IT@JCU

CP1401 – Module 1 Computers and Problem Solving

JAMES COOK UNIVERSITY AUSTRALIA

Learning outcomes – you will be able to:

- Understand the subject expectations, goals, assessment, etc.
- Understand basic computing hardware and software
- Appreciate historical developments in computers and programming
- Understand and apply the basic problem-solving technique of decomposition
- Write simple algorithms using pseudocode and flowcharts

Overview of CP1401

- Learning outcomes:
  - Problem-solving skills
  - Programming skills, using Python
- The basis for other subjects, including
  - CP1404 – Programming 2 (Python)
  - CP2406 - Programming 3 (Java)
- Introduces fundamental principles and practices of computer programming with an industry-standard programming language
- Best-practice from the start

IT@JCU

CP1401 Assessment is mostly practical
(skills not just knowledge)

- Design and code programs in Python
  - Assessment item 1: Software development / creation, 60%
    - Due: 11 December 2022 (A1) – 24%
    - Due: 22 January 2023 (A2) – 36%
  - Assessment item 2: Practical assessment/practical skills demonstration, 40%
    - Due: weekly
    - 5 Online Tests
- All assessment graded at an “introductory” level
- All assessment is individual (no group work)

CP5639 Assessment

- Design and code programs in Python
    - Assessment item 1: Assignment – Software development/creation, 50%
        - Due: 11 February 2022 (A1) – 20%
        - Due: 22 January 2023 (A2) – 30%
    - Assessment item 2: Practical assessment/practical skills demonstration, 30%
        - Due: Weekly
        - 5 Online Tests
    - Assessment item 3: Report & Presentation, 20%
        - Due: 02 January 2023
- All assessment is individual (no group work)

Stay active and engaged

- Lecture videos (in short chunks) and activities each week
  - Content explanations, coding demonstrations and guided activities
- Help/meetup sessions
  - Opportunity to ask questions and get help
  - Based on the content
  - Do the lecture (and activities) *before* coming to the session
- Practical
  - Implementing and practicing what you have learned

"Do this now" activities will really help you

- In lecture videos there will be "do this now" activities
- If you don't know how to do it all:
  - Do the parts you do know
  - Take note of what you still need to learn - then revise these topics

"Do this now" activities have helped others

- "Do this now activities were helpful in understanding the content"

- "Lecture activities were quite useful in helping me understand how to code."

Do this now

Write a set of instructions for how to wash your hair

- These should be simple and easy for someone else to follow

Write this down:

100%

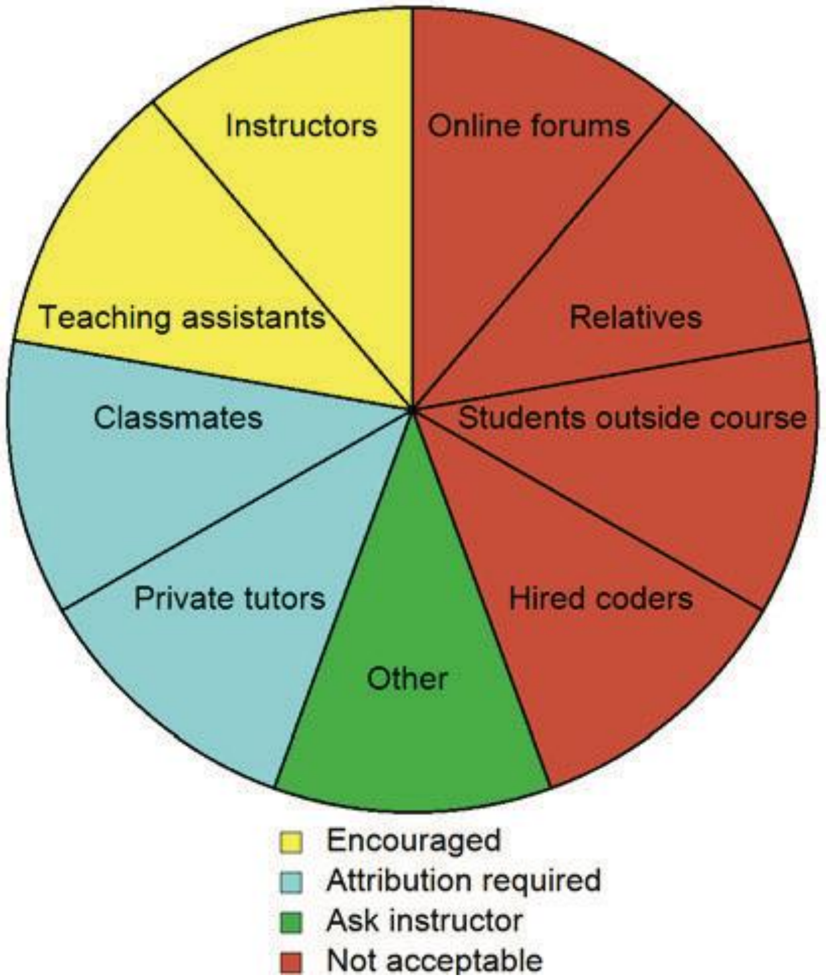
Academic Integrity

- The goals of this subject include fundamental programming concepts and skills. You need to develop these skills by completing the work and gaining the understanding yourself.
- The subject materials contain **all** you need. The reason you should *not* use online resources or other people to get assistance is because then you would not achieve the learning outcomes.

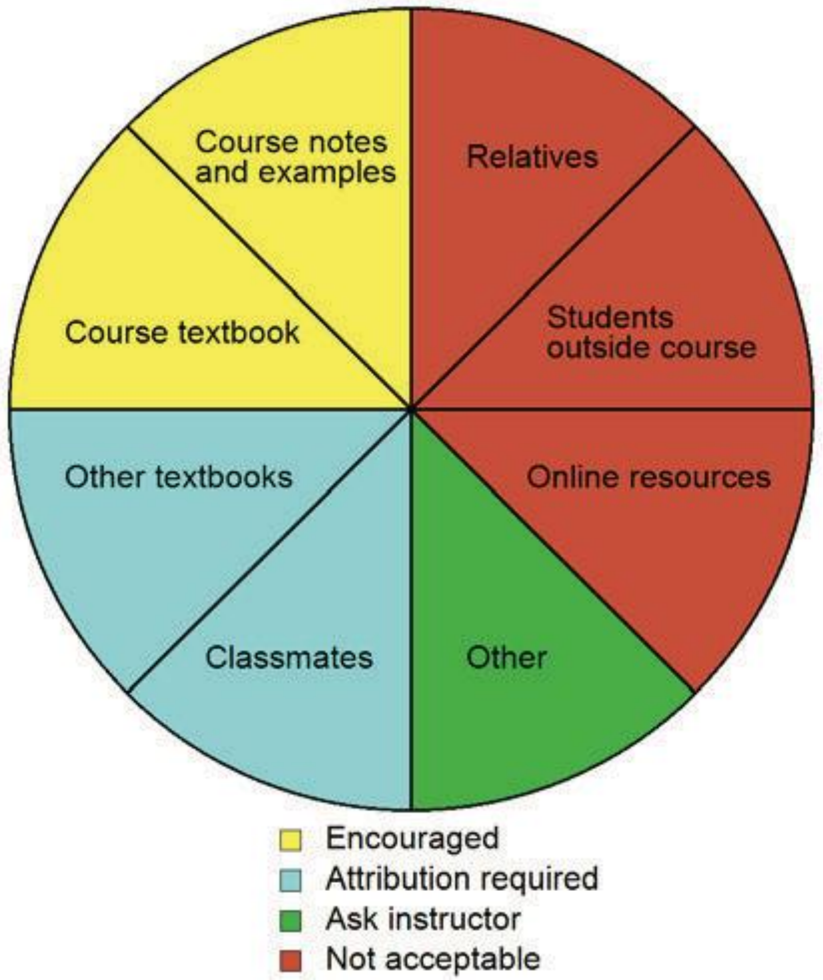
- Note that in higher-level subjects, the goals will be different (since you have learned the fundamentals), so your use of resources and assistance will change (e.g., write less code yourself)

Doing your own work helps you learn

Assistance:
Who can you get help from?

