CMSC201
Computer Science I for Majors

Lecture 06 – Decision Structures

Prof. Katherine Gibson
Prof. Jeremy Dixon

Based on concepts from: https://blog.udemy.com/python-if-else/

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)

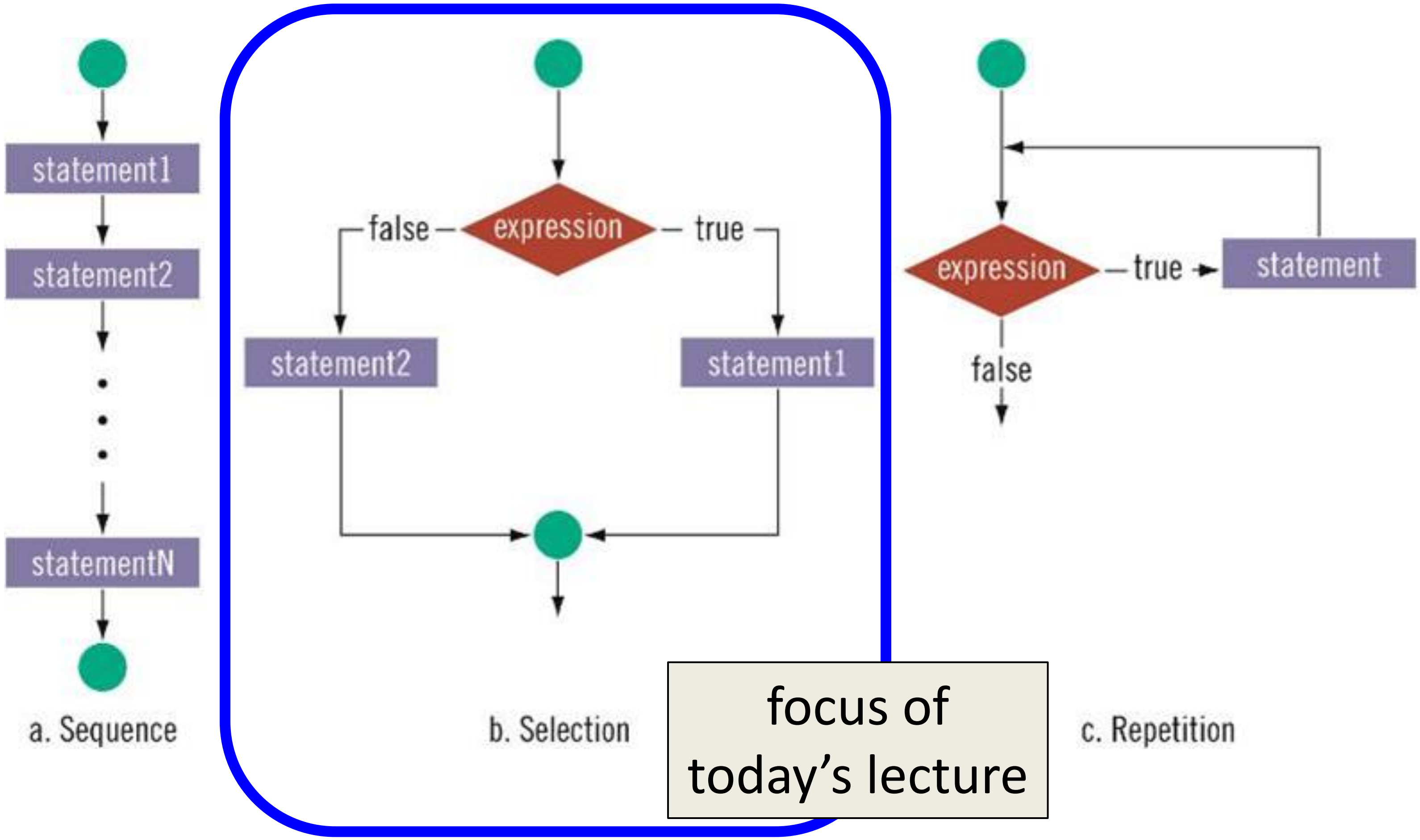
Python	Mathematics	Meaning
`<`	$<$	Less than
`<=`	$\leq$	Less than or equal to
`==`	$=$	Equal to
`>=`	$\geq$	Greater than or equal to
`>`	$>$	Greater than
`!=`	$\neq$	Not equal to

Control Structures (Review)

