Making Decisions In Python

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

Decision Making Is All About Choices

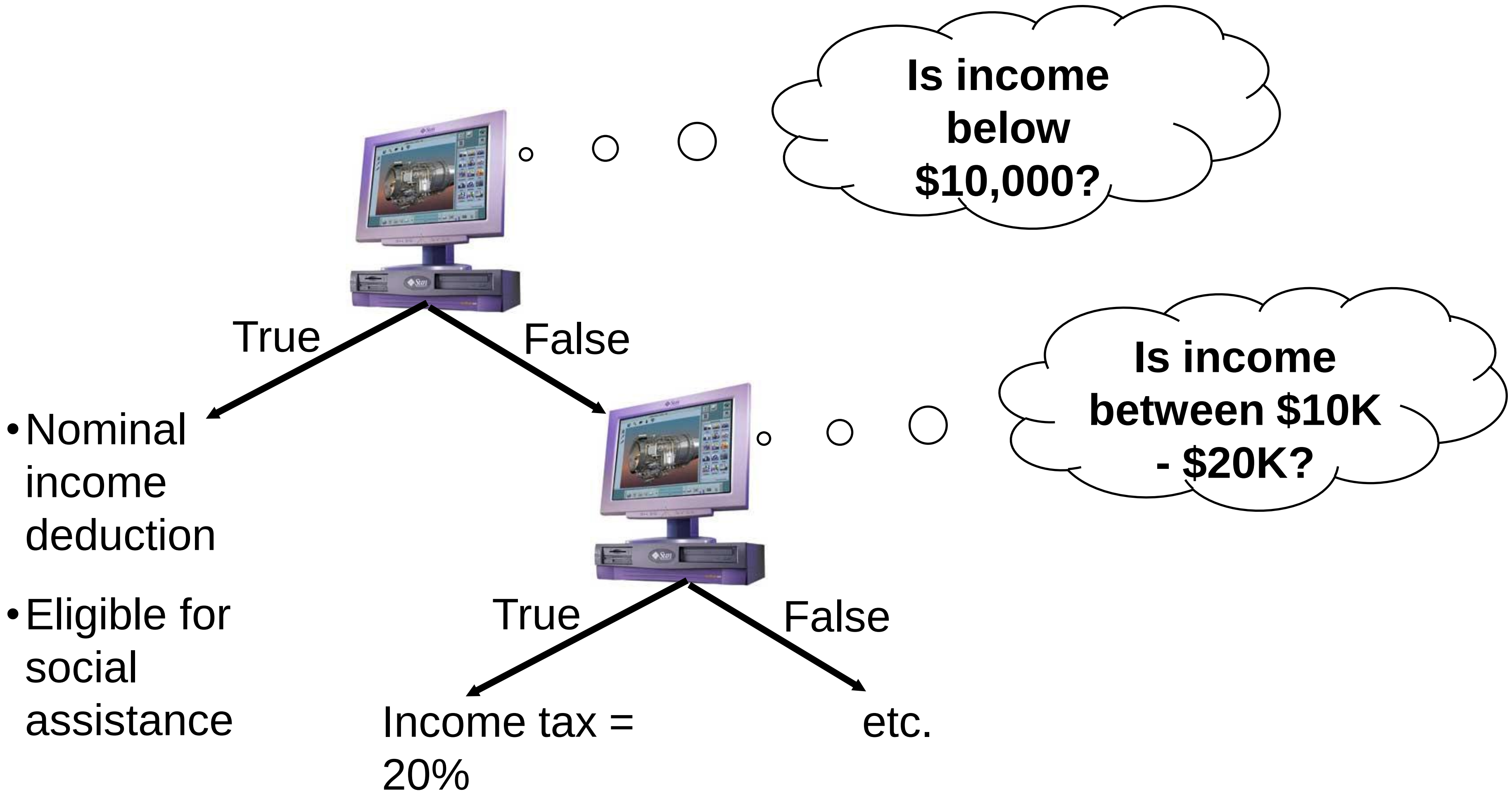
My next vacation?

Images: courtesy of James Tam

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.

High Level View Of Decision Making For The Computer


Is income below $10,000?
True
False
•Nominal income deduction
•Eligible for social assistance
Is income between $10K - $20K?
True
False
Income tax = 20%
etc.

