ICS 104 - Introduction to Programming in Python and C

Loops

Reading Assignment

- Chapter 4 Sections 1, 2, 3, 5, 6, 7, 8 and 9.

Chapter Learning Outcomes

At the end of this chapter, you will be able to

- implement while and for loops
- become familiar with common loop algorithms
- understand nested loops
- process strings

Loops

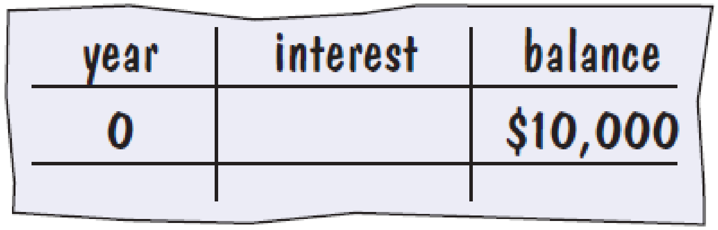
- In a **loop**, a part of a program is repeated over and over until a specific goal is reached.
- Loops are important for calculations that require repeated steps, and for processing input consisting of many data items.

© photo75/iStockphoto.

The While Loop

- For example, You put $10,000 into a bank account that earns 5 percent interest per year.
- How many years does it take for the account balance to be double the original investment?

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.
    Add 1 to the year value.
    Compute the interest as balance x 0.05 (i.e., 5 percent interest).
    Add the interest to the balance.
Report the final year value as the answer.
```

- Question: How we can implement the "Repeat steps while the balance is less than $20,000?"

- Answer: Using the while loop statement

The While Loop

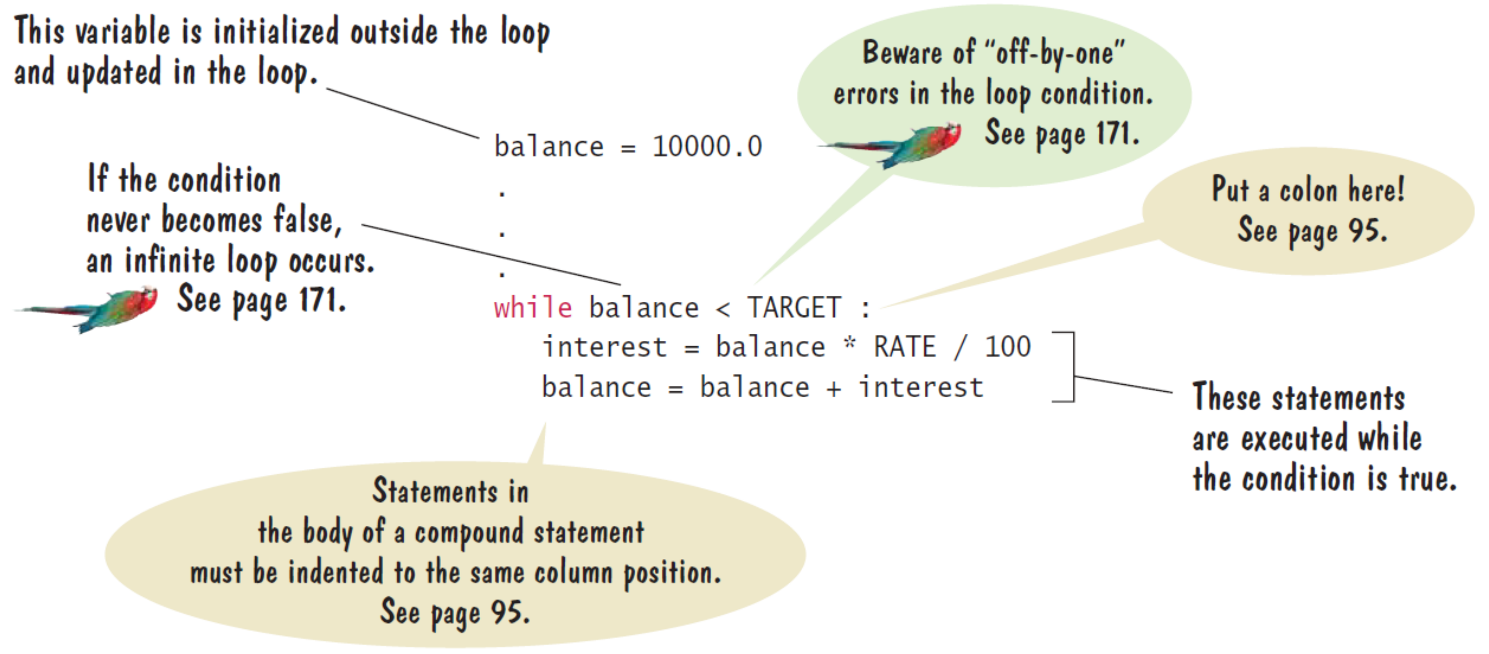
`while` *condition* `:`

