

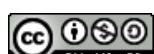
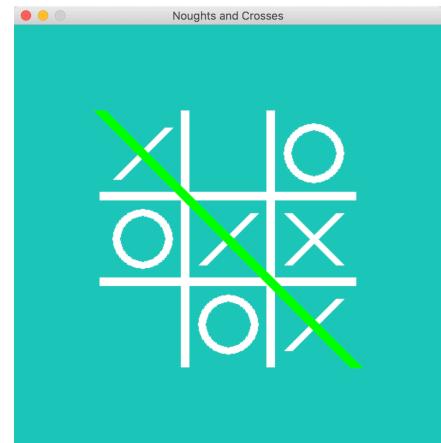
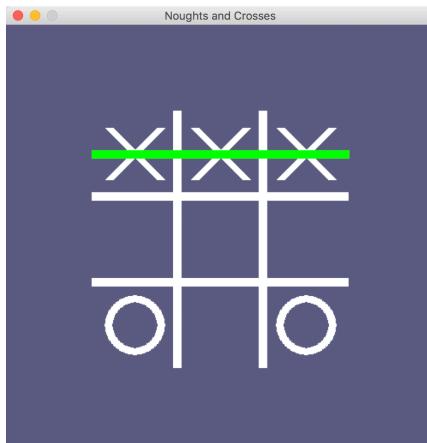
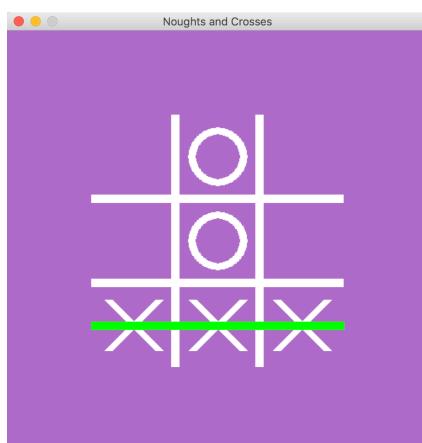
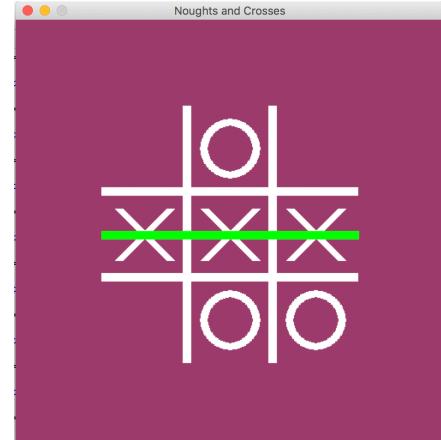
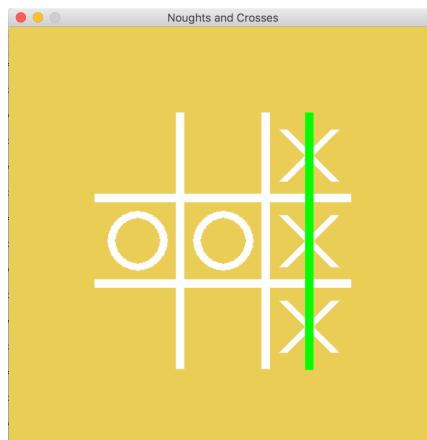
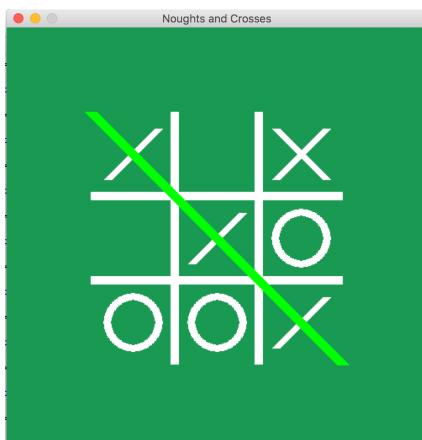


# Noughts & Crosses

In this project you will create a noughts and crosses game in Python that is able to learn from how you play.

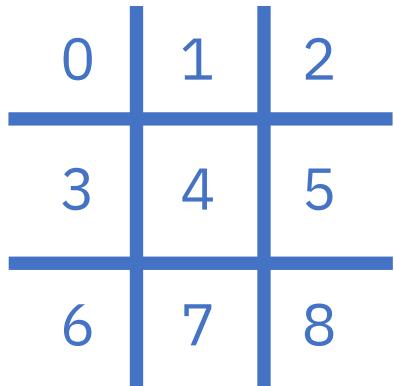
You won't give it instructions for how to play, or tell it what the objective or rules of the game are.

Instead, you'll show it examples of you playing the game. When it's seen enough examples to start trying to play for itself, you'll tell when it beats you.