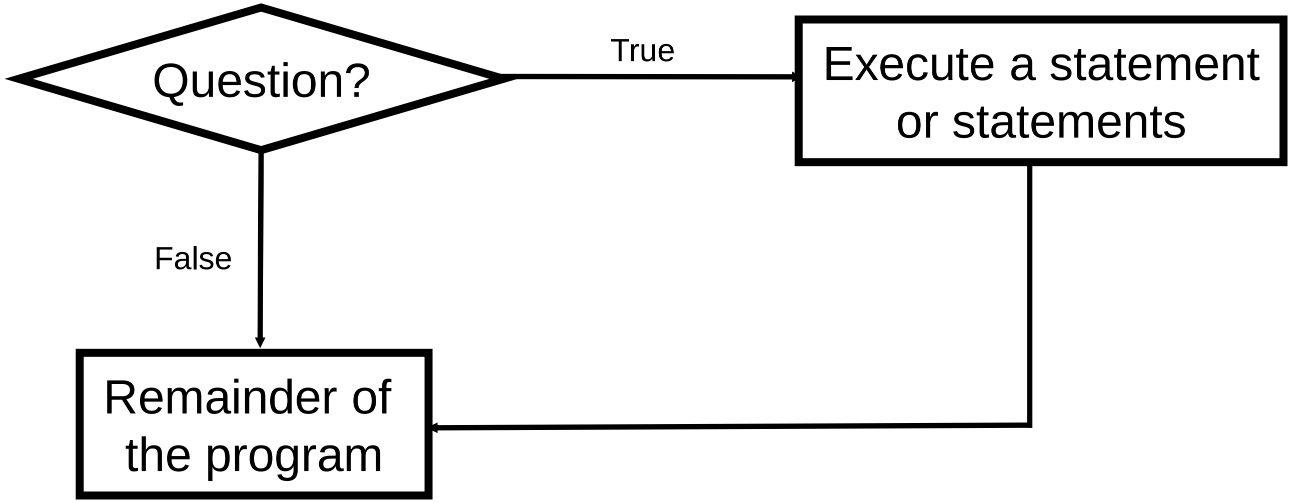
How To Determine If Branching Can Be Applied

- Under certain circumstances or conditions events will occur (the program reacts in a certain way).
- Examples:
  - Users who don't meet the age requirement of the website will not be allowed to sign up (conversely users who do meet the age requirement will be allowed to sign up).
  - Employees who are deemed as too inexperienced and too expensive to keep on staff will be laid off.
  - When a person clicks on a link on a website for a particular location then a video will play showing tourist 'hot spots' for that location.
  - If a user enters invalid age information (say negative values or ones greater than 114) then the program will display an error message.

Decision-Making In Python (Programming)

- Decisions are questions with answers that are either true or false (Boolean expressions) e.g., Is it true that the variable ‘num’ is positive?
- The program branches one way or another depending upon the answer to the question (the result of the Boolean expression).
- Decision making/branching constructs (mechanisms) in Python:
  - If
  - If-else
  - If-elif-else

Decision Making With An 'If'



Terminology

- The ‘body’ of a branch is the part of the program that will execute when the Boolean expression evaluates to true.

Travel website

Click link for Middle East? (T/F)

Click on link for North America? (T/F)

Body: show video for Abu Dhabi

Body: show video for Lethbridge

Images: courtesy of James Tam

Terminology

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

Example:
2 + 3
2 * (-3)

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
```

(Specific structure)

```
if (operand relational operator operand):
    body
```

Boolean expression

Note: Indenting the body is mandatory!

The ‘If’ Construct (2)

- **Example:**

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

Allowable Operands For Boolean Expressions

Format:

If (**operand** relational operator **operand**) then:

Example:

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

Some operand types

- integer
- floats (real)
- String

Make sure that you are comparing operands of the same type!

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).
- Usually (always?) a syntax error:

  if (1 = num)

1 This 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.

A Similar Mistake

- Example (Python) logic error:

```
num == 1   Not the same as    num = 1
```

If (Simple Body)

- Body of the if consists of a single statement
- **Format:**

```
if (Boolean expression):
    s1
s2
```

Body

Example:

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

Indenting is used to indicate what statement is the body

If (Compound Body)

- Body of the if consists of multiple statements
- **Format:**

```
if (*Boolean expression*):
    $s^1$
    $s^2$
    :
    $s^n$
$s^{n+1}$
```

Body

End of the indenting denotes the end of decision-making

If (Compound Body(2))

- **Program name:** if1.py
- **Example:**

```
taxCredit = 0
taxRate = 0.2
income = float(input("What is your annual income: "))
if (income < 10000):
    print ("Eligible for social assistance")
    taxCredit = 100
tax = (income * taxRate) - taxCredit
print ("Tax owed $%.2f" %tax)
```

An Application Of Branches

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

