Code Academy

Chapter 4: Loops (4.5 – 4.6)

PART2: COMMON LOOP ALGORITHMS, THE FOR STATEMENT

Chapter 3-Part2:

- **Goals**
  - To become familiar with common loop algorithms

- **Contents:**
- Common Loop Algorithms
- The **for** loop

Code Academy

Common Loop Algorithms

1. Sum and Average Value
2. Counting Matches
3. Prompting until a Match Is Found
4. Maximum and Minimum
5. Comparing Adjacent Values

Average Example

Average of Values

- First total the values
- Initialize count to 0
  - Increment per input
- Check for count 0
  - Before divide!

```
total = 0.0
count = 0
inputStr = input("Enter value: ")
while inputStr != "" :
   value = float(inputStr)
   total = total + value
   count = count + 1
   inputStr = input("Enter value: ")

if count > 0 :
   average = total / count
else :
   average = 0.0
```

Sum Example

- Sum of Values
  - Initialize total to 0
  - Use while loop with sentinel

```
total = 0.0
inputStr = input("Enter value: ")
while inputStr != "" :
   value = float(inputStr)
   total = total + value
   inputStr = input("Enter value: ")
```

Counting Matches (e.g., Negative Numbers)

- Counting Matches
  - Initialize negatives to 0
  - Use a while loop
  - Add to negatives per match

```
negatives = 0
inputStr = input("Enter value: ")
while inputStr != "“ :
   value = int(inputStr)
   if value < 0 :
      negatives = negatives + 1
   inputStr = input("Enter value: ")

print("There were", negatives,
"negative values.")
```

Prompt Until a Match is Found

- Initialize boolean flag to False
- Test sentinel in while loop
  - Get input, and compare to range
    - If input is in range, change flag to True
    - Loop will stop executing

```
valid = False
while not valid :
   value = int(input("Please enter a positive value < 100: "))
   if value > 0 and value < 100 :
      valid = True
   else :
      print("Invalid input.")
```

This is an excellent way to validate provided inputs

Program will loop until a valid value is entered

Maximum

- Get first input value
  - By definition, this is the largest that you have seen so far
- Loop while you have a valid number (non-sentinel)
  - Get another input value
  - Compare new input to largest (or smallest)
  - Update largest if necessary

```
largest = int(input("Enter a value: "))
inputStr = input("Enter a value: ")
while inputStr != "" :
   value = int(inputStr)
   if value > largest :
      largest = value
   inputStr = input("Enter a value: ")
```

Minimum

- Get first input value
  - This is the smallest that you have seen so far!
- Loop while you have a valid number (non-sentinel)
  - Get another input value
  - Compare new input to largest (or smallest)
  - Update smallest if necessary

```
smallest = int(input("Enter a value: "))
inputStr = input("Enter a value: ")
while inputStr != "" :
   value = int(inputStr)
   if value < smallest :
      smallest = value
   inputStr = input("Enter a value: ")
```

