Algorithms, Pseudocodes and Flowcharts

DA2303 – Python Programming

Computational thinking

- As a programmer, you focuses on using computers to create solutions to problems.

- Computational thinking is the thought process for creating solutions that can be carried out by a computer.

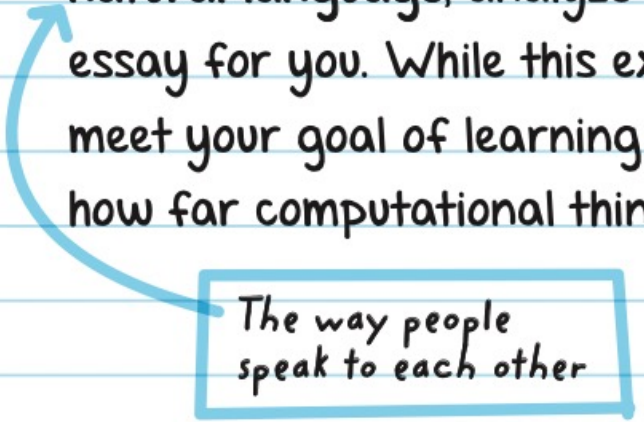
Computational thinking (cont.)

FOR EXAMPLE, if your problem is that you need to write a book report, a common solution is to read the book, plan out the essay, then write about what you read.

A way you might solve the same problem using computational thinking might be to collect data on the number of times each character talks to another and plot the information on a graph. You might even develop a program that analyzes the data for you. The graph would give you data-driven evidence of the relationships between the characters in the story. The output of your program and data could be used to tell you what to write in your essay.

Computational thinking (cont.)

An extreme example may be to program an artificial intelligence (AI) program that can understand **natural language**, analyze the book, and write the essay for you. While this example might not help you meet your goal of learning about the book, it shows how far computational thinking can go.



Major ideas in computational thinking

- There are four major ideas in computational thinking:
  - Decomposition
  - Pattern Recognition
  - Abstraction
  - Algorithm Design

Decomposition

- Before beginning work on a solution, you need to understand all the parts of the problem.
- Once you understand each part, you can break the problem into smaller tasks.
- Decomposition makes problems more manageable to solve.

Decomposition (cont.)

FOR EXAMPLE, when writing a book report, you would break the assignment down into smaller, simpler tasks:

1. Read the book.
2. Identify the main characters.
3. Identify the theme.
4. Analyze tone, plot, and character relationships.
5. Write about the tone, plot, and characters in the book.

Decomposition helps to identify where to start and the tasks that need to be completed.

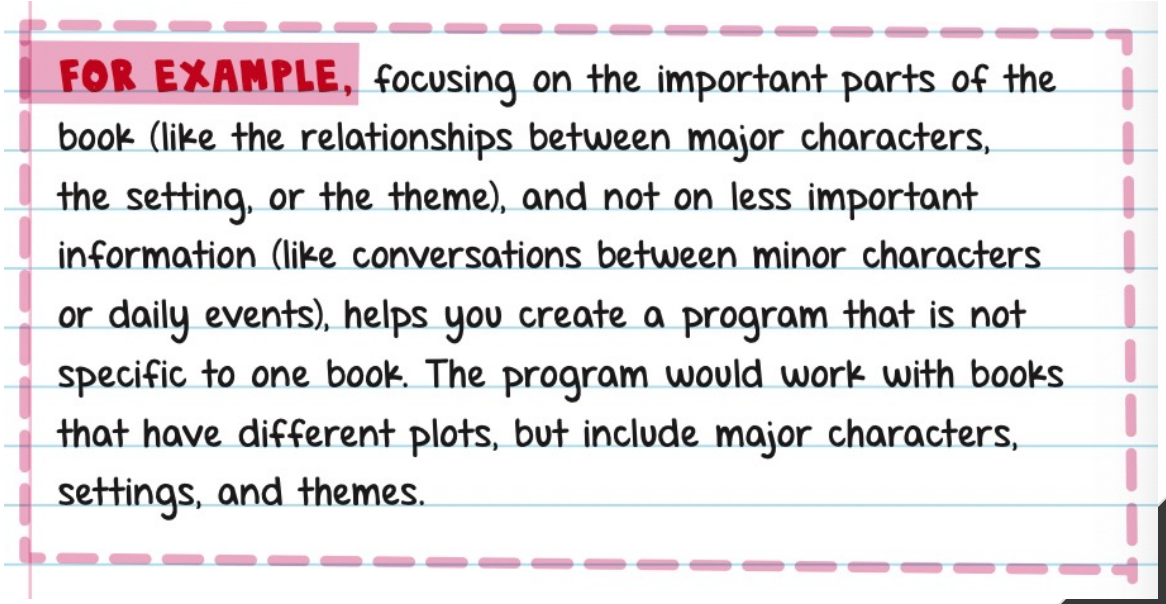
Pattern Recognition

- Patterns are events that are repeated.
- Recognising where you have created solutions to similar problems before will help you create solutions that can be used to complete different tasks.

FOR EXAMPLE, if you've used a format or a template for writing book report outlines in the past, you can include that same process in your program. A book report outline may be different, but the process of writing the report is the same.

Abstraction

- Abstraction is focusing on the important ideas of a problem and ignoring details that will not help you find a solution.



Algorithm design

- Algorithm design means writing out the steps you need to follow so that you can get the same solution every time.

- When a solution is carried out by an algorithm, the solution becomes reusable.

