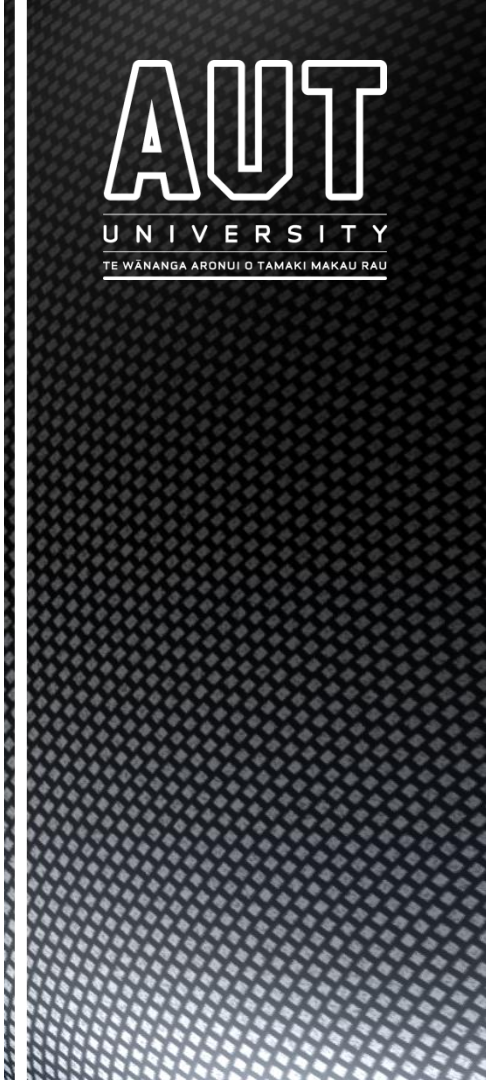
175003 Programming for Creativity

Lecture 04

What If? Conditionals and Loops



Recap Lecture 3

- Primitive Variable Types
  - `boolean`
  - `int` (`short`, `long`)
  - `float` (`double`)
  - `char`
  - `color`
- Operators
- Coding Conventions

Overview

- This session will cover:
  - More useful in-built Processing functions
  - Conditionals
  - Loops
    - While loops
    - For loops

Assignment 2

- Online now
- Deadline: Thursday, 10th of April, 8pm
- You will learn the necessary bits this week and the following week, mostly in the labs!

IN-BUILT PROCESSING COMMANDS

Processing Commands

- Processing has a wide range of in-built commands (actually they are *functions*…)
- E.g., Assignment 1: **`mousePressed()`**
  - This is something called an "event handler"
  - It allows you to write code that is performed when the event occurs, in this case when the mouse is pressed
- Processing has some very useful commands, functions and event handlers

Interaction (More Event Handlers!)

- **`void mouseReleased()`**
  - If you define this function, it will be called whenever the user releases the pressed mouse button
- **`void keyPressed()`**
  - If you define this function, it will be called whenever the user hits a key on the keyboard

Useful Functions

- **`println(`*`expression`*`)`**
  - Prints expression to the screen
  - **`println("Hello");`** // prints “Hello” (no quotes)
  - Look where Processing prints this – useful for debugging your programs, more on that in a few weeks

Randomness

- **`random(`*`max`*`)`**
  - Returns a random number between 0 (inclusively) and *max* (exclusively)
  - e.g., **`random(4)`** returns a number between 0 and 3
- **`random(`*`from, to`*`)`**
  - Returns a random number between *from* (inclusively) and *to* (exclusively)
  - e.g., **`random(4, 8)`** returns a number between 4 and 7

Setting Limits

- **`constrain(`*`value`*`, `*`min`*`, `*`max`*`)`**
  - Restricts the value of *value* to fall between *min* and *max*
  - constrain() returns the limited value
  - Basically the same as writing two if-statements
- **`frameRate(`*`rate`*`)`**
  - Controls the rate at which draw() redraws the screen
  - The default frame rate is 60 frames per second

