Introduction to Programming

Week 1, Topic 2

The Tasks and Assessment

- What we look at in this topic:
  - Assessment – portfolios
  - Examples of student work
  - Basic Programming Concepts for this Week
  - Tasks for Week 1
  - What else to do this week

Portfolio Assessment

- You pick the grade level you want to aim for
- Work through the tasks
- There are tasks to do each tutorial/laboratory and ones to do out of class.
- You need to be coming to tutorial/labs to get your feedback and to demonstrate your tasks.
- Start the tasks when indicated and work at your own pace.
- The idea is that you can try, fail and try again - mistakes are part of learning

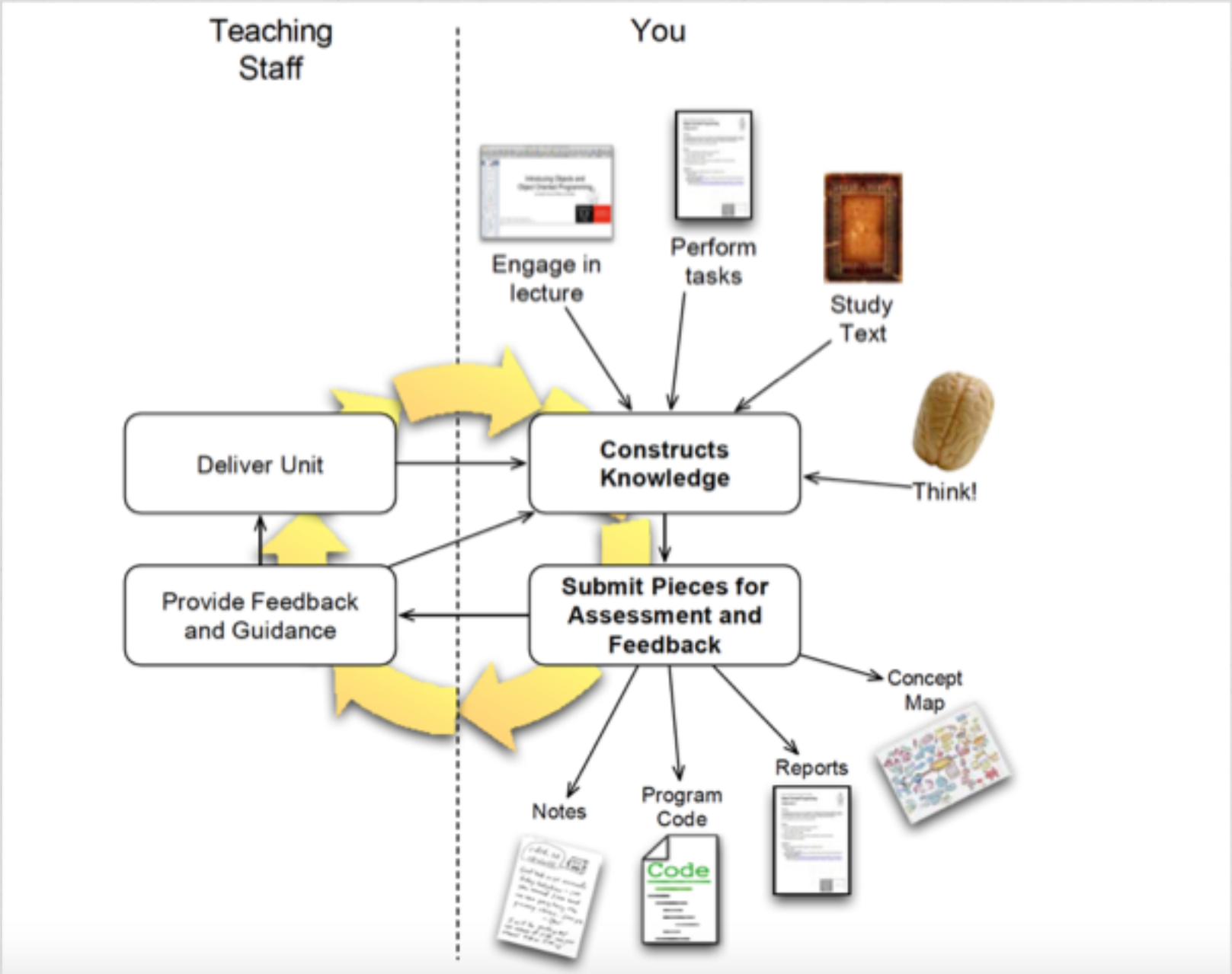
Tests (and Workshops)

- There are two ‘tests’
- The first is a timed task held in the tutorial in Week 7. This is for ***formative feedback*** (i.e information for you and your tutors on who you are going). But you are required to do this as part of your portfolio.
- The second test is ***summative assessment*** – it is to demonstrate you have reached the minimum level of competency to pass the unit. **You must pass this test!**
- The second test is **held in the Workshops in Week 10**, and if you need to redo it, then you can – in the workshop in Friday of Week 12. These might be the only times you attend a workshop.