Comparing Adjacent Values

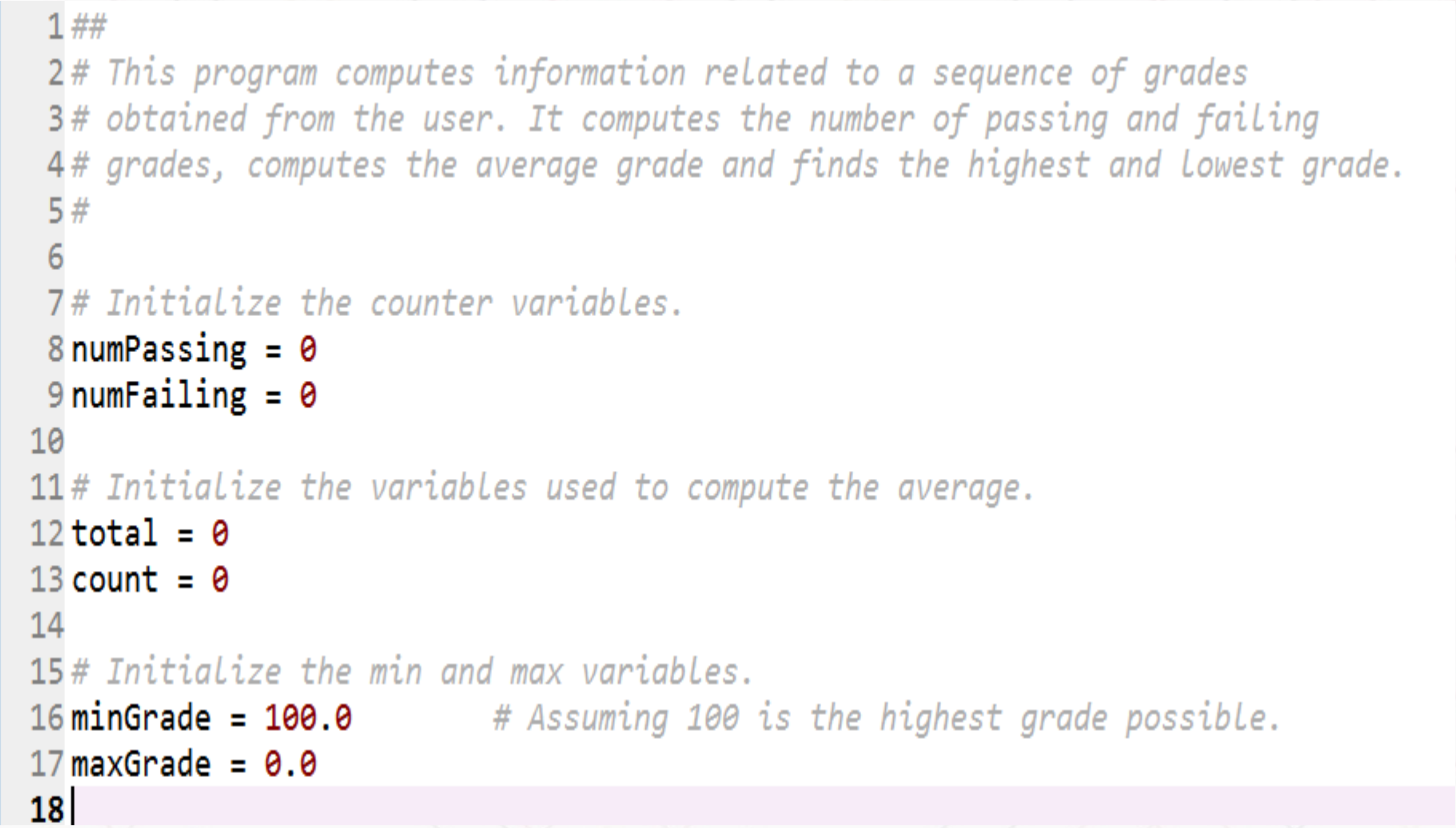
- Get first input value
- Use while to determine if there are more to check
  - Copy input to previous variable
  - Get next value into input variable
  - Compare input to previous, and output if same

```
value = int(input("Enter a value: "))
inputStr = input("Enter a value: ")
while inputStr != "“ :
   previous = value
   value = int(inputStr)
   if value == previous :
      print("Duplicate input")
   inputStr = input("Enter a value: ")
```

Grades Example

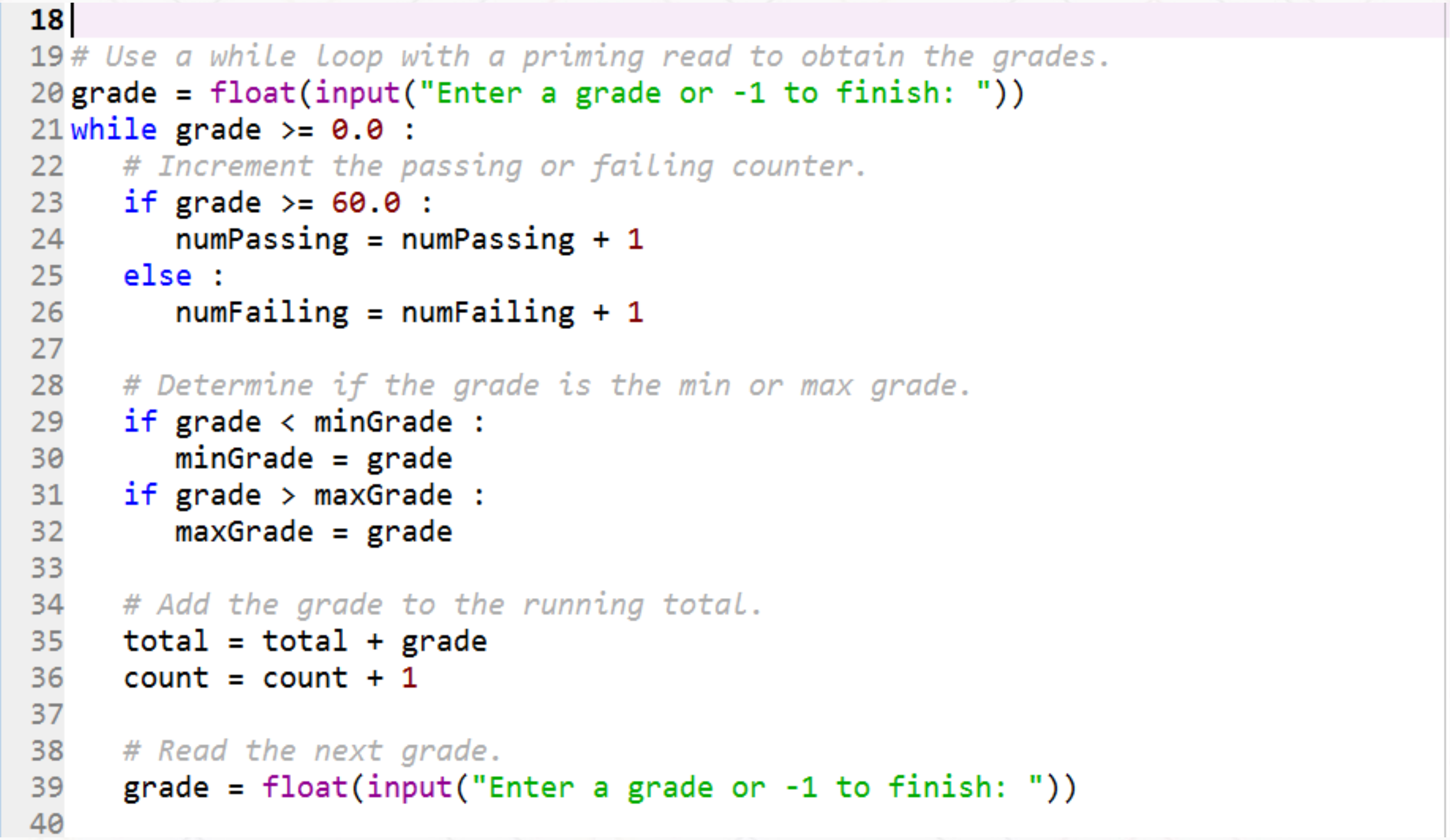
- Examine the file: Grades.py
- Look carefully at the source code.
- The maximum possible score is read as user input
  - There is a loop to validate the input
- The passing grade is computed as 60% of the available points