Don’t Forget…

- The Processing website contains a useful reference of in-built commands/functions
- It also defines what is called a “function prototype” for each function
  - This defines what the arguments of the function should be – more on arguments next week
  - Rather than guess what the values should be, go and look it up!!!
  - If you are not sure, write a comment in your code to remind you and/or use really obvious variable names as a reminder

Bad Example

- Undocumented and not very useful variable names

```
float a = 50;
size(500, 500, P3D);
noStroke();
lights();
translate(58, 78, 0);
sphere(a);
```

Good Example

- Well documented and useful variables

```
// Create a 500px square canvas using the 3D renderer
size(500, 500, P3D);

int radius = 50; // Radius of sphere
int x      = 58; // Amount to translate shape along the x-axis
int y      = 78; // Amount to translate shape along the y-axis
int z      =  0; // Amount to translate shape along the z-axis

noStroke(); // Turn off lines on primitive shapes
lights();   // Turn on lighting effects in 3D renderer

// Move object in 3D space according to values of x, y & z
translate(x, y, z);

// Draw a sphere of the given radius
sphere(radius);
```

Conditional Statements

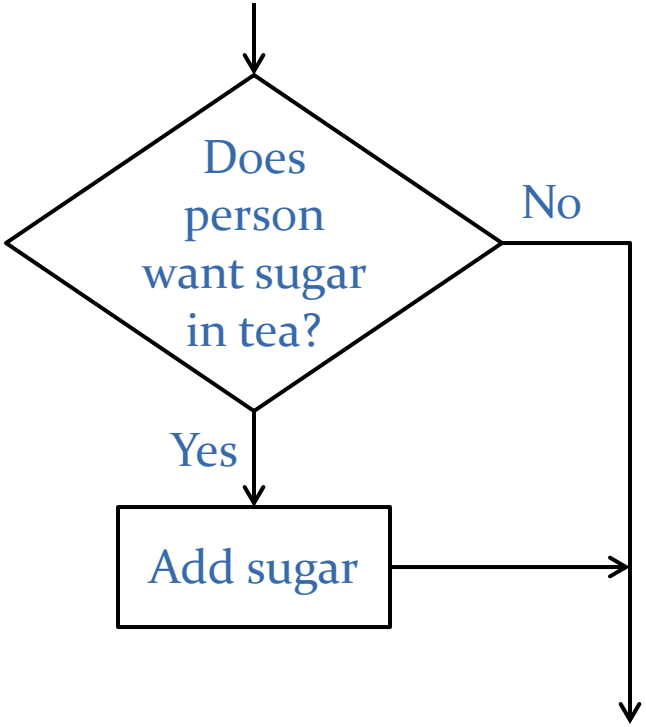
Conditional Statements

- Using conditional statements allows you to create multiple pathways (or “flows of control”) through the code you write
- This allows your program to behave differently when presented with different situations, e.g.
  - Choosing an action from a menu
  - Determining what sales tax rate to charge,
  - Testing for collision of an asteroid with the spaceship
  - etc.

Program Design

- As your programs get bigger and more complex, the more it becomes important to design rather than to just jump into code
- One of the simplest design techniques is simply drawing flow charts
  - Can be translated into “pseudo-code” before writing the program
  - Really easy to show conditional statements

Example




```
if ( wantsSugar == true )
{
   addSugar();
}
```

True and False

- Boolean variables were introduced last week
- A boolean expression is one whose value is either true or false (but not both!)

```
a > b
```

```
var1 == var2
```

- Boolean (logical) equality operator: ==
- Common programming error: using '=' (assignment) instead of "==" (equality)

```
if (x = 5)    // WRONG!
if (x == 5)   // Correct
```

Boolean Operations

- Processing makes decisions using boolean logic (i.e., values of true and false) and operations
- Two special keywords (**`true`** and **`false`**) are used for this purpose
- Relational operators are used to compare two values
- Logical operators combine smaller boolean expressions into a single, larger expression

Relational Operators

- What is the relation of one variable/value to another:

==	equality
>	greater than
>=	greater than or equal to
!=	inequality
<	less than
<=	less than or equal to

