NCP Computer Science

- The learning curve from Year 11 to Year 12 is a big one.
- These resources are there to help bridge the gap and also to help you to improve some of the independent study skills you will need to develop over time.
- Bring the completed workbook to your first lesson at the start of Y12.
- This is a great opportunity for you to show your skills and interests.
- Please read each slide carefully, and enjoy!

1. Who are you?

Compulsory – must do!

Maths grade	
English grade	
Computer Science grade	
Average point score (if known)	

In this task you get to tell me a little bit about yourself. Who are you and what do you enjoy about computer science?

Name:

1. What are your main interests in school? What subjects and clubs do you enjoy and why?

2. What are your interests outside of school?

3. Why did you choose Computer Science?

4. What are your plans beyond New College? What would like to get out of studying Computer Science?

Expected completion time: 10-30 mins

2. The course

Optional – strongly recommended!

Exercise: Research the following

How is the OCR A level Computer Science qualification structured?

1a. What is the name of the first paper?

1b. How long is the exam?

1c. How much of the course is it worth?

2a. What is the name of the second paper?

2b. How long is the exam?

2c. How much of the course is it worth?

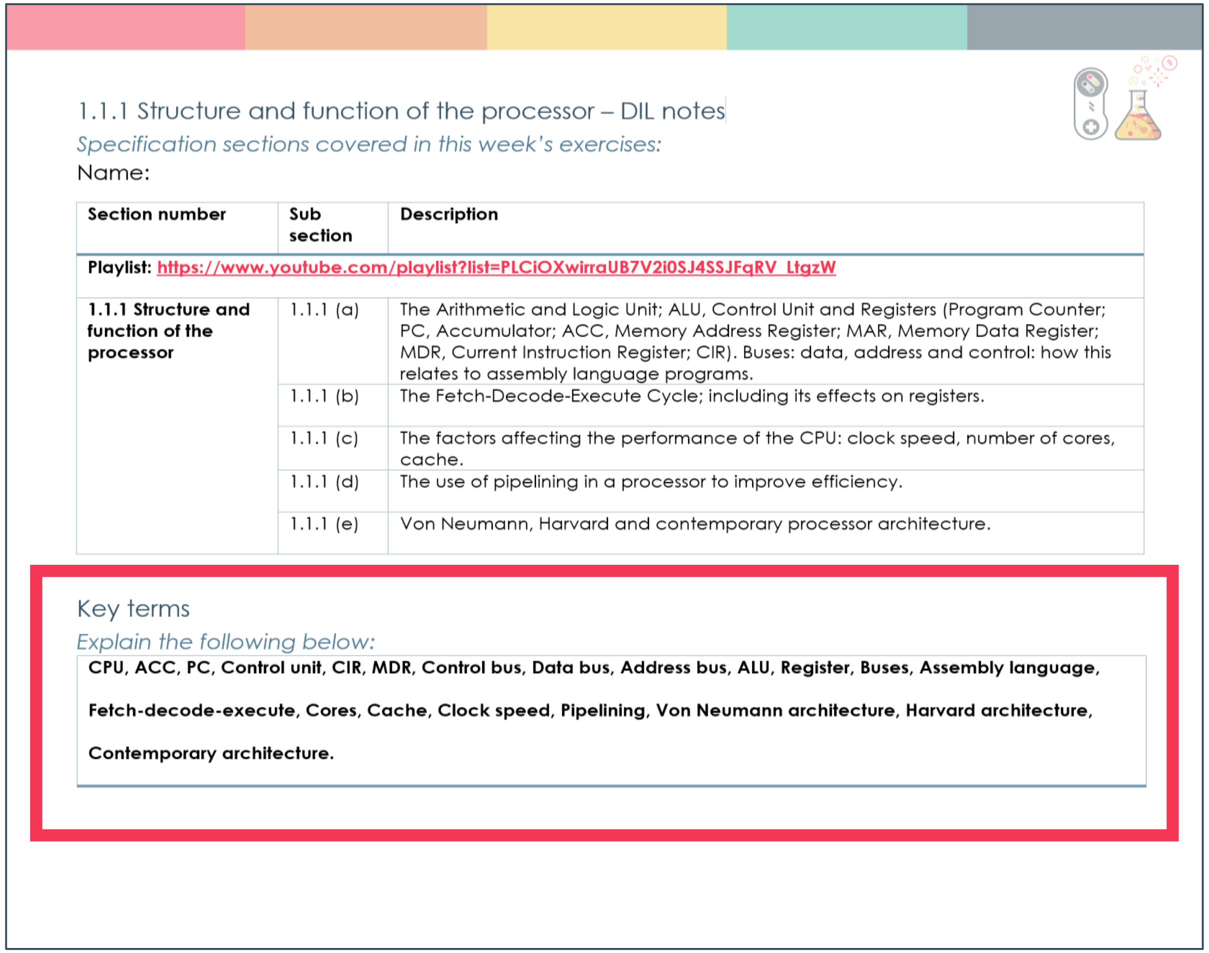
3a. What is the final part of the course?

c. How much of the course is it worth?

3. Flipped learning

Compulsory – must do!

At New College you are expected to preview the following weeks learning. We do this using a method of note taking called Cornell notes, we will look at them on the following pages. On the first page of each Cornell note booklet we begin by listing key terms for a topic, and your task is to find a definition for each key term, as shown below.



1.1.1 Structure and function of the processor – DIL notes

Specification sections covered in this week's exercises:

Name:

Section number	Sub section	Description
Playlist: https://www.youtube.com/playlist?list=PLCiOXwirraUB7V2i0SJ4SSJFqRV_LtgzW		
1.1.1 Structure and function of the processor	1.1.1 (a)	The Arithmetic and Logic Unit; ALU, Control Unit and Registers (Program Counter; PC, Accumulator; ACC, Memory Address Register; MAR, Memory Data Register; MDR, Current Instruction Register; CIR). Buses: data, address and control: how this relates to assembly language programs.
	1.1.1 (b)	The Fetch-Decode-Execute Cycle; including its effects on registers.
	1.1.1 (c)	The factors affecting the performance of the CPU: clock speed, number of cores, cache.
	1.1.1 (d)	The use of pipelining in a processor to improve efficiency.
	1.1.1 (e)	Von Neumann, Harvard and contemporary processor architecture.

Key terms

Explain the following below:

CPU, ACC, PC, Control unit, CIR, MDR, Control bus, Data bus, Address bus, ALU, Register, Buses, Assembly language,

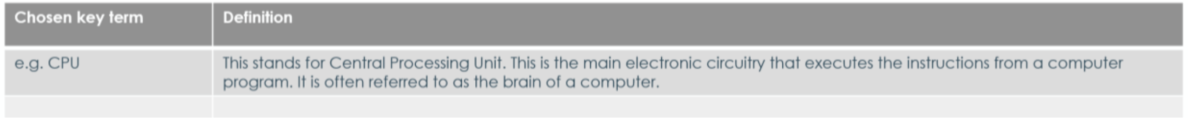
Fetch-decode-execute, Cores, Cache, Clock speed, Pipelining, Von Neumann architecture, Harvard architecture,

Contemporary architecture.

On the following slide, pick 10 of the key terms highlighted (apart from CPU) to the left, and find a definition for what they do.

If the word is an acronym like CPU, do not simply write what the letters stand for, also write what a CPU does.

For example:



Chosen key term	Definition
e.g. CPU	This stands for Central Processing Unit. This is the main electronic circuitry that executes the instructions from a computer program. It is often referred to as the brain of a computer.

Compulsory – must do!

3. Flipped learning

Exercise: Write your chosen ten key terms and their definition below.

