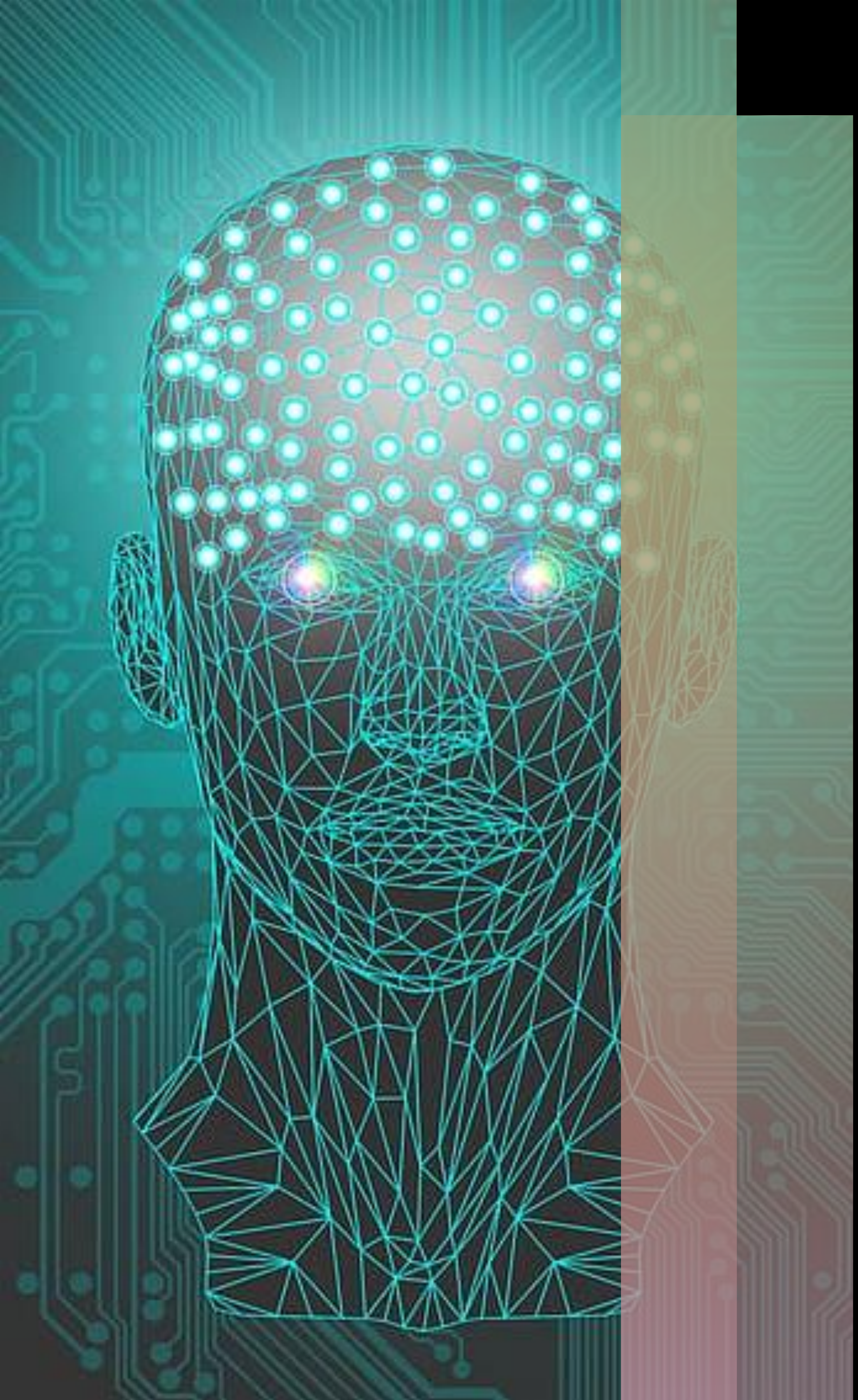
COM4013 Introduction to Software Development

- Week 13
- Structures (struct)
- Propositions, Logic, and Truth Tables

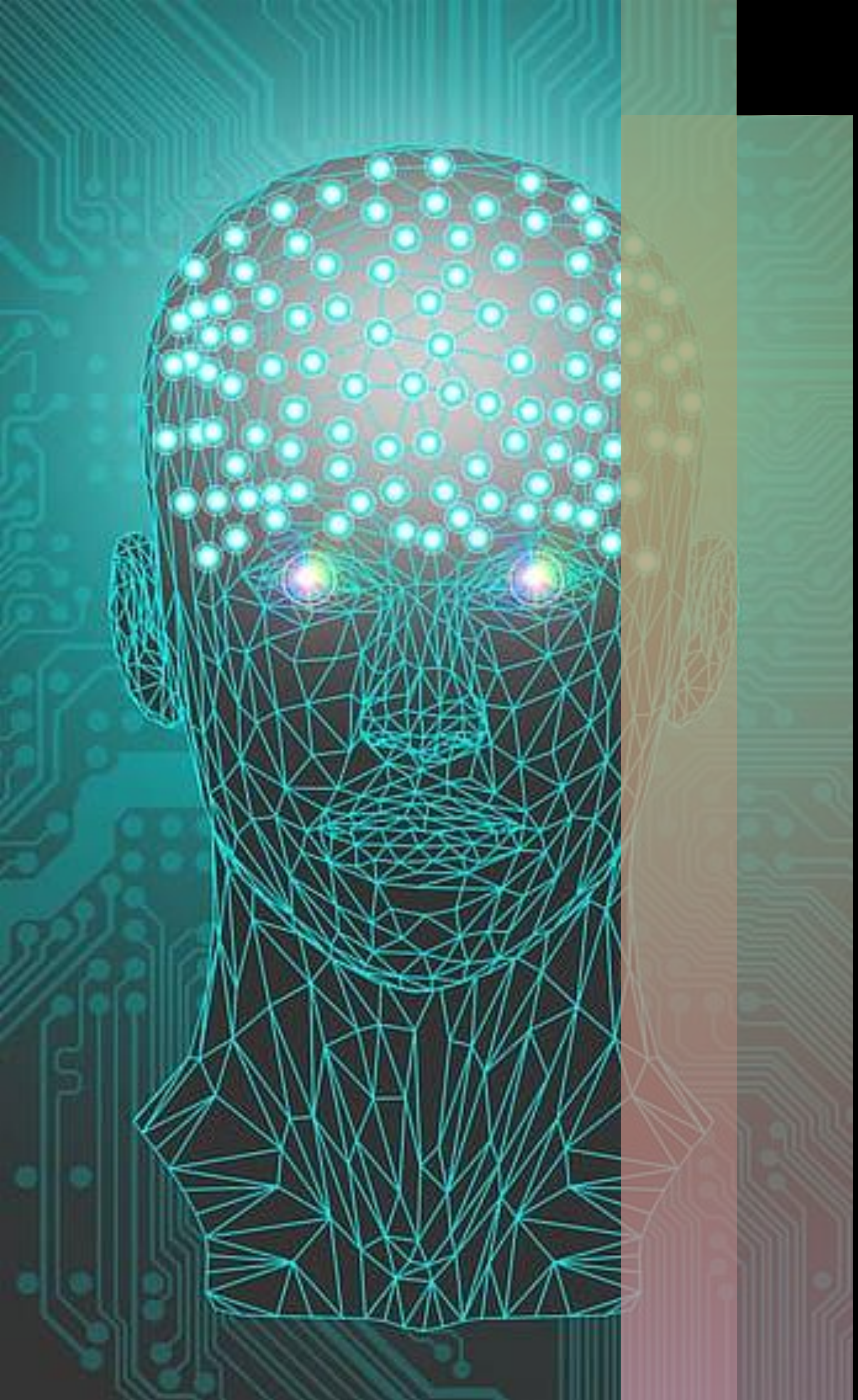
- *Umar Arif – u.arif@leedstrinity.ac.uk*



Structures

- **Structures** are a way to hold multiple values in a single object.
- Structures allow you to define your own data types

- Why useful?
  - Convenient to lump data together. A structure is an "aggregate" data type
  - For example, a student has a name, but also an id
  - A bank account has a balance, a sort code, name and address
  - Like the fields used in database record



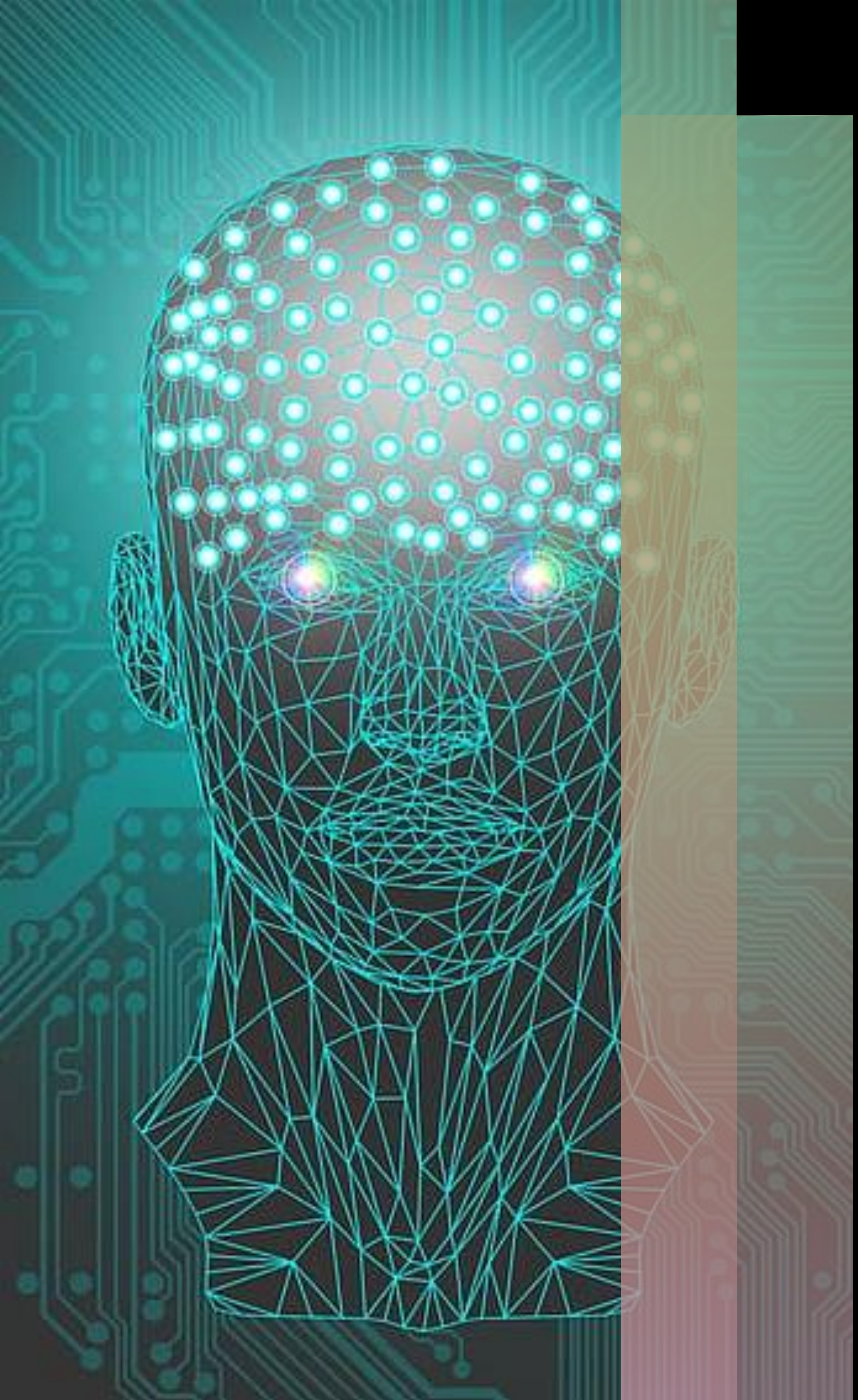
Example Structures

```
struct SPerson
{
	string surname;
	string firstname;
	int age;
};

struct SStudent
{
	string name;
	string id;
};
```

```
struct SBankAccount
{
	string sortCode;
	float balance;
	string name;
	string address;
};
```

- Declare before the main function in C++
- Keyword is struct, and the structure needs a name
- Curly brackets around the definition, and a terminating
- They are a means for combining several related items
  - The items may be of different data types



