MEIM | MASTER IN ENTREPRENEURSHIP INNOVATION MANAGEMENT IN COLLABORATION WITH MIT SLOAN

IN COLLABORATION WITH MIT MANAGEMENT SLOAN SCHOOL | UNIVERSITÀ DEGLI STUDI DI NAPOLI PARTHENOPE

MASTER MEIM 2021-2022

Python Programming Course
Lesson 4

Iterative Statements

Lesson given by prof. Mariacarla Staffa

Prof. Computer Science at the University of Naples Parthenope

Chapter Goals

- To implement while and for loops
- To hand-trace the execution of a program
- To become familiar with common loop algorithms
- To understand nested loops
- To implement programs that read and process data sets
- To use a computer for simulations

In this chapter, you will learn about loop statements in Python, as well as techniques for writing programs that simulate activities in the real world.

The while Loop

The **while** Loop

- Examples of loop applications
  - Calculating compound interest
  - Simulations, event driven programs
  - Drawing tiles…
- Compound interest algorithm (Chapter 1)

Start with a year value of 0, a column for the interest, and a balance of $10,000.

year	interest	balance
0		$10,000

Repeat the following steps while the balance is less than $20,000.

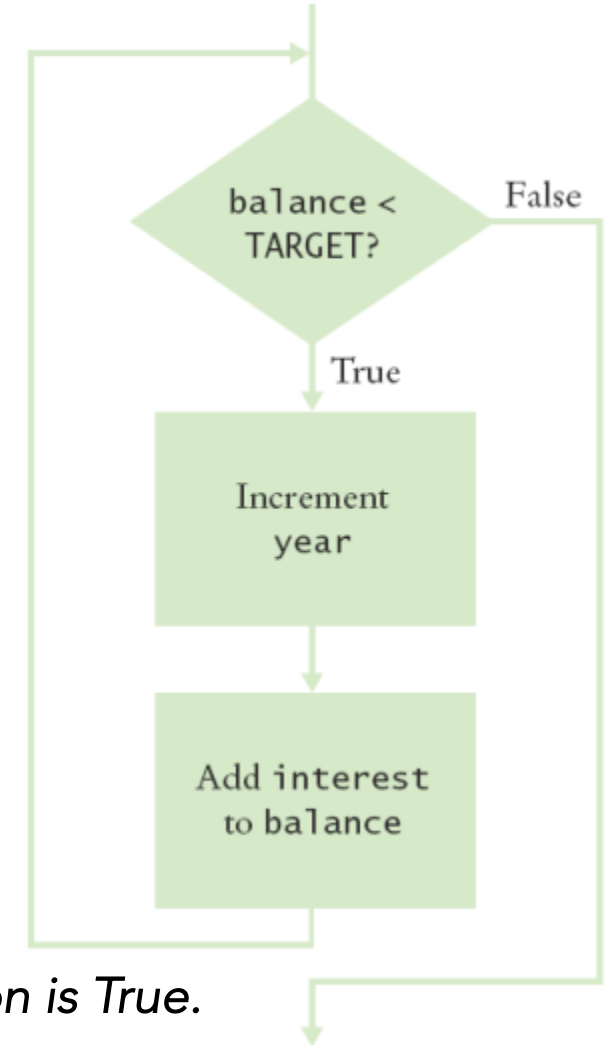
Steps

1. Add 1 to the year value.
2. Compute the interest as balance x 0.05 (i.e, 5 percent interest).
3. Add the interest to the balance.

Report the final year value as the answer.

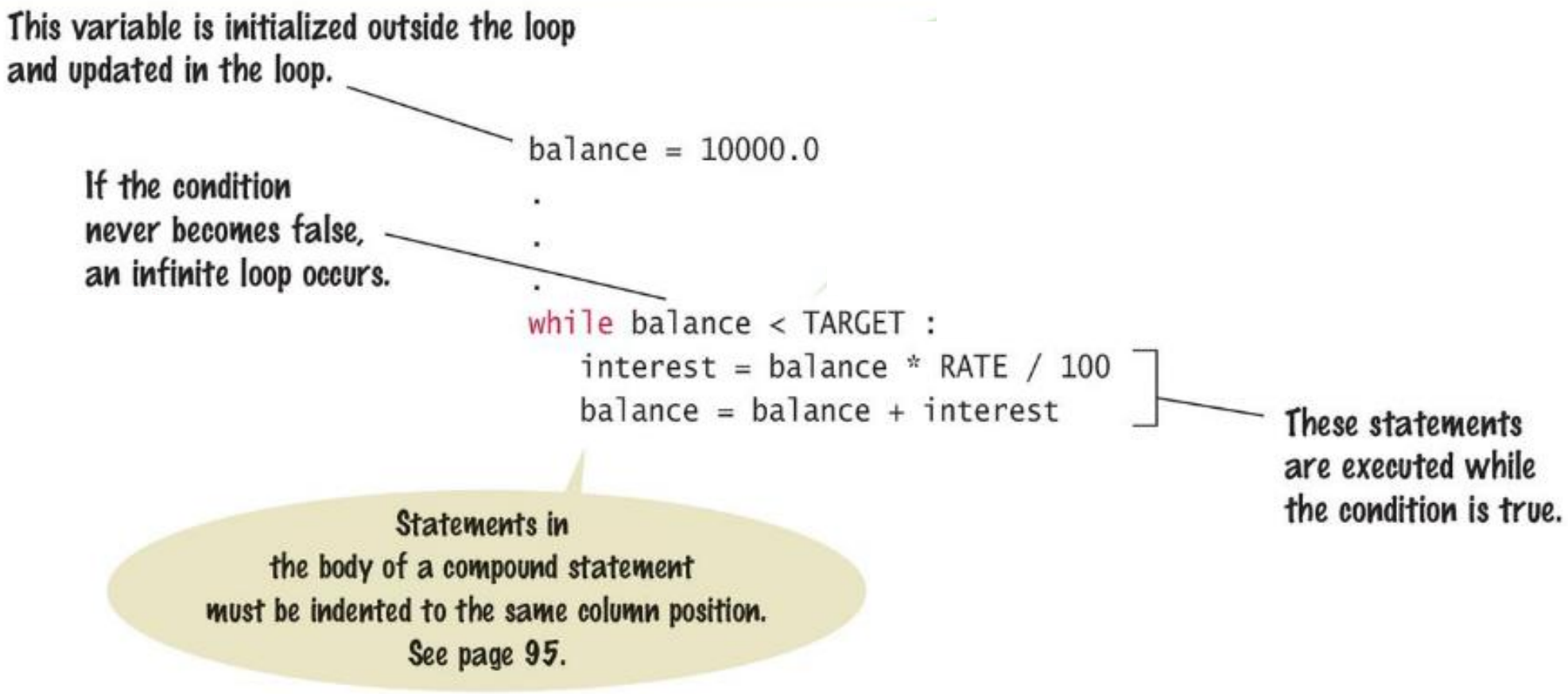
Planning the while Loop

```
balance = 10.0
target = 100.0
year = 0
rate = 0.025
while balance < TARGET :
   year = year + 1
   interest = balance * RATE/100
   balance = balance + interest
```



A loop executes instructions repeatedly while a condition is True.

