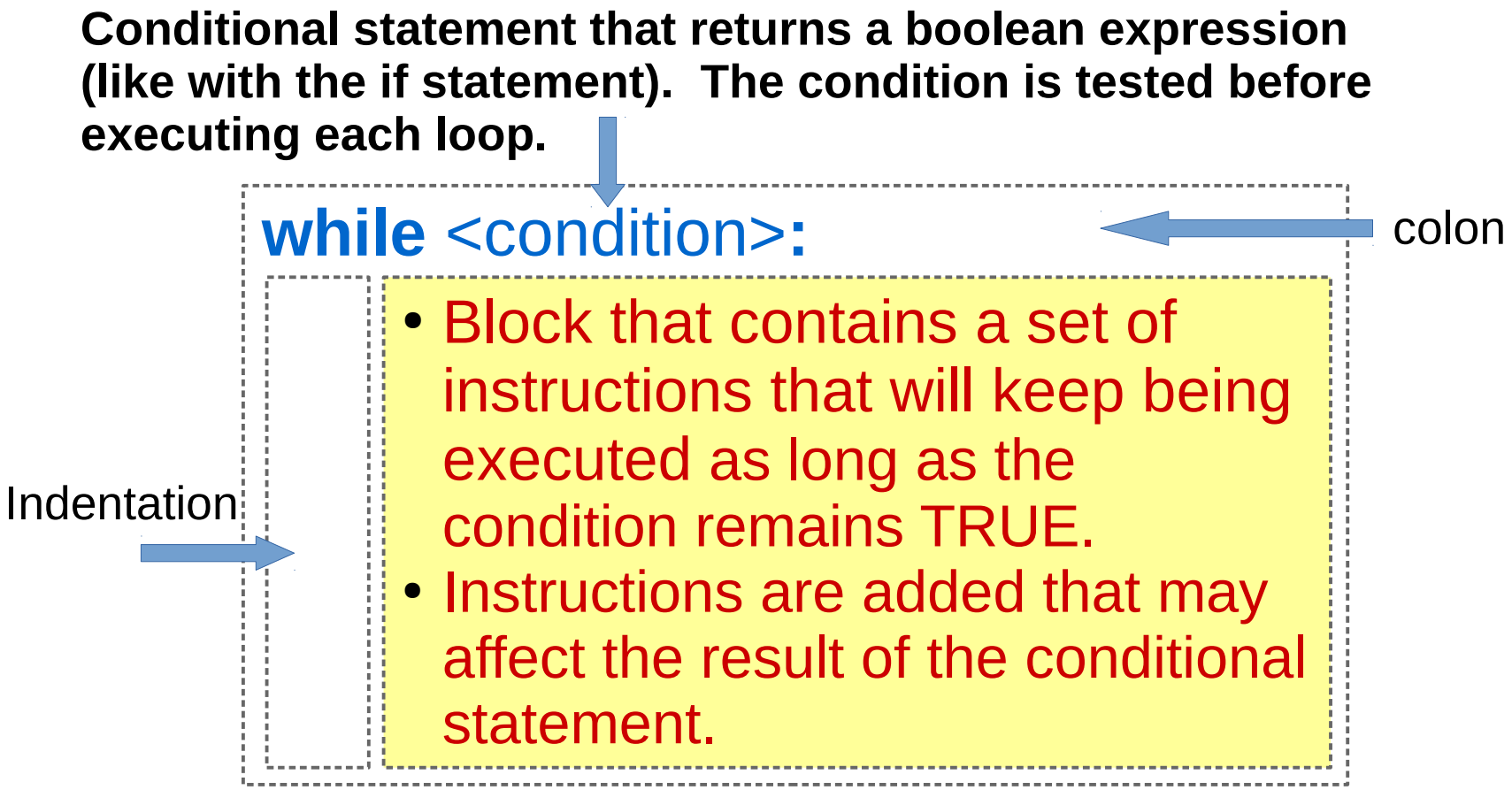
```
my_list=[0,2,1,1,4,3]
for i in my_list:
    print("Enter loop",i)
```

```
animals=["lion","giraffe","elephant","zebra","monkey","tiger"]
for item in animals:
    print(item)
```

- **for** loop: restrictions
  - The program must know exactly (in advance) how many iterations to do.
  - In some situations, the number of iterations we need has to be decided at runtime.