Grades.py (1)



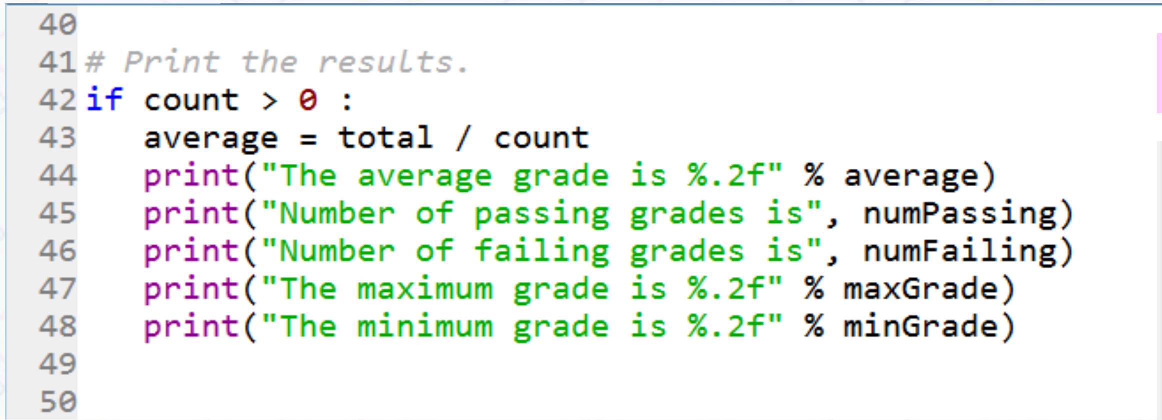
```
##
# This program computes information related to a sequence of grades
# obtained from the user. It computes the number of passing and failing
# grades, computes the average grade and finds the highest and lowest grade.
#

# Initialize the counter variables.
numPassing = 0
numFailing = 0

# Initialize the variables used to compute the average.
total = 0
count = 0

# Initialize the min and max variables.
minGrade = 100.0        # Assuming 100 is the highest grade possible.
maxGrade = 0.0

```

Grades.py (2)



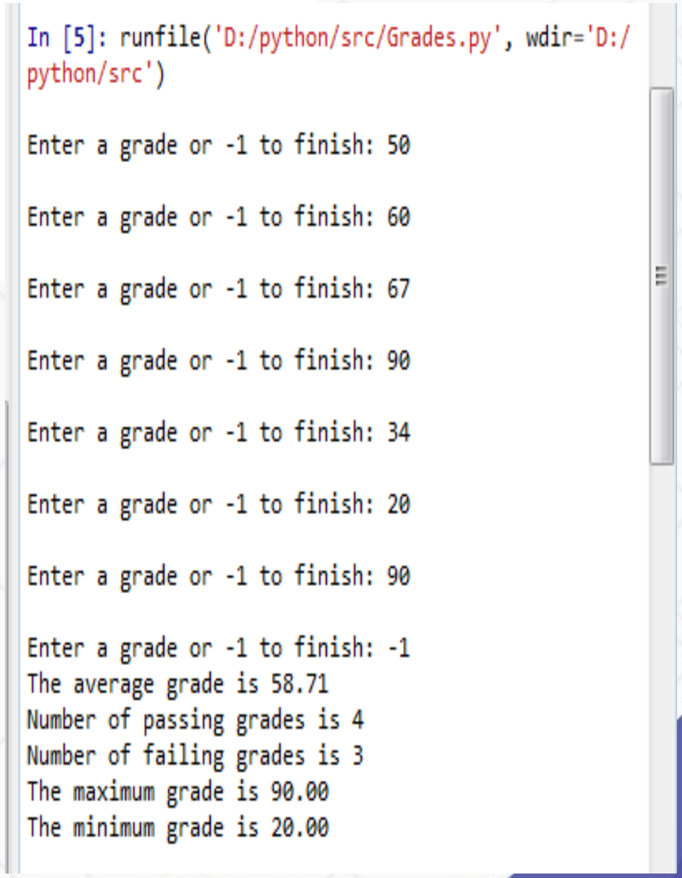
```python

# Use a while loop with a priming read to obtain the grades.
grade = float(input("Enter a grade or -1 to finish: "))
while grade >= 0.0 :
   # Increment the passing or failing counter.
   if grade >= 60.0 :
      numPassing = numPassing + 1
   else :
      numFailing = numFailing + 1

   # Determine if the grade is the min or max grade.
   if grade < minGrade :
      minGrade = grade
   if grade > maxGrade :
      maxGrade = grade

   # Add the grade to the running total.
   total = total + grade
   count = count + 1

   # Read the next grade.
   grade = float(input("Enter a grade or -1 to finish: "))