Algorithms

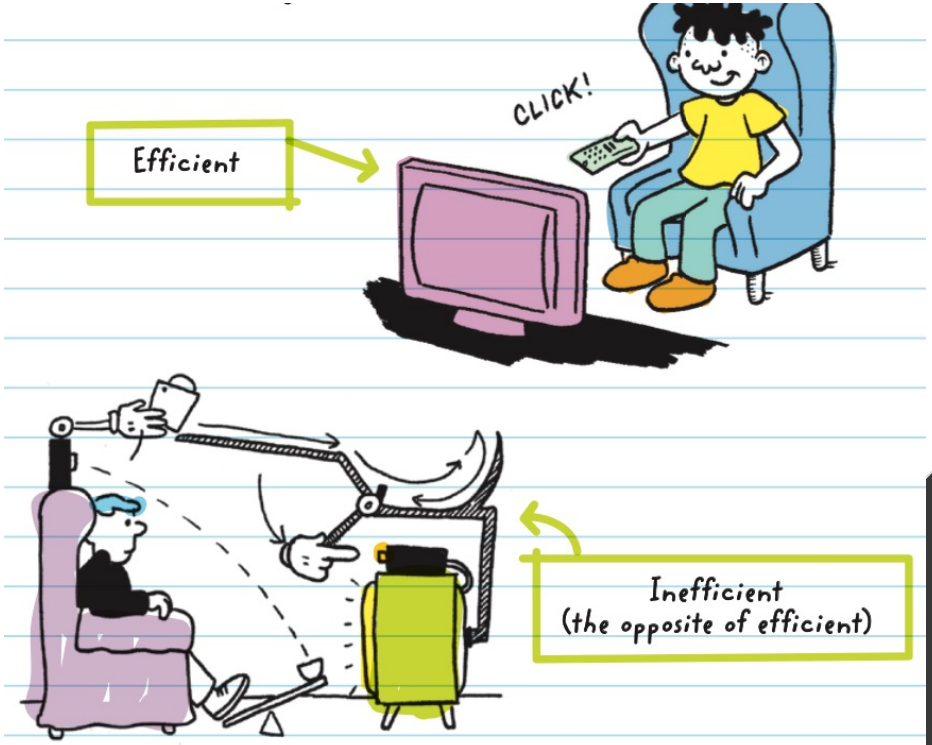
- Both humans and computers use algorithms.
- Algorithms are processes or steps that can be followed.
- A recipe is an example of an algorithm that both humans and some computers can use.
- Algorithms give clear instructions for repeating tasks.

Algorithms (cont.)

- Different algorithms could be used to complete the same task.
- For example, there are different ways to divide two numbers using long division or by doing repeated subtraction.
- Both ways give you the same result, but one way may be better.
- Typically, we are always looking for better algorithms.

Algorithms (cont.)

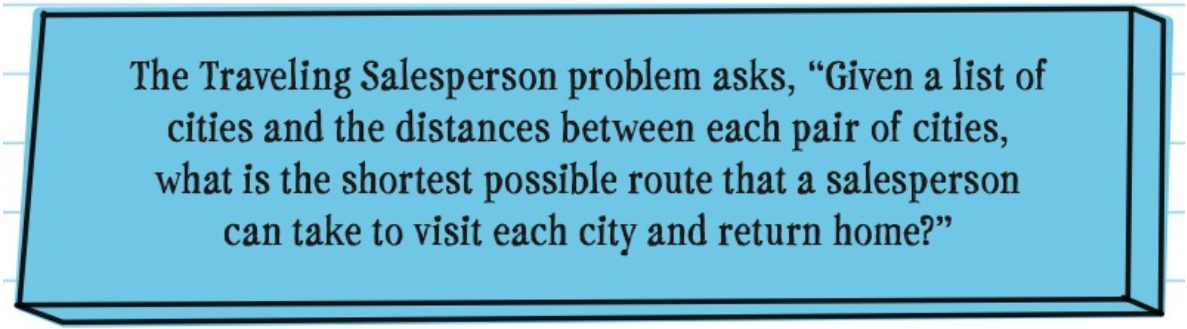
- Better algorithms are faster, simpler, or more efficient.
- Algorithms are useful because they can provide better ways to complete tasks.



Studying Algorithms

- Developing algorithms is its own branch of computer science and is very similar to the study of mathematics.

- For example, mathematicians and computer scientists have been studying the TRAVELING SALESPERSON PROBLEM for almost 100 years.



Studying Algorithms (cont.)

- The goal of the Traveling Salesperson problem is to develop the most efficient algorithm that can determine the shortest path that a traveling salesperson should take.
- The problem can be applied to real life.
- For example, a delivery truck dispatcher uses a shortest-route algorithm to determine all their deliveries for the day.
- The algorithm helps the trucking company save time and money.

Programs

- The job of a computer scientist is to come up with a solution to a problem, turn the solution into an algorithm, and then code the algorithm into a program so that it can be read by a computer.

- A program is an algorithm that has been translated (coded) into instructions for a computer.

- Algorithms are often written out in a way that most people can understand.

Programs (cont.)

- It’s less likely that most people can read a computer program and understand it, because it uses a combination of numbers, letters, and symbols in an arrangement we are not used to- but a computer would know just what it says.

- Programmers give very specific instructions to computers using PROGRAMMING LANGUAGES.

- Programming languages use combinations of numbers, words, symbols, and formatting to tell a computer what to do in a way that it can understand.

Programs (cont.)

FOR EXAMPLE: Suppose you wanted to find out if today is your friend's birthday. You could write this algorithm or program.

ALGORITHM	PROGRAM (PYTHON)
1. Look up today's date.	#Import datetime library from datetime import *

2. Ask a friend when their birthday is.	#Get Today's Date
	today = date.today()

3. Determine if today is your friend's birthday.	#Get User's Birthday
	dob_str = input("What is your Date of Birth? dd/mm/yyyy")

Programs (cont.)

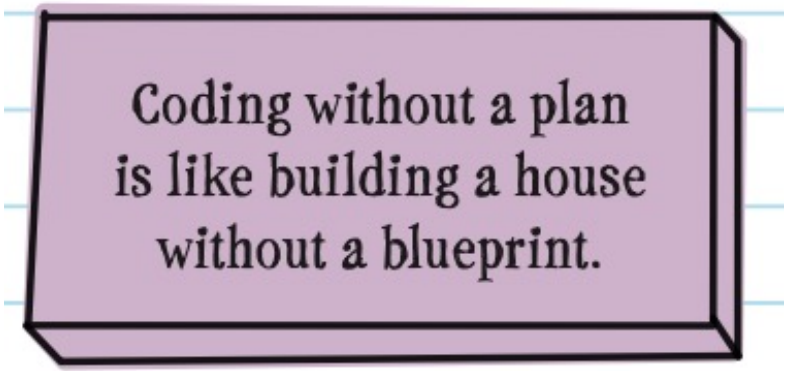
ALGORITHM	PROGRAM (PYTHON)

4. If today is your friend's birthday, say "Happy Birthday!" Otherwise, tell them, "Today is not your birthday."	#Convert user input into a date
	dob_data = dob_str.split("/")
	dobDay = int(dob_data[0])
	dobMonth = int(dob_data[1])
	dobYear = int(dob_data[2])
	dob = date(dobYear,dobMonth,dobDay)
	#Determine if today is the user's birthday
	thisYear = today.year
	nextBirthday = date(thisYear,dobMonth,dobDay)
	
	if today == nextBirthday:
	  print("Happy Birthday!")
	else:
	  print("Today is not your birthday.")