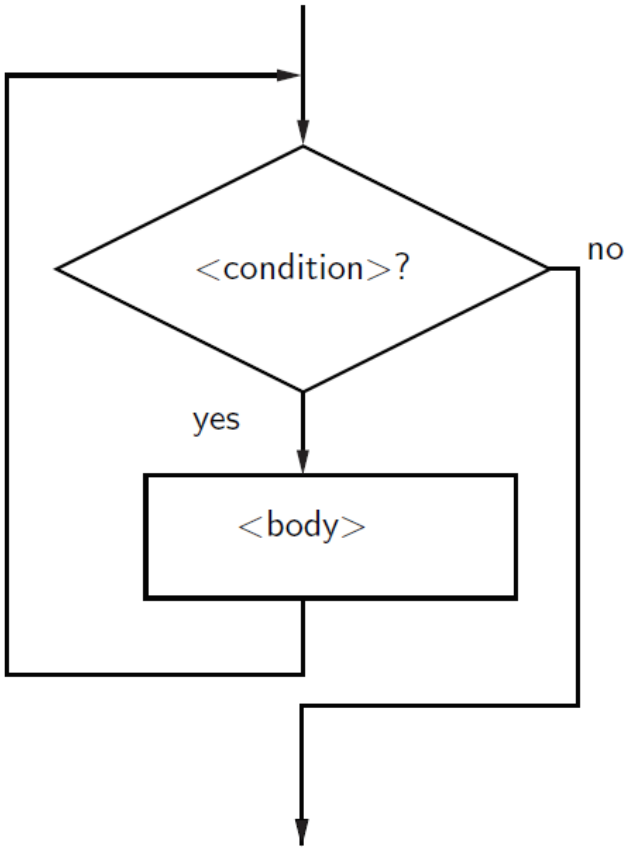
while loops: definition

- The **while** loop. This is an ***indefinite/conditional*** loop that keeps executing some instructions until certain conditions are met.
- This can easily be accomplished using the while statement



while loops: definition

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



while loops: for vs while

- A **while** loop can do everything that a **for** loop does.
  Example: printing from 0 to 9

```
for i in range(10):
    print("Enter loop",i)
```

```
i=0
while i<10: #i<=9 ok too
    print("Enter loop",i)
    i=i+1
```

```
Enter loop 0
Enter loop 1
Enter loop 2
Enter loop 3
Enter loop 4
Enter loop 5
Enter loop 6
Enter loop 7
Enter loop 8
Enter loop 9
```

  - The while loop requires us to manage the loop variable i by initializing it to 0 before the loop and incrementing it at the bottom of the body.
  - In the for loop this is handled automatically.

while loops: for vs while

- More Examples

```
for i in range(-5,7,2):
    print("Enter loop",i)
```

```
i=-5
while i<=5:
    print("Enter loop",i)
    i=i+2
```

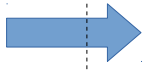
```
Enter loop -5
Enter loop -3
Enter loop -1
Enter loop 1
Enter loop 3
Enter loop 5
```

```
for i in range(5,-7,-2):
    print("Enter loop",i)
```

```
i=5
while i>=-5:
    print("Enter loop",i)
    i=i-2
```

```
Enter loop 5
Enter loop 3
Enter loop 1
Enter loop -1
Enter loop -3
Enter loop -5
```

```
animals=["lion","giraffe","elephant","zebra","monkey","tiger"]
i=0
while i<len(animals):
    print(animals[i])
    i=i+1
```



```
lion
giraffe
elephant
zebra
monkey
tiger
```

while loops: danger

- The while statement is simple, but yet powerful and dangerous – Here is a common source of program errors:

```
i=0
while i<10:
      print("Enter loop",i)
```

```
Enter loop 0
Enter loop 0
Enter loop 0
Enter loop 0
...
<infinite loop>
```

(value of i=0 is never updated-condition is always true)

  - What should you do it you are caught in an infinite loop?
    - First, try pressing control-c
    - If that doesn't work, try control-alt-delete
    - If that doesn't work, push the reset button!

while loops: interactive use

- One good use of the indefinite loop is to write interactive loops. Interactive loops allow a user to repeat certain portions of a program on demand.

```
count=1
result=int(input("Can you guess? 2+3="))
while result!=5:
    result=int(input("Nope...try again="))
    count=count+1
print("Well done after "+str(count)+" trial(s)")
```

```
Can you guess? 2+3=6
Nope...try again=4
Nope...try again=3
Nope...try again=5
Well done after 4 trial(s)
```

while loops: interactive use

- Remember the average calculation using the **for** loop

```
#compute average
n = int(input("How many numbers? "))
sum = 0.0
for i in range(n):
   x = float(input("Enter a number: "))
   sum = sum + x
print("The average is", sum / n)
```

user needs to enter how many items to expect before the loop start

accumulated summation stores in sum

```
How many numbers? 4
Enter a number: 45
Enter a number: 73
Enter a number: 89
Enter a number: 77
The average is 71.0
```