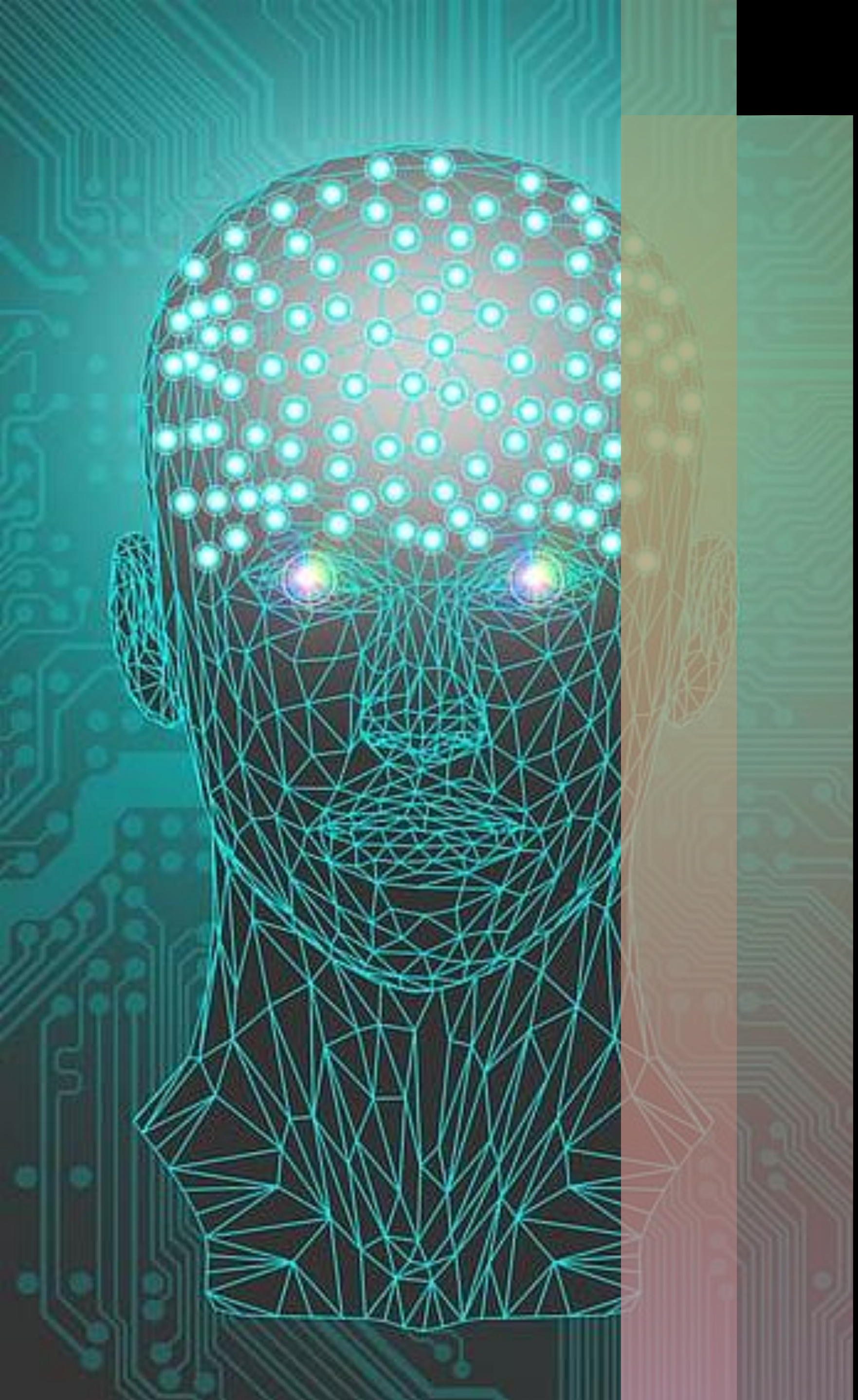
Example Structures

```
struct SAccount
{
    string name;
    float balance;
};

int main( )
{
    SAccount currentAccount;
    currentAccount.name = "fred";
    currentAccount.balance = 20000;

    SAccount savingAccount = { "Ben", 10000 };
}
```

- As shown above, you can initialise a struct in two ways
- To set or retrieve variables, simply use the dot operator
- Note that it is possible to have functions and structures within other structs. And structs within your classes



Proposition

- **Propositions** are fundamental for logical reasoning
- They are a declarative statement, which either resolves to true or false
- It is of the utmost importance to understand that propositions alone are the building blocks for more complex logic

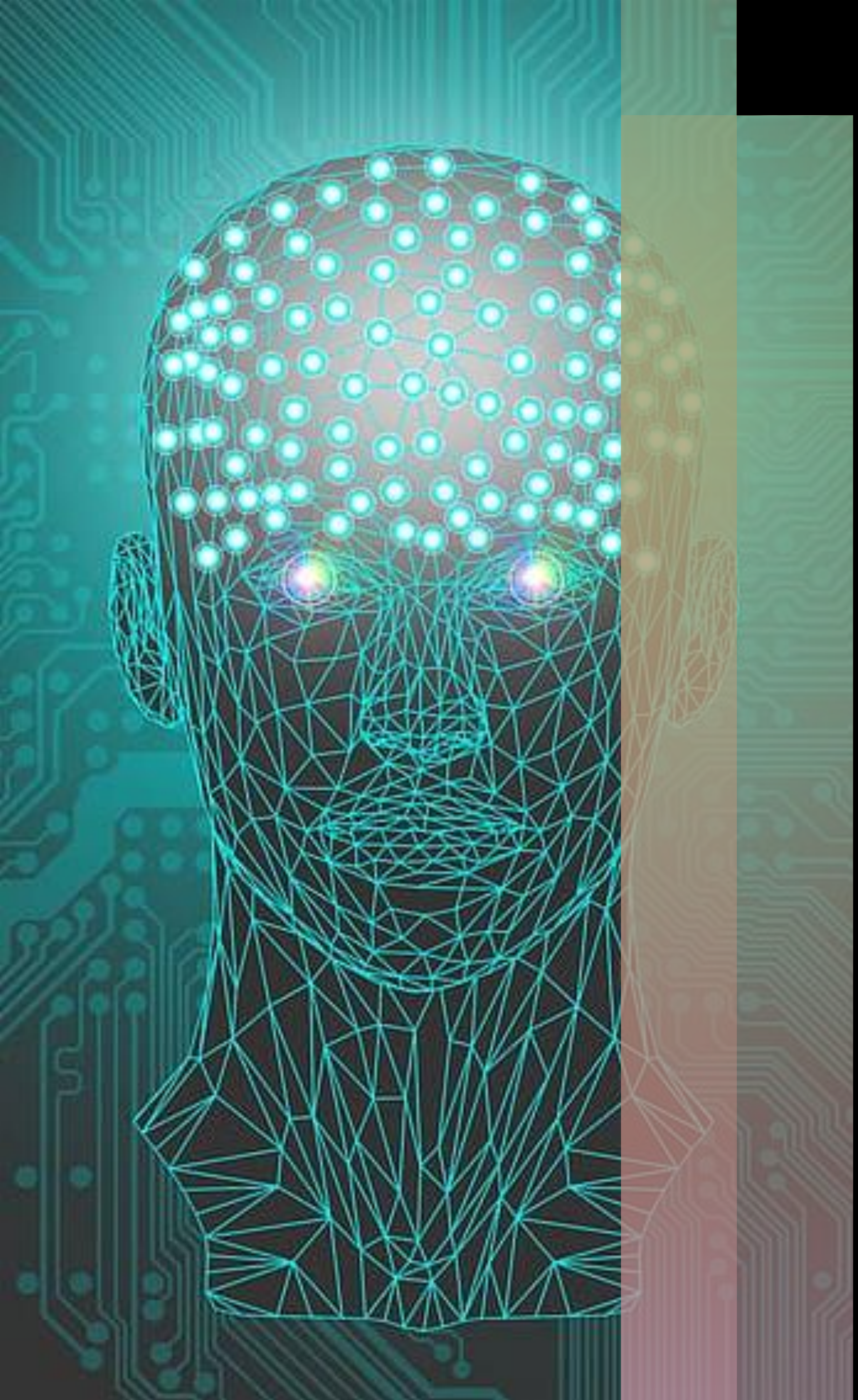
- Why is this important?
    - Logic provides us with the means of making decisions and solving problems
    - A solid understanding of logic allows us to make good decisions
    - In programming we use Booleans expressions which evaluate to true or false allowing dynamic decision making
        - We do this using programming syntax: i.e., if, for, while, and do-while statements
        - `if (x > 10) { // do something }`

Proposition Example

- The simplest, and most abstract logic we can study is called propositional logic
- A proposition is a **statement** that can be either true or false; it must be one or the other, and it cannot be both

- The following are propositions:
  - The sun rises in the east – true
  - Dylan is a rare name – false
  - Snow is white – true
  - 2 * 2 = 10 – false

- Whereas the following are not:
  - How are you feeling today? // Open ended question
  - 5 * 7 // Mathematical expression
  - Go to bed Dylan // Command
  - Hooray! // Exclamation not statement

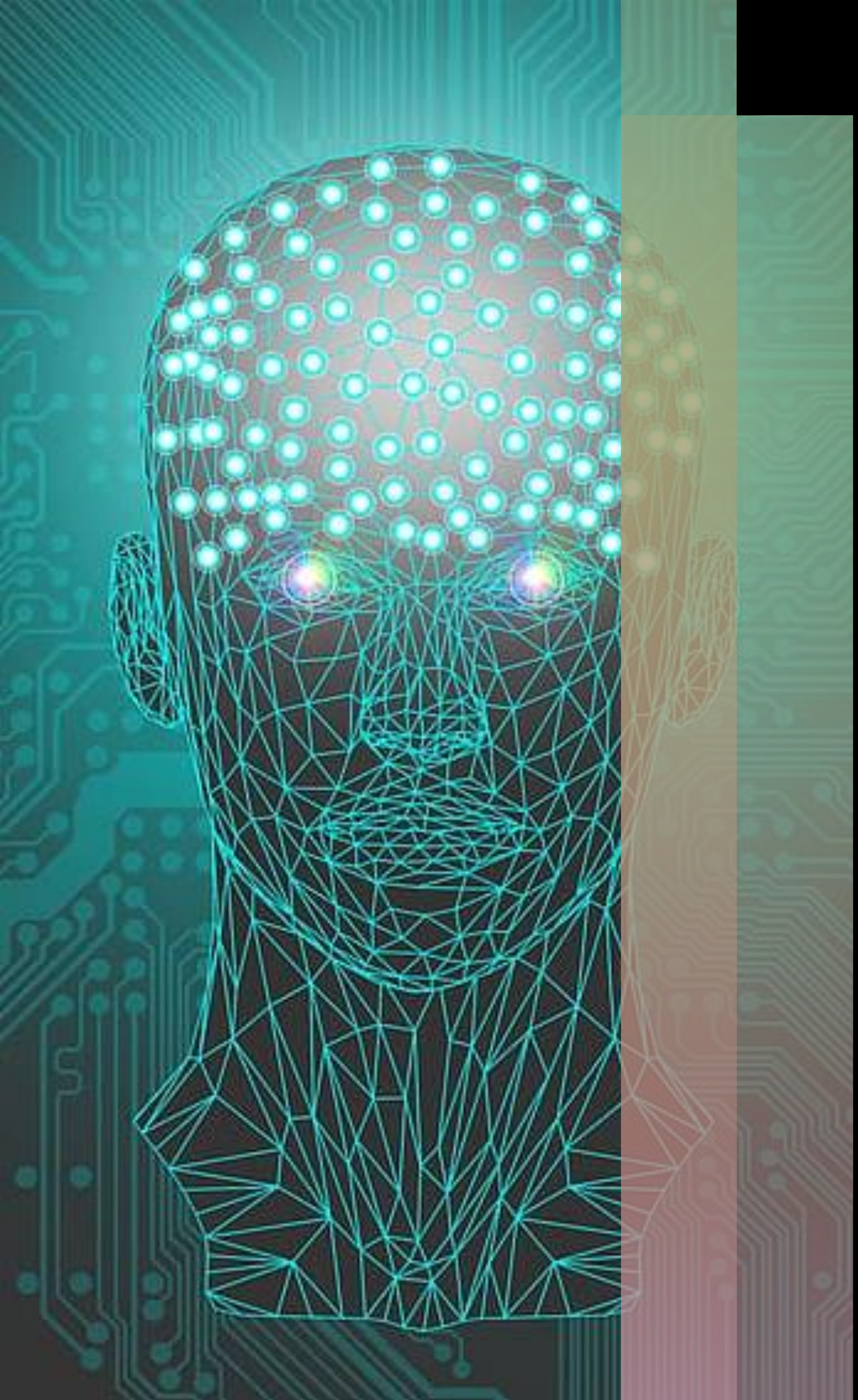


How to Determine a Proposition?

