

CMSC201

Computer Science I for Majors

Lecture 06 – Decision Structures

Prof. Katherine Gibson

Last Class We Covered

- Just a bit about `main()`
- More of Python's operators
 - Comparison operators
 - Logical operators
- LOTS of practice using these operators
 - Reinforced order of operations
- Boolean variables

Any Questions from Last Time?

Today's Objectives

- Understand decision structures
 - One-way, two-way, and multi-way
 - Using the **if**, **if-else**, and **if-elif-else** statements
- Review control structures & conditional operators
- More practice using the Boolean data type
- Learn how to implement algorithms using decision structures

Simple Decisions

- So far, we've only seen programs with sequences of instructions
 - This is a fundamental programming concept
 - But it's not enough to solve every problem
- We need to be able to control the flow of a program to suit particular situations
 - What can we use to do that?

Conditional Operators (Review)

Python	Mathematics	Meaning
<		
<=		
==		
>=		
>		
!=		

Conditional Operators (Review)

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

Control Structures (Review)

- A program can proceed:

- In sequence

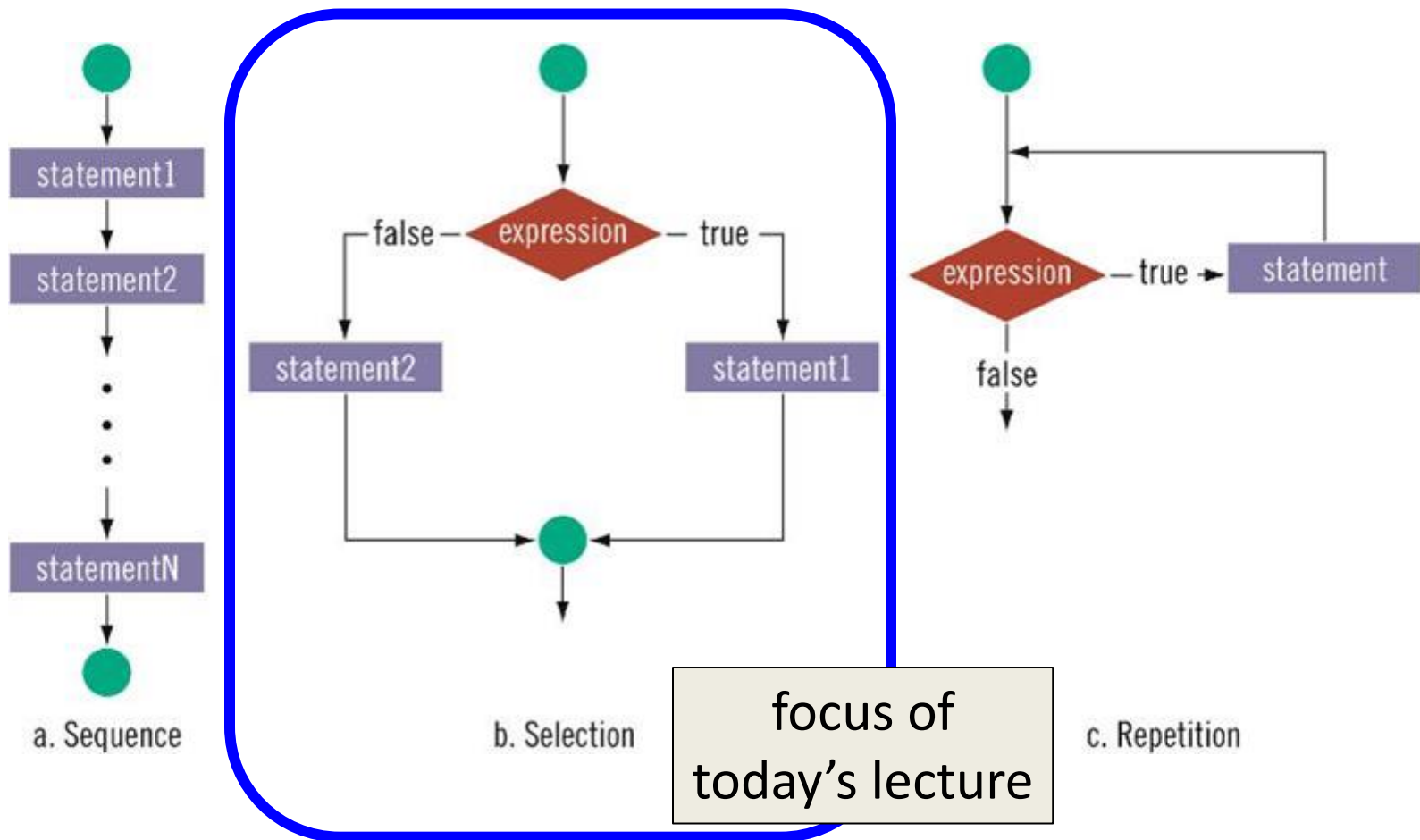
focus of
today's lecture

- **Selectively (branching): make a choice**

- Repetitively (iteratively): looping

- By calling a function

Control Structures: Flowcharts



One-Way Selection Structures

One-Way Selection Structures

- Selection statements allow a computer to make choices
 - Based on some condition

```
def main():  
    weight = float(input("How many pounds is your suitcase? "))  
  
    if weight > 50:  
        print("There is a $25 charge for luggage that heavy.")  
  
    print("Thank you for your business.")
```

```
main()
```

Temperature Example

- Convert from Celsius to Fahrenheit

```
def main():  
    celsius = eval(input("What is the Celsius temperature? "))  
    fahrenheit = 9/5 * celsius + 32  
  
    print("The temperature is ", fahrenheit,  
          " degrees Fahrenheit.")  
  
main()
```

Temperature Example - Modified

- Let's say we want to modify the program to print a warning when the weather is extreme
- Any temperature that is...
 - Over 90 degrees Fahrenheit
 - Will cause a hot weather warning
 - Lower than 30 degrees Fahrenheit
 - Will cause a cold weather warning