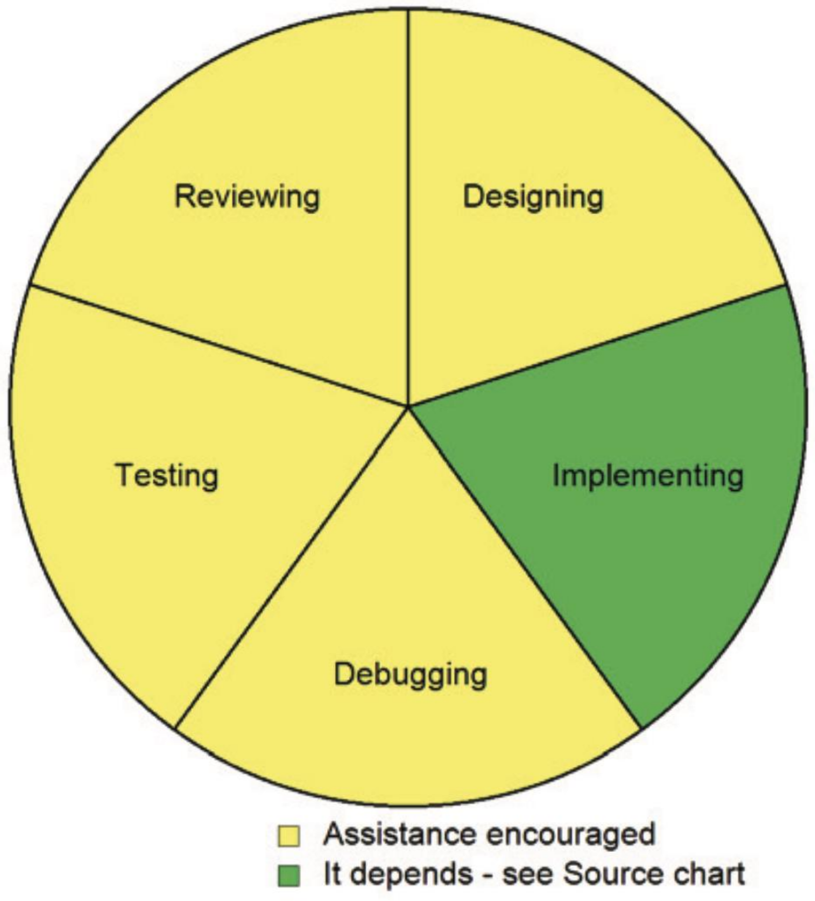


Resources:
Where can you get code from?

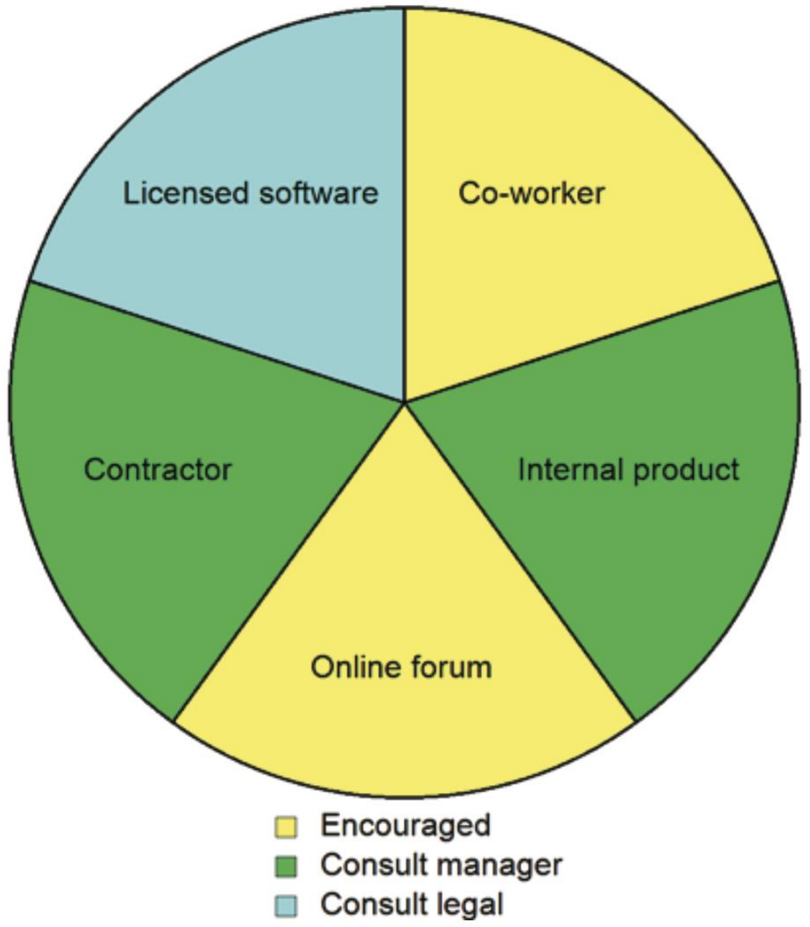


As you progress, the learning outcomes change and so do the guidelines for plagiarism

Assistance:
What task are you doing?



Source:
How are you getting assistance?



Follow these simple rules in our subject

- Never look at someone else’s assignment work
- Never let someone look at your assignment work

We care about style (and so do you)

- You are strongly encouraged to follow best practices and write good, readable code.
- This subject lays a foundation for your future programming development, so we will try to “get things right” from the beginning.
- Do *not* just focus on getting code working.
- It’s not OK to write code that works but has bad form and style.

IT@JCU

We care about style (and so do you)

- Python has recommended conventions (e.g., PEP8) and we will follow these as closely as we can, including naming conventions, commenting, formatting, etc.
- Assignment marking will include these conventions.
- It’s not just about getting it working – but learning to meet requirements.

- … this will make you a better programmer!

Let’s establish some foundations

- Hardware and Software
- How Computers Store Data
- How a Program Works
- Using Python

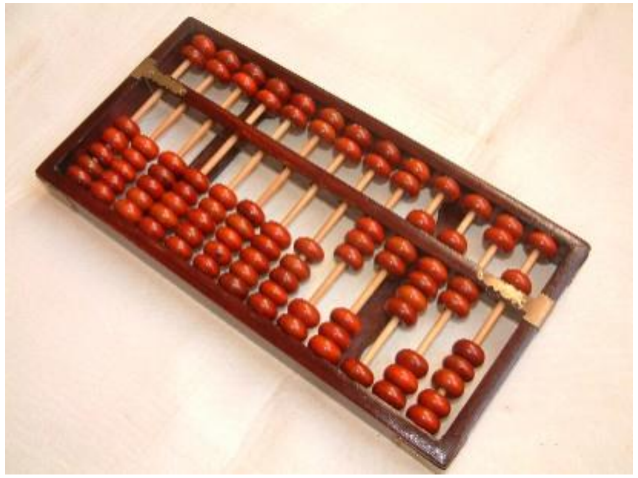
Programmers control computers with programs

- Computers can be programmed
  - Designed to do any job that a program tells them to
- <u>Program</u>: set of instructions that a computer follows to perform a task - *Software*
- <u>Programmer</u>: person who can design, create, and test computer programs - *Software Developer*
- **YOU are now a programmer!**

What is a computer?

Tally sticks (? BC)

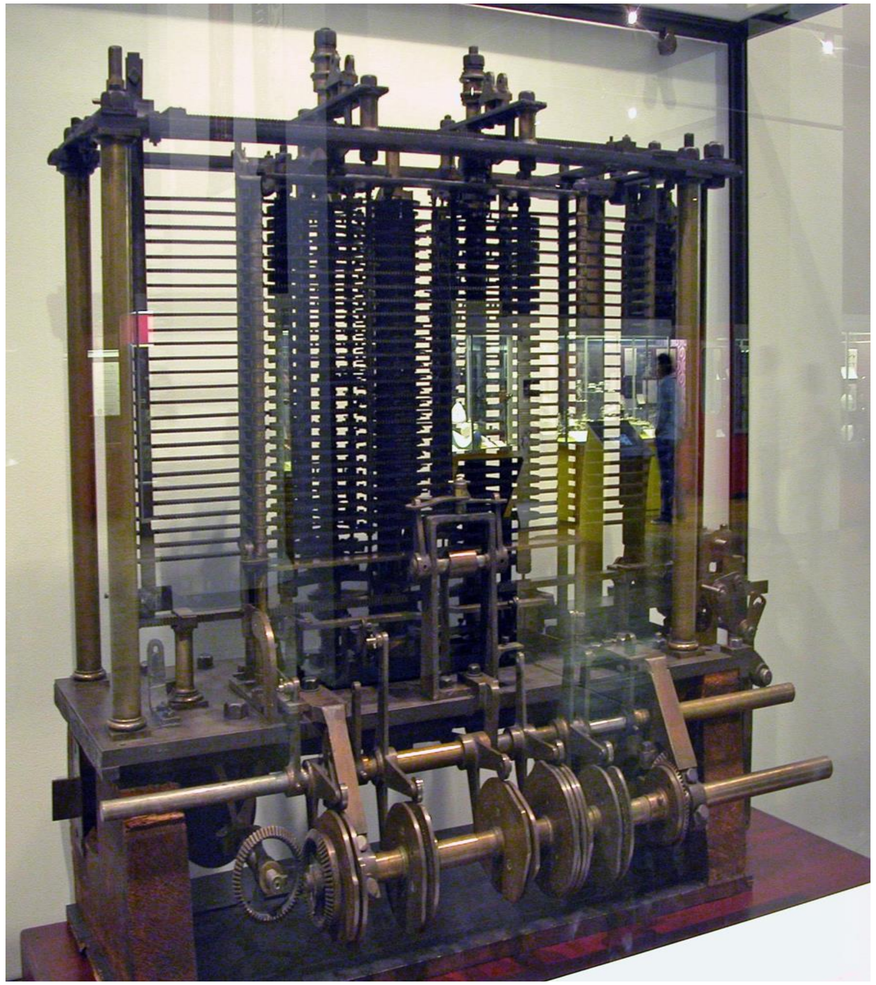
Abacus (~2500 BC)