- If you want to discern whether a **potential statement** is a proposition, we can use the following **sentence starter**:

```
It is true that...
```

- If the sentence still makes sense with this sentence starter in place, then it probably is a proposition
- Single statement propositions are known as **atomic propositions**
  - Atomic as in its truthiness of falsity are not dependent on other factors (i.e., other propositions)
  - All the examples I have provided (thus far) have been atomic



Abbreviate Propositions

- From this point forth I will refer to our propositions as lower-case roman letter.
- These are known as **propositional variables**

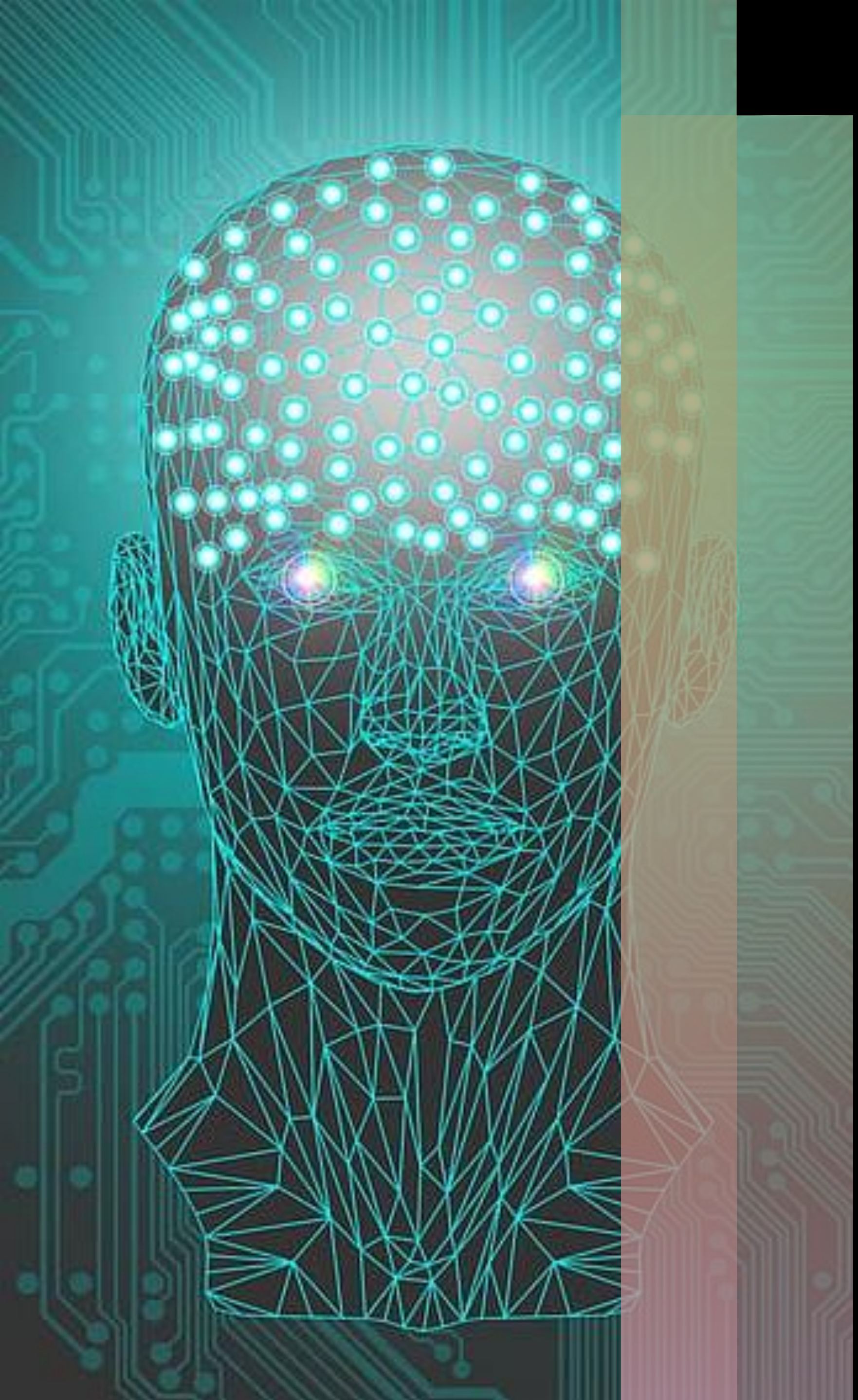
p, q, r, . . .

- So, instead of saying "it is sunny today", I will refer to this proposition as p.

- Of course, now that I've done this, for clarity let's note this:

Let p be it is sunny today.

- When evaluating your own propositions do make it clear what your propositional variables stand for...

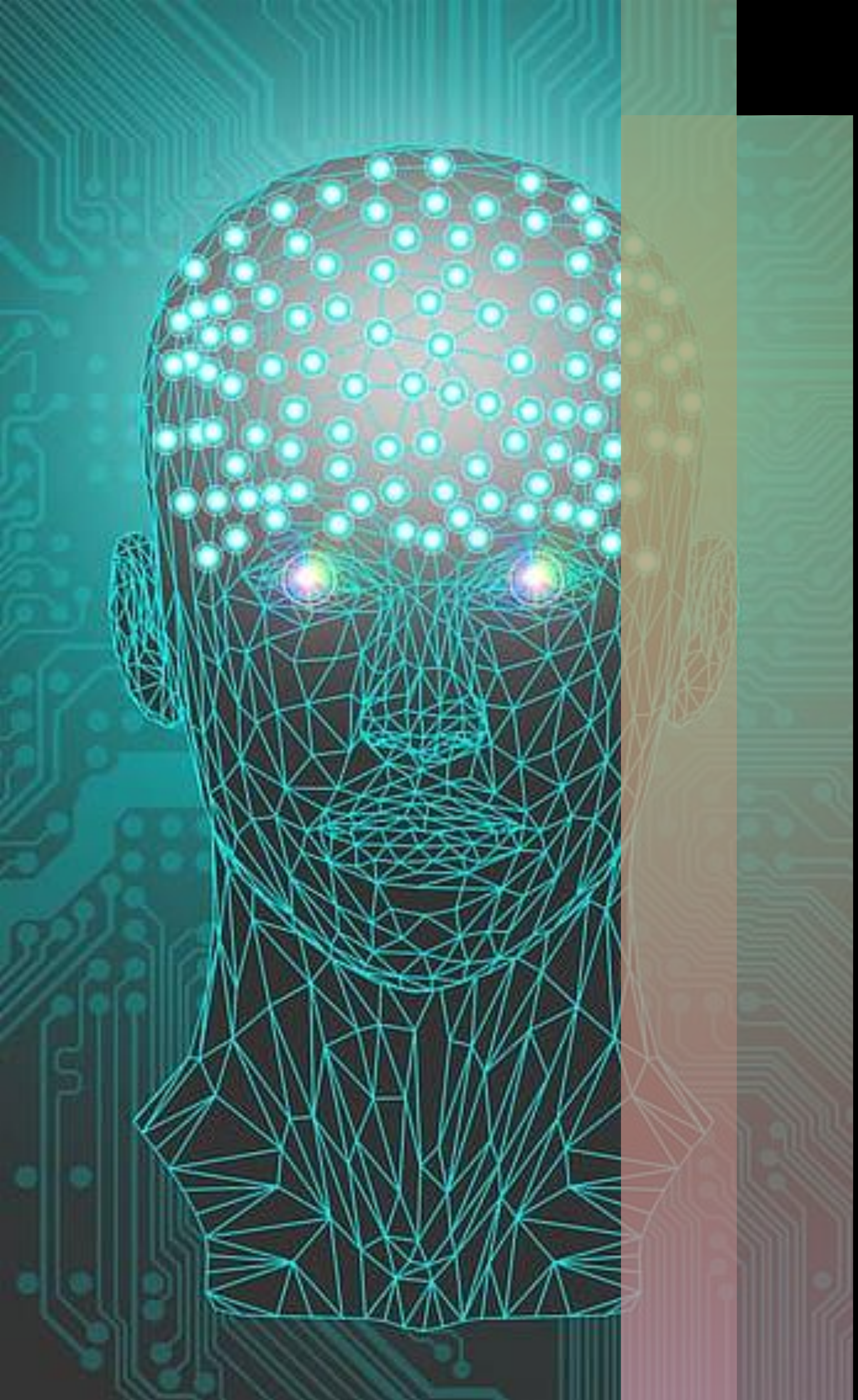


Connectives

- Now that we understand propositions and atomic propositions, we can move onto creating **complex propositions**
- Complex propositions are propositions that are made up of **2 or more** combined propositions
- The complex propositions are built up using **connectives**
- These connect the propositions together. Like glue :)
- The connectives we will discuss today include:

$\wedge$	and	(& or .)
$\vee$	or	(| or +)
$\neg$	not	(~)
$\Rightarrow$	implies	($\supset$ or $\rightarrow$)
$\Leftrightarrow$	iff	

- The symbols on the left may be different in textbooks that you read (alternatives noted on the right)



AND (∧)

- Any two propositions can be combined to form a third proposition called the **conjunction** of the original propositions.
- If p and q are arbitrary propositions, then the conjunction of p and q is written as:

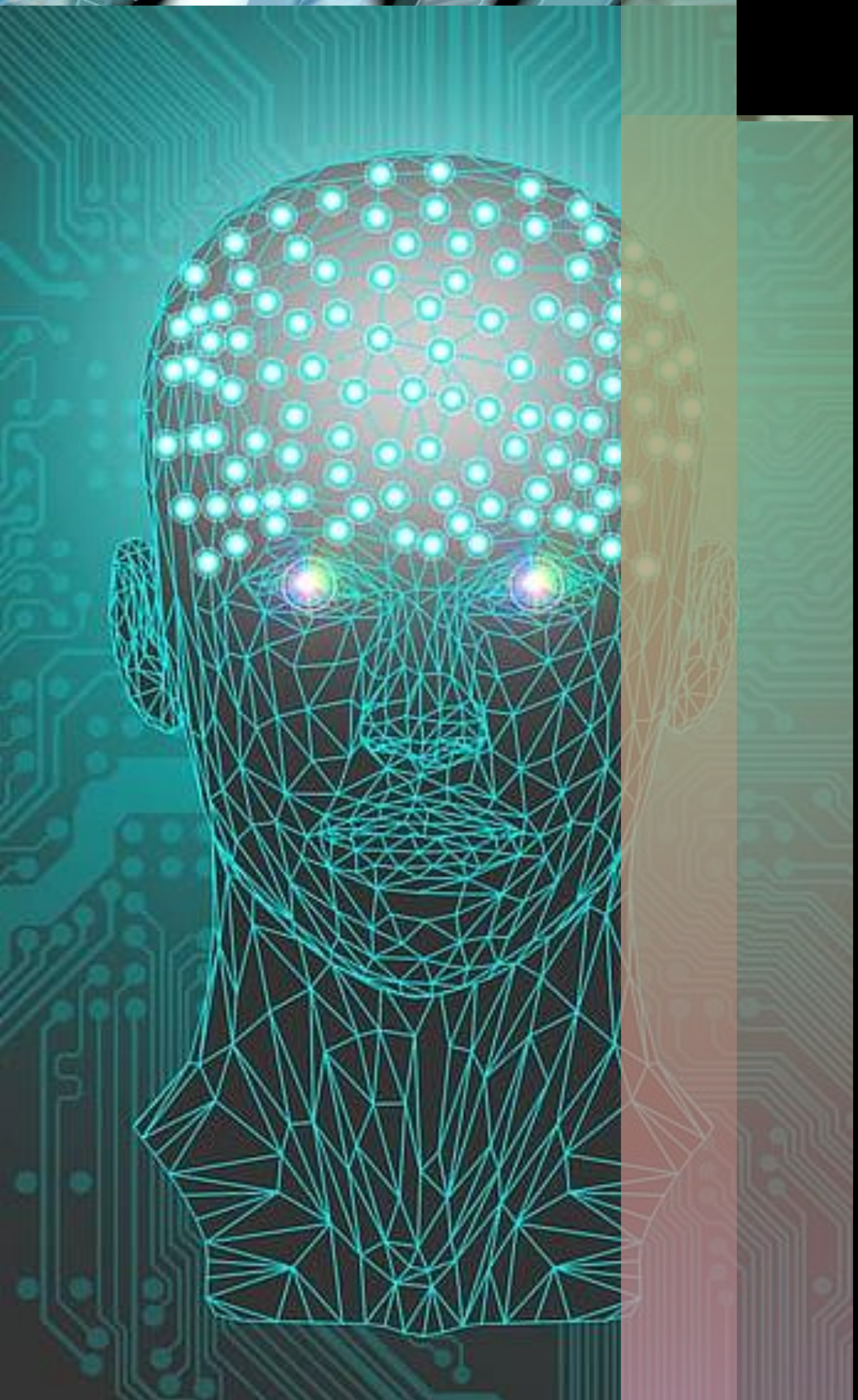
$$p \wedge q$$

- and will be true if both p and q are true

- We can summaries all the permutations of these propositions using a truth table - (T means true and F means false)
- To figure out the number of different combination of truth values we can use the following formula:

No Of Proposition Permutation Combinations = 2^n

- In this case, 2 to the power of 2 is 4 (combinations)



