



Introduction to Computer Science and Programming

Lecture 2

Sebastian Wandelt

Beihang University

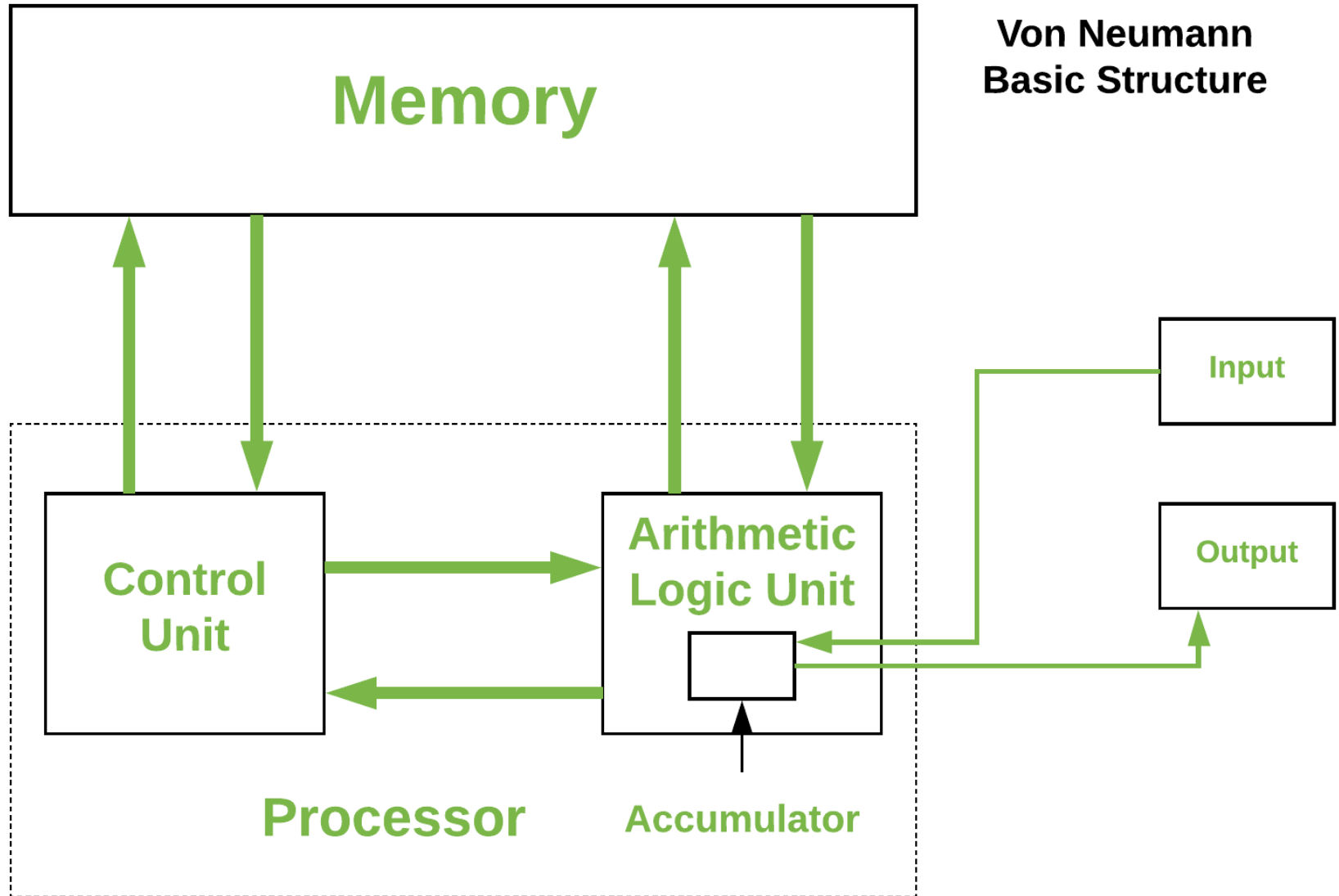
Outline

- Review
- Python
 - Variables
 - Input/Output
 - Operators
 - Making Decisions (if, elif, ...)

Review – Last lecture

What do you remember?

Computer Architecture (in one slide)



Python - Syntax - Variables

What is syntax?

- **Syntax** refers to how something is denoted
 - It is about the form/shape of elements
- This is different from the meaning, which is **Semantic**
 - It is about the association/meaning in somebody's head
- Example:
 - Syntax: „*What time is it?*“ (English)
 - Semantics: *Somebody wants to know the current time.* (Meaning)

Semantics example

- What is the problem with the following sentence?
 - **Flying planes can be dangerous?**

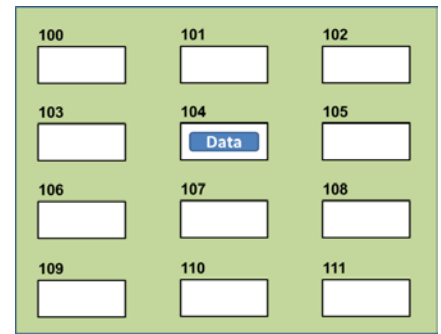


Semantics example

- What is the problem with the following sentence?
 - **Flying planes can be dangerous?**
- Different interpretations (=different semantics):
 - The action of flying a plane can be dangerous
 - Planes which fly can be dangerous
- Computer languages have only one meaning. It needs to be well defined (and understood)



Variables



- Set aside a location in memory.
- Used to store information (temporary).
- Some types of information which can be stored in variables include: integer (whole), floating point (fractional), strings (essentially any characters you can type and more)

Format (creating):

<name of variable> = <Information to be stored in the variable>

Examples (creating):

`x=5`

`name="John"`

`pi=3.14159`

`radius=2.2`

`area=pi*(radius**2)`

(Simple) data types in python

We start with **three** very simple datatypes for now:

- Integers:
 - 28, 5, 3, 2, 63464, 389575734849837538975, ...
- Floats:
 - 0.352453, 32985.2349823, -9.0, ...
- Strings:
 - "python", "p", ...