Syntax: `while` Statement



Count-Controlled Loops

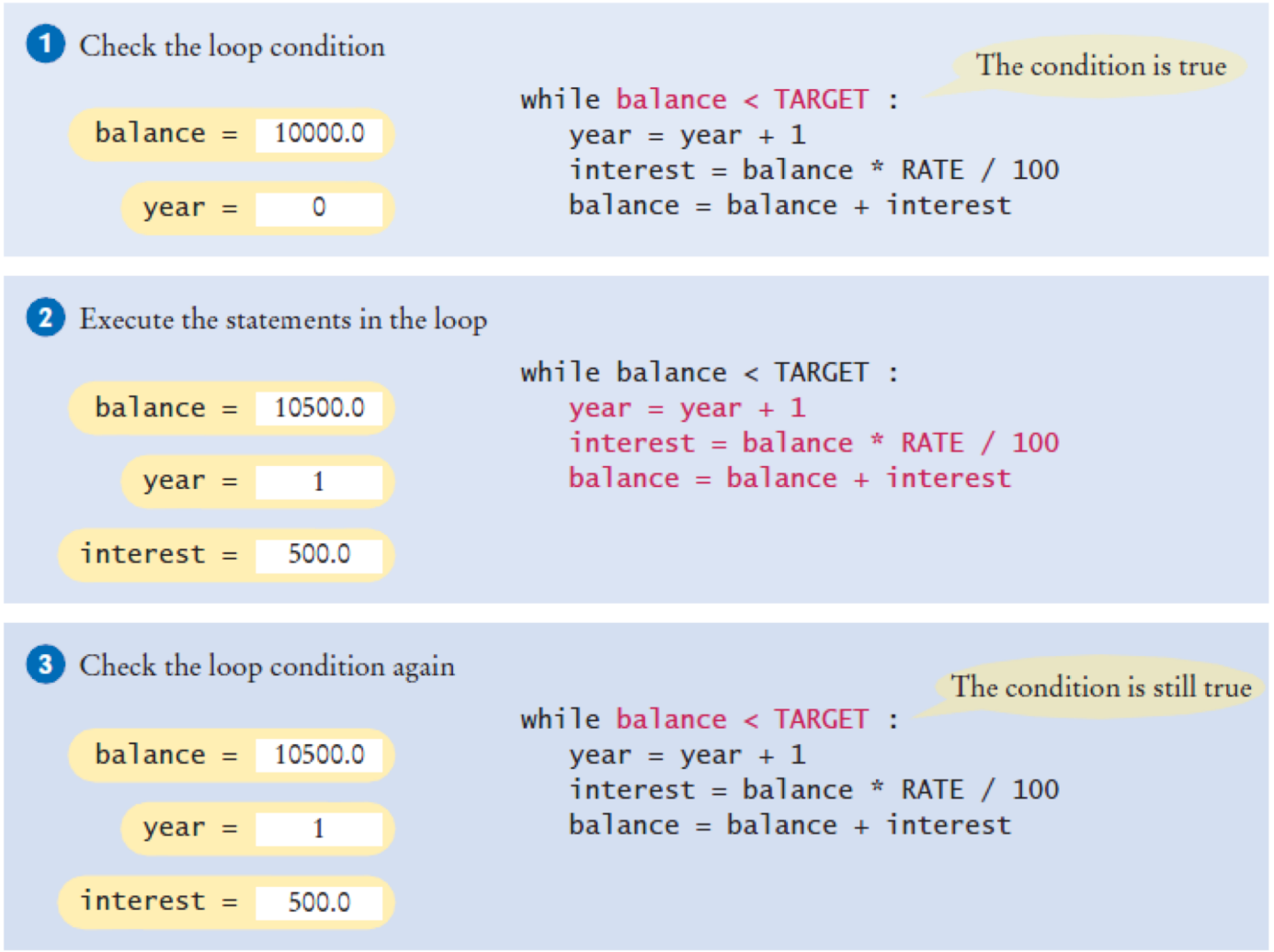
- A **while** loop that is controlled by a counter

```
counter = 1                        # Initialize the counter
while counter <= 10 :              # Check the counter
  print(counter)
  counter = counter + 1            # Update the loop variable
```