```

Grades.py (3)



```python

# Print the results.
if count > 0 :
    average = total / count
    print("The average grade is %.2f" % average)
    print("Number of passing grades is", numPassing)
    print("Number of failing grades is", numFailing)
    print("The maximum grade is %.2f" % maxGrade)
    print("The minimum grade is %.2f" % minGrade)


```

Program run:



```
In [5]: runfile('D:/python/src/Grades.py', wdir='D:/
python/src')

Enter a grade or -1 to finish: 50

Enter a grade or -1 to finish: 60

Enter a grade or -1 to finish: 67

Enter a grade or -1 to finish: 90

Enter a grade or -1 to finish: 34

Enter a grade or -1 to finish: 20

Enter a grade or -1 to finish: 90

Enter a grade or -1 to finish: -1
The average grade is 58.71
Number of passing grades is 4
Number of failing grades is 3
The maximum grade is 90.00
The minimum grade is 20.00
```

The `for` Loop

- Uses of a **`for`** loop:
  - The **`for`** loop can be used to iterate over the contents of any **container**.
  - A **container** is is an object (Like a **string**) that contains or stores a collection of elements
  - A **string** is a container that stores the collection of characters in the string

An Example of a `for` Loop

- Notice the difference between the while loop and the for loop in the example.
- In the while loop, the *index variable* **`i`** is assigned 0, 1, and so on.
- In the for loop, the *variable* ***letter*** is assigned stateName[0], stateName[1], and so on.

```
stateName = "Virginia"
i = 0
while i < len(stateName) :
   letter = stateName[i]
   print(letter)
   i = i + 1
```

while version

```
stateName = "Virginia"
for letter in stateName :
   print(letter)
```

for version

The for Loop (2)

- Uses of a for loop:
  - A for loop can also be used as a count-controlled loop that iterates over a range of integer values.

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

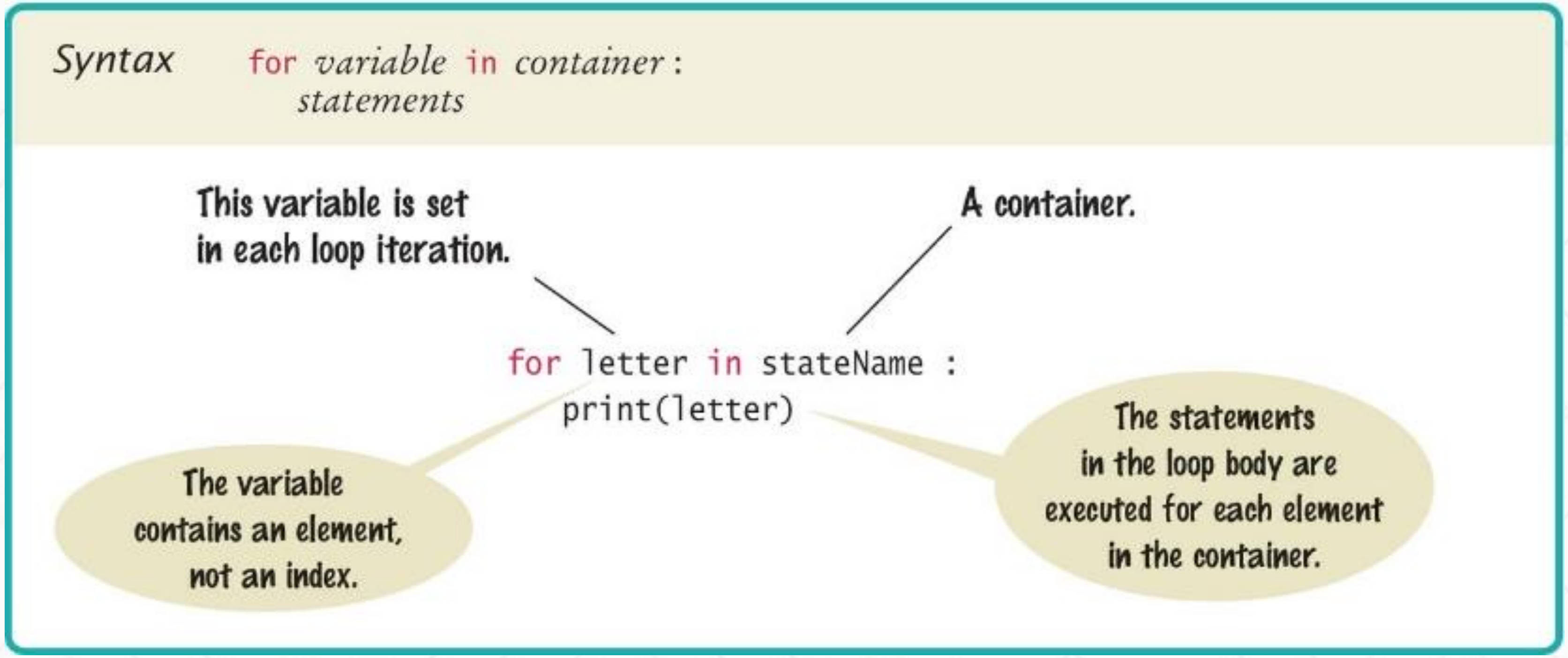
while version

```
for i in range(1, 10) :
   print(i)
```