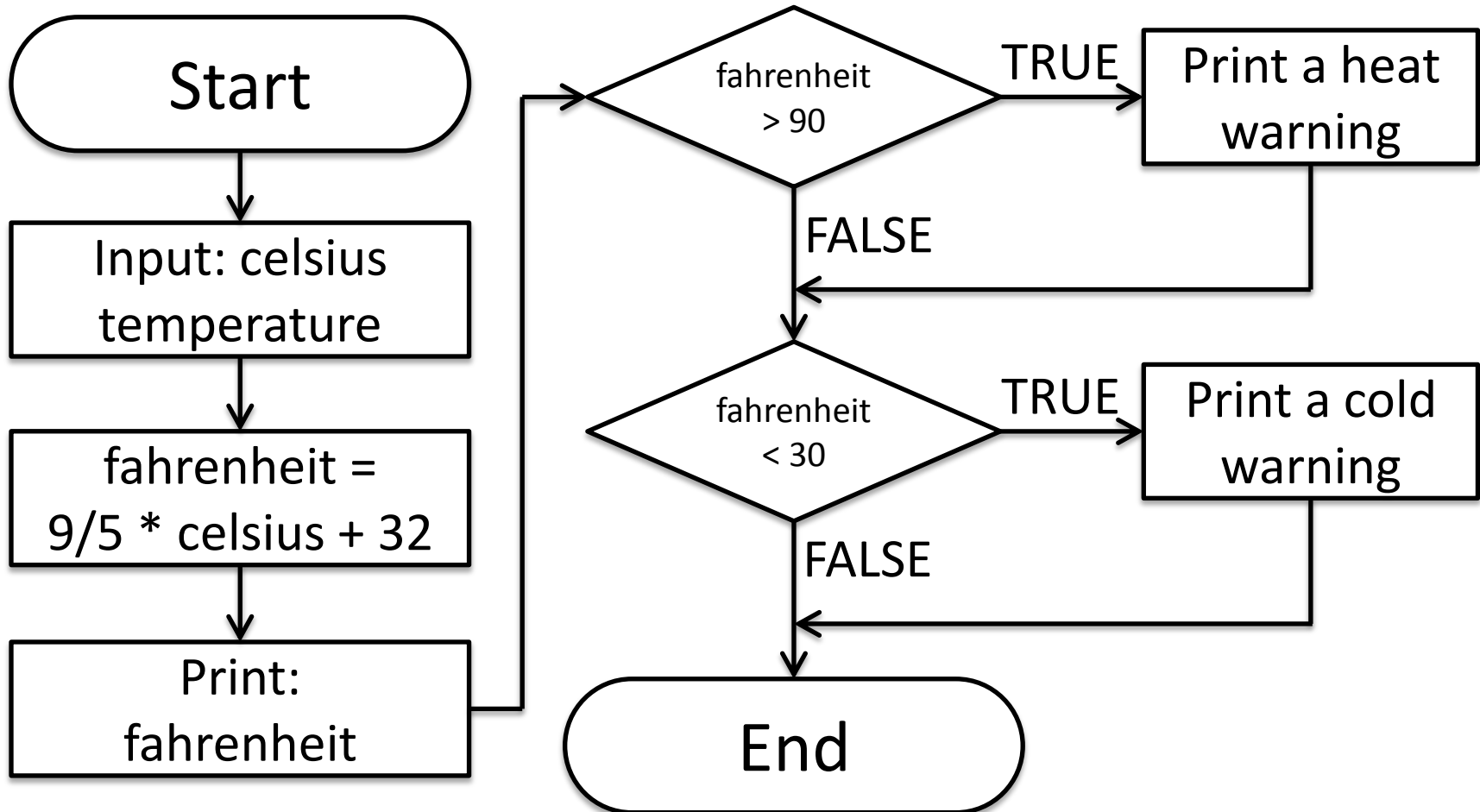
Temperature Example - Modified

- **Input:**
 - The temperature in degrees Celsius (call it `celsius`)
- **Process:**
 - Calculate `fahrenheit` as $9/5 * celsius + 32$
- **Output:**
 - `fahrenheit`
 - If `fahrenheit > 90`
 - Print a heat warning
 - If `fahrenheit < 30`
 - Print a cold warning

Temperature Example - Modified

- 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

Temperature Example Flowchart



Temperature Example Code

```
def main():
    celsius = eval(input("What is the Celsius temp? "))
    fahrenheit = 9 / 5 * celsius + 32
    print("The temp is ", fahrenheit,
          " degrees fahrenheit.")
    if fahrenheit > 90:
        print("It's really hot out there, be careful!")
    if fahrenheit < 30:
        print("Brrrrrr. Be sure to dress warmly!")

main()
```

“if” Statements

“if” Statements

- The Python `if` statement is used to implement the decision
- `if <condition>:`
 `<body>`
- The **body** is a sequence of one or more statements indented under the `if` heading

“if” Semantics

- The semantics of the **if** should be clear
 - First, the condition in the heading is evaluated
 - If the condition is **True**
 - The statements in the body are 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.

One-Way Decisions

- The body of the **if** either executes or not depending on the condition
- Control then passes to the next (non-body) statement after the **if**
- This is a *one-way* or *simple* decision

Practicing Conditions

What is a Condition?

- What does a condition look like?
- Answer:
 - All of our comparison (relational) operators plus the logical (Boolean) operators

Example – Dangerous Dinosaurs

- You have just been flown to an island where there are a wide variety of dinosaurs
- You are unsure which are dangerous so we have come up with some rules to figure out which are dangerous and which are not

LIVECODING!!!