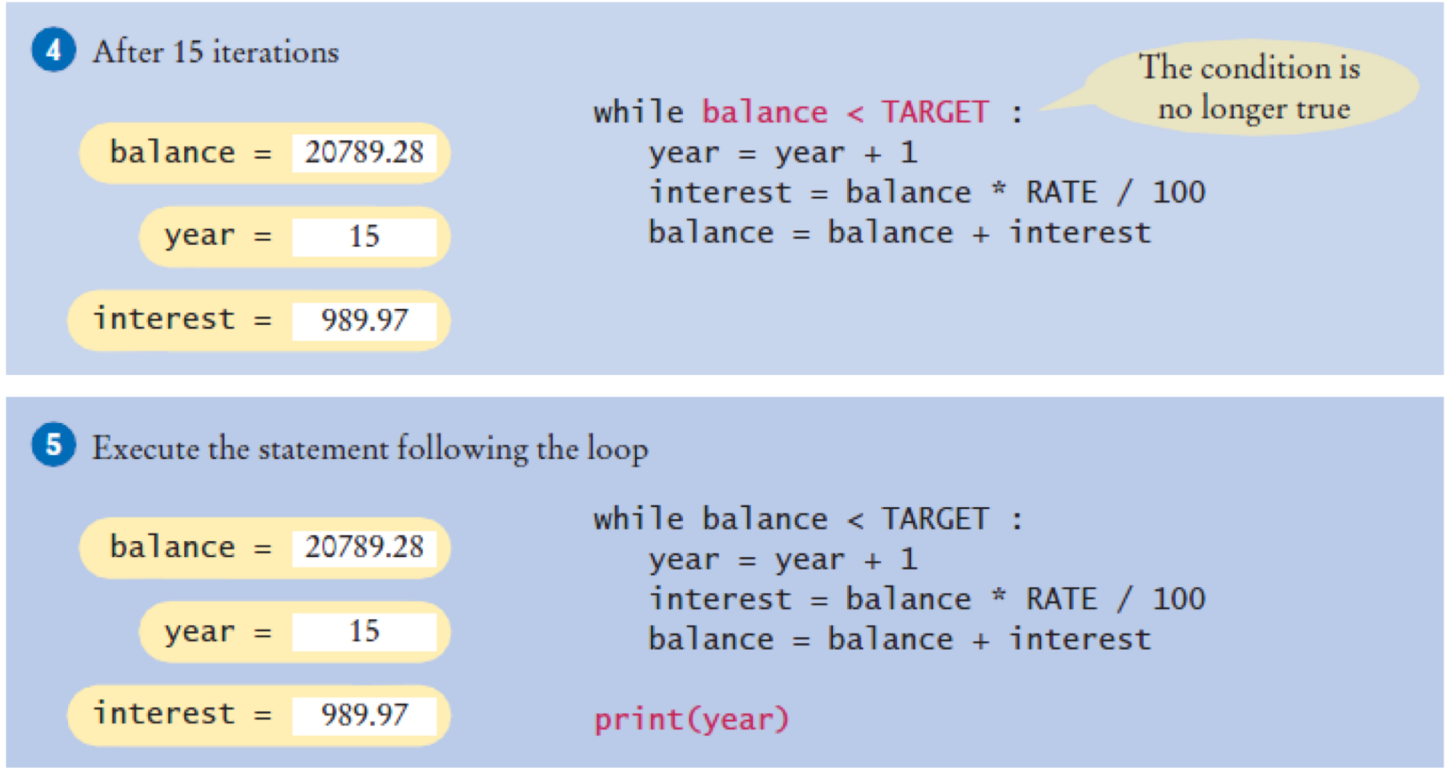
Exercises from Booklet

- Exercise 5.6
- Exercise 5.7

Execution of the Loop



Execution of the Loop (2)



`while` Loop Examples

Loop	Output	Explanation

`i = 0`
`total = 0`
`while total < 10 :`
`  i = i + 1`
`  total = total + i`
`  print(i, total)`
	`1 1`
	`2 3`
	`3 6`
	`4 10`
		When `total` is 10, the loop condition is false, and the loop ends.

`i = 0`
`total = 0`
`while total < 10 :`
`  i = i + 1`
`  total = total - 1`
`  print(i, total)`
	`1 -1`
	`2 -3`
	`3 -6`
	`4 -10`
	`. . .`
		Because `total` never reaches 10, this is an "infinite loop" (see Common Error 4.2 on page 161).

`i = 0`
`total = 0`
`while total < 0 :`
`  i = i + 1`
`  total = total - i`
`  print(i, total)`
	(No output)	The statement `total < 0` is false when the condition is first checked, and the loop is never executed.

`while` Loop Examples (2)

Loop	Output	Explanation

`i = 0`
`total = 0`
`while total >= 10 :`
`  i = i + 1`
`  total = total + i`
`  print(i, total)`
	(No output)	The programmer probably thought, "Stop when the sum is at least 10." However, the loop condition controls when the loop is executed, not when it ends (see Common Error 4.2 on page 161).

`i = 0`
`total = 0`
`while total >= 0 :`
`  i = i + 1`
`  total = total + i`
`print(i, total)`
	(No output, program does not terminate)	Because `total` will always be greater than or equal to 0, the loop runs forever. It produces no output because the `print` function is outside the body of the loop, as indicated by the indentation.

Common Error: Incorrect Test Condition

- The loop body will only execute if the test condition is True.
- If bal is initialized as less than the TARGET and should grow until it reaches TARGET
  - Which version will execute the loop body?

```
while bal >= TARGET :
    year = year + 1
    interest = bal * RATE
    bal = bal + interest
```

```
while bal < TARGET :
    year = year + 1
    interest = bal * RATE
    bal = bal + interest
```

Common Error: Infinite Loops

- The loop body will execute until the test condition becomes False.
- What if you forget to update the test variable?
  - bal is the test variable (TARGET doesn't change)
  - You will loop forever! (or until you stop the program)

```
while bal < TARGET :
    year = year + 1
    interest = bal * RATE
    bal = bal + interest
```

Common Error: Off-by-One Errors

- A 'counter' variable is often used in the test condition
- Your counter can start at 0 or 1, but programmers often start a counter at 0
- If I want to paint all 5 fingers on one hand, when I am done?
  - If you start at 0, use "<"
  - 0, 1, 2, 3, 4

  If you start at 1, use "<="

  1, 2, 3, 4, 5

```
finger = 0
FINGERS = 5
while finger < FINGERS :
    # paint finger
    finger = finger + 1
```

```
finger = 1
FINGERS = 5
while finger <= FINGERS :
    # paint finger
    finger = finger + 1
```

Exercise: sum_digits.py

- Write a program using the while loop to compute the sum of the first N integer. N is inserted by the user.

Sentinel Values

Processing Sentinel Values

- Sentinel values are often used:
  - When you don't know how many items are in a list, use a 'special' character or value to signal the "last" item
  - For numeric input of positive numbers, it is common to use the value -1

A sentinel value denotes the end of a data set, but it is not part of the data.

```
salary = 0.0
while salary >= 0 :
   salary = float(input())
      if salary >= 0.0 :
         total = total + salary
         count = count + 1
```

Averaging a Set of Values

- Declare and initialize a ‘total’ variable to 0
- Declare and initialize a ‘count’ variable to 0
- Declare and initialize a ‘salary’ variable to 0
- Prompt user with instructions
- Loop until sentinel value is entered
  - Save entered value to input variable (‘salary’)
  - If salary is not -1 or less (sentinel value)
    - Add salary variable to total variable
    - Add 1 to count variable
- Make sure you have at least one entry before you divide!
  - Divide total by count and output.
  - Done!

