Objectives

- To understand the programming pattern simple decision and its implementation using a Python if statement.
- To understand the programming pattern two-way decision and its implementation using a Python if–else statement.
Objectives (cont.)

- To understand the programming pattern multi-way decision and its implementation using a Python if–elif–else statement.
- To understand the idea of exception handling and be able to write simple exception handling code that catches standard Python run-time errors.

Objectives (cont.)

- To understand the concept of Boolean expressions and the bool data type (requires Python 2.3 and newer)
- To be able to read, write, and implement algorithms that employ decision structures, including those that employ sequences of decisions and nested decision structures.
Simple Decisions

- So far, we’ve viewed programs as sequences of instructions that are followed one after the other.
- While this is a fundamental programming concept, it is not sufficient in itself to solve every problem. We need to be able to alter the sequential flow of a program to suit a particular situation.

Simple Decisions

- Control structures allow us to alter this sequential program flow.
- In this chapter, we’ll learn about decision structures, which are statements that allow a program to execute different sequences of instructions for different cases, allowing the program to “choose” an appropriate course of action.
Example:
Temperature Warnings

Let’s return to our Celsius to Fahrenheit temperature conversion program from Chapter 2.

```python
# convert.py
# A program to convert Celsius temps to Fahrenheit
# by: Susan Computewell

def main():
    celsius = input("What is the Celsius temperature? ")
    fahrenheit = 9.0 / 5.0 * celsius + 32
    print("The temperature is", fahrenheit, "degrees Fahrenheit.")
main()
```

Example:
Temperature Warnings

Let’s say we want to modify that program to print a warning when the weather is extreme.

- Any temperature over 90 degrees Fahrenheit and lower than 30 degrees Fahrenheit will cause a hot and cold weather warning, respectively.
Example:
Temperature Warnings

- Input the temperature in degrees Celsius (call it celsius)
- Calculate fahrenheit as 9/5 celsius + 32
- Output fahrenheit
- If fahrenheit > 90
  print a heat warning
- If fahrenheit > 30
  print a cold warning

Example:
Temperature Warnings

- This new algorithm has two decisions at the end. The indentation indicates that a step should be performed only if the condition listed in the previous line is true.
Example:
Temperature Warnings

```python
# convert2.py
#   A program to convert Celsius temps to Fahrenheit.
#   This version issues heat and cold warnings.

def main():
    celsius = input("What is the Celsius temperature? ")
    fahrenheit = 9.0 / 5.0 * celsius + 32
    print("The temperature is", fahrenheit, "degrees fahrenheit."
    if fahrenheit >= 90:
        print("It's really hot out there, be careful!")
    if fahrenheit <= 30:
        print("Brrrrr. Be sure to dress warmly")

main()
```
Example:
Temperature Warnings

- The Python if statement is used to implement the decision.

```python
if <condition>:
    <body>
```

- The body is a sequence of one or more statements indented under the if heading.

Example:
Temperature Warnings

- The semantics of the if should be clear.
  - First, the condition in the heading is evaluated.
  - If the condition is true, the sequence of statements in the body is executed, and then control passes to the next statement in the program.
  - If the condition is false, the statements in the body are skipped, and control passes to the next statement in the program.
Example: Temperature Warnings

- The body of the if either executes or not depending on the condition. In any case, control then passes to the next statement after the if.
- This is a one-way or simple decision.
Forming Simple Conditions

- What does a condition look like?
- At this point, let’s use simple comparisons.
- `<expr> <relop> <expr>`
- `<relop>` is short for relational operator

<table>
<thead>
<tr>
<th>Python</th>
<th>Mathematics</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;</code></td>
<td><code>&lt;</code></td>
<td>Less than</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td><code>&lt;=</code></td>
<td>Less than or equal to</td>
</tr>
<tr>
<td><code>==</code></td>
<td><code>=</code></td>
<td>Equal to</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td><code>&gt;=</code></td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td><code>&gt;</code></td>
<td>Greater than</td>
</tr>
<tr>
<td><code>!=</code></td>
<td><code>≠</code></td>
<td>Not equal to</td>
</tr>
</tbody>
</table>
Forming Simple Conditions

- Notice the use of `==` for equality. Since Python uses `=` to indicate assignment, a different symbol is required for the concept of equality.
- A common mistake is using `=` in conditions!

---

Forming Simple Conditions

- Conditions may compare either numbers or strings.
- When comparing strings, the ordering is lexicographic, meaning that the strings are sorted based on the underlying ASCII codes. Because of this, all upper-case letters come before lower-case letters. (“Bbbb” comes before “aaaa”)
Forming Simple Conditions

- Conditions are based on Boolean expressions, named for the English mathematician George Boole.
- When a Boolean expression is evaluated, it produces either a value of true (meaning the condition holds), or it produces false (it does not hold).
- Some computer languages use 1 and 0 to represent “true” and “false”.

