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
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: ")
```
Average Example

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

```python
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
```
Counting Matches (e.g., Negative Numbers)

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

```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.")
```
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

```python
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

```python
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

```python
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

```python
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

• Open 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
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
while loop -> for loop

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

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

stateName = "Virginia"
for letter in stateName :
    print(letter)  # for version
```
while loop -> 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.

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

```python
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.

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

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

9/21/16
Syntax of a **for** Statement (Container)

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

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

- 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.
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.

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

```
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
```
Planning a **for** Loop

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

<table>
<thead>
<tr>
<th>Year</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10500.00</td>
</tr>
<tr>
<td>2</td>
<td>11025.00</td>
</tr>
<tr>
<td>3</td>
<td>11576.25</td>
</tr>
<tr>
<td>4</td>
<td>12155.06</td>
</tr>
<tr>
<td>5</td>
<td>12762.82</td>
</tr>
</tbody>
</table>

```python
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

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

```
# 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))
```
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.

```python
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.

```python
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) :
```
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.
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:

```python
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
Nested Loops
Loops Inside of Loops

• In Chapter Three 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 $10$

<table>
<thead>
<tr>
<th>$x^1$</th>
<th>$x^2$</th>
<th>$x^3$</th>
<th>$x^4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>27</td>
<td>81</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>1000</td>
<td>10000</td>
</tr>
</tbody>
</table>
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

\[\begin{array}{cccc}
  x^1 & x^2 & x^3 & x^4 \\
  1 & 1 & 1 & 1 \\
  2 & 4 & 8 & 16 \\
  3 & 9 & 27 & 81 \\
  \ldots & \ldots & \ldots & \ldots \\
  10 & 100 & 1000 & 10000 \\
\end{array}\]
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

<table>
<thead>
<tr>
<th>n</th>
<th>$x^1$</th>
<th>$x^2$</th>
<th>$x^3$</th>
<th>$x^4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>27</td>
<td>81</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>1000</td>
<td>10000</td>
<td></td>
</tr>
</tbody>
</table>
Flowchart of a Nested Loop

x = 1

x <= 10?

n = 1

n <= 4?

Print $x^n$

n = n + 1

Print new line

x = x + 1

Done

Inner Loop
The `end=""` suppresses the new line, so the numbers are all printed on the same line.
The Results

```
[evaluate Powertable header.py]

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>27</td>
<td>81</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>64</td>
<td>256</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td>125</td>
<td>625</td>
</tr>
<tr>
<td>6</td>
<td>36</td>
<td>216</td>
<td>1296</td>
</tr>
<tr>
<td>7</td>
<td>49</td>
<td>343</td>
<td>2401</td>
</tr>
<tr>
<td>8</td>
<td>64</td>
<td>512</td>
<td>4096</td>
</tr>
<tr>
<td>9</td>
<td>81</td>
<td>729</td>
<td>6561</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>1000</td>
<td>10000</td>
</tr>
</tbody>
</table>
```
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>
<thead>
<tr>
<th>Nested Loops</th>
<th>Output</th>
<th>Explanation</th>
</tr>
</thead>
</table>
| ```py for i in range(3):  
  for j in range(4):  
    print("*", end="")  
  print()```             | ****       | Prints 3 rows of 4 asterisks each.                    |
| ```py for i in range(4):  
  for j in range(3):  
    print("*", end="")  
  print()```             | ***        | Prints 4 rows of 3 asterisks each.                    |
| ```py for i in range(4):  
  for j in range(i + 1):  
    print("*", end="")  
  print()```             | *          | Prints 4 rows of lengths 1, 2, 3, and 4.             |
Hand Tracing the Loop

```python
for i in range(4):
    for j in range(i + 1):
        print("*", end="")
    print()
```

• i will have the values:
  • 0, 1, 2, 3 – So we will have four lines of stars

• j will have the values
  • 0 - So we will have one star
  • 0, 1 - So we will have two stars
  • 0, 1, 2 - So we will have three stars
  • 0, 1, 2, 3 - So we will have four stars
# Nested Loop Examples (2)