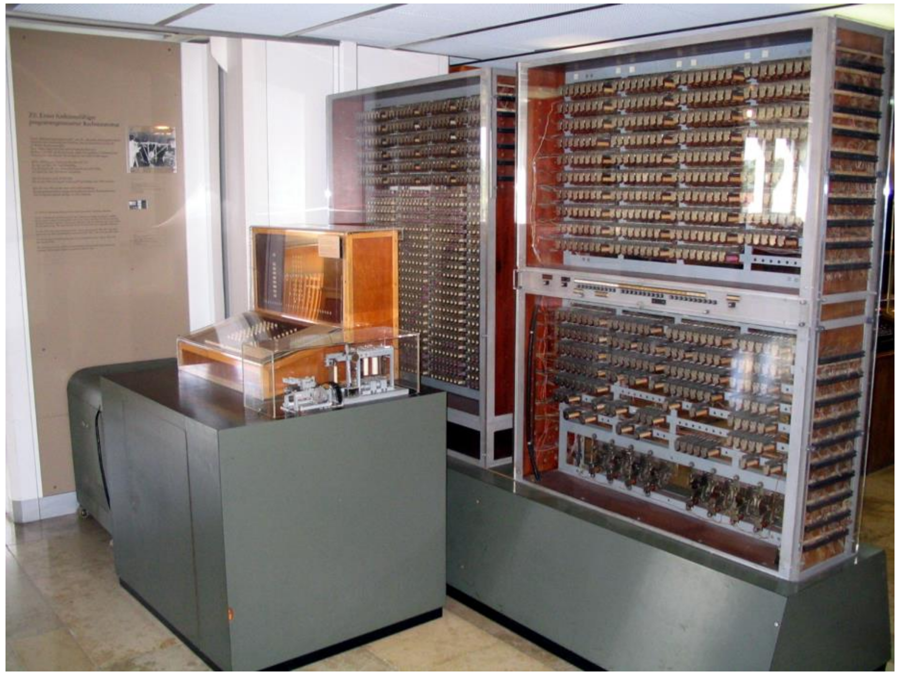
Slide Rule (~1625 AD)

What is a computer?

Analytical Engine (~1830s)

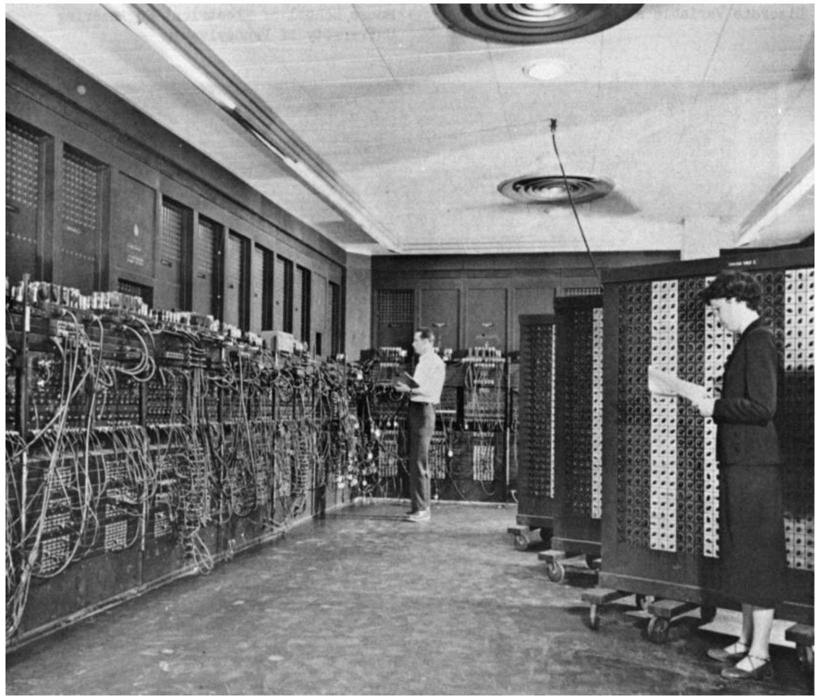


Zuse Z3 (1941)

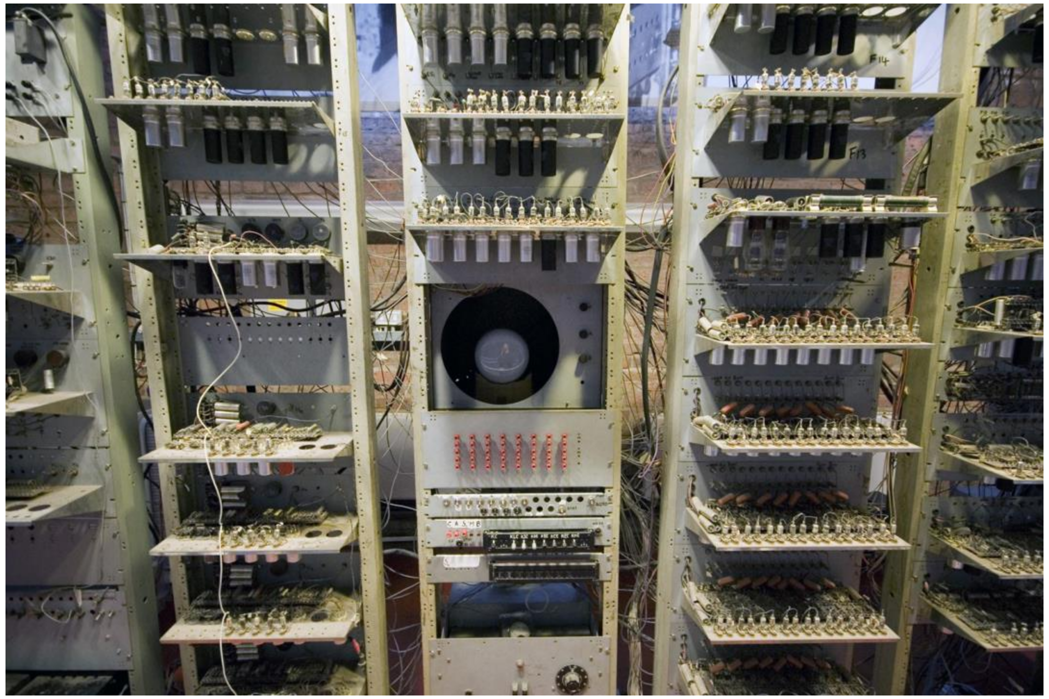


What is a computer?

ENIAC (~1946)



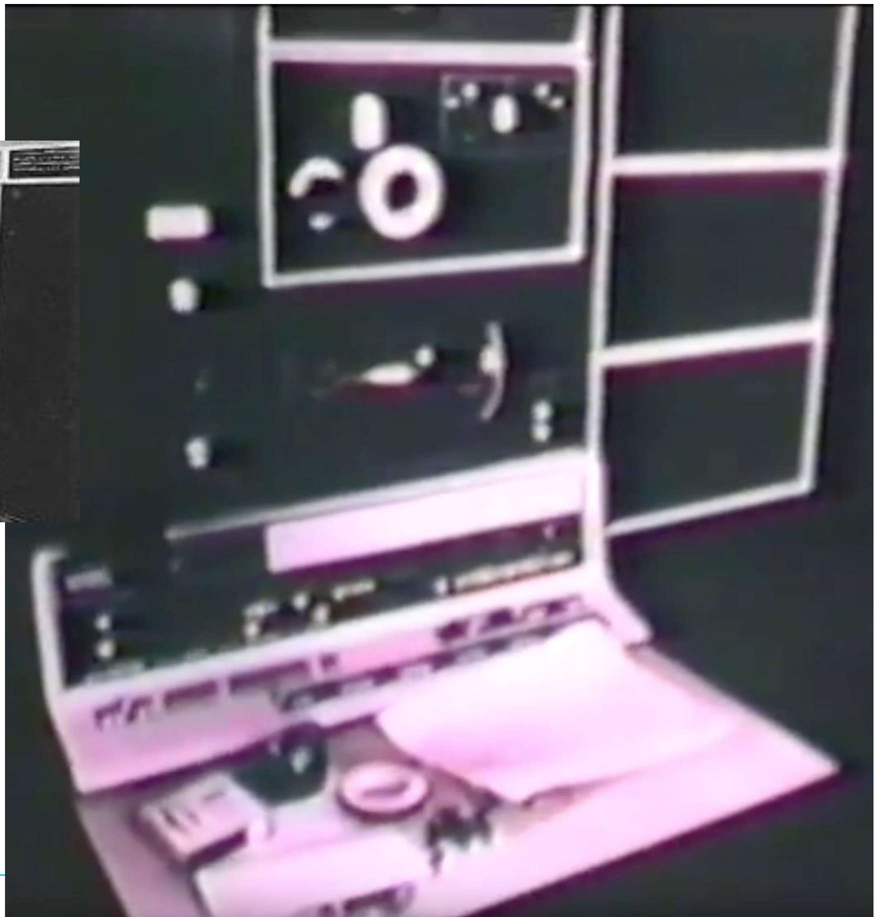
Baby (~1948)