Creating Algorithms

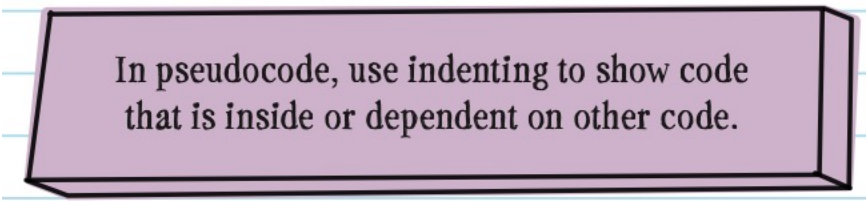
- Before typing out a program, programmers first come up with algorithm, or list of steps, that will be used in the code.

- They often use PSEUDOCODE and FLOWCHARTS to design or plan algorithms before they begin programming.



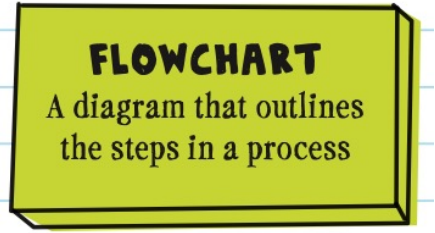
Pseudocode

- Pseudocode is written in a style or format that's similar to the programming language a programmer is going to use.
- There are no rules on how to write pseudocode, but it is meant to be read by humans, not a computer.
- Most programmers format their pseudocode so that each line represents a line of real code in their final program.
- The pseudocode doesn't have to be perfect. It should just give an idea of what the final program may look like.



Flowcharts

- Flowcharts help programmers visualise the steps in an algorithm.
- Just like making an outline before writing an essay, flowcharts help organise ideas, and they use specific symbols to represent different parts of an algorithm.



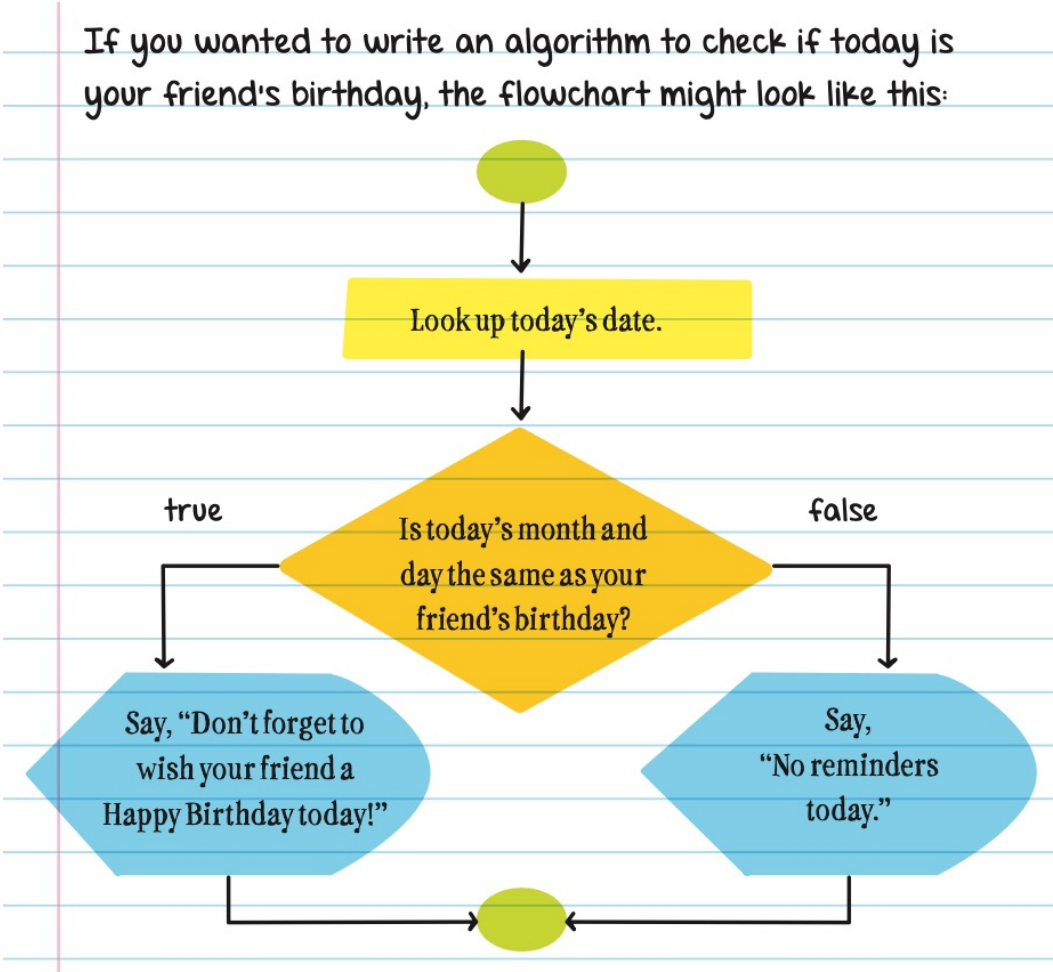
Each symbol in a flowchart has a meaning.

Flowcharts (cont.)

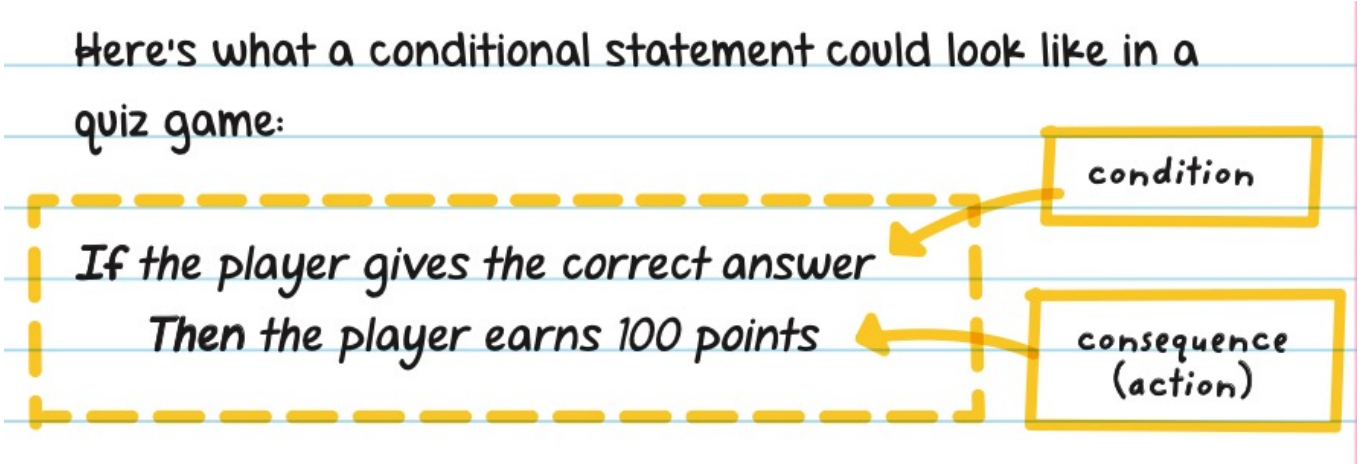
NAME	SYMBOL	MEANING IN FLOWCHART
Oval		The beginning or end of the program
Parallelogram		An input operation
Rectangle		A process to be carried out (addition, subtraction, division, etc.)
Diamond		A decision (or branch) to be made; the program should continue along one of the two routes
Hybrid		An output operation
Flow line		The direction of flow in the program

Flowcharts (cont.)

