



ENGG1003

INTRODUCTION TO PROCEDURAL PROGRAMMING



STAFF

- Course Coordinator: Prof. Steven Weller
 - Steven.Weller@newcastle.edu.au
- Prof. Sarah Johnson
 - Sarah.Johnson@newcastle.edu.au
- Mr. Brenton Schulz
 - Brenton.Schulz@newcastle.edu.au
- See course outline for consultation hours
- Lab demonstrators
 - Too many to mention, you'll meet them in labs
 - Mix of postgrad and undergrad students
 - Ask them about their work, future studies, etc!



Steve



Sarah

BLACKBOARD

- Accessed via:
<http://uonline.newcastle.edu.au>
 - Does anyone use QR codes? Didn't think so, have one anyway.
- All courses upload notes, lecture recordings, announcements, grades, etc. to Blackboard
- Your responsibility to check regularly, typically daily.



DISCORD

- We will be utilizing Discord
 - Invite link: <https://discord.gg/sfgpR4kMbN>
- Great for:
 - Quick questions
 - Much faster than email!
 - Connecting with peers
- Can be used as online consultation if Zoom doesn't work
- Staff will participate



COURSE CLASSES

- Two lectures per week – both via Zoom
 - This one (Mon 9-11am for those watching the recording)
 - Thursday 4-5pm
- Two computer labs - **Start this week!**
 - One on campus
 - You sit at a PC among 20-40 other students and get given tasks to do
 - Tasks distributed on Blackboard, typically a PDF, maybe template code
 - One or two demonstrators are paid to be there and answer your questions
 - One via Zoom – links will be on Blackboard

TEXTBOOKS

- S. Linge & H.P. Langtangen, Programming for Computations – Python (2nd edition), Springer Open, 2020. ISBN 978-3-030-16876-6 ISBN 978-3-030-16877-3 (eBook)
- This textbook is an open access publication, available for free as an eBook via the University of Newcastle library.
- Direct link: <https://link.springer.com/book/10.1007%2F978-3-030-16877-3>

THINKING OF DROPPING OUT?

- Come talk to us! What can be done to help?
- HECS Census: **19 March**
 - Last day to withdraw without financial or academic penalty
- Withdrawing between 20 March and 4 June does not incur academic penalty
 - You still pay for the course
- Withdrawing 12:00:01am on **5 June** or after results in a fail

WHERE TO FIND HELP

■ Google

- Copy/paste error messages
- Search for Python tutorials (there are *lots*)

■ Discord

- A peer or staff member might be around to help you

■ Your lab demonstrator

- Only during enrolled lab times
- They will help you do simple debugging, search for solutions, read documentation, etc

■ The textbook

ASSESSMENTS

- Passing grade is 50%
 - Must also score 40% or higher in the final exam
- Assessed Laboratory 1 Week 4, 5%
- Mid-semester quiz Week 6, 15%
- Programming assignment 1 Week 7, 20%
- Assessed Laboratory 2 Week 9, 5%
- Assessed Laboratory 3 Week 11, 5%
- Programming assignment 2 Week 13, 15%
- Final Exam Final exam period, 35%

ATTENDANCE

- An overall attendance of at least 80% is required for **on-campus labs**
 - ie: mandatory attendance applies to weekly 2-hour computer lab in ENGG1003
 - Uni-wide policy applies to all students in 1000-level courses e.g. ENGG1003
- Mandatory attendance does **NOT** apply to the weekly 1-hour Zoom lab
- Do **NOT** attend a lab if you are unwell!
 - Attending a later lab session does not require a medical certificate
 - If attending a later lab session, advise the demonstrator in that session

ASSESSMENTS

- Programming is quite unforgiving
 - If you develop code on a private machine it may not work on the university computers
 - Assessment demonstration on privately owned laptops is totally fine, if not preferred
- All assessments (except the final exam) are graded during your lab session
 - Assessment in a different lab session requires approval

WHAT IS "PROCEDURAL PROGRAMMING"?

- Telling a computer what to do via a list of steps
- Written in a language the computer can understand
 - Ideally, the human writing understands it too
- This course uses the language “Python”
 - Top language in IEEE survey multiple years running
 - Incredibly useful in isolation; fantastic platform to learn basics before learning other languages

WHY DO I NEED PROGRAMMING?