- Let us investigate three different other approaches using the **while** loop

while loops: interactive use

- Average calculation example- 1st approach (the long way)

```
sum = 0.0 # accumulated sum
count=0   # counter for number of loops
more_data="yes" # user's answer
while more_data=="yes":
    x = float(input("Enter a number: "))
    sum = sum + x
    count=count+1
    more_data = input("Any more number (yes,no): ")

print("The average is", sum / count)
```

```
Enter a number: 45
Any more number (yes,no): yes
Enter a number: 73
Any more number (yes,no): yes
Enter a number: 89
Any more number (yes,no): yes
Enter a number: 77
Any more number (yes,no): no
The average is 71.0
```

while loops: interactive use

- Average calculation example- 2nd approach (sentinel loop)- A sentinel loop continues to process data until reaching a special value (sentinel) that signals the end.
  - The sentinel must be distinguishable from the data since it is not processed as part of the data (such as negative number in our case)

```
sum = 0.0 # accumulated sum
count=0   # counter for number of loops
x = float(input("Enter a number (<0 quit): "))# initialization
while x>=0:       # continue as long as the data>=0
    sum = sum + x
    count=count+1
    x = float(input("Enter a number (<0 quit): "))

print("The average is", sum / count)
```

```
Enter a number (<0 quit): 45
Enter a number (<0 quit): 73
Enter a number (<0 quit): 89
Enter a number (<0 quit): 77
Enter a number (<0 quit): -1
The average is 71.0
```

while loops: interactive use

- Average calculation example- 3rd approach- (2nd approach cannot average positive and negative numbers). For general cases, the sentinel cannot be a number anymore, must be a string. It could simply be the empty string "".

```
sum = 0.0 # accumulated sum
count=0   # counter for number of loops
s = input("Enter a number (Enter to quit): ") # initialization
while s!="":    # continue as long as string s not empty
    sum = sum + float(s)     # convert s to float
    count=count+1
    s = input("Enter a number (Enter to quit): ")

print("The average is", sum / count)
```

```
Enter a number (Enter to quit): 34
Enter a number (Enter to quit): 77
Enter a number (Enter to quit): -35
Enter a number (Enter to quit): -18
Enter a number (Enter to quit):
The average is 14.5
```

loops: control statements

- Loop control statements change execution from its normal sequence.
  - **continue** statement: Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

```
text="python"
i = 0
while i<len(text):
    if text[i] == "h":
        i=i+1        # increment
        continue  # restart loop
    print("Current letter", text[i])
    i=i+1 # increment
```

```
Current letter p
Current letter y
Current letter t
Current letter o
Current letter n
```

Can also be used with **for** loop

```
text="python"
for i in range(len(text)):
    if text[i] == "h":
        continue  # restart loop
    print("Current letter", text[i])
```

loops: control statements

- **break** statement: Terminates the loop statement and transfers execution to the statement immediately following the loop.

```
text="python"
i = 0
while i<len(text):
    if text[i] == "h":
        break  # stop the loop/leave
    print("Current letter", text[i])
    i=i+1 # increment
```

```
Current letter p
Current letter y
Current letter t
```

Can also be used with **for** loop

```
text="python"
for i in range(len(text)):
    if text[i] == "h":
        break  # stop loop/leave
    print("Current letter", text[i])
```

loops: control statements

- Applications: loop and a half
  - way to avoid the priming read (initialization) in a sentinel loop
  - Example: average calculation 2nd approach revisited (exit with negative number)

```
sum = 0.0 # accumulated sum
count=0   # counter for number of loops
while True:       # always continue
    x = float(input("Enter a number (<0 quit): "))
    if x<0:
        break  # exit the loop
    sum = sum + x
    count=count+1

print("The average is", sum / count)
```

```
Enter a number (<0 quit): 45
Enter a number (<0 quit): 73
Enter a number (<0 quit): 89
Enter a number (<0 quit): 77
Enter a number (<0 quit): -1
The average is 71.0
```

loops: control statements

- Example: average calculation 3rd approach revisited (exit with empty string)

```
sum = 0.0 # accumulated sum
count=0   # counter for number of loops
while True:       # always continue
    s = input("Enter a number (Enter to quit): ")
    if s=="":
        break  # exit the loop
    sum = sum + float(s)
    count=count+1

print("The average is", sum / count)
```

```
Enter a number (Enter to quit): 34
Enter a number (Enter to quit): 77
Enter a number (Enter to quit): -35
Enter a number (Enter to quit): -18
Enter a number (Enter to quit):
The average is 14.5
```