CMSC201
Computer Science I for Majors

Lecture 05 – Decision Structures

Last Class We Covered

- Expressions
- Python's operators
  - Including mod and integer division
- The order of operations
- Different variables types
  - How to cast to a type
- Constants (and why using them is important)

Any Questions from Last Time?

Today's Objectives

- To learn a bit about **`main()`**
- To learn more of Python's operators
  - Comparison operators
  - Logical operators
- To start covering decision structures
  - Using **`if`** and **`else`**
- Practice using the Boolean data type

Quick Note about `main()`

`main()`

- In Lab 2, we introduced the code

  `def main():`

  as the first line of code in our file

- **`main()`** is an example of a **function**
- We can use functions to organize our code

Functions

- We'll cover functions in more detail later

- For now, think of them as something similar to a variable
  - Variables hold data
  - Functions hold code

Calling `main()`

- With variables, we use the variable name to access the data they store

- We must do the same with functions like **`main()`**, using the function name to execute the code they store

Using `main()` for Your Code

- From now on, use `main()` in your code:

declaring our `main()` function

```
def main():
    className = input("What class is this? ")
    print(className, "is awesome!")

main()
```

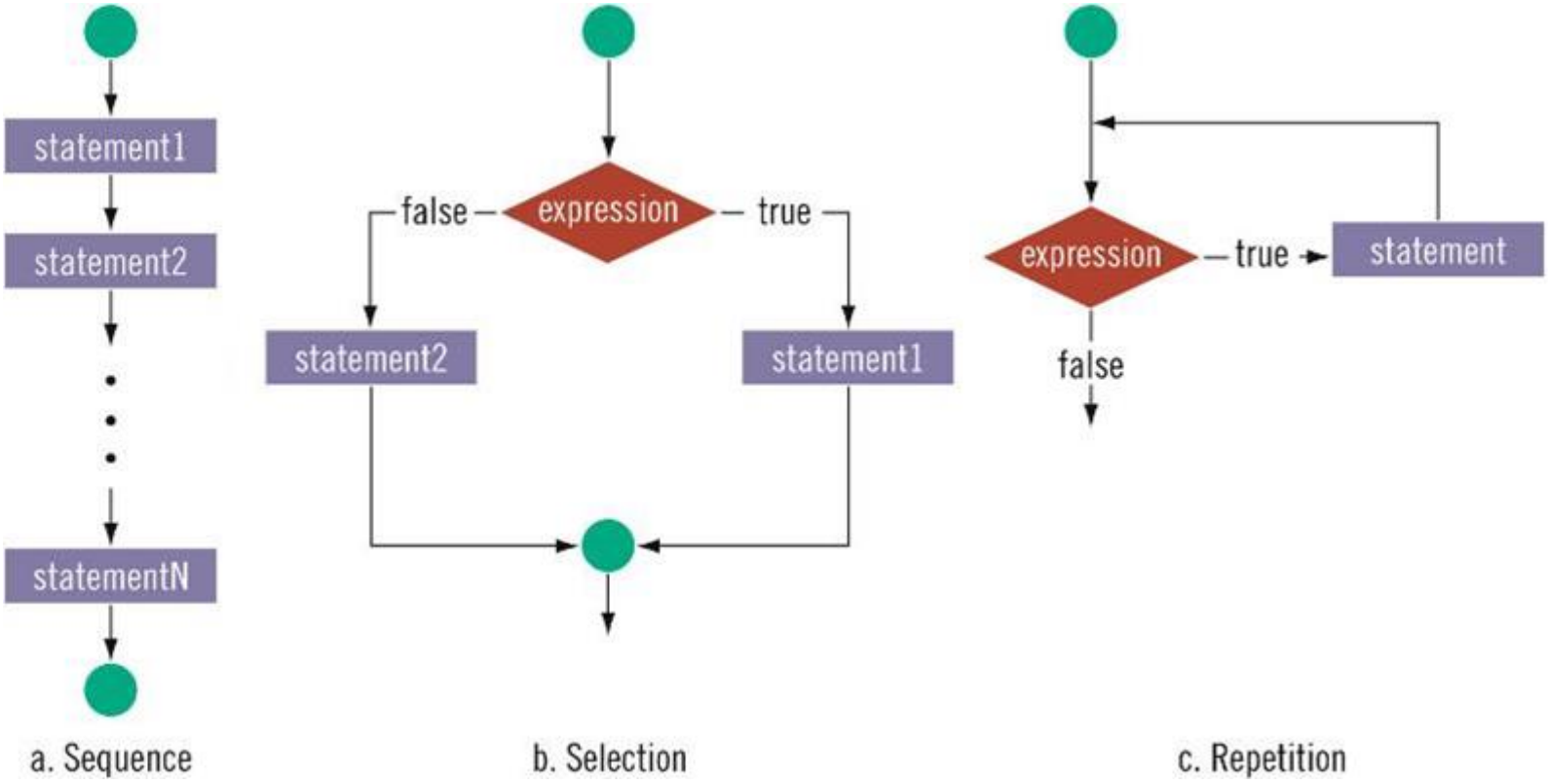
calling our `main()` function

Review:
Control Structures & Operators

Control Structures

- What are the three control structures?
  - Sequential
  - Decision Making
    - Also known as “Selection”
  - Looping
    - Also known as “Repetition”
- (We can also call a function)

Control Structures: Flowcharts



Types of Operators in Python

- Arithmetic Operators ✓
- Comparison (Relational) Operators
- Assignment Operators ✓
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

focus of today's lecture

Comparison Operators

Vocabulary

- Comparison operators
- Relational operators
- Equality operators
  - Are all the same thing

- Include things like **>**, **>=**, **<**, **<=**, **==**, **!=**

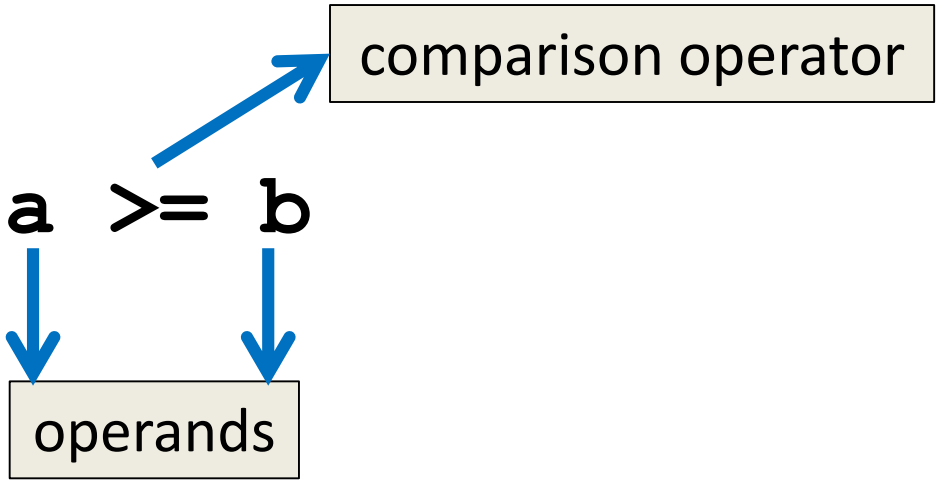
Vocabulary

- Logical operators
- Boolean operators
  - Are the same thing

- Include **`and`**, **`or`**, and **`not`**

Comparison Operators

- Always return a Boolean result
  - **`True`** or **`False`**
  - Indicates whether a relationship holds between their operands



Comparison Examples

- What are the following comparisons asking?

  `a >= b`

  – Is `a` greater than or equal to `b`?

  `a == b`

  – Is `a` equivalent to `b`?