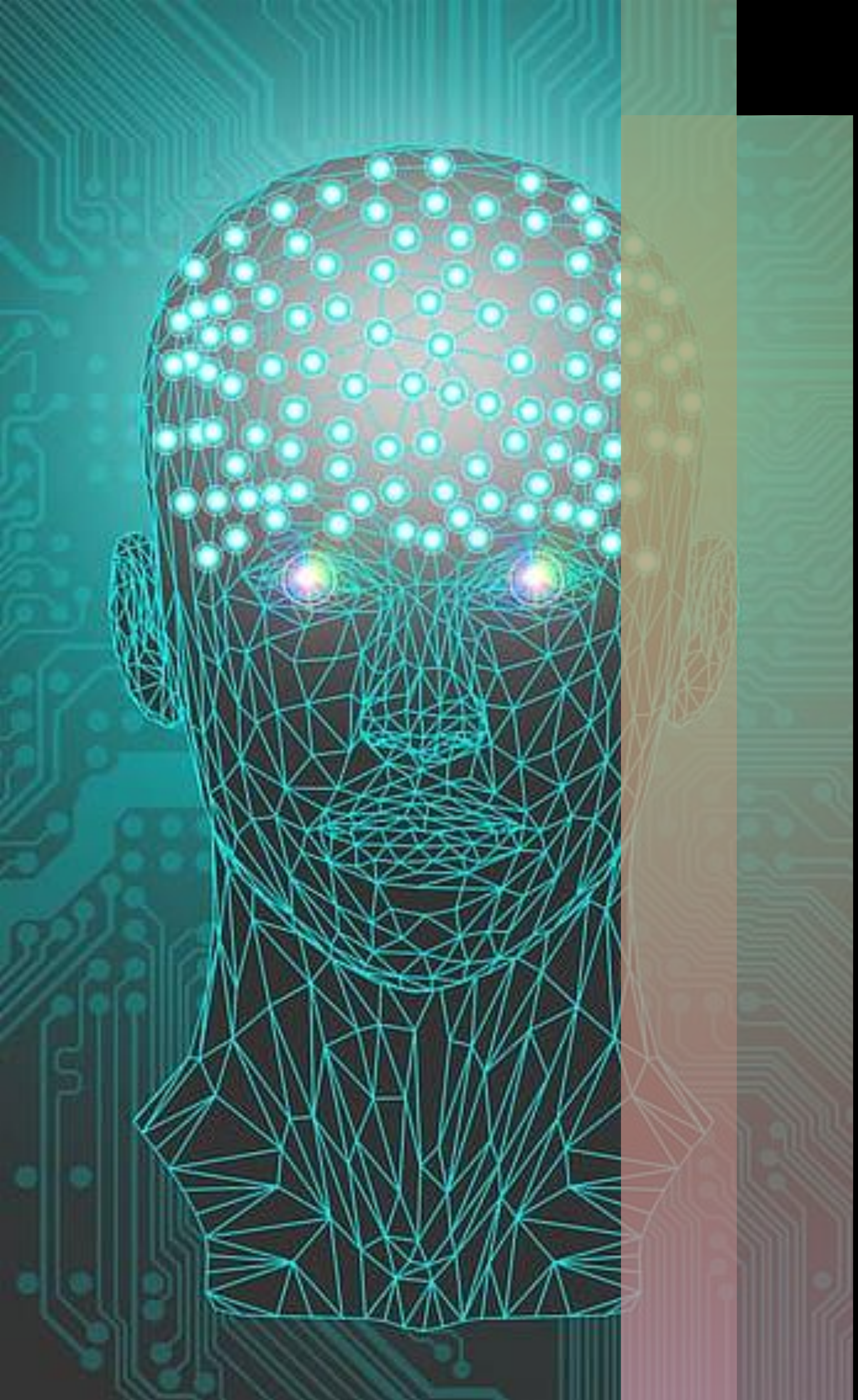
AND (∧)

- This is every permutation of the **AND** statement.

  Let p be it is sunny today.
  Let q be it is snowing today.

- Using truth table, we can predict whether the **conjunction** p ∧ q will **evaluate** to true or false for the above example
- Let's say p is true and q is false (red in below truth table)
- For the conjunction of p ∧ q to evaluate to true, **both** p and q must be true, thus p ∧ q evaluates to false
- This logic is the same when writing conditions in code too...

p	**q**	**p ∧ q**
T	T	T
T	F	F
F	T	F
F	F	F



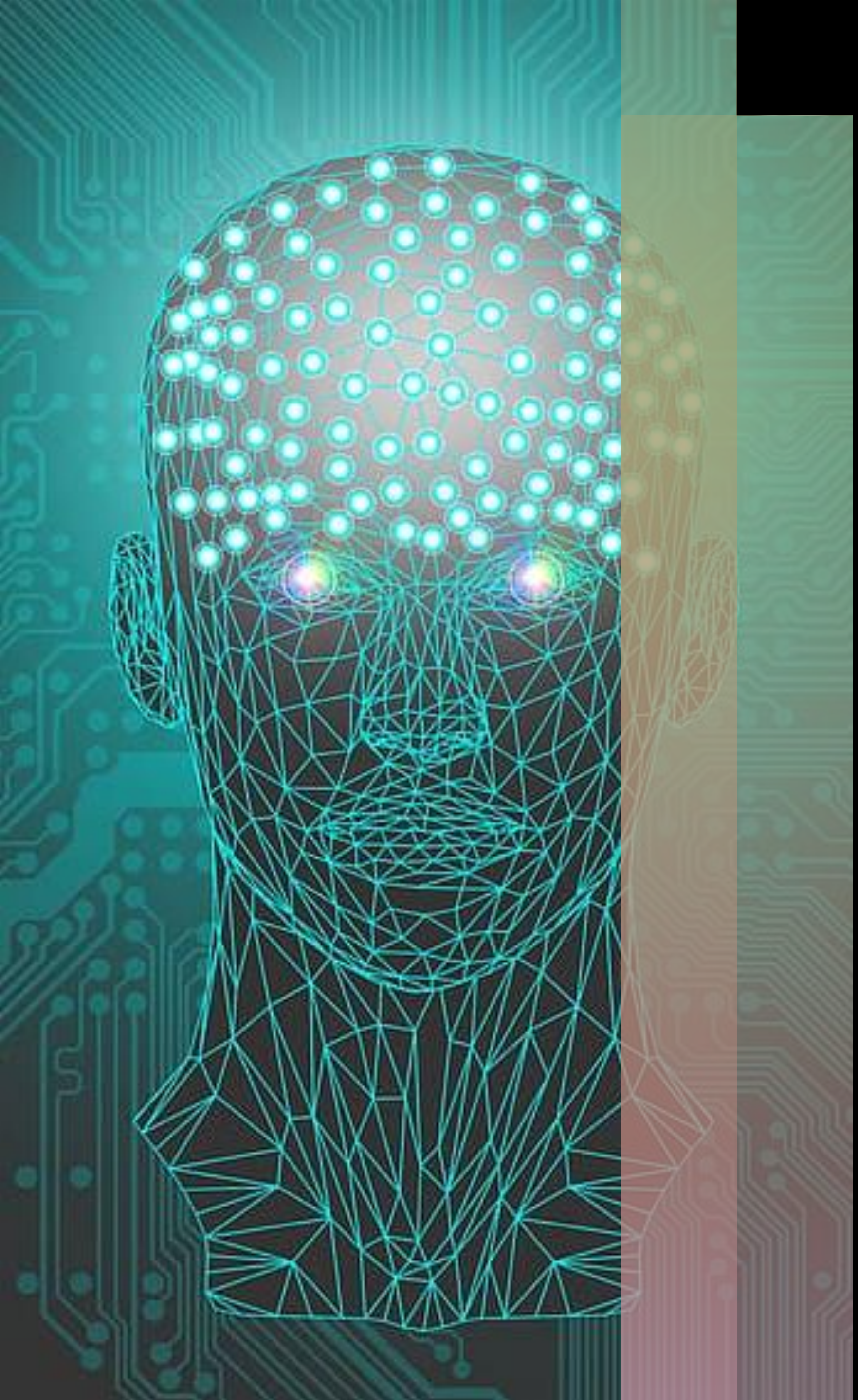
OR (∨)

- Any two propositions can be combined with an **"or"** to form a third proposition called the **disjunction** of the originals
- If p and q are arbitrary propositions, then the disjunction of p and q is written as:

$$p \vee q$$

- and will be true if either p or q are true, or both p and q are true
- Again, the same is true for the conditions (or better identified as you write in your code…

p	**q**	**p ∨ q**
T	T	T
T	F	T
F	T	T
F	F	F



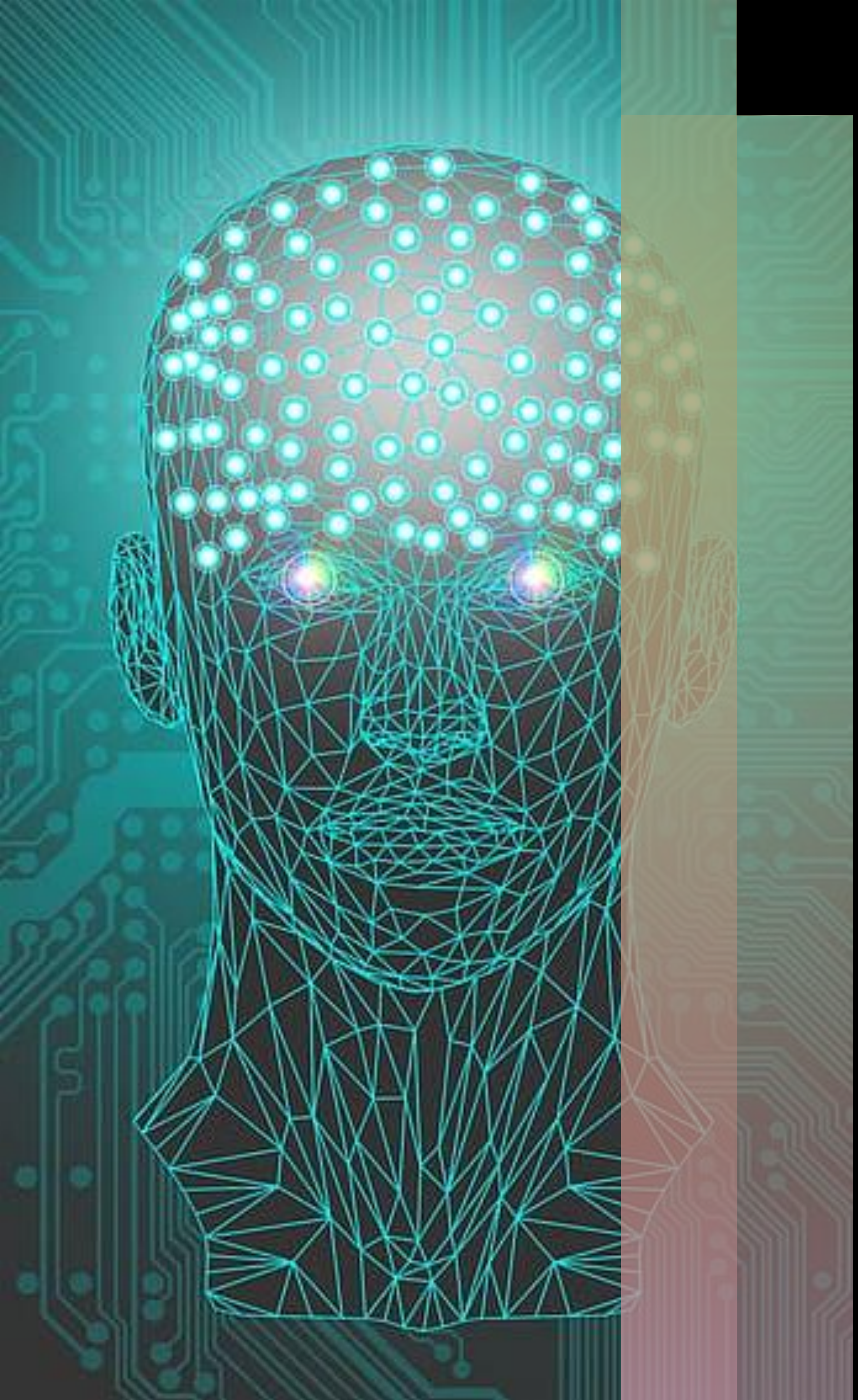
IF... THEN... (⇒)