IT@JCU

JCU had a state-of-the-art computer in 1970

James Cook University's first computer was as heavy as a house and used as much electricity as a suburb.
PDP 10 arrives at James Cook University - https://www.youtube.com/watch?v=_13TAKuazhM

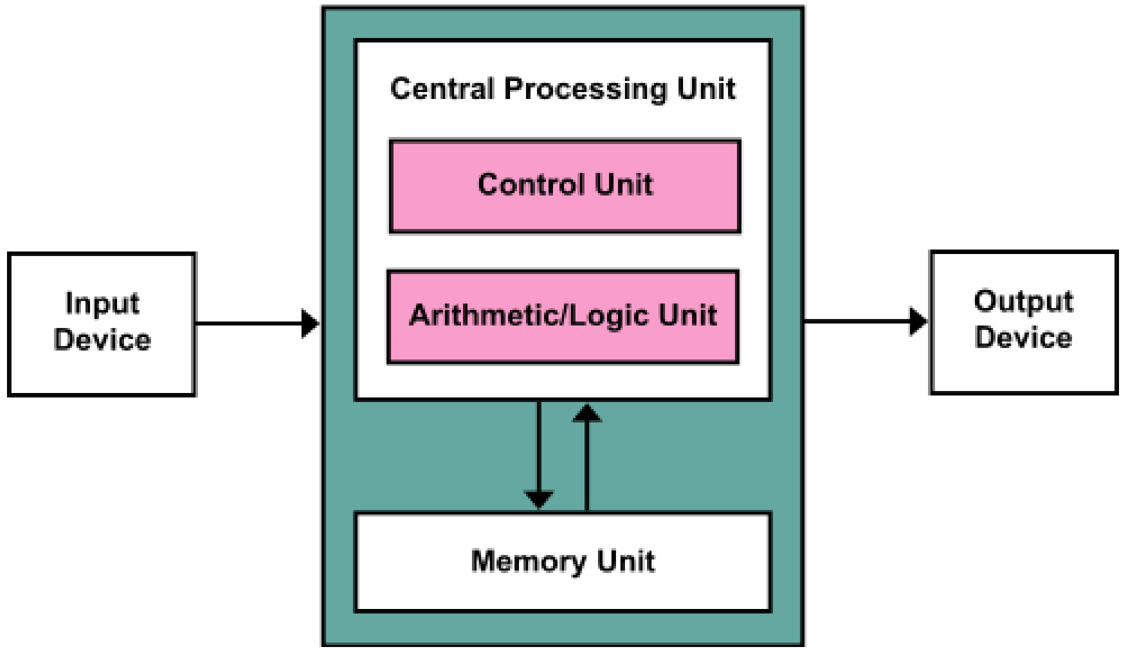


Room Te
Kitchen
Bathroom
Nursery
edroom
Smart Home
Room Lighting
Alarm system
Video Security
Room Temperature

jcu.edu.au

Hardware and Software

- Hardware: The physical devices that make up a computer
- Typical major components:
  - Central processing unit (CPU)
  - Main memory (RAM)
  - Secondary storage devices (Disk)
  - Input and output devices (Keyboard, mouse, touch… screen, audio…)

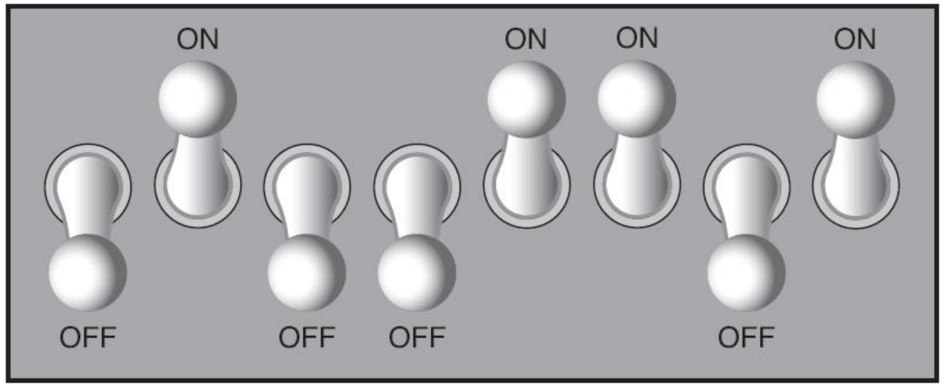


von Neumann architecture of a typical computer

Computers store all data in binary format

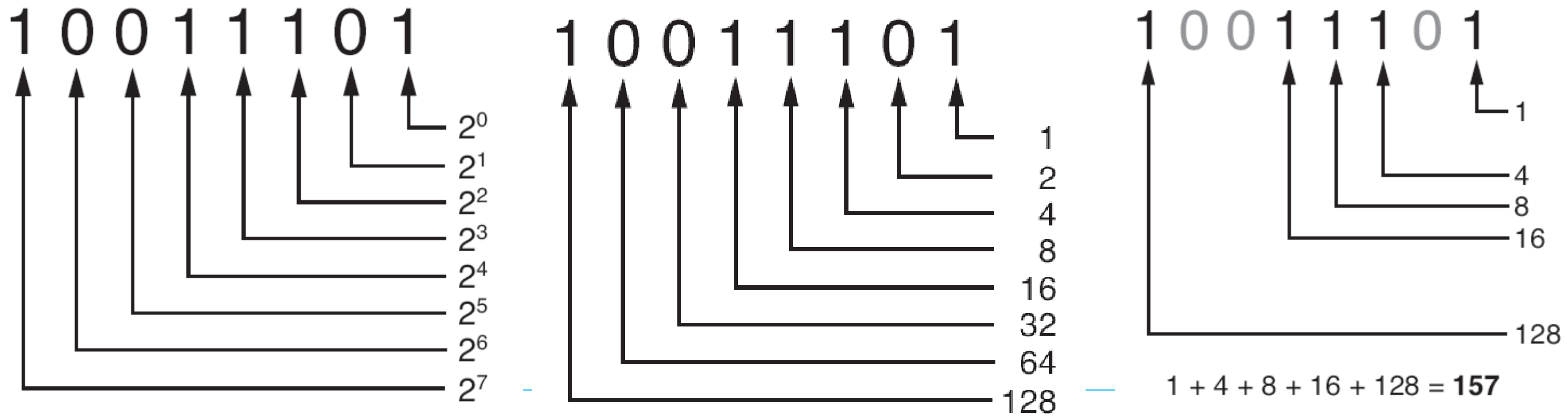
- All digital data is stored in sequences of 0s and 1s - **Binary**
- **Bit**: electrical component that can hold positive or negative charge, like on/off switch (1 or 0)
- **Byte**: 8 bits - just enough memory to store letter or small number
  - The on/off (1 or 0) pattern of bits in a byte represents data stored

Figure 1-7 Think of a byte as eight switches



Computers use the binary number system

- Position of digit j is assigned the value 2^{j-1}
- To determine the value of a binary number, sum the position values of the 1s
- Byte value range: 0 = all bits off; 255 = all bits on
  - To store larger numbers, use multiple bytes



Advanced Number Storage

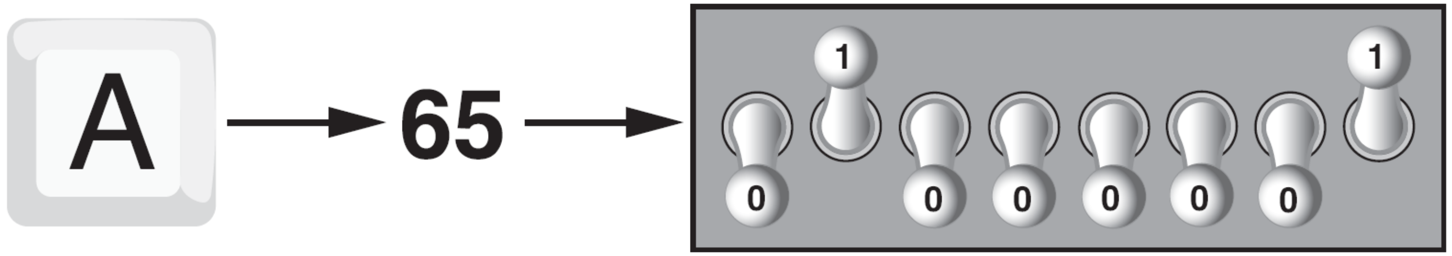
- To store negative numbers and real numbers, computers use binary numbering and encoding schemes
  - Negative numbers encoded using two’s complement
  - Real numbers encoded using floating-point notation

IT@JCU

Storing Characters

- Data stored in computer must be stored as binary number
- Characters are converted to numeric codes stored in memory
  - Most important coding scheme is ASCII
    - American Standard Code for Information Interchange
    - ASCII is limited: defines codes for only 128 characters
  - Unicode scheme is a more modern standard
    - Can represent characters from other languages

Figure 1-14 The letter A is stored in memory as the number 65



All types of data are stored in binary

- Digital: describes any device that stores data in binary
- Digital **images** are composed of pixels
  - each pixel is converted to a binary number representing the pixel's colour
- Digital **audio** consists of sections called samples
  - each sample is converted to a binary number
- Digital **video** consists of frames containing images

How do computer programs work?