- It might look like this in pseudocode:
    - Look up today's date.
    - If the month and day of your friend's birthdate are the same as today's date:
        - Then say "Don't forget to wish your friend a Happy Birthday today!"
        - Else, say "No reminders today."

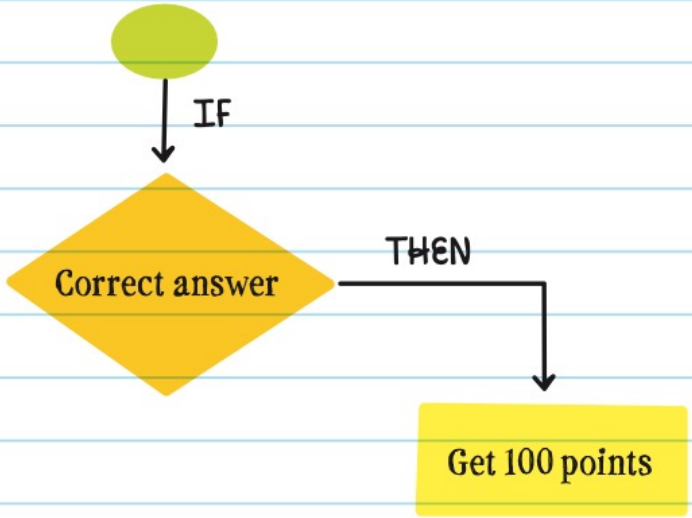


Conditional Statements



Conditional Statements (cont.)

An If . . . Then statement can be shown with a flowchart. For example, in the quiz game where a correct answer earns the player 100 points, a basic conditional statement might look like this:



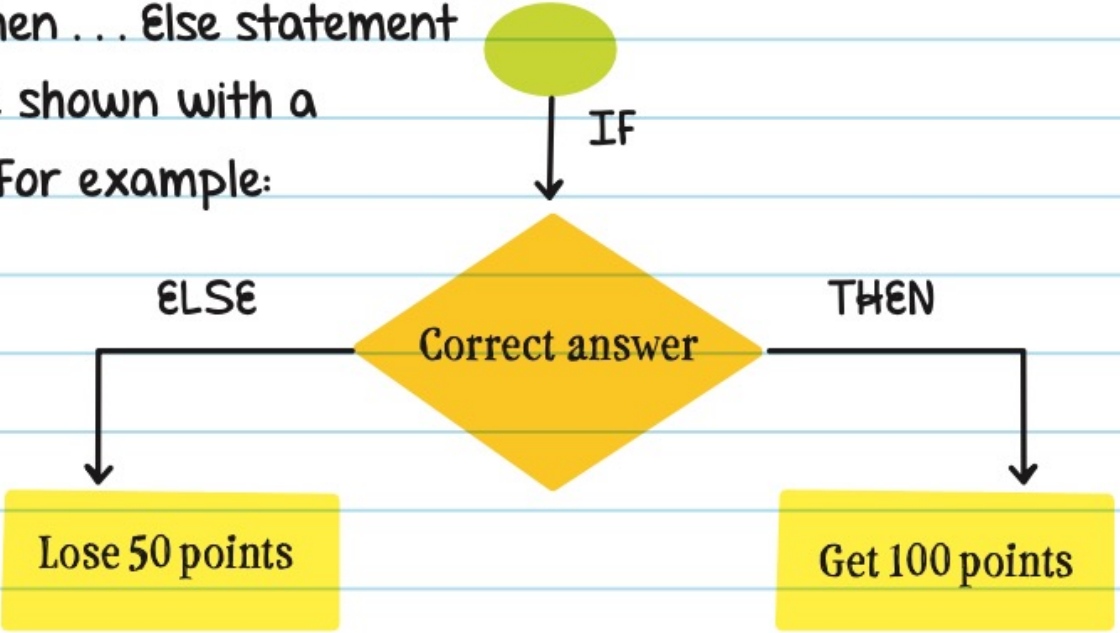
Conditional Statements (cont.)

Here's what a conditional statement with an "else" consequence could look like in the quiz game:

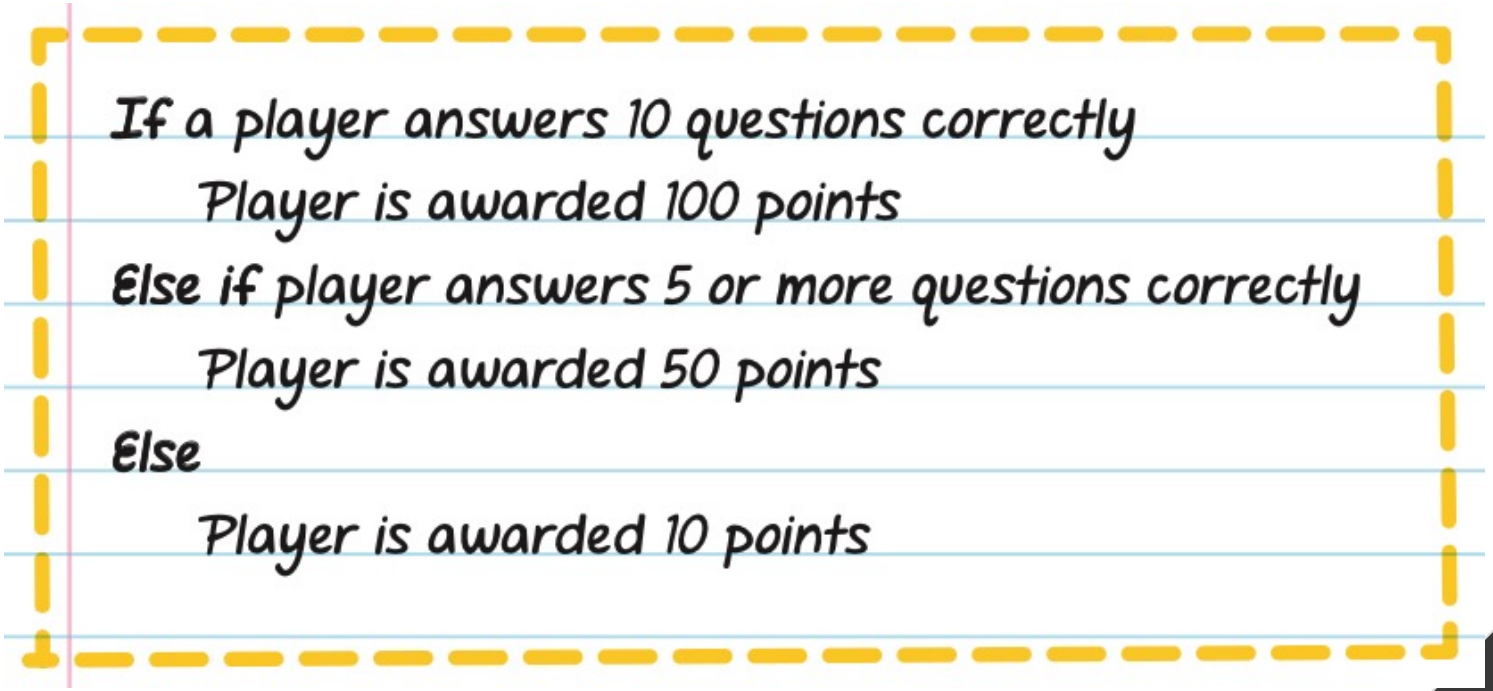
```
If the player gives the correct answer
    Then the player earns 100 points
Else
    The player loses 50 points
```

