

~ IT104 – INTRO TO PROGRAMMING ~ 1
PROF. HALTON 678-687-6104 MHALTON@ITTTECH.EDU

::UNIT 9 HANDOUT::

Unit 9: Repetitive Processing II

Objectives

6. Create computer programs that can do repetitive processing.
 - 6.5: Evaluate the counter-controlled For loops.
 - 6.6: Use sentinel values in creating computer programs.
 - 6.7: Evaluate nested loops.

Readings

Starting Out with Programming Logic & Design. Pearson Custom Publishing, 2008.

- Chapter 5, pp. 183-211

Key Concepts That Must Be Covered in Class

The following key concepts must be covered in class in order to achieve the course objectives.

1. For loops
2. Sentinel values
3. Nested loops
4. Designing comprehensive program software with input validation, decision making, and repetition

Teaching Tips for This Unit

- Review the usefulness of counter-controlled repetition structures for programs that require a set number of reiterations of a process. Introduce the concept of a **For loop** using a counter variable. Review the three-step process listed on p. 184 of the textbook that a For loop iteration goes through. Use Figure 5-12 on p. 184 and Figure 5-13 on p. 185 to illustrate the For loop process.
- Review the pseudocode for a For loop using the sample code found on p. 186 of the textbook. Discuss the “Hello world” program as illustrated in Program 5-8 on p. 186 and Figure 5-14 on p. 187. Present what the “Hello world” program could look like implemented in Python, as follows:

```
for counter in range(5):  
    print 'Hello World'
```

Note that just like a While loop in Python, indentation of the loop body is essential because there is no End For statement. Illustrate another example

::UNIT 9 HANDOUT::

of the same loop in Python specifying a starting value for the range function:

```
for counter in range(1,6):  
    print 'Hello World'
```

Note that when a starting value is specified in range(), the loop ends just before the ending value. Finally, illustrate that this loop could also have been written by explicitly specifying the counter values:

```
for counter in [1,2,3,4,5]:  
    print 'Hello World'
```

Note that all three versions of the Python code are equivalent, but the later versions give the programmer increasingly more control over the counter values.

- Consider looping by increments other than one in a For loop. Remind students that this is useful for such tasks as processing every other item in a list or counting backwards. Review the basic structure of the pseudocode For loop that uses a “Step” option as illustrated in the pseudocode at the bottom of p. 190 and Program 5-10 on p. 191 of the textbook. Review the pseudocode to decrement the counter (see example at the bottom of p. 193). Illustrate the Python version of the countdown example:

```
for counter in range(10,0,-1):  
    print counter
```

- Use Program 5-18 on pp. 202-203, Figure 5-18 on p. 202, and Figure 5-19 on p. 204 of the textbook as illustrations of some of the applications of For loops. Note that although the example uses a For loop, a While loop could also be used. As a review of While loops, get the students to convert the For loop of Program 5-19 on pp. 206-207 into a While loop. Note that for any situation that a For loop is used, a While loop can be used to replace it.
- Consider the concept of sentinel values for use in exiting While loop iterations. **Important!!!** sentinel values must be unique and must never be a

::UNIT 9 HANDOUT::

regular value. Use the property tax program, Program 5-19 on pp. 206-207, and Figure 5-20 on p. 208 of the textbook to illustrate the use of a sentinel value.

- Revisit the concept of nested loops. As this is the most complex topic that the students have come across so far, take extra time to discuss this if necessary. Use the pseudocode on p. 209 and Figure 5-21 on p. 210 of the textbook to trace the operation of the clock simulator program. Note that nested loops can be any combination of While and For loops. If time permits, convert some of the For loops into While loops in the clock simulator program to illustrate the point.
- **Important!!!** The students will present their Comprehensive Lab Practicum in the next unit.

9 — Repetitive Processing II and Intro to Object oriented Programming

Read from *A Byte of Python*, Chapter 11

Read from *Starting Out with Programming Logic & Design*, Chapter 5,

“Repetition Structures,” pp. 183-211

- **Lab: 9.1 Create a Class (optional)**
- **Lab: 9.2 Repetition Structures (optional)**
- **Lab 9.3 Practicum Work**
- **Assignment: 9.1** Answer the following questions from Chapter 5 of *Starting Out with Programming Logic & Design*:
 - Multiple Choice Review Questions 6, 9, and 10 starting on p. 211
 - Algorithm Workbench Review Questions 3, 4, 6, 9, and 10 starting on p. 213

Submit your answers in a word-processed document to your instructor in Unit 10.