- CPU is designed to perform simple operations on pieces of data
  - Understands instructions written in machine language and included in its instruction set
    - Each brand of CPU has its own instruction set
- To carry out meaningful calculation, CPU must perform many operations

Computers can perform 6 basic operations

- Receive information (input)
- Put out information (output)
- Perform arithmetic
- Assign a value to a variable (memory location)
- Compare two values and select 1 of 2 alternative actions
- Repeat a group of actions

A modern computer understands machine code

- Machine code is a set of primitive instructions for performing arithmetic and logic operations
- A program *can* be written as machine code

- The Control Unit executes machine code stored in memory using the fetch-decode-execute cycle:
  - **Fetch** the next instruction
  - **Decode** that instruction
  - **Execute** that instruction

What is a program?

- A series of instructions that a modern computer can execute
- A program allows you to tell a computer what to do
- We write programs that allow us to solve problems with computers

History of programming

- Some historians recognise Ada Lovelace's work in 1842 as the world's first computer program
- Herman Hollerith encoded data on punch cards (used for train tickets) around 1890
- Programming languages designed to communicate instructions to a modern electronic computer were written in the 1950s

- http://www.levenez.com/lang/ provides an interesting chart of the development of 50 programming languages

What is a programming language?

- It is hard to write even a simple program in machine code - humans don’t think like a computer!
- Programming languages were invented to make programming easier for humans
- There are many programming languages because different problems are easier to solve using different syntax and semantics
  - **R** is commonly used for statistics
  - **C** for embedded systems
  - **PHP** for website backends, **JavaScript** for web frontends
  - **Assembly language** for impressing others with your nerdiness

Python is a great language to learn and use

- General-purpose
- Easy to learn (compared to most other languages)
- Well-known (easy to find help)
- Lots of great libraries to extend its functionality
- Interpreted (can run programs line-by-line)

```python
name = input("Enter your name: ")
print("Hello", name)
print("Welcome to CP1401!")
```

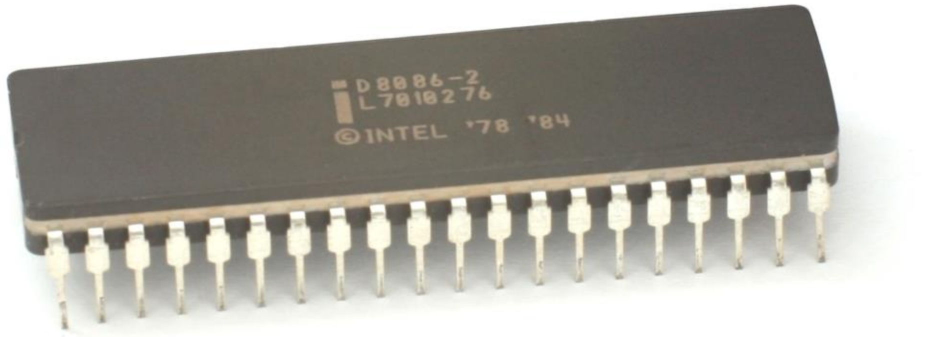
Programs are translated into machine code

- Programs written in high-level languages must be translated into machine language to be executed
- **Compiler**: translates high-level language program into separate machine language program
- **Interpreter**: translates and executes instructions in high-level language program
    - Used by Python language
    - Interprets/runs one instruction at a time
    - No separate machine language program

IT@JCU

Many computers use the Intel x86 chip architecture

8086: ~1978



Haswell: ~2013

The x86 machine code is complex...

Code	**Meaning**
ADC	add with carry
ADD	add
AND	logical and
CALL	function call pushing program counter onto the stack
DEC	decrease by 1
DIV	unsigned divide
HLT	stop the computer
hundreds more...	

Python compiles to something called bytecode
(closer to machine code)

Python code	Python bytecode

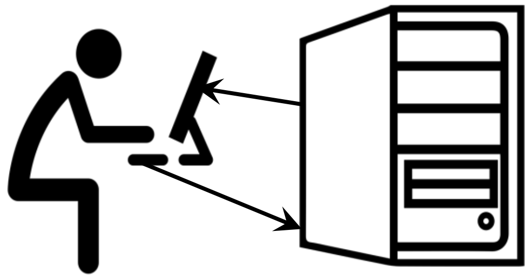
```
value = 1
value = value + 1
print(value)
```
	```
	1           0 LOAD_CONST          0 (1)
	            3 STORE_NAME          0 (value)

	2           6 LOAD_NAME           0 (value)
	            9 LOAD_CONST          0 (1)
	           12 BINARY_ADD
	           13 STORE_NAME          0 (value)

	3          16 LOAD_NAME           1 (print)
	           19 LOAD_NAME           0 (value)
	           22 CALL_FUNCTION       1 (1 positional, 0 keyword pair)
	           25 POP_TOP
	           26 LOAD_CONST          1 (None)
	           29 RETURN_VALUE
	```

Python is translated into bytecode before it is executed on a real computer.

IT@JCU

A typical program needs to: get input, process data, and output results

- A program gets input from a variety of input devices
  - Keyboard, mouse, touch screen, sensors, ...
- It then processes the inputs as data in various ways
- Results are then formatted and sent to various output devices
  - Screen, printer, sent across the Internet, ...



However, the reality is non-trivial

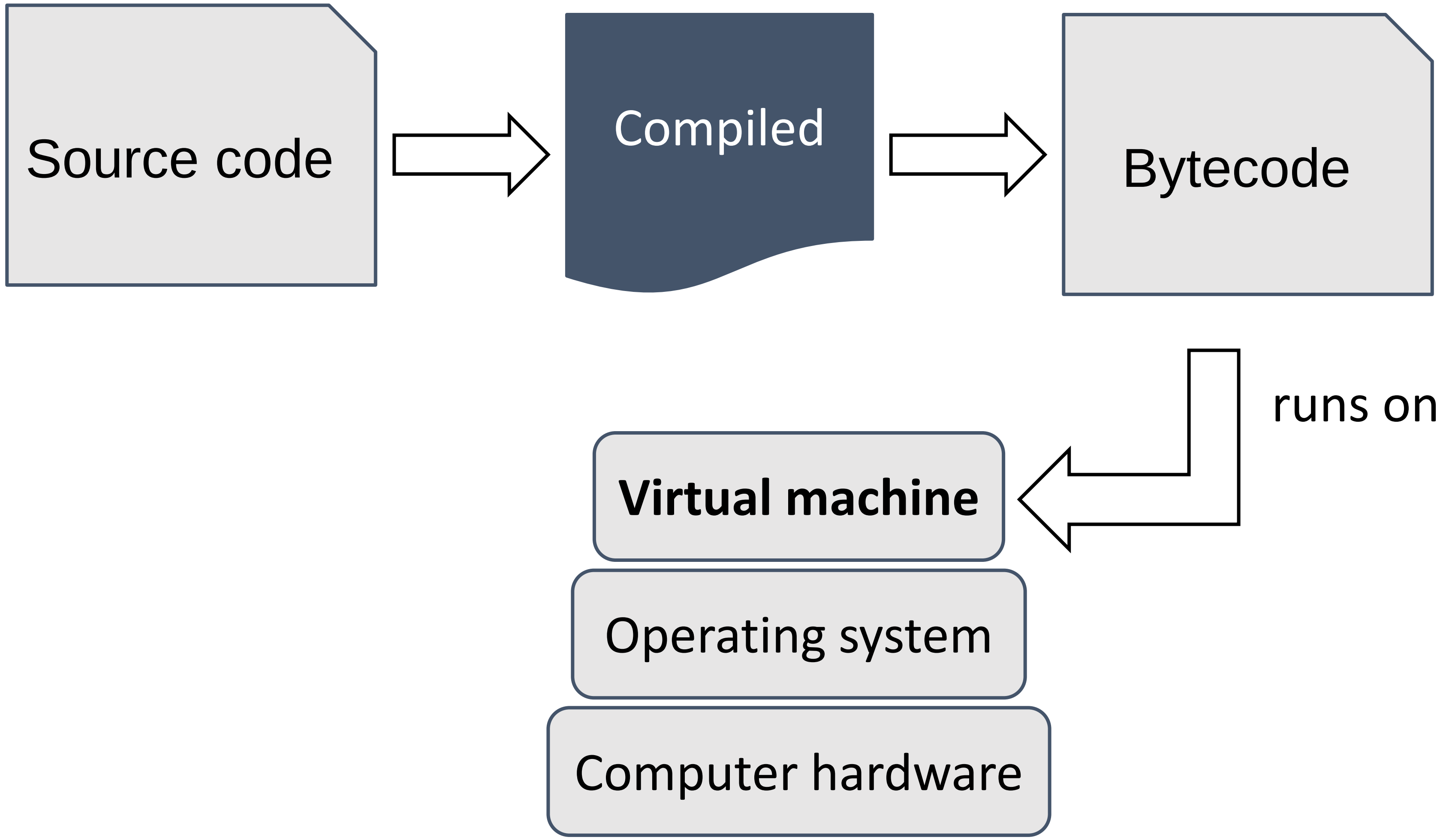
- You have to rewrite or **compile** your program to suit each type of Operating System and computer hardware - not very practical!