- A program can proceed:
  - In sequence
  - Selectively (branching): make a choice
  - Repetitively (iteratively): looping
  - By calling a function

focus of today’s lecture

Control Structures: Flowcharts



focus of today’s lecture

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 = float(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

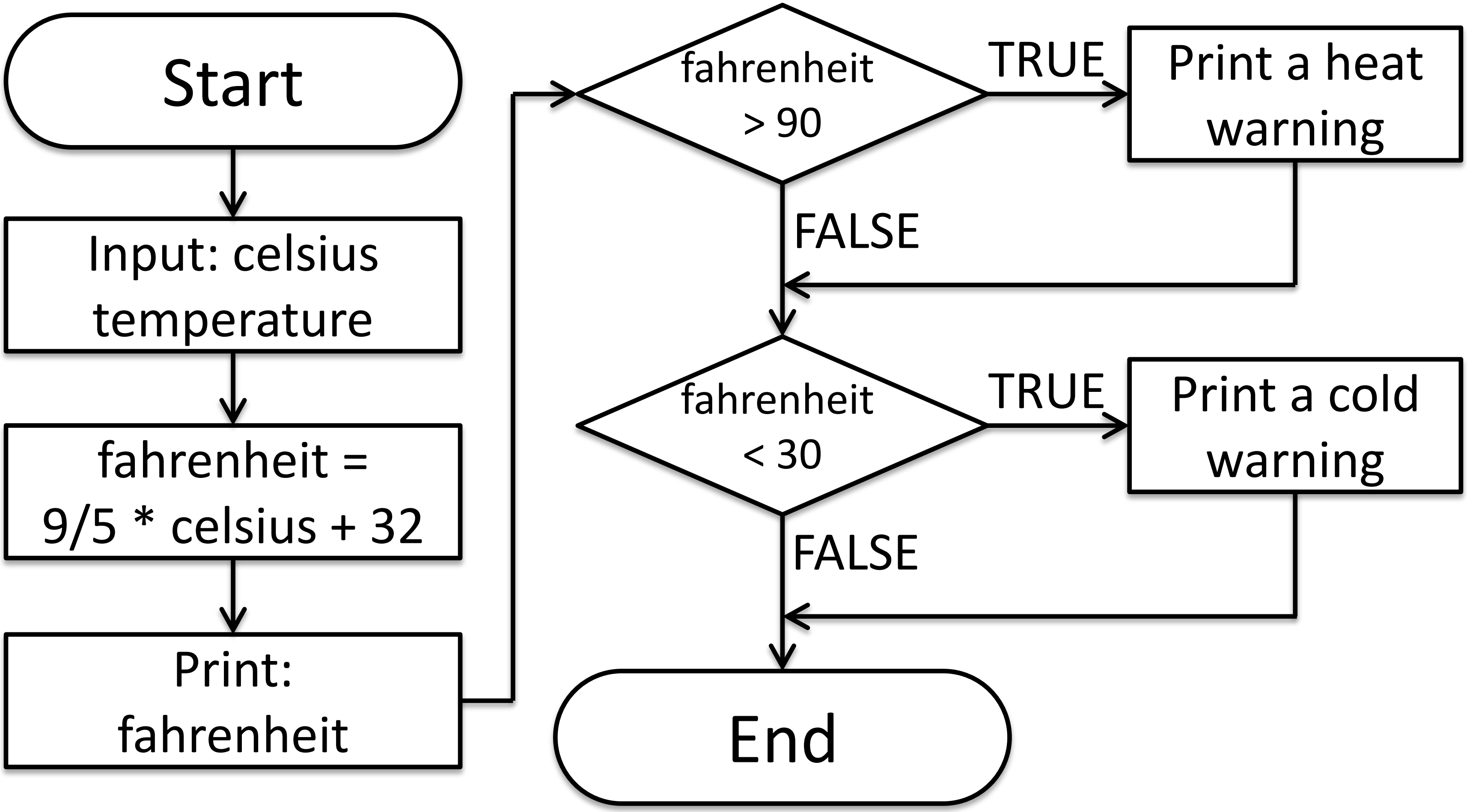
- **Input:**
  - The temperature in degrees Celsius (call it **`celsius`**)
- **Process:**
  - Calculate **`fahrenheit`** as **`9/5 * celsius + 32`**
- **Output:**
  - Temperature in Fahrenheit
  - If **`fahrenheit`** > 90
    - Display a heat warning
  - If **`fahrenheit`** < 30
    - Display a cold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