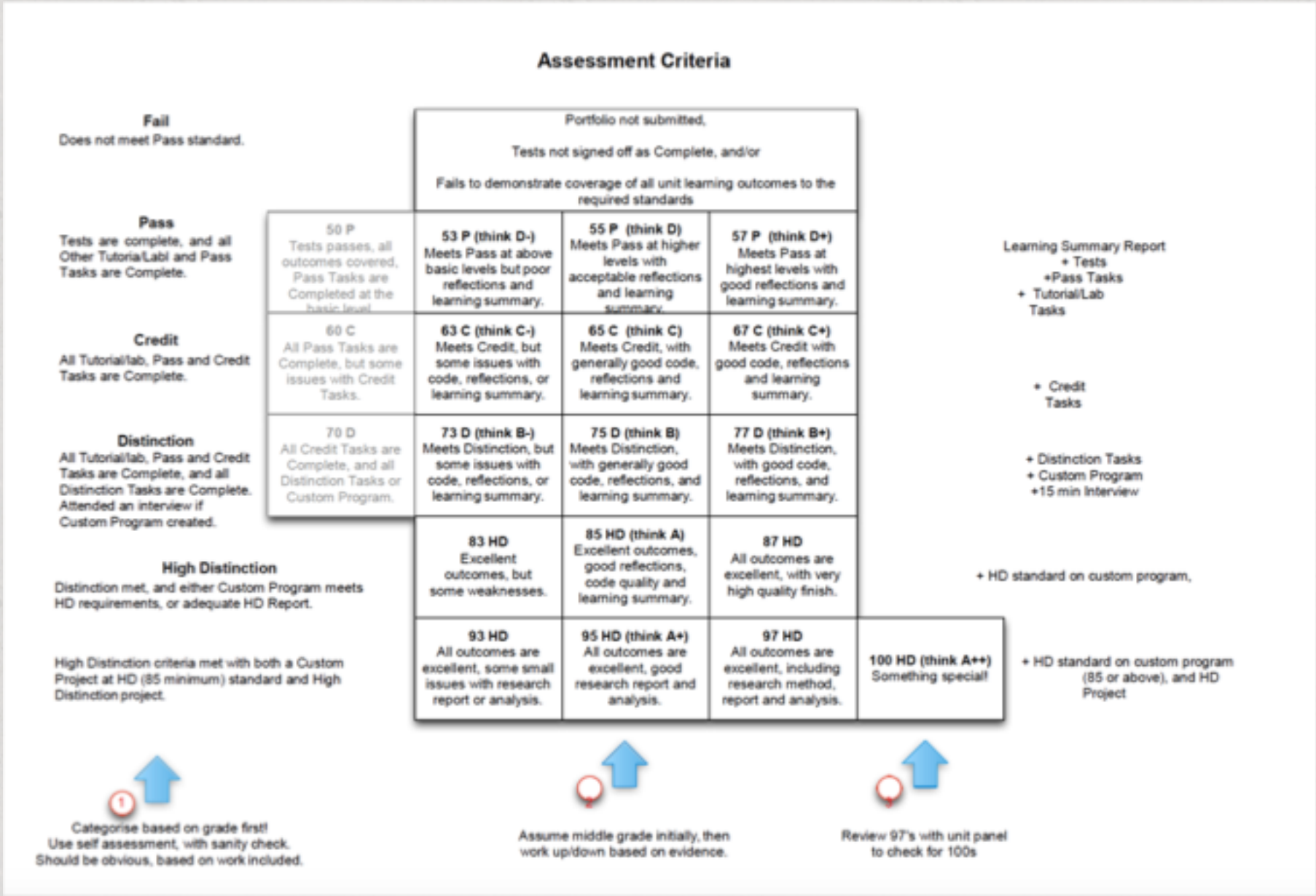
Portfolio completion and Submission

- Complete all the tasks.
- At the end of the unit complete a Learning Summary Reflection.
- Submit the portfolio (you must do this at the end of the unit, or your portfolio may not be graded).

Portfolio Process



Portfolio Assessment



Examples of student work

- Pass and Credit students do a Music Player as their major task.

- Distinction and High Distinction students also do a custom project.

Lets see some examples from past semesters …

Lets start learning programming!

- Programming principles – sequence, selection, iteration, modularisation.
- Variables, constants, literals, types.
- Input and Output – basic input/output, sources and destinations (eg: file, terminal, network). Formatted output. Formatted input.
- Desk checking, testing, selecting test data

Sequence

- Sequence
  - One thing follows another e.g:
    - Get out of bed
    - Have a shower
    - Get dressed
    - Have breakfast
- Is Sequence important in the above?

  (we see another example later)

Selection

- A point of decision
- You may have seen this in flow charts
- Eg:

We will look at how you do this in a program in Week 3.

