ICSP

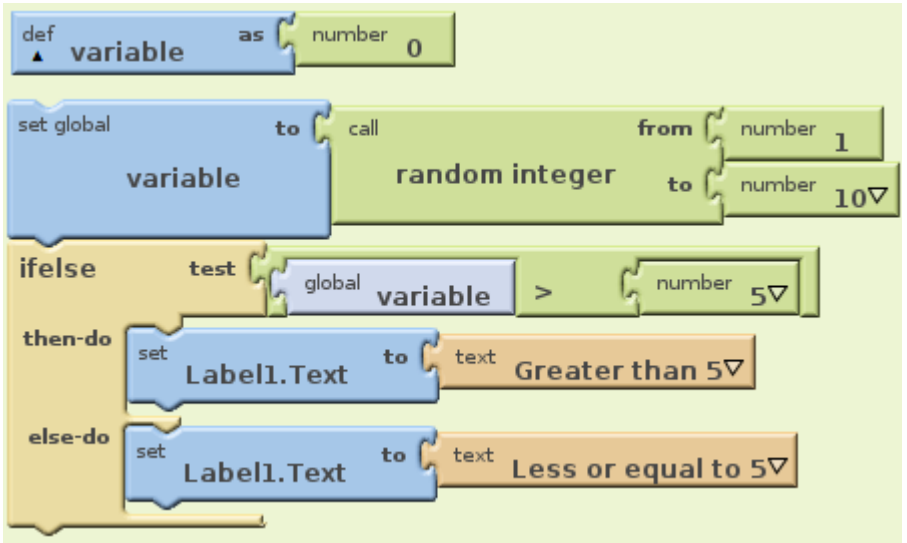
AppInventor
Lecture 2

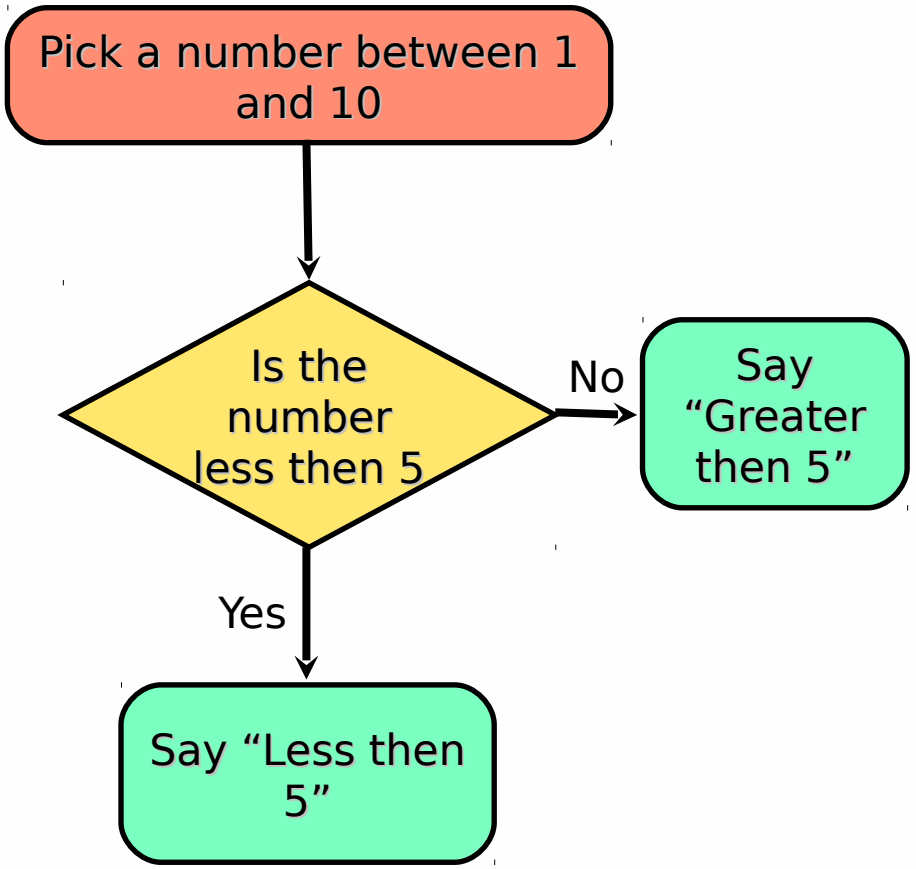
Outline

- Conditional Statements
- Boolean Logic
- Loops
- Concurrency
- Variable Scope

Conditional statements

- Control flow is the execution order of statements
- Execution of conditional statements decides which of two or more control flows that will be followed