## Table 3: Nested Loop Examples

<table>
<thead>
<tr>
<th>Code</th>
<th>Result</th>
</tr>
</thead>
</table>
| ```python
  for i in range(3):
    for j in range(5):
      if j % 2 == 1:
        print("*", end="")
      else:
        print("-", end="")
  print()
``` | -*-*-\n  -*-*-\n  -*-*-  | Prints alternating dashes and asterisks. |
| ```python
  for i in range(3):
    for j in range(5):
      if i % 2 == j % 2:
        print("*", end="")
      else:
        print(" ", end="")
  print()
``` | * * *\n  * *\n  * * *  | Prints a checkerboard pattern. |
Second Problem Statement

• Print the following pattern of brackets:
  [[]][][]
  [[]][][]
  [[]][][]
  [[]][][]

• The pattern consists of:
  • Three rows
  • Each row has four pairs of brackets

• What do we know?
  • We need two nested loops
    • The first loop (the outer loop) will print each of the three rows
    • The second loop (the inner loop) will print the four pairs of brackets
Pseudocode Code, Results

For $i = 1$ to $3$
  For $j = 1$ to $4$
    Print "[]"
  Print a new line

```python
for i in range(3):
    for j in range(4):
        print("["]", end="""
    print()
```

[evaluate nested loop example three.py]
Exam Averages Problem Statement

• It is common to repeatedly read and process multiple groups of values:
  • Write a program that can compute the average exam grade for multiple students.
  • Each student has the same number of exam grades
  • Prompt the user for the number of exams
  • When you finish a student prompt the user to see if there are more students to process

• What do we know?
• What do we need to compute?
• What is our algorithm / approach?
Step One: Understand the Problem

- To compute the average grade for a student, we must read and tally all of the grades for that student
  - We can use a loop to do this. *(we have working code to do this portion)*

- We need to compute grades for multiple students
  - That implies a set of nested Loops
    - The outer loop processes each student
    - The inner loop process the student’s grades
Step Two

• Compute the grade for one student
• Set up the variables and loop
• We know how many grades to process, so we can use a count-controlled loop

```
total score = 0
For i in range (1, number of exams + 1) :
    Read the next exam score
    Add the exam score to the total score
Compute the exam average
Print the exam average
```

9/21/16
Step Three

• Repeat the process for each student

• Since we don’t know how many students there are, we will use a while loop with a sentinel value
  • For simplicity we will use “Y” as the sentinel value
Step Four: Translate to Python

```python
##
# This program computes the average exam grade for multiple students.
#
# Obtain the number of exam grades per student.
numExams = int(input("How many exam grades does each student have? "))
#
# Initialize moreGrades to a non-sentinel value.
moreGrades = "Y"
#
# Compute average exam grades until the user wants to stop.
while moreGrades == "Y":
    # Compute the average grade for one student.
    print("Enter the exam grades.")
    total = 0
    for i in range(1, numExams + 1):
        score = int(input("Exam %d: " % i))  # Prompt for each exam grade.
        total = total + score

    average = total / numExams
    print("The average is %.2f" % average)
    # Prompt as to whether the user wants to enter grades for another student.
    moreGrades = input("Enter exam grades for another student (Y/N)? ")
    moreGrades = moreGrades.upper()
```
Exam Averages Example

• Open the file:
  • examaverages.py

• Notice that the second loop (the **for** loop) is nested inside the **while** loop

• In Wing you should see a line (the indent guide) connecting the **for** loop on line 17 down to the statement on line 21
  • The line is showing you the statements that are included in the **for** loop

• If you don’t see the indent guide:
  • Click on the edit tab
  • Select “Preferences...”
  • Under Editor, select Indentation
  • Click the “Show Indent Guides” box
  • Click the Apply button
  • Click the Okay Button
Turning the Indent Guides On
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 $\geq 0$ 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

```python
ch04/dice.py

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)
```

Program Run

| 1 | 5 |
| 6 | 4 |
| 1 | 1 |
| 4 | 5 |
| 6 | 4 |
| 3 | 2 |
| 4 | 2 |
| 3 | 5 |
| 5 | 2 |
| 4 | 5 |