THE UNIVERSITY OF WESTERN AUSTRALIA

Lecture 7
Decisions

Objectives of this Lecture

- A little revision
- To understand the conditional (decision) statement
  - *if*
  - *if-else*
- *Comparison operators*
- *Logical operators*

Revision: Accumulator Algorithm

The general form of an accumulator algorithm looks like:

- *Initialize the accumulator variable*
- *Perform computation*
  (e.g., in case of factorial multiply by the next smaller number)
- *Update accumulator variable*
- *Loop until final result is reached*
  (e.g., in case of factorial the next smaller number is 1)
- *Output accumulator variable*

This is called a Pattern, or Software Design Pattern. That is, a recurring, reusable generalised set of instructions

Decision making

Example: Quadratic Equation

- The formula for computing the roots of a quadratic equation of the form: $ax^2 + bx + c = 0$

  is

$$root = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- The only part of this that we don't know how to do yet is the square root function.
  - *How could you use ** to calculate a square root?*

Algorithm: Roots of quadratic equation

- Print an introduction
- Input the coefficients (`a, b and c`)
- Calculate roots:

$$root = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- Output the roots

Code: Roots of quadratic equation

```
# quadratic.py
#    A program that computes the real roots of a quadratic equation.
#    Author: Unit Coordinator


def main():
    print("This program finds the roots of a quadratic  equation")
    print()

    a = float(input("Please enter coefficient a: "))
    b = float(input("Please enter coefficient b: "))
    c = float(input("Please enter coefficient c: "))

    discRoot = (b * b - 4 * a * c)**(1/2)
    root1 = (-b + discRoot) / (2 * a)
    root2 = (-b - discRoot) / (2 * a)

    print()
    print("The solutions are:", root1, root2 )
    return # optional to write it if function is returning nothing
```

Code: Roots of quadratic equation

Running the program

```
>>> main()
This program finds the roots of a quadratic equation
Please enter coefficient a: 3
Please enter coefficient b: 4
Please enter coefficient c: -1

The solutions are: 0.215250437022 -1.54858377035
```

Code: Roots of quadratic equation

Try again with 1, 2, 3

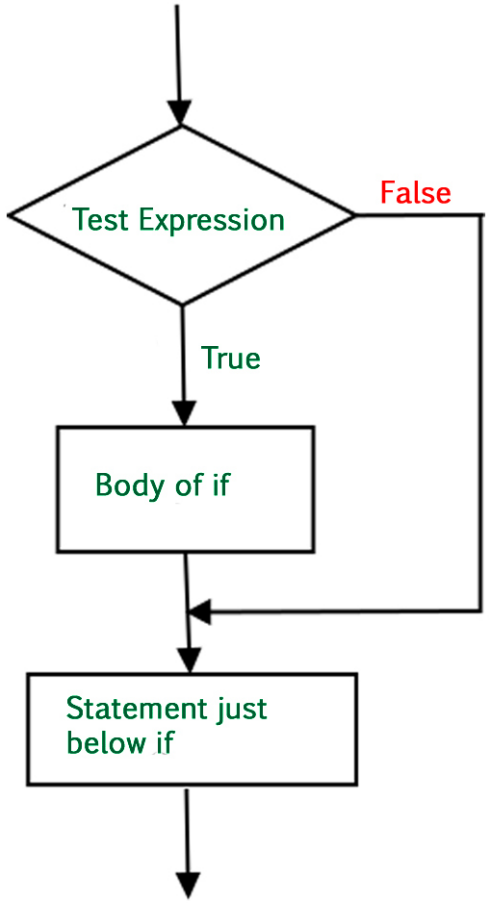
```
>>> main()
This program finds the roots of a quadratic equation
Please enter coefficient a: 1
Please enter coefficient b: 2
Please enter coefficient c: 3

The solutions are: (-
  0.9999999999999999+1.4142135623730951j) (-1-
  1.4142135623730951j)
```

If a = 1, b = 2, c = 3, then we are trying to take the square root of a negative number.