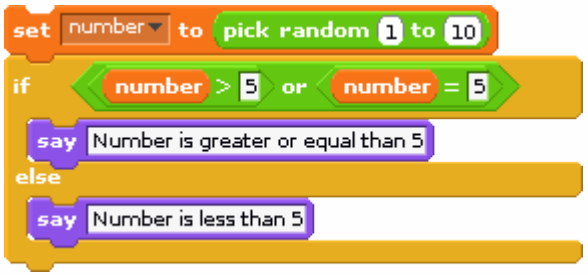
Flow chart representing the Scratch statement

Boolean logic

- George Boole (November 2, 1815 – December 8, 1864) English mathematician and philosopher. First professor of mathematics of then Queen's College, Cork (now University College Cork). Developed boolean algebra in 1854.
- Deals with the values 0 and 1. Can be considered two integers, or as the truth values false and true
- In software programming many conditions are represented using boolean logic

Boolean operations OR

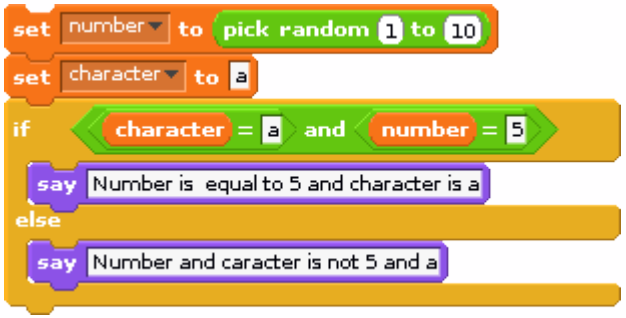
- OR (disjunction) the truth table to the right shows the value of the operation



X	Y	X OR Y
False	False	False
True	False	True
False	True	True
True	True	True

Boolean operations AND

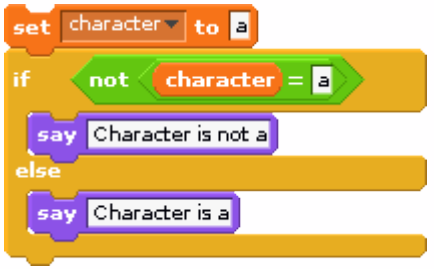
- AND (conjunction) the truth table to the right shows the value of the operation



X	Y	X AND Y
False	False	False
True	False	False
False	True	False
True	True	True

Boolean operations NOT

- NOT (negation) the truth table to the right shows the value of the operation



X	NOT X
False	True
True	False

Joining Boolean operators

- At most two terms are joined by a boolean operator. Further sets can be joined using additional operators.
  - Example: a AND b AND c
- Any number of ANDs or ORs can be joined without ambiguity. If AND and OR are combined parentheses can be used to clarify the order of operations. The operations in the innermost pair are performed first etc.
  - Example: a AND (b OR (c AND d))

Properties of boolean logic

A Boolean Algebra can formally be defined as a set S of elements [a,b,...] where 0 is False, 1 is True and:

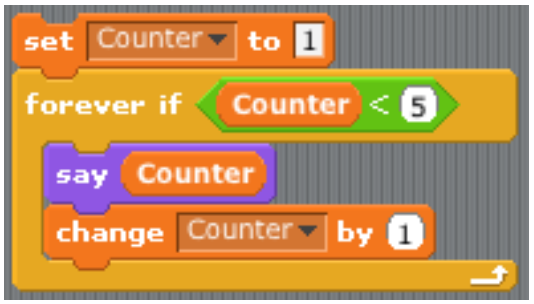
- a OR (b OR c) = (a OR b) OR c; a AND (b AND c) = (a AND b) AND c (*associativity*)
- a OR b = b OR a; a AND b = b AND a (*commutativity*)
- a OR (a AND b) = a; a AND (a OR b) = a (*absorption*)
- a OR (b AND c) = (a OR b) AND (a OR c); a AND (b OR c) = (a AND b) OR (a AND c) (*distributivity*)
- a OR NOT a = 1; a AND NOT a = 0 (*complements*)