Chosen key term	Definition
e.g. CPU	This stands for Central Processing Unit. This is the main electronic circuitry that executes the instructions from a computer program. It is often referred to as the brain of a computer.
1.	
2.	
3.	
4.	
5.	
6.	
7.	
8.	
9.	
10.	

Expected completion time: 30 mins

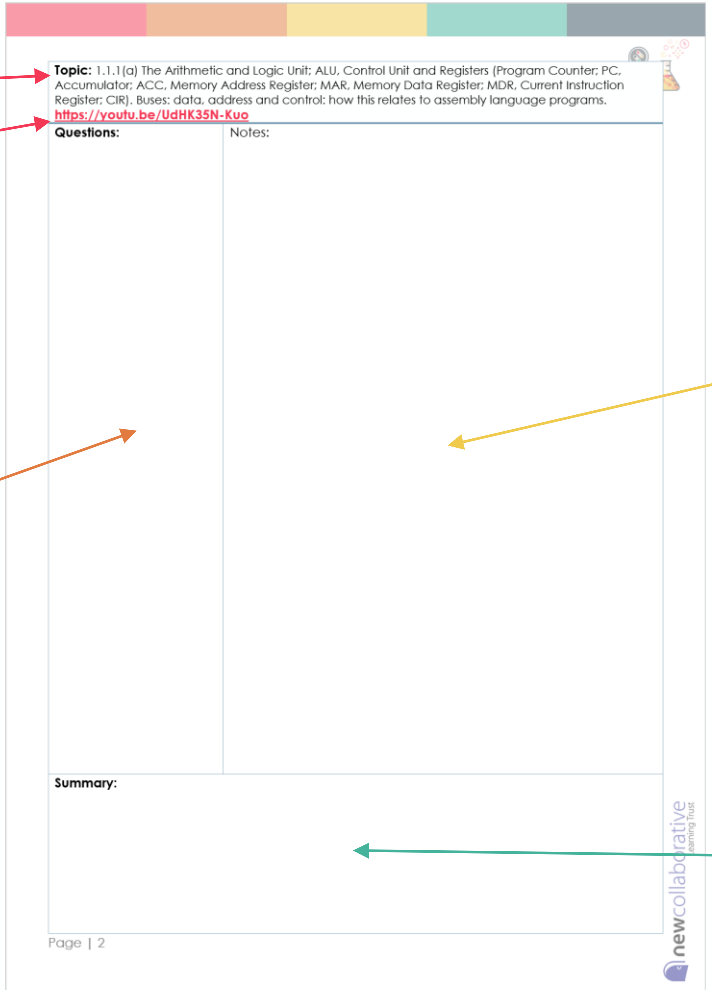
3. Flipped learning

Compulsory – must do!

Cornell note taking

Once you have completed the key terms, then you will have to complete the Cornell notes for each specification point using online videos. The structure of a Cornell note page is shown below.

1. The point from the specification document that this page links to.
2. The link to the YouTube video that you need to make notes about.



Topic: 1.1.1(a) The Arithmetic and Logic Unit; ALU, Control Unit and Registers (Program Counter; PC, Accumulator; ACC, Memory Address Register; MAR, Memory Data Register; MDR, Current Instruction Register; CIR). Buses: data, address and control: how this relates to assembly language programs.
https://youtu.be/UdHK35N-Kuo

Questions:

Notes:

Summary:

Page | 2

3. The main notes section should include information about the theory discussed in the video. They should be **hand written in pen** or **hand written electronically**, ***never typed***. We need to replicate what you will be doing in the exam from day one.

4. Any diagrams shown should also be included, to refer back to when revising.

5. *If you need more space you should duplicate the page, or make the notes section larger so you go onto two pages, don't think you have to stick to the page given.*

6. The questions section should include questions that you think you could be asked about in the exam.

7. It should also include questions that you have for your teacher to clarify.

8. It is good practice to use the 'Content clarification guide' to help you write your questions. More about this in September.

9. The summary should be a short explanation of what this page is about. It is usually 5 – 6 sentences long. Writing short, to the point explanations is a key skill for the exam.

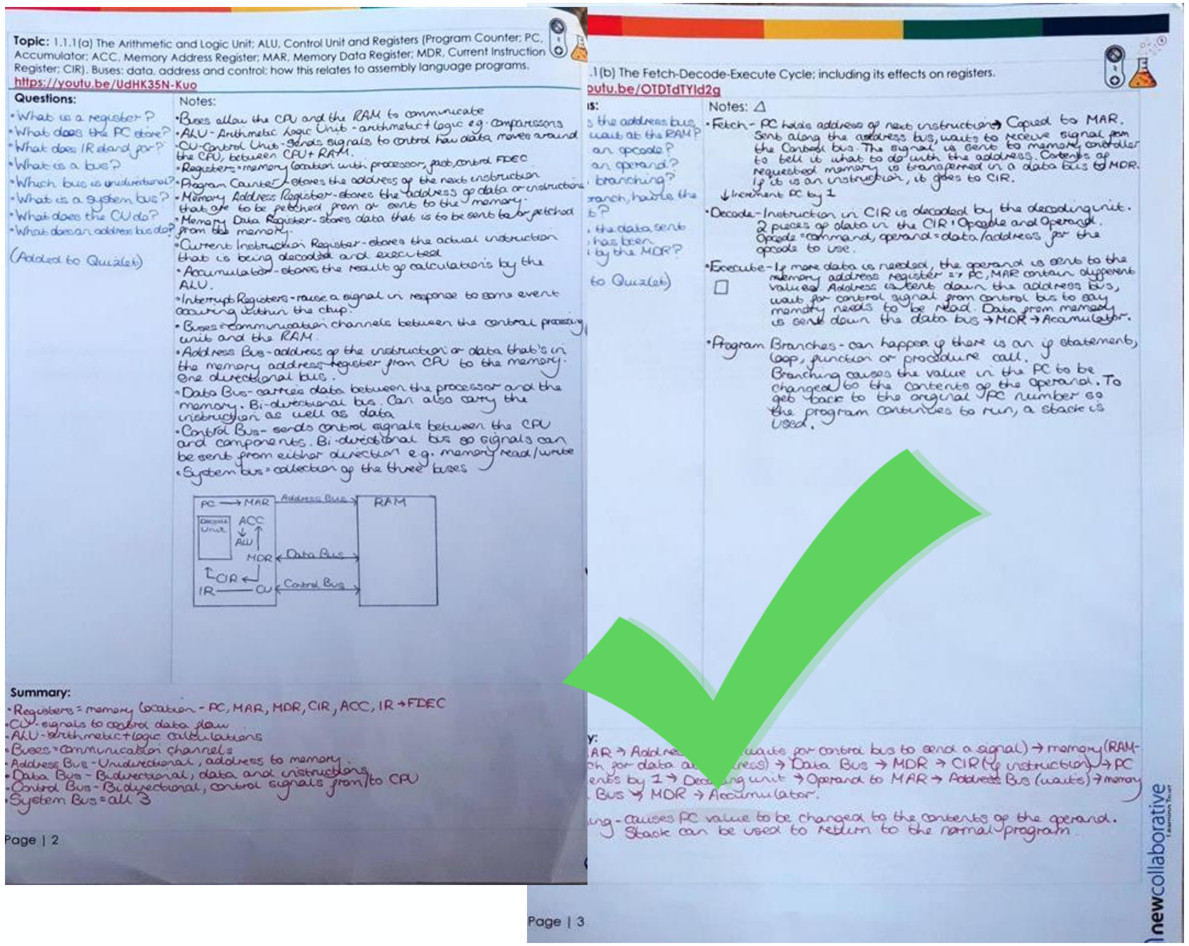
How to take Cornell notes

Compulsory – must do!

