THE UNIVERSITY OF WESTERN AUSTRALIA

Lecture 7
Decisions

Objectives of this Lecture

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

Revision: Accumulator Algorithm

The general form of an accumulator algorithm looks like:

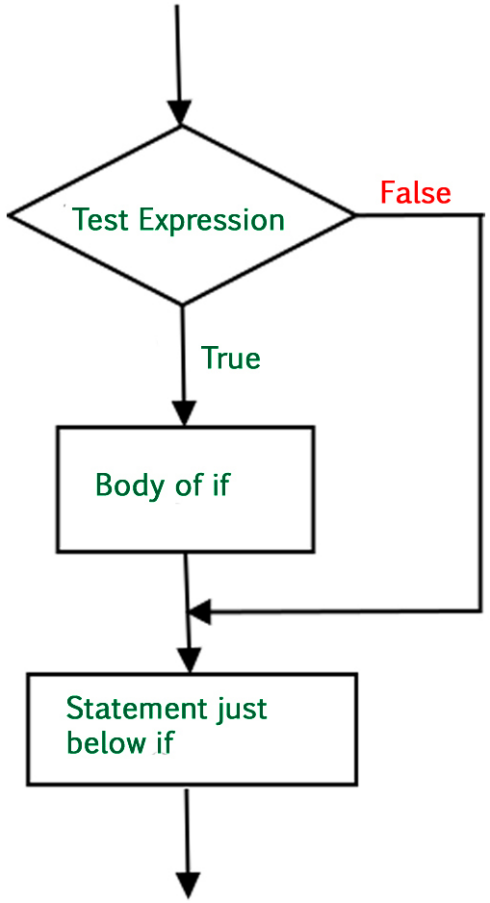
- *Initialize the accumulator variable*
- *Perform computation*
  (e.g. in case of factorial multiply by the next smaller number)
- *Update accumulator variable*
- *Loop until final result is reached*
  (e.g. in case of factorial the next smaller number is 1)
- *Output accumulator variable*

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

Decision making

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> :

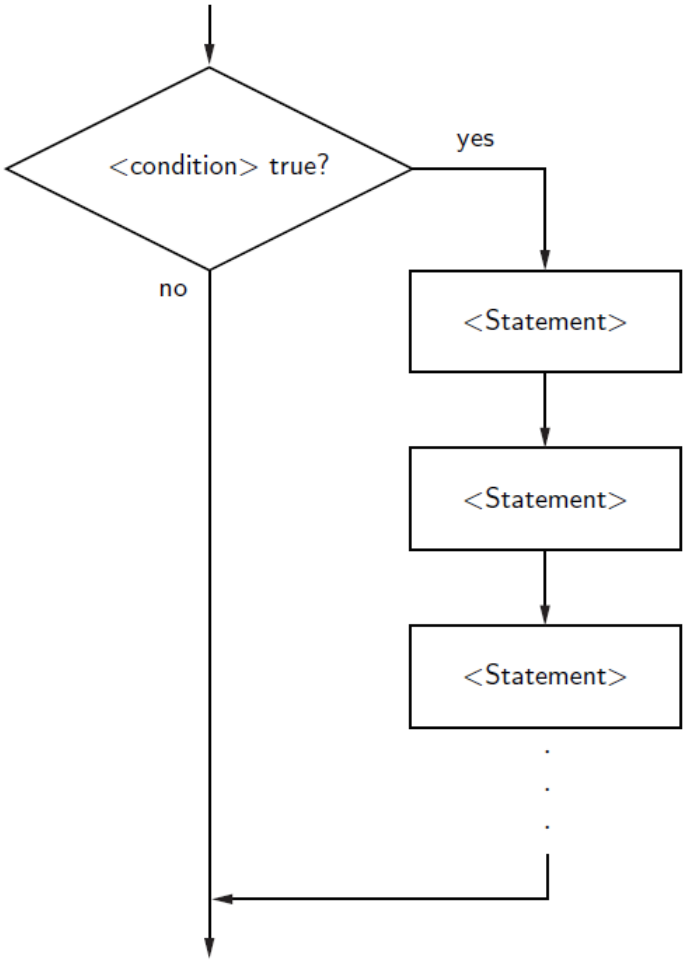
  <statements to execute if condition is True>

Don't forget the ":"

- 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

Note ==

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 operators

- 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 operators

- 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 operators

- 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

Consider the quadratic equation program.

```
# quadratic.py
#    A program that computes the real roots of a quadratic equation.
#    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\n")

    a = float(input("Enter coefficient a: "))
    b = float(input("Enter coefficient b: "))
    c = float(input("Enter coefficient c: "))

    discRoot = math.sqrt(b * b - 4 * a * c)
    root1 = (-b + discRoot) / (2 * a)
    root2 = (-b - discRoot) / (2 * a)

    print("\nThe solutions are:", root1, root2)
```

What does \n do?