- Many statements are of the form, If p is true, then q is true
- Another way of saying the same thing is to write, p implies q
- In propositional logic, we have a connective that combines two propositions into a new proposition called the **conditional**, or **implication** of the originals
- If p and q are arbitrary propositions, then the conditional of p and q is written as:

$$p \vee q$$

- and will be true if either p is false or q is true

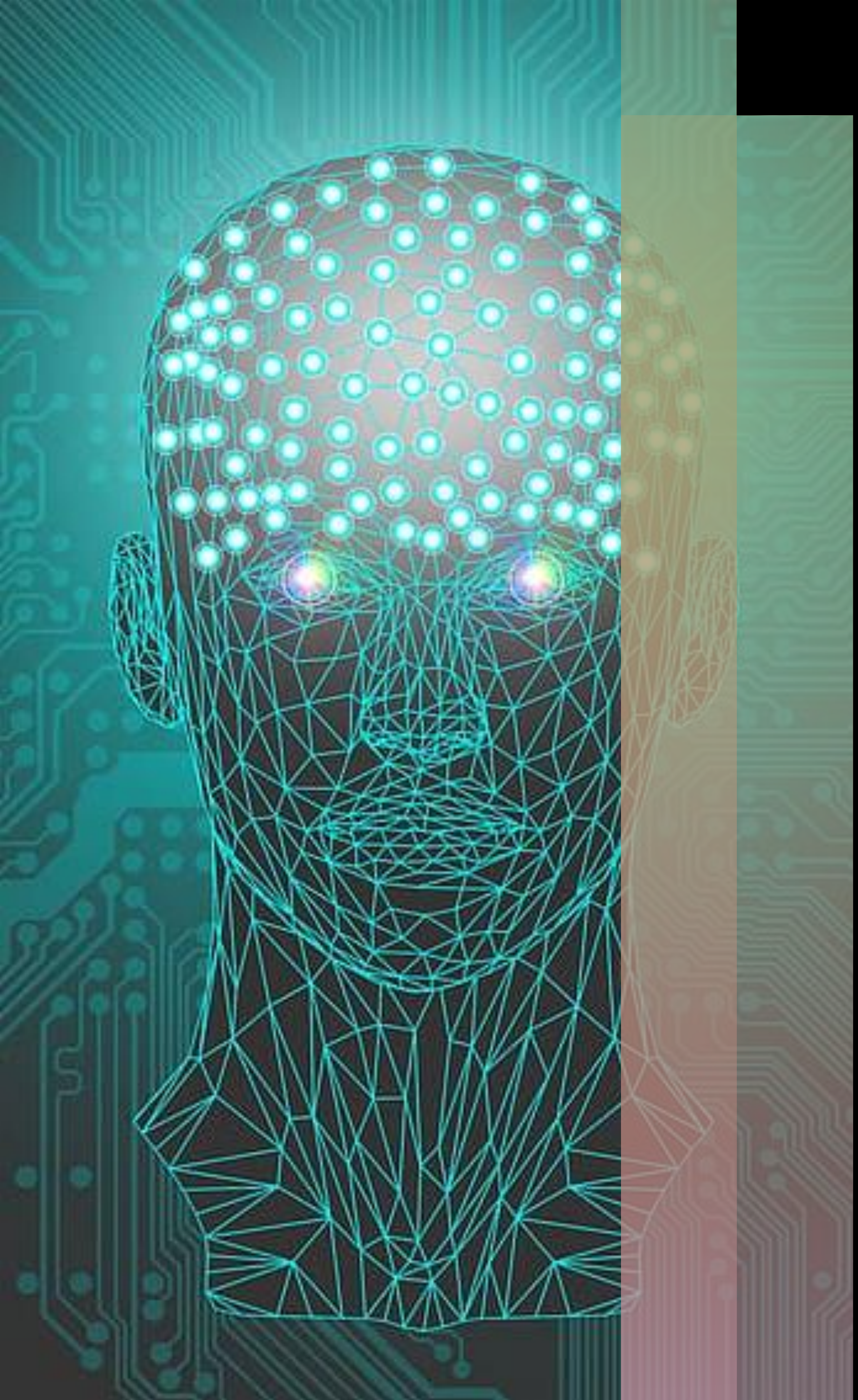
p	q	p ∨ q
T	T	T
T	F	F
F	T	T
F	F	T



IF… THEN… (⇒)

- The ⇒ operator is the hardest to understand of the operators we have considered so far, and yet it is extremely important
- If you find it difficult to understand, just remember that the p ⇒ q means ‘if p is true, then q is true
- If p is false, then we don’t care about q, thus by default p ⇒ q evaluates to true

- As p is implying q, q cannot be false if p is true



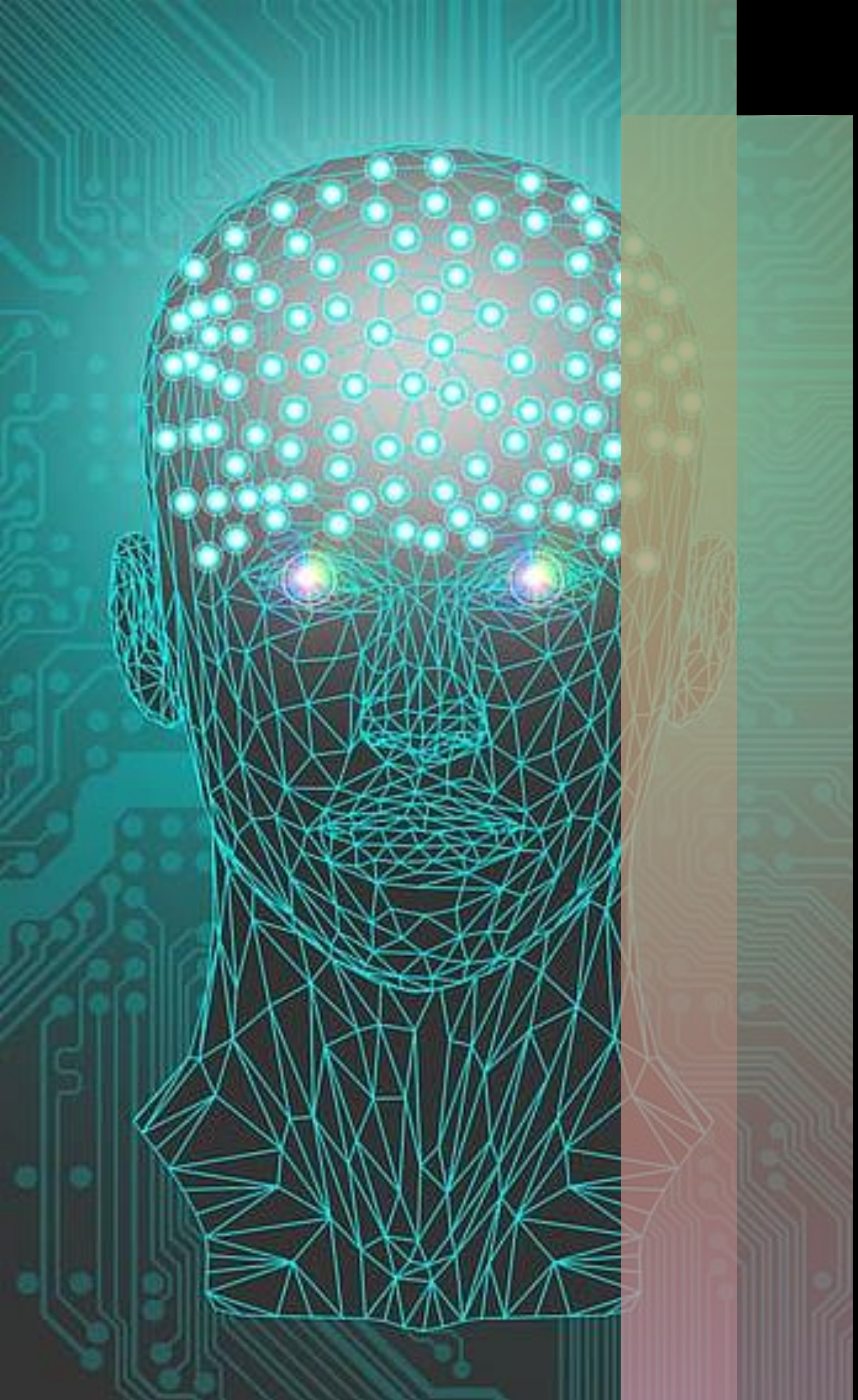
IF AND ONLY IF - IFF (⇔)

- Many statements are of the form, p is true if, and only if, q is true
- The sense of such statements is captured using the **biconditional** operator.
- If p and q are arbitrary propositions, then the biconditional of p and q is written as:

$$p \Leftrightarrow q$$

- and will be true iff either:
- p and q are both false; or
- p and q are both true

p	**q**	**p ⇔ q**
T	T	T
T	F	F
F	T	F
F	F	T



IF AND ONLY IF - IFF (⇔)

- If p ⇔ q is true, then p and q are said to be *logically equivalent*. They will be true under exactly the same circumstances.

Let p be it is raining

Let q be there is rain

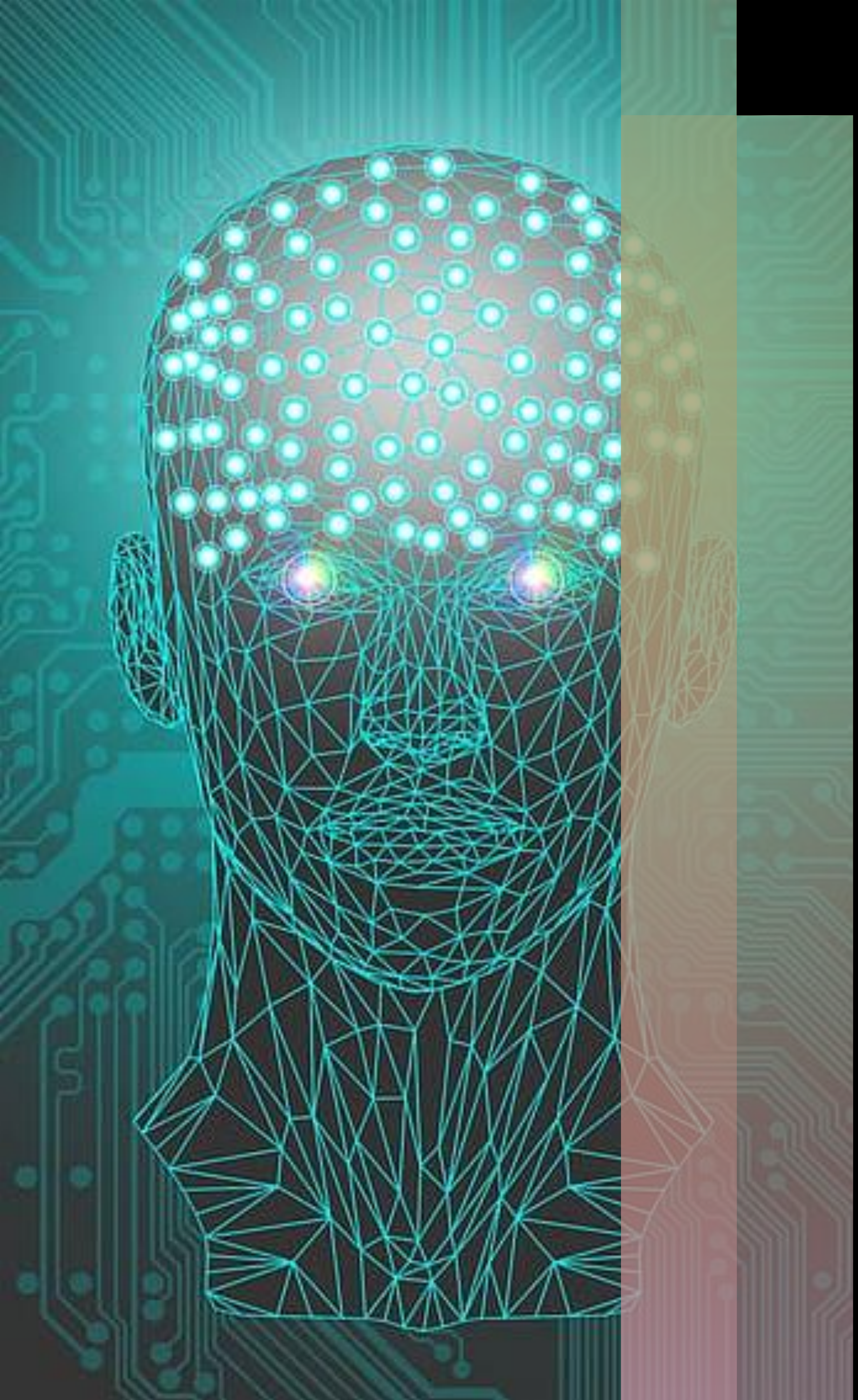
Let's say in England that you can't have rain without wind. So, if it's raining then there will be wind.