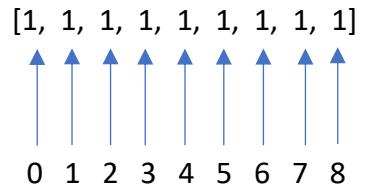
This project worksheet is licensed under a Creative Commons Attribution Non-Commercial Share-Alike License  
<http://creativecommons.org/licenses/by-nc-sa/4.0/>

# Representing noughts and crosses in Python



The positions of spaces on the noughts and crosses board are numbered from 0 to 8.

They are then stored in a list.



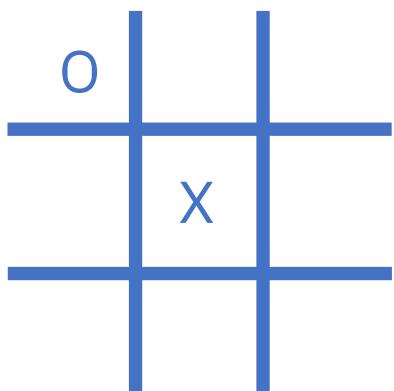
---

Empty = 1  
O = 2  
X = 3

An empty space is shown as a 1.

A nought O is shown as a 2.

A cross X is shown as a 3.



The board is represented as a list of 9 1's, 2's and 3's, one for each space

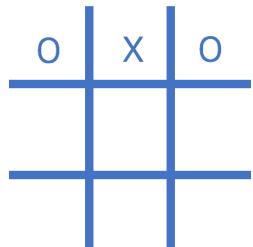
For example, at this point:

[2, 1, 1, 1, 3, 1, 1, 1, 1]

## What are you going to do?

You're going to train a computer to play noughts and crosses. You'll do this by showing it examples of how you play the game.

Imagine the board looks like this and it's X's turn.

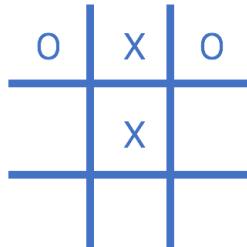


Imagine you decide to put your X in the centre space.

|               |          |
|---------------|----------|
| top-left      | opponent |
| top-middle    | player   |
| top-right     | opponent |
| middle-left   | empty    |
| middle-middle | empty    |
| middle-right  | empty    |
| bottom-left   | empty    |
| bottom-middle | empty    |
| bottom-right  | empty    |

choice : middle-middle

Imagine the board looks like this and it's O's turn.



Imagine you decide to put your O in the bottom middle space.

|               |          |
|---------------|----------|
| top-left      | player   |
| top-middle    | opponent |
| top-right     | player   |
| middle-left   | empty    |
| middle-middle | opponent |
| middle-right  | empty    |
| bottom-left   | empty    |
| bottom-middle | empty    |
| bottom-right  | empty    |

choice : bottom-middle

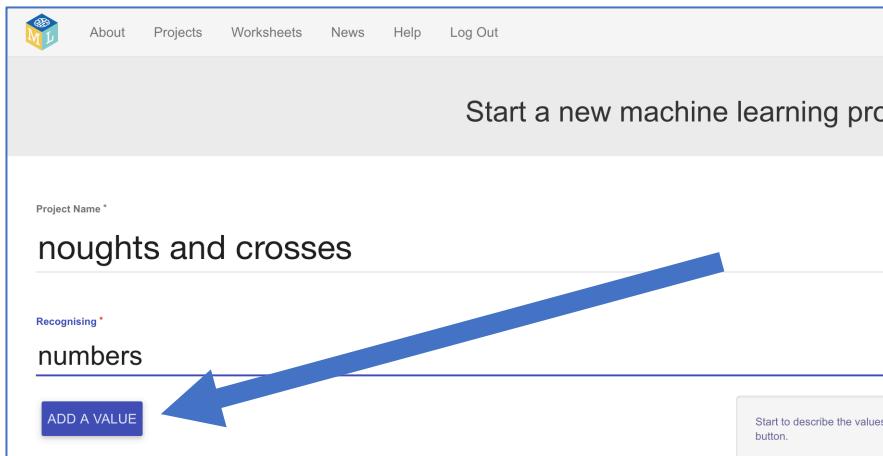
Using “opponent” and “player” instead of “nought” and “cross” means the computer can learn from both nought and cross moves.

You'll use examples of moves from the player that wins the game to train the computer.

If you (X) win, you'll use your moves as examples to train the computer. If the computer (O) wins, you'll use the computer's moves to train with.