- ELEC/MECHA/Computer systems engineering
 - Embedded systems, programming small computers in home appliances, UAVs, wireless sensors, internet of things (IoT) devices, etc
 - You will all do this in ENGG1500 on the “STM32” microcontroller platform using microPython
 - Control systems – MATLAB (possibly also Python)
 - Designing mathematical models which make a thing do a thing
 - Eg: Car cruise control, temperature control, controlling robot arms, etc
 - Numerical methods
 - Catch-all term for any kind of heavy lifting arithmetic done on a PC or supercomputer
 - Applications typically quite specific

WHY DO I NEED PROGRAMMING?

- MECH/CHEM/Medical/Aerospace
 - Many of you will program embedded systems in C later on
 - Eg: MECH students use Arduinos in 2nd year
 - Almost all medical equipment is an "embedded" system
 - MATLAB is used extensively for various applications, Python slowly taking over
 - Ask your demonstrators in other courses?

WHAT IS A COMPUTER?

- How is this relevant to this course?
 - In order to write instructions (programming), you must have a relevant understanding of how computers work
- A computer is an electronic device designed to perform calculations very quickly
- This seems rather restrictive, just performing mathematics
- But when you consider its other capabilities
 - Speed
 - Communication with other electronic devices (peripherals)
- Then mathematics gives you
 - A word processor, sinus rhythms of a person's heart, how the ailerons should move to bank a plane, how a robot should weld a car body, how much heat is needed to maintain a chemical reaction, weather predictions, etc.

FUNDAMENTAL COMPONENTS

- Fundamental components of every computer



INPUT

- Computers only understand electrical signals
- More specifically those signals represent two states ON or OFF (binary 0 or 1)
- What about a keyboard?
 - It is a device which converts each keystroke into a series of ON/OFF voltages
- What about a mouse?
 - It is a device which converts movements into a series of ON/OFF voltages
- For our model we consider *input* to be
 - Any series of ON/OFF voltages which the computer needs to perform its calculations
- A device that generates input we will call an *input device*

PROCESSING

- *Processing* is the main function of a computer
- Once the input for a calculation is available, the computer will perform some processing
- Processing is a series of manipulations performed on the input
 - This involves following a very specific set of instructions
 - Instructions can be ADD, SUB, MUL, DIV, MOV, SQRT, etc...
 - The writing of those instructions is called *programming*

OUTPUT

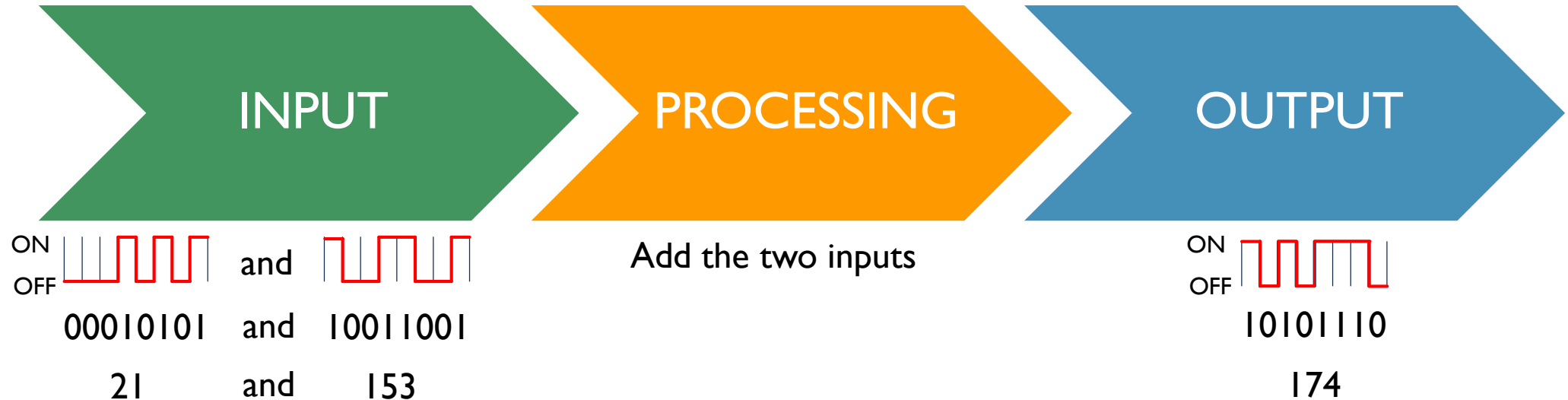
- Once the processing is complete
- The computer must have a way of presenting the results
- This is the *output*
 - Output is any series of ON/OFF voltages that represents the results of processing
- To make the output more useful we need an *output device*