Sentinel.py (1)

```
 5  # Initialize variables to maintain the running total and count.
 6  total = 0.0
 7  count = 0
 8
 9  # Initialize salary to any non-sentinel value.
10  salary = 0.0

13  while salary >= 0.0 :

14      salary = float(input("Enter a salary or -1 to finish: "))
15      if salary >= 0.0 :

16          total = total + salary
17          count = count + 1
```

Outside the while loop: declare and initialize variables to use

Since `salary` is initialized to 0, the while loop statements will execute at least once

Input new `salary` and compare to sentinel

Update running `total` and `count` (to calculate the average later)

Sentinel.py (2)

```
19  # Compute and print the average salary.
20  if count > 0 :
21     average = total / count
22     print("Average salary is", average)

23  else :
24     print("No data was entered.")
```

Prevent divide by 0

Calculate and output the average salary using the `total` and `count` variables

Program Run

```
Enter salaries, -1 to finish: 10 10 40 -1
Average salary: 20
```

Priming Read

- Some programmers don't like the "trick" of initializing the input variable with a value other than a sentinel.

```
# Set salary to a value to ensure that the loop
# executes at least once.
salary = 0.0
while salary >= 0 :
```

- An alternative is to change the variable with a read before the loop.

```
salary = float(input("Enter a salary or -1 to finish: "))
while salary >= 0 :
```

Modification Read

- The input operation at the bottom of the loop is used to obtain the next input.

```
# Priming read
salary = float(input("Enter a salary or -1 to finish: "))
while salary >= 0.0 :
    total = total + salary
    count = count + 1
    # Modification read
    salary = float(input("Enter a salary or -1 to finish: "))
```

Boolean Variables and Sentinels

- A boolean variable can be used to control a loop
  - Sometimes called a 'flag' variable

```
done = False
while not done :
    value = float(input("Enter a salary or -1 to
   finish: "))
    if value < 0.0:
        done = True
    else :
        # Process value
```

Initialize done so that the loop will execute

Set done 'flag' to True if sentinel value is found

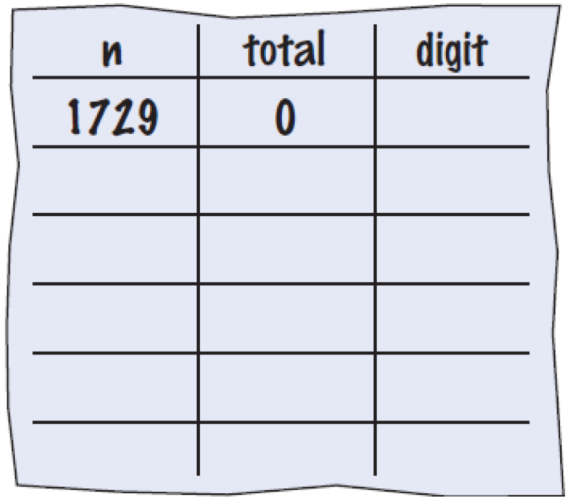
Exercise: id_student_while.py

- Write a program which ask a student to insert his/her id until the id is not conform to the standard format ("N85005656" as in previous example)

Hand Tracing Loops

Hand-Tracing Loops

- Example: Calculate the sum of digits (1+7+2+9)
  - Make columns for key variables (n, total, digit)
  - Examine the code and number the steps
  - Set variables to state before loop begins



n	total	digit
1729	0	

Tracing Sum of Digits

n	total	digit
1729	0	

```
n = 1729
total = 0
while n > 0 :
    digit = n % 10
    total = total + digit
    n = n // 10

print(total)
```

- Start executing loop body statements changing variable values on a new line
  - Cross out values in previous line

Tracing Sum of Digits

n	total	digit
1729	~~0~~	
	9	9

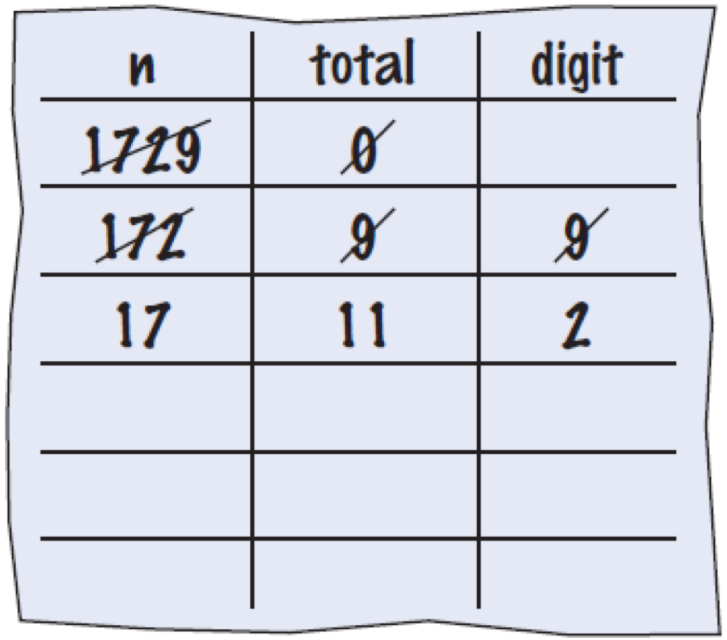
```
n = 1729
total = 0
while n > 0 :
    digit = n % 10
    total = total + digit
    n = n // 10

print(total)
```

- Continue executing loop statements changing variables
  - 1729 / 10 leaves 172 (no remainder)

Tracing Sum of Digits

- Test condition. If True, execute loop again
  - Variable n is 172, Is 172 > 0?, True!
- Make a new line for the second time through and update variables



n	total	digit
~~1729~~	~~0~~	
~~172~~	~~9~~	~~9~~
17	11	2

```
n = 1729
total = 0
while n > 0 :
    digit = n % 10
    total = total + digit
    n = n // 10

print(total)
```

Tracing Sum of Digits

- Third time through
  - Variable n is 17 which is still greater than 0
- Execute loop statements and update variables

n	total	digit
~~1729~~	~~0~~	
~~172~~	~~9~~	~~9~~
~~17~~	~~11~~	~~2~~
1	18	7

```
n = 1729
total = 0
while n > 0 :
    digit = n % 10
    total = total + digit
    n = n // 10

print(total)
```

Tracing Sum of Digits

- Fourth loop iteration:
  - Variable n is 1 at start of loop. 1 > 0? True
  - Executes loop and changes variable n to 0 (1/10 = 0)

n	total	digit
~~1729~~	~~0~~	
~~172~~	~~9~~	~~9~~
~~17~~	~~11~~	~~2~~
~~1~~	~~18~~	~~7~~
0	19	1