```
    statements
```

In [ ]:
```
while balance < TARGET :
    year = year + 1
    interest = balance * RATE / 100
    balance = balance + interest
```

Syntax

```
while condition :
    statements
```



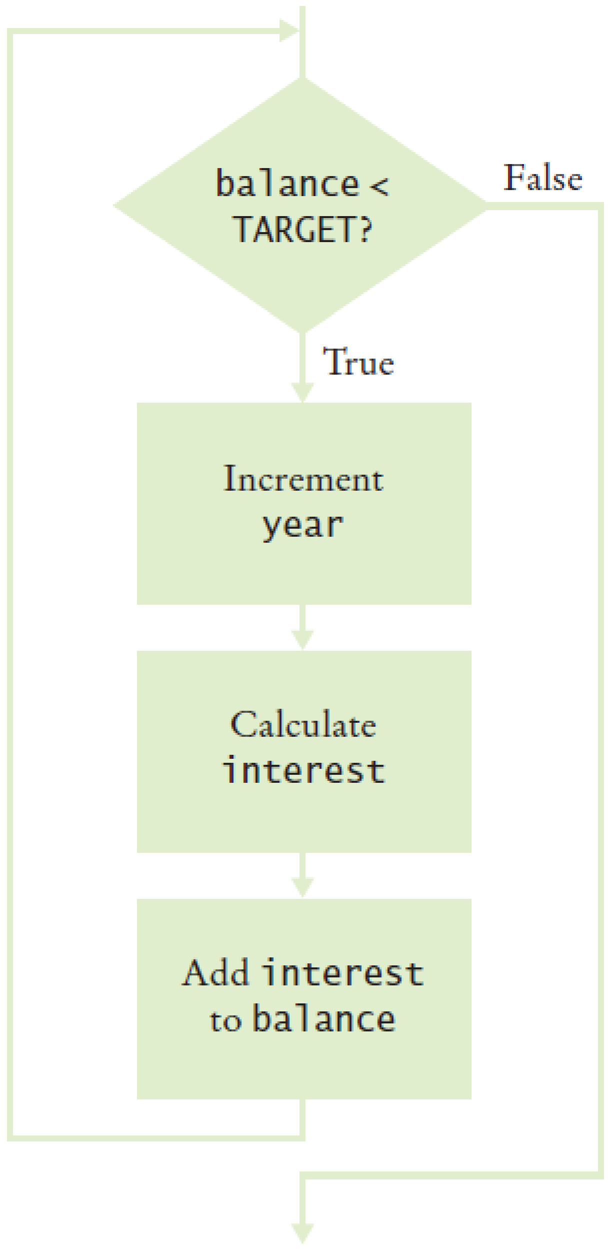
```
balance = 10000.0
.
.
.
while balance < TARGET :
   interest = balance * RATE / 100
   balance = balance + interest
```

This variable is initialized outside the loop and updated in the loop.
If the condition never becomes false, an infinite loop occurs. See page 171.
Beware of "off-by-one" errors in the loop condition. See page 171.
Put a colon here! See page 95.