MyProgram
Mac version

MyProgram
PC version

MyProgram
Sun version

One solution: software runtime environment, called a virtual machine that runs platform-independent bytecode

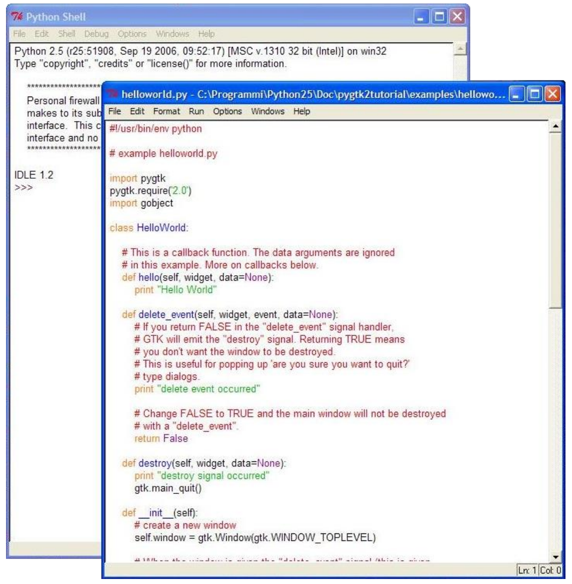


Using Python

- Python must be installed and configured prior to use
  - One of the items installed is the Python interpreter
- Python interpreter can be used in two modes:
  - Interactive mode: enter statements in the console
  - Script mode: save statements in a text file called a Python script

IDLE comes with Python

- IDLE is an editor for writing and running Python programs
  - Automatically installed when Python is installed
  - Runs in interactive mode or script mode
  - Text editor with very basic features (not for us)

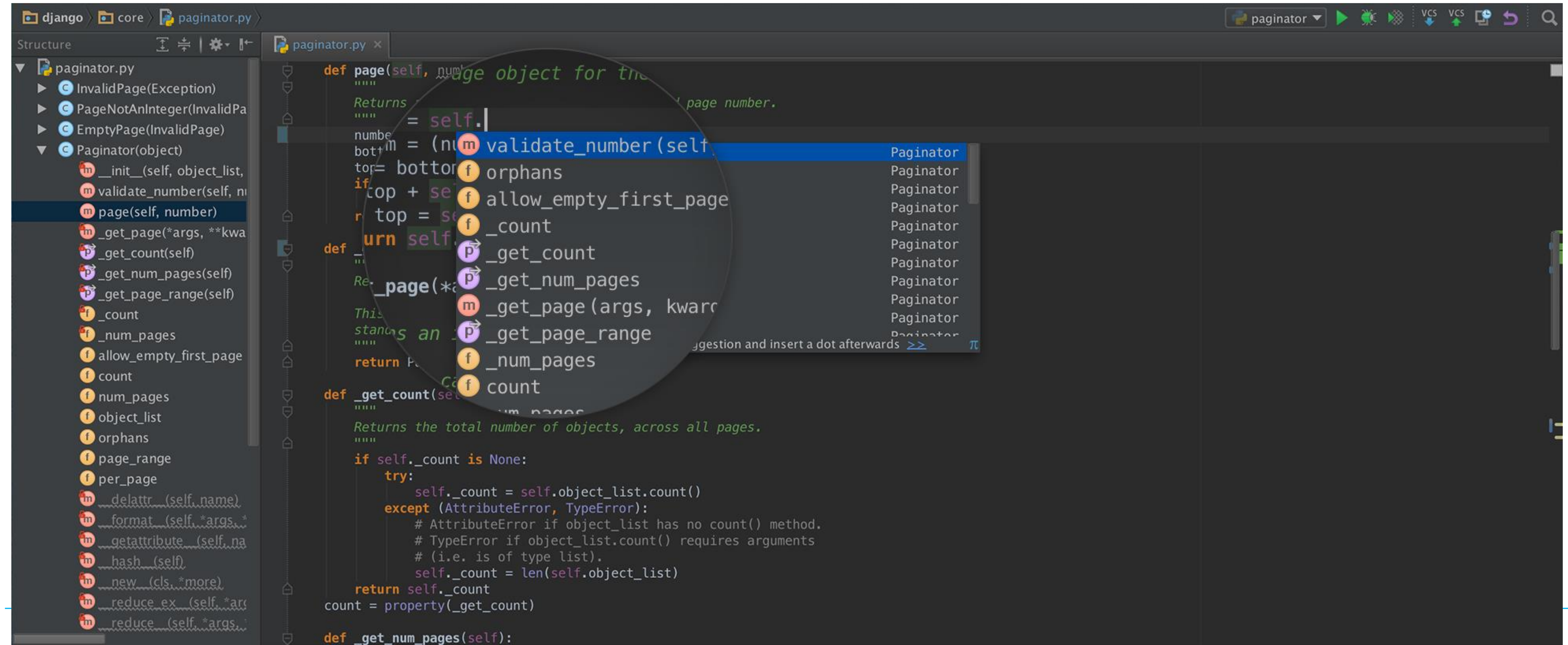

Python Shell
File Edit Shell Debug Options Windows Help
helloworld.py - C:\Programmi\Python25\Doc\pygtk2tutorial\examples\hellowo...
File Edit Format Run Options Windows Help
Ln: 1 Col: 0

```
Python 2.5 (r25:51908, Sep 19 2006, 09:52:17) [MSC v.1310 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

    ********************
    Personal firewall
    makes to its sub
    interface.  This c
    interface and no
    ********************

IDLE 1.2
>>>
```

```
#!/usr/bin/env python

# example helloworld.py

import pygtk
pygtk.require('2.0')
import gobject

class HelloWorld:

    # This is a callback function. The data arguments are ignored
    # in this example. More on callbacks below.
    def hello(self, widget, data=None):
        print "Hello World"

    def delete_event(self, widget, event, data=None):
        # If you return FALSE in the "delete_event" signal handler,
        # GTK will emit the "destroy" signal. Returning TRUE means
        # you don't want the window to be destroyed.
        # This is useful for popping up 'are you sure you want to quit?'
        # type dialogs.
        print "delete event occurred"

        # Change FALSE to TRUE and the main window will not be destroyed
        # with a "delete_event".
        return False

    def destroy(self, widget, data=None):
        print "destroy signal occurred"
        gtk.main_quit()

    def __init__(self):
        # create a new window
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
```

We like PyCharm

- PyCharm is a full-featured Integrated Development Environment (IDE) used by many professionals (and us).
  - Many helpful time-saving features
  - Helps you learn to program in Python with relevant code highlighting



```
    def _get_count(self):
        """
        Returns the total number of objects, across all pages.
        """
        if self._count is None:
            try:
                self._count = self.object_list.count()
            except (AttributeError, TypeError):
                # AttributeError if object_list has no count() method.
                # TypeError if object_list.count() requires arguments
                # (i.e. is of type list).
                self._count = len(self.object_list)
        return self._count
    count = property(_get_count)

    def _get_num_pages(self):
```