Conditional Statements (cont.)

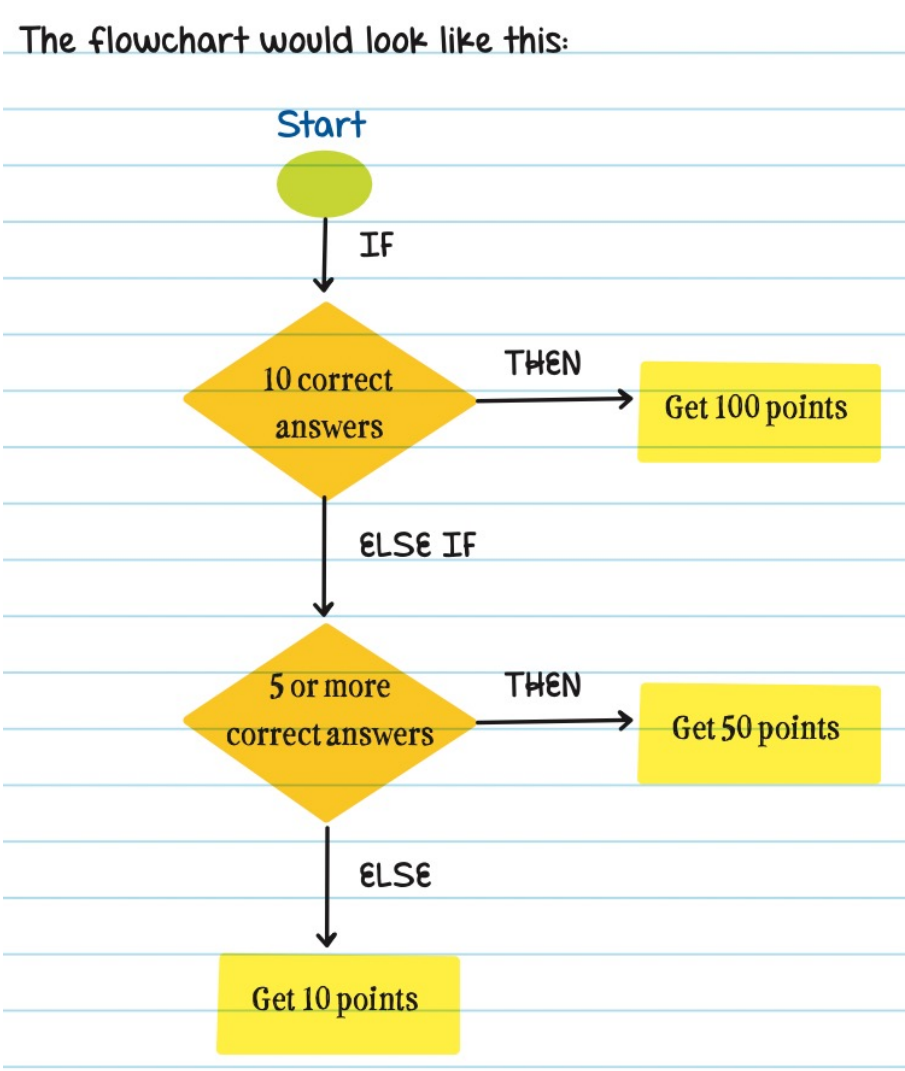
An If . . . Then . . . Else statement can also be shown with a flowchart. For example:



Conditional Statements (cont.)