List of Operators

Operation	Meaning
`<`	strictly less than
`<=`	less than or equal
`>`	strictly greater than
`>=`	greater than or equal
`==`	equal
`!=`	not equal

Comparison Examples (Continued)

- What do these evaluate to if `a = 10` and `b = 20`?

  `a >= b`

  - Is `a` greater than or equal to `b`?
  - Is `10` greater than or equal to `20`?
  - **FALSE**

Comparison Examples (Continued)

- What do these evaluate to if `a = 10` and `b = 20`?

  `a == b`

  - Is `a` equivalent to `b`?
  - Is `10` equivalent to `20`?
  - **FALSE**

Comparison vs Assignment

- A common mistake is to use the assignment operator (=) in place of the relational (==)
  - This is a very common mistake to make!

- This type of mistake does trigger an error in Python, but you may still make it on paper!

Equals vs Equivalence

- What does `a = b` do?
  - Sets `a` equal to `b`
  - Replaces `a`'s value with the value of `b`

- What does `a == b` do?
  - Checks if `a` is equivalent to `b`

Evaluating to Boolean Values

Comparison Operators and Simple Data Types

- Examples:

`8 < 15`	evaluates to	**`True`**
`6 != 6`	evaluates to	**`False`**
`2.5 > 5.8`	evaluates to	**`False`**
`5.9 <= 7.5`	evaluates to	**`True`**

"Value" of Boolean Variables

- When we discuss Boolean outputs, we think

  `True` and **`False`**

- We can also think of it in terms of

  `1` and `0`

- **`True  = 1`**
- **`False = 0`**

"Value" of Boolean Variables

- Other data types can also be seen as "**`True`**" or "**`False`**" in Python

- Anything empty or zero is **`False`**
  - `""` (empty string), `0`, `0.0`
- Everything else is **`True`**
  - `81.3`, `77`, `-5`, `"zero"`, `0.01`
  - Even `"0"` evaluates to **`True`**

Comparison Operation Examples

```
a = 10
b = 20
c = 30

bool1 = int(a == a)
bool2 = a >= 10
bool3 = (a == a) + (b == b) + (c == c)

print(bool1, bool2, bool3)
```

Prints:

```
1 True 3
```

Logical Operators

Logical Operators

- There are three logical operators:
  - `and`
  - `or`
  - `not`

- They allow us to build more complex Boolean expressions
  - By combining simpler Boolean expressions

Logical Operators – `and`

- Let's evaluate this expression

```
bool1 = a and b
```

Value of `a`	Value of `b`	Value of `bool1`
`True`	`True`	
`True`	`False`	
`False`	`True`	
`False`	`False`	

Logical Operators – `and`

- Let's evaluate this expression

```
bool1 = a and b
```

Value of `a`	Value of `b`	Value of `bool1`
`True`	`True`	`True`
`True`	`False`	`False`
`False`	`True`	`False`
`False`	`False`	`False`

- For `a and b` to be `True`, both `a` and `b` must be true

Examples of **and**

```
a = 10
b = 20
c = 30

ex1 = a < b
ex2 = a < b and b < c
ex3 = a + b == c and b - 10 == a
      and c / 3 == a

print (ex1, ex2, ex3)
```

Prints:

```
True True True
```

Logical Operators – `or`

- Let's evaluate this expression

```
bool2 = a or b
```

Value of `a`	Value of `b`	Value of `bool2`
`True`	`True`	
`True`	`False`	
`False`	`True`	
`False`	`False`	

Logical Operators – `or`

- Let's evaluate this expression

```
bool2 = a or b
```

Value of `a`	Value of `b`	Value of `bool2`
`True`	`True`	`True`
`True`	`False`	`True`
`False`	`True`	`True`
`False`	`False`	`False`

- For `a or b` to be `True`, either `a` or `b` must be true

Usage Example

- Here's an easy way to remember how the **`and`** and **`or`** logical operators work