Iteration (repetition)

- Do something over and over again. E.g:
  1. Lay first brick
  2. Get next brick
  3. Lay next brick
  4. Repeat Steps 1 – 3 until wall is finished (or lunch time – whichever comes first)
- We will look at how you do this in a program also in Week 3.

Modularisation

- Having different separate (simpler) components, rather than one (usually more complex) integrated unit. E.g:

NAD stereo system from Audio Trends in Ringwood – with separate components

Crosley Record player from ASOS. All-in-one.

Writing Ruby Programs

We will look now at:

- Basic program structure
- Basic data types
- Variables
- Basic input and output

Keywords

- These are words which are reserved as having a special meaning in the language.
- Eg: **begin**, **end**.
- They are highlighted as black in the Atom IDE (Integrated Development Environment)
- Atom, Sublime Text, Notepad++ are all editors you can use to write Ruby code.
- But we recommend using Visual Studio Code.

Basic Program Structure

```
1
2  # Below is the procedure called main.
3  # main includes a block of code.
4  def main
5    # Below is the second line of code executed
6    puts "Hello World"
7    # code in a block should be indented.
8  end
9
10 # This is the first line of code executed
11 # It calls the procedure main() above.
12 main
13
```

2. The program goes to the called procedure (block)

3. Then goes through in sequence

1. The program starts at the first statement not in a block

Basic Data Types

For now we will deal with three main data types:

- Strings – these are a ‘string’ of characters eg: “hello”
- Integers – these are whole numbers e.g: 10
- Real (or floating point) numbers e.g: 3.142
- Booleans – these are used to represent true or false (which are keywords)

Any of these values can be stored in the computer using *variables* - which are a named part of the computer’s memory which can have its contents changed as the program executes.

We can also declare *constants*, whose value is set once and should never change after that. Constants are usually given upper-case names e.g:

PI = 3.142

Variables (and printing to the terminal)

To store the values above we create and initialize variables as follows:

```
1
2   def main
3     greeting = "Hello"
4     age = 10
5     pi = 3.142
6     puts greeting
7     puts age
8     puts pi
9   end
10
11  main
```

The = sign **assigns** the value of 10 to the variable called *age*.

puts prints the value of the variable to the terminal.

```
MacBook-Pro:Code in Progress mmitchell$ ruby example.rb
Hello
10
3.142
```

Reading from the terminal

gets reads strings from the terminal. If we want to read in a number then use it to do calculations we must change what we read from a string type to an integer type (i.e we must convert it). Here is how we do that:

```
1
2  def main
3    puts "Enter your child's age: "
4    age = gets.chomp.to_i
5    age = age + 2
6    puts age
7  end
8
9  main
10
```

gets reads a string from the terminal, then *chomp* removes whitespace before *to_i* converts the value to an integer.

Before we can **assign** a new value to *age* value we need to **evaluate** the current value of *age.*
In this case we **increment** the value of *age* by 2.

```
MacBook-Pro:Code in Progress mmitchell$ ruby example.rb
Enter your child's age:
10
12
```

Operators and Statements

We have seen one operator already - that is the **assignment** operator: = .

Most languages have the standard mathematical operators such as **+**, ***** (multiplication) and **/** (divide).

The first line of the code below takes a hard-coded value (10) and **assigns** it to the variable *sub_total*. The next line contains a **statement** that **evaluates** the value of the variable *sub_total* then multiples that by the value of the constant *FRACTION* before assigning the result of the addition to the variable total.

```
1 FRACTION = 0.5
2
3 sub_total = 10
4 total = sub_total * FRACTION
5 puts total
6
7
```

Scope

Variables exist within a **scope**.

The following code shows two types of scope - the first variable *FRACTION* has **global** scope - it is accessible to all blocks of code (i.e procedures).

The second variable *total* has only **local** scope - i.e it is only accessible in the block in which it was first used (or defined) in this case the block is the procedure add_to_total:

```
1  FRACTION = 0.5
2
3  def add_two_numbers(a, b)
4      total = a = b
5      puts 'total is: ' + total.to_s
6  end
7
8  def main
9      add_two_numbers(16, 5)
10 end
11
12 main
```

Designing and Testing Programs

When we write a program we need to think about:

- What do we want the program to do?
- How can we check it is doing what we expect?

Ideally we will have some test data that we can use to check that the program is doing what we expect.

We will look at how to approach testing a program in the tutorial/lab tasks this week.

Tasks for This Week

- Tutors will go over these in the lab classes.
- You should try and complete (or at least start) tasks marked with a T in the tutorial/lab classes. These are referred to as Tutorial Tasks.
- Tasks with a P, C, D or HD in their abbreviation are referred to as Level tasks.

What else to do this week?

- Visit the Programming Help Desk in ATC620.
- You can work there even if you are not seeking help (but maybe if it is crowded allow space for those who need help).
- Check the timetable for when COS10009 staff are rostered on.
- Use the booking system at the front of the room to add your name to the list.

That is it!

Make sure you attempt this weeks tutorial/lab task and the first Pass task.