Calculating “Return” Values

- When you write something like…

  ```
  temp >= 98.6
  ```

- …during execution of your program,
  Processing actually performs the check
  between the operands and “replaces” the whole expression
  with a **`true`** or **`false`** value
  - If the temp variable had a value of 102, this becomes “true”
  - If the temp variable had a value of 92, this becomes “false”

Logical Operators

- Logical Operators are:

&& logical AND

a	b	a && b
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

|| logical OR

a	b	a || b
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

! logical NOT

a	!a
FALSE	TRUE
TRUE	FALSE

The if-statement

- General form:

  `if ( condition )`

  Followed by statement (or block of statements) to be executed if condition is true

- Example

```
if (length < 2)
   println("Too short!");
```

Code Blocks

- Processing allows the use of if-statements without defining of code blocks, provided that there is only one command following the if-statement
- But this just gets confusing
- General practice: **always** define a code block using { } brackets

Code Blocks

- Example:

```
if (length < 2) {
    println("Too short!");
}
```

Open code block with {
on the same line as the if-statement

Indent code in the block by one tab-stop

Close the code block with } aligned on the same horizontal "level" as the original if-statement

Code Blocks

- This convention works well,
  even with multiple-nested layers of code:

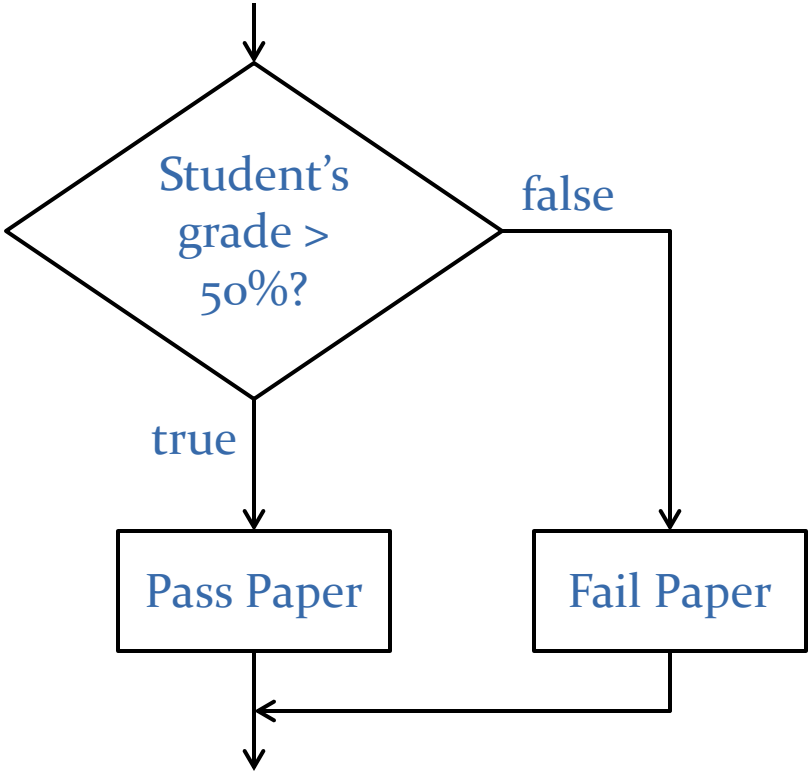
```
if (length < 2) {
  if (age > 16) {
    println("Too old");
  }
  println("Too short");
}
```

The if-else-Statement

- Select one of two possible execution paths, based on the result of a comparison
- General format:

```
if ( expression ) {
   statement block 1
}
else {
   statement block 2
}
```

Example




```
if ( grade > 50 ){
   passPaper();
}
else {
   failPaper();
}
```

More on if-Statements

- An if statement executes its body
  when (and only when) the condition is true
- If the condition is false, the body is skipped,
  and execution picks up at the first statement **after** the if
- By default, the body of an if statement
  is restricted to the first statement
  that follows the "if (condition)" line
  - Indentation doesn't matter
  - This can lead to trouble…

Empty Statements

- A semicolon by itself
  is a valid (but non-functional) statement
- Common mistake:
  putting a semicolon immediately after an if statement

```
if (x > 5);
   println("x is greater than 5!");
```

