Computer Programming in Python

Chapter 5

Repetition Structures

Chapter 5 Repetition Structures

- Repetition Structures
  - Program statements often need to be repeated
    - Example: Computing compound interest

Start with an account balance
Compute the interest amount
Add it to the balance
Compute the interest on the new balance
Add it to the balance
Compute the interest on the new balance
And so on…

Chapter 5 Repetition Structures

- Repetition Structures
  - Entire programs often need to be repeated
    - Instead of restarting the program

1. Get input from the user
2. Compute the result
3. Display the result
4. Ask the user if they would like to compute another
5. If Yes
6. Go back to Step #1
7. If NO
8. End the program

Chapter 5 Repetition Structures

- Repetition Structures (Loops)
  - Used to repeat portions of programs or entire programs
    - The statements within the loop execute until a final result has been reached and the loop ends
    - Each execution of a loop is referred to as an ***iteration*** of the loop

Loops repeat statements while a condition is true

Chapter 5 Repetition Structures

- Repetition Structures (Loops)
  - Consider the Theater Program

Sell tickets

If the Theater is not sold out

    Sell more tickets

If it is still not sold out

    Sell more tickets

And so on…

While the Theater is not sold out, continue to sell tickets

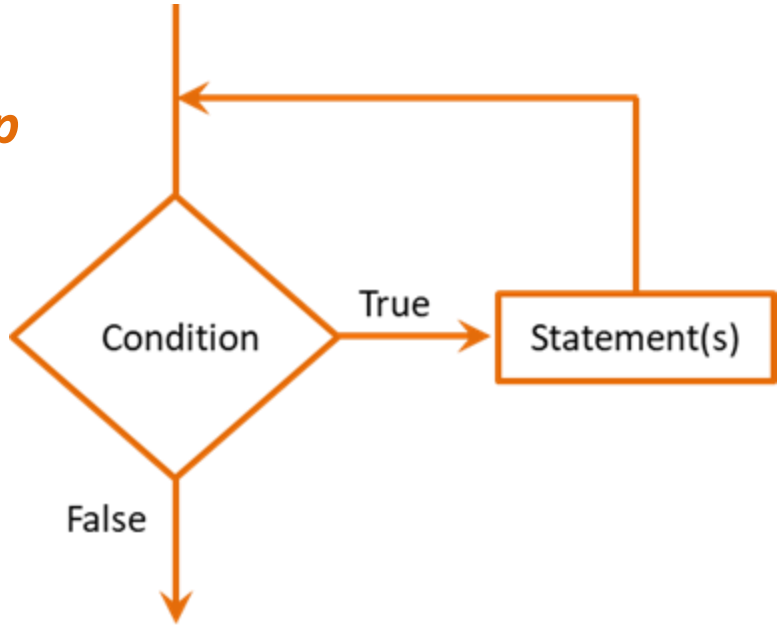
Chapter 5 Repetition Structures

- The ***While*** loop
  - A ***condition controlled*** loop
    - The statement(s) within the loop execute while some conditional expression is true
      - Therefore, the condition must eventually be false
  - Similar to conditional expressions in format

```
while condition:
    statment1
    statment2
    etc.
```

Chapter 5 Repetition Structures

- The ***While*** loop
  - Flowchart representation
    - A condition and an arrowed line going back to just prior to the conditional expression
  - The while loop is a ***pre-test loop***
    - The condition is tested first
    - It may or may not execute



Chapter 5 Repetition Structures

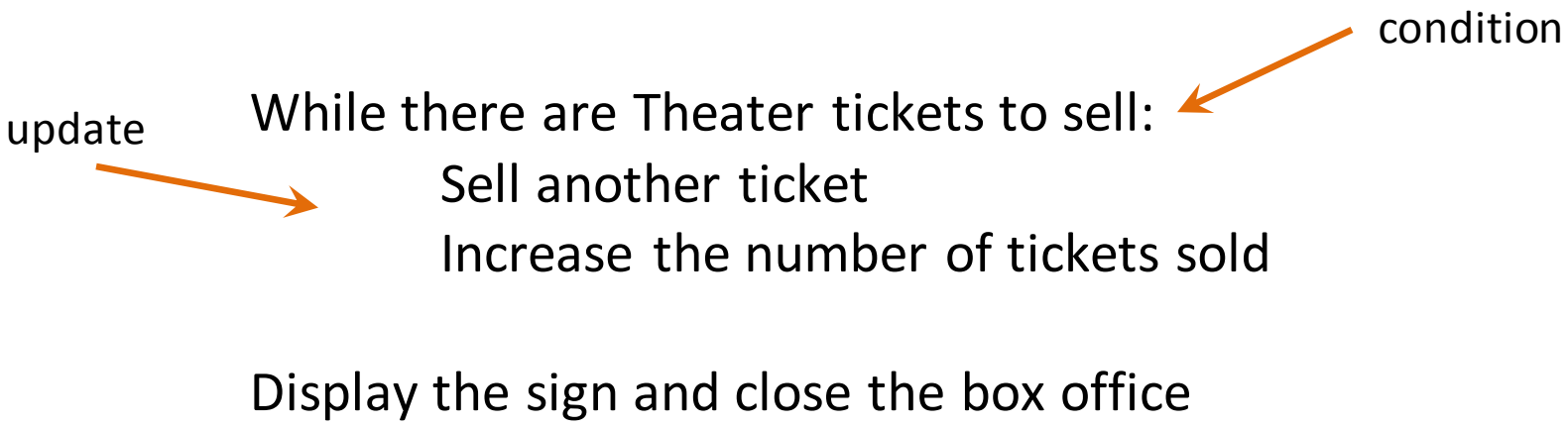
- The Theater Program Enhanced
  - Tickets should be sold until the Theater is sold out
    - Being sold out is the condition…all tickets sold