4. Flipped learning

Cornell note taking

To prepare you for the exam from day one – we replicate the exam, so you can print and hand write, you can hand write electronically if you have a tablet or 2 in 1. Copying and pasting doesn't involve any processing, so typed and scanned Cornell Notes are expected!



Topic: 1.1.1(a) The Arithmetic and Logic Unit; ALU, Control Unit and Registers (Program Counter; PC, Accumulator; ACC, Memory Address Register; MAR, Memory Data Register; MDR, Current Instruction Register; CIR). Buses: data, address and control: how this relates to assembly language programs.
https://youtu.be/UdHK35N-Kuo

Questions:

- What is a register?
- What does the PC store?
- What does IR stand for?
- What is a bus?
- Which bus is unidirectional?
- What is a system bus?
- What does the CU do?
- What does an address bus do?

(Added to Quizlet)

Notes:

- Buses allow the CPU and the RAM to communicate
- ALU - Arithmetic Logic Unit - arithmetic + logic e.g. comparissons
- CU - Control Unit - sends signals to control how data moves around the CPU, between CPU + RAM.
- Registers = memory location with processor, fast, control FDEC
- Program Counter - stores the address of the next instruction
- Memory Address Register - stores the address of data or instructions that are to be fetched from or sent to the memory.
- Memory Data Register - stores data that is to be sent to or fetched from the memory.
- Current Instruction Register - stores the actual instruction that is being decoded and executed
- Accumulator - stores the result of calculations by the ALU.
- Interrupt Registers - raise a signal in response to some event occuring within the chip.
- Buses = communication channels between the central processing unit and the RAM.
- Address Bus - address of the instruction or data that's in the memory address register from CPU to the memory. One directional bus.
- Data Bus - carries data between the processor and the memory. Bi-directional bus. Can also carry the instruction as well as data
- Control Bus - sends control signals between the CPU and components. Bi-directional bus so signals can be sent from either direction e.g. memory read/write
- System bus = collection of the three buses

Summary:

- Registers = memory location - PC, MAR, MDR, CIR, ACC, IR → FDEC
- CU - signals to control data flow
- ALU - arithmetic + logic calculations
- Buses = Communication channels
- Address Bus - Unidirectional, address to memory.
- Data Bus - Bidirectional, data and instructions
- Control Bus - Bidirectional, control signals from/to CPU
- System Bus = all 3

Page | 2

.1(b) The Fetch-Decode-Execute Cycle; including its effects on registers.
outu.be/OTDTdTYId2g

s:

s the address bus
wait at the RAM?
an opcode?
an operand?
branching?
ranch, how is the
t?
the data sent
has been
by the MDR?

to Quizlet)

Notes: △

- Fetch - PC holds address of next instruction → Copied to MAR. Sent along the address bus, waits to receive signal from the Control bus. The signal is sent to memory controller to tell it what to do with the address. Contents of requested memory is transferred in a data bus to MDR. If it is an instruction, it goes to CIR.
  ↓ Increment PC by 1
- Decode - Instruction in CIR is decoded by the decoding unit. 2 pieces of data in the CIR: Opcode and Operand. Opcode = command, operand = data/address for the opcode to use.
- Execute - If more data is needed, the operand is sent to the memory address register ∴ PC, MAR contain different values. Address is sent down the address bus, wait for control signal from control bus to say memory needs to be read. Data from memory is sent down the data bus → MDR → Accumulator.
  □
- Program Branches - can happen if there is an if statement, loop, function or procedure call. Branching causes the value in the PC to be changed to the contents of the operand. To get back to the original PC number so the program continues to run, a stack is used.

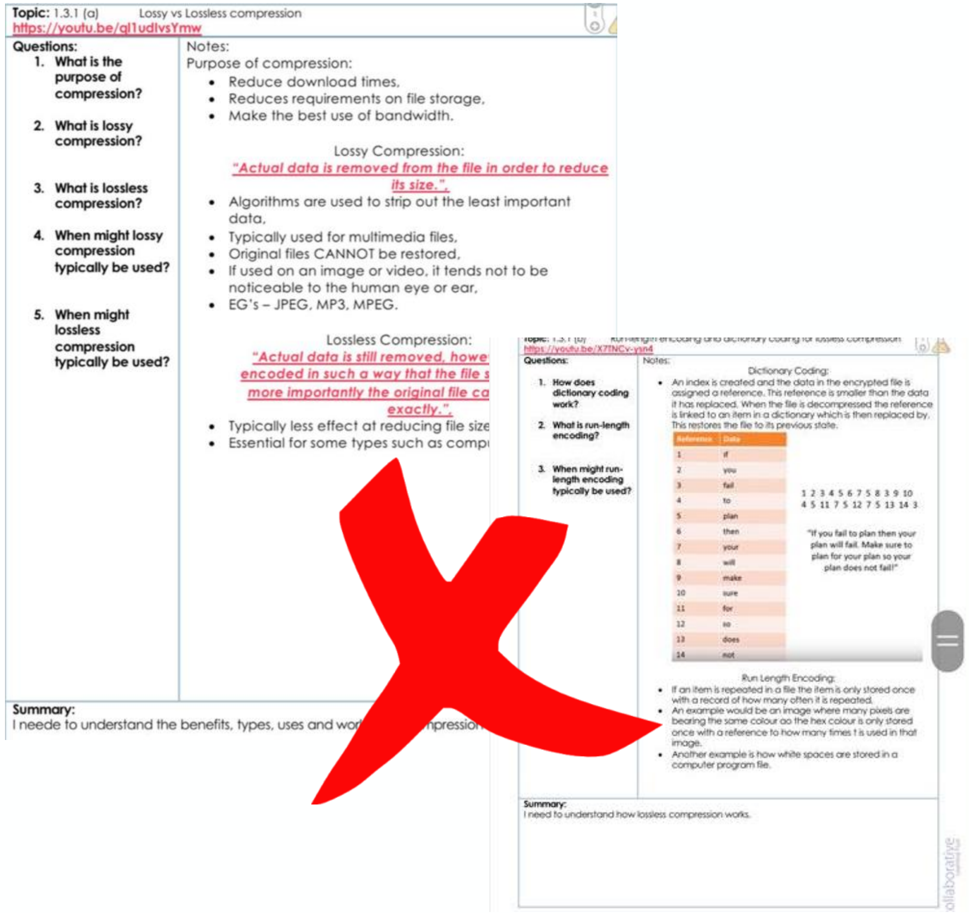
y:

AR → Addre waits for control bus to send a signal) → memory (RAM-
ch for data a ress) → Data Bus → MDR → CIR (if instruction) → PC
ents by 1 → Dec ing unit → Operand to MAR → Address Bus (waits) → memory
Bus → MDR → Accumulator.

ing - causes PC value to be changed to the contents of the operand. Stack can be used to return to the normal program

Page | 3

newcollaborative Learning Trust



Topic: 1.3.1 (a) Lossy vs Lossless compression
https://youtu.be/ql1udlvsYmw

Questions:

1. **What is the purpose of compression?**
2. **What is lossy compression?**
3. **What is lossless compression?**
4. **When might lossy compression typically be used?**
5. **When might lossless compression typically be used?**

Notes:

Purpose of compression:

- Reduce download times,
- Reduces requirements on file storage,
- Make the best use of bandwidth.

Lossy Compression:

"Actual data is removed from the file in order to reduce its size.",

- Algorithms are used to strip out the least important data,
- Typically used for multimedia files,
- Original files CANNOT be restored,
- If used on an image or video, it tends not to be noticeable to the human eye or ear,
- EG's – JPEG, MP3, MPEG.