Temperature Example - Modified

- This new algorithm has two *decisions* at the end

- The indentation after the “if” is important
- It means that a step should be performed **only** if the condition in the previous line is True

Temperature Example Flowchart



Temperature Example Code

```
def main():
    celsius = float(input("What is the Celsius temp? "))
    fahrenheit = 9 / 5 * 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()
```

Temperature Example Code

```
def main():
    celsius = float(input("What is the Celsius temp? "))
    fahrenheit = 9 / 5 * 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()
```

this is the main level of our program

this level of the code is only executed if `fahrenheit > 90`

this level of the code is only executed if `fahrenheit < 30`

“`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
    - Control passes to the next statement in the program
  - If the condition is **`False`**
    - The statements in the body are skipped
    - 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

What is a Condition?

- Conditions
  - Can use any comparison (rational) operators
  - Can use any logical (Boolean) operators
  - Evaluate to **`True`** or **`False`**

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 sees this structure, it evaluates the condition
  - If the condition is **`True`**, the set of statements under the **`if`** are executed
  - If the condition is **`False`**, the set of statements under the **`else`** are executed

- The code after the **`if-else`** is only executed <u>after</u> one of the sets of statements is executed

Two-Way Code Framework

```
if theCondition == True:
    <code1>
else:
    <code2>
```

- Only execute code1 if **`theCondition`** is True
- If **`theCondition`** 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()
```

this is the main level of our program

this level of the code is only executed if `x > 5` is `True`

this level of the code is only executed if `x > 5` is `False`

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 whether a number is even or odd.

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

Time for…

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 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 – Another Way

changes are in blue

```
def main():
    print("Welcome to DinoCheck 1.0")
    print("Please answer '0' (no) or '1' (yes) 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!")

main()
```

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 "elif" statements if needed
else:
    <default statements>
```

"else" statement is optional

Multi-Way Selection Example

- A a computer science professor gives a five-point quiz at the beginning of every class
- Possible grades are as follows:

5 points: A	3 points: C	1 point: F
4 points: B	2 points: D	0 points: F

- To print out the letter grade based on the raw points, what would the code need to look like?

Multi-Way Selection Solution

```
def main():
    score = int(input("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()
```

Multi-Way Selection Solution

```
def main():
    score = int(input("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()
```

these are five separate statements

since this is an `if-elif-else` block, only <u>one</u> of the five statements will be executed

Nested Selection Structures

Nested Selection Structures

- Up until now, we have only used a single level of decision making
- What if we want to make decisions within decisions?
- These are called *nested* selection structures
  - We'll first cover nested **`if-else`** statements

Nested Selection Structure Examples

- For example, we may
  - Ask the user if they have a pet
  - **`if`** they have a pet
    - Ask the user what type of pet
    - **`if`** they have a dog, take it for a walk
    - **`elif`** they have a cat, clean the litter box
    - **`else`** clean the cage/stable/tank

Nested Selection Structures Code

```
if condition1 ==  True:
    if condition2 == True:
        execute codeA
    elif condition3 == True:
        execute codeB
    else:
        execute codeC
else:
    execute codeD
```

Nested Selection Structures Code

```
if condition1 ==  True:
    if condition2 == True:
        execute codeA
    elif condition3 == True:
        execute codeB
    else:
        execute codeC
else:
    execute codeD
```

this is the main level of our program: an `if-else` block

this is the next level, inside the first `if` statement

codeA, codeB, and codeC are separate statements

since this is an `if-elif-else` block, only one of them will be executed

if our first `if` statement was false, we would skip here and execute codeD

Nested Selection Structure Example

- You recently took a part-time job to help pay for your student loans 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 only 2%

Nested Selection Solution

```
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 pay period.")
main()
```

Design Example: Max of Three

Study in Design: Max of Three

- With decision structures, we can solve more complicated programming problems
- However, designing and coding these programs becomes more complicated too!
- Let’s create an algorithm to find the largest of three numbers

Max of Three: Code Framework

- Here’s the “easy” part of our code completed:

```
def main():
    x1, x2, x3 = int(input("Please enter three values: "))

    # we need to write the missing code that sets
    # "maximum" to the value of the largest number

    print("The largest value is ", maximum)

main()
```

Strategy 1: Compare Each to All

- This looks like a three-way decision, where we need to execute <u>one</u> of the following:

```
maximum = x1
maximum = x2
maximum = x3
```