While there are Theater tickets to sell:
    Sell another ticket
    Increase the number of tickets sold

Display the sign and close the box office

Chapter 5 Repetition Structures

- The Theater Program Enhanced
  - Tickets are sold until the Theater is sold out



The update will eventually make the condition false

- The Theater Program Enhanced
  - The Theater is sold out when 400 tickets have been sold
    - Note that tickets sold is updated for each loop iteration

```
while tickets_sold < 400:
    print('Sell another ticket.')
    tickets_sold = tickets_sold + 1


print('Display "Sold Out" sign')
print('Close the box office')
```

- Assignment Statements Revisited
  - Recall - a single equal sign is the assignment operator (not equals)
  - Therefore, a variable can be on both sides of an assignment

```
total = total + grade
```

  - What the statement tells the computer:
    - Go to the memory location for *total* and find out the value that is there
    - Then go to the memory location for *grade* and find out what is there
    - Next, add the two together and place the result in the memory location for *total* (over-writing the previous value)

- Condition-controlled Loops
  - There must be something changing inside the loop to eventually make the condition false to end the loop
    - Otherwise, it will never end resulting in an ***infinite loop***

```
value = 3

while value < 10:
    print('Stuck in a loop')
```

Infinite loops never end…

Chapter 5 Repetition Structures

- Condition-controlled Loops
  - Running the entire program multiple times
    - The user controls the number of times

```
another = 'y'

while another == 'y':
    tempF = float(input('\nEnter a Fahrenheit temperature '))
    C = (tempF - 32) * 1.8
    print(tempF, 'Fahrenheit is ', C, 'Celsius\n')
    another = input('Convert another? (enter y for yes) ')


print('Have a nice day.')
```

Chapter 5 Repetition Structures

- ***Counter-controlled*** Loops
  - The number of iterations is a specific number of times
  - The programmer determines the number of times that the loop will execute when designing the loop
  - The structure in Python, is the ***for loop***
    - Designed to be used with a sequence or range of items

Chapter 5 Repetition Structures

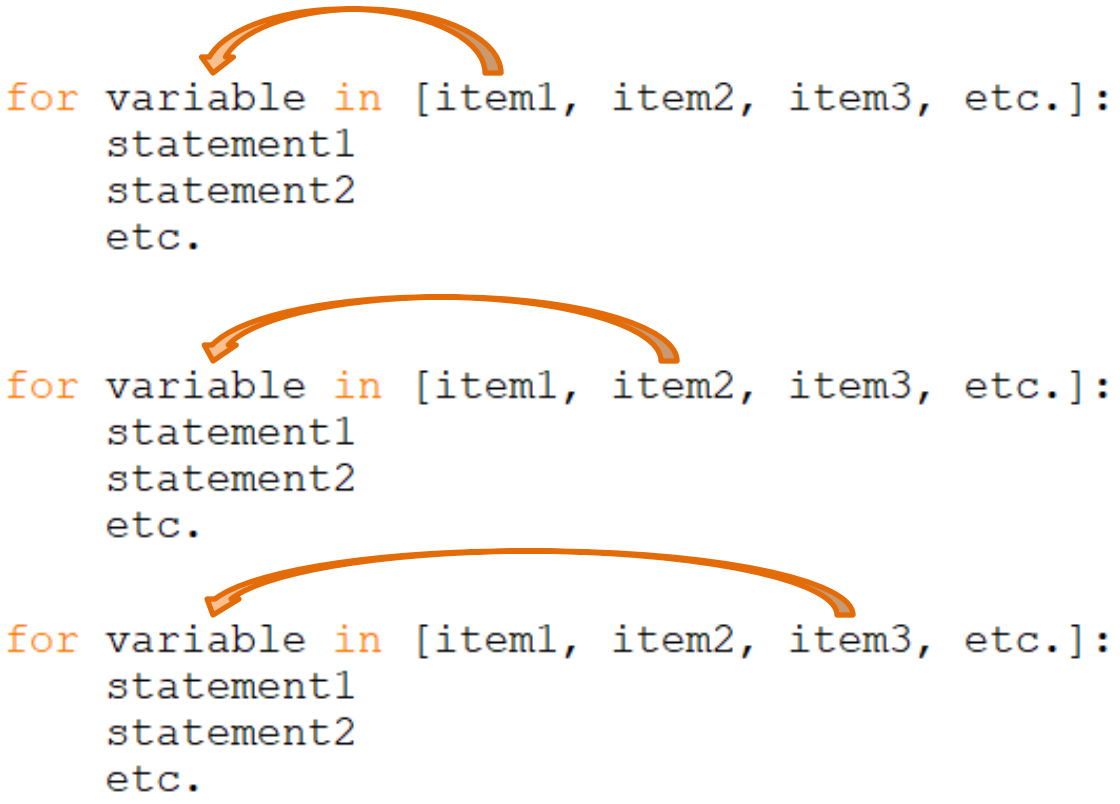
- The ***For Loop***
  - One implementation uses a sequence of items in square brackets
    - For each item in the sequence, place a copy in a variable

```
for variable in [item1, item2, item3, etc.]:
    statement1
    statement2
    etc.
```