- With the semicolon, the print statement
  will execute regardless of the value of x

Compound Statements

- **`if`** and **`else`** only execute a single following statement
- We can get around this
  by enclosing multiple statements in braces
- The resulting block is called a compound statement
- Always use braces around the body of an if or else clause
- If you get in to the habit
  of always following an if-statement with a { } block,
  then it helps to avoid the semicolon error

Nested if-Statements

- A statement block may contain another if statement

```
if (age > 18) {
  if (size < 1.50) {
    println("Pretty small adult");
  }
}
```

- Even if **`size`** is less than 1.50,
  the text won't be printed unless **`age`** is more than > 18

If-else Chains

- Another useful command in Processing is “else if”

```
if ( expression_1 ) {
   statement 1
}
else if ( expression_2 ) {
   statement 2
}
else {
   statement 3
}
```

- What is the “order” of these statements?
  - Write some code and try it for yourself!!

Switch Statement

Switch-Case Statements

- You aren't restricted to just using if-statements
  - switch()
  - case
  - default
  - break
  - continue
  - ?: (conditional)

This is actually a shorthand way of writing if-else
Avoid it like the plague
as it mostly does not improve the clarity of your program!

Switch-Case Statements

- Using switch-case statements can help avoid accidentally nested if-statements
- Basically works like a series of if-/else-if-statements where each option is mutually exclusive

Simple Switch-Case Example

```
int num = (int) random(2);

switch (num) {
  case 0:
    println("Zero");
    break;
  case 1:
    println("One");
    break;
}
```

The “break” command is important,
it ends the code block and tells the program to exit the switch and continue the rest of the program

Switch-Case Problems

- Sometimes variables have values that you didn't expect
- This causes weird behaviour with your switch-case statements
- The solution is to **always** have a default case!

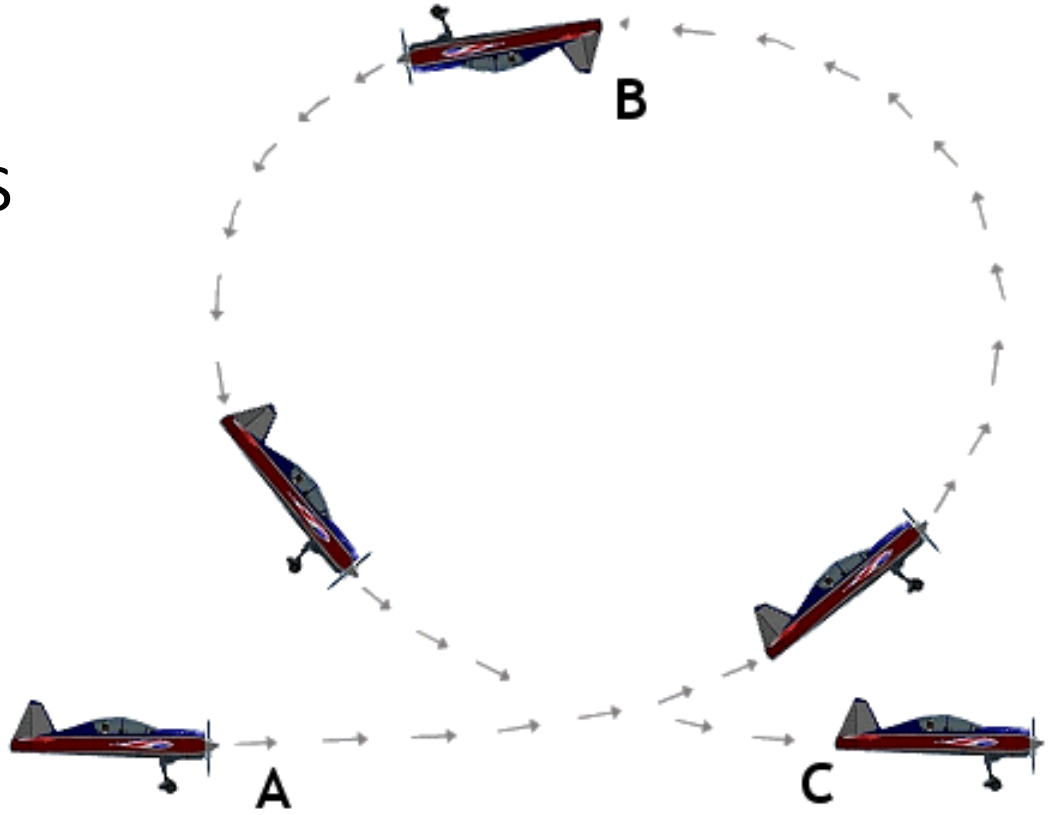
Switch-Case with Default Example

```
char letter = (char) random('A', 'Z'+1);

switch(letter) {
  case 'A':
    println("Alpha");
    break;
  case 'B':
    println("Bravo");
    break;
  default:  // Default executes if no match
    println("None");
    break;
}
```