IT@JCU

Begin with the end in mind…

Quick demo of a moderate-sized Python program

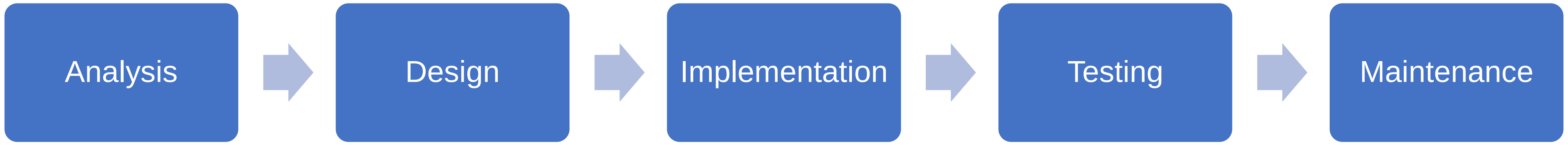
- nearly as big as our second assignment
- input, output, processing, selection, repetition, functions

https://github.com/CP1401/subject/blob/master/cateringcalculator.py

Problem Solving

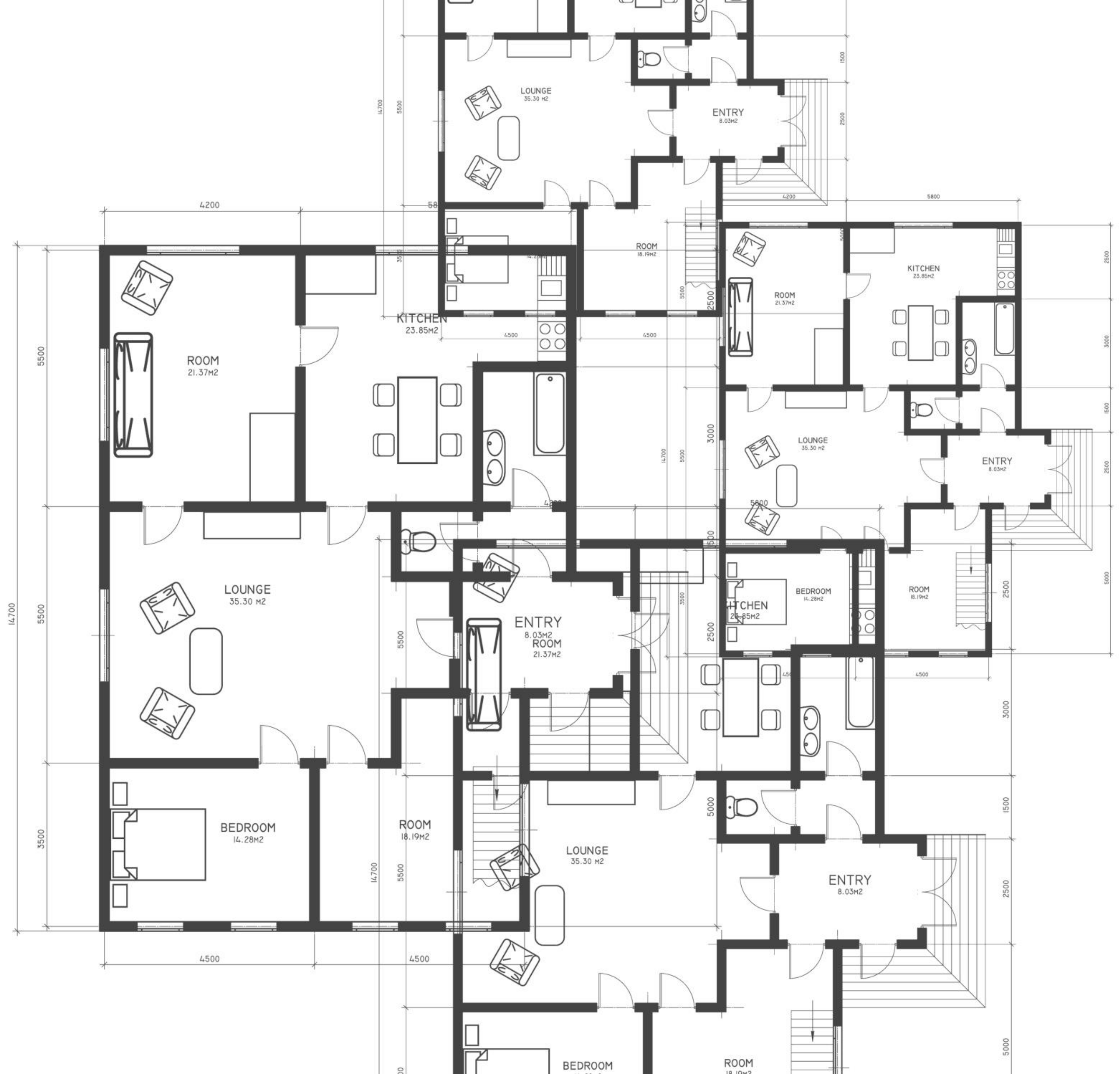
Designing a Program

- Programs must be designed (planned) before they are written
- There are many variations of the development process, but they all include the same steps in different names and groups.



Designing a Program

- **Analysis** is describing **what** a program will do
- **Design** is describing **how** the program will do it
- This is the most important part of the development cycle
- Understand the task that the program is to perform
    - Work with the customer to get a sense of what the program is meant to do
    - Ask questions about program details
    - Create one or more software requirements


ROOM
21.37M2
KITCHEN
23.85M2
LOUNGE
35.30 M2
ENTRY
8.03M2
BEDROOM
14.28M2
ROOM
18.19M2
4200
4500
5500
3500
14700
5800
5000
3000
2500
1500

Designing a Program

- Determine the steps that must be taken to perform the task
  - Break down required tasks into a series of steps - **decomposition**
  - Create an algorithm, listing logical steps that must be taken
- **Algorithm**: set of well-defined logical steps that must be taken to perform a task
  - Cooking recipes, directions and instructions for almost anything are all algorithms (rinse and repeat…)

Problem decomposition

Description:

- We need to know how much a box can hold, assuming that we have been given its dimensions. We have been given the length, width and depth of the box, and we need to calculate the volume of the box.

- What are the **nouns** or ‘things’ here?
- What are the **verbs** or ‘things to do’ here?

Problem decomposition

- We need to know how much a box can hold, assuming that we have been given its dimensions. We have been given the length, width and depth of the box, and we need to calculate the volume of the box.
- Things to do: '**calculate the volume**'
- Things (to work with): **length, width, depth, volume**
- So we need to calculate volume from length width and depth

```
volume = length * width * height
```

How can we communicate a solution?

- Option One – Wall of text
  - Get the length, width and depth of the box. Once that is done then calculate the volume of the box by multiplying the length by the width by the depth. Then display the volume of the box.

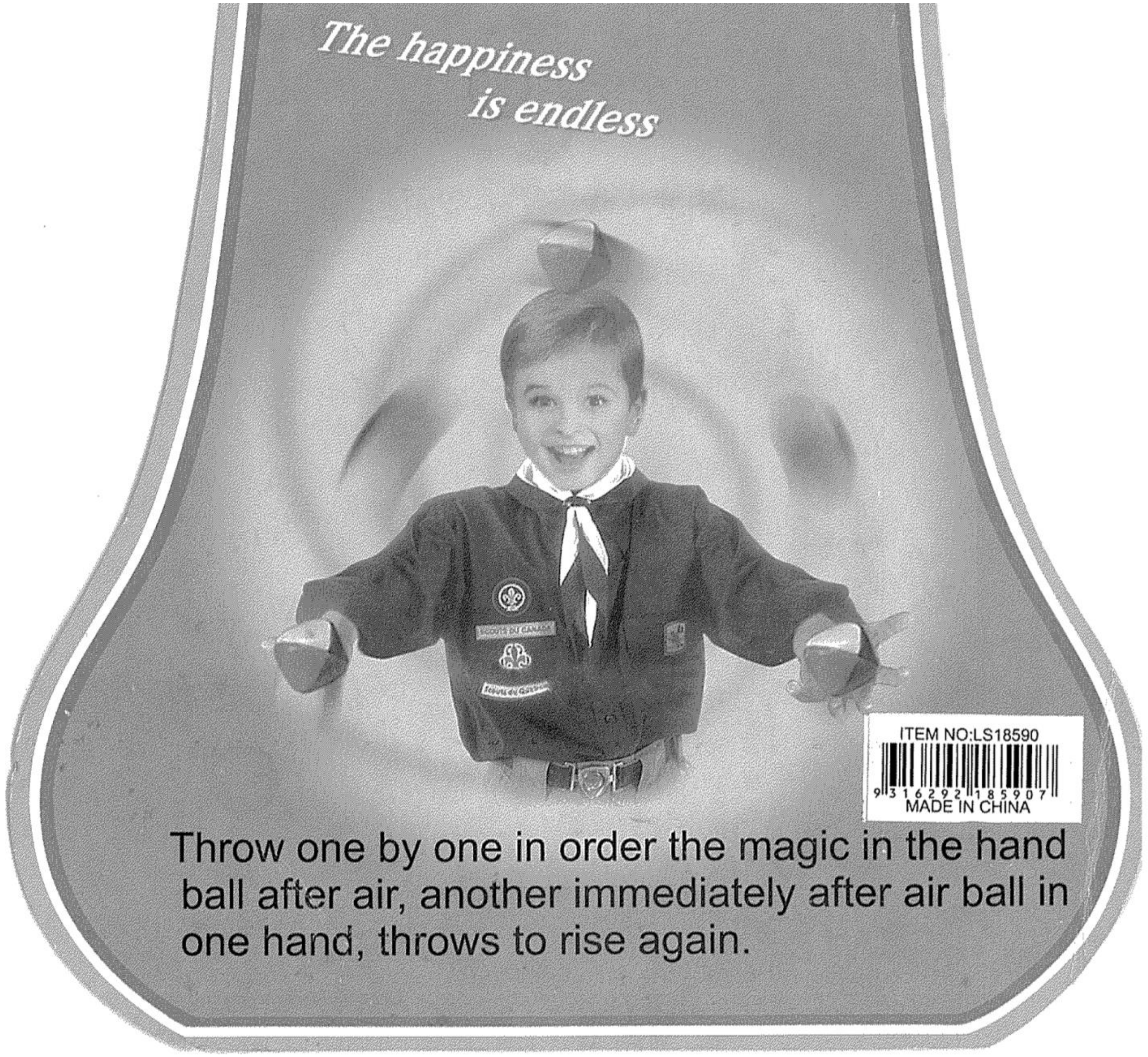
- We can do better…

Algorithm

```
get length, width, depth
volume = length * width * depth
display volume
```

- The code to implement this algorithm will depend on the programming language chosen

Grab 3 balls and try following this algorithm now...