Dinosaurs Example

- Sample rules:
 - If the dinosaur has sharp teeth, it is dangerous
 - If the dinosaur is behind a large wall, it is **not** dangerous
 - If the dinosaur is walking on two legs, it is dangerous
 - If the dinosaur has sharp claws and a beak, it is dangerous

Dinosaurs Example - Variables

- What are some reasonable variables for this code?

isSharp for sharp teeth

isWalled for behind large wall

isBiped for walking on two legs

isClawed for sharp claws

isBeaked for has beak

Dinosaurs Example - Code

```
def main():
    print("Welcome to the DinoCheck 1.0")
    print("Please answer 'True' or 'False' for each question")
    isSharp = input ("Does the dinosaur have sharp teeth? ")
    isWalled = input ("Is the dinosaur behind a large wall? ")
    isBiped = input ("Is the dinosaur walking on two legs? ")
    isClawed = input ("Does the dinosaur have sharp claws? ")
    isBeaked = input ("Does the dinosaur have a beak? ")

    if isSharp == "True":
        print("Be careful of a dinosaur with sharp teeth!")
    if isWalled == "True":
        print("You are safe, the dinosaur is behind a big wall!")
    if isBiped == "True":
        print("Be careful of a dinosaur who walks on two legs!")
    if (isClawed == "True") and (isBeaked == "True"):
        print("Be careful of a dinosaur with sharp claws and a beak!")
    print("Good luck!")
```

```
main()
```

Dinosaurs Example v2 - Code

```
def main():
    print("Welcome to the DinoCheck 1.0")
    print("Please answer '0' or '1' for each question")
    isSharp = int(input("Does the dinosaur have sharp teeth? "))
    isWalled = int(input("Is the dinosaur behind a large wall? "))
    isBiped = int(input("Is the dinosaur walking on two legs? "))
    isClawed = int(input("Does the dinosaur have sharp claws? "))
    isBeaked = int(input("Does the dinosaur have a beak? "))

    if isSharp:
        print("Be careful of a dinosaur with sharp teeth!")
    if isWalled:
        print("You are safe, the dinosaur is behind a big wall!")
    if isBiped:
        print("Be careful of a dinosaur who walks on two legs!")
    if isClawed and isBeaked:
        print("Be careful of a dinosaur with sharp claws and a beak!")
    print("Good luck!")
```

changes are in blue

```
main()
```

Two-Way Selection Structures

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>
```

How Python Handles `if-else`

- 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 only executed **after** one of the sets of statements are executed.

Two-Way Code Framework

```
if condition1 == True:
```

```
    execute code1
```

```
else:
```

```
    execute code2
```

- **Only** execute code1 if condition1 is True
- If condition1 is not True, run code2

Formatting Selection Structures

- Each **if-else** statement must close with a colon (:)
- Code in the body (that is executed as part of the **if-else** statement) must be indented
 - By four spaces
 - Hitting the “Tab” key in many editors (including emacs) will automatically indent it by four spaces

Simple Two-Way Example

```
def main():  
    x = 5  
    if x > 5:  
        print("X is larger than five!")  
    else:  
        print("X is less than or equal to five!")  
  
main()
```

Simple Two-Way Example #2

```
def main():  
    num = int(input("Enter a number: "))  
  
    if num % 2 == 0:  
        print("Your number is even.")  
    else:  
        print("Your number is odd.")  
  
main()
```

What does
this code do?

It checks if a number
is even or odd

Multi-Way Selection Structures

Bigger (and Better) Decision Structures

- One-Way and Two-Way structures are useful
- But what if we have to check multiple exclusive conditions?
 - *Exclusive* conditions do not overlap with each other
 - *e.g.*, Value of a playing card, letter grade in a class
- What could we use?

Multi-Way Code Framework

```
if <condition1>:
```

```
    <case1 statements>
```

```
elif <condition2>:
```

```
    <case2 statements>
```

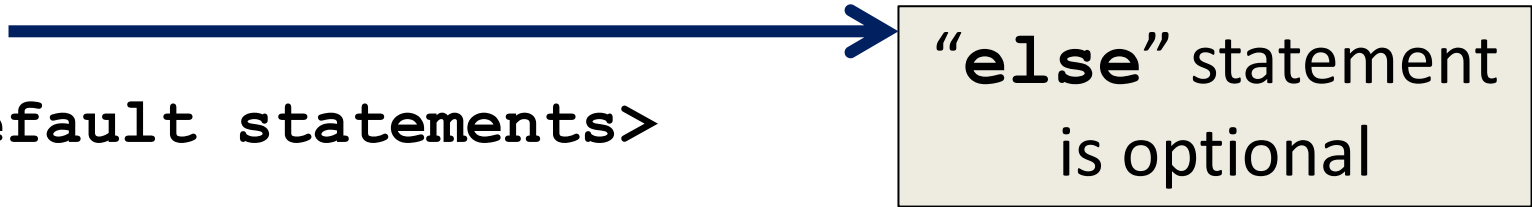
```
elif <condition3>:
```