OUTPUT

- An output device is any peripheral that takes a series of ON/OFF voltages and manipulates them into something useful or human readable
- Examples:
 - Printer
 - Monitor
 - Automotive cruise control throttle actuators
 - LCD showing the oxygenation level of a patient's blood
 - Rudder adjustment in a fly-by-wire system

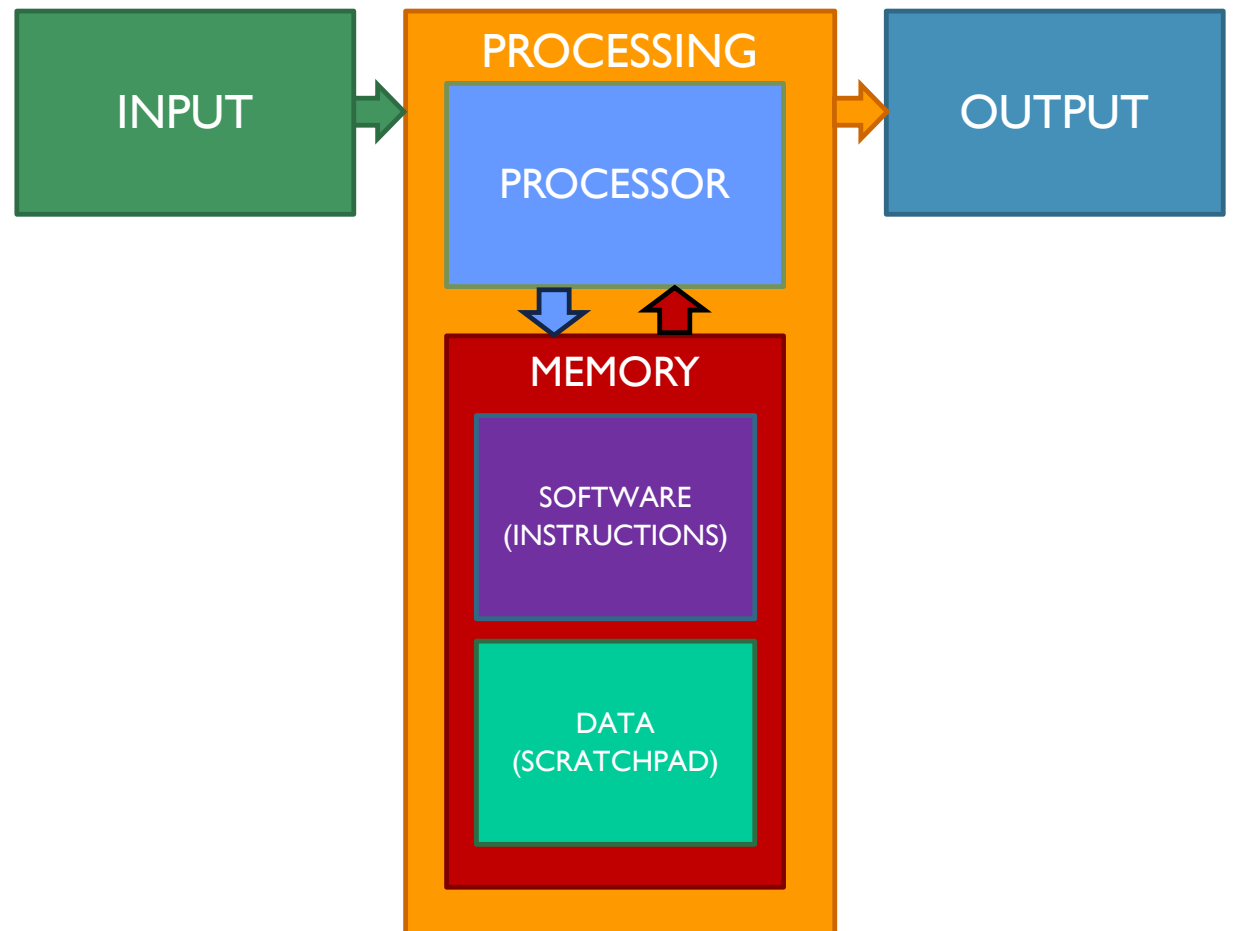
EXAMPLE FUNDAMENTAL COMPONENTS

- Fundamental components of every computer



PROCESSING IN DETAIL

- Processing is complex
- Requires multiple key sub-components to help
- For our purposes we define
 - PROCESSOR
 - MEMORY
 - SOFTWARE