Often referred to as the "for-in" loop

Chapter 5 Repetition Structures

- Copies a value, then executes the statements



- The ***For Loop***
  - Example: the names in the sequence are copied into the variable *person* one at a time

```
for person in ['Abe', 'Beth', 'Jermain']:
    print('Hi', person)
```

```
Hi Abe
Hi Beth
Hi Jermain
```

Chapter 5 Repetition Structures

- The ***For-in*** Loop
  - Example: the letters of a string can be accessed
    - Copied into the variable *letter* one at a time

```
for letter in 'Word':
    print(letter)
```

```
W
o
r
d
```

Chapter 5 Repetition Structures

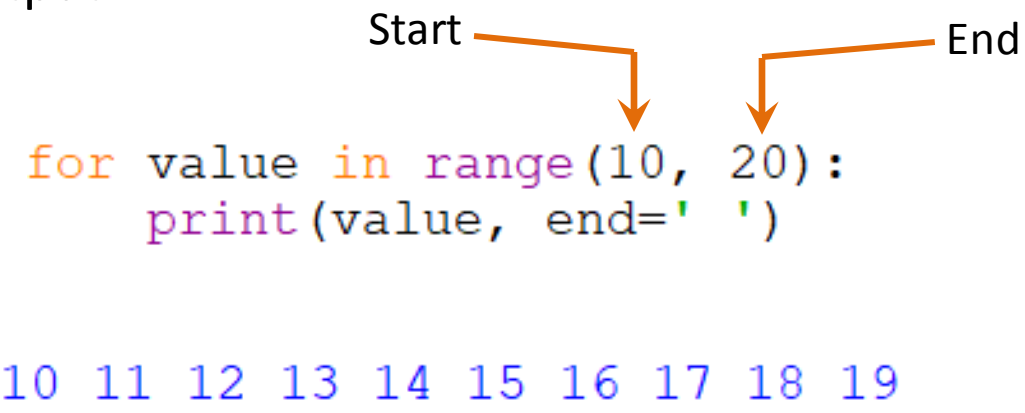
- The ***Range*** Function and For Loop
  - Can accept one, two, or three arguments
  - When one argument is passed, it uses **zero** as the start of the range, and the **argument** as the limit (not included)

```
for value in range(10):
    print(value, end=' ')
```

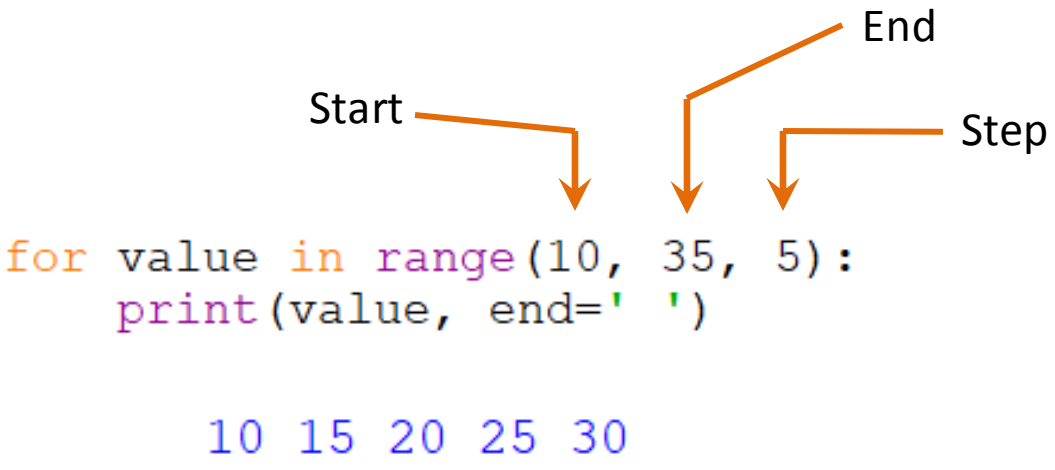
```
0 1 2 3 4 5 6 7 8 9
```

The range function can accept one, two, or three arguments

- The ***Range*** Function and For Loop
  - When two arguments are passed to *range*, they are used as the starting and ending limits of the series
    - Again, note that twenty is the limit and is not included in the output.