Lab 9.1: Creating a class (Optional)

(Derived from [Al Lukaszewski](#), for About.com)

Task 1: We can create a [class](#) called Felicitations:

```
import re #python built-in functions
```

~ IT104 - INTRO TO PROGRAMMING ~ **4**
PROF HALTON 678-687-6104 MHALTON@ITTTECH.EDU

::UNIT 9 HANDOUT::

```
import string #Python built-in functions

listed = ['hello', 'Cheerful coder', ',']

greeting = listed[0]
addressee = listed[1]
punctuation = listed[2]

listed = ['hello', 'Cheerful coder', ',']

greeting = listed[0]
addressee = listed[1]
punctuation = listed[2]
class Felicitations(object):

    def __init__(self):

        self.felicitations = [ ]

def addon(self, word):

    self.felicitations.append(word)

def printme(self):

    greeting = string.join(self.felicitations[0:],
    "") ## be careful here - this was a line-wrap, and
```

::UNIT 9 HANDOUT::

```
## not a return ( remember that # is a  
## comment, not a command )  
  
print greeting
```

The class is based off of another type of object called 'object'. The first [method](#) is mandatory if you want the object to know anything about itself (instead of being a brainless mass of functions and variables; the class must have a way of referring to itself. The second method simply adds the value of **word** to the Felicitations object. Finally, this class has the ability to print itself via a method call

Task 2: Now, we can make a function which calls the last method of the class:

```
def prints(string):  
  
    string.printme()  
  
    return
```

Task 3: Next, we define two more functions. These illustrate how to pass arguments to and how to receive output from functions. The strings in parentheses are arguments on which the function depends. The value returned is signified in the 'return' statement at the end.

```
def hello(i):  
  
    string = "hell" + i  
  
    return string  
  
def caps(word):  
  
    value = string.capitalize(word)  
  
    return value
```

::UNIT 9 HANDOUT::

The first of these functions takes an argument 'i' which is later concatenated on to the base 'hell' and returned as a variable named 'string'. As we shall see in the main() function, this variable is hardwired in the program as 'o', but you could easily make it user-defined by using sys.argv[3] or similar.

The second function is used to capitalize the parts of the output. It takes one argument, the phrase to be capitalized, and returns it as a value 'value'

Task 4: Define a main() function:

```
def main():
    salut = Felicitations()
    if greeting != "Hello":
        cap_greeting = caps(greeting)
    else:
        cap_greeting = greeting
    salut.addon(cap_greeting)
    salut.addon(", ")

    cap_addressee = caps(addressee)
    lastpart = cap_addressee + punctuation
    salut.addon(lastpart)

    prints(salut)
```

Several things happen in this function:

1. We create an instance of the 'Felicitations' class and call it 'salut'. This allows us to access the parts of 'Felicitations' as they exist in 'salut'.
2. Next, if 'greeting' does not equate to the string "Hello", then, using function **caps()** we capitalize the value of 'greeting' and assign it to 'cap_greeting'. Otherwise, 'cap_greeting' is assigned the value of 'greeting'. If this seems tautological, it is, but it is also illustratory of conditional statements in Python.
3. Whatever the outcome of the if...else statements, the value of 'cap_greeting' is added onto the value of 'salut', using class **object's** **append** method.
4. We then append a comma and a space to salut in preparation for the addressee.

::UNIT 9 HANDOUT::

5. Next, the value of 'addressee' is capitalized and assigned to 'cap_addressee'.
6. The values of 'cap_addressee' and 'punctuation' are then concatenated and assigned to 'lastpart'.
7. The value of 'lastpart' is then appended to the content of 'salut'.
8. Finally, the object 'salut' is sent to the 'prints' function to be printed to the screen

Task 5: Alas, we are not done yet. If the program were executed now, it would end with no output whatsoever. This is because the function main() is never called. Here is how to call main() when the program is executed:

```
if __name__ == '__main__': main()
```

Save the program as 'hello.py' (without the quotes). Now, you can start the program. Assuming the Python interpreter is in your execution path, you can type

```
"python hello.py hello world !" ## In Idle, just  
## click 'run'  
## run module
```

and you will be rewarded with the familiar output:

```
Hello...
```

Lab 9.2: Repetition Structures (Optional)

What is the purpose?

In this lab, you will practice designing condition-controlled and counter-controlled loop structures using While and For loops. You will design While and For loop structures using pseudocode and flowcharts, and then implement them in Python.

What are the steps?

- Task 1: Lab 6.1—For Loop and Pseudocode
Procedure

1. Complete Lab 6.1—For Loop and Pseudocode on pp. 99-102 of the Lab Manual to Accompany Starting Out with Programming Logic & Design.

::UNIT 9 HANDOUT::

2. Submit your work to your instructor for grading.
- Task 2: Lab 6.2—For Loop and Flowcharts
Procedure
 1. Complete Lab 6.2—For Loop and Flowcharts on pp. 103-108 of the Lab Manual to Accompany Starting Out with Programming Logic & Design.
 2. Submit your flowchart to your instructor for grading.
 - Task 3: Lab 6.3—Python Code
Procedure
 1. Complete Lab 6.3—Python Code on pp. 109-113 of the Lab Manual to Accompany Starting Out with Programming Logic & Design.
 2. Submit your completed code to your instructor for grading.
 - Task 4: Lab 6.4—Programming Challenge 1: Average Test Scores
Procedure
 1. Complete Lab 6.4—Programming Challenge 1: Average Test Scores on p. 115 of the Lab Manual to Accompany Starting Out with Programming Logic & Design.

Submit your work to your instructor for grading.

Lab: 9.3 Practicum Work

Keep working on your project. Next week is your Presentation. Remember to dress nicely next week, and pretend you are presenting your project to a client.