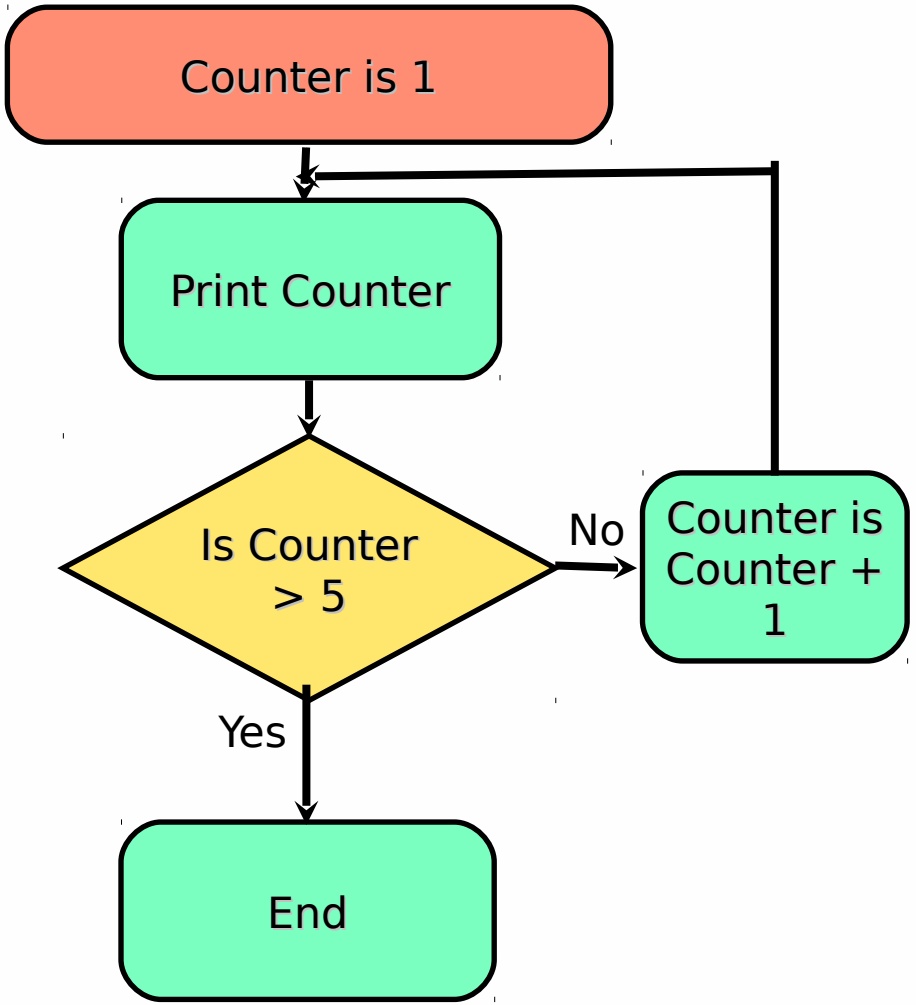
More boolean logic properties

- a OR a = a AND a = a (*idempotency*)
- a OR 0 = a; a AND 1 = a (*boundedness*)
- a OR 1 = 1; a AND 0 = 0
- NOT 0 = 1; NOT 1 = 0 (0 and 1 are complements)
- NOT (a OR b) = NOT a AND NOT b; NOT (a AND b) = NOT a OR NOT b (*de Morgan's laws*)
- NOT NOT a = a (*involution*)

Loops

- Iterating, looping or repeating all refer to a sequence of statements that will be carried out several times in succession
- Loops are useful when there are parts of the code that will be repeated which can be both tedious and error prone. (Although too compact code can sometimes be quite difficult to read)





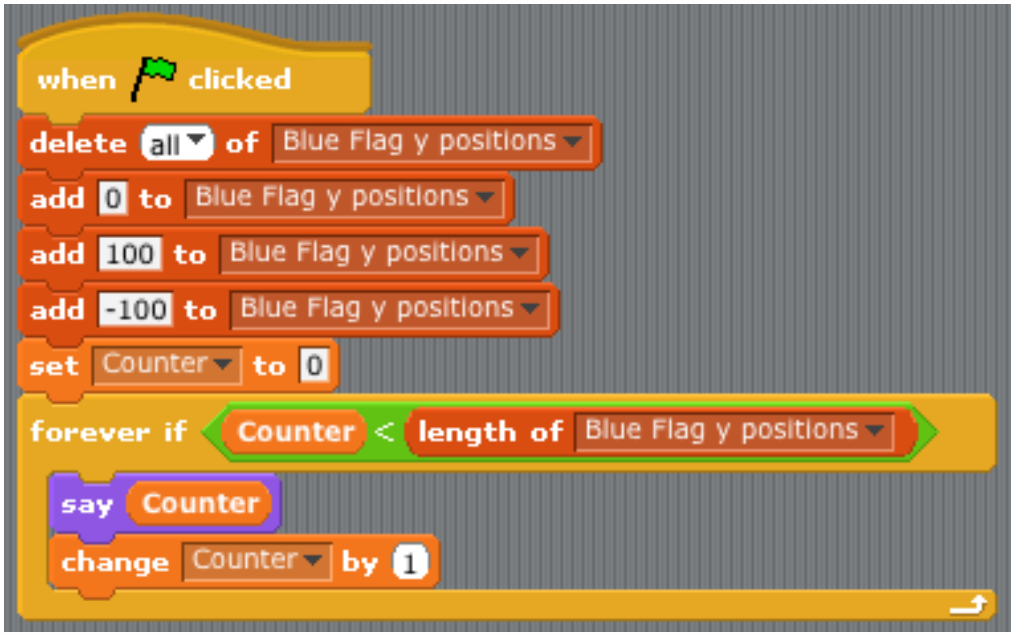
Flow chart representing the Scratch statement to the left. It says 12345