i.e., if it rains, then there is wind | p implies q | p ⇒ q

If the case also existed that if it was windy that there would also be windy then the biconditional can be used. Same for false

i.e., it is raining if and only if it is windy | p iff q | p ⇔ q

it is not raining if and only if it is not windy | p iff q | p ⇔ q



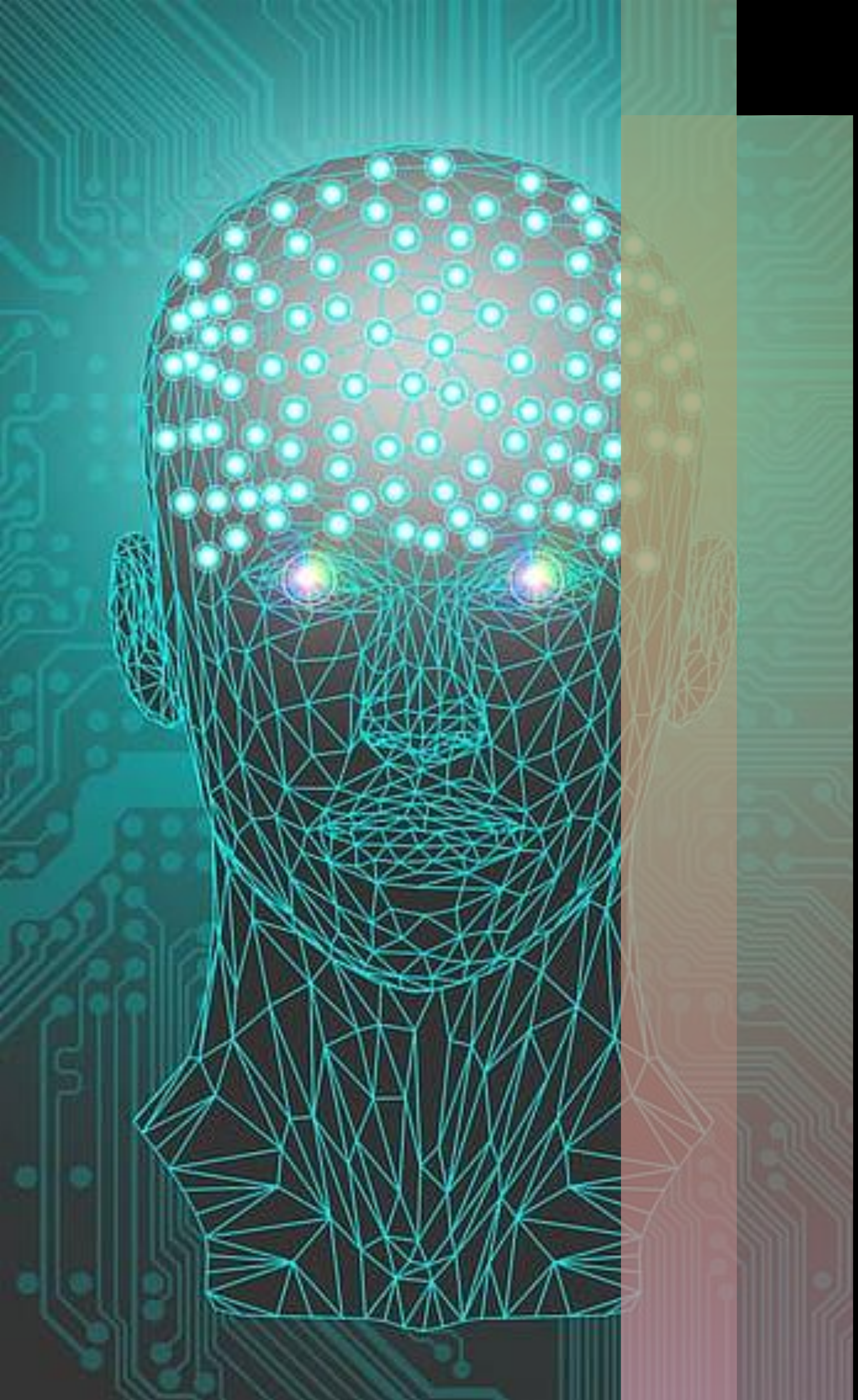
NOT (¬)

- All the connectives we have considered so far have been binary: they have taken **two arguments**
- The final connective we consider here is **unary**. It only takes **one argument**
- Any proposition can be prefixed by the word ‘not’ to form a second proposition called the **negation** of the original
- If p is an arbitrary proposition, then the negation of p is written as:

$$\neg p$$

- and will be true iff p is false (it is usually the opposite of whatever you are negating)

p	**¬p**
F	T
T	F

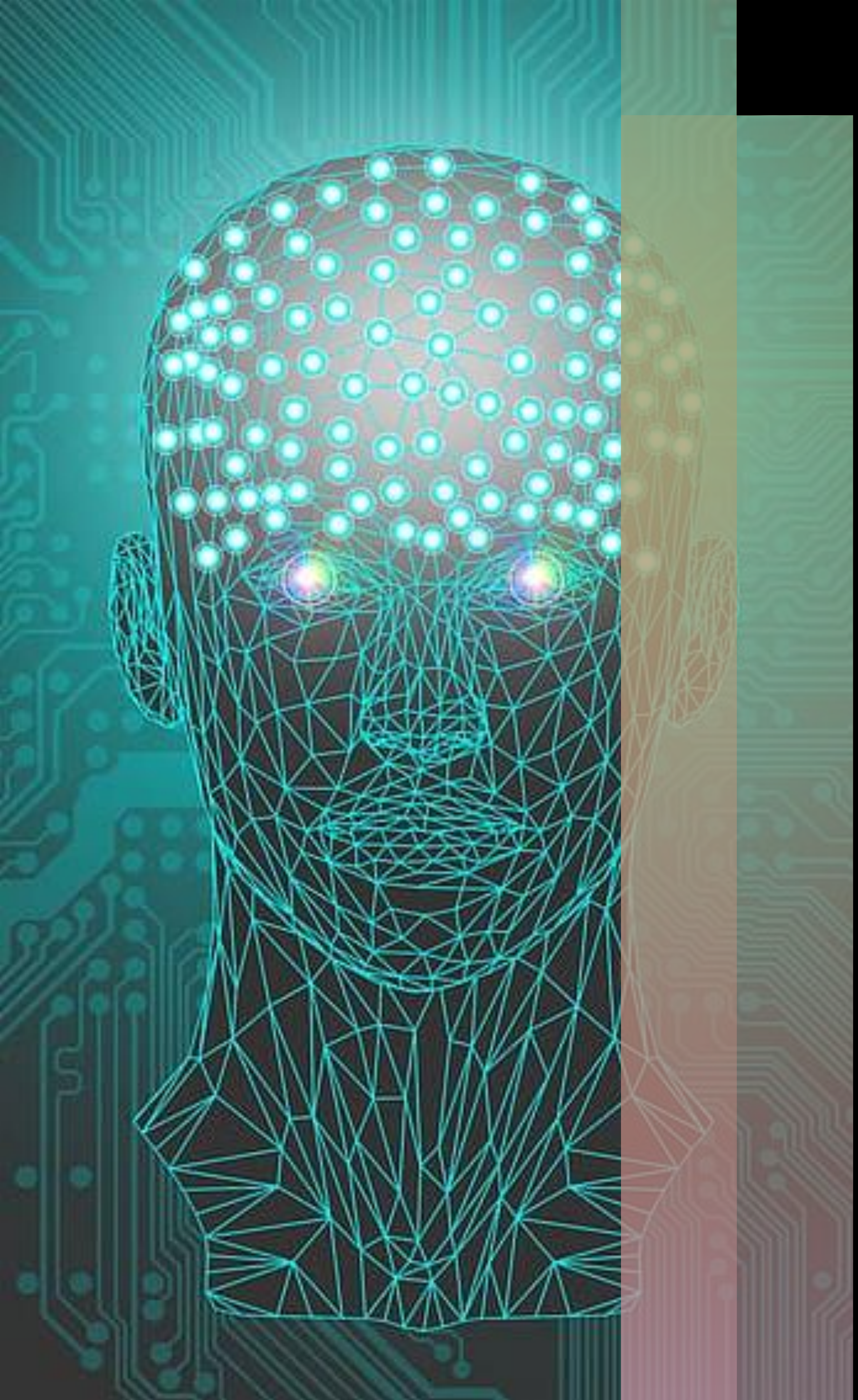


Summary Comments

- We can nest complex formulae as deeply as we want to
- We rely on parenthesis () to **disambiguate** and understand these complex propositions
- You are, and have been, using all these logical operators within your code (including **not** | !)

$$p \wedge q \Rightarrow r$$
$$p \wedge (q \Rightarrow r)$$
$$(p \wedge (q \Rightarrow r)) \vee s$$
$$((p \wedge (q \Rightarrow r)) \vee s) \wedge t$$

- How many permutations | combination of answers are possible from the fourth complex proposition?

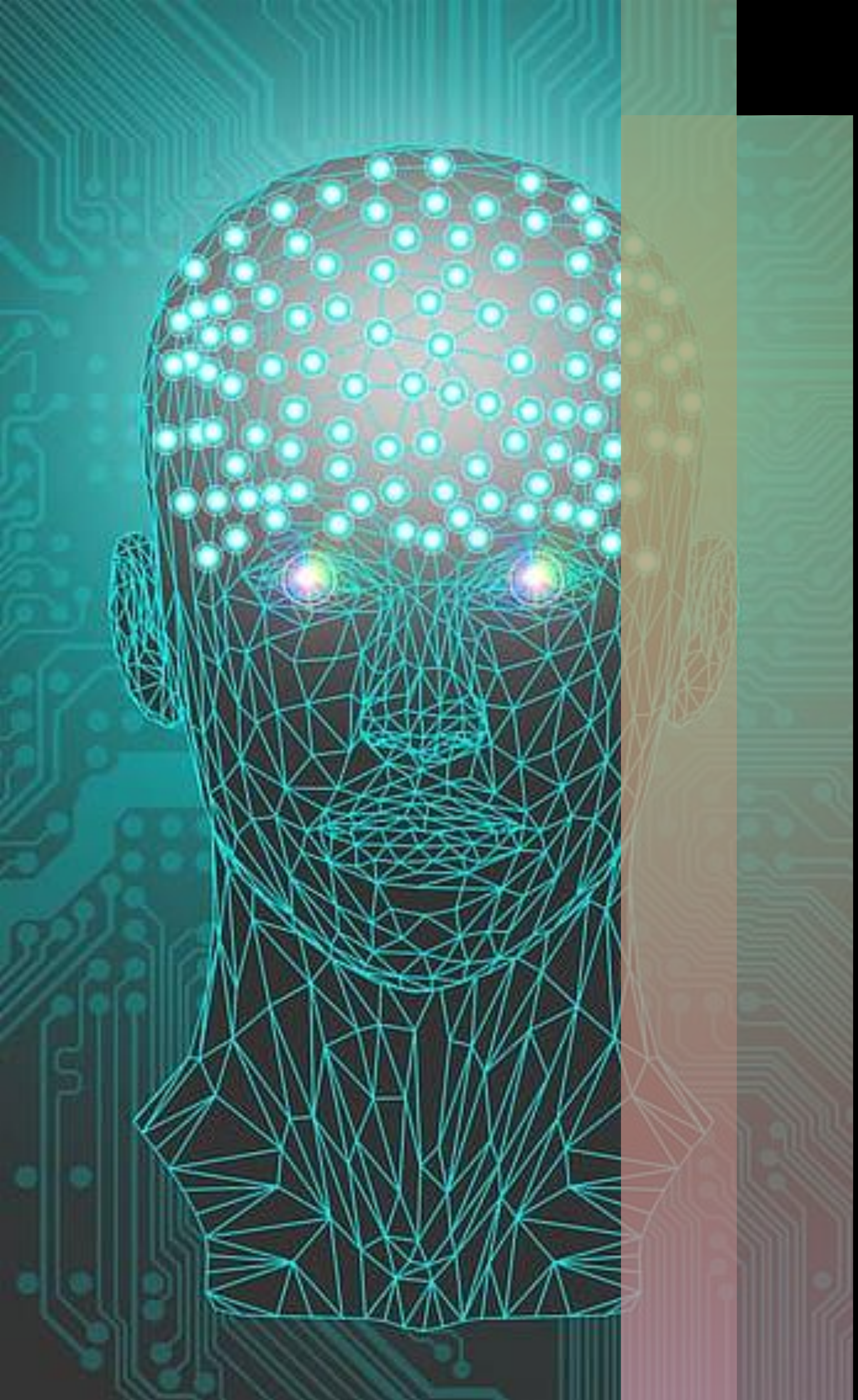


Valuation

- When faced with complex formula, it is difficult to discern if they will evaluate to true or false (the complex propositions below fit within this category)
- Can you tell what they will evaluate to?

$$p \wedge q \Rightarrow r$$
$$p \wedge (q \Rightarrow r)$$
$$(p \wedge (q \Rightarrow r)) \vee s$$
$$((p \wedge (q \Rightarrow r)) \vee s) \wedge t$$

- We must perform a **valuation** of the individual components of the complex propositions
- Only once we have conducted the valuation for a combination can, we say for certain that it evaluate as expected