These statements are executed while the condition is true.
Statements in the body of a compound statement must be indented to the same column position. See page 95.

The While Loop

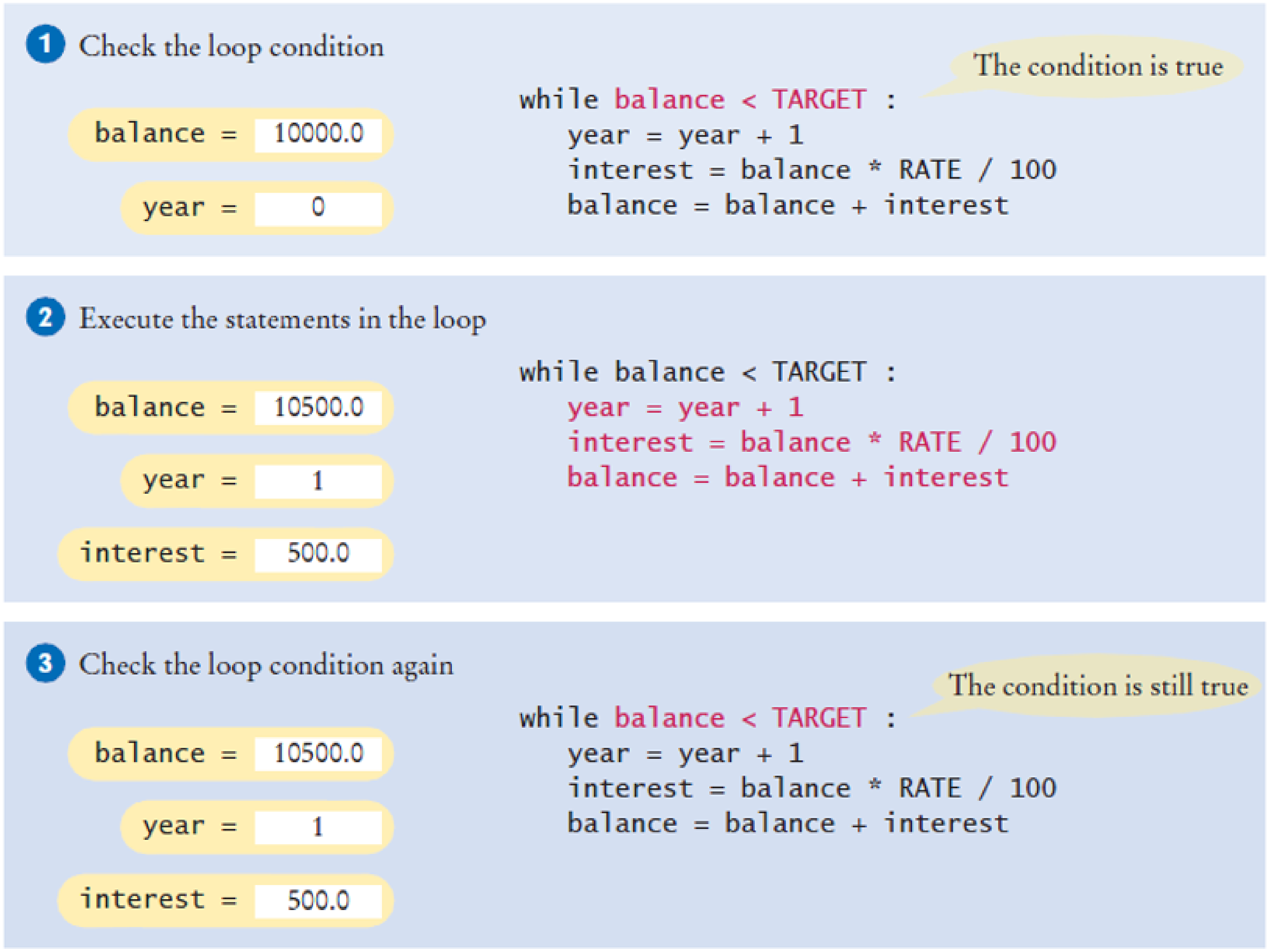
- As long as the condition remains true, the statements inside the **while** statement are executed.
- This statement block is called the **body** of the while statement.

- For example, we want to increment the year counter and add interest while the balance is less than the target balance of $20,000:


balance < TARGET?
False
True
Increment year
Calculate interest
Add interest to balance

The While Loop

Execution of the Loop