Lossless Compression:

"Actual data is still removed, howe encoded in such a way that the file s more importantly the original file ca exactly.",

- Typically less effect at reducing file size
- Essential for some types such as compu

Summary:
I neede to understand the benefits, types, uses and wor mpression

Topic: 1.3.1 (b) Run-length encoding and dictionary coding for lossless compression
https://youtu.be/X7TNCv-ysn4

Questions:

1. **How does dictionary coding work?**
2. **What is run-length encoding?**
3. **When might run-length encoding typically be used?**

Notes:

Dictionary Coding:

- An index is created and the data in the encrypted file is assigned a reference. This reference is smaller than the data it has replaced. When the file is decompressed the reference is linked to an item in a dictionary which is then replaced by. This restores the file to its previous state.

Reference	Data
1	if
2	you
3	fail
4	to
5	plan
6	then
7	your
8	will
9	make
10	sure
11	for
12	so
13	does
14	not

1 2 3 4 5 6 7 5 8 3 9 10
4 5 11 7 5 12 7 5 13 14 3

"If you fail to plan then your plan will fail. Make sure to plan for your plan so your plan does not fail!"

Run Length Encoding:

- If an item is repeated in a file the item is only stored once with a record of how many often it is repeated.
- An example would be an image where many pixels are bearing the same colour so the hex colour is only stored once with a reference to how many times t is used in that image.
- Another example is how white spaces are stored in a computer program file.

Summary:
I need to understand how lossless compression works.

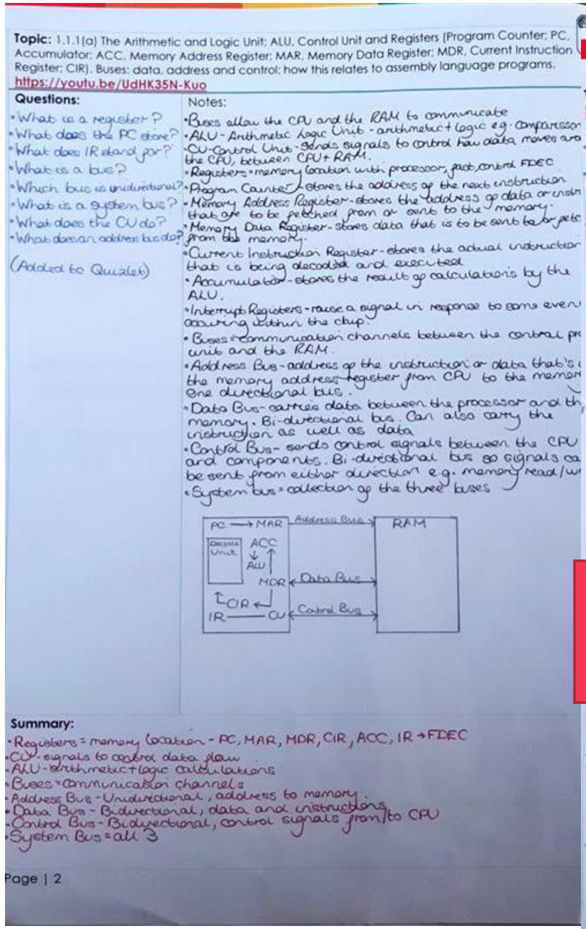
Example Cornell Notes

4. Flipped learning

Compulsory – must do!

Cornell note taking

Here are some examples – you can see how different students interpret the videos differently and how they used diagrams, colour, bullets and written explanations to aid their explanation.



Topic: 1.1.1(a) The Arithmetic and Logic Unit; ALU, Control Unit and Registers (Program Counter; PC, Accumulator; ACC, Memory Address Register; MAR, Memory Data Register; MDR, Current Instruction Register; CIR). Buses: data, address and control: how this relates to assembly language programs.
https://youtu.be/UdHK35N-Kuo

Questions:

- What is a register?
- What does the PC store?
- What does IR stand for?
- What is a bus?
- Which bus is unidirectional?
- What is a system bus?
- What does the CU do?
- What does an address bus do?

(Added to Quizlet)

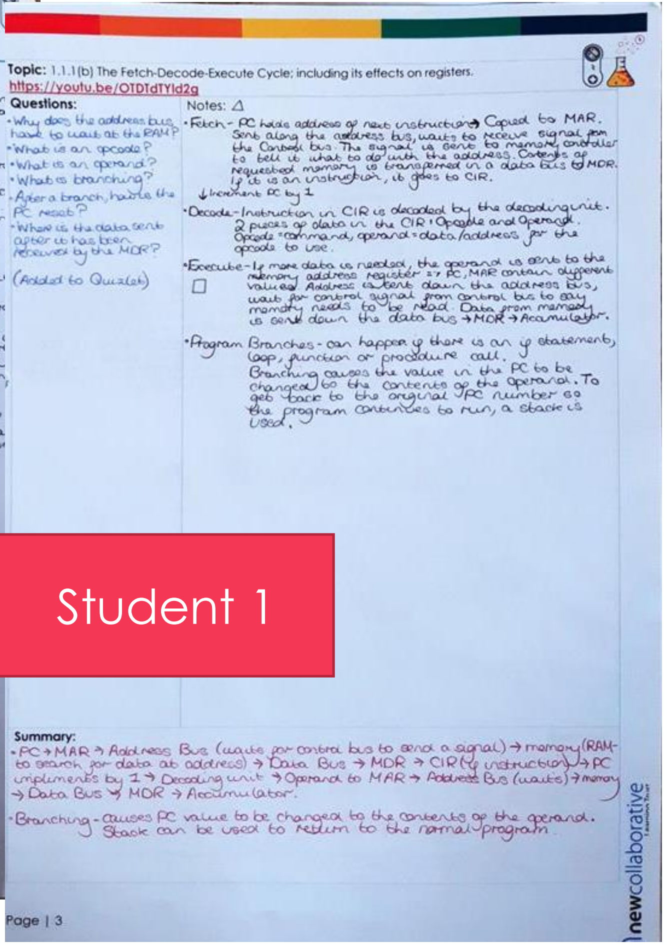
Notes:

- Buses allow the CPU and the RAM to communicate
- ALU - Arithmetic Logic Unit - arithmetic + logic e.g. comparison
- CU - Control Unit - sends signals to control how data moves aro the CPU, between CPU + RAM.
- Registers = memory location with processor, fast, control FDEC
- Program Counter - stores the address of the next instruction
- Memory Address Register - stores the address of data or instr that are to be fetched from or sent to the memory.
- Memory Data Register - stores data that is to be sent to or fetc from the memory.
- Current Instruction Register - stores the actual instruction that is being decoded and executed
- Accumulator - stores the result of calculations by the ALU.
- Interrupt Registers - raise a signal in response to some even occuring within the chip.
- Buses = communication channels between the central pr unit and the RAM.
- Address Bus - address of the instruction or data that's the memory address register from CPU to the memor One directional bus.
- Data Bus - carries data between the processor and th memory. Bi-directional bus. Can also carry the instruction as well as data
- Control Bus - sends control signals between the CPU and components. Bi-directional bus so signals ca be sent from either direction e.g. memory read/w
- System bus = collection of the three buses

Summary:

- Registers = memory location - PC, MAR, MDR, CIR, ACC, IR → FDEC
- CU - signals to control data flow
- ALU - arithmetic + logic calculations
- Buses = communication channels
- Address Bus - Unidirectional, address to memory.
- Data Bus - Bidirectional, data and instructions
- Control Bus - Bidirectional, control signals from/to CPU
- System Bus = all 3

Page | 2



Topic: 1.1.1(b) The Fetch-Decode-Execute Cycle; including its effects on registers.
https://youtu.be/OTDTdTYld2g

Questions:

- Why does the address bus have to wait at the RAM?
- What is an opcode?
- What is an operand?
- What is branching?
- After a branch, how is the PC reset?
- Where is the data sent after it has been received by the MDR?