for version

Syntax of a **for** Statement (Container)

- Using a for loop to iterate over the contents of a container, an element at a time.



Syntax

```
for variable in container :
    statements
```

This variable is set in each loop iteration.

A container.

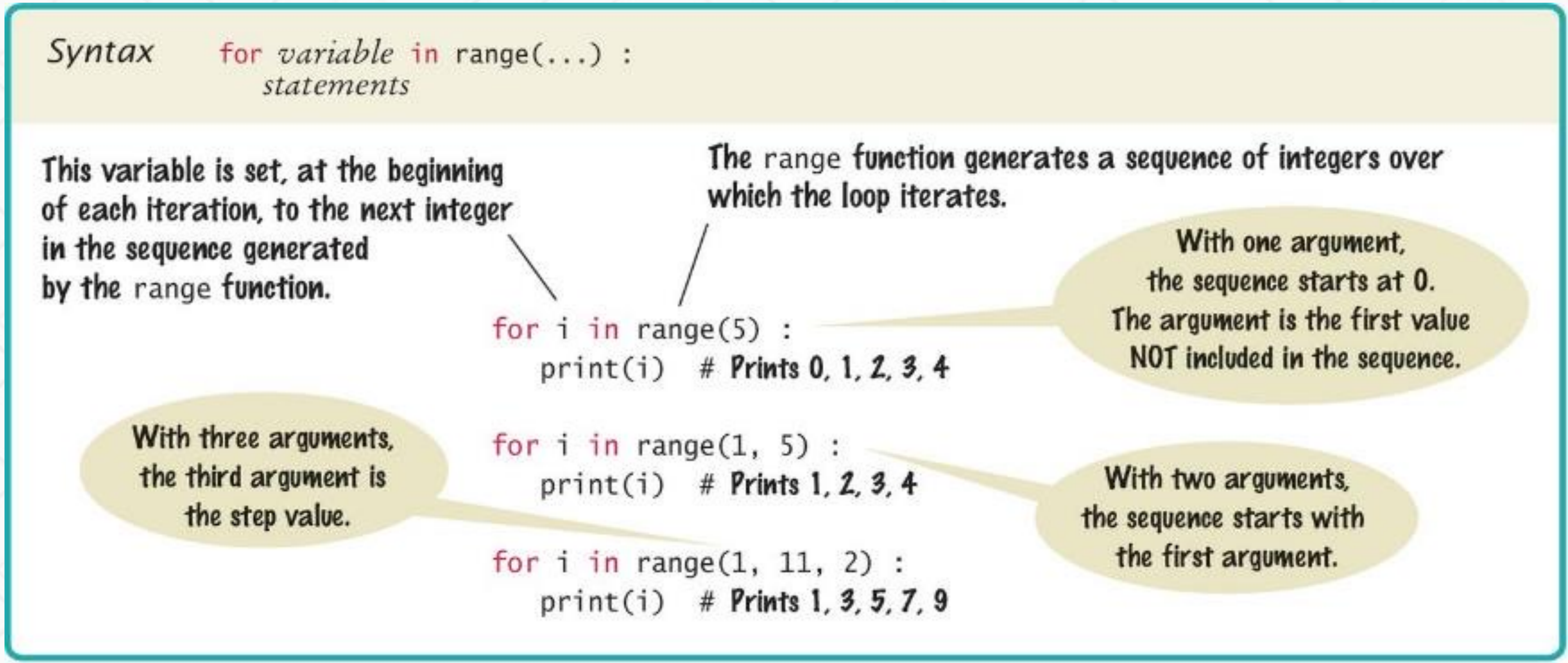
```
for letter in stateName :
    print(letter)
```

The variable contains an element, not an index.