```
if (error condition has occurred)
    React to the error
```

- Example:

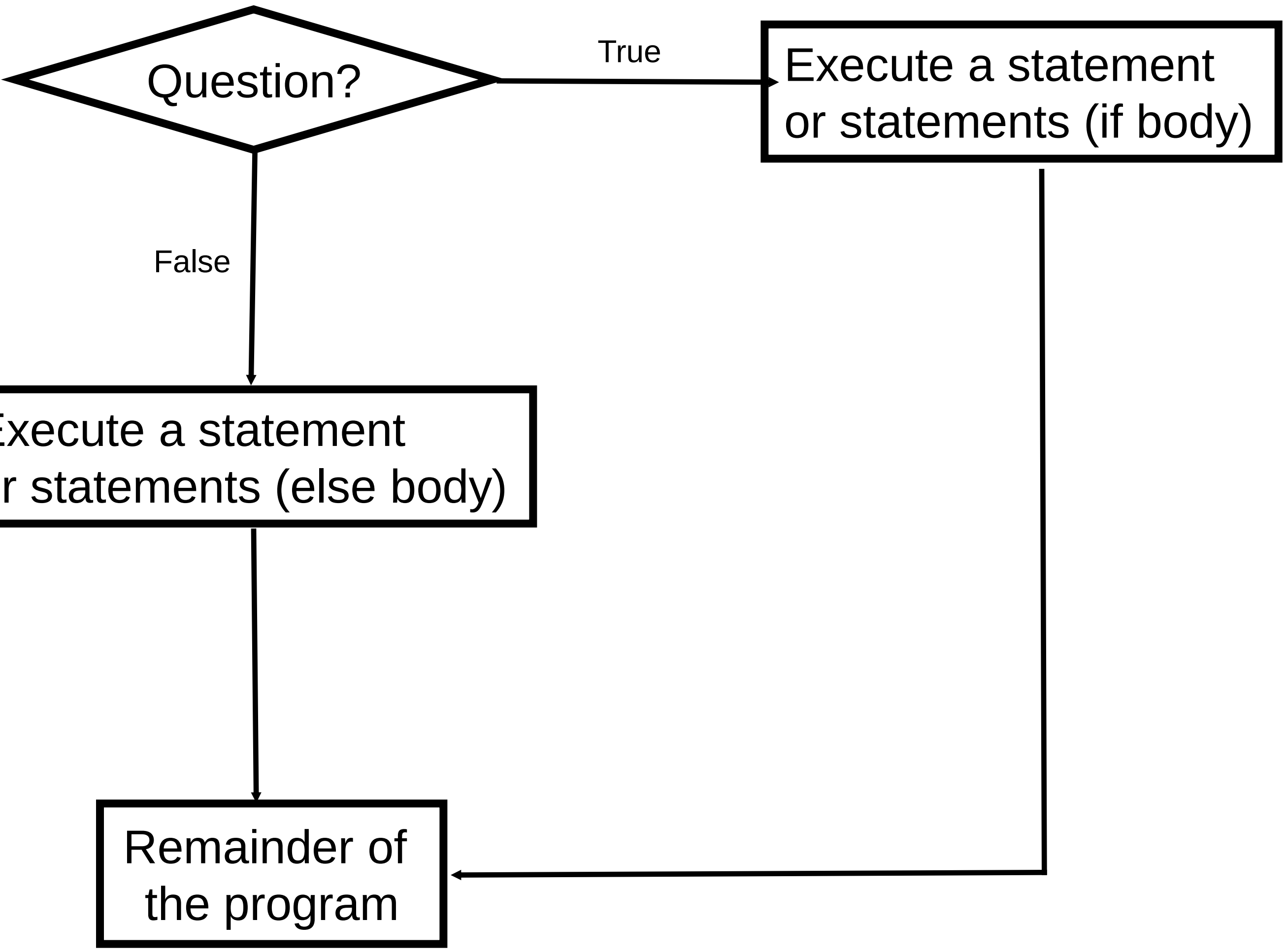
```
if (age < 0):
    print("Age cannot be a negative value”)
```

JT's tip: if data can only take on a certain value (or range) do not automatically assume that it will be valid. Check the validity 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'


Question?
True
Execute a statement or statements (if body)
False
Execute a statement or statements (else body)
Remainder of the program

The If-Else Construct

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

- **Program name:** if_else1.py
- **Example:**

```
if (age < 18):
    print ("Not an adult")
else:
    print ("Adult")
print ("Tell me more about yourself")
```

Lesson: Read Things The Way *They're Actually Stated* (Instead of How You Think They're Stated)

You this read wrong

Lesson: Read Things The Way *They're Actually Stated* (Instead of How You Think They're Stated)

- **Example: Actual Code**

```
if (age >= 18):
    print (“Adult")
else:
    print (“Not an adult")
print ("Tell me more about yourself")
```

Lesson: Read Things The Way They're Actually Stated (Instead of How You Think They're Stated)

- **Example: How One Class Interpreted The Code (Optical Illusion?)**