```
    <case3 statements>
```

```
# more possible "elif" statements
```

```
else:
```

```
    <default statements>
```



“else” statement
is optional

Multi-Way Decision - Example

- Let's pretend that a mean CS professor gives a five-point attendance quiz at the beginning of every class.
- Grades are as follows:
5-A, 4-B, 3-C, 2-D, 1-F, 0-F
- What would the code look like?

Multi-Way Decision - Example

```
def main():
    score = int(input("Enter your quiz score out of 5:"))
    if score == 5:
        print("You earned an A")
    elif score == 4:
        print("You earned a B")
    elif score == 3:
        print("You earned a C")
    elif score == 2:
        print("You earned a D")
    else:
        print("You failed the quiz")

main()
```

Nested Selection Structures

Nested Decision Structures

- Up until now, we have only used a single level of decision making
- What if we want to make decisions *within* decisions?

Nested If-Else Statements

Nested Decision Structures

```
if condition1 == True:
    if condition2 == True:
        execute code1
    elif condition3 == True:
        execute code2
    else:
        execute code3
else:
    execute code4
```

Nested Decision Structures - Example

- You recently took a part-time job to help pay for your video game addiction at a local cell-phone store
- If you sell at least \$1000 worth of phones in a pay period, you get a bonus
- Your bonus is 3% if you sold at least 3 iPhones, otherwise your bonus is 2%

Nested Decision Structures - Example

```
def main():
    totalSales = float(input("Please enter your total sales:"))
    if totalSales >= 1000.00:
        iPhonesSold = int(input("Enter the number of iPhones
            sold:"))
        if iPhonesSold >= 3:
            bonus = totalSales * 0.03
        else:
            bonus = totalSales * 0.02
        print("Your bonus is $", bonus)
    else:
        print ("Sorry, you do not get a bonus this time.")
main()
```

Example: Max of Three

Study in Design: Max of Three

- Now that we have decision structures, we can solve more complicated programming problems.
- The downside is that writing these programs becomes more complicated too!
- Suppose we need an algorithm to find the largest of three numbers.

Study in Design: Max of Three

```
def main():  
    x1, x2, x3 = eval(input("Please enter three values: "))  
  
    # missing code sets max to the value of the largest  
  
    print("The largest value is", max)  
main()
```

Strategy 1:

Compare Each to All

- This looks like a three-way decision, where we need to execute *one* of the following:

`max = x1`

`max = x2`

`max = x3`

- All we need to do now is preface each one of these with the right condition!

Strategy 1: Compare Each to All

- Let's look at the case where x_1 is the largest.
- **`if x1 >= x2 >= x3:`**
 `max = x1`
- Is this syntactically correct?
 - Many languages would not allow this *compound condition*
 - Python does allow it, though. It's equivalent to $x_1 \geq x_2 \geq x_3$.

Strategy 1: Compare Each to All

- Whenever you write a decision, there are two crucial questions:
 - When the condition is true, is executing the body of the decision the right action to take?
 - x_1 is at least as large as x_2 and x_3 , so assigning max to x_1 is OK.
 - Always pay attention to borderline values!!

Strategy 1: Compare Each to All

- Secondly, ask the converse of the first question, namely, are we certain that this condition is true in all cases where x_1 is the max?
 - Suppose the values are 5, 2, and 4.
 - Clearly, x_1 is the largest, but does $x_1 \geq x_2 \geq x_3$ hold?
 - We don't really care about the relative ordering of x_2 and x_3 , so we can make two separate tests: $x_1 \geq x_2$ *and* $x_1 \geq x_3$.

Strategy 1: Compare Each to All

- We can separate these conditions with *and*!