The statements in the loop body are executed for each element in the container.

Syntax of a **for** Statement (Range)

- You can use a for loop as a count-controlled loop to iterate over a range of integer values
- We use the range function for generating a sequence of integers that less than the argument that can be used with the for loop



Syntax

```
for variable in range(...) :
   statements
```

This variable is set, at the beginning of each iteration, to the next integer in the sequence generated by the range function.

The range function generates a sequence of integers over which the loop iterates.

```
for i in range(5) :
   print(i)   # Prints 0, 1, 2, 3, 4
```

With one argument, the sequence starts at 0. The argument is the first value NOT included in the sequence.

```
for i in range(1, 5) :
   print(i)   # Prints 1, 2, 3, 4
```

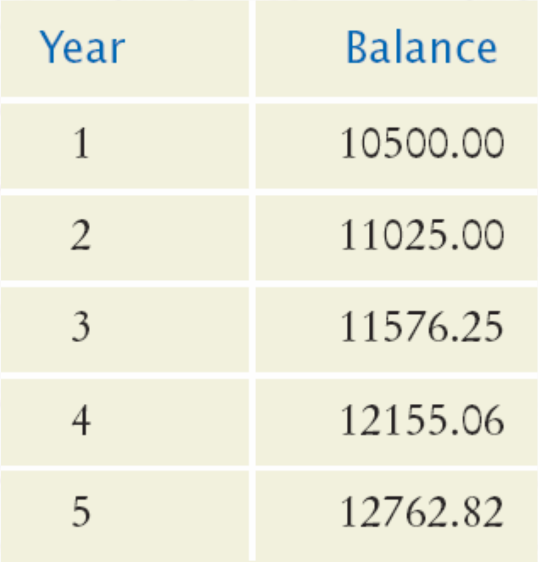
With two arguments, the sequence starts with the first argument.

```
for i in range(1, 11, 2) :
   print(i)   # Prints 1, 3, 5, 7, 9
```

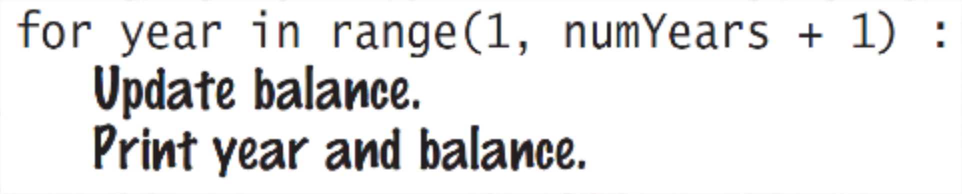
With three arguments, the third argument is the step value.

Planning a **for** Loop

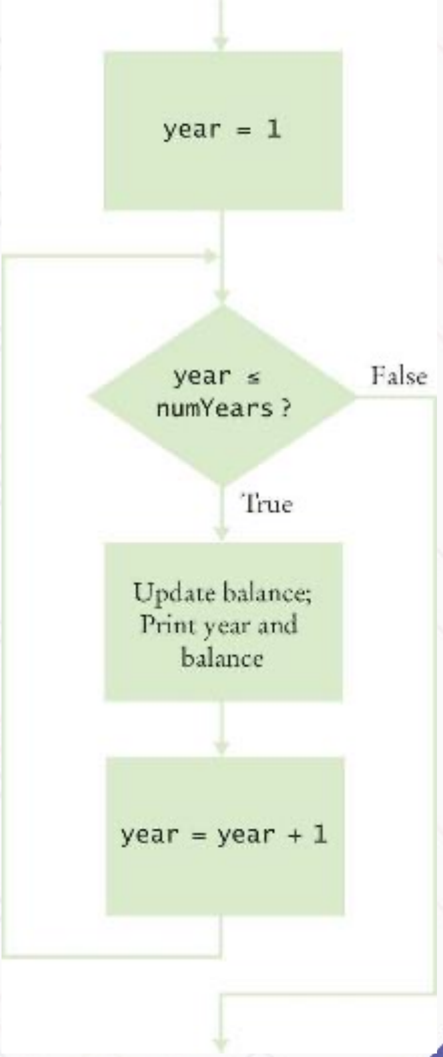
- Print the balance at the end of each year for a number of years



Year	Balance
1	10500.00
2	11025.00
3	11576.25
4	12155.06
5	12762.82



```
for year in range(1, numYears + 1) :
    Update balance.
    Print year and balance.
```



Good Examples of **for** Loops

- Keep the loops simple!