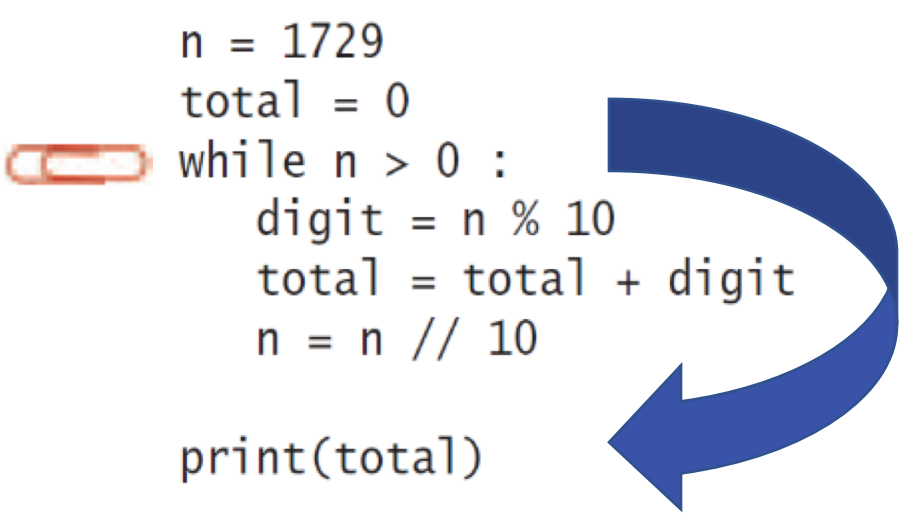
```
n = 1729
total = 0
while n > 0 :
    digit = n % 10
    total = total + digit
    n = n // 10

print(total)
```

Tracing Sum of Digits

- Because n is 0, the expression(n > 0) is False
- Loop body is not executed
  - Jumps to next statement after the loop body
- Finally prints the sum!

n	total	digit	output
~~1729~~	~~0~~		
~~172~~	~~9~~	~~9~~	
~~17~~	~~11~~	~~2~~	
~~1~~	~~18~~	~~7~~	
0	19	1	19



```
n = 1729
total = 0
while n > 0 :
    digit = n % 10
    total = total + digit
    n = n // 10

print(total)
```

Summary of the while Loop

- `while` loops are very common
- Initialize variables before you test
  - The condition is tested BEFORE the loop body
    - This is called pre-test
    - The condition often uses a counter variable
  - Something inside the loop should change one of the variables used in the test
- Watch out for infinite loops!

Common Loop Algorithms

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 use provided inputs

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: ")
```

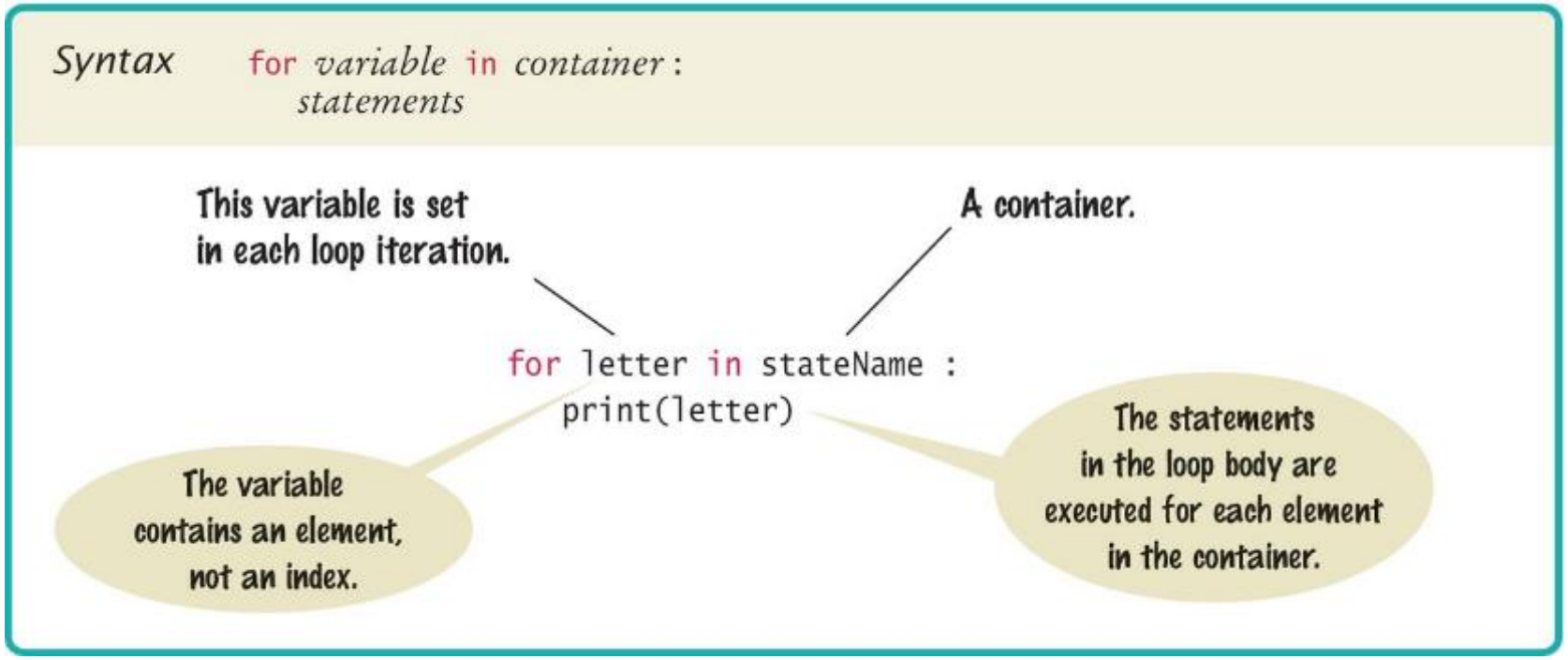
The for Loop

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 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

Syntax of a for Statement (Container)

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



An Example of a for Loop

- Note an important difference between the while loop and the for loop.
- In the while loop, the *index variable* i is assigned 0, 1, and so on.
- In the for loop, the *element variable* 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
```