```
if x1 >= x2 and x1 >= x3:
```

```
    max = x1
```

```
elif x2 >= x1 and x2 >= x3:
```

```
    max = x2
```

```
else:
```

```
    max = x3
```

- We're comparing each possible value against all the others to determine which one is largest.

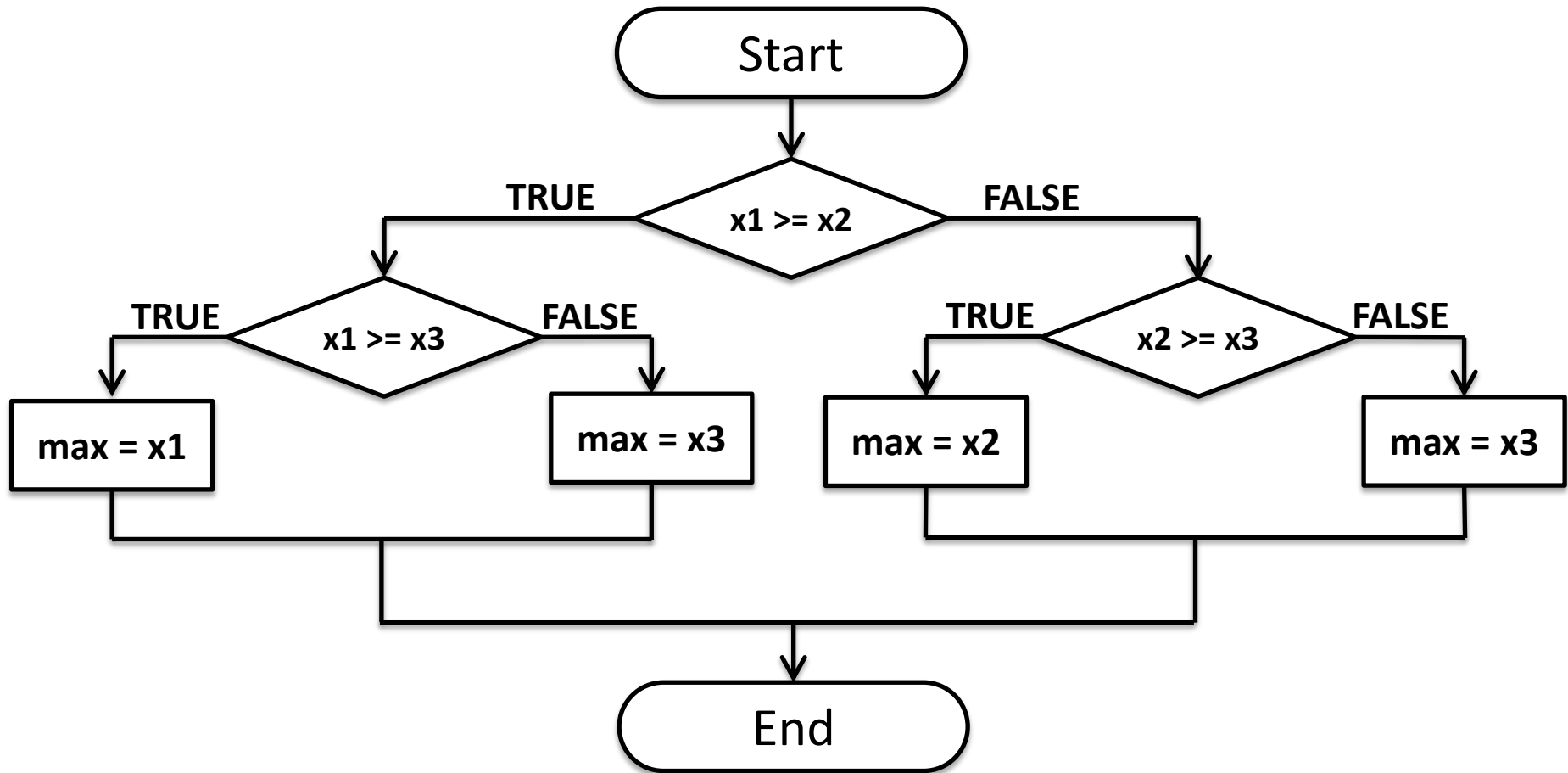
Strategy 1: Compare Each to All

- What would happen if we were trying to find the max of five values?
- We would need four Boolean expressions, each consisting of four conditions *anded* together.
- Yuck!

Strategy 2: Decision Tree

- We can avoid the redundant tests of the previous algorithm using a *decision tree* approach.
- Suppose we start with $x1 \geq x2$. This knocks either $x1$ or $x2$ out of contention to be the max.
- If the condition is true, we need to see which is larger, $x1$ or $x3$.

Strategy 2: Decision Tree



Strategy 2: Decision Tree

- `if x1 >= x2:`
 - `if x1 >= x3:`
 - `max = x1`
 - `else:`
 - `max = x3`
 - `else:`