- The ***Range*** Function and For Loop
  - When three arguments are passed to the range function, the third argument is used as the step value for the series
    - Lines feeds have been replaced by a space



- The ***Range*** Function and For Loop
  - Example using variables:

```
start = 24
limit = 49
step = 6
for value in range(start, limit, step):
    print (value, end=' ')
```

```
24 30 36 42 48
```

- User Controlled Loops
  - Example:

```
start = int(input('Enter a starting temperature '))
end = int(input('Enter an ending temperature '))

print()
print(' Fahrenheit    Celsius')

for tempF in range(start, end):
    C = (tempF - 32) * 1.8
    print('     ', tempF, '         ', C,)

print()
print('Have a nice day.')
```

Chapter 5 Repetition Structures

- Loop ***Accumulator***
  - A variable that tallies the values as the loop iterates and contains the total when the loop finishes
  - Used when computing a running total or accumulating values

Accumulator 

```
total = 0.0

num_grades = int(input('How many grades are there? '))

for counter in range(num_grades):
    grade = float(input('Enter a grade '))
    total = total + grade

average = total/num_grades
print('The average grade is', format(average, '.1f'))
```

Chapter 5 Repetition Structures

- Loop Counters
  - When the number of iterations is unknown but needed, a ***counter variable*** is placed inside the loop
  - As an example, a program that computes the average of a set of values needs to know the number of values that were entered

Counter variables count the loop iterations

Chapter 5 Repetition Structures

- Loop Counters
  - Note the initialization of the variables in the example

```
more = 'y'
counter = 0
total = 0

while more == 'y':
    num = float(input('Enter a sales amount '))
    total = total + num
    counter = counter + 1
    more = input('Enter "y" to continue or "n" to stop.')

average = total / counter
print('The average sale is ', format(average, '.2f'))
```

accumulator

counter

Chapter 5 Repetition Structures

- Common Loop Algorithms
  - Accumulate a total
  - Compute an average
  - Validating input
    - While a user hasn't provided valid input, prompt again
  - Finding the minimum or maximum value
  - Finding a match

Chapter 5 Repetition Structures

- Common Loop Algorithms
  - Validating input
    - While a user hasn't provided valid input, prompt again

```
num = int(input('Enter a number between 1 and 10 '))

while num < 1 or num > 10:
    print('\tThat is not a valid number.')
    num = int(input('Enter a number between 1 and 10 '))


print('That is a valid number.')
```

- Common Loop Algorithms
  - Finding the minimum value

```
more = 'y'

num = int(input('Enter an integer '))
smallest = num

while more == 'y' or more == 'Y':
    num = float(input('Enter another integer '))
    if num < smallest:
        smallest = num
    more = input('Enter "y" to continue or "n" to stop.')

print('The smallest number entered was ', smallest)
```

Chapter 5 Repetition Structures

- Common Loop Algorithms
  - Finding the minimum and maximum values

```
num = int(input('Enter an integer '))
smallest = num
largest = num

while more == 'y' or more == 'Y':
    num = float(input('Enter another integer '))
    if num < smallest:
        smallest = num
    if num > largest:
        largest = num
```

Chapter 5 Repetition Structures

- Sentinels
  - A ***sentinel*** is often used to indicate the end of the input
  - The user enters a number that could not be part of the set of values
    - Request positive numbers and -1 when there are no more
    - Request test grades and 999 when finished