Any idea why we need the "+1"?

Loops



Iterative Programming

- Many programs perform the same task many times
- Operations are repeated on different data
  - Adding a list of numbers
  - Displaying frames of a movie file
- Repetitive tasks are specified using loops
  - You've already used loops without realising it!
  - The **`draw()`** function is part of an infinite "while" loop

Loop Elements

- All loop constructs share four basic elements:
  1. Initialization
  2. Testing the loop condition
  3. Loop body (the task to be repeated)
  4. Loop update

Initialisation

- This section of code is used to set starting values
- For example, setting a total to 0 initially
- This can be done as part of the loop,
  or separately before the loop code begins

Loop Tests

- Test expressions are used to determine whether the loop should execute (again)
- Tests compare one value/variable with another
- If the test evaluates to **`true`**, then the loop will execute (again)
- If the test is always **`true`**, you get an infinite loop

Loop Body

- Basically a code block that defines the commands that are iteratively repeated

Loop Update

- This step changes the value(s) of the loop variable(s) before the loop repeats,
  e.g., moving to the next item to process
- This can be done explicitly as part of the loop,
  or it can be done inside the loop body.
  There is no right way of doing this, either works fine.

While Loops

- A while loop executes as long as the test condition is **`true`**
- It does not have a fixed number of iterations, therefore has the potential to be an infinite loop which is normally not desirable!
- General form:

```
initialization
while ( loop condition test ) {
   loop body
   loop update
}
```

While Loop Example

- What does this loop do?
- How often does it iterate?
- What is the output?

```
int countDown = 5;
while ( countDown >= 0 ) {
  println(countDown + "...");
  countDown = countDown - 1;
}
```

While Loop Example

- What does this loop do?
- How often does it iterate?
- What is the output?

```
int countDown = 5;
while ( countDown >= 0 ) {
  println(countDown + "...");
  countDown = countDown + 1;
}
```

For Loops

- A for loop executes a fixed number of times
- General form:

```
for ( initialization ;
      loop condition test ;
      loop update ) {
  loop body
}
```

For Loop Example

- What does this loop do?
- How often does it iterate?
- What is the output?

```
for (int i = 0; i < 10; i = i + 1) {
  println(i);
}
```

Choosing Loop Types

- For a fixed number of iterations:
  For-loops are generally considered the way to go
- For a variable number of iterations:
  While-loops can execute 0 or more times
- However, each type of loop can be rewritten as the other type

Nested Loops

- The body of a loop can contain any other type of statement(s)
- Even other loops
- If the outer loop executes n times, and the inner loop executes m times, the body of the inner loop will execute (n × m) times
- You can even mix while and for loops, Processing doesn't care!

Nested Loop Example

- Always use indentation and good coding style!

```
for (int i = 0; i < 4; i++) {
  for (int j = 0; j < 4; j++) {
    print("*");
  }
  println();
}
```

Advanced Loops

- There are a whole bunch of subtleties we will explore about loops as we go
- We have just covered the basics for now, it's plenty to be going on with!

Time to Program

You won't learn to program
just by listening to me talking about concepts....

... so let's do some programming!