 - `if x2 >= x3:`
 - `max = x2`
 - `else`
 - `max = x3`

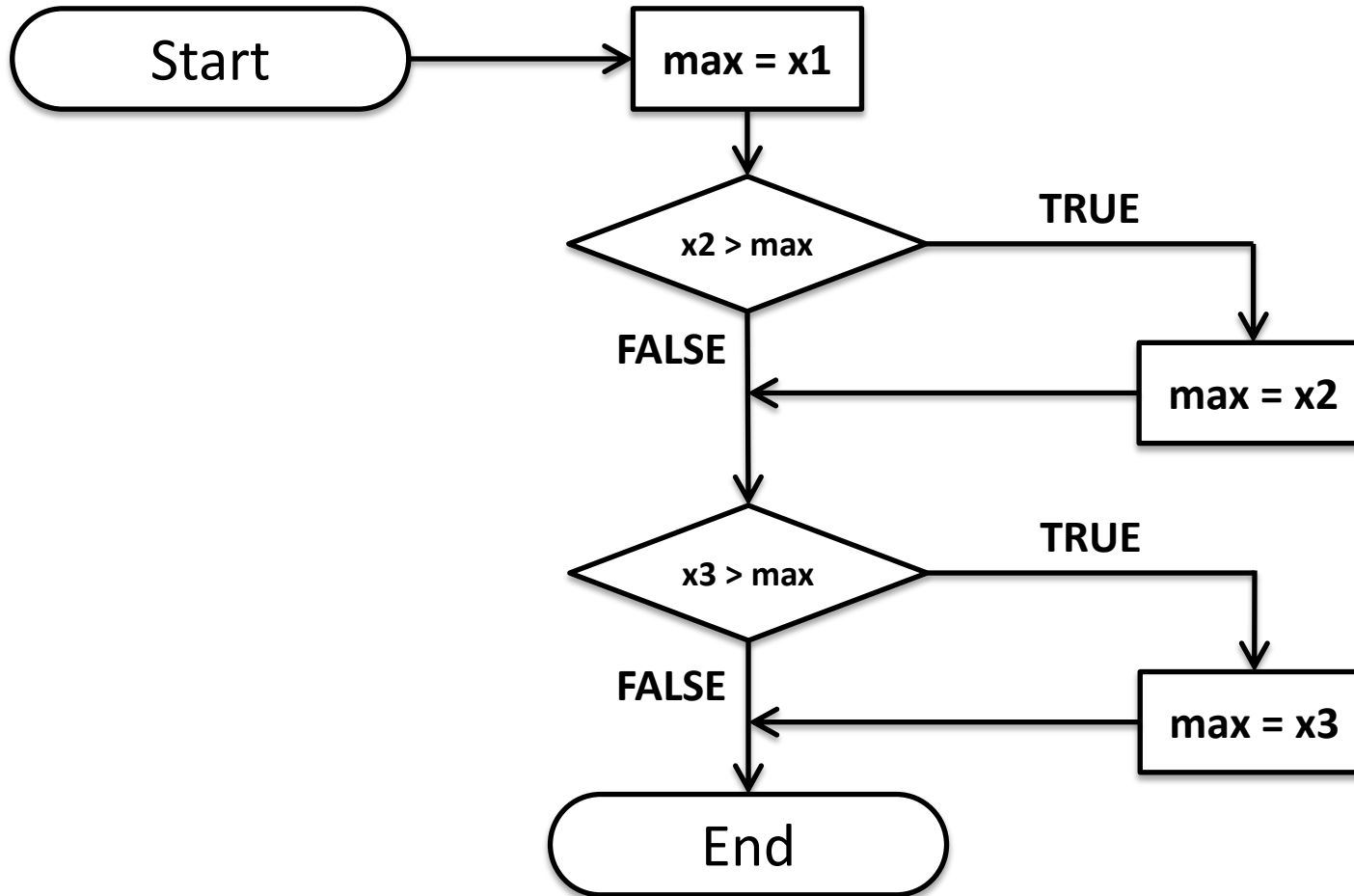
Strategy 2: Decision Tree

- This approach makes exactly two comparisons, regardless of the ordering of the original three variables.
- However, this approach is more complicated than the first
 - To find the max of four values you'd need **if-elses** nested three levels deep with eight assignment statements.

Strategy 3: Sequential Processing

- How would you solve the problem?
- You could probably look at three numbers and just *know* which is the largest. But what if you were given a list of a hundred numbers?
- One strategy is to scan through the list looking for a big number. When one is found, mark it, and continue looking. If you find a larger value, mark it, erase the previous mark, and continue looking.

Strategy 3: Sequential Processing



Strategy 3: Sequential Processing

- This idea can easily be translated into Python.