```
if (age >= 18):
    print (“Adult")
else:
    print (“Not an adult")
    print ("Tell me more about yourself")
```

JT's tip: one way of making sure you read the program code the way it actually is written rather than how you think it should be is to take breaks from writing

If-Else (Compound Body(2))

- **Program name:** if_else2.py
- **Example:**

```
if (income < 10000):
    print ("Eligible for social assistance")
    taxCredit = 100
    taxRate = 0.1
else:
    print ("Not eligible for social assistance")
    taxRate = 0.2
tax = (income * taxRate) - taxCredit
```

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:
  - $\underbrace{\text{Pick up your son}}_{\textbf{Condition I}}$ AND $\underbrace{\text{pick up your daughter}}_{\textbf{Condition II}}$ after school today.
- Logical AND can be specified more formally in the form of true table.

Truth table (AND)		
C1	**C2**	**C1 AND C2**
False	False	False
False	True	False
True	False	False
True	*True*	*True*

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
True	*True*	*True*	*True*

Evaluating Logical AND Expressions

- True **AND** True **AND** True
- False **AND** True **AND** True
- True **AND** True **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 $\underbrace{\text{online textbook}}_{\textbf{Condition I}}$ OR the $\underbrace{\text{paper textbook}}_{\textbf{Condition II}}$ available in the bookstore.
- Similar to AND, logical OR can be specified more formally in the form of true table.

Truth table

C1	**C2**	**C1 OR C2**
False	*False*	*False*
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**
False	*False*	*False*	*False*
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
- False **OR** False **OR** False **OR** False

Logical NOT

- The everyday usage of logical NOT negates (or reverses) a statement.
- Example:
  - $\underbrace{\text{I am finding this class quite stimulating and exciting}}_{\textbf{Statement (logical condition)}}\text{.....}\underbrace{\textit{NOT!!!}}_{\textbf{Negation of the statement/condition}}$

- The truth table for logical NOT is quite simple:

Truth table	
S	**Not S**
False	True
True	False

James Tam

Evaluating More Complex Logical Expressions

- True **OR** True **AND** True
- **NOT** (False **OR** True) **OR** True
- (False **AND** False) **OR** (False **AND** True)
- False **OR** (False **OR** True) **AND** False
- **NOT NOT NOT NOT** True
- **NOT NOT NOT NOT** False
- **NOT NOT NOT** False

Extra Practice

- (From "Starting out with Python (2nd Edition)" by Tony Gaddis)

  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 <= c and a > 3	
1 != b and c != 3	
a >-1 or a <= b	
not (a > 2)	

Logic Can Be Used In Conjunction With Branching

- Typically the logical operators AND, OR are used with multiple conditions/Boolean expression:
  - 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.

Decision-Making With Multiple Expressions

- **Format:**

```
if (Boolean expression) logical operator (Boolean expression):
   body
```

- **Example:**

```
if (x > 0) and (y > 0):
    print ("All numbers positive")
```

Forming Compound Boolean Expressions With The "OR" Operator

- **Format:**

```
if (Boolean expression) or (Boolean expression):
    body
```

- **Name of the online example: "if_hiring.py"**

```
gpa = float(input("Grade point (0-4.0): "))
yearsJobExperience = int(input("Number of years of job experience: "))

if (gpa > 3.7) or (yearsJobExperience > 5):
    print("You are hired")
else:
    print("Insufficient qualifications")
```

Forming Compound Boolean Expressions With The "AND" Operator

- **Format:**