```
while balance < TARGET :
   year = year + 1
   interest = balance * RATE / 100
   balance = balance + interest
```

```
while balance < TARGET :
   year = year + 1
   interest = balance * RATE / 100
   balance = balance + interest
```

```
while balance < TARGET :
   year = year + 1
   interest = balance * RATE / 100
   balance = balance + interest
```

The While Loop

Event-Controlled Loops

In [ ]:

```
#  This program computes the time required to double an investment.
# Create constant variables.
RATE = 5.0
INITIAL_BALANCE = 10000.0
TARGET = 2 * INITIAL_BALANCE

# Initialize variables used with the loop.
balance = INITIAL_BALANCE
year = 0

# Count the years required for the investment to double.
while balance < TARGET :
   year = year + 1
   interest = balance * RATE / 100
   balance = balance + interest

# Print the results.
print("The investment doubled after", year, "years.")
```

The While Loop

Count-Controlled Loops

- A while loop that is controlled by a counter:

In [ ]:
```
counter = 1                  # Initialize the counter

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

The While Loop - Student Activity

- What does the following loop print?

In [ ]:

```
n =1
while n < 100:
    n = 2* n
    print(n)
```

The While Loop - Student Activity

- What does the following loop print?

In [ ]:
```
i = 0
total = 0
while total < 10:
    i = i + 1
    total = total + i
    print(i,total)
```

The while Loop - Student Activity

- What does the following loop print?

In [ ]:
```
i = 0
total = 0
while total < 0:
    i = i + 1
    total = total - i
    print(i,total)
```

The While Loop - Student Activity

- We want to write loops that read and process a sequence of input values.
- A **sentinel value** denotes the end of a data set, but it is not part of the data.

- We want to write a program that computes the average of a set of salary values.
- We will use any negative value as the sentinel.
    - An employee would surely not work for a negative salary.

In [ ]:

```python
#  This program prints the average of salary values that are terminated with
#  a sentinel.

# Initialize variables to maintain the running total and count.
total = 0.0
count = 0

# Initialize salary to any non-sentinel value.
salary = 0.0

# Process data until the sentinel is entered.
while salary >= 0.0 :
   salary = float(input("Enter a salary or -1 to finish: "))
   if salary >= 0.0 :
      total = total + salary
      count = count + 1

# Compute and print the average salary.
if count > 0 :
   average = total / count
   print("Average salary is", average)
else :
   print("No data was entered.")
```

Common Loop Algorithms

- Sum and Average Values:

```
In [ ]:
total = 0.0
inputStr = input("Enter value: ")
while inputStr !="":
    value = float(inputStr)
    total = total + value
    inputStr = input("Enter value: ")
print("Sum: ",total)
```

Common Loop Algorithms

- Sum and Average Value:

In [ ]:
```
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
print("Average: ",average)
```

Common Loop Algorithms

- Counting Matches - You want to count how many negative values are included in a sequence of integers.
- Keep a **counter**, a variable that is initialized with 0 and incremented whenever there is a match.

In a loop that counts matches, a counter is incremented whenever a match is found.

In [ ]:

```python
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.")
```

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 a sequence of characters.

The for Loop

- Suppose we want to print a string, with one character per line.
- We cannot simply print the string using the **print** funciton.
- Instead, we need to iterate over the characters in the string and print each character individually.

- 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 with a string container `stateName`, the element variable is assigned `stateName[0]`, `stateName[1]`, and so on.