- In order to pass the class, you must have:

```
(grade >= 80) and (cheating == False)
```

- For the grade to count for CMSC/CMPE majors:

```
ltrGrade == "A" or ltrGrade == "B"
```

Logical Operators – `not`

- Let's evaluate this expression

```
bool3 = not a
```

Value of `a`	Value of `bool3`
`True`	`False`
`False`	`True`

- `not a` calculates the Boolean value of `a` and returns the opposite of that

Complex Expressions

- We can put multiple operators together!

```
bool4 = a and (b or c)
```

- What does Python do first?
  - Computes **`(b or c)`**
  - Computes **`a and`** the result

Complex Expression Example

```
bool4 = a and (b or c)
```

Value of a	Value of b	Value of c	Value of bool4
True	True	True	
True	True	False	
True	False	True	
True	False	False	
False	True	True	
False	True	False	
False	False	True	
False	False	False	

Complex Expression Example

```
bool4 = a and (b or c)
```

Value of a	Value of b	Value of c	Value of bool4
True	True	True	True
True	True	False	True
True	False	True	True
True	False	False	False
False	True	True	False
False	True	False	False
False	False	True	False
False	False	False	False

Truth Table Layout

- Truth tables follow a pattern for their values

Value 1	Value 2	Value 3	Answer
True	True	True	
True	True	False	
True	False	True	
True	False	False	
False	True	True	
False	True	False	
False	False	True	
False	False	False	

“Short Circuit” Evaluation

Short Circuit Evaluation

- "**`and`**" statements short circuit as soon as an expression evaluates to **`False`**

- "**`or`**" statements short circuit as soon as an expression evaluates to **`True`**

Short Circuiting – `and`

- Notice that in the expression:

```
bool1 = a and (b or c)
```

- If `a` is `False`
- The rest of the expression doesn't matter

- Python will realize this, and if `a` is false won't bother with the rest of the expression

Short Circuiting – `or`

- Notice that in the expression:

```
bool1 = a or (b or c)
```

- If `a` is `True`
- The rest of the expression doesn't matter

- Python will realize this, and if `a` is true won't bother with the rest of the expression

More Practice

- Given:

  a = 4
  b = 5
  c = 6
  d = True
  e = False

```
bool1 = d and (a > b)
        False
bool2 = (not d) or (b != c)
        True
bool3 = (d and (not e)) or (a > b)
        True
bool4 = (a%b==2) and ((not d) or e)
        False
```

More More Practice

- Given:

  a = 4
  b = 5
  c = 6
  d = True
  e = False

```
bool1 = (d + d) >= 2 and (not e)
        True
bool2 = (not e) and (6*d == 12/2)
        True
bool3 = (d or (e)) and (a > b)
        False
```

Decision Making

- So, why do we care about comparison operators and logical operators so much?
- We can use them to *control* how our program works and what code it runs
  - Using decision structures

Simple 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?