```
max = x1
if x2 > max:
    max = x2
if x3 > max:
    max = x3
```

Strategy 3: Sequential Programming

- This process is repetitive and lends itself to using a loop.
- We prompt the user for a number, we compare it to our current max, if it is larger, we update the max value, repeat.

Strategy 3: Sequential Programming

```
# maxn.py
#     Finds the maximum of a series of numbers

def main():
    n = eval(input("How many numbers are there? "))

    # Set max to be the first value
    max = eval(input("Enter a number >> "))

    # Now compare the n-1 successive values
    for i in range(n-1):
        x = eval(input("Enter a number >> "))
        if x > max:
            max = x

    print("The largest value is", max)
main()
```


Strategy 4: Use Python

- Python has a built-in function called **max** that returns the largest of its parameters.

```
def main():  
    x1, x2, x3 = eval(input("Please enter three values: "))  
    print("The largest value is", max(x1, x2, x3))
```

Some Lessons

- There is usually more than one way to solve a problem.
 - **Don't rush to code the first idea** that pops out of your head. Think about the design and ask if there's a better way to approach the problem.
 - Your first task is to find a correct algorithm. After that, strive for clarity, simplicity, efficiency, scalability, and elegance.

Some Lessons

- **“BE”** the computer.
 - One of the best ways to formulate an algorithm is to ask yourself how you would solve the problem.
 - This straightforward approach is often simple, clear, and efficient enough.

Some Lessons

- Generality is good.
 - Considering a more general problem can lead to a better solution for a special case.
 - If the max of n program is just as easy to write as the max of three, write the more general program because it's more likely to be useful in other situations.

Some Lessons

- Don't reinvent the wheel.
 - If the problem you're trying to solve is one that lots of other people have encountered, find out if there's already a solution for it!
 - As you learn to program, designing programs from scratch is a great experience!
 - Truly expert programmers know when to borrow.
 - LAZINESS!

Announcements

- Your Lab 3 is meeting normally this week!
 - Make sure you attend your correct section
 -
- Homework 3 is out
 - Due by Thursday (Sept 24th) at 8:59:59 PM
- Homeworks are on Blackboard
 - Weekly Agendas are also on Blackboard