(Added to Quizlet)

Notes: Δ

- Fetch - PC holds address of next instruction → Copied to MAR. Sent along the address bus, waits to receive signal from the Control bus. The signal is sent to memory controller to tell it what to do with the address. Contents of requested memory is transferred in a data bus to MDR. If it is an instruction, it goes to CIR.
  ↓ Increment PC by 1
- Decode - Instruction in CIR is decoded by the decoding unit. 2 pieces of data in the CIR: Opcode and Operand. Opcode = command, operand = data/address for the opcode to use.
- Execute - If more data is needed, the operand is sent to the memory address register ∴ PC, MAR contain different values. Address is sent down the address bus, wait for control signal from control bus to say memory needs to be read. Data from memory is sent down the data bus → MDR → Accumulator.
  □
- Program Branches - can happen if there is an if statement, loop, function or procedure call.
  Branching causes the value in the PC to be changed to the contents of the operand. To get back to the original PC number so the program continues to run, a stack is used.

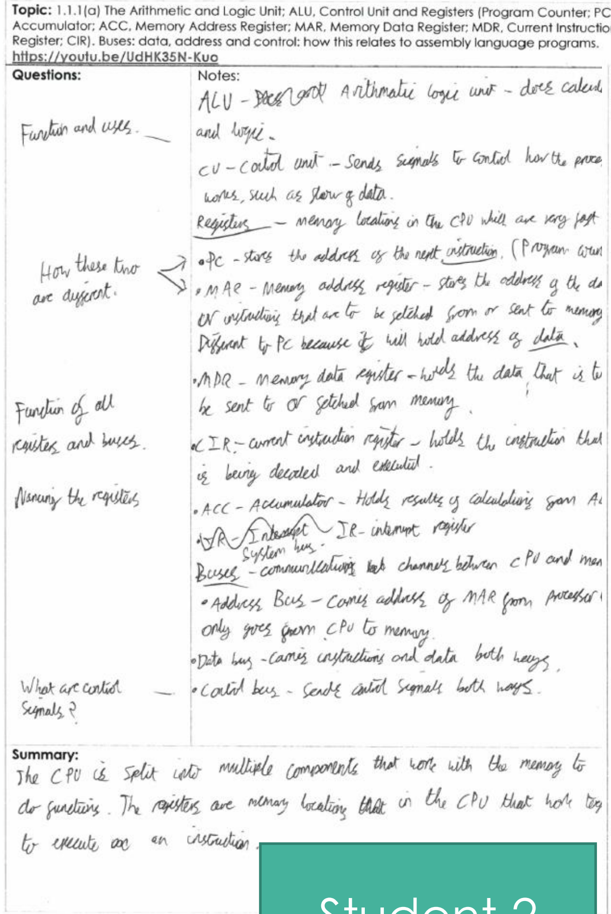
Summary:

- PC → MAR → Address Bus (waits for control bus to send a signal) → memory (RAM - to search for data at address) → Data Bus → MDR → CIR (if instruction) → PC increments by 1 → Decoding unit → Operand to MAR → Address Bus (waits) → memory → Data Bus → MDR → Accumulator.
- Branching - causes PC value to be changed to the contents of the operand. Stack can be used to return to the normal program

Page | 3

newcollaborative

Student 1



Topic: 1.1.1(a) The Arithmetic and Logic Unit; ALU, Control Unit and Registers (Program Counter; PC, Accumulator; ACC, Memory Address Register; MAR, Memory Data Register; MDR, Current Instruction Register; CIR). Buses: data, address and control: how this relates to assembly language programs.
https://youtu.be/UdHK35N-Kuo

Questions:

Function and uses. —

How these two are different.

Function of all registers and buses.

Naming the registers

What are control signals? —

Notes:

ALU - ~~Does~~ Arithmatic logic unit - does calcul and logic.

CU - Control unit - Sends signals to control how the proce works, such as flow of data.

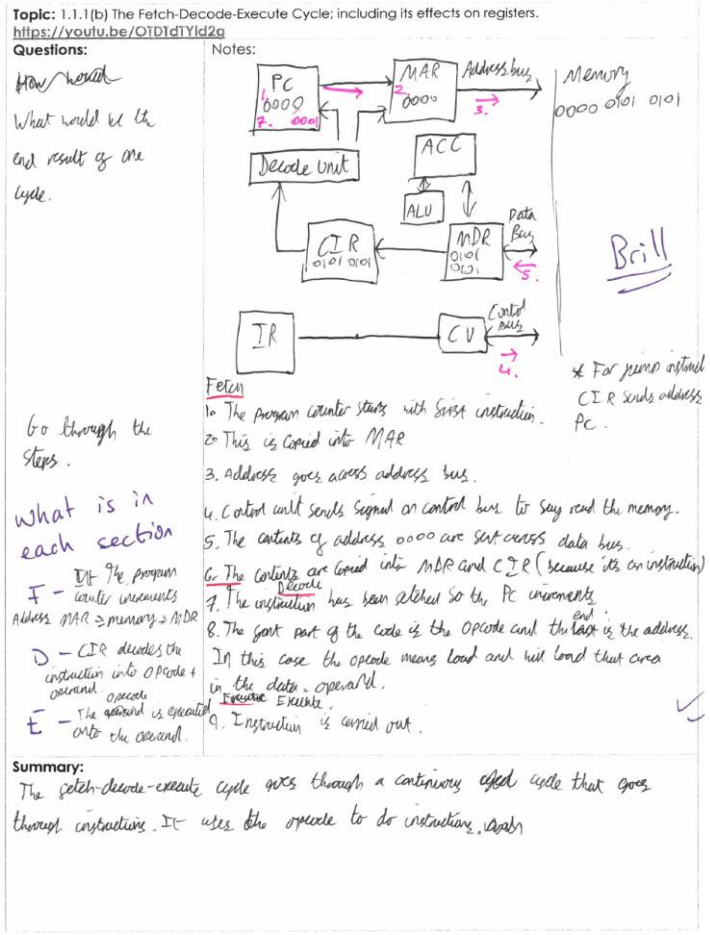
Registers - memory locations in the CPU which are very fast

- PC - stores the address of the next instruction. (Program coun
- MAR - Memory address register - stores the address of the da or instructions that are to be fetched from or sent to memory. Different to PC because it will hold address of data.
- MDR - memory data register - holds the data that is to be sent to or fetched from memory.
- CIR - current instruction register - holds the instruction that is being decoded and executed.
- ACC - Accumulator - Holds results of calculations from A
- ~~IR - Interrupt~~ IR - interrupt register

Buses - System bus - communications channels between CPU and mem

- Address Bus - Carries address of MAR from processor only goes from CPU to memory.
- Data bus - Carries instructions and data both ways.
- Control bus - Sends control signals both ways.

Summary:
The CPU is split into multiple components that work with the memory to do functions. The registers are memory locations that in the CPU that work tog to execute ~~an~~ an instruction.



Topic: 1.1.1(b) The Fetch-Decode-Execute Cycle; including its effects on registers.
https://youtu.be/OTDTdTYld2g

Questions:

~~How~~ ~~would~~

What would be the end result of one cycle.

Go through the steps.

what is in each section

F - The program counter increments
Address MAR ⇒ memory ⇒ MDR

D - CIR decodes the instruction into opcode + operand

E - The opcode is executed onto the operand

Notes:

* For jump instructions CIR sends address PC.