In [ ]:

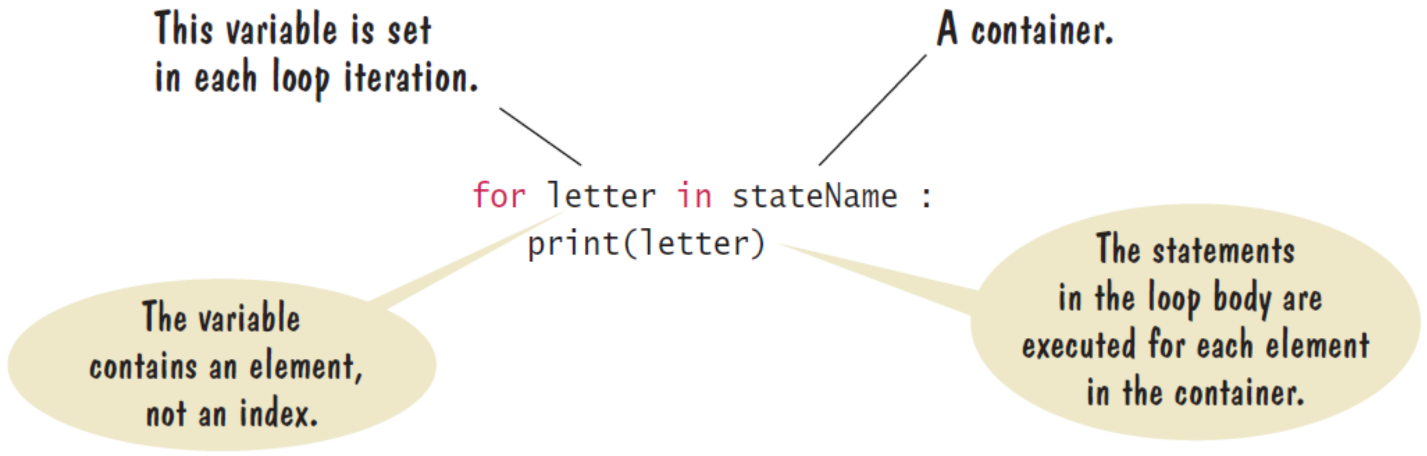
```
stateName = "Virginia"
for letter in stateName :
    print(letter)
```

- The loop body is executed for each character in the string `stateName`, starting with the first character.
- At the beginning of each loop iteration, the next character is assigned to the variable letter.
- Then the loop body is executed.

for Statement

Syntax

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



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

This variable is set in each loop iteration.
A container.
The variable contains an element, not an index.
The statements in the loop body are executed for each element in the container.

The for Loop

- Can we write this program using **while** loop?

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

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

The for Loop

- The **for** loop can be used with the `range` function to iterate over a range of integer values.
- When we write `range(i,j)`, what are the range values (assuming that $i < j$?)