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
  - What is the input? The output?
  - What is the process?

```
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

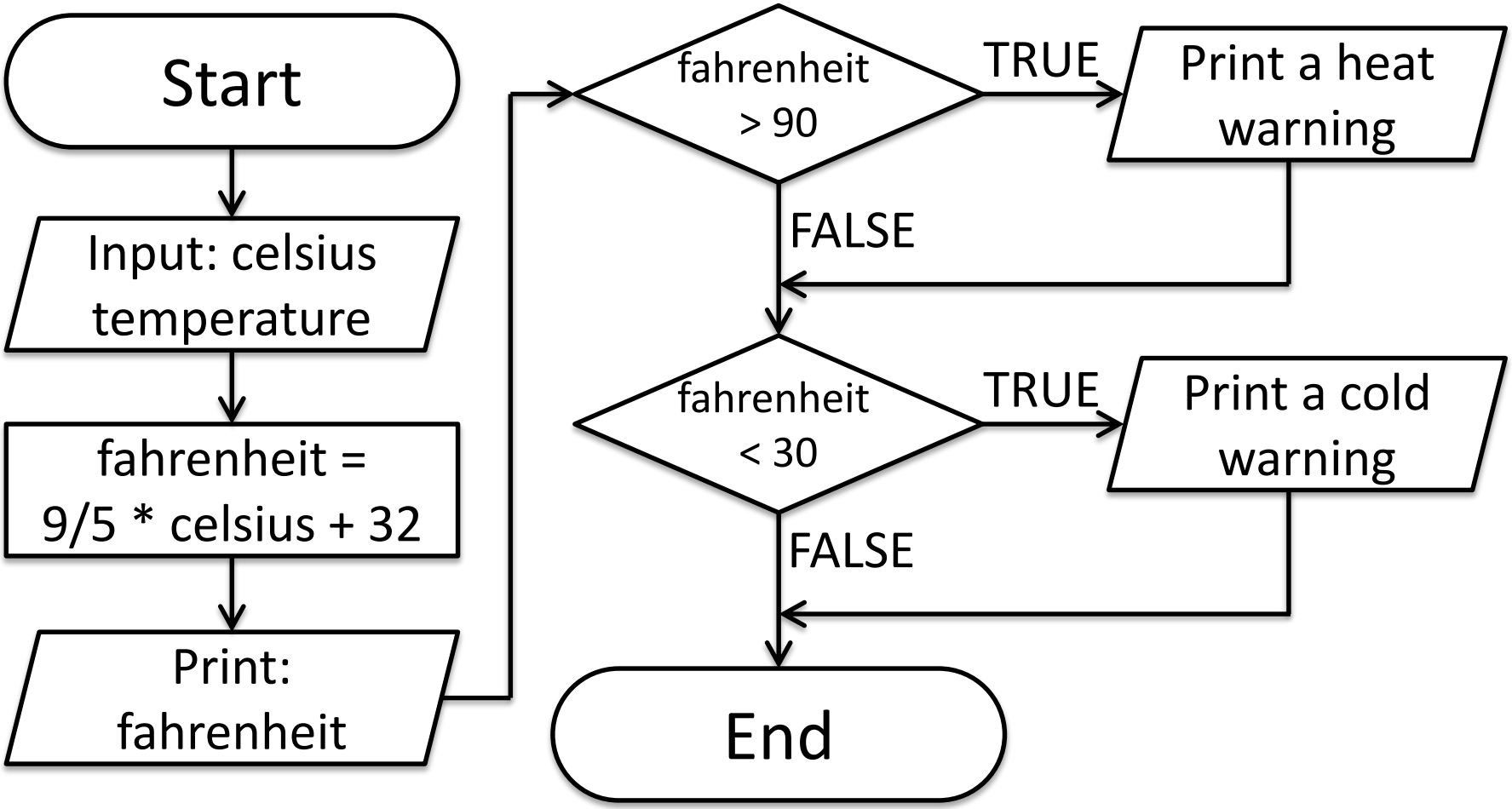
- **Input:**
  - The temperature in degrees Celsius (call it **`celsius`**)
- **Process:**
  - Calculate **`fahrenheit`** as **`9/5 * celsius + 32`**
- **Output:**
  - Temperature in Fahrenheit
  - If **`fahrenheit`** > 90
    - Display a heat warning
  - If **`fahrenheit`** < 30
    - Display a cold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

What is a Condition?

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

Formatting Selection Structures

- Each `if` statement must close with a colon (`:`)

- Code in the body (that is executed as part of the `if` statement) must be indented
  - By four spaces
  - Hitting the "Tab" key in many editors (including emacs) will automatically indent it by four spaces

"`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

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 after 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

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.

Announcements

- Your Lab 3 is meeting this week!
- Homework 2 is out
  - Due by Wednesday (Sept 21st) at 8:59:59 PM
  - You must take the Academic Integrity Quiz!
- Homework 3 will come out Wednesday night