Fetch

1. The program counter starts with first instruction.
2. This is copied into MAR
3. Address goes across address bus.
4. Control unit sends signal on control bus to say read the memory.
5. The contents of address 0000 are sent across data bus.
6. The contents are copied into MDR and CIR (because its an instruction)

Decode

7. The instruction has been fetched so the PC increments.
8. The front part of the code is the opcode and the end is the address. In this case the opcode means load and will load that area in the data operand.

Execute

9. Instruction is carried out.

Summary:
The fetch-decode-execute cycle goes through a continuous cycle that goes through instructions. It uses the opcode to do instructions.

Student 2

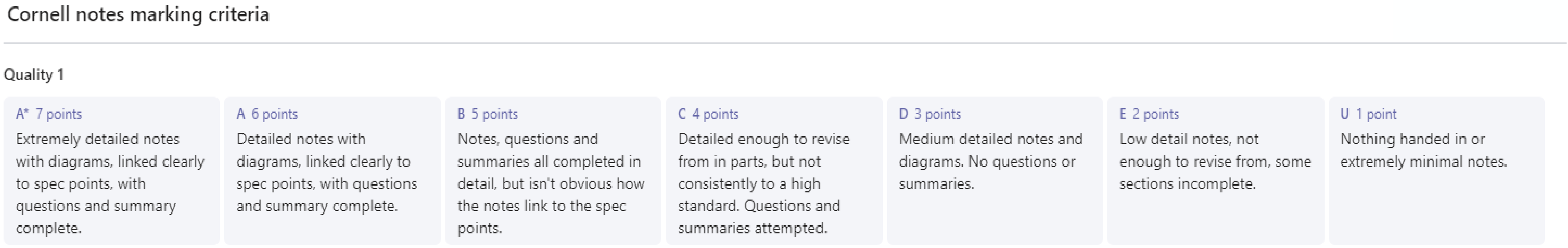
Example Cornell Notes

Optional – strongly recommended!

4. Flipped learning

Cornell note taking

Exercise: From the example notes on the previous page, look at what they wrote, and then given them a score using the rubric below.



Cornell notes marking criteria

Quality 1

A* 7 points	A 6 points	B 5 points	C 4 points	D 3 points	E 2 points	U 1 point
Extremely detailed notes with diagrams, linked clearly to spec points, with questions and summary complete.	Detailed notes with diagrams, linked clearly to spec points, with questions and summary complete.	Notes, questions and summaries all completed in detail, but isn't obvious how the notes link to the spec points.	Detailed enough to revise from in parts, but not consistently to a high standard. Questions and summaries attempted.	Medium detailed notes and diagrams. No questions or summaries.	Low detail notes, not enough to revise from, some sections incomplete.	Nothing handed in or extremely minimal notes.

Student 1:

Your score:

Reasons for your score:

Student 2:

Your score:

Reasons for your score:

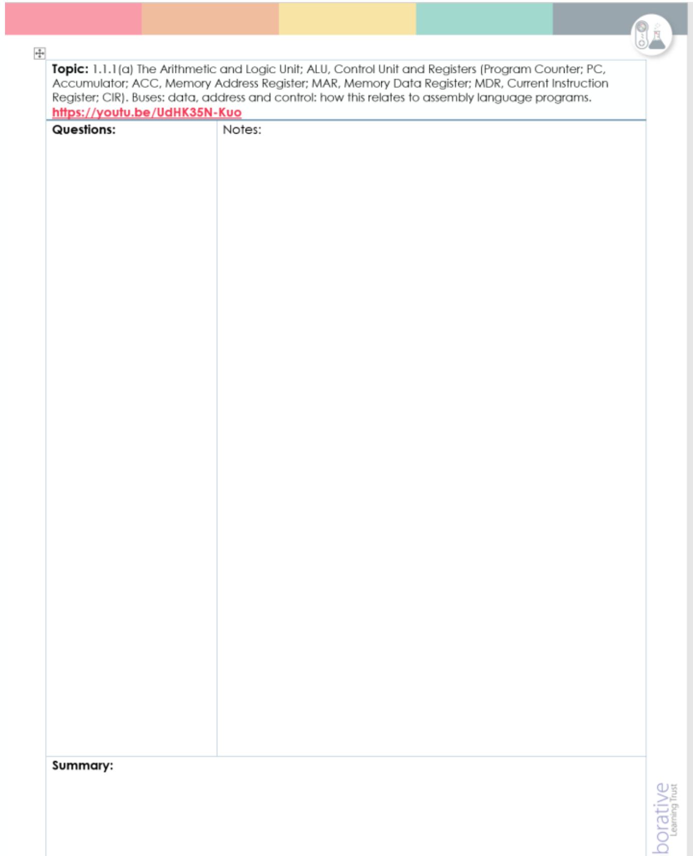
Expected completion time: 30 mins

4. Flipped learning

Compulsory – must do!

Cornell note taking

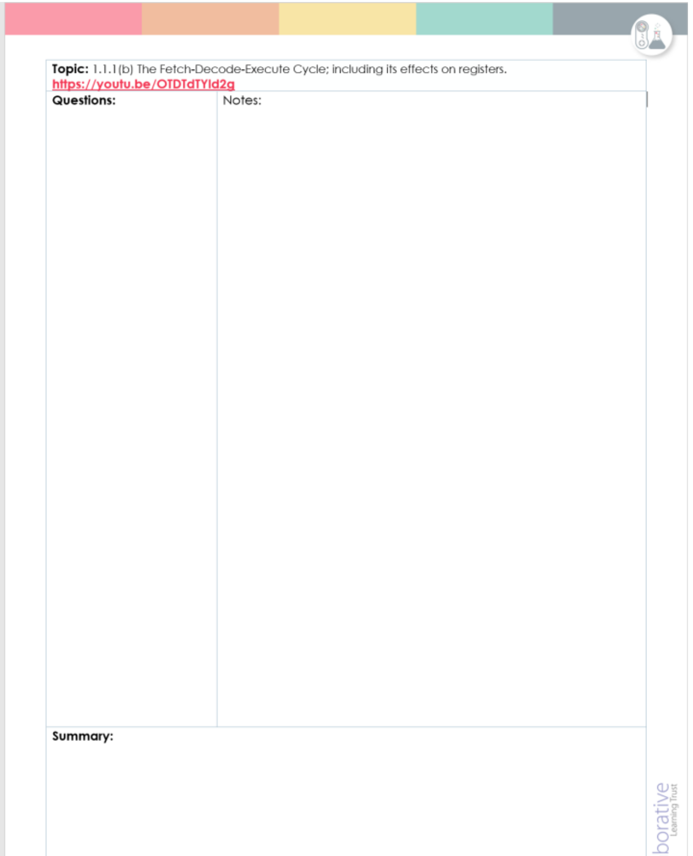
Exercise: Write Cornell notes for the 1.1.1 videos. The blank document with the video links can be found on the link below. Make the notes as detailed as you can.



Topic: 1.1.1(a) The Arithmetic and Logic Unit; ALU, Control Unit and Registers (Program Counter; PC, Accumulator; ACC, Memory Address Register; MAR, Memory Data Register; MDR, Current Instruction Register; CIR). Buses: data, address and control: how this relates to assembly language programs.
https://youtu.be/UdHK35N-Kuo

Questions:	Notes:

Summary:



Topic: 1.1.1(b) The Fetch-Decode-Execute Cycle; including its effects on registers.
https://youtu.be/OTDTdTYld2g

Questions:	Notes:

Summary:

Download file: https://learn-cs.com/wp-content/uploads/2020/05/1.1.1-Cornell-notes-DIL.pdf

Expected completion time: 1 hour 30 mins

Optional – strongly recommended!