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

Strategy 1: Sample Code

- Let’s look at the case where **`x1`** is the largest

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

- Is this syntactically correct?
  - Yes, Python allows this
  - It’s equivalent to **x1 ≥ x2 ≥ x3**

Aside: Writing Decisions

- When writing a decision, there are two critical questions:
  1. Does the condition accurately and correctly test what we want it to test?
     - Are we certain the condition is true in all cases where **`x1`** is the max?
  2. When the condition is true, does the body of the decision perform the correct action?
     - In our example, if **`x1`** is at least as large as **`x2`** and **`x3`**, then the maximum should be **`x1`**

Writing Decisions: Conditions

- Is the condition doing what we want it to?

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

- What is this actually testing?
  - What happens if **`x3`** is bigger than **`x2`**?
  - It returns **`False`**! But **`x2`** vs **`x3`** doesn't matter
  - Split it into two separate tests:

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

Writing Decisions: Body Statements

- Is the body statement doing what is appropriate when the conditional is **`True`**?

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

- Yes! If **`x1`** is at least as large as both **`x2`** and **`x3`**, we can set its value to be the maximum

Strategy 1: Solution

- Here’s our completed code:

```
def main():
    x1, x2, x3 = int(input("Please enter three values: "))
    if x1 >= x2 and x1 >= x3:
        maximum = x1
    elif x2 >= x1 and x2 >= x3:
        maximum = x2
    else:
        maximum = x3

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

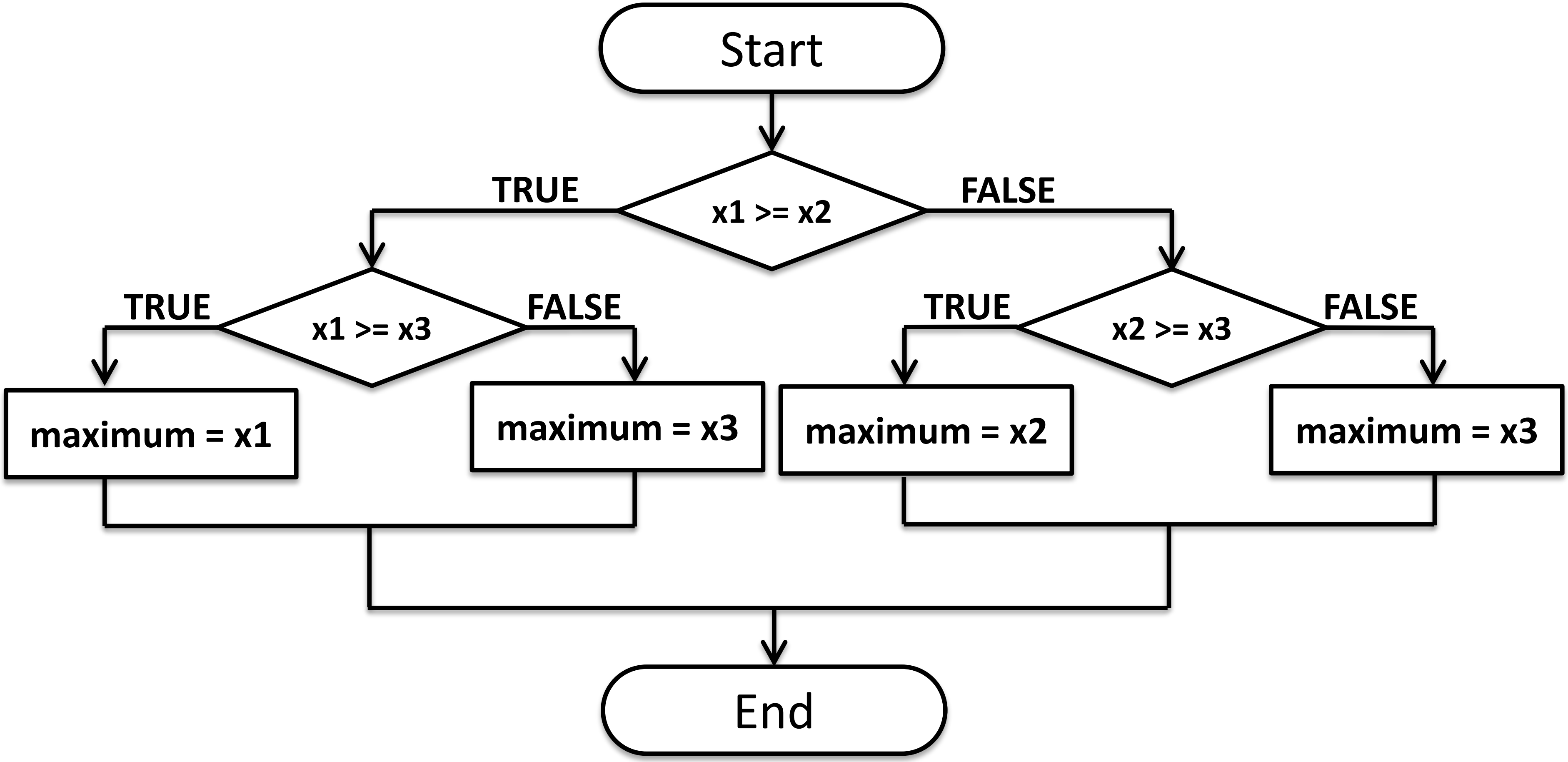
Strategy 1: Downsides

- 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 **`and`**’ed together
- What about twenty values?
  - We would need nineteen Boolean expressions, with nineteen conditions each
- There has to be a better way!

Strategy 2: Decision Tree

- We can avoid the redundant tests of the previous algorithm using a *decision tree* instead
- Suppose we start with **`x1 >= x2`**
  - This knocks either **`x1`** or **`x2`** out of the running to be the maximum value
  - If the condition is **`True`**, then we need to check whether **`x1`** or **`x3`** is larger

Strategy 2: Decision Tree Flowchart



Strategy 2: Decision Tree Code

- Here’s the code for the previous flowchart