```
if (Boolean expression) and (Boolean expression):
    body
```

- **Name of the online example: "if_firing.py"**

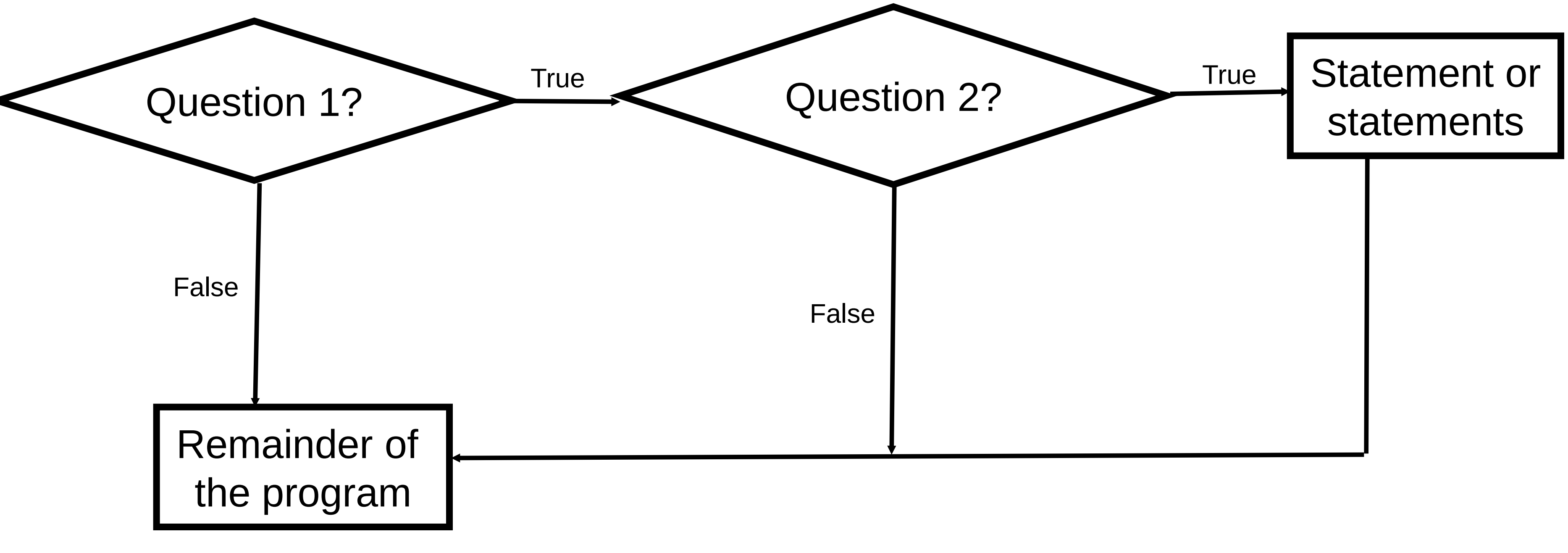
```
yearsOnJob = int(input("Number of years of job experience: "))
salary = int(input("Annual salary: "))

if (yearsOnJob <= 2) and (salary > 50000):
    print("You are fired")
else:
    print("You are retained")
```

Quick Summary: Using Multiple Expressions

- Use multiple expressions when multiple questions must be asked and the result of each expression may have an effect on the other expressions:
- AND:
  - 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:
  - 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.

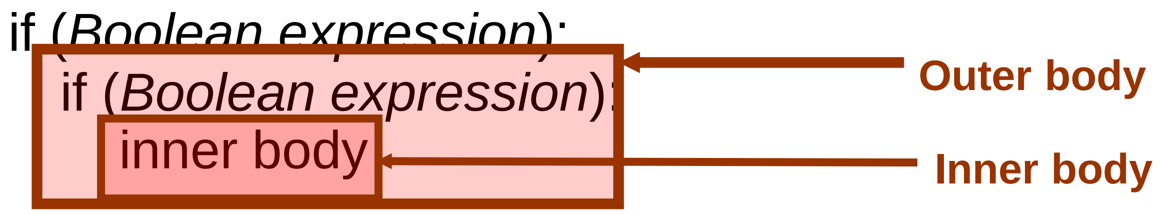
Nested Decision Making

- Decision making is dependent.
- The first decision must evaluate to true 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):
    if (Boolean expression):
        inner body
```

Nested Decision Making (2)