The Assignment Operator: =

- The assignment operator '=' used in writing computer programs does not have the same meaning as mathematics.
 - Don't mix them up!
- Example:
 - $y = 3$
 - $x = y$
 - $x = 6$
 - $y = 13$
- What is the end result?



Why does it make sense to have datatypes?



Why does it make sense to have datatypes?

- Datatypes provide often-used operators for working with variables of such a type
- Examples:
 - Addition, Multiplication
 - Concatenation
 - Removal
 -

Variable Naming Conventions

1. The name should be meaningful.
2. Names *must* start with a letter (Python requirement) and *should not* begin with an underscore (style requirement).
3. Names are case sensitive but avoid distinguishing variable names only by case.
4. Can't be a keyword (see next slide).

Feedback from previous year

- These concepts might be difficult to follow, for students who have not been programming before. See some more examples.
- Here you go:
 - Assign a variable `z` with the value of an integer 5
 - Let address of `john` be the same value as address of `jane`
 - Create three variables which all have the value: 3.1415
 - Assign the value of -10° C to variable `temperature`



Feedback from previous year

- These concepts might be difficult to follow, for students who have not been programming before. See some more examples.
- Here you go:
 - Assign a variable `z` with the value of an integer 5
 - `z=5`
 - Let address of `john` be the same value as address of `jane`
 - `AddressJohn=AddressJane`
 - Create three variables which all have the value: 3.1415
 - `x=3.1415`
 - `y=3.1415`
 - `z=3.1415`
 - Assign the value of -10° C to variable `temperature`
 - `Temperature=-10`
 - **Be careful about °C !**

Keywords In Python¹

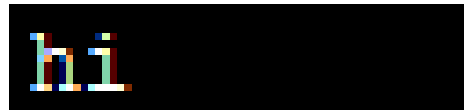
and	del	from	not	while
as	elif	global	or	with
assert	else	if	pass	yield
break	except	import	print	
class	exec	in	raise	
continue	finally	is	return	
def	for	lambda	try	

¹ From “*Starting out with Python*” by Tony Gaddis

Python - Syntax – Input/Output

Displaying Output Using The `print()` Function

- This function takes zero or more arguments (inputs)
 - Multiple arguments are separated with commas
 - `print()` will display all the arguments followed by a blank line (move the cursor down a line).
- Simple Example:
`print("hi")`

A screenshot of a terminal window with a black background. The text "hi" is displayed in a colorful, pixelated font with shades of blue, green, and red. The text is positioned on the left side of the terminal, and the rest of the terminal area is empty.

Printing multiple things

- Simply separate things by a comma
- Examples:

```
x=5
```

```
print("The value of x is",x)
```

```
name = "John"
```

```
print(name)
```

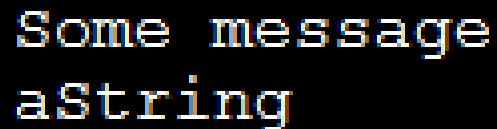
```
print("name")
```



`print("... ")` Vs. `print(<name>)`

- Enclosing the value in brackets with quotes means the value in between the quotes will be literally displayed onscreen.
- Excluding the quotes will display the contents of a memory location.
- Example:

```
aString = "Some message"  
print(aString)  
print("aString")
```



```
Some message  
aString
```

Input

- The computer program getting *string information* from the user.
- Strings cannot be used for calculations (information for getting numeric input will be provided shortly).

- **Format:**

`<variable name> = input()`

OR

`<variable name> = input("<Prompting message>")`

- **Example:**

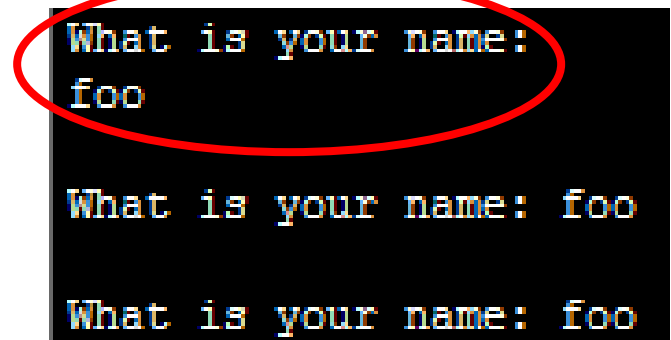
```
print("What is your name: ")
```

```
name = input()
```

OR

```
name = input("What is your name: ")
```

Avoid alignment issues such as this



```
What is your name:
foo

What is your name: foo

What is your name: foo
```

Python – Operators and Conversion

This is about working with variables!

An operator modifies a variable (or more than one) => operation on variables

Arithmetic Operators

Operator	Description	Example
=	Assignment	num = 7
+	Addition	num = 2 + 2
-	Subtraction	num = 6 - 4
*	Multiplication	num = 5 * 4
/	Division	num = 9 / 2 4.5
//	Integer division	num = 9 // 2 4
%	Modulo	num = 9 % 2 1
**	Exponent	num = 9 ** 2 81

Order Of Operation

- First level of precedence: top to bottom
- Second level of precedence
 - If there are multiple operations that are on the same level then precedence goes from left to right.

()	Brackets (inner before outer)
**	Exponent
*, /, //, %	Multiplication, division, modulo
+, -	Addition, subtraction
=	Assignment

Example

$x = 5 * 2 ** 3$

Vs.

$x = (5 * 2) ** 3$

Order Of Operation And Style

- Even for languages where there are clear rules of precedence (e.g., Java, Python) it's good style to explicitly bracket your operations and use blank spaces as separators.

`x = (a * b) + (c / d)`

- It not only makes it easier to read complex formulas but also a good habit for languages where precedence is not always clear (e.g., C++, C).