Boolean conditions are of type `bool` and the Boolean values of true and false are represented by the literals `True` and `False`.

```python
>>> 3 < 4
True
>>> 3 * 4 < 3 + 4
False
>>> "hello" == "hello"
True
>>> "Hello" < "hello"
True
```
Two-Way Decisions

Let's look at the quadratic program as we left it.

```python
# quadratic.py
# A program that computes the real roots of a quadratic equation.
# Illustrates use of the math library.
# Note: This program crashes if the equation has no real roots.

import math  # Makes the math library available.

def main():
    print("This program finds the real solutions to a quadratic")
    print()
    a, b, c = input("Please enter the coefficients (a, b, c): ")
    disRoot = math.sqrt(b * b - 4 * a * c)
    root1 = (-b + disRoot) / (2 * a)
    root2 = (-b - disRoot) / (2 * a)
    print
    print("The solutions are!", root1, root2)

main()
```

Two-Way Decisions

As the comment implies, when $b^2-4ac < 0$, the program tries to take the square root of a negative number, and then crashes.

This program finds the real solutions to a quadratic

```
Please enter the coefficients (a, b, c): 1,1,2
Traceback (most recent call last):
  File "C:\Documents and Settings\Terry\My Documents\Teaching\W04\CS 120\Textbook\code\chapter3\quadratic.py", line 21, in <module>
    main()
  File "C:\Documents and Settings\Terry\My Documents\Teaching\W04\CS 120\Textbook\code\chapter3\quadratic.py", line 14, in main
    disRoot = math.sqrt(b * b - 4 * a * c)
ValueError: math domain error
```
Two-Way Decisions

- We can check for this situation. Here's our first attempt.

```python
# quadratic.py
# A program that computes the real roots of a quadratic equation.
# Bad version using a simple if to avoid program crash

import math

def main():
    print "This program finds the real solutions to a quadratic\n"

    a, b, c = input("Please enter the coefficients (a, b, c): ")
    discrim = b * b - 4 * a * c
    if discrim >= 0:
        discRoot = math.sqrt(discrim)
        root1 = (-b + discRoot) / (2 * a)
        root2 = (-b - discRoot) / (2 * a)
        print "The solutions are:", root1, root2
```

Two-Way Decisions

- We first calculate the discriminant \(b^2-4ac\) and then check to make sure it's nonnegative. If it is, the program proceeds and we calculate the roots.

- Look carefully at the program. What's wrong with it? Hint: What happens when there are no real roots?
Two-Way Decisions

- This program finds the real solutions to a quadratic

Please enter the coefficients (a, b, c): 1,1,1

- This is almost worse than the version that crashes, because we don’t know what went wrong!

Two-Way Decisions

- We could add another if to the end:
  ```python
  if discrim < 0:
      print "The equation has no real roots!"
  ```

- This works, but feels wrong. We have two decisions, with mutually exclusive outcomes (if discrim >= 0 then discrim < 0 must be false, and vice versa).
Two-Way Decisions

- In Python, a two-way decision can be implemented by attaching an `else` clause onto an `if` clause.
- This is called an `if-else` statement:
  ```python
  if <condition>:
      <statements>
  else:
      <statements>
  ```
Two-Way Decisions

- When Python first encounters this structure, it first evaluates the condition. If the condition is true, the statements under the `if` are executed.
- If the condition is false, the statements under the `else` are executed.
- In either case, the statements following the `if-else` are executed after either set of statements are executed.

```python
# quadratic3.py
# A program that computes the real roots of a quadratic equation.
# Illustrates use of a two-way decision

import math

def main():
    print "This program finds the real solutions to a quadratic equation."
    a, b, c = input("Please enter the coefficients (a, b, c): ")
    discrim = b * b - 4 * a * c
    if discrim < 0:
        print "The equation has no real roots!"
    else:
        discriminant = math.sqrt(b * b - 4 * a * c)
        root1 = (-b + discriminant) / (2 * a)
        root2 = (-b - discriminant) / (2 * a)
        print "The solutions are:\", root1, root2

main()
```

Two-Way Decisions

---

Python Programming, 1/e

31

Python Programming, 1/e

32
Two-Way Decisions

>>> This program finds the real solutions to a quadratic

Please enter the coefficients (a, b, c): 1,1,2

The equation has no real roots!

>>> This program finds the real solutions to a quadratic

Please enter the coefficients (a, b, c): 2, 5, 2

The solutions are: -0.5 -2.0