- **Example:**

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

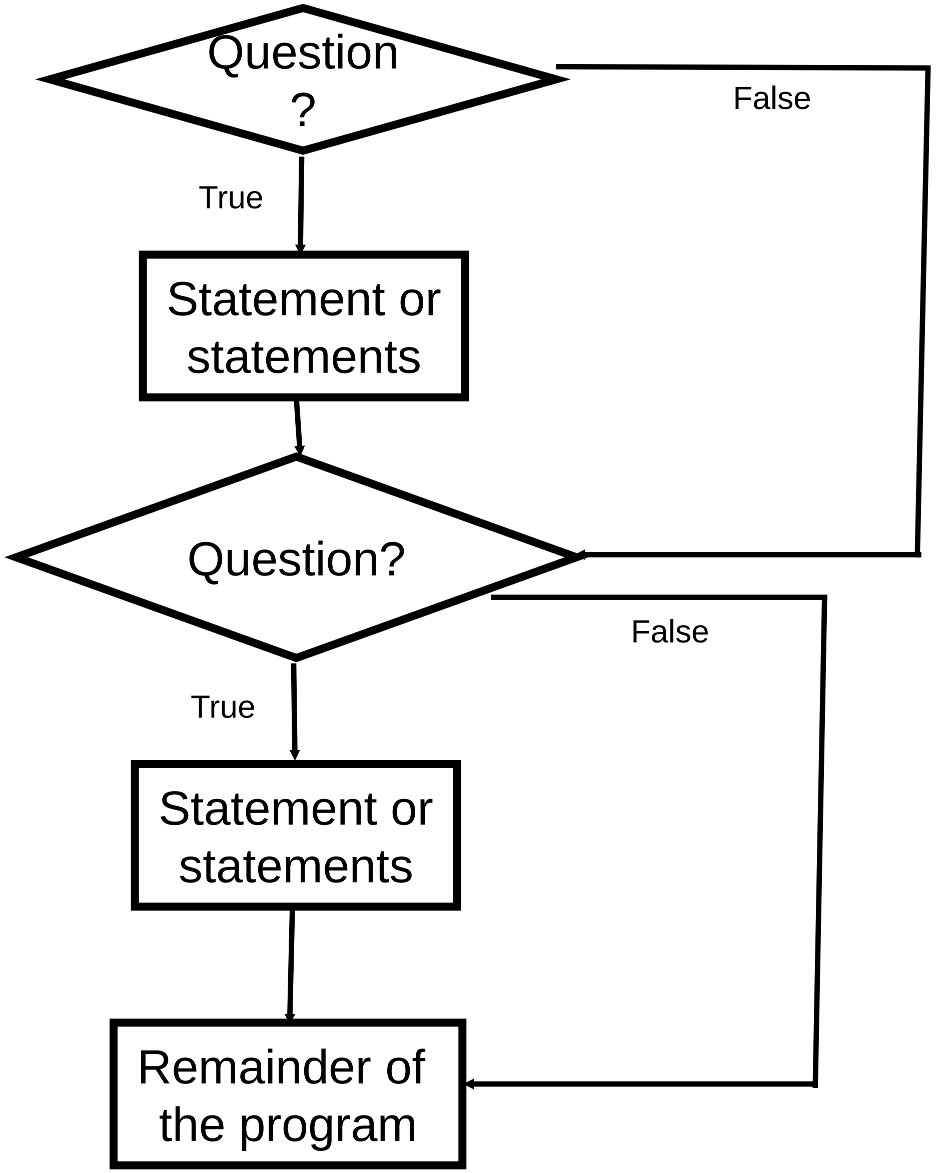
Question

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

Decision-Making With Multiple Alternatives/Question

- IF (single question)
  - Checks a condition and executes the body of code 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)
- **Format:**

```
if (Boolean expression 1):
    body 1
if (Boolean expression 2):
    body 2
        :
statements after the conditions
```

Multiple If's: Non-Exclusive Conditions (Example)

- **Example:**

```
if (num1 > 0):
    print (“num1 is positive”)
if (num2 > 0):
    print (“num2 is positive”)
if (num3 > 0):
    print (“num3 is positive”)
```