Decision Structures

- 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 *if* statements

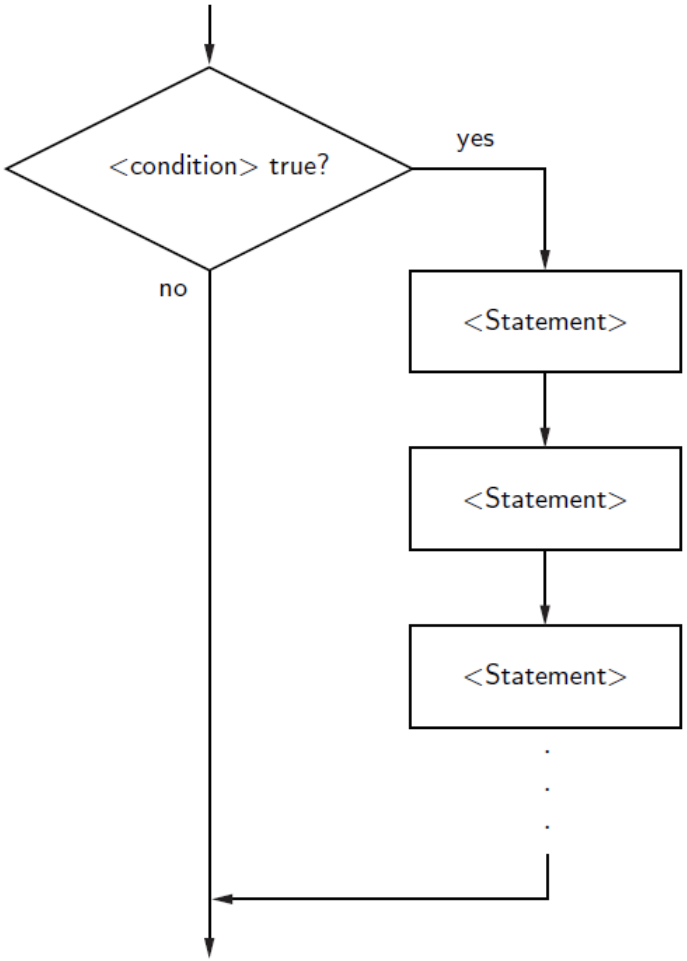
- if <condition> : Don't forget the ":"

  <statements to execute if condition is True>

- The condition is a Boolean expression, i.e., evaluates to values `True` or `False`

- The condition statement is evaluated;
  - *If it evaluates to `True`, the (indented) statements in the body are executed;*
  - *Otherwise, execution proceeds to next statement*

Simple *if* statements



Boolean Expressions - Comparisons

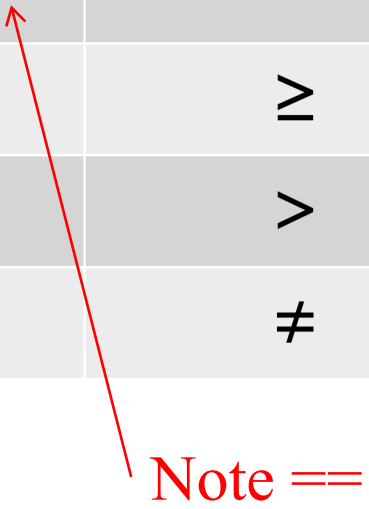
- What does a Boolean expression, i.e., condition, look like?
- At this point, let's use simple comparisons.

  <expr> <relop> <expr>

  – *<relop> is short for relational/comparison operator*

Comparison operators

Python	Mathematics	Meaning
<	<	Less than
<=	≤	Less than or equal to
==	=	Equal to
>=	≥	Greater than or equal to
>	>	Greater than
!=	≠	Not equal to



Comparison Operators

```
>>> 8 < 10
True
>>> 8 > 10
False
```

- 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 comparisons!

```
>>> 8 == 10
False
```

Forming Comparisons

- When comparing strings, the ordering is *lexicographic*, meaning that the strings are sorted based on the underlying Unicode.
  - *Unicode (and before that, ASCII) is a way of representing characters as integers*
  - *Because of this, all upper-case Latin letters come before lower-case letters. ("Bbbb" comes before "aaaa")*

```
>>> "Hello" < "hello"
True
```

Logical/Boolean operators

Operation	Meaning
not	Inverse the comparison result e.g. not x return True if x is False or vice versa
and	Returns True only if both inputs are True e.g. x and y return True only when x is True and y is True else it return False
or	Returns False only if both inputs are False e.g. x and y return False only when x is False and y is False else it return True.

Logical operators are used to combine comparisons

Logical operator : `and`

- The `and` of two Boolean expressions is `True` exactly when both of the Boolean expressions are `True`.
- We can represent this in a *truth table*.
- *P* and *Q* represents Boolean expressions.
- Since each Boolean expression has two possible values, there are four possible combinations of values.