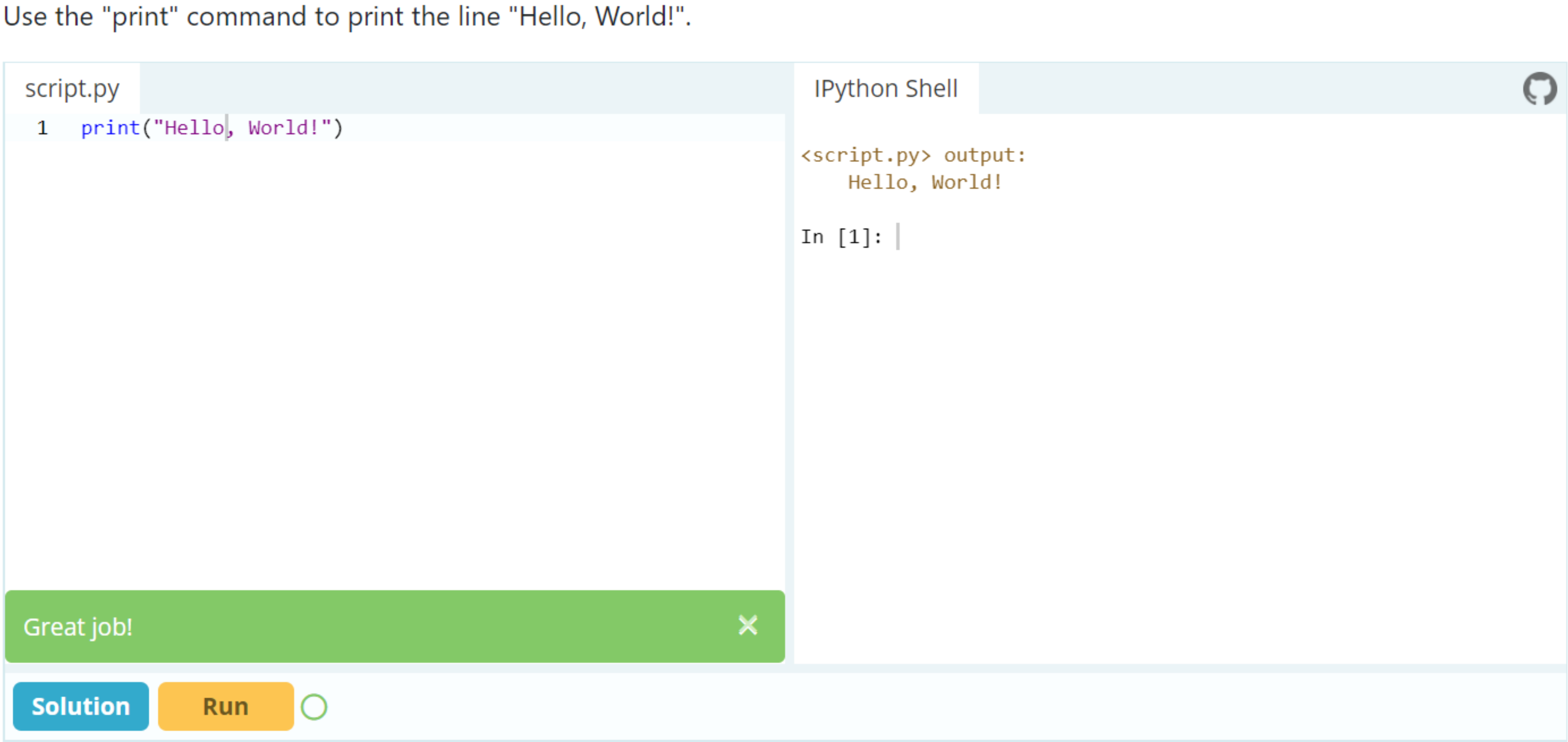
5. Programming

Hello, world!

https://www.learnpython.org/en/Hello%2C_World%21

Exercise:

1. Have a go at each of the warm up tasks, be sure to read them carefully
2. Paste your code for the final exercise below:



Use the "print" command to print the line "Hello, World!".

```
print("Hello, World!")
```

```
<script.py> output:
    Hello, World!

In [1]:
```

Great job!

Optional – strongly recommended!

5. Programming

Variables and types

https://www.learnpython.org/en/Variables_and_Types

Exercise:

1. Have a go at each of the warm up tasks, be sure to read them carefully
2. Paste your code for the final exercise below:

Optional – strongly recommended!

5. Programming

Lists

https://www.learnpython.org/en/Lists

Exercise:

1. Have a go at each of the warm up tasks, be sure to read them carefully
2. Paste your code for the final exercise below:

Optional – strongly recommended!

5. Programming

String formatting

https://www.learnpython.org/en/String_Formatting

Exercise:

1. Have a go at each of the warm up tasks, be sure to read them carefully
2. Paste your code for the final exercise below:

Optional – strongly recommended!

5. Programming

String operations

https://www.learnpython.org/en/Basic_String_Operations

Exercise:

1. Have a go at each of the warm up tasks, be sure to read them carefully
2. Paste your code for the final exercise below:

Expected completion time: 10 – 20 mins

Optional – strongly recommended!

5. Programming

Conditions - branching

https://www.learnpython.org/en/Conditions

Exercise:

1. Have a go at each of the warm up tasks, be sure to read them carefully
2. Paste your code for the final exercise below:

Optional – strongly recommended!

5. Programming

Loops - iteration

https://www.learnpython.org/en/Loops

Exercise:

1. Have a go at each of the warm up tasks, be sure to read them carefully
2. Paste your code for the final exercise below:

Optional – strongly recommended!

5. Programming

Functions – reusable code

https://www.learnpython.org/en/Functions

Feel free to continue with the online tutorials, but you should have enough knowledge to be able to attempt the 'challenge' exercises on the next slides.

Exercise:

1. Have a go at each of the warm up tasks, be sure to read them carefully
2. Paste your code for the final exercise below:

6. Programming

Preparation for challenge 1

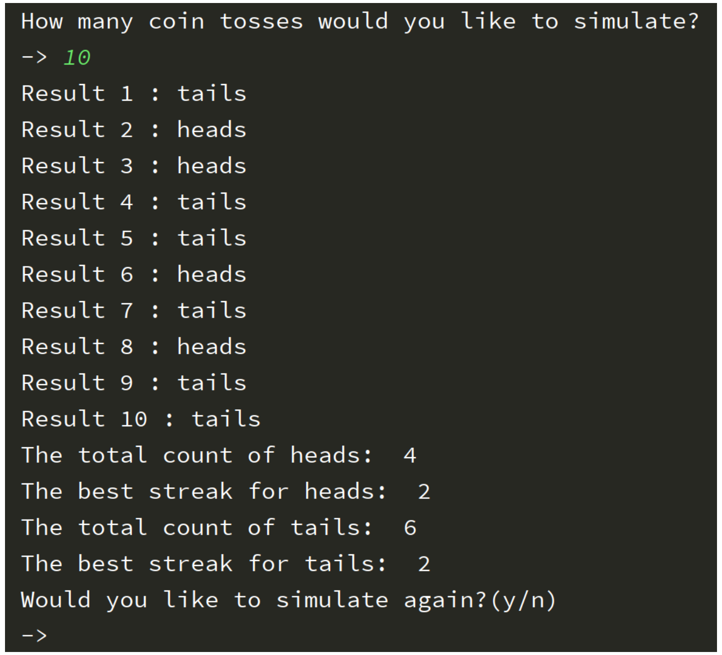
Heads or tails

Compulsory – must do!