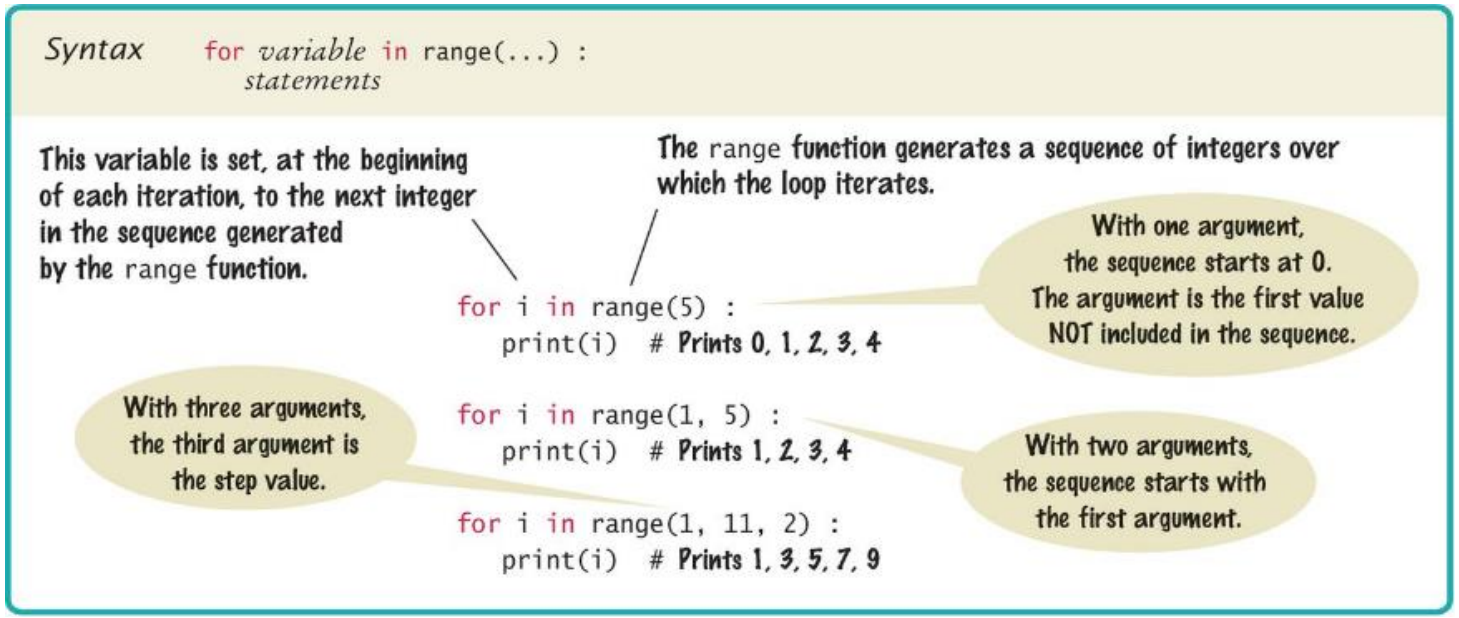
while version

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

for version

Syntax of a for Statement (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



Good Examples of for Loops

- Keep the loops simple!

Table 2 for Loop Examples

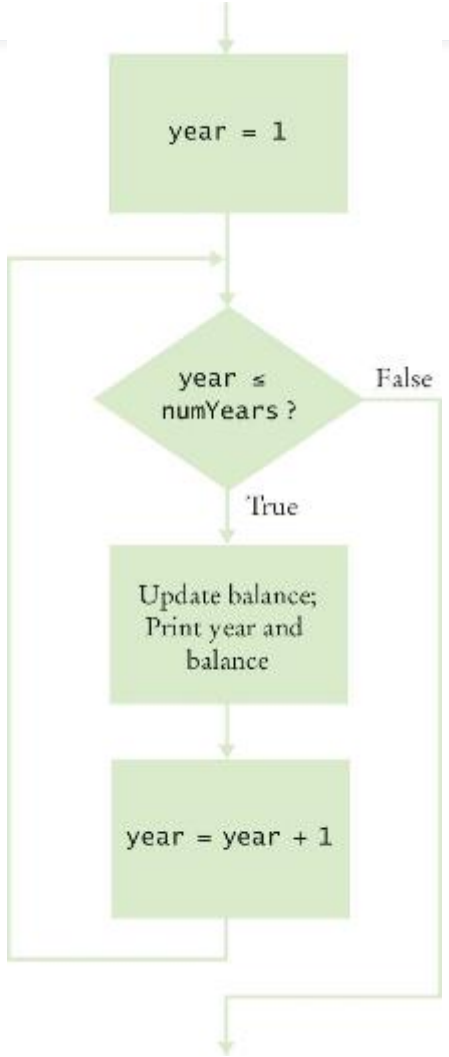
Loop	Values of i	Comment
`for i in range(6) :`	0, 1, 2, 3, 4, 5	Note that the loop executes 6 times.
`for i in range(10, 16) :`	10, 11, 12, 13, 14 15	The ending value is never included in the sequence.
`for i in range(0, 9, 2) :`	0, 2, 4, 6, 8	The third argument is the step value.
`for i in range(5,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

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.
```



Investment Example

```
1  ##
2  #  This program prints a table showing the growth of an investment.
3  #
4
5  # Define constant variables.
6  RATE = 5.0
7  INITIAL_BALANCE = 10000.0
8
9  # Obtain the number of years for the computation.
10 numYears = int(input("Enter number of years: "))
11
12 # Print the table of balances for each year.
13 balance = INITIAL_BALANCE
14 for year in range(1, numYears + 1) :
15    interest = balance * RATE / 100
16    balance = balance + interest
17    print("%4d %10.2f" % (year, balance))
```

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

Nested Loops

Loops Inside of Loops

- We learned how to nest **if** statements to allow us to make complex decisions
  - Remember that to nest the **if** statements we need to indent the code block
- Complex problems sometimes require a nested loop, one loop nested inside another loop
  - The nested loop will be indented inside the code block of the first loop
- A good example of using nested loops is when you are processing cells in a table
  - The outer loop iterates over all of the rows in the table
  - The inner loop processes the columns in the current row

Our Example Problem Statement

- Print a Table Header that contains x^1, x^2, x^3, and x^4
- Print a Table with four columns and ten rows that contain the powers of x^1, x^2, x^3, and x^4 for x = 1 to 10

x^1	x^2	x^3	x^4
1	1	1	1
2	4	8	16
3	9	27	81
...	...	...	...
10	100	1000	10000

Applying Nested Loops

- How would you print a table with rows and columns?
  - Print top line (header)
    - Use a for loop
  - Print table body…
    - How many rows are in the table?
    - How many columns in the table?
  - Loop per row
    - Loop per column
- In our example there are:
  - Four columns in the table
  - Ten rows in the table

x^1	x^2	x^3	x^4
1	1	1	1
2	4	8	16
3	9	27	81
…	…	…	…
10	100	1000	10000

Pseudocode to Print the Table

Print the table header

```
for x from 1 to 10
  print a new table row
  print a new line
```

- How do we print a table row?