Using the Math Library

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

Traceback (most recent call last):
  File "quadratic_roots.py", line 21, in <module>
  main()
  File "quadratic_roots.py", line 15, in main
    discRoot = math.sqrt(b * b - 4 * a * c)
ValueError: math domain error
```

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 = math.sqrt(discrim)
        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 version that crashes, because we don't know what went wrong!
  - *Don't even know that there is a problem*

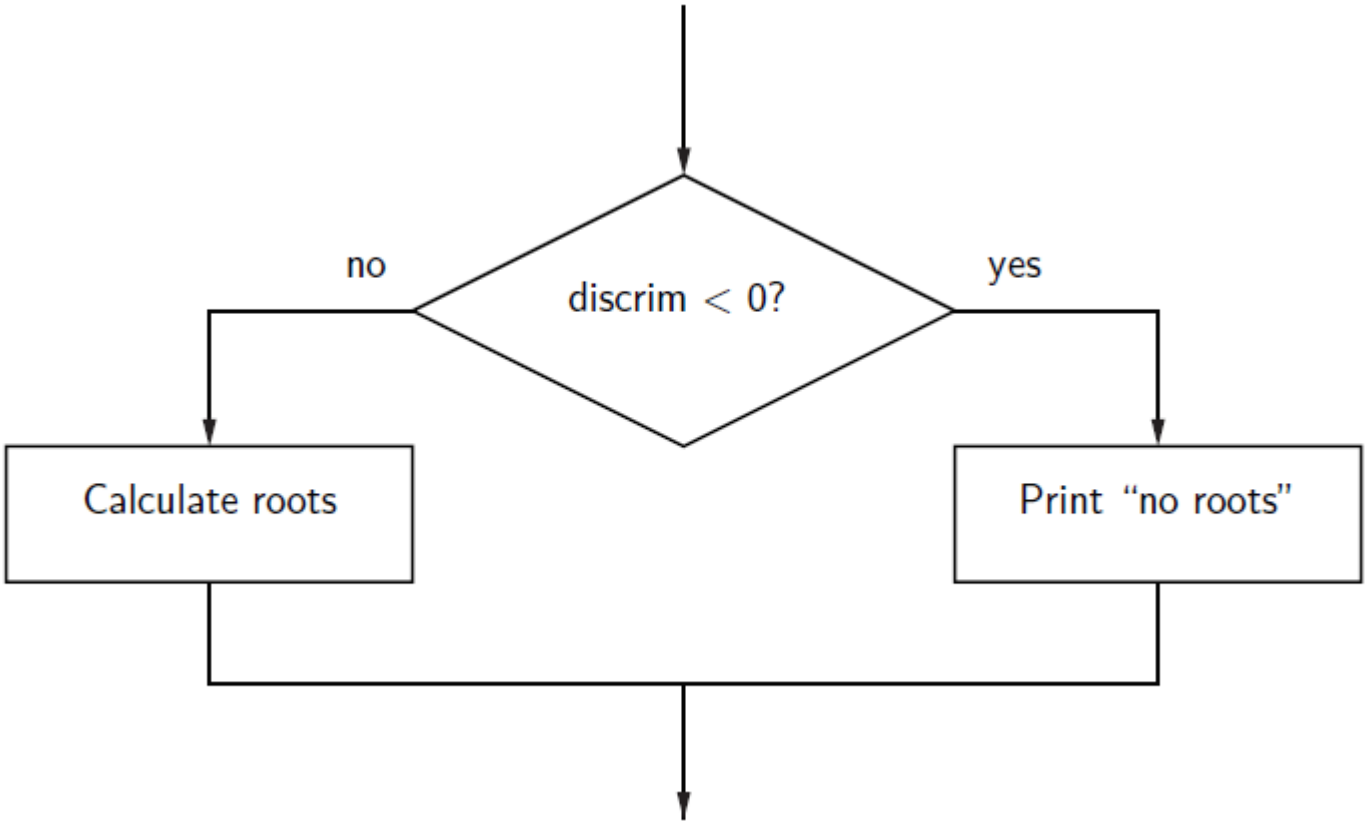
Two-Way Decisions

- 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 = math.sqrt(b * b - 4 * a * c)
        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
```

Two-Way Decisions

The newest program is great, but it still has some quirks!