PROCESSING IN DETAIL - PROCESSOR

- Processor
 - Performs mathematical and data manipulation tasks
 - Has sub-components, but they are not relevant for this course
 - More detail in ELEC1710

PROCESSING IN DETAIL - EVERYTHING IS A NUMBER

- Computers can only process numbers
- Things that aren't numbers need numerical "codes"
 - Text: ASCII and Unicode
 - Pictures: most commonly, each *pixel* is allocated a red-green-blue intensity value
 - Sound: the “waveform” is sampled at regular intervals and stored as a series of numbers
 - (NB: multiple standards exist for all the above)

PROCESSING IN DETAIL - MEMORY

- Memory
 - Like humans, computers need to store intermediate results
 - Memory acts like a set of written notes for the computer
 - Further relevant subdivision
 - Software – Instructions for the calculation
 - Data – The information required for the calculation

PROCESSING IN DETAIL - SOFTWARE

- Software
 - This is the main event for this course (the Python *interpreter*)
 - These are the detailed instructions that the processor will follow to perform the desired calculations
 - Instructions **directly** understood by the processor are
 - Very specific to each processor (known as the *instruction set*)
 - Limited in number
 - Simple, eg: add number in memory location 1 to the number in memory location 2 and put the result in memory location 3
 - Again each instruction is encoded as a series of ON/OFF voltages

SOFTWARE

- Software for our purpose will be divided into three groups
 - Machine code
 - Assembly language
 - “High level” languages

```
17 string sInput;  
18 int iLength, iN;  
19 double dblTemp;  
20 bool again = true;  
21  
22 while (again) {  
23     iN = -1;  
24     again = false;  
25     getline(cin, sInput);  
26     system("cls");  
27     stringstream(sInput) >> dblTemp;  
28     iLength = sInput.length();  
29     if (iLength < 4) {  
30         again = true;  
31         continue;  
32     } else if (sInput[iLength - 3] != '.') {  
33         again = true;  
34         continue;  
35     } while (++iN < iLength) {  
36         if (isdigit(sInput[iN])) {  
37             continue;  
38         } else if (iN == (iLength - 3)) {  
39             continue;  
40         }  
41     }  
42 }
```

SOFTWARE – MACHINE CODE

- The processor can only understand machine code
 - Example, one instruction for an x86-based CPU
 - 0110 0110 1000 0011 1100 0000 0000 1010
 - Difficult for humans to understand
 - Very processor-specific
 - Will not be understood by another processor




SOFTWARE – ASSEMBLY LANGUAGE

- Assembly Language
 - Uses simple mnemonics to describe the purpose of the instruction
 - Example, one instruction for a x86 based CPU
 - Machine code: 0110 0110 1000 0011 1100 0000 0000 1010
 - Assembly language: `ADD AX, 10`
 - A bit easier for humans to understand
 - Still processor-specific




SOFTWARE – HIGHER-LEVEL LANGUAGES

- High level Languages (C, MATLAB, Java, Python, C++, FORTRAN,...)
 - Uses more human readable text-based code
 - Increases the complexity of each line of code so that common calculations can be done with fewer lines
 - Example, one instruction for a x86-based CPU
 - Machine code: 0110 0110 1000 0011 1100 0000 0000 1010
 - Assembly language: `ADD AX, 10`
 - In Python: `x = x + 10`
 - Much easier for humans to understand
 - Not processor specific
 - Allows writing of much more complicated instructions