Repeating iterations

- Chose the loop that best suites your problem
  - Counting loops
  - Conditional loops
- Make sure you know how the loop ends. The loop will run forever if nothing tells it to stop.
- Loops in Scratch
  - Forever (Conditional loop)
    - Forever if
  - Repeat (Counting loop)
    - Repeat until

Looping over lists

- Create a list
- Loop over the elements in the list (Repeat “list length”)
- Lists are convenient for looping



Scratch script says: 012

Variable scope

- Scope is the context of where variable may be used.
  - E.g in Scratch a variable can be used by all Sprites or by only the Sprite assigned to the variable
- Limiting the scope of a variable can help to avoid unexpected behavior.
  - E.g. in Scratch a Script might changes a variable that affects another Script unexpectedly
- It is often good practice to limit the scope of variables to only include what is necessary
  - E.g. If Sprite1 is using a variable to count the number of elements in a list then this variable could be restricted to Sprite1 in order to avoid conflict with Sprite2 that also uses a variable to keep track of the number of elements in a different list

Routing and Deadlock in Networks

- Adapted from Computer Science Unplugged
- Aim is for each person to end up holding the papers labelled with their own letter.
  - Played at each table
- There are two papers with each participants's letter on them, except for one, who only has one paper to ensure that there is always an empty hand.
- Distribute the papers randomly at the table. Each participant has two papers, except for one who has only one. (No one should have a paper with their letter on it.)
- Pass the papers around until each participant gets the papers labelled with their name. You must follow two rules:
  - a) Only one paper may be held in a hand.
  - b) A paper can only be passed to an empty hand of an immediate neighbour at the table.

Serial and Parallel execution

- Instructions can be executed in
  - Serial, each instruction in one turn after another
    - E.g in Scratch when any sequence of blocks executes
  - Parallel, perform in execution of instructions in parallel, i.e. It can be faster, but also more complicated to program
    - E.g. In Scratch when two “Green Flag scripts” exist for a sprite, they are executed in parallel.

Concurrency continued

- When resources of the computer are shared (E.g. Memory, Screen) problems can arise, one is Race Conditions which implies unpredictable behavior
  - A race condition is when the output of a process is dependent on the timing of other events
    - E.g. Signals racing to see who can first produce the output.
  - Mutual exclusion can prevent race conditions but can cause deadlock or livelock
  - Mutual exclusion might also increase the execution time of the program

Race condition example

- A bank account has 1000 Euro
- 2 transactions are executed simultaneously
  - Transfer 1: Insert 10 Euro
  - Transfer 2: Insert 100 Euro
  - Expected Balance 1110 Euro
  - The balance after the transfers will depend on how the parallel processing is implemented. Here are 2 scenarios

Unsynchronized transfers

- Transfer 1 reads balance 1000 Euro
- Transfer 2 reads balance 1000 Euro
- Transfer 1 increases balance by 10 to 1010
- Transfer 2 increases balance by 100 to 1100
- Transfer 1 writes 1010 to account
- Transfer 2 writes 1100 to account
- Your account balance is 1100

Synchronized transfers

- Transfer 1 locks account and reads balance 1000 Euro
- Transfer 2 tries to read account but is told to wait
- Transfer 1 increases balance by 10 to 1010
- Transfer 1 writes 1010 to account
- Transfer 1 releases lock and wakes Transfer 2
- Transfer 2 locks account and reads balance 1010 Euro
- Transfer 2 increases balance by 100 to 1110
- Transfer 2 writes 1110 to account and releases lock
- Your account balance is 1110