```
if x1 >= x2:
    if x1 >= x3:
        maximum = x1
    else:
        maximum = x3
else:
    if x2 >= x3:
        maximum = x2
    else:
        maximum = x3
```

Strategy 2: (Dis)advantages

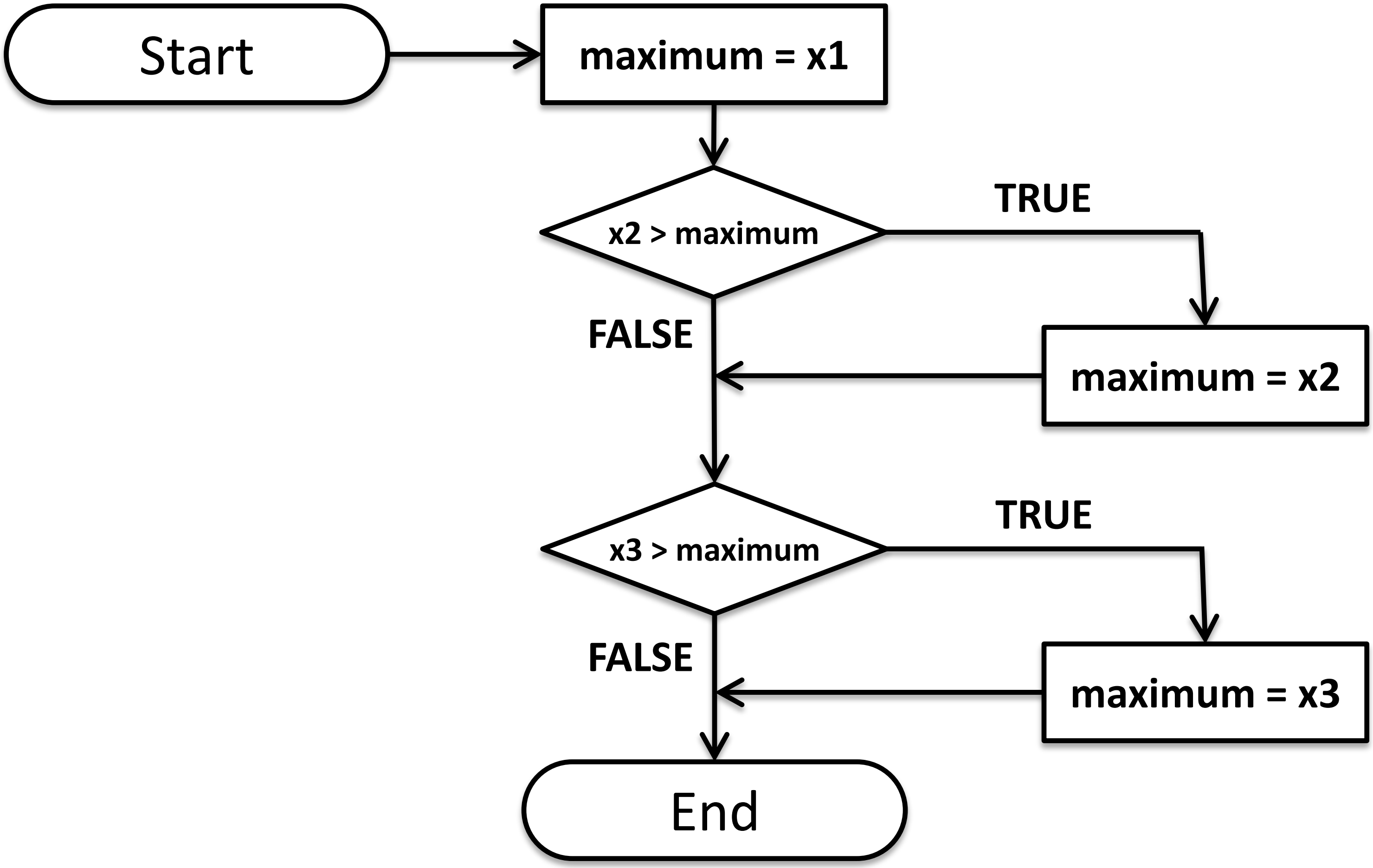
- This approach makes exactly two comparisons between the three variables

- However, this approach is more complicated than the first
  - To find the max of four values you'd need **`if-else`**s nested three levels deep with eight assignment statements
  - This isn't much better than the last method!

Strategy 3: Sequential Processing

- How would *you* solve the problem?
- Since you’re not a computer, you could look at three numbers and know which is the largest
  - But what if there were one hundred numbers?
- One strategy is to scan the list 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 Code

- This idea can be easily done in Python code

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

Why do we use two **`if`** statements?

What would happen if we used an **`if-elif`** statement?

Strategy 3: Sequential Processing

- This process is pretty repetitive
  - Which means we could use a loop!
- We would repeat the following steps:
  1. Prompt the user for a number
  2. Compare it to the current maximum
  3. If it is larger, update the max value
  - Repeat until the user is done entering numbers

- We'll talk about this more when we cover loops

Strategy 4: Take Advantage of Python

- Python has a built-in function called **`max`**
  - It takes in numbers and returns the max value

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

  - This is why we called our variable "**`maximum`**" instead of **`max`** – because **`max`** is already defined!

Lessons Learned

Avoid "Cowboy Coding"

- There is usually more than one way to solve a problem
  - So **<u>don't</u> rush to code the first idea** that pops into 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