Converting Between Different Types Of Information

- Example motivation: you may want numerical information to be stored as a string (for built-in string functions e.g., check if a string consists only of numbers) but also you want to perform calculations).
- Some of the conversion mechanisms (functions) available in Python:

Format:

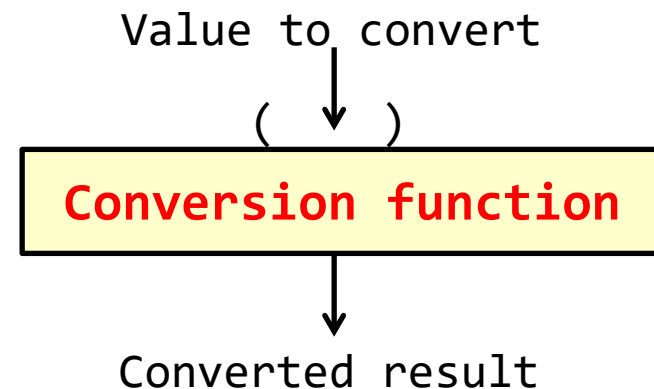
```
int(<value to convert>)  
float(<value to convert>)  
str(<value to convert>)
```

Examples:

Program name: convert1.py

```
x = 10.9  
y = int(x)  
print(x,y)
```

```
10.9 10
```



Converting Types: Extra Practice

- Determine the output of the following program:

```
print(12+33)
print('12'+ '33')
x = 12
y = 21
print(x+y)
print(str(x)+str(y))
```



Don't know what datatype your variable has?

```
x=5
```

```
>>>type(x)
```

```
integer
```

What can you do so far?

- How to get user input in Python
- How to provide output to the user
- How to deal with different types of variables storing information (optional/extra for now)
- How/why to convert between different types

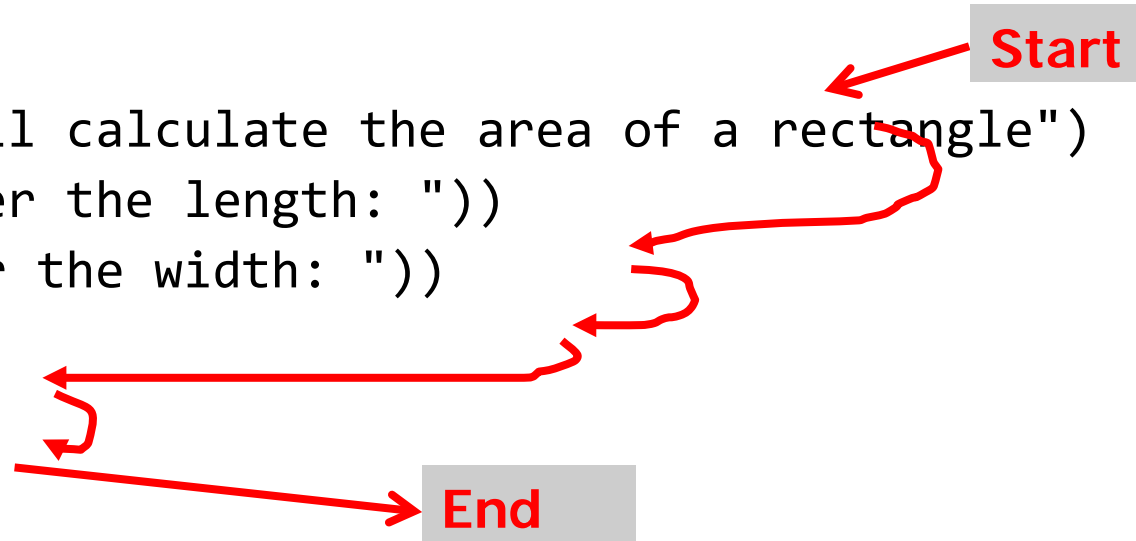
**In summary, the programs you can write so far are:
Top-down linear programs.**

Making Decisions In Python

In this section you will learn how to have your programs choose between alternative courses of action.

Recap: Programs You've Seen So Far Produces Sequential Execution

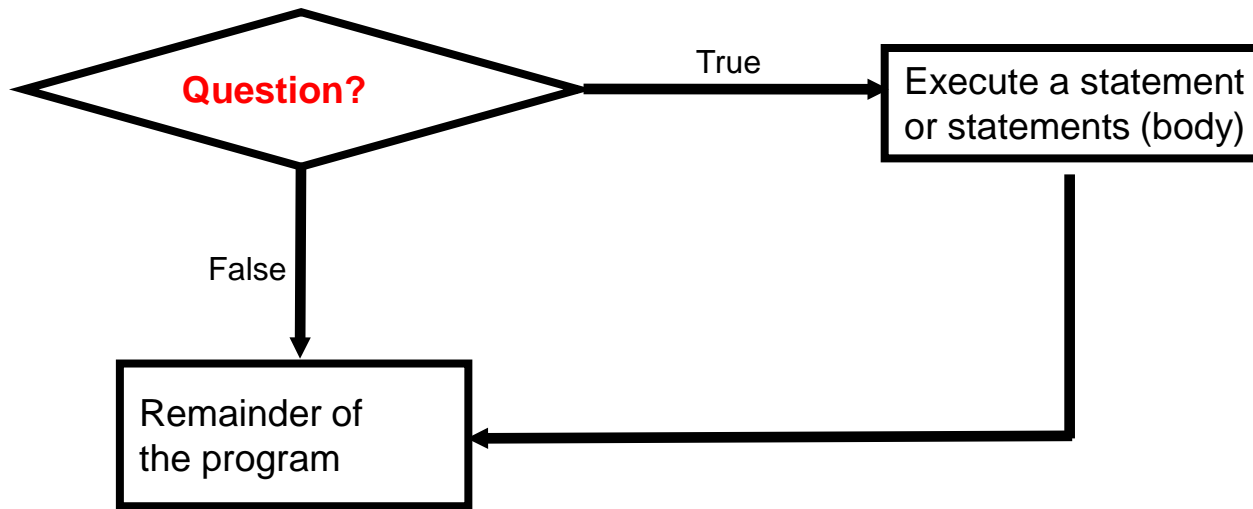
```
print ("This program will calculate the area of a rectangle")
length = int(input("Enter the length: "))
width = int(input("Enter the width: "))
area = length * width
print("Area: ", area)
```



Programming: Decision Making Is Branching

- Decision making is choosing among alternates (branches).
- Why is it needed?
 - When alternative courses of action are possible and each action may produce a different result.
- In terms of a computer program the choices are stated in the form of a question that only yield an answer that is either true or false
 - Although the approach is very simple, modeling decisions in this fashion is a very useful and powerful tool.