P	*Q*	*P and Q*
T	T	T
T	F	F
F	T	F
F	F	F

Logical operator : `or`

- The `or` of two Boolean expressions is `True` when either Boolean expression is true.
- The only time or is false is when both Boolean expressions are `False`.
- Also, note that `or` is `True` when both Boolean expressions are `True`.

P	*Q*	*P or Q*
T	T	T
T	F	T
F	T	T
F	F	F

Logical operator : `not`

- The `not` operator computes the opposite of a Boolean expression.
- `not` is a *unary* operator, meaning it operates on a single Boolean expression.

P	*not P*
T	F
F	T

- We can put these operators together to make arbitrarily complex Boolean expressions.
- The interpretation of the expressions relies on the precedence rules for the operators.

Example: Temperature Warnings

Design

Input: A value representing a Celsius temperature

Process: (None)

Output:

If temperature greater than 40 print warning

If temperature less than 1 print warning

Simple *if* statements: Example

```
>>> def stayhome():
        temp = float(input("What is the temperature today? "))
        if temp >= 40:
            print(“The temperature is too high”)
            print(“Stay at home")
        if temp <= 0 :
            print(“The temperature is too low”)
            print(“Stay at home")
        print("Have a good day”)

>>> stayhome()
What is the temperature today? 42
The temperature is too high
Stay at home
Have a good day
```

What happens for 36?

Simple *if* statements: Example

```
>>> def stayhome():
        temp = float(input("What is the temperature today? "))
        if temp >= 40 or temp <= 0:
            print(“The temperature is not appropriate”)
            print(“Stay at home")
        print("Have a good day”)

>>> stayhome()
What is the temperature today? 36
Have a good day
```

Quadratic Equation Example

```
# quadratic.py
#    A program that computes the real roots of a quadratic equation.
#    Author: Unit Coordinator


def main():
    print("This program finds the roots of a quadratic  equation")
    print()

    a = float(input("Please enter coefficient a: "))
    b = float(input("Please enter coefficient b: "))
    c = float(input("Please enter coefficient c: "))

    discRoot = (b * b - 4 * a * c)**(1/2)
    root1 = (-b + discRoot) / (2 * a)
    root2 = (-b - discRoot) / (2 * a)

    print()
    print("The solutions are:", root1, root2 )
    return # optional to write it if function is returning nothing
```

Quadratic Equation Example

Running the program

```
>>> main()
This program finds the real solutions to a quadratic
Please enter coefficient a: 3
Please enter coefficient b: 4
Please enter coefficient c: -1

The solutions are: 0.215250437022 -1.54858377035
```

Decisions

Noting the comment, when $b^2-4ac < 0$, the program crashes.

```
>>> main()
This program finds the real solutions to a quadratic
Please enter coefficient a: 1
Please enter coefficient a: 2
Please enter coefficient a: 3

The solutions are: (-
  0.9999999999999999+1.4142135623730951j) (-1-
  1.4142135623730951j)
```

Complex roots look unnatural. How to get rid of them?

Decisions

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

```
# quadratic2.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 = float(input("Enter coefficient a: "))
    b = float(input("Enter coefficient b: "))
    c = float(input("Enter coefficient c: "))
    discrim = b * b - 4 * a * c
    if discrim >= 0:
        discRoot = (discrim)**(1/2)
        root1 = (-b + discRoot) / (2 * a)
        root2 = (-b - discRoot) / (2 * a)
        print("\nThe solutions are:", root1, root2)
```

Two-Way Decisions

- We first calculate the discriminant (b^2-$4ac$) and then check to make sure it's non-negative. 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