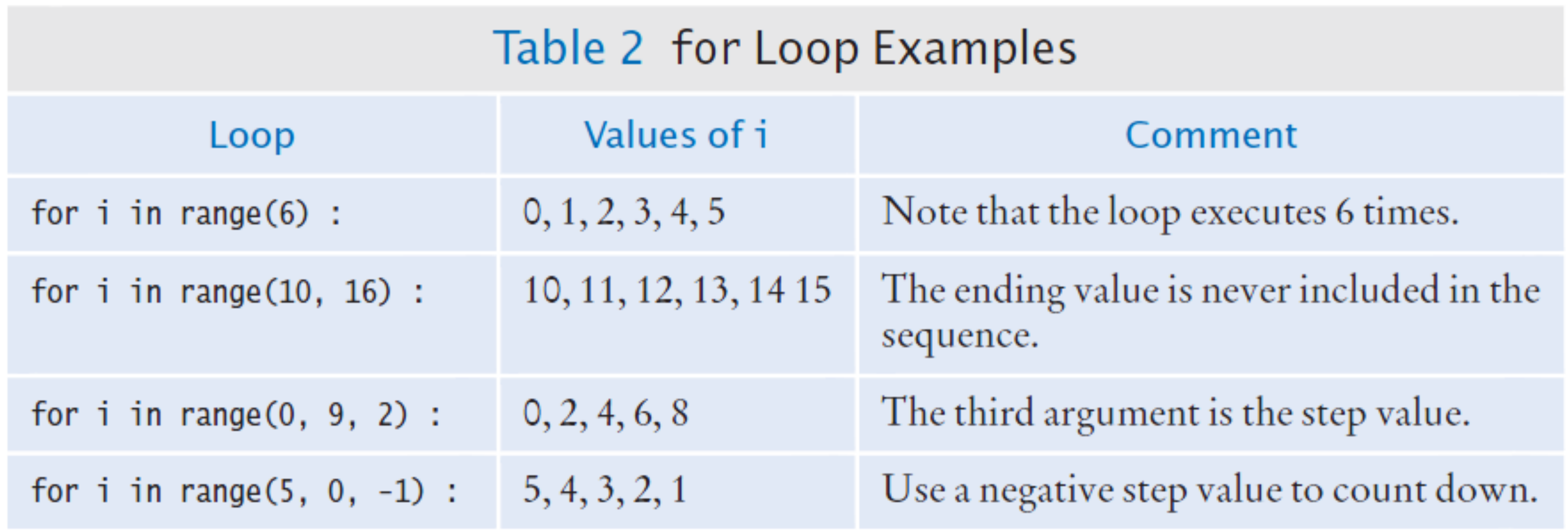
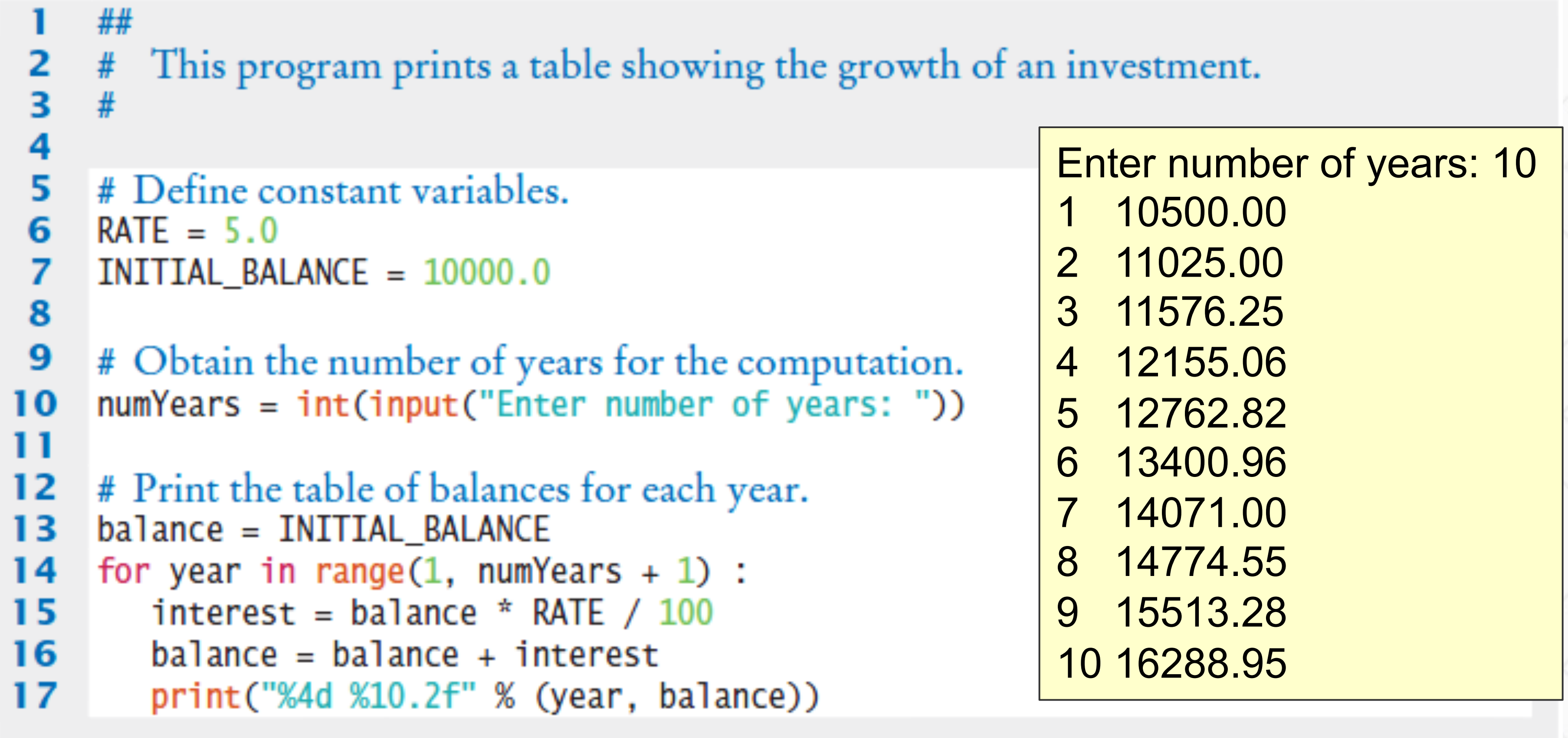