Decision Making With An 'If'



New Terminology

- **Boolean expression:** An expression that must work out (evaluate to) to either a true or false value.
 - e.g., it is over 45 Celsius today
 - e.g., the user correctly entered the password
- **New term, body:** A block of program instructions that will execute under a specified condition (for branches the body executes when the Boolean expression evaluates to/works out to true)

```
name=input("Name: ")  
print(name)
```

- The 'body' is indented (4 spaces)

This/these instruction/instructions run when you give the Python interpreter the name of a file, the 'body' of the Python program runs

The 'If' Construct

- Decision making: checking if a condition is true (in which case something should be done).

- **Format:**

(General format)

```
if (Boolean expression):
```

```
    body
```

(Detailed structure)

```
if (<operand> <relational operator> <operand>):
```

```
    body
```

Boolean expression



Note: Indenting the body is mandatory!



The 'If' Construct (2)

- **Example:**

```
age = int(input("Age: "))  
if (age >= 18):  
    print("You are an adult")
```

Note On Indenting (1)

- In Python indenting is mandatory in order to determine which statements are part of a body (**syntactically required** in Python).

Single statement body

```
if (num == 1):  
    print("Body of the if")  
print("After body")
```

Multi-statement body (program 'if2.py')

```
if (num == 1):  
    print("Body of the if")  
    print("After body")
```

Note On Indenting (1)

- A (slightly) larger example program for calculating taxes:

```
taxDeduction = 0
taxRate = 0.2
income = float(input("What is your annual income: "))
if (income < 10000):
    print("Eligible for social assistance")
    taxDeduction = 100
tax = (income * taxRate) - taxDeduction
if tax<0:
    tax=0
print("Tax owed:",tax,"yuan")
```


Note On Indenting (2)

- A “sub-body” (IF-branch) is indented by an additional 4 spaces (8 or more spaces) if one IF-branch is inside the body of another IF-branch (this is called ‘nesting’ – more details later).
 - Usually, you simply press TAB key one time for indentation

New Terminology

- **Operator/Operation**: action being performed
- **Operand**: the item or items on which the operation is being performed.

Examples:

2 + 3

2 * (-3)

Allowable **Operands** For Boolean Expressions

Format:

if (**operand** relational operator **operand**):

Example:

if (**age** >= **18**):

The operand types in the lecture so far are:

- integer
- floats
- String
- Boolean (True or False)

Make sure that you are comparing operands of the same type or at the very least they must be comparable!

Allowable **Relational Operators** For Boolean Expressions

if (operand **relational operator** operand) then

Python operator	Mathematical equivalent	Meaning	Example
<	<	Less than	5 < 3
>	>	Greater than	5 > 3
==	=	Equal to	5 == 3
<=	≤	Less than or equal to	5 <= 5
>=	≥	Greater than or equal to	5 >= 4
!=	≠	Not equal to	x != 5

Common Mistake

- Do not confuse the equality operator '==' with the assignment operator '='.
- **Example (Python syntax error)¹:**

```
if (num = 1):    # Not the same as    if (num == 1):
```

To be extra safe some programmers put unnamed constants on the left hand side of an equality operator (which always/almost always results in a syntax error rather than a logic error if the assignment operator is used in place of the equality operator).

- A way of producing syntax rather than a logic error:

```
if (1 = num)
```

¹ This is not a syntax error in all programming languages so don't get complacent and assume that the language will automatically "take care of things" for you.

An Application Of Branches

- Branching statements can be used to check the validity of data (if the data is correct or if the data is a value that's allowed by the program).
- **General structure:**

```
if (error condition has occurred)  
    React to the error (at least display an error message)
```
- **Example:**