Valuation 1

- Let's valuate the following complex proposition

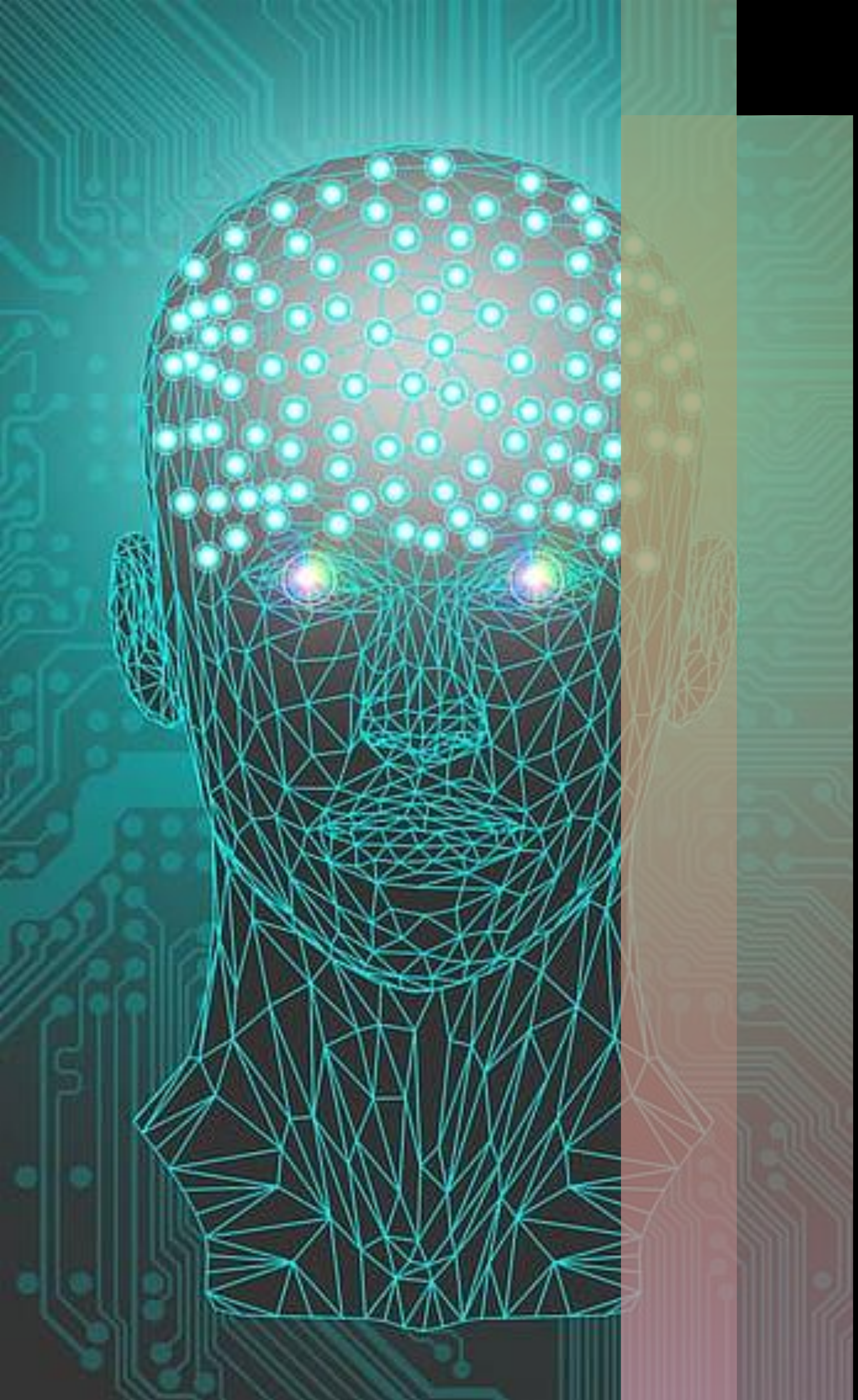
$$(p \land (q \Rightarrow r)) \lor \neg q$$

- We must first discern the number of combinations
- As there are 4 propositions (p, q, r, and s) we must do 2^3 which is 2*2*2 = 8

- Creating a binary (T|1 || F|0) table might initially appear challenging, but here's a helpful strategy:

- Begin by populating the first row entirely with Ts and the last row with Fs.
- Next, fill the upper half of the first column with Ts and the lower half with Fs.
- Proceed by alternating between Ts and Fs in half of the next columns.

p	q	r
T	T	T
T	T	F
T	F	T
T	F	F
F	T	T
F	T	F
F	F	T
F	F	F



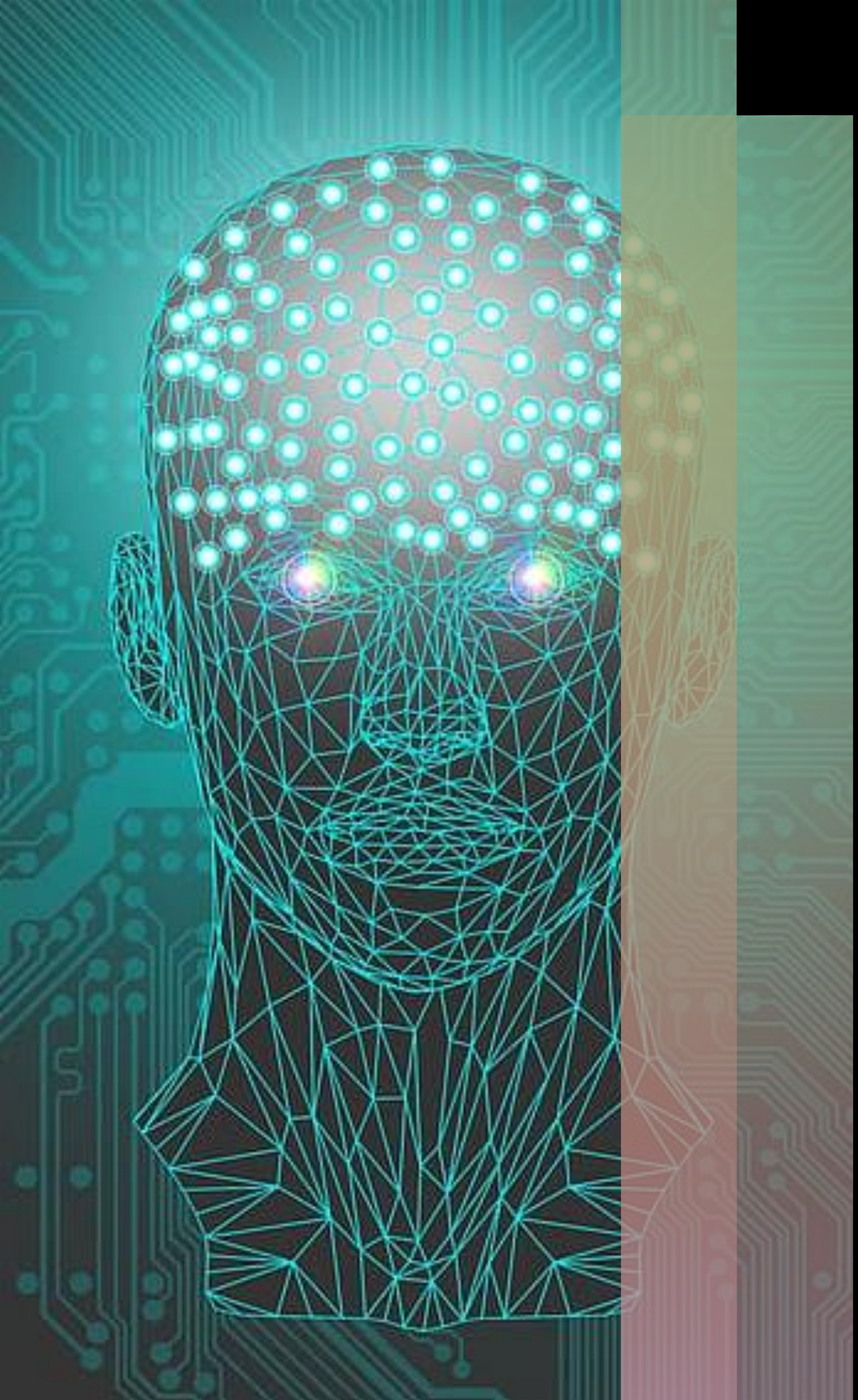
Valuation 2

- Let's valuate the following complex proposition

$$(p \wedge (q \Rightarrow r)) \vee \neg q$$

- We currently have the values for p so let's look at $(q \Rightarrow r)$
- If q is true and r is false, the implication "q implies r" evaluates to false

p	q	r	$(q \Rightarrow r)$
T	T	T	T
T	T	F	F
T	F	T	T
T	F	F	T
F	T	T	T
F	T	F	F
F	F	T	T
F	F	F	T



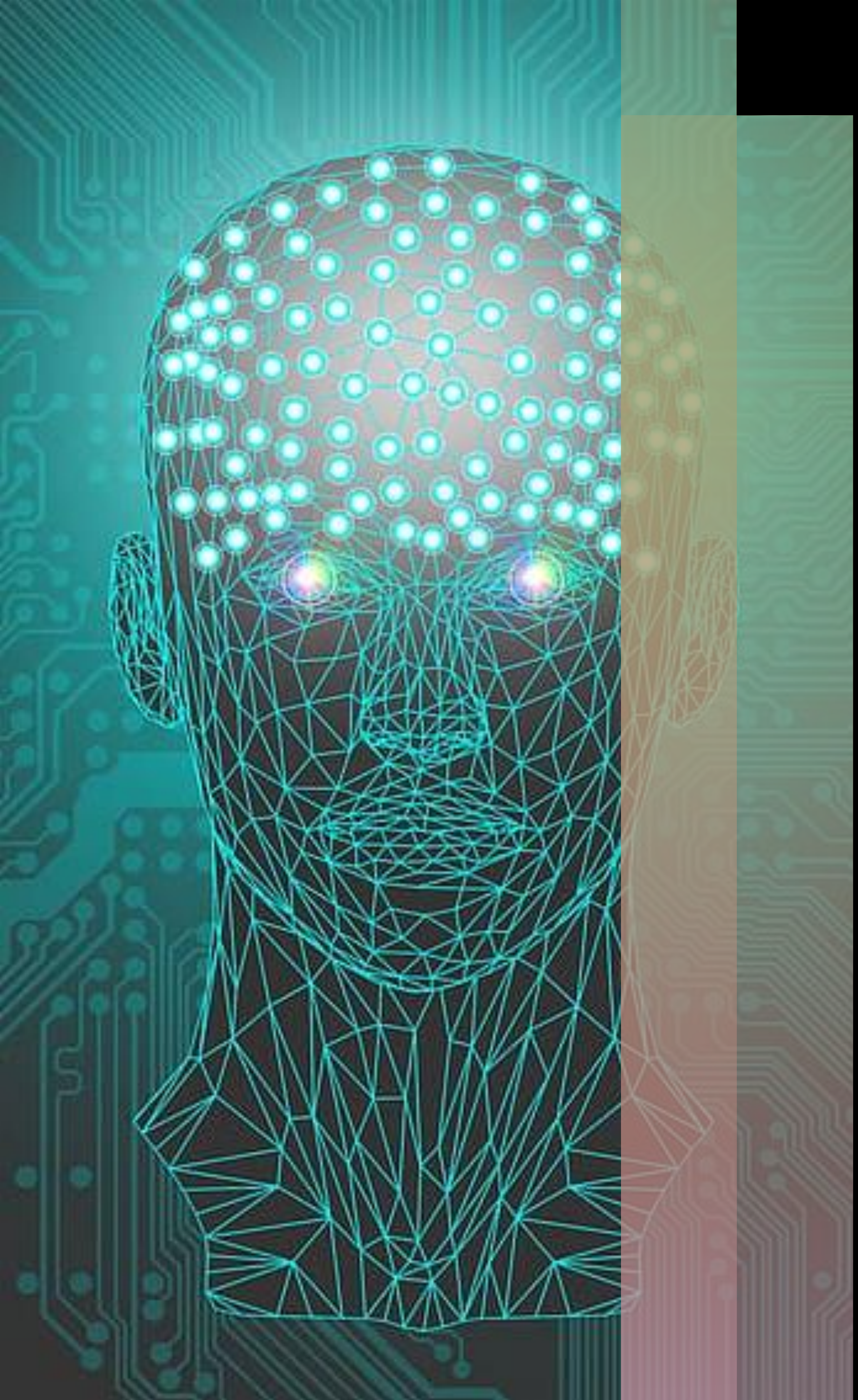
Valuation 3

- Let's valuate the following complex proposition

$$(p \land (q \Rightarrow r)) \lor \neg q$$

- We currently have the values for p and (q ⇒ r)
- It's time that we find the conjunction of these propositions.
- Will be true, if both p and (q ⇒ r) are true