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

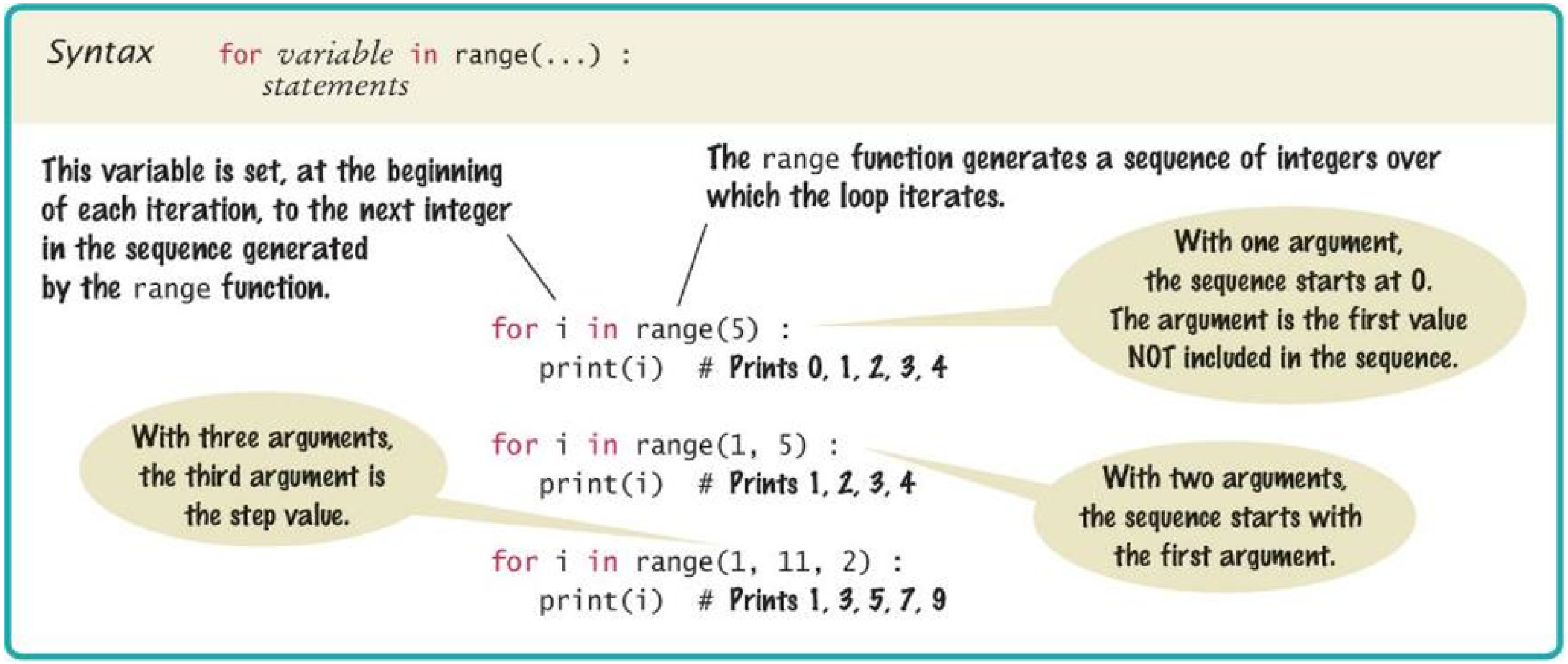
Student Activity

- Write an equivalent while loop for the previous example:

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

The range Function

- 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 are less than the argument that can be used with the for loop.



Syntax

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

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

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

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

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.
With one argument, the sequence starts at 0. The argument is the first value NOT included in the sequence.
With three arguments, the third argument is the step value.
With two arguments, the sequence starts with the first argument.

The for Loop - Student Activity

- Use the **for** loop to print only the odd values between 1 and 10 using the **range function.

In [ ]:
```
for i in range(1,10,2):
    print(i)
```

Nested Loop

- When the body of a loop contains another loop, the loops are nested.
- A typical use of nested loops is printing a table with rows and colums.

- For example, we will print the powers of x as in the following 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

Nested Loop

- The pseudocode for printing the table is as follows:

```
Print table header.
For x from 1 to 10
    Print table row.
    Print new line.
```