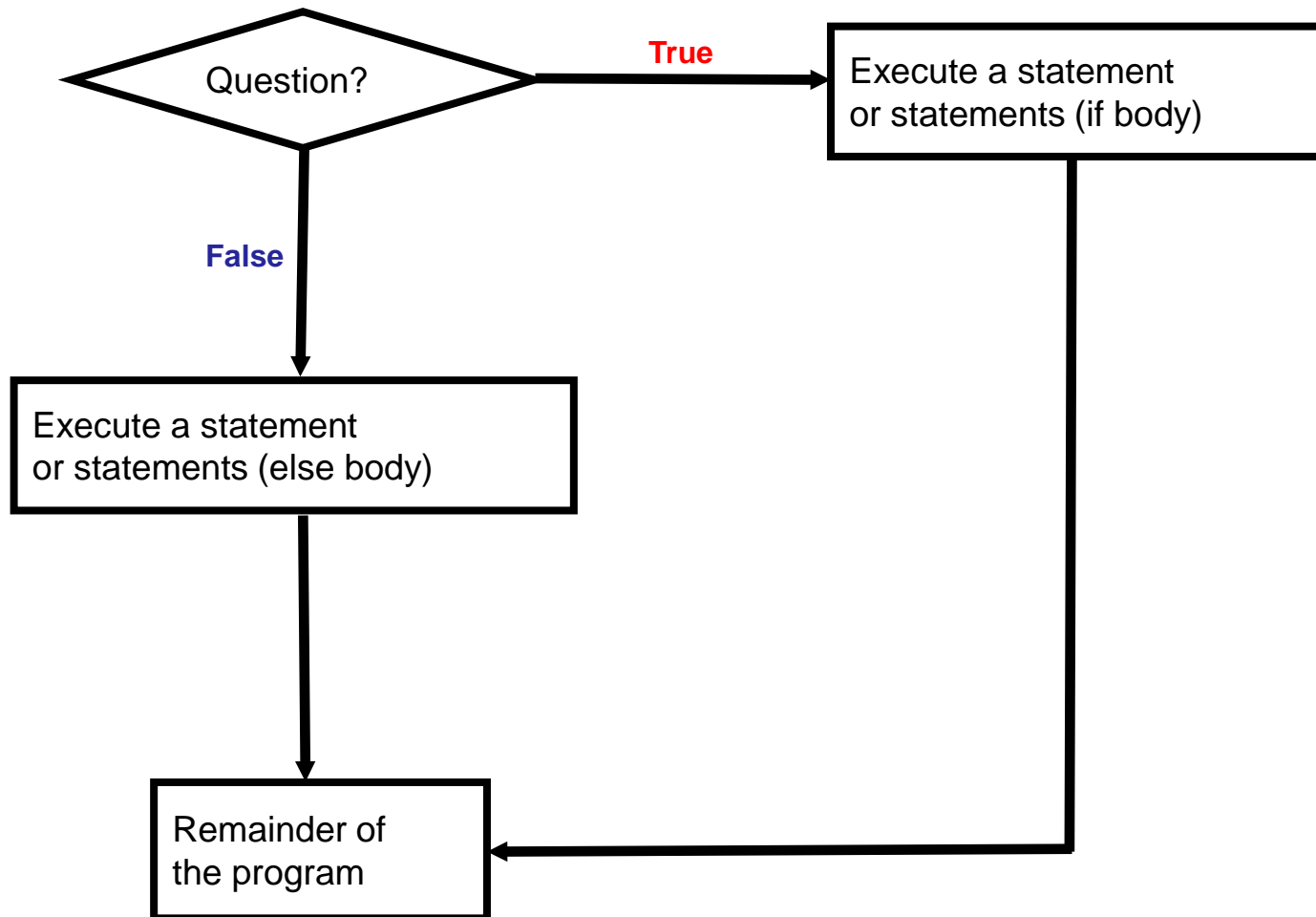
```
if (age < 0):  
    print("Age cannot be a negative value")
```

Tip: if data can only take on a certain value (or range) do not automatically assume that it will be valid. Check the validity of range before proceeding onto the rest of the program.

Decision Making With An 'If': Summary

- Used when a question (Boolean expression) evaluates only to a true or false value (Boolean):
 - If the question evaluates to true then the program reacts differently. It will execute the body after which it proceeds to the remainder of the program (which follows the if construct).
 - If the question evaluates to false then the program doesn't react differently. It just executes the remainder of the program (which follows the if construct).

Decision Making With An 'If-Else'



The If-Else Construct

- Decision making: checking if a condition is true (in which case something should be done) but unlike 'if' *also reacting if the condition is not true (false)*.
- **Format:**
 if (operand relational operator operand):
 body of 'if'
 else:
 body of 'else'
 additional statements

If-Else Construct (2)

- **Partial example:**

```
age=int(input("How old are you?"))
if (age < 18):
    print("Not an adult")
else:
    print("Adult")
print("Tell me more about yourself!")
```

```
[csc branches 13 ]> python if_else1.py
How old are you? 17← If case
Not an adult
Tell me more about yourself
[csc branches 14 ]>
[csc branches 14 ]> python if_else1.py
How old are you? 27← Else case
Adult
Tell me more about yourself
[csc branches 15 ]> █
```


Quick Summary: If Vs. If-Else

- If:
 - Evaluate a Boolean expression (ask a question).
 - If the expression evaluates to true then execute the 'body' of the `if`.
 - No additional action is taken when the expression evaluates to false.
 - Use when your program is supposed to react differently only when the answer to a question is true (and do nothing different if it's false).
- If-Else:
 - Evaluate a Boolean expression (ask a question).
 - If the expression evaluates to true then execute the 'body' of the `if`.
 - If the expression evaluates to false then execute the 'body' of the `else`.
 - That is: *Use when your program is supposed to react differently for both the true and the false cases.*

Logical Operations

- There are many logical operations but the three most commonly used in computer programs include:
 - Logical *and*
 - Logical *or*
 - Logical *not*

Logical “and”

- The popular usage of the logical *and* applies when *ALL* conditions must be met.
 - Example:
 - Pick up your son and pick up your daughter after school today.
- 
- Logical *and* can be specified more formally in the form of a truth table.

Truth table (and)		
C1	C2	C1 and C2
False	False	False
False	True	False
True	False	False
<i>True</i>	<i>True</i>	<i>True</i>

Logical “and”: Three Input Truth Table

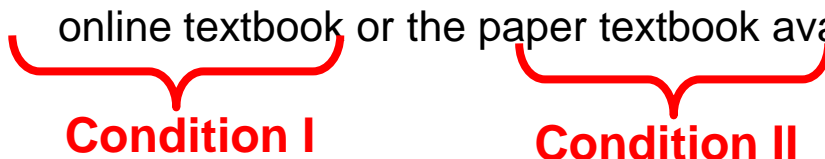
Truth table			
C1	C2	C3	C1 and C2 and C3
False	False	False	False
False	False	True	False
False	True	False	False
False	True	True	False
True	False	False	False
True	False	True	False
True	True	False	False
<i>True</i>	<i>True</i>	<i>True</i>	<i>True</i>

Evaluating Logical “and” Expressions

- True **and** True **and** True
- False **and** True **and** True
- True **and** True **and** True **and** False



Logical “or”

- The correct everyday usage of the logical *or* applies when *ATLEAST* one condition must be met.
 - Example:
 - You are using additional recommended resources for this course: the online textbook or the paper textbook available in the bookstore.
- 
- Condition I** **Condition II**
- Similar to *and*, logical *or* can be specified more formally in the form of a truth table.