A sentinel is a value that intentionally outside reasonable input

Chapter 5 Repetition Structures

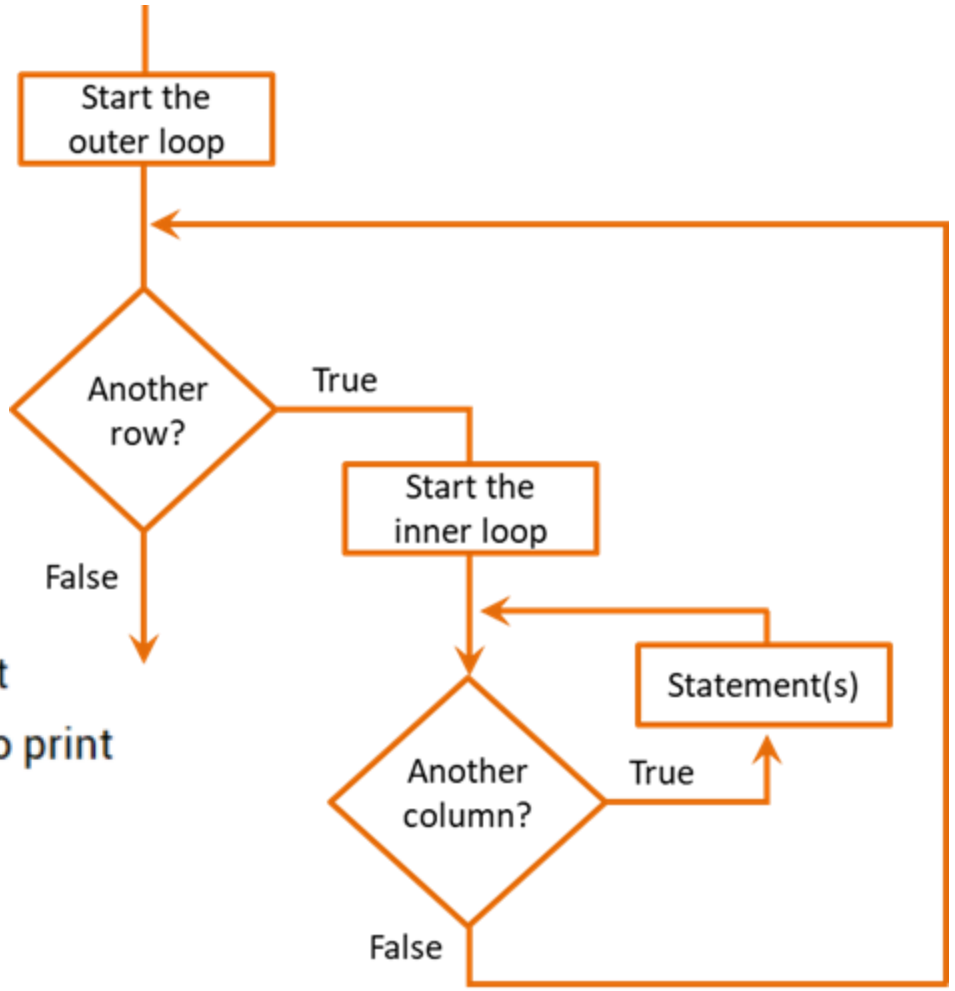
- Nested Loop
  - A loop contained inside another loop is called a ***nested loop***
  - An outer loop is entered and an inner loop executes
  - When the inner loop completes, if there are more outer loop iterations to complete, it again initiates the inner loop

```
While there is a row of values to print
    While there is a column value to print
    print the value
```

A good example for a nested loop is a set of rows and columns

Chapter 5 Repetition Structures

- Nested Loop

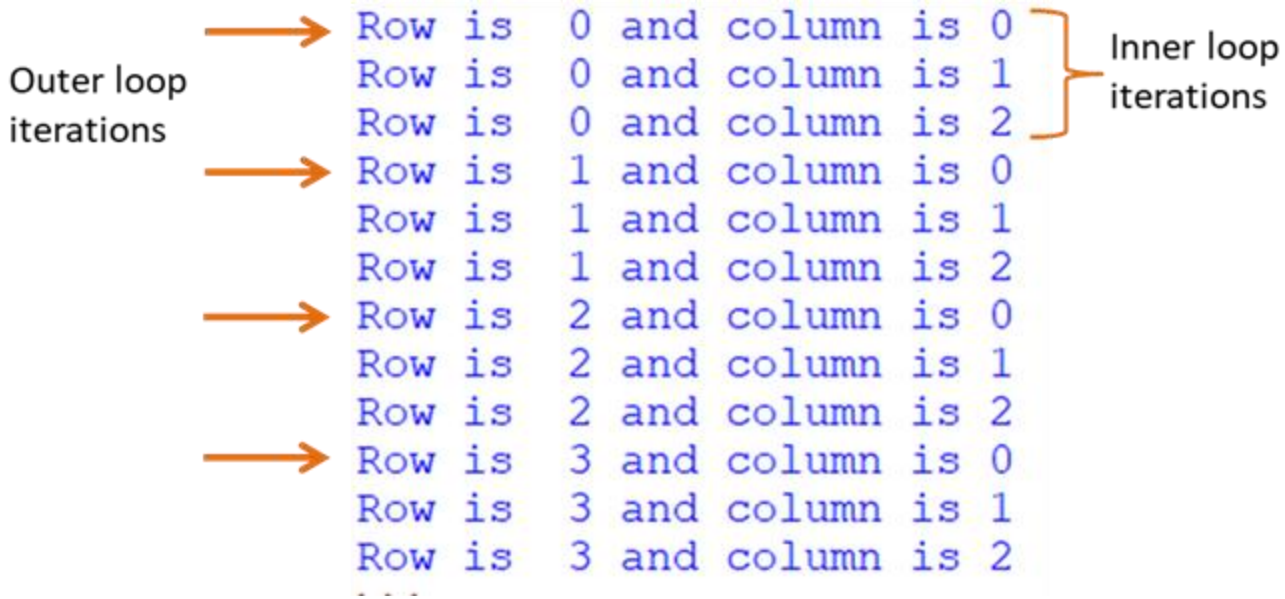


While there is a row of values to print
    While there is a column value to print
        print the value

Chapter 5 Repetition Structures

- Nested Loop Example:

```
for row in range(4):
    for column in range(3):
        print('Row is ', row, 'and column is', column)
```



Chapter 5 Repetition Structures

- Common Loop Errors
  - Off-by-one errors
    - The conditional expression or range was written incorrectly
    - The loop is executing one too many or one too few times
    - Easily corrected after running and testing the program
  - Confusing what should be inside the loop and what should be outside the loop
    - Testing and adding print statements can help to debug these

A Complete Example

- ***Complete Example***
  - Requirements:

    Write a program for a Financial Adviser that computes the number of years to double a $10,000 investment at an annual interest rate input by the user
  - Requirements Decomposition
    - Pseudocode, Flowchart, both?