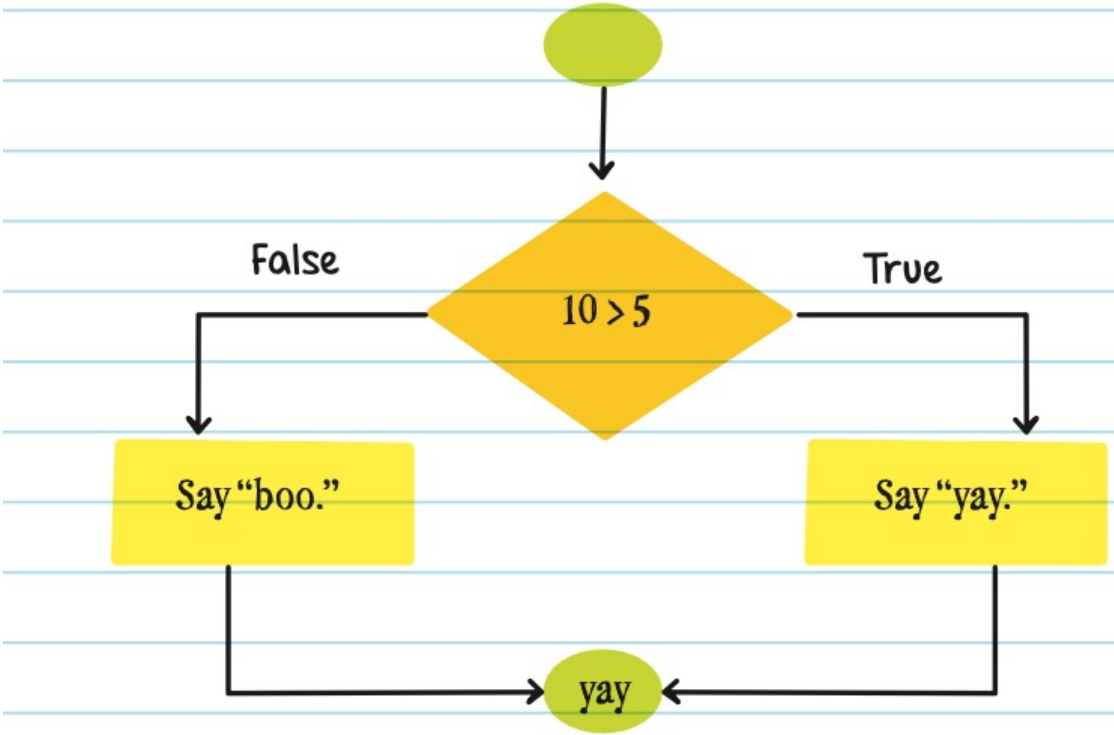


Conditional Statements (cont.)



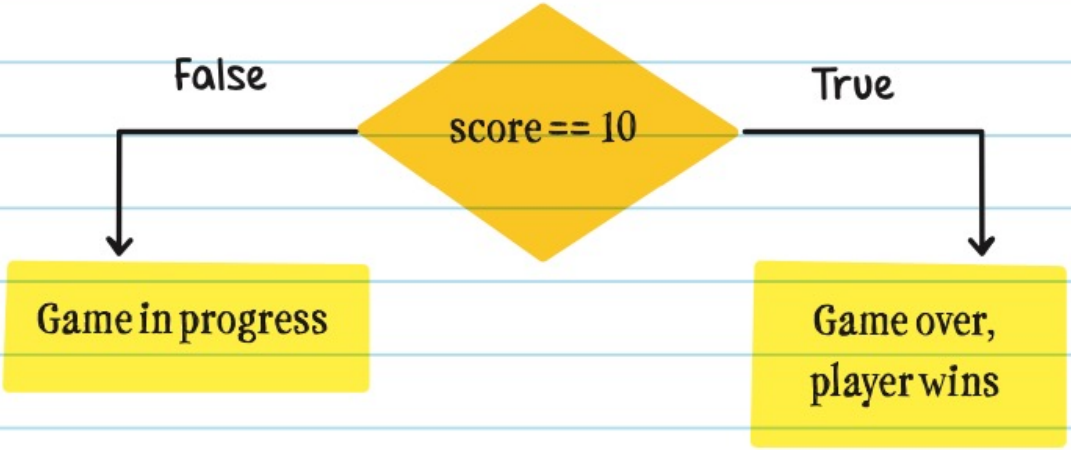
Comparison operators

Comparison operators can be used in flowcharts. This flowchart shows that if 10 is greater than 5, you'll say "yay," else you'll say "boo." You'll say "yay" because 10 > 5 is True.



Comparison operators (cont.)

You can use the == operator to compare two values. For example, you could see if a game is over or not based on the player reaching a score of 10 like this:



Comparison operators (cont.)

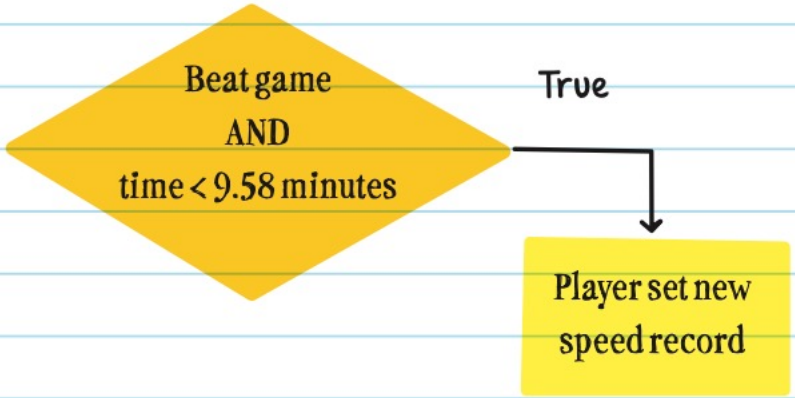
You could also use a Boolean variable by itself.

For example, in a race game, you could have a variable called "boost." The value of the variable is set to True if the player earns a speed boost. The program uses a conditional statement to check if boost is true (the player has earned a boost). If so, then the program should increase the player's speed:

Compound conditionals

FOR EXAMPLE, a game player sets a new speed run record only if they beat the game **AND** finish in less time than the current record.

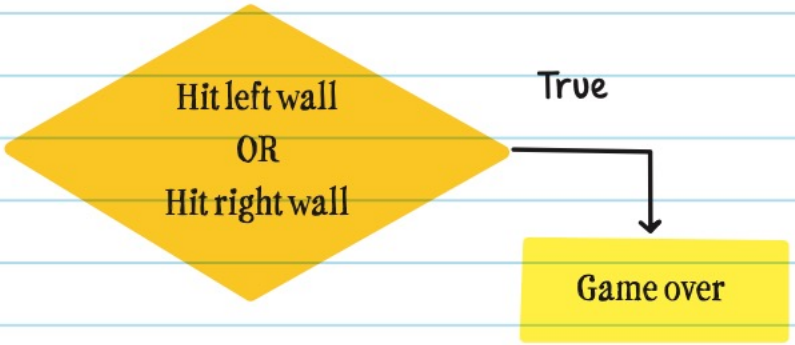
If the player just finishes the game but doesn't do it in less time, or doesn't finish the game at all (if both conditions are not met), then the player does not set a new record.



Compound conditionals (cont.)

FOR EXAMPLE, in a ball game, if the player hits the right wall OR if they hit the left wall, then they lose and the game ends.

If either condition of hitting either wall is met, then the consequence (losing) becomes true.