Multiple If's: Mutually Exclusive Conditions

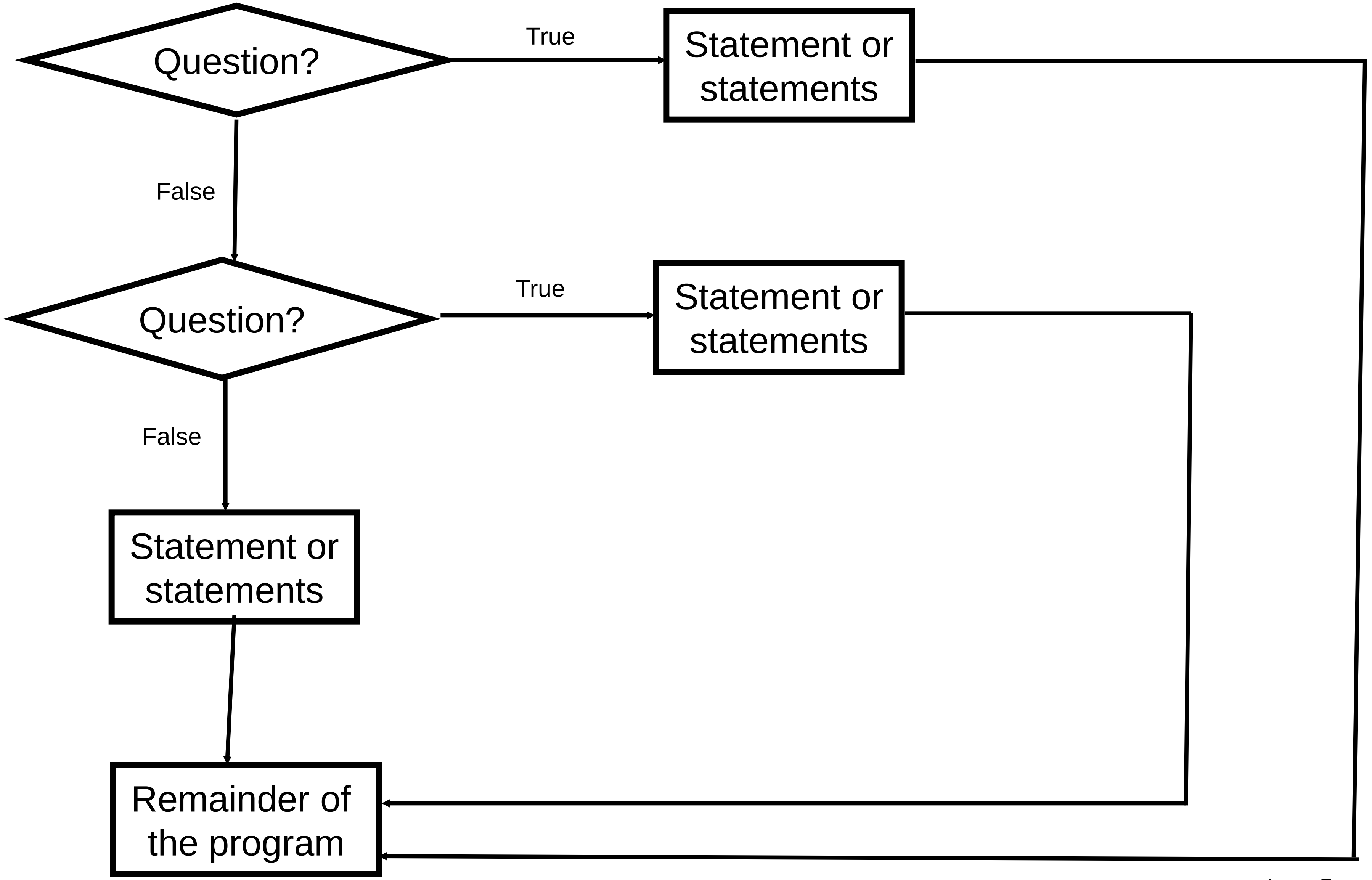
- At most *only one* of many conditions can be true
- Can be implemented through multiple if's

Inefficient combination!

- **Example**: The name of the complete online program is: "inefficient.py"

```
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: Mutually Exclusive Conditions

- **Format:**

```
if (Boolean expression 1):
    body 1
elif (Boolean expression 2):
    body 2
        :
else
     body n
statements after the conditions
```

Multiple If, Else-If's: Mutually Exclusive Conditions (Example)

- **Example**: The name of the complete online program is: “efficient.py”

```
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 '1'")
```

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

<u>Extra benefit:</u>

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

When To Use Multiple-If’s

- When all conditions must be checked (all the Boolean expressions for each ‘if’ must be evaluated).
- Example:
  - Survey questions:
    - When all the questions must be asked
    - The answers to previous questions will not affect the asking of later questions

Extra Practice

- (From “Starting out with Python (2nd Edition)” by Tony Gaddis).

Write a program that prompts the user to enter a number within the range of 1 through 10. The program should display the Roman numeral version of that number. If the number is outside the range of 1 through 10, the program should display an error message.

The table on the next slide shows the Roman numerals for the numbers 1 through 10.

Extra Practice (2)

Number	Roman Numeral
1	I
2	II
3	III
4	IV
5	V
6	VI
7	VII
8	VIII
9	IX
10	X

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

Construct	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) to execute when all the if-elif Boolean expressions are false.

Recap: When To Use Compound And Nested Decision Making Constructs (2)

Construct	**When To Use**
Compound decision making	More than one Boolean expression must be evaluated before the innermost body can execute. All expressions must evaluate to true (AND) or at least one expression must evaluate to true (OR).
Nested decision making	The outer Boolean expression must be true before the inner expression will even be evaluated. (Inner Boolean expression is part of the body of the outer Boolean expression).

Testing Decision Making Constructs

- Make sure that the body of each decision making construct executes when it should.
- Test:
  1) Obvious true cases
  2) Obvious false cases
  3) Boundary cases

Testing Decisions: An Example

```
num = int(input(“Type in a value for num: "))
if (num >= 0):
    print ("Num is non-negative.“)
else:
    print ("Num is negative.“)
```

Lesson: Avoid Using A Float When An Integer Will Do

Program name: real_test.py

```
num = 1.0 - 0.55
if (num == 0.45):
    print ("Forty five")
else:
    print ("Not forty five")
```

Problem Solving: Branches

- Write a program that converts percentages to one of the following letter grades: A (90 – 100%), B (80 – 89%), C (70 – 79%), D (60 – 69%), F (0 – 59%).
- The percentage score should come from the user.
- After determining the letter grade, the original percentage and it’s corresponding letter should be displayed.
- The program should display an error message for percentages outside of the above ranges.

Outline Of Solution

- Get the percentage score.
- Determine the letter grade
- Display the result

Developing A Solution: Start With The Easier Parts