Think Like a Computer

- Try to "**BE**" the computer
  - One of the best ways to design an algorithm is to ask yourself how you would solve the problem
    - (Try to keep in mind the restrictions a computer has when you're doing this)
  - This straightforward approach often makes for simple, clear, and efficient code

Design for the Future

- Generality is good!
  - Considering a more general problem can lead to a better solution for a special case
  - If a "max of ***N*** numbers" program is just as easy to write as the max of three, write the more general program
    - It's more likely to be useful in other situations

Don’t Duplicate Effort

- 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 are learning to program, designing programs from scratch is a great experience!
  - Truly expert programmers know when to borrow

But, as beginning programmers, you are not allowed to use built-in functions to solve assignments -- we need to see your understanding!

Announcements

- Your Lab 2 is meeting normally this week!
  - If you had Lab on Monday, see BB for instructions
- Homework 3 is out
  - Due by Monday (Feb 22nd) at 8:59:59 PM
  - Homework 2 due date extended to Feb 16
- Homeworks and Pre-Labs are on Blackboard

Practice Problem

- Create a choose-your-own-adventure program using **`if-else-elif`** statements
- For example:

```
print("You enter a dark room with two doors.")
print("Do you go through door #1 or door #2?")
door = int(input("Choose a door: "))
if door == 1:
    print("There's a bear eating a cheese cake.")
    print("You can run, hide, or talk to it.")
    # and so on...
```

Practice Problems

- From the Zelle textbook:
  - Chapter 7, Programming Exercises
    - #1 (overtime)
    - #6 (speeding tickets)
    - #8 (political eligibility)
    - #11 (leap years)

- Be creative: come up with a problem and solve it in Python code. Trade problems with a friend!