Truth table		
C1	C2	C1 or C2
<i>False</i>	<i>False</i>	<i>False</i>
False	True	True
True	False	True
True	True	True

Logical “or”: Three Input Truth Table

Truth table			
C1	C2	C3	C1 or C2 or C3
<i>False</i>	<i>False</i>	<i>False</i>	<i>False</i>
False	False	True	True
False	True	False	True
False	True	True	True
True	False	False	True
True	False	True	True
True	True	False	True
True	True	True	True

Evaluating Logical “or” Expressions

- True **or** True **or** True
- False **or** True **or** True
- False **or** False **or** False **or** True



Logical “not”

- The everyday usage of logical not negates (or reverses) a statement.
- Example:
 - I am finding this class quite stimulating and exciting

Statement (logical condition)

.....*not!!!*

**Negation of the
statement/condition**

- The truth table for logical not is quite simple:

Truth table	
S	not S
False	True
True	False



Evaluating More Complex Logical Expressions

Order of operation (left to right evaluation if the 'level' is equal)

1. Brackets (inner first)
2. not
3. and
4. or

Evaluating More Complex Logical Expressions

- True **or** True **and** True
- **not** (False **or** True) **or** True
- (False **and** False) **or** (False **and** True)
- **not not not not** True
- **not not not** False



Extra Practice

Assume the variables $a = 2$, $b = 4$, $c = 6$

For each of the following conditions indicate whether the final value is true or false.

Expression	Final result
$a == 4$ or $b > 2$	
$6 \leq c$ and $a > 3$	
$1 \neq b$ and $c \neq 3$	
$a > -1$ or $a \leq b$	
not ($a > 2$)	



Logic Can Be Used In Conjunction With Branching

- Typically the logical operators AND, OR are used with multiple conditions/Boolean expressions:
 - If multiple conditions *must all be met* before the body will execute. (AND)
 - If *at least one condition* must be met before the body will execute. (OR)
- The logical NOT operator can be used to check for inequality (not equal to).
 - E.g., If it's true that the user *did not* enter an invalid value the program can proceed.

The “NOT” Operator

- **Format:**

```
if not (Boolean expression):  
    body
```

- **Name of the online example: `if_not.py`**

- (An equivalent solution can be implemented using the inequality operator '!=')

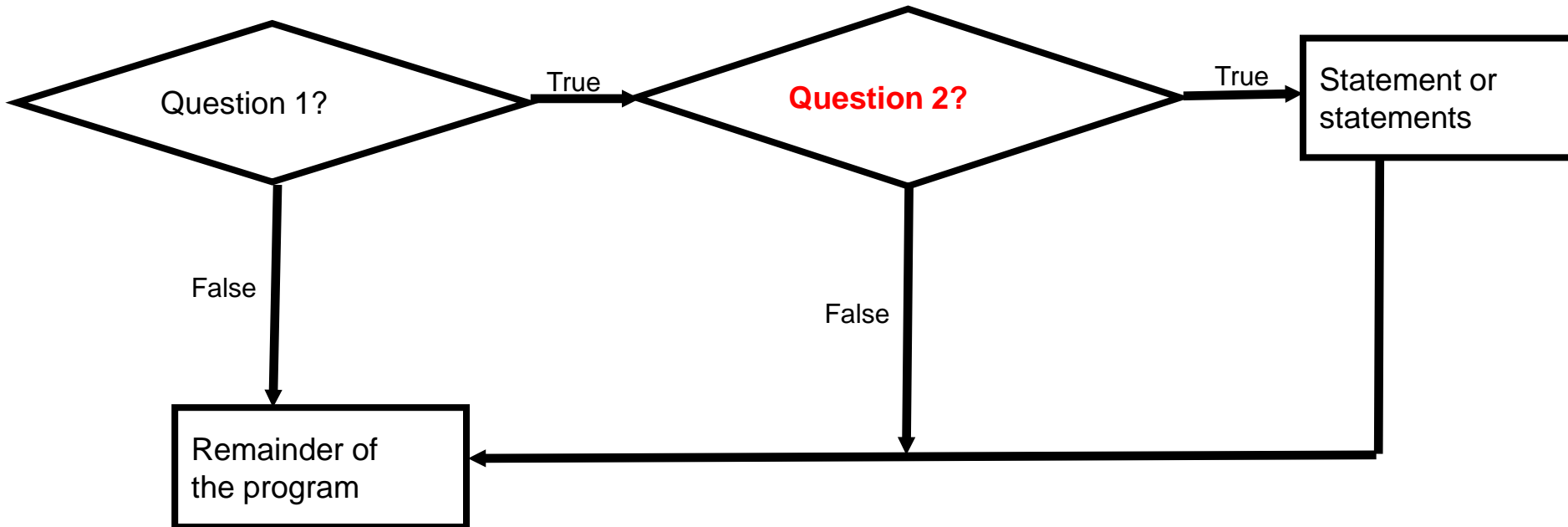
```
SYSTEM_PASSWORD = "password123"  
userPassword = input("Password: ")  
if not (userPassword == SYSTEM_PASSWORD):  
    print("Using logical NOT-operator: Wrong password")
```


Quick Summary: Using Multiple Expressions

- Use multiple expressions when multiple questions must be asked and the result of expressions are related:
- AND (strict: all must apply):
 - All Boolean expressions must evaluate to true before the entire expression is true.
 - If any expression is false then whole expression evaluates to false.
- OR (less restrictive: at least one must apply):
 - If any Boolean expression evaluates to true then the entire expression evaluates to true.
 - All Boolean expressions must evaluate to false before the entire expression is false.
- Not:
 - Negates or reverses the logic of a Boolean expression
 - May sometimes be super ceded by the use of an inequality operator

Nested Decision Making

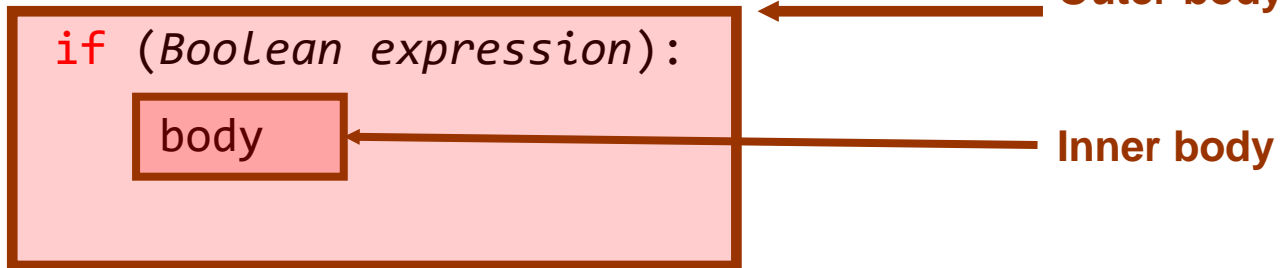
- Decision making is dependent.
- The first decision must evaluate to true ("gate keeper") before successive decisions are even considered for evaluation.