```
percentage = 0.0
letter = ' '
percentage = float(input ("Enter the percentage score: "))

# Determine letter grade: don’t look at the solution until you’ve tried to
# come up with a solution yourself.

print ("Percentage: ", percentage, "%\t Letter: ", letter)
```

Determining The Correct Ranges

- Before directly implementing a solution (i.e., writing Python code) make sure that you have a clear idea of what's entailed.
  - This WHAT the problem must solve (e.g., balance a chequing account) NOT HOW it will be solved.
- The solution indicates how the problem will be solved.
- Depending upon the complexity of the problem the process of specifying the problem and solution may be formal (e.g., drawing diagrams, writing text descriptions, using detailed and specific notations etc.) or informal (e.g., going over the solution in your head).
- Also if your solution is not working (contains errors) then return back to the specifying the problem but do it more formally and in a more detailed form.

Determining Ranges: A Solution (Don’t Look Until You’ve Tried It Yourself)

```
if (percentage <= 100) or (percentage >= 90):
    letter = 'A'
elif (percentage <= 89) or (percentage >= 80):
    letter = 'B'
elif (percentage <= 79) or (percentage >= 70):
    letter = 'C'
elif (percentage <= 69) or (percentage >= 60):
    letter = 'D'
elif (percentage <= 59) or (percentage >= 0):
    letter = 'F'
else:
    print("Percent score is outside the allowable range (0 - 100%)“)
    letter = 'Error'
```

Determining Ranges: A Solution (Don’t Look Until You’ve Tried It Yourself)

```
if (percentage <= 100) and (percentage >= 90):
    letter = 'A'
elif (percentage <= 89) and (percentage >= 80):
    letter = 'B'
elif (percentage <= 79) and (percentage >= 70):
    letter = 'C'
elif (percentage <= 69) and (percentage >= 60):
    letter = 'D'
elif (percentage <= 59) and (percentage >= 0):
    letter = 'F'
else:
    print ("Percent score is outside the allowable range (0 - 100%)“)
    letter = 'Error'
```

Extra Practice

- (From "Starting out with Python (2nd Edition)" by Tony Gaddis)

The following code contains several nested if-else statements. Unfortunately it was written without proper alignment and indentation. Rewrite the code and use the proper conventions of alignment and indentation.

Extra Practice (2)

```
If (score >= A_SCORE):
print("Your grade is A")
else:
If (score >= B_SCORE):
print("Your grade is B")
else:
If (score >= C_SCORE):
print("Your grade is C")
else:
If (score >= D_SCORE):
print("Your grade is D")
else:
print("Your grade is F")
```

Common student question: If there isn't a pre-created solution then how do I know if I "got this right"?

Rule Of Thumb: Branches

- Be careful that your earlier cases don’t include the later cases if each case is supposed to be handled separately and exclusively.

Example 1

```
if (num >= 0):
elif (num >= 10):
elif (num >= 100):
```

Example 2

```
if (num >= 100):
elif (num >= 10):
elif (num >= 0):
```

Decision Making: Checking Matches

- Python provides a quick way of checking for matches within a set.
  - E.g., for a menu driven program the user's response is one of the values in the set of valid responses.

Format:

(Strings)

```
if <string variable> in "<string_1> <string_2>...<string_n>":
    body
```

(Numeric)

```
if <numeric variable> in (<number_1>, <number_2>,...<number_n>):
    body
```

Decision Making: Checking Matches (2)

Example:

(String):

```
if answer in (“userName1 userName2 userName3“):
    print (“User name already taken“)
else:
    print (“User name is allowed“)
```

(Numeric):

```
if num in (1, 2, 3):
    print ("in set“)
```

After This Section You Should Now Know

- What are the three decision making constructs available in Python:
  - If
  - If-else
  - If-elif-else
  - How does each one work
  - When should each one be used
- Three logical operations:
  - AND
  - OR
  - NOT
- How to evaluate and use decision making constructs:
  - Tracing the execution of simple decision making constructs
  - How to evaluate nested and compound decision making constructs and when to use them

You Should Now Know (2)

- How the bodies of the decision making construct are defined:
  - What is the body of decision making construct
  - What is the difference between decision making constructs with simple bodies and those with compound bodies
- What is an operand
- What is a relational operator
- What is a Boolean expression
- How multiple expressions are evaluated and how the different logical operators work
- How to test decision making constructs