- ***Complete Example***

Program Pseudocode:

Step 1 Prompt for the interest rate

Step 2 Compute the interest on the balance

Step 3 Add the interest to the balance

Step 4 Increment the number of years

Step 5 Is the balance < $20,000.00

Yes, go back to Step 2

No, got to Step 6

Step 6 Display the number of years

- ***Complete Example***
  - Verbalizing and walking through the steps that the program will take is referred to as ***Story-boarding***
  - Can often help with determining the sequence and order of operations for the program
    - What should (or must) happen first
    - What happens second
    - And so on…

- ***Complete Example - Story-boarding***

"Set up the program by initializing the balance and years, and obtain the interest rate from the user."

"While the balance is less than $20,000.00, compute the interest on the balance and add it to the previous balance".

"When the balance is no longer less than $20,000, display the number of years".

- ***Complete Example - Design***
  - Variables
    - Balance, float since it is in dollars
      - Begins at $10,000.00
      - Is updated when the interest amount is computed
    - Years, integer to store years
      - Begins at zero
      - Is updated for each computation
    - Interest rate, float since it may have a decimal
      - Is input by the user

- ***Complete Example - Design***
  - The balance and years must be initialized
  - The interest rate is obtained from the user
  - As long as (*while*) the balance is less than the target amount
    - Compute the interest
    - Add it to the balance
    - Increment the years
  - Display the result in years

- ***Complete Example - Development***
  - *Print statement within the loop added for testing*

```
balance = 10000
years = 0

int_rate = float(input('Enter the interest rate : '))

while balance < 20000:
    interest = balance * (int_rate / 100)
    balance = balance + interest
    years = years + 1
    print('balance is $', format(balance, ',.2f'))


print('The balance doubled in ', years, 'years')
```

- ***Complete Example - Testing***
  - *The print statements help with validating the algorithm*

```
Enter the interest rate : 4.5
balance is $ 10,450.00
balance is $ 10,920.25
balance is $ 11,411.66
balance is $ 11,925.19
balance is $ 12,461.82
balance is $ 13,022.60
balance is $ 13,608.62
balance is $ 14,221.01
balance is $ 14,860.95
balance is $ 15,529.69
balance is $ 16,228.53
balance is $ 16,958.81
balance is $ 17,721.96
balance is $ 18,519.45
balance is $ 19,352.82
balance is $ 20,223.70
The balance doubled in  16 years
```

- Testing and Debugging
  - Verifies the program for accuracy
  - Can also surface questions about the requirements
    - The output states that the balance doubled in 16 years, but the balance is actually more than double the initial amount at that point
    - Since interest is being computed and added annually, the program meets the requirements
    - But it might be a good idea to ask the Financial Advisor if this is what she had in mind

Another Example

Chapter 5 Repetition Structures

- Program Requirements:

  Pilots simulate engine failure for emergency training purposes. Write a program that computes the altimeter reading and distance traveled for a single-engine plane that is gliding due to simulated engine failure.

  Input: altitude of the plane (600 – 5,000 ft.)

  Output: altimeter reading every ten (10) seconds, and distance traveled each interval in nautical miles

Chapter 5 Repetition Structures

- Program Requirements:

  Gliding descent rate: 1,000 ft./minute

  Glide speed: 60 knots (nautical miles per hour)

  Glide distance: 1.6 nautical miles (6,076 ft.) per 1000 ft. of altitude

  Design consideration:

  Compute every second and only display every 10th second?

  Compute for the ten second interval and reset at the 6th?

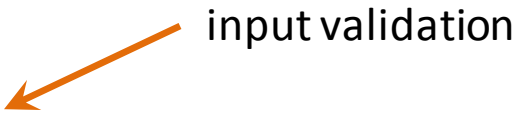
Chapter 5 Repetition Structures

- Program Setup
  - Variables
    - Things that are input
    - Things that are changing and being displayed

