Chapter 3: Branching, While Loops and Program Planning
Objectives

In this Chapter, you will learn how to:

- Generate random numbers using the function `randrange()`
- Use `if` structure to execute code based on a condition
- Use `if – else` structures to make a choice based on a condition
  - execute one of 2 sections of your code based on a condition
- We will learn the `if-elif-else` structure, to make a choice based on several conditions
  - execute only one of several sections of your code based on several conditions
- Use `while` loop structure to repeat parts of your program
- Write pseudocode
Generate random numbers using the function

```
random.randrange()
```

- **randrange()**
  - Argument (input) is an Integer (e.g., 1, 2, 3, 4, 5, 6..)
  - Outputs an Integer value (value returned is always less than the input value)
  - Your program must include an “import random” statement before calling randrange(), Otherwise it will fail

- **import random**

  - Now generate a random number between 0 and 5 inclusive:
    - `My_Number = random.randrange(6)`

**Example.. Craps Roller Program**
if Structure

Syntax

- All *if* structures have a “condition”
  - A condition is an expression that is either *True* or *False*
    - Example: 2 is greater than 1 (or 2 > 1)

- Syntax:

  
  ```
  if 2 > 1:
    print “This is a test.”
  ```

Example: Password Program
if Structure

**Comparison Operators**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>equal to</td>
</tr>
<tr>
<td>!=</td>
<td>not equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
</tr>
</tbody>
</table>
if Structure:

**Indentation**

- Using **Indentations** to create blocks of statements
  - Use one tab (for now) to indent statement(s)
  - All statements in the IF block are executed if the condition in the “if” statement is TRUE

```python
n = 4  # (Note: Variable must be initialized before it is referenced in the if statement)
if n > 1:
    print "This is the first line of the IF Block"
    print "This is the second line of the IF block"
    print "Each line in the block must be indented"
print"This line is not indented; and therefore not part of the ‘if’ block”
raw_input ("Press enter to exit")
```
if – else Structure

Making a Choice

```python
if condition:
    Execute this block (if block) of statement if condition is true
    ...
    ...
    ...
else:
    Execute this block (else block) of statements if condition is false
    ........
    ....
    ...

raw_input ("Program resumes here because the statement is not intended")
```

Granted or Denied Program
if – *elif*-else Structure (*Multiple Choices*)

**if** condition1:

If this condition is true, execute block1 and exit the “If–elif-else” structure

**elif** condition2:

if condition1 is FALSE and condition2 is TRUE, execute block2 and exit the “If–elif-else” structure

**elif** condition3:

If condition1 and condition2 are FALSE, and condition3 is TRUE, execute block3 and exit the “if–elif-else” structure

**elif** condition4:

If condition1, condition2 and condition3 are FALSE, and condition4 is TRUE, execute block4 and “exit the “if–elif-else” structure

**else:**

If all conditions are false, execute this default block and exit the “If–elif-else” structure

The program execution resumes here

Mood Computer Program
While Loops

- **Repeat** one or more similar operations:

- Operations:
  - Single Instruction
    - Example: Adding a finite set of integer numbers:
      - 1+2, 2+3, 3+4, 4+5, 5+6,.....
  - Block(s) of statements
    - Example: Find the average of any set of numbers, and print result:
      - (1.0 + 5.0 + 0.0 + 10.0 )/4.0
While
Syntax

```
while condition:
    Statement 1
    Statement 2
    Statement 3
    Statement 4
```

This statement (#5) is not in the while loop

`while` loop: Repeats Statements 1, 2, 3, and 4 if `condition` is True

When `condition` is False, exit the loop and jump to statement #5

**Must update** `condition` within loop at each iteration until it takes a value FALSE to avoid infinite loop
Creating While Loops

- Example 1:
  - Simulate conversation with a 3-year old child
Infinite Loops: Oops!!

- Avoid Infinite Loops!!
  - Always check the Loop counter (i.e sentry variable) is initialized before its referenced in the loop
  - Check that counter is correctly updated within the Loop
  - Check that the condition on the sentry variable will eventually evaluate to False
Interpreting Numbers as True or False

- Can you evaluate $30 + 2$?
- Now evaluate $30$ as True or False

**Concept:**
- Number $0$ is always False
- All other non-zero Numbers are True

**Program Simulation**
- *Let’s examine the Maitre Program*
Interpreting Strings as True or False

- Can you evaluate "Kent" as True or False?
- Evaluate the empty string: "" as True or False

Concept:
- Empty string is always False
- All other Strings are True

Is “0” True or False?  Is “ ” True or False?
Key Words
True & False

- Think of True as number value 1
- Think of False as the number value 0

Examples

```python
if True:
    print "You are a winner"

if False:
    print "You win"

else:
    print "Better luck next time"
```
The `break` statement

Exit a loop

```python
while True:
    count = count + 1
    if count > 10:
        break
raw_input ("\n\nPress the enter key to exit")
```
The **continue** statement
Jump to top of loop

while True:
    count = count + 1
    if count == 5:
        **continue**
    print count
    raw_input ("\n\nPress the enter key to exit")

Program Simulation: *Let’s examine the Finicky Program*
Compound Conditions

- **Simple Conditions**
  - Score > 90
  - Score ≤ 100

- **Compound Conditions:**
  - Score is in the range 90 and 100 inclusive
Create Compound Conditions

Use Logical Operators

- not
- and
- or
Compound Statement
Using the **not** Logical Operator

```python
username = ""
while *not* username:
    username = raw_input("Enter Username: ")
```

Let \( A \) be an expression or a Variable name, then:
**not** \( A \) is **True** when \( A \) is **False** and vice-versa.
Compound Statement

Using the **and** Logical Operator

score is in the range 90 and 100 inclusive

\[
\text{score} \geq 90 \text{ and } \text{score} \leq 100
\]

Simulation Program: Exclusive Network

Username == “S.Meier”

password == “civilization”

username == “S.Meier” and password == “civilization”

<table>
<thead>
<tr>
<th>True</th>
<th>True</th>
<th>True</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>
# Compound Statement

## Using the **or** Logical Operator

Major equals math or major equals computer science

`major == "math" or major == "computer science"`

<table>
<thead>
<tr>
<th>Username == “guest”</th>
<th>password == “guest”</th>
<th>username == “guest” or password == “guest”</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>