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


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


The solutions are: -1.0 -1.0
```

Program looks broken, when it isn't

Two-Way Decisions

- While correct, this program output might be confusing for some people.
  - *It looks like it has mistakenly printed the same number twice!*
- A single root occurs when the discriminant is exactly 0, and then the root is $-b/2a$.
- It looks like we need a three-way decision!

Two-Way Decisions

- Check the value of discrim
  when < 0: handle the case of no roots
  when = 0: handle the case of a single root
  when > 0: handle the case of two distinct roots

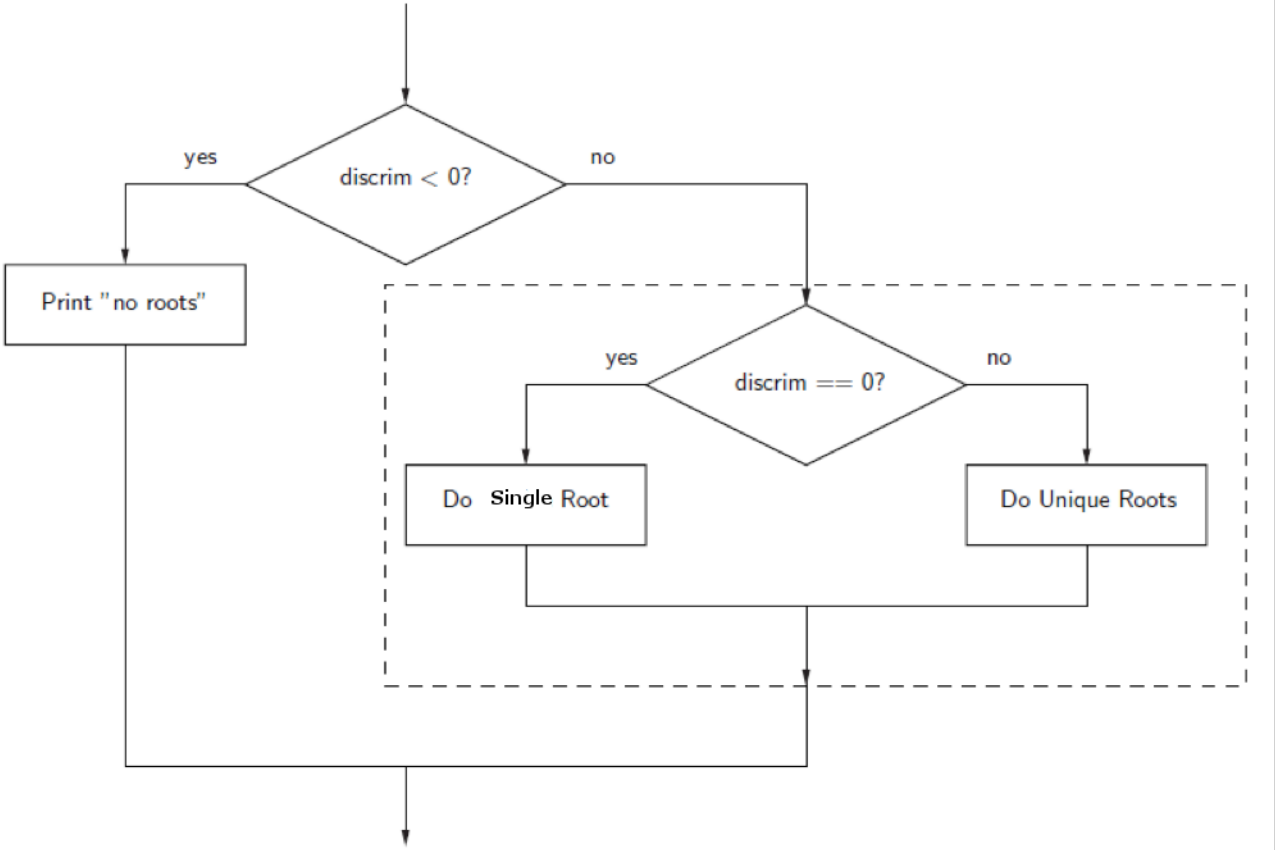
- We can do this with two if-else statements, one inside the other.

- Putting one compound statement inside of another is called *nesting*.

Two-Way Decisions