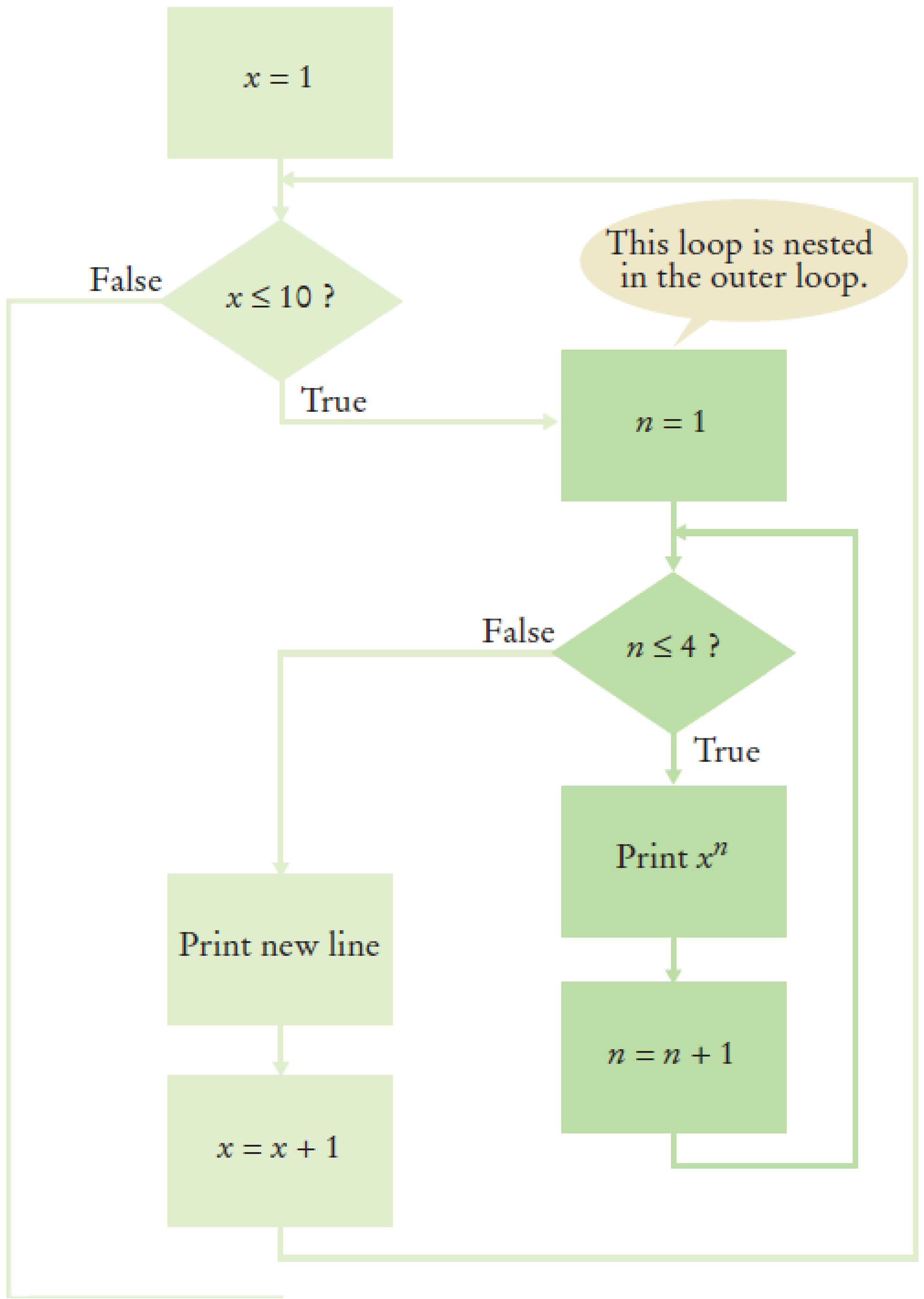
- How do you print a table row?

- You need to print a value for each component.
- This requires a second loop.

```
For n from 1 to 4
    Print $x^n$.
```

- This loop must be placed inside the preceding loop. We say that the inner loop is **nested** inside the outer loop.

Nested Loop


$x = 1$
False
$x \leq 10$?
True
This loop is nested in the outer loop.
$n = 1$
False
$n \leq 4$?
True
Print x^n
Print new line
$n = n + 1$
$x = x + 1$

Side Note Regarding the `print` Function

- The `print` function displays an end of line by default.
- If we want to change this behavior, we can set the end parameter to another string.
    - The default value of the end parameter is \n.
- Consider the following example

In [ ]:
```
course = "ICS 104"
University = "KFUPM"
print(course, end = "@")
print(University)
```

Nested Loop

In [ ]:

```python
#  This program prints a table of powers of x.
# Initialize constant variables for the max ranges.
NMAX = 4
XMAX = 10

# Print table header.
for n in range(1, NMAX + 1) :
    print("%10d" % n, end="")

print()
for n in range(1, NMAX + 1) :
    print("%10s" % "x ", end="")

print("\n", "    ", "-" * 35)

# Print table body.
for x in range(1, XMAX + 1) :
   # Print the x row in the table.
    for n in range(1, NMAX + 1) :
        print("%10.0f" % x ** n, end="")
    print()
```

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.