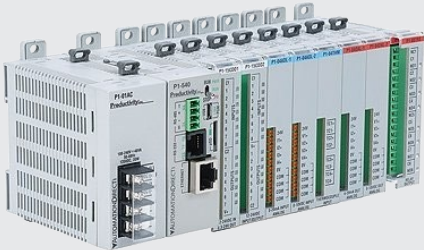

RECOGNISING COMPUTERS

Device	Specification
Desktop PC 	INPUT DEVICE: Keyboard and Mouse OUTPUT DEVICE: Monitor, Speakers PROCESSING: PROCESSOR: Intel i7 64-bit CPU MEMORY: 8GB RAM
Laptop 	INPUT DEVICE: Keyboard and Touch Display OUTPUT DEVICE: Monitor, Speakers PROCESSING: PROCESSOR: Intel i7 64-bit CPU MEMORY: 8GB RAM
Smart Phone 	INPUT DEVICE: Touch Sensor, Microphone, Accelerometers, GPS receiver, 4G Receiver OUTPUT DEVICE: Display, Speaker PROCESSING: PROCESSOR: Qualcomm Snapdragon 808 ARMv8-A 64-bit CPU MEMORY: 8GB RAM


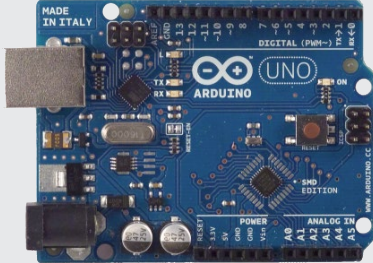
RECOGNISING COMPUTERS

Device	Specification
Raspberry Pi 3+ 	INPUT DEVICE: Keyboard, Electrical signals (I/O) OUTPUT DEVICE: HDMI Port to Display, USB, Audio PROCESSING: PROCESSOR: Broadcom BCM2837B0 64-bit quad-core ARM Cortex-A53 CPU MEMORY: 1 GB RAM
Sony PlayStation 4 	INPUT DEVICE: Gaming Controller OUTPUT DEVICE: HDMI Port to Display PROCESSING: PROCESSOR: AMD x86-64 “Jaguar” CPU MEMORY: 8GB RAM
Apple TV 	INPUT DEVICE: Remote control OUTPUT DEVICE: HDMI Port to Display PROCESSING: PROCESSOR: Apple A10X Fusion ARM 64-bit CPU MEMORY: 2GB RAM

RECOGNISING COMPUTERS

Device	Specification
Smart TV 	INPUT DEVICE: Remote OUTPUT DEVICE: Display PROCESSING: PROCESSOR: Dual-core ARM Cortex-A9 1Ghz MEMORY: 1GB RAM
PLC (Programmable Logic Controller) 	INPUT DEVICE: Electrical signals OUTPUT: Electrical signals PROCESSING: PROCESSOR: Intel 8051 CPU MEMORY: SOFTWARE: 2KB RAM DATA: 128B RAM
Defibrillator 	INPUT DEVICE: Electrical signals from electrodes OUTPUT: Defibrillation current PROCESSING: PROCESSOR: STM32 STM32F429 ARM-Cortex M4 32-bit CPU MEMORY: 256KB RAM

RECOGNISING COMPUTERS

Device	Specification
Network Router 	INPUT DEVICE: Ethernet, Radio Signals OUTPUT DEVICE: Ethernet, Antennae PROCESSING: PROCESSOR: Broadcom BCM21664T Dual-core ARM Cortex-A9 32-bit CPU MEMORY: 1 GB RAM
Arduino UNO 	INPUT DEVICE: Electrical signals OUTPUT: Electrical signals PROCESSING PROCESSOR: Microchip ATmega328 8-bit Microcontroller (CPU) MEMORY: SOFTWARE: 32KB RAM DATA: 2KB RAM

RECOGNISING COMPUTERS

- All these devices are
 - Computers
 - Have the fundamental elements input, processing, and output
- What does this mean for you as a programmer?
 - Be aware that your target computer may have limitations
 - Different computers have different programming requirements

INTRODUCTION TO PYTHON – FUNDAMENTAL CONCEPTS

- Python is an *interpreted* language
 - This means an *interpreter* takes code from a text file or command prompt and converts it into machine instructions “on the fly”
 - Machine instructions are then *executed* by a computer
 - In this course "computer" will be a PC or laptop
 - Could be a microcontroller, mobile phone, supercomputer cluster, etc
- We will use the PyCharm integrated development environment (IDE)
 - Download “community edition” from:
<https://www.jetbrains.com/pycharm/download/#section=windows>