```
if discrim < 0:
    print("Equation has no real roots")
else:
    if discrim == 0:
        root = -b / (2 * a)
        print("There is a single root at", root)
    else:
        # Do stuff for two roots
```

Nested Two-Way Decisions



Multi-Way Decisions

- Imagine if we needed to make a five-way decision using nesting. The if-else statements would be nested four levels deep!
- There is a construct in Python that achieves this, combining an `else` followed immediately by an `if` into a single `elif`.

Multi-Way Decisions

```
if <condition1>:
   <statements>
elif <condition2>:
   <case2 statements>
elif <condition3>:
   <case3 statements>
…
else:
   <default statements>
```

Multi-Way Decisions

- Python evaluates each condition in turn looking for the first one that evaluates to `True`. If a true condition is found, the statements indented under that condition are executed, and control passes to the next statement after the entire `if-elif-else`.
- If none are `True`, the statements under `else` are performed.
- The final `else` is optional. If there is no `else`, it's possible no indented block would be executed.

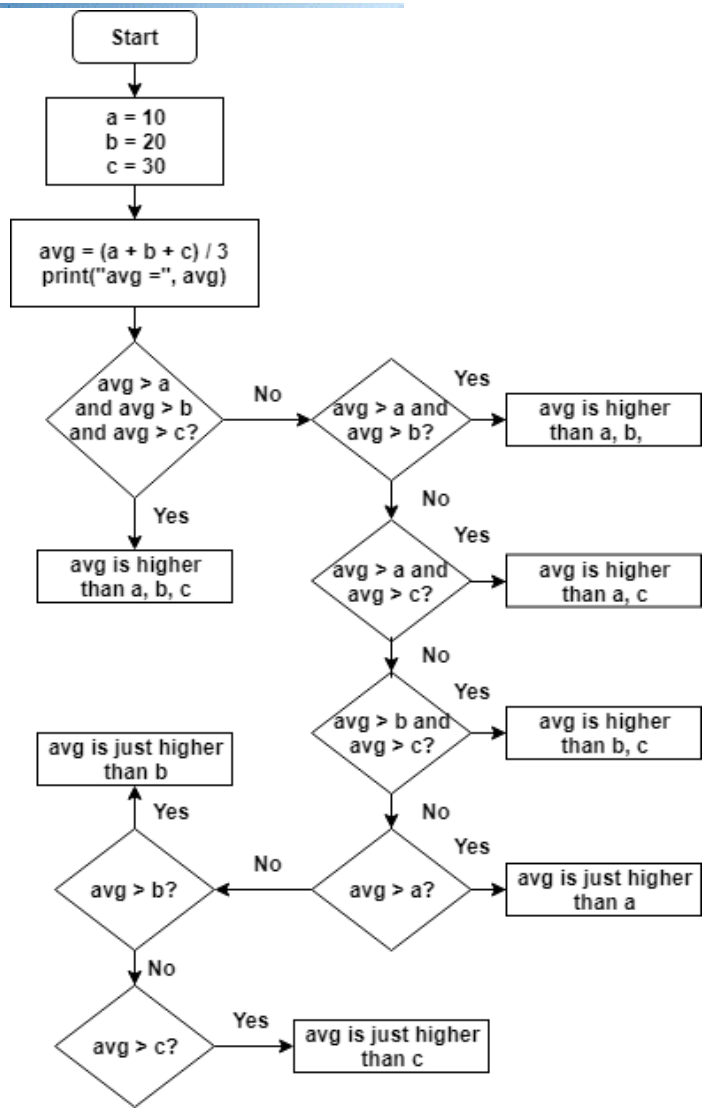
Three-Way Decisions

```
# quadratic4.py
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!")
    elif discrim == 0:
        root = -b / (2 * a)
        print("\nThere is a single root at", root)
    else:
        discRoot = math.sqrt(discrim)
        root1 = (-b + discRoot) / (2 * a)
        root2 = (-b - discRoot) / (2 * a)
        print("\nThe solutions are:", root1, root2 )
```

Nested Two-way Decisions

- What is the purpose of this algorithm?



Anti-bugging

- In the quadratic program we used decision structures to avoid taking the square root of a negative number, thus avoiding a run-time error.
- This is true for many programs: decision structures are used to protect against rare but possible errors.
- Some authors describe this as anti-bugging; before processing some data have tests to ensure procedures will be safe.

Lecture Summary

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