Processing Strings

Counting Matches

- For example, suppose you need to count the number of uppercase letters contained in a string.

In [ ]:

```
string = "This is a Test Message"
uppercase = 0
for char in string:
    if char.isupper():
        uppercase = uppercase + 1
print("The number of uppercase letters are:",uppercase)
```

Processing Strings

Finding All Matches

- For example, suppose you are asked to print the position of each uppercase letter in a sentence.

In [ ]:

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

Processing Strings

Finding the First or Last Match

- When you count the value that fulfill a condition, you need to look at all values.
- However, if your task is to find a match, then you can stop as soon as the condition is fulfilled.

In [ ]:

```
string = "A1"
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.")
```

Processing Strings - Student Activity

- What if we need to find the position of the last digit in the string?

In [ ]:

```
string = "A1B2"
found = False
position = len(string) -1
while not found and position >=0:
    if string[position].isdigit():
        found = True
    else:
        position = position -1
if found:
    print("Last digit occurs at position",position)
else:
    print("The string does not contain a digit.")
```

Processing Strings - Student Activity

- It is important to validate user input before it is used in computations.
- But data validation is not limited to verifying that user input is a specific value or falls within a valid range.
- It is also common to require user input to be entered in a specific format.
- For example, consider the task of verifying whether a string contains a correctly formatted telephone number.
  - In USA, telephone numbers consist of three parts, area code, exchange, and line number **(###)###-####**.
- Hint: We will need a loop that can exit early if an invalid character or an out of place symbol is encountered while processing the string:

In [ ]:

```
string = "(323)570-1234"
valid = len(string) == 13
position = 0
while valid and position < len(string):
    if position == 0:
        valid = string[position] == "("
    elif position == 4:
        valid = string[position] == ")"
    elif position == 8:
        valid = string[position] == "-"
    else:
        valid = string[position].isdigit()
    position = position + 1
if valid:
    print("The string contains a valid phone number.")
else:
    print("The string does not contain a valid phone number")
```

Application: Random Numbers and Simulations

- A simulation program uses the computer to simulate an activity in the real world.
- Simulations are commonly used for predicting climate change, analyzing traffic, picking stocks, and many other applications in science and business.
- In many simulations, one or more loops are used to modify the state of a system and observer the change.

Application: Random Numbers and Simulations

Generating Random Numbers

- Many events in the real world are difficult to predict with absolute precision, yet we can sometimes know the average behavior quite well.
- For example, a store may know from experience that a customer arrives every five minutes.
    - Of course, that is an average—customers don't arrive in five minute intervals.
    - To accurately model customer traffic, you want to take that random fluctuation into account.
    - Now, how can you run such a simulation in the computer?

Application: Random Numbers and Simulations

- The Python library has a random number generator that produces numbers that appear to be completely random.
- Calling random() yields a random floating-point number that is ≥ 0 and < 1.
- Call random() again, and you get a different number.
- The random function is defined in the random module.

In [ ]:
```
from random import random
for i in range(10):
    value = random()
    print(value)
```

Application: Random Numbers and Simulations

Simulating Die Tosses

- For example, to simulate the throw of a die, you need random integers between 1 and 6.
- Hint: Python provides a separate function for generating a random integer within a given range:
    - randint(a,b)

In [ ]:
```
#  This program simulates tosses of a pair of dice.
from random import randint

for i in range(10) :
   # Generate two random numbers between 1 and 6, inclusive.
   d1 = randint(1, 6)
   d2 = randint(1, 6)

   # Print the two values.
   print(d1, d2)
```

Summary

- while loops
- for loops
- while loops are very commonly used (general purpose)
- Use 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.