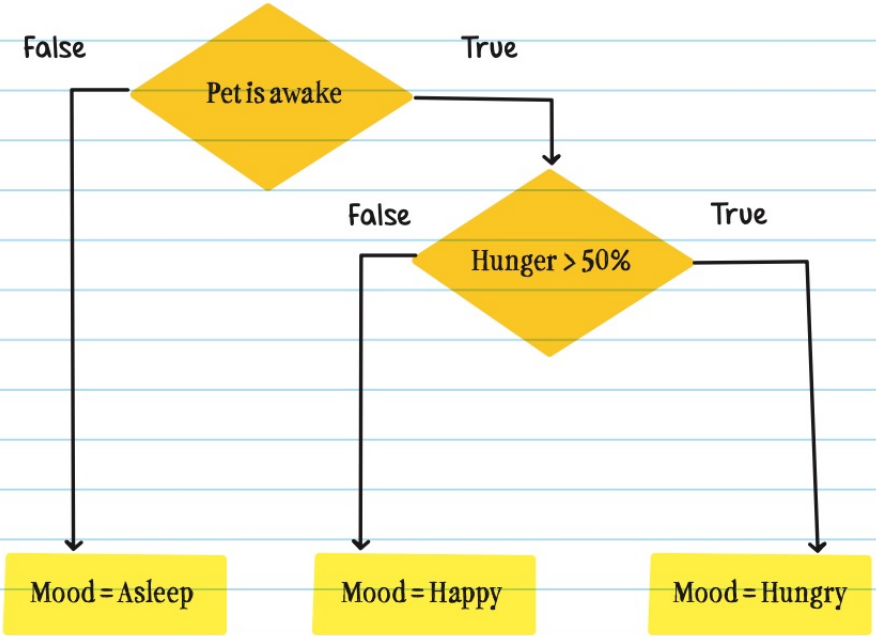


Nested conditionals

If the pet is awake, and if the pet's hunger is below 50 percent, then show a happy mood, else show a hungry mood.

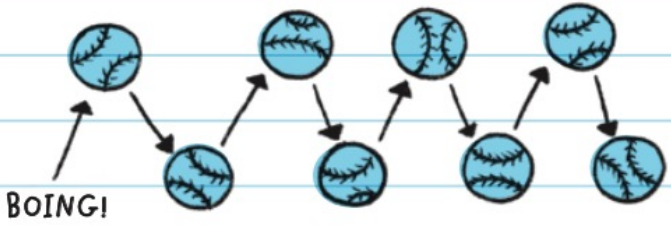
So, if the pet is awake and its hunger level is at 75 percent, then its mood will be hungry.

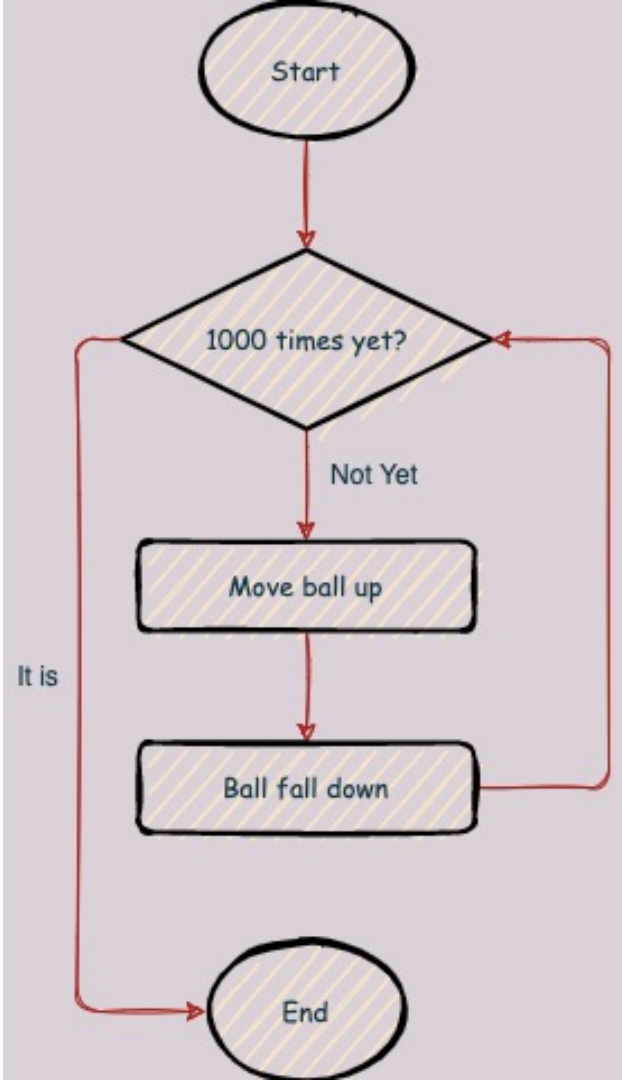
The flowchart looks like this:



Loops

Repeat 1,000 times
move up, then fall down





For Loops

FOR EXAMPLE, we could use a *for* loop to list all the top-ten players of a game whose names are stored in an array.

an array containing the names of the top-ten players

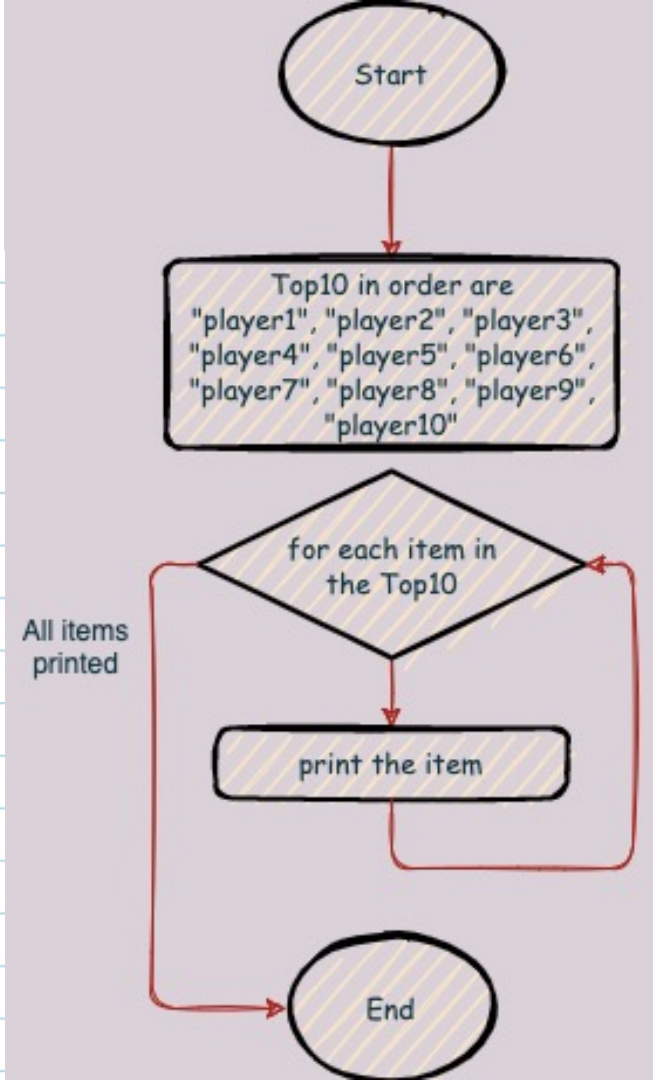
```
Top10 = ["player1", "player2", "player3", "player4", "player5",
"player6", "player7", "player8", "player9", "player10"]
```

begin the for loop, repeat the code in the loop for each item in the Top10 list

```
for item in Top10:
```

the code that's repeated – print out each name in the Top10 list

```
    print(item)
```