p	q	r	(q ⇒ r)	(p ∧ (q ⇒ r))
T	T	T	T	T
T	T	F	F	F
T	F	T	T	T
T	F	F	T	T
F	T	T	T	F
F	T	F	F	F
F	F	T	T	F
F	F	F	T	F



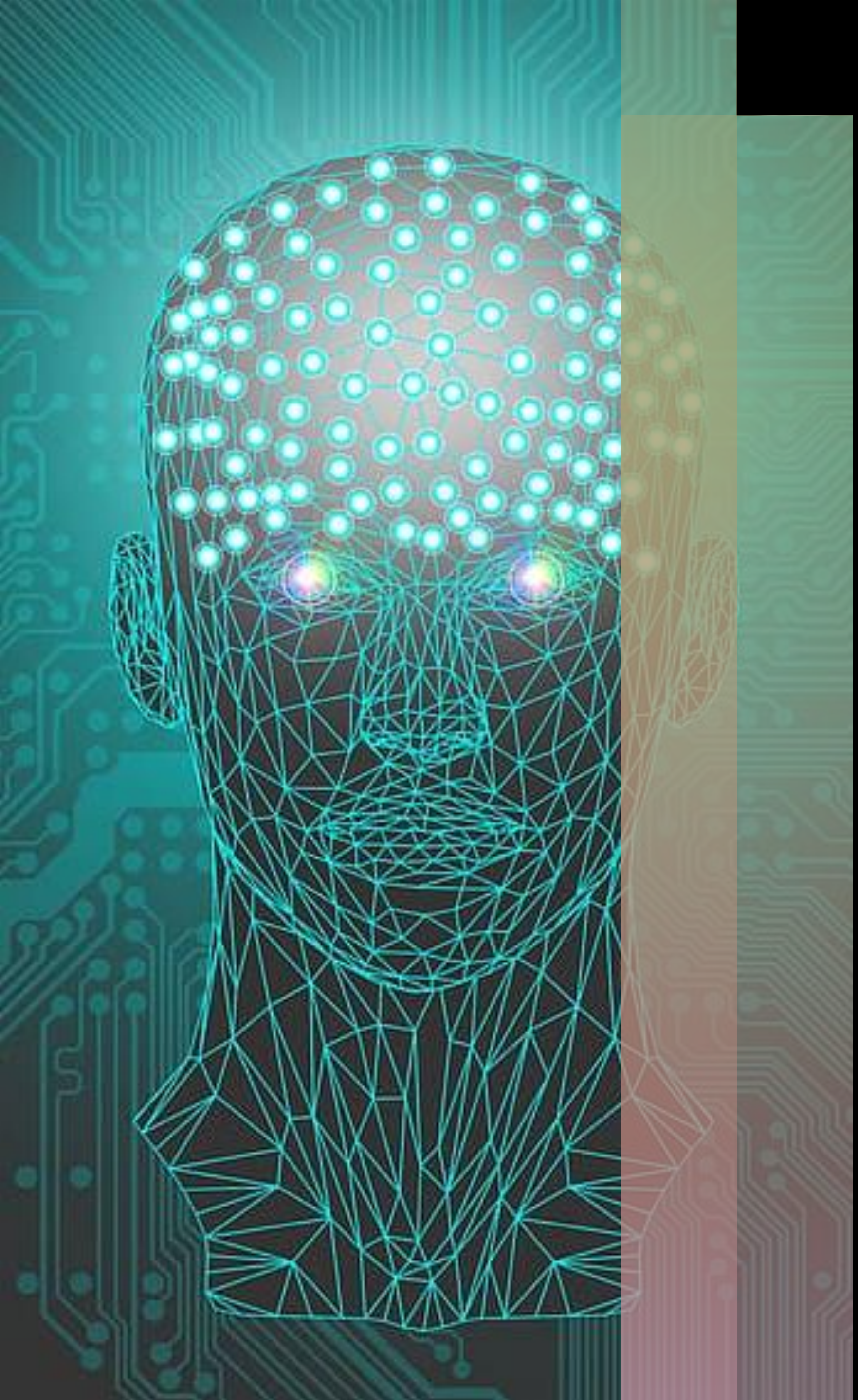
Valuation 4

- Let's valuate the following complex proposition

$$(p \wedge (q \Rightarrow r)) \vee \neg q$$

- We currently have the values for $(p \wedge (q \Rightarrow r))$
- It's time that we find the negation of p
- Will be true, if p is false and (vice versa)

p	q	r	$(q \Rightarrow r)$	$(p \wedge (q \Rightarrow r))$	$\neg q$
T	T	T	T	T	F
T	T	F	F	F	F
T	F	T	T	T	T
T	F	F	T	T	T
F	T	T	T	F	F
F	T	F	F	F	F
F	F	T	T	F	T
F	F	F	T	F	T



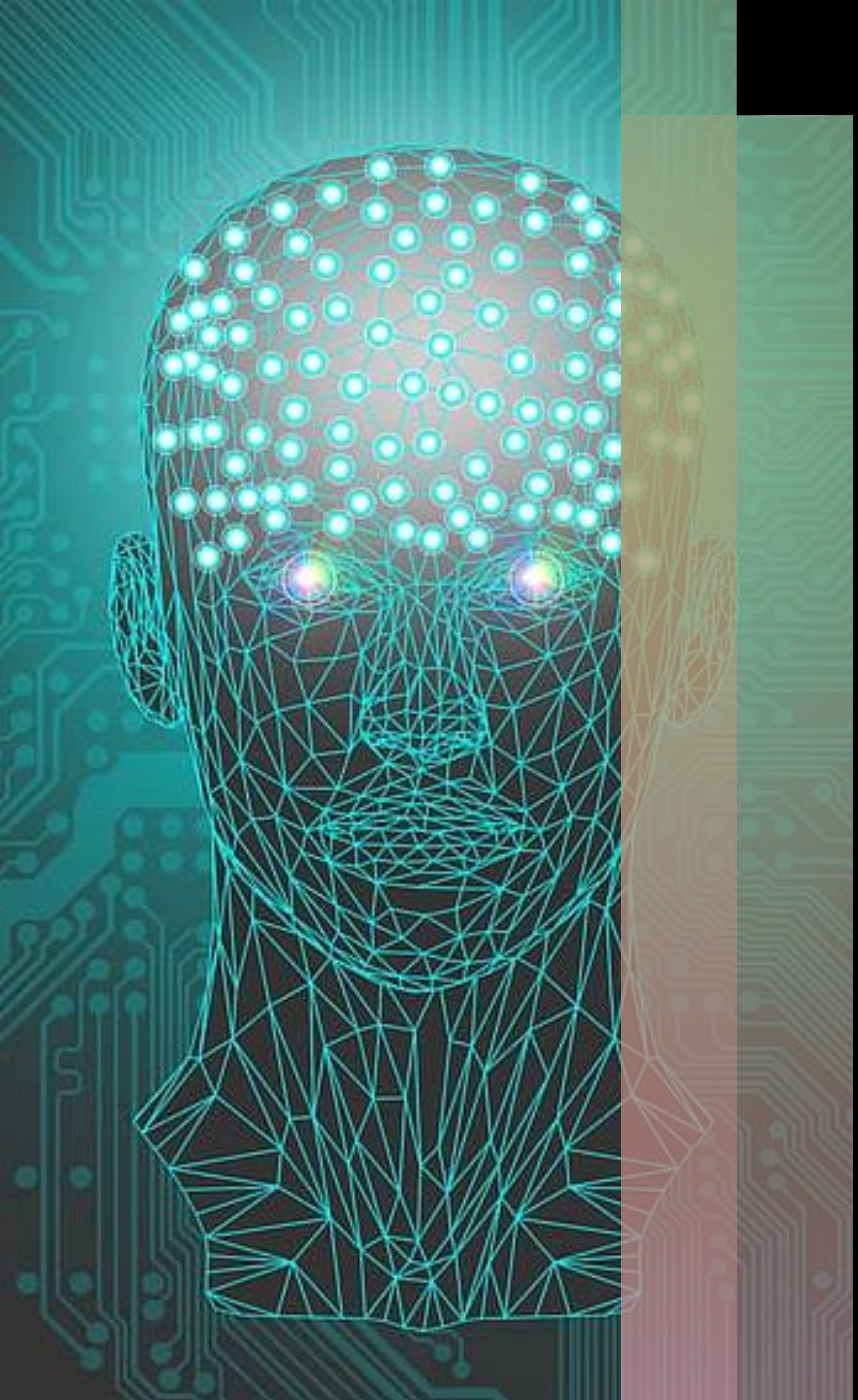
Valuation 5

- Let's valuate the following complex proposition

(p ∧ (q ⇒ r)) ∨ ¬q

- We currently have the values for (p ∧ (q ⇒ r)) and ¬q
- It's time that we find the disjunction of both
- It will be true, if one, or both statements are true

p	q	r	(q ⇒ r)	(p ∧ (q ⇒ r))	¬q	(p ∧ (q ⇒ r)) ∨ ¬q
T	T	T	T	T	F	T
T	T	F	F	F	F	F
T	F	T	T	T	T	T
T	F	F	T	T	T	T
F	T	T	T	F	F	F
F	T	F	F	F	F	F
F	F	T	T	F	T	T
F	F	F	T	F	T	T



Tautology

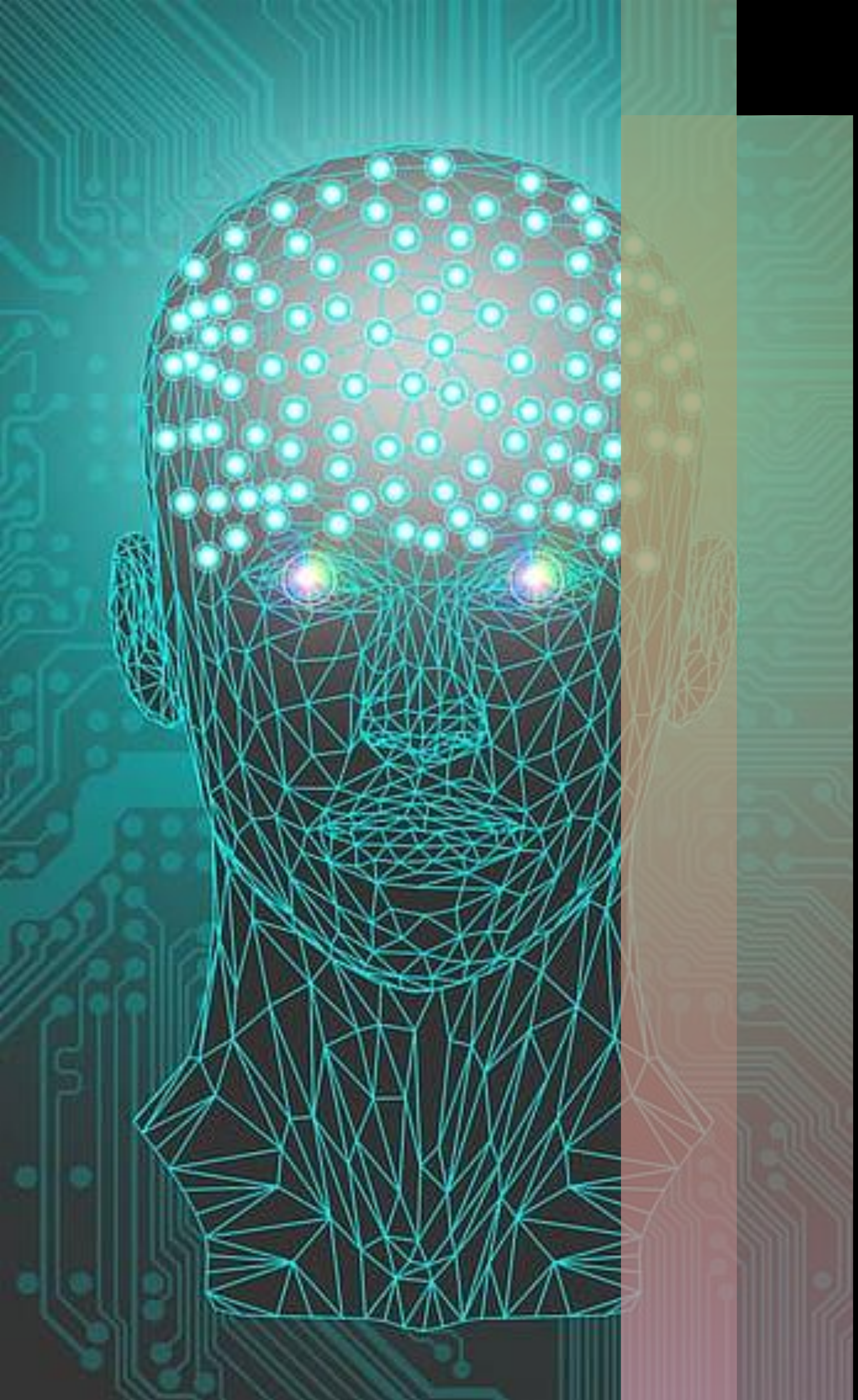
- Let's valuate the following complex proposition

$$(p \land (p \Rightarrow q)) \Rightarrow q$$

- As you can see every valuation, permutation, and combination of the following complex proposition evaluates to true
- This is an example of a **tautology**

p	q	$(p \Rightarrow q)$	$(p \land (p \Rightarrow q))$	$(p \land (p \Rightarrow q)) \Rightarrow q$
T	T	T	T	T
T	F	F	F	T
F	T	T	F	T
F	F	T	F	T

- If at least one of the valuations evaluates to true, then we can describe the formula as **consistent**
- If all evaluate to false, then the formula is described as **inconsistent**



Your Turn

- Valuate the following complex proposition

$$p \lor q$$

$$(p \land (p \Rightarrow q)) \Leftrightarrow q$$

$$(p \land (p \Rightarrow q)) \Leftrightarrow \neg(q \lor \neg r)$$

Summary

We've covered quite a few topics today:

- Introducing propositions, Propositional Logic, and tautologies

Any questions?