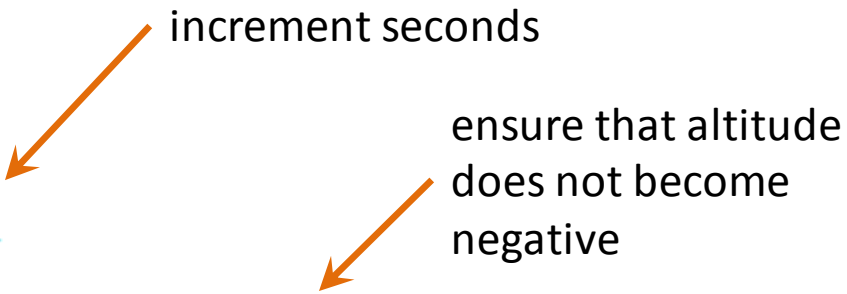
Input:	altitude of the plane (600 – 5,000 ft.)
Output:	altimeter reading (altitude)
	time elapsed every ten (10) seconds
	distance traveled in NM

- Program Setup
  - Variables and setup using 1 second intervals

```
altitude = 0
distance_per_sec = 1.6/60
distance = 0
descent_per_sec = 1000/60
seconds = 0
minutes = 0

while altitude <= 600 or altitude > 5000:
    altitude = float(input('Enter an altitude (600 - 5,000): '))


while altitude > 0:
```

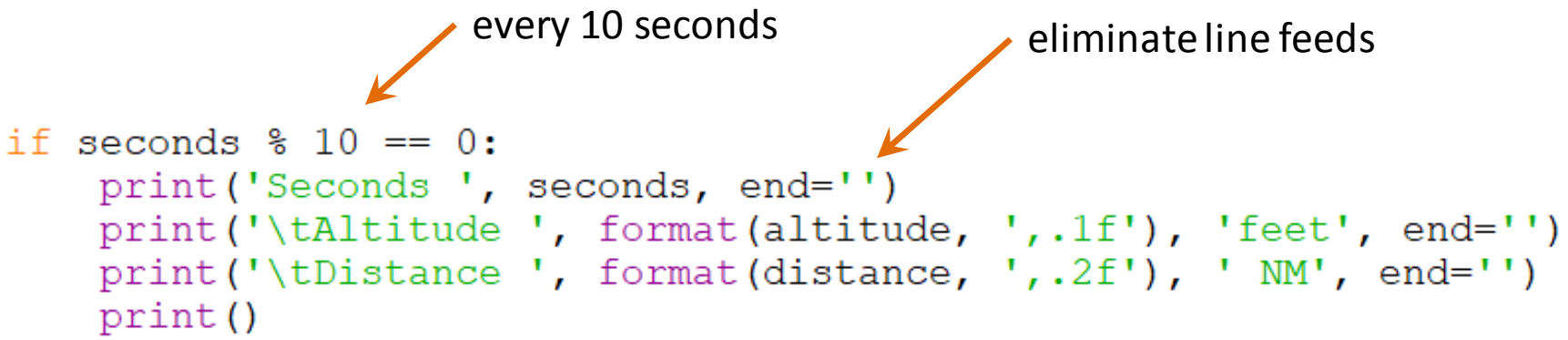


- Program Setup
  - Computations



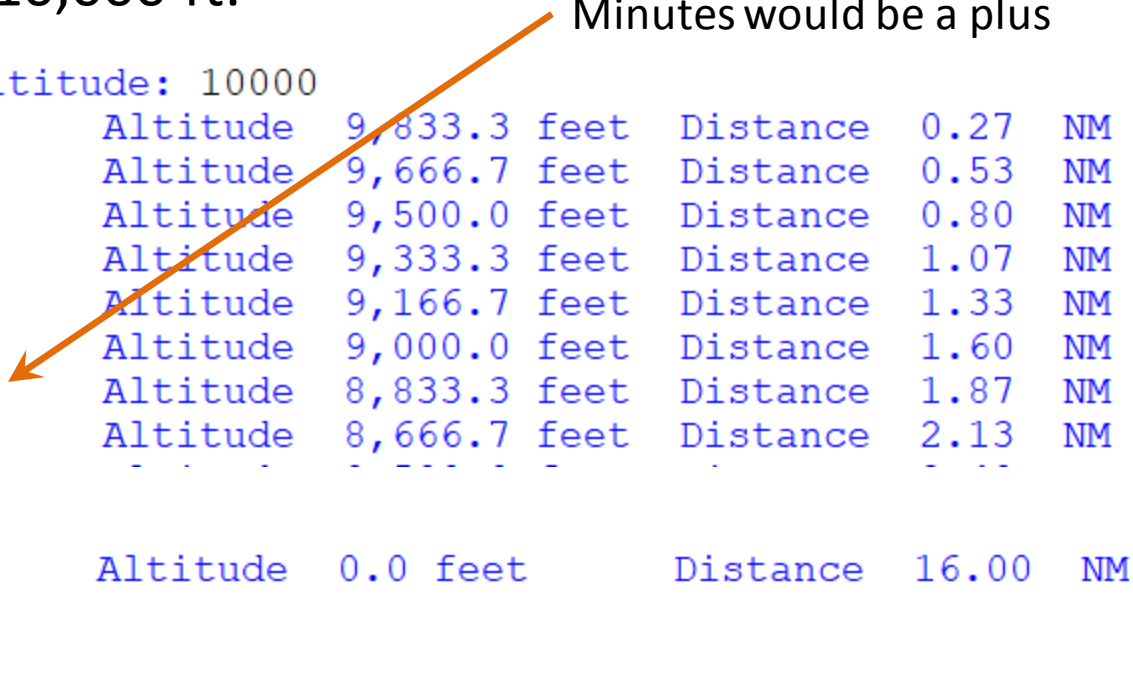

```
while altitude > 0:

    seconds = seconds + 1

    if altitude - descent_per_sec > 0:
        distance = distance + distance_per_sec
        altitude = altitude - descent_per_sec
    else:
        altitude = 0
```