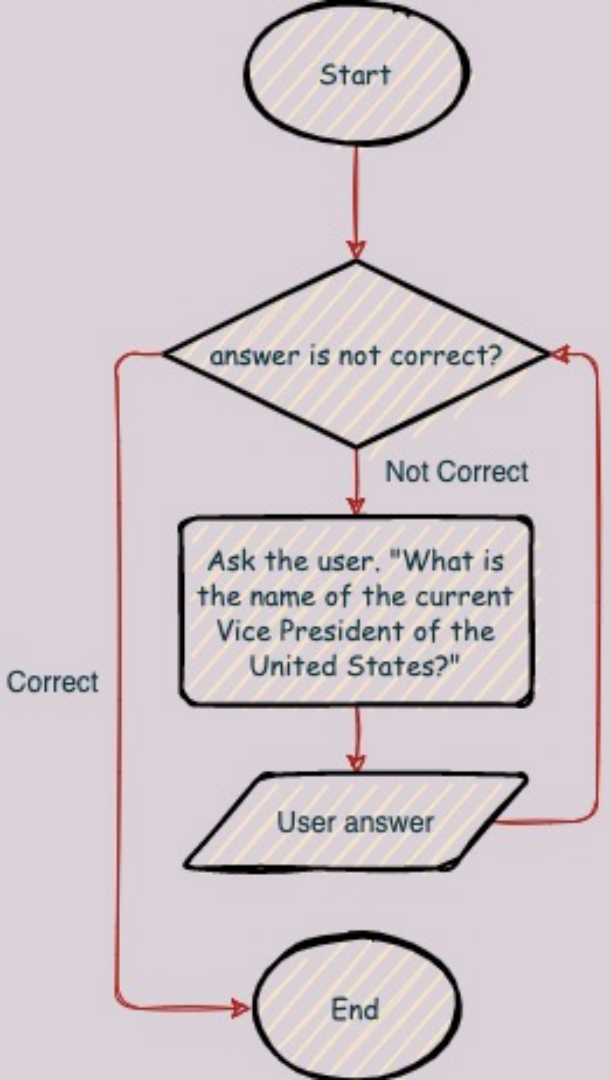


While Loops

FOR EXAMPLE, in a quiz game you can repeatedly ask a question until the user gets the correct answer:

```
while answer is not correct:
    Ask the user, "What is the name of the current
    Vice President of the United States?"
print "you are right!"
```

The quiz game would continue to ask the question until the user got the right answer, even if it took 100 attempts!



Nested Loops

FOR EXAMPLE, you could program background music for a game using nested loops.

A nested loop can repeat a beat until the game is over. An inner loop can repeat a bass drum sound 7 times before a snare sound is played:

OUTER LOOP

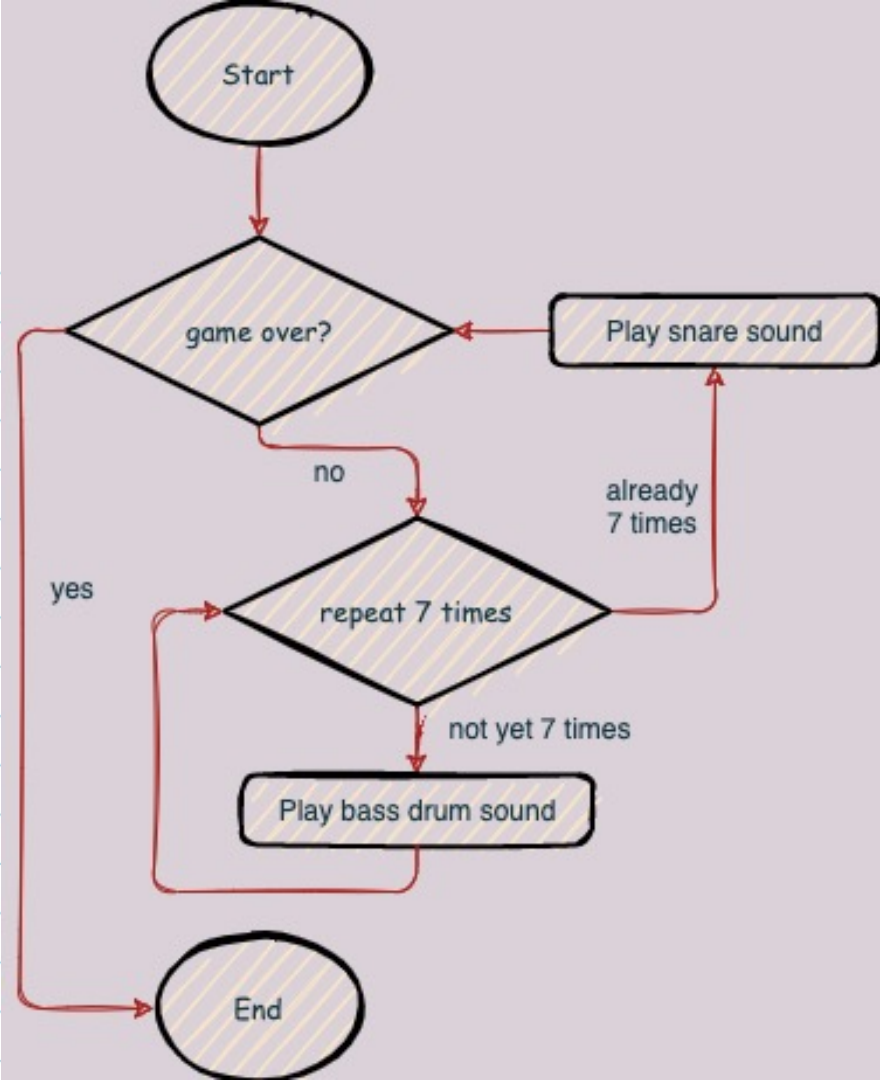
Repeat until the game is over:

INNER LOOP

Repeat 7 times:

Play bass drum sound

Play snare sound



Thanks

Do you have any question?

jailani.rahman@pb.edu.bn
(+673) 223 4466 ext 241
Unit 6.01, Ong Sum Ping Campus,
Politeknik Brunei.