```
For n from 1 to 4
  print x^n
```

- We have to place this loop inside the preceding loop
  - The inner loop is *"nested"* inside the outer loop

Pseudocode to Print the Table

Print the table header:

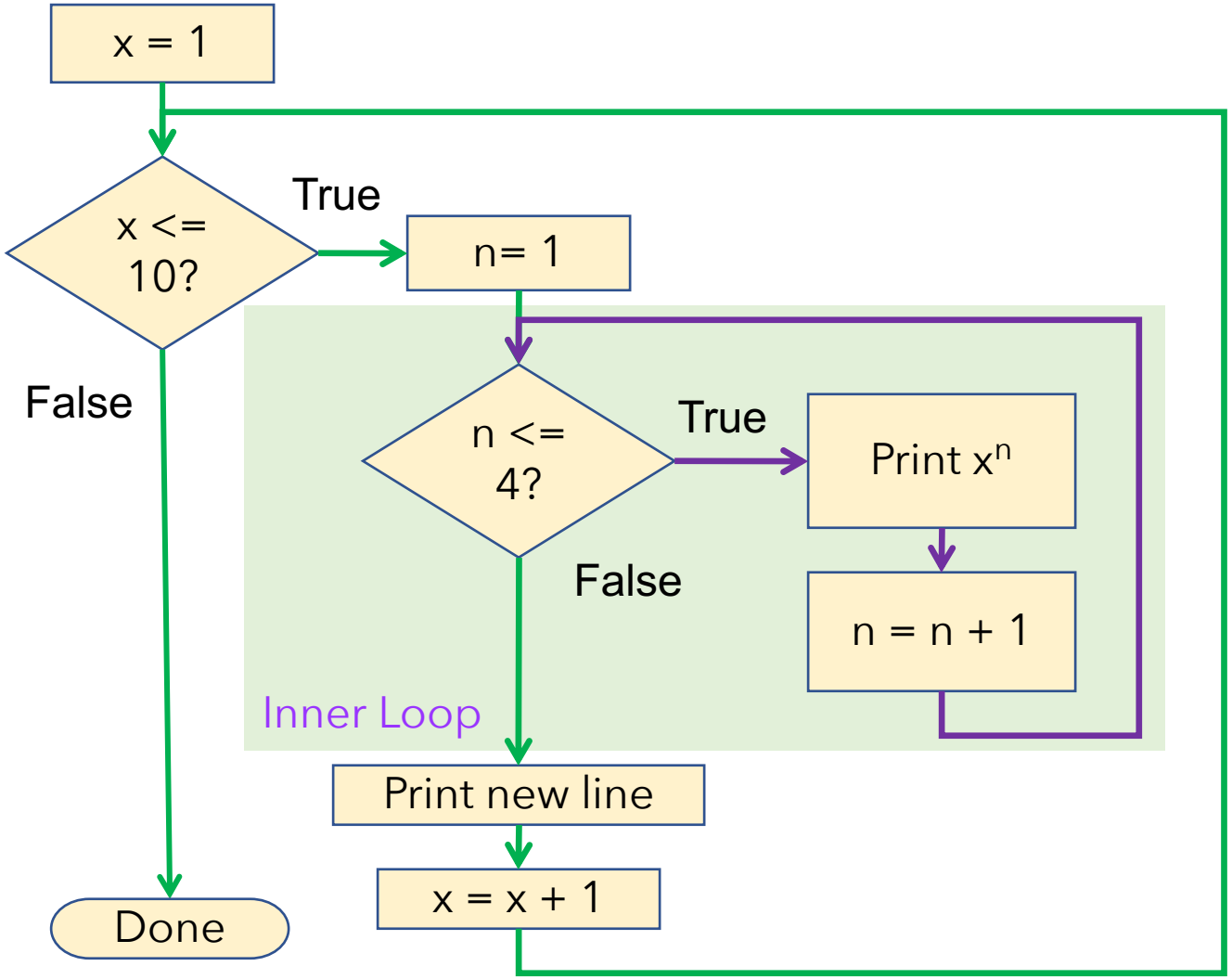
```
for x from 1 to 10
  for n from 1 to 4
    print X^n
  print a new line
```

n →

x ↓

x^1	x^2	x^3	x^4
1	1	1	1
2	4	8	16
3	9	27	81
…	…	…	…
10	100	1000	10000

Flowchart of a Nested Loop



Exercise: powertable.py

Write a program that given a integer number **x** in the range [1,10] and integer number **n** computes the powers from 1 to n of all the numbers from 1 to x by presenting them in a tabular format as shown in figure:

x^1	x^2	x^3	x^4
1	1	1	1
2	4	8	16
3	9	27	81
…	…	…	…
10	100	1000	10000

Powertable.py

```
1   #
2   #  This program prints a table of powers of x.
3   #
4
5   # Initialize constant variables for the max ranges.
6   NMAX = 4
7   XMAX = 10
8
9   # Print table header.
10  #
11
12  for n in range(1, NMAX + 1) :
13     print("%10d" % n, end="")
14
15  print()
16  for n in range(1, NMAX + 1) :
17     print("%10s" % "x ", end="")
18
19  print("\n", "    ", "-" * 35)
20
21  # Print table body.
22  #
23
24  for x in range(1, XMAX + 1) :
25     # Print the x row in the table.
26     for n in range(1, NMAX + 1) :
27        print("%10.0f" % x ** n, end="")
28
29     print()
30
```

The **end=""** suppresses the new line, so the numbers are all printed on the same line

Body of outer loop, x = 1 ➔ 10

Body of inner loop, n = 1 ➔ 4

First Exercise

- Open the program:
  - powertable.py
- Run the program and review the results
- Make the following changes:
  - Change the value of NMAX to 6 and run the program
  - What changes in the table?
  - Change the value of NMAX back to 4
  - Change the value of XMAX to 4
  - What changes in the table?

Nested Loop Examples

Table 3 Nested Loop Examples

Nested Loops	Output	Explanation

`for i in range(3) :`
`  for j in range(4) :`
`  print("*", end="")`
`  print()`
	****
	****
	****
		Prints 3 rows of 4 asterisks each.

`for i in range(4) :`
`  for j in range(3) :`
`  print("*", end="")`
`  print()`
	***
	***
	***
	***
		Prints 4 rows of 3 asterisks each.

`for i in range(4) :`
`  for j in range(i + 1) :`
`  print("*", end="")`
`  print()`
	*
	**
	***
	****
		Prints 4 rows of lengths 1, 2, 3, and 4.

Nested Loop Examples (2)

Table 3 Nested Loop Examples

`for i in range(3) :`
`  for j in range(5) :`
`  if j % 2 == 1 :`
`  print("*", end="")`
`  else :`
`  print("-", end="")`
`  print()`
	-*-*-
	-*-*-
	-*-*-
		Prints alternating dashes and asterisks.

`for i in range(3) :`
`  for j in range(5) :`
`  if i % 2 == j % 2 :`
`  print("*", end="")`
`  else :`
`  print(" ", end="")`
`  print()`
	* * *
	 * *
	* * *
		Prints a checkerboard pattern.

Processing Strings

Processing Strings

- A common use of loops is to process or evaluate strings.
- For example, you may need to count the number of occurrences of one or more characters in a string or verify that the contents of a string meet certain criteria.

String Processing Examples

- Counting Matches
- Finding All Matches
- Finding the First or Last Match
- Validating a String
- Building a New String

Counting Matches

- Suppose you need to count the number of uppercase letters contained in a string.
- We can use a for loop to check each character in the string to see if it is upper case
- The loop below sets the variable **char** equal to each successive character in the string
- Each pass through the loop tests the next character in the string to see if it is uppercase