Nested Decision Making

- One decision is made inside another.
- Outer decisions must evaluate to true before inner decisions are even considered for evaluation.
- **Format:**

`if (Boolean expression):`



Nested Decision Making (2)

- **Partial example:** nesting.py

```
if (income < 10000):  
    if (citizen == 'y'):  
        print("This person can receive social assistance")  
        taxCredit = 100  
tax = (income * TAX_RATE) - taxCredit
```

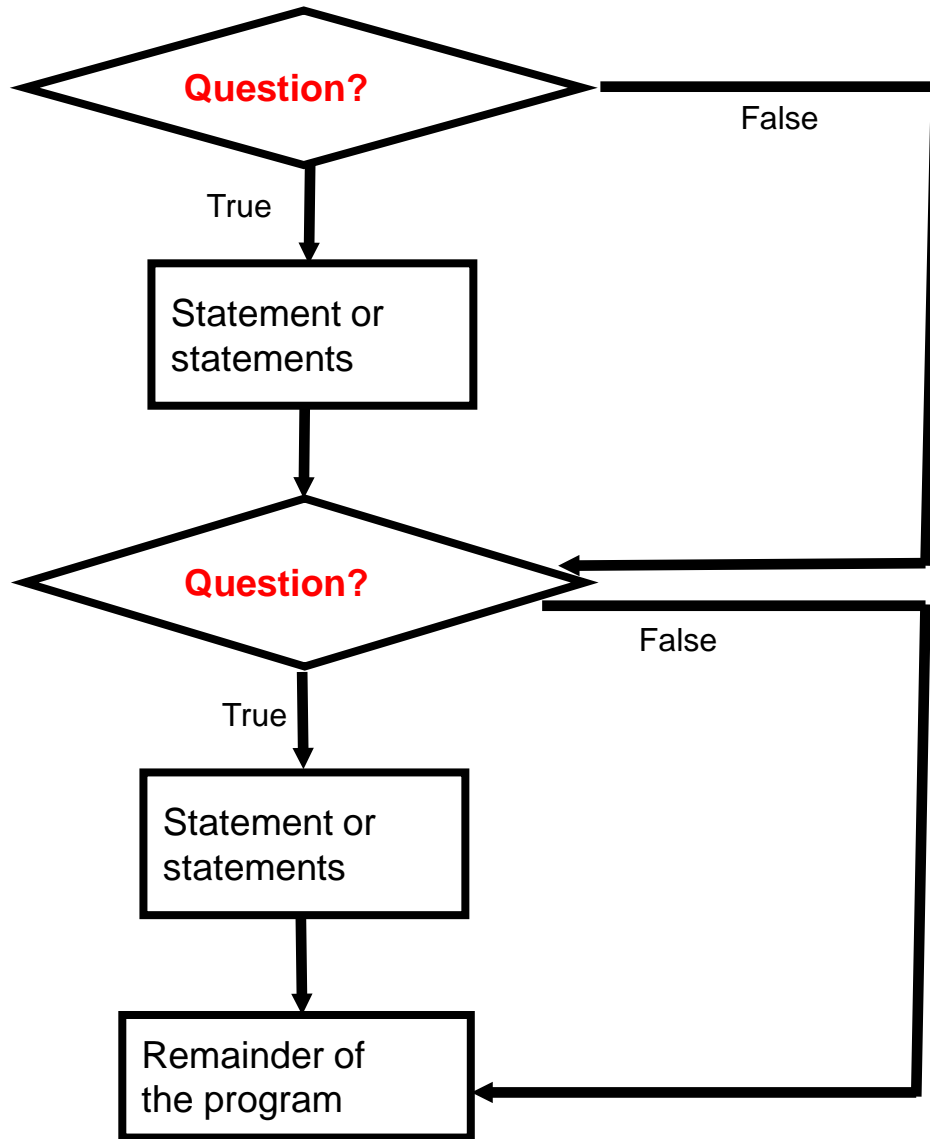
- What's the difference between employing nested decision making and a logical AND?



Decision-Making With Multiple Alternatives/Questions

- IF (single question)
 - Checks a condition and executes a body if the condition is true
- IF-ELSE (single question)
 - Checks a condition and executes one body of code if the condition is true and another body if the condition is false
- Approaches for multiple (two or more) questions
 - **Multiple IF's**
 - **IF-ELIF-ELSE**

Decision Making With **Multiple If's**



Multiple If's: Non-Exclusive Conditions

- Any, all or none of the conditions may be true (independent)
- Employ when a series of independent questions will be asked
- **Format:**

if (*Boolean expression 1*):

body 1

if (*Boolean expression 2*):

body 2

:

statements after the conditions

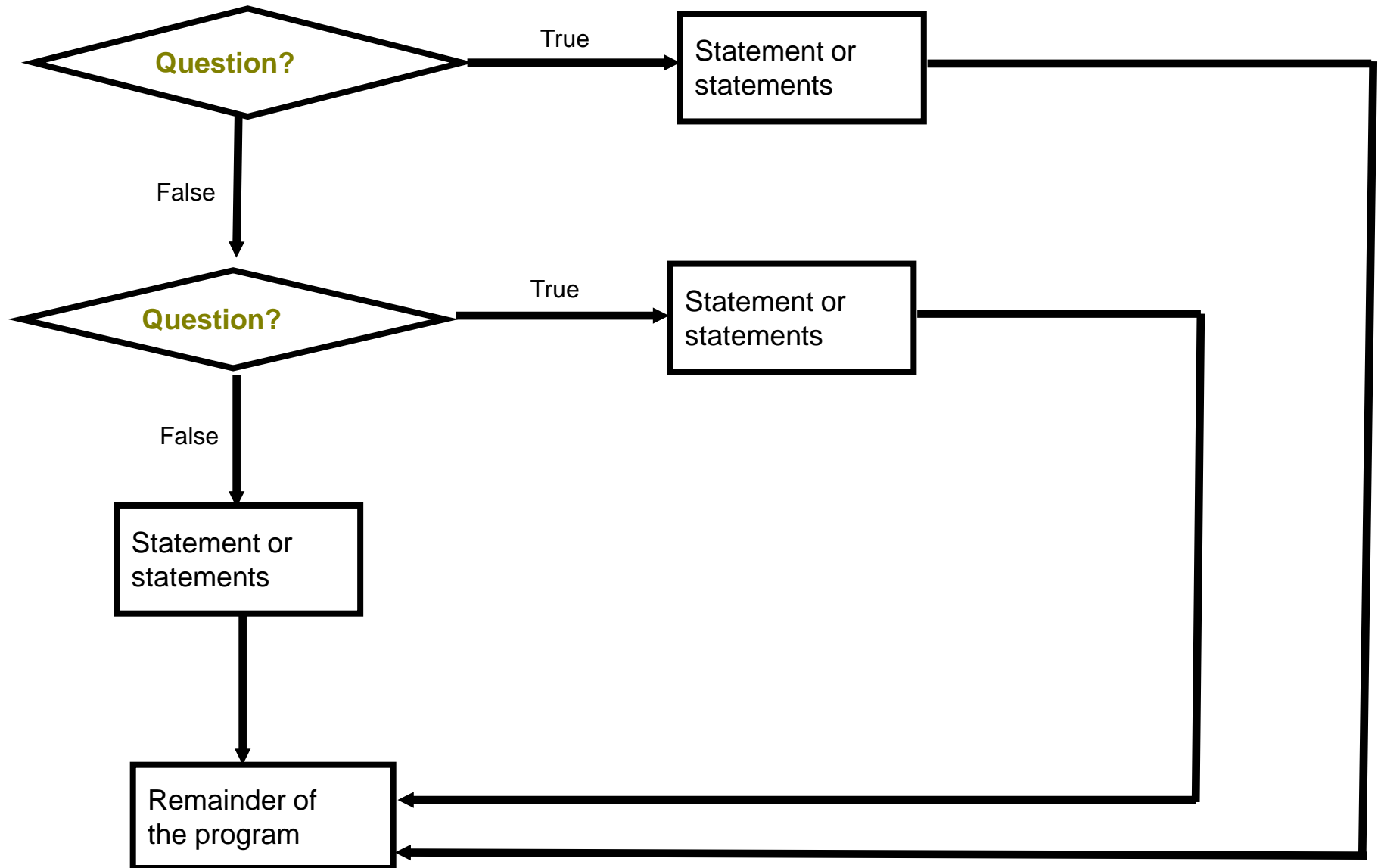
Multiple If's: Mutually Exclusive Conditions

- At most *only one* of many conditions can be true
 - Can be implemented through multiple if's
- 
- Inefficient combination!**

- **Example:**

```
if (gpa == 4):  
    letter = 'A'  
  
if (gpa == 3):  
    letter = 'B'  
  
if (gpa == 2):  
    letter = 'C'  
  
if (gpa == 1):  
    letter = 'D'  
  
if (gpa == 0):  
    letter = 'F'
```


Decision Making With **If-Elif-Else**



Multiple **If-Elif-Else**: Use With Mutually Exclusive Conditions

- **Format:**

if (*Boolean expression 1*):

body 1

elif (*Boolean expression 2*):

body 2

else:

body n

statements after the conditions

Mutually exclusive

- One condition evaluating to true excludes other conditions from being true
- Example: having your current location as 'Calgary' excludes the possibility of the current location as 'Edmonton', 'Toronto', 'Medicine Hat'

If-Elif-Else: Mutually Exclusive Conditions (Example)

- Better xample:

```
if (gpa == 4):  
    letter = 'A'
```

```
elif (gpa == 3):  
    letter = 'B'
```

```
elif (gpa == 2):  
    letter = 'C'
```

```
elif (gpa == 1):  
    letter = 'D'
```

```
elif (gpa == 0):  
    letter = 'F'
```

```
else:
```

```
    print("GPA must be one of '4', '3', '2', '1' or '0'")
```

This approach is more efficient when at most only one condition can be true.

Extra benefit:

The body of the else executes only when all the Boolean expressions are false. (Useful for error checking/handling).

Recap: What Decision Making Mechanisms Are Available /When To Use Them

Mechanism	When To Use
If	Evaluate a Boolean expression and execute some code (body) if it's true
If-else	Evaluate a Boolean expression and execute some code (first body: 'if') if it' s true, execute alternate code (second body: 'else') if it's false
Multiple if' s	Multiple Boolean expressions need to be evaluated with the answer for each expression being independent of the answers for the others (non-exclusive). Separate instructions (bodies) can be executed for each expression.
If-elif-else	Multiple Boolean expressions need to be evaluated but zero or at most only one of them can be true (mutually exclusive). Zero bodies or exactly one body will execute. Also it allows for a separate body (else-case) to execute when all the if-elif Boolean expressions are false.

Final task

- Implement a program which:
 1. Reads two numbers (x and y) from the user and prints the smaller of the two numbers.
 2. If both numbers are equal, the program should just print that they are equal.
 3. After termination, the program should thank the user for using the program

Hint

- Feedback from the previous year:

*Understanding these slides is critical for **all** future lectures. Therefore, you should review these slides after the class one time, to ensure you get the key concepts and understand the syntax.*

Thank you very much!

**If you have any questions,
please get in touch with me:
wandelt@buaa.edu.cn**