INTRODUCTION TO PYTHON – FUNDAMENTAL CONCEPTS

- Moving data into and out of a Python program
 - *Standard input*: text characters read from a keyboard
 - *Standard output*: text characters sent to the screen
 - Typically printed to a *console*
 - *File I/O*: from or to files stored on a hard disk / USB flash drive / etc
 - Covered in later weeks

INTRODUCTION TO PYTHON –FUNDAMENTAL CONCEPTS

- Other input/output methods beyond this course:
 - Microcontroller pins (GPIO – in ENGG1500 and ELEC1710)
 - Embedded systems communication standards, covered in ELEC2720, ELEC3730, MCHA-something
 - I2C
 - SPI
 - UART
 - TCP/IP networking
 - USB devices
 - Loads of others

SOME BASIC PYTHON PROGRAMS

- Being an *interpreted* language Python programs can be just one line:
 - Examples (introduce PyCharm, run live!)
 - `print("hello")`
 - `1+1`
 - `print(1+1)`

SYNTAX

- What on Earth is *syntax*?
 - In human languages: the order of words in a sentence
 - Are you going to the movies on Tuesday?
 - Are you on Tuesday to the movies going?
 - In computer languages: the structure of the text given to the compiler
 - For example, `print("hello")` is different from `print(hello)` or `print hello`

SYNTAX

- The syntax rules in programming are typically **very** strict
- Incorrect syntax can result in Python generating syntax errors
- Eg:

```
>>> print("hello")
hello
>>> print hello
  File "<input>", line 1
    print hello
        ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print(hello)?
```

CASE SENSITIVITY

- Most programming languages are *case sensitive*
- This means that `print()` is **totally different** to `Print()` or `PRINT()`
- Fundamental reason: `p` and `P` are different ASCII characters
 - ASCII maps letters to numbers because computers only work with numbers

FUNDAMENTAL CONCEPTS – INPUT

- We saw `print()` for text output
- One function which reads *standard input* is `input()`
- It reads input text and converts it into other datatypes
 - Eg: converts the text “13” to the *number* 13
- Example: run `x = input("Type a number:")` followed by `print(x)`
- Huh? What was that `x` thing?
 - New fundamental concept: **variables**

FUNDAMENTAL CONCEPTS - VARIABLES

- A *variable* is something that stores data
 - "Data" is one or more numbers
 - Each variable needs a unique name
 - They are used to store numbers while your program runs
- Today we will run an example with *integer* variables
 - In Python 3.x integers have no (practical) upper limit
 - Factorial example later on
 - More details in the coming weeks

FUNDAMENTAL CONCEPT: ASSIGNMENT

- Computer languages use the $=$ character for *assignment*
 - This is **distinctly different** from algebraic equality!
- Assignment means:
 - "Take what's on the right side and store it in the thing on the left"
 - You can read " $a = a + 5$ " as "a **becomes** a plus 5"
 - The value of " $a+5$ " is calculated and replaces the old value of a
- Eg: Give the variable x the value 2:
 - $x = 2$
- Eg: Add a and b together, store the result in c :
 - $c = a + b$

PUTTING IT ALL TOGETHER

- Example 1: Type Python commands which:
 - Reads 2 integers from the keyboard (from the “Python console”)
 - Multiplies them together
 - Prints the result to the Python console
- Example 2: Type Python commands which
 - Reads an integer from the Python console
 - Calculates its factorial
 - Prints the result to the Python console
 - NB: This will use the math *library* – more on libraries next week

PUTTING IT ALL TOGETHER

- Repeat the previous 2 examples without using `input()` or `print()`
 - There are always multiple ways to solve the same problem!

WHAT NEXT?

- Install PyCharm “Community” edition
 - Download from: <https://www.jetbrains.com/pycharm/download/>
 - Installation details in Week 1 lab notes
 - Brenton has put together a short video stepping through PyCharm installation: <https://bit.ly/3blcsoK>
- Read textbook Chapter 1:
 - Direct link: <https://link.springer.com/book/10.1007%2F978-3-030-16877-3>
 - Run examples as you go!
- Read Week 1 lab notes
- Attend your on-campus and Zoom labs this week