Enter coefficient a: 1
Enter coefficient b: 1
Enter coefficient c: 1
>>>
```

- This is worse than the earlier version that shows complex number, because we don't know what went wrong or happened!

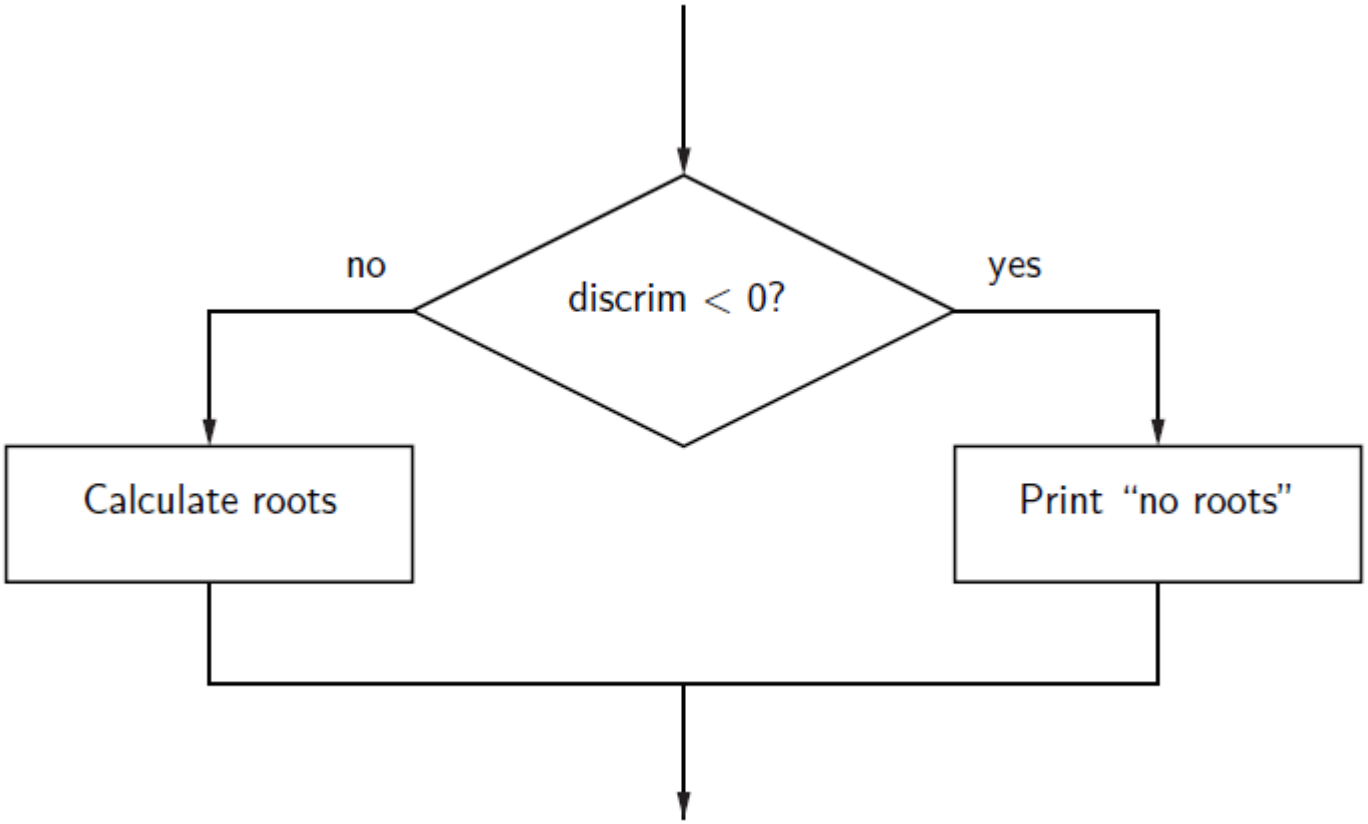
Two-Way Decisions

- We could add another `if` after first `if`:

```
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



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:

```
if <condition>:
    <statements>
else:
    <statements>
```

Two-Way Decisions

- When Python encounters if-else structure, it first evaluates the condition. If the condition evaluates to `True`, the statements under the `if` are executed.
- If the condition evaluates to `False`, the statements under the `else` are executed.
- In either case, the statement following the `if-else` structure is then executed

Two-Way Decisions

```
# 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\n"
    a = float(input("Enter coefficient a: "))
    b = float(input("Enter coefficient b: "))
    c = float(input("Enter coefficient c: "))
    discrim = b * b - 4 * a * c
    if discrim < 0:
        print("\nThe equation has no real roots!")
    else:
        discRoot = (b * b - 4 * a * c)**(1/2)
        root1 = (-b + discRoot) / (2 * a)
        root2 = (-b - discRoot) / (2 * a)
        print ("\nThe solutions are:", root1, root2 )
```

Two-Way Decisions

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

Enter coefficient a: 1
Enter coefficient b: 1
Enter coefficient c: 2

The equation has no real roots!
>>>
This program finds the real solutions to a quadratic

Enter coefficient a: 2
Enter coefficient b: 5
Enter coefficient c: 2

The solutions are: -0.5 -2.0
```

Lecture Summary

- We learned about decision making in computer program
- We learned about boolean expressions and their use in `if` and `if-else` decision statements.
- We learned about Logical and Comparison operators