Throw one by one in order the magic in the hand ball after air, another immediately after air ball in one hand, throws to rise again.

Pseudocode is “fake” (“sort of”) code for algorithms

- Informal language that has no strict syntax rules
- Not meant to be compiled or executed
- Used by people to create model programs
    - Developer can focus on program’s design, not syntax
    - Can be translated directly into actual code in any programming language
- But there are some strong guidelines:
    - Indenting is required to show structure
    - Be consistent with names/terms
    - Must be a 'complete' solution – no more thinking for programmer
    - See our Guide to Good Pseudocode

What does this pseudocode do?

```
total = 0
count = 0
get age
while age >= 0
    total = total + age
    count = count + 1
    get age
display total
display total / count
```

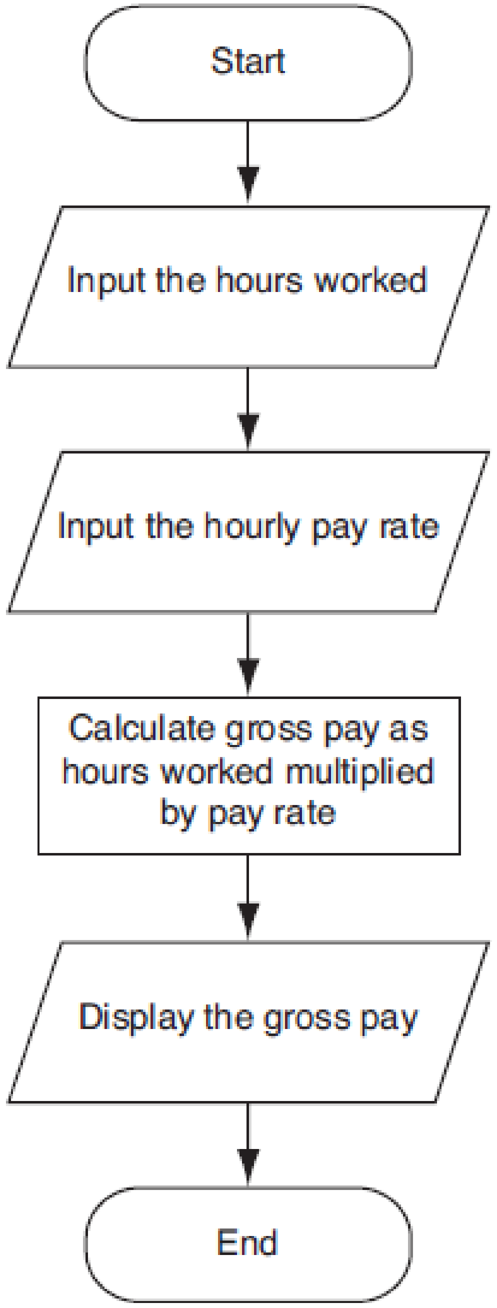
Pseudocode is (should be) easy to read & understand, and easy to write (just use the words and style you see in our guide and examples).

Flowcharts help describe algorithms visually

- Flowchart: diagram that graphically depicts the steps in a program
  - Ovals are terminal symbols
  - Parallelograms are input and output symbols
  - Rectangles are processing symbols
  - Symbols are connected by arrows that represent the flow of the program

Step 1
Step 2
Step 3

Figure 2-2 Flowchart for the pay calculating program


Start
Input the hours worked
Input the hourly pay rate
Calculate gross pay as hours worked multiplied by pay rate
Display the gross pay
End

Looking ahead... more complex flowcharts

Figure 3-1 A simple decision structure

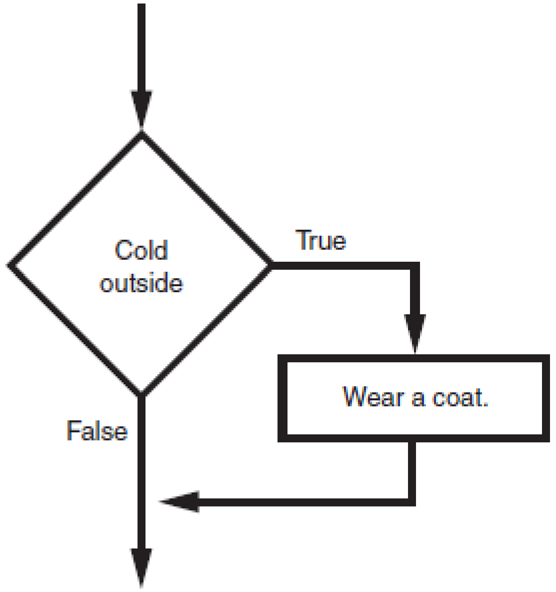
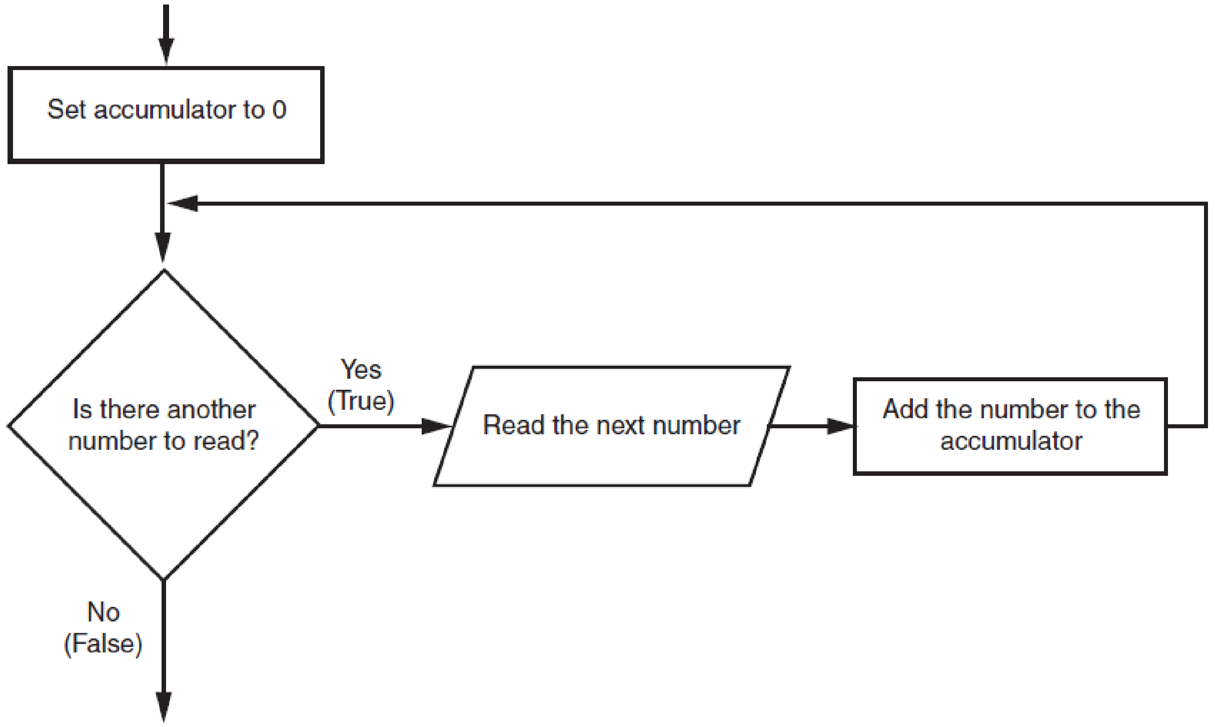


Figure 4-6 Logic for calculating a running total


Set accumulator to 0
Is there another number to read?
Yes (True)
Read the next number
Add the number to the accumulator
No (False)

Do this now

Draw a flowchart for how to wash your hair

Now do these next steps

- Start thinking of things you do in terms of algorithms – clear steps that someone could follow
- Install Python 3 and PyCharm: https://github.com/CP1404/Starter/wiki/Software-Setup
- Read chapter 2 of your textbook