- Program Setup
  - Add the output and test the program



Chapter 5 Repetition Structures

- Program Setup
  - Test with a known value
  - 16 NM at 10,000 ft.



Chapter 5 Repetition Structures

- Program Development/Test
  - When testing, note other aspects of the program
  - Very often testing surfaces changes that would enhance the user experience
  - Stay away from the “Good enough” trap
  - So, when is it done?

Stay away from the “Good enough” trap

Chapter 5 Repetition Structures

- Program Development/Test
  - modify and test

add minutes and reset seconds

```
seconds = seconds + 1

if seconds % 60 == 0:
    minutes = minutes + 1
    seconds = 0

if seconds % 10 == 0:
    print('Time elapsed ', format(minutes,'2'), ':', seconds, sep='', end='')
    print('\tAltitude ', format(altitude, ',.1f'), 'feet', end='')
    print('\tDistance ', format(distance, ',.2f'), ' NM', end='')
    print()
```

- Program Development/Test
  - Another issue surfaces

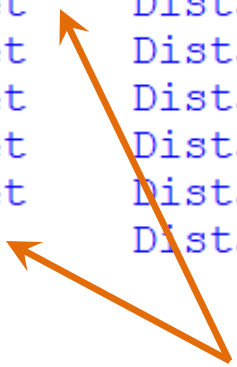
```
Enter the altitude: 10000
Time elapsed  0:10      Altitude  9,833.3 feet
Time elapsed  0:20      Altitude  9,666.7 feet
Time elapsed  0:30      Altitude  9,500.0 feet
Time elapsed  0:40      Altitude  9,333.3 feet
Time elapsed  0:50      Altitude  9,166.7 feet
Time elapsed  1:0       Altitude  9,000.0 feet
Time elapsed  1:10      Altitude  8,833.3 feet
Time elapsed  1:20      Altitude  8,666.7 feet
Time elapsed  1:30      Altitude  8,500.0 feet
Time elapsed  1:40      Altitude  8,333.3 feet
Time elapsed  1:50      Altitude  8,166.7 feet
Time elapsed  2:0       Altitude  8,000.0 feet
```

- Program Development/Test
  - One solution is to build the time as a string before the output

```
if seconds == 0:
    sec = '00'
else:
    sec = str(seconds)

time = str(format(minutes,'2') + ':' + sec)

if seconds % 10 == 0:
    print('Time elapsed ', time, end='')
```

- Program Development/Test
  - Another alignment issue

```
Altitude  1,500.0 feet  Distance  3.20  NM
Altitude  1,333.3 feet  Distance  3.47  NM
Altitude  1,166.7 feet  Distance  3.73  NM
Altitude  1,000.0 feet  Distance  4.00  NM
Altitude  833.3 feet    Distance  4.27  NM
Altitude  666.7 feet    Distance  4.53  NM
Altitude  500.0 feet    Distance  4.80  NM
Altitude  333.3 feet    Distance  5.07  NM
Altitude  166.7 feet    Distance  5.33  NM
Altitude  0.0 feet      Distance  5.60  NM
```



alignment concern

Chapter 5 Repetition Structures

- Program Development/Test
  - Easy fix

```
print('\tAltitude ', format(altitude, '7,.1f'), 'feet', end=''
print('\tDistance ', format(distance, ',.2f'), ' NM', end='')
print()
```

```
Time elapsed   2:20     Altitude  1,166.7 feet  Distance  3.73  NM
Time elapsed   2:30     Altitude  1,000.0 feet  Distance  4.00  NM
Time elapsed   2:40     Altitude    833.3 feet  Distance  4.27  NM
Time elapsed   2:50     Altitude    666.7 feet  Distance  4.53  NM
Time elapsed   3:00     Altitude    500.0 feet  Distance  4.80  NM
Time elapsed   3:10     Altitude    333.3 feet  Distance  5.07  NM
Time elapsed   3:20     Altitude    166.7 feet  Distance  5.33  NM
Time elapsed   3:30     Altitude      0.0 feet  Distance  5.60  NM
Completed.
>>>
```

Chapter 5 Repetition Structures

- Program Design
  - Repetitive tasks in a program are common
    - Decide on variables beforehand
    - Determine the loop condition carefully
    - Consider what should go inside and what should go outside the loop
    - Test with known values
    - Critique the output
    - Modify, test, modify, test,…

Chapter 5 Repetition Structures