Table 2 for Loop Examples

Loop	Values of i	Comment
`for i in range(6) :`	0, 1, 2, 3, 4, 5	Note that the loop executes 6 times.
`for i in range(10, 16) :`	10, 11, 12, 13, 14 15	The ending value is never included in the sequence.
`for i in range(0, 9, 2) :`	0, 2, 4, 6, 8	The third argument is the step value.
`for i in range(5, 0, -1) :`	5, 4, 3, 2, 1	Use a negative step value to count down.

Investment Example



```python
##
#   This program prints a table showing the growth of an investment.
#

# Define constant variables.
RATE = 5.0
INITIAL_BALANCE = 10000.0

# Obtain the number of years for the computation.
numYears = int(input("Enter number of years: "))

# Print the table of balances for each year.
balance = INITIAL_BALANCE
for year in range(1, numYears + 1) :
   interest = balance * RATE / 100
   balance = balance + interest
   print("%4d %10.2f" % (year, balance))
```

```
Enter number of years: 10
1   10500.00
2   11025.00
3   11576.25
4   12155.06
5   12762.82
6   13400.96
7   14071.00
8   14774.55
9   15513.28
10 16288.95
```

Programming Tip

- Finding the correct lower and upper bounds for a loop can be confusing.
  - Should you start at 0 or at 1?
  - Should you use <= b or < b as a termination condition?
- Counting is easier for loops with asymmetric bounds.
  - The following loops are executed b - a times.

```
int i = a
while i < b :
   . . .
   i = i + 1
```

```
for i in range(a, b) :
   . . .
```

Programming Tip

- The loop with symmetric bounds (“<=”, is executed b - a + 1 times.
  - That “+1” is the source of many programming errors.

```
i = a
while i <= b :
   . . .
   i = i + 1
```

```
# For  this version of the loop the ‘+1’ is
# very noticeable!. You must specify an upper
# bound that is one more than the last value
# to be included in the range.

for year in range(1, numYears + 1) :
```

Steps to Writing a Loop

- Planning:
  - Decide what work to do inside the loop
  - Specify the loop condition
  - Determine loop type
  - Setup variables before the first loop
  - Process results when the loop is finished
  - Trace the loop with typical examples
- Coding:
  - Implement the loop in Python

A Special Form of the **`print`** Function

- Python provides a special form of the print function that does not start a new line after the arguments are displayed
- This is used when we want to print items on the same line using multiple print statements
- For example the two statements:

```
print(“00”, end=””)
print(3 + 4)
```

- Produce the output:

```
007
```

- Including **end=“”** as the last argument to the print function prints an empty string after the arguments, instead on a new line
- The output of the next **print** function starts on the same line

Summary of the **for** Loop

- **for** loops are very powerful
- The **for** loop can be used to iterate over the contents of any container, which is an object that contains or stores a collection of elements
  - a string is a container that stores the collection of characters in the string.
- A **for** loop can also be used as a count-controlled loop that iterates over a range of integer values.

Summary: Two Types of Loops

- **while** Loops
- **for** Loops
- **while** loops are very commonly used (general purpose)
- Uses of the **for** loop:
  - The **for** loop can be used to iterate over the contents of any container.
  - A **for** loop can also be used as a count-controlled loop that iterates over a range of integer values.

Summary

- Each loop requires the following steps:
  - Initialization (setup variables to start looping)
  - Condition (test if we should execute loop body)
  - Update (change something each time through)
- A loop executes instructions repeatedly while a condition is True.
- An off-by-one error is a common error when programming loops.
  - Think through simple test cases to avoid this type of error.