```
uppercase = 0
for char in string :
   if char.isupper() :
      uppercase = uppercase + 1
```

Counting Vowels

- Suppose you need to count the vowels within a string
- We can use a for loop to check each character in the string to see if it is in the string of vowels "aeiuo"
- The loop below sets the variable **char** equal to each successive character in the string
- Each pass through the loop tests the lower case of the next character in the string to see if it is in the string "aeiou"

```
vowels = 0
for char in word :
   if char.lower() in "aeiou" :
      vowels = vowels + 1
```

Finding All Matches Example

- When you need to examine every character in a string, independent of its position we can use a for statement to examine each character
- If we need to print the position of each uppercase letter in a sentence we can test each character in the string and print the position of all uppercase characters
- We set the range to be the length of the string
  - We test each character
  - If it is uppercase we print I,

    its position in the string

```
sentence = input("Enter a sentence: ")
for i in range(len(sentence)) :
   if sentence[i].isupper() :
      print(i)
```

Finding the First Match

- This example finds the position of the first digit in a string.

```
found = False
position = 0
while not found and position < len(string) :
   if string[position].isdigit() :
      found = True
   else :
      position = position + 1

if found :
   print("First digit occurs at position", position)
else :
   print("The string does not contain a digit.")
```

Finding the Last Match

- Here is a loop that finds the position of the last digit in the string.
- This approach uses a while loop to start at the last character in a string and test each value moving from the end of the string to the start of the string
  - Position is set to the length of the string - 1
  - If the character is not a digit, we decrease position by 1
  - Until we find a digit, or process all the characters

```
found = False
position = len(string) - 1
while not found and position >= 0 :
   if string[position].isdigit() :
      found = True
   else :
      position = position - 1
```

Validating a String

- In the United States, telephone numbers consist of three parts—area code exchange, and line number—which are commonly specified in the form (###)###-####.

Validating a String (code)

- We can examine a string to ensure that it contains a correctly formatted phone number. (e.g., (703)321-6753)
- The loop test each character to see it it is correct for its position, or a number

```
valid = len(string) == 13
position = 0
while valid and position < len(string) :
   valid = ((position == 0 and string[position] != "(")
       or (position == 4 and string[position] != ")")
       or (position == 8 and string[position] != "-")
       or (position != 0 and position != 4 and position != 8
          and string[position].isdigit())) :
   position = position + 1
```

Building a New String

- One of the minor annoyances of online shopping is that many web sites require you to enter a credit card without spaces or dashes, which makes double-checking the number rather tedious.
- How hard can it be to remove dashes or spaces from a string?

Building a New String (code)

- The contents of a string cannot be changed.
- But nothing prevents us from building a new string.
- Here is a loop that builds a new string containing a credit card number with spaces and dashes removed:
  - We read the credit card number
  - We initialize a new string to the empty string
  - We test each character in the user input
    - If the character is not a space or dash we append it to the new string

```
userInput = input("Enter a credit card number: ")
creditCardNumber = ""
for char in userInput :
   if char != " " and char != "-" :
      creditCardNumber = creditCardNumber + char
```

Application: Random Numbers and Simulations

Random Numbers/Simulations

- Games often use random numbers to make things interesting
  - Rolling Dice
  - Spinning a wheel
  - Pick a card
- A simulation usually involves looping through a sequence of events
  - Days
  - Events

Generating Random Numbers

- The Python library has a *random number generator* that produces numbers that appear to be random
  - The numbers are not completely random. The numbers are drawn from a sequence of numbers that does not repeat for a long time
  - random() returns a number that is >= 0 and < 1

Simulating Die Tosses

- Goal:
  - To generate a random integer in a given range we use the randint() function
  - Randint has two parameters, the range (inclusive) of numbers generated

ch04/dice.py

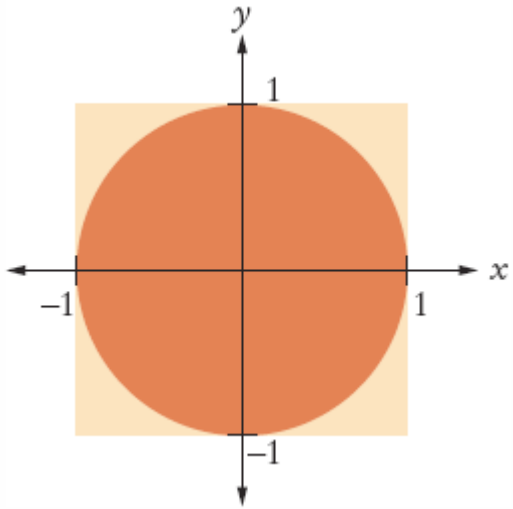
```
1  ##
2  #  This program simulates tosses of a pair of dice.
3  #
4
5  from random import randint
6
7  for i in range(10) :
8     # Generate two random numbers between 1 and 6, inclusive.
9     d1 = randint(1, 6)
10    d2 = randint(1, 6)
11
12    # Print the two values.
13    print(d1, d2)
```

Program Run

```
1 5
6 4
1 1
4 5
6 4
3 2
4 2
3 5
5 2
4 5
```

The Monte Carlo Method

- Used to find approximate solutions to problems that cannot be precisely solved
- Example: Approximate PI using the relative areas of a circle inside a square
  - Uses simple arithmetic
  - Hits are inside circle
  - Tries are total number of tries
  - Ratio is 4 x Hits / Tries



Monte Carlo Example

```
1  ##
2  #  This program computes an estimate of pi by simulating dart throws onto a square
3  #
4
5  from random import random
6
7  TRIES = 10000
8
9  hits = 0
10 for i in range(TRIES) :
11
12    # Generate two random numbers between -1 and 1
13    r = random()
14    x = -1 + 2 * r
15    r = random()
16    y = -1 + 2 * r
17
18    # Check whether the point lies in the unit circle
19    if x * x + y * y <= 1 :
20       hits = hits + 1
21
22 # The ratio hits / tries is approximately the same as the ratio
23 # circle area / square area = pi / 4.
24
25 piEstimate = 4.0 * hits / TRIES
26 print("Estimate for pi:", piEstimate)
```

Program Run

```
Estimate for pi: 3.1464
```

MASTER MEIM 2021-2022

Thank you for your attention