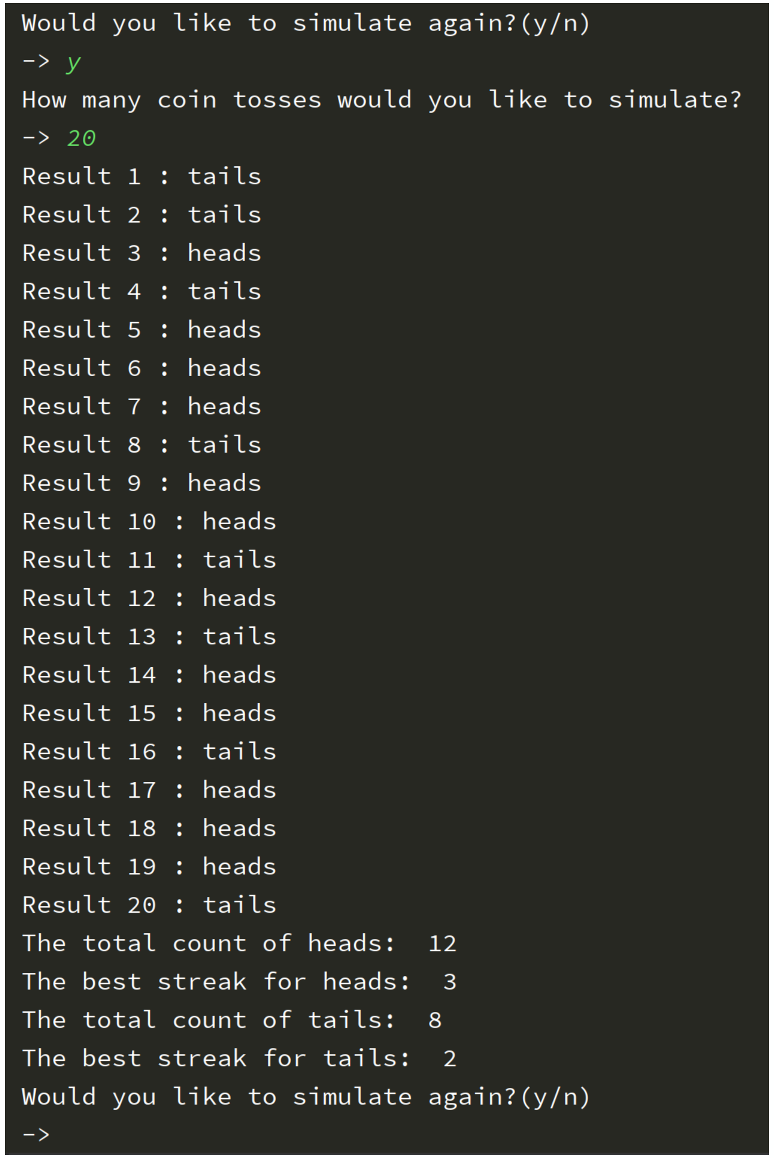
Create a program that asks the user how many times they would like to simulate tossing a coin. It should then generate either a 'head' or 'tails' randomly, for the specified number of times. It should print out at the end:

1. How many coin tosses it simulated
2. How many heads in total
3. How many tails in total
4. The best head streak – best run of heads in a row without any tails
5. The best tails streak – best run of tails in a row without any heads

To the right are example outputs to help you work out how to program the solution:



```
How many coin tosses would you like to simulate?
-> 10
Result 1 : tails
Result 2 : heads
Result 3 : heads
Result 4 : tails
Result 5 : tails
Result 6 : heads
Result 7 : tails
Result 8 : heads
Result 9 : tails
Result 10 : tails
The total count of heads:  4
The best streak for heads:  2
The total count of tails:  6
The best streak for tails:  2
Would you like to simulate again?(y/n)
->
```



```
Would you like to simulate again?(y/n)
-> y
How many coin tosses would you like to simulate?
-> 20
Result 1 : tails
Result 2 : tails
Result 3 : heads
Result 4 : tails
Result 5 : heads
Result 6 : heads
Result 7 : heads
Result 8 : tails
Result 9 : heads
Result 10 : heads
Result 11 : tails
Result 12 : heads
Result 13 : tails
Result 14 : heads
Result 15 : heads
Result 16 : tails
Result 17 : heads
Result 18 : heads
Result 19 : heads
Result 20 : tails
The total count of heads:  12
The best streak for heads:  3
The total count of tails:  8
The best streak for tails:  2
Would you like to simulate again?(y/n)
->
```

Expected completion time: 1 hour

Compulsory – must do!

6. Programming

Code for challenge 1

Heads or tails

Exercise: Paste your solution and proof of working here:

6. Programming

Compulsory – must do!

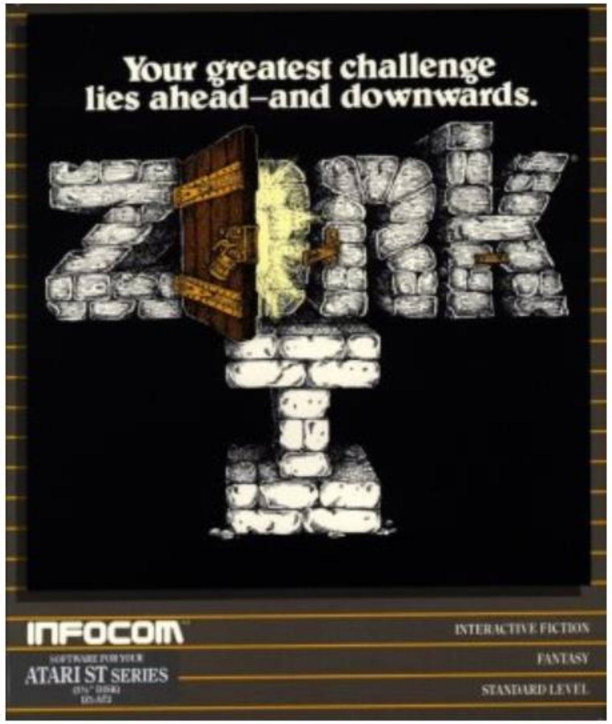
Preparation for challenge 2

Exercise: Play the text adventure game Zork

That's right, I am asking you to play a game, and that forms part of your Summer Independent Learning!

At the time of writing this link worked to a web version of Zork http://textadventures.co.uk/games/view/5zyoqrsugeopel3ffhz_vq/zork

If it doesn't, then Google "Text Adventure Zork Online" and you should be able to find a link.



There is method to my madness, playing a cutting edge game like this (for 1977ish) allows you to think about all the skills a programmer needs:

- Use of variables
- Inputs
- Outputs
- Lists
- Operators
- Formatting strings
- Conditions and branching
- Loops or iteration
- Functions

As you play think about what must be happening under the hood, how do you collect items, have battles, have choices, get random responses, move around?

If you get stuck, try the help guide here: http://www.eristic.net/games/infocom/zork1.html

Lets play some games! Expected completion time: I have no idea! I got lost inside this dungeon for what seemed like an eternity, maybe you will have more luck!

6. Programming

Compulsory – must do!

Making your version of the game

Exercise: Outline the basic idea behind your own text adventure game here.

To think about:

1. Who are your characters?
2. Are they playable?
3. Who are the non playable characters?
4. What skills or weapons can you pick up along the way?
5. What items can you carry and how many?
6. How do you win?
7. Will you have health or lives?
8. How do you manoeuvre around?

My ideas:

My ideas continued:

6. Programming

Making your version of the game

Exercise: Paste your code here – if you need more space just duplicate the slide or put it on a word document and bring it with you to your first lesson in September. Feel free to delete the help boxes.

Help 1: Tech with Tim
https://youtu.be/DEcFCn2ubSg

This is a basic introduction to a text adventure game on YouTube – a good place to get you started.

Help 2: Invent with Python
https://inventwithpython.com/invent4thed/chapter5.html

This game was has a lot of similar features to Zork and will help if you read the explanation and code.

Expected completion time: It is up to you – it depends how far you want to take it!