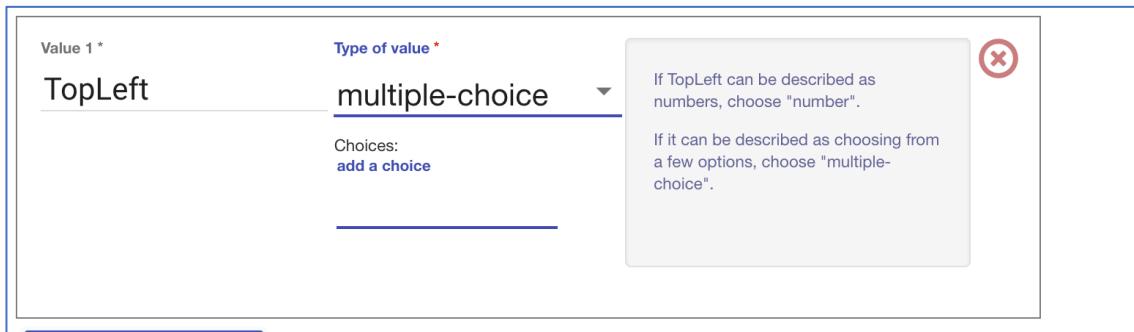
These **examples of moves that lead to winning** will teach the computer how to play to win!

1. Go to <https://machinelearningforkids.co.uk> in a browser.
2. Click on “**Get started**”
3. Click on “**Log In**” and type in your username and password  
*If you don't have a username, ask your teacher to create one for you.*  
*If you can't remember your password, ask your teacher to reset it for you.*
4. Click on “**Projects**” on the top menu bar
5. Click on the “**+ Add a new project**” button.
6. Name your project “**noughts and crosses**” and set it to learn how to recognise “**numbers**”
7. Click “**Add a value**”



The screenshot shows a web interface for creating a machine learning project. At the top, there's a navigation bar with links for About, Projects, Worksheets, News, Help, and Log Out. Below that, a large button says "Start a new machine learning project". The main area is titled "noughts and crosses" under "Project Name". Under "Recognising", the word "numbers" is listed. At the bottom left is a blue button labeled "ADD A VALUE". A blue arrow points from the "Recognising" field towards this button. On the right side, there's a note: "Start to describe the values button."

8. Name a value “**TopLeft**” and make it “**multiple-choice**”



The screenshot shows a configuration page for a value. It has fields for "Value 1" (containing "TopLeft") and "Type of value" (set to "multiple-choice"). Below the type field, there's a "Choices" section with a link to "add a choice". To the right, a tooltip provides guidance: "If TopLeft can be described as numbers, choose "number". If it can be described as choosing from a few options, choose "multiple-choice"."

- 9.** Type “EMPTY” into the “add a choice” box and press Enter  
 Type “PLAYER” into the “add a choice” box and press Enter  
 Type “OPPONENT” into the “add a choice” box and press Enter  
*These are the possible contents for the top-left space in the noughts and crosses board. It can be empty, or it can have the player’s own shape (cross) in it, or it can have the opponent’s shape in it (nought).*

Value 1 \* **TopLeft**

Type of value \* **multiple-choice**

Choices:

- EMPTY
- PLAYER
- OPPONENT

[add a choice](#)

Type in another choice to use in your multiple-choice list, then press Enter.

- 10.** Click “Add another value”

Project Name \* **noughts and crosses**

Recognizing \* **numbers**

Value 1 \* **TopLeft** Type of value \* **multiple-choice**

Choices:

- EMPTY
- PLAYER
- OPPONENT

[add a choice](#)

**ADD ANOTHER VALUE**

**CREATE** **CANCEL**

- 11.** Call the next value “**TopMiddle**” and make it “**multiple-choice**”

Project Name \* **noughts and crosses**

Recognizing \* **numbers**

Value 1 \* **TopLeft** Type of value \* **multiple-choice**

Choices:

- EMPTY
- PLAYER
- OPPONENT

[add a choice](#)

Value 2 \* **TopMiddle** Type of value \* **multiple-choice**

Choices:

- [add a choice](#)

If TopMiddle can be described as numbers, choose “number”.  
 If it can be described as choosing from a few options, choose “multiple-choice”.

**ADD ANOTHER VALUE**

**12.** Add “EMPTY”, “PLAYER”, and “OPPONENT” to “TopMiddle” as you did to “TopLeft”

**13.** Repeat for the other positions on a noughts-and-crosses board  
*Each example is the state of the board before a move that led to a win.*  
TopLeft, TopMiddle, TopRight,  
MiddleLeft, MiddleMiddle, MiddleRight,  
BottomLeft, BottomMiddle, BottomRight

**It's very important that you spell “EMPTY”, “PLAYER” and “OPPONENT” in the same way for all nine positions, and put them in the same order.**

**14.** Click “Create”

Project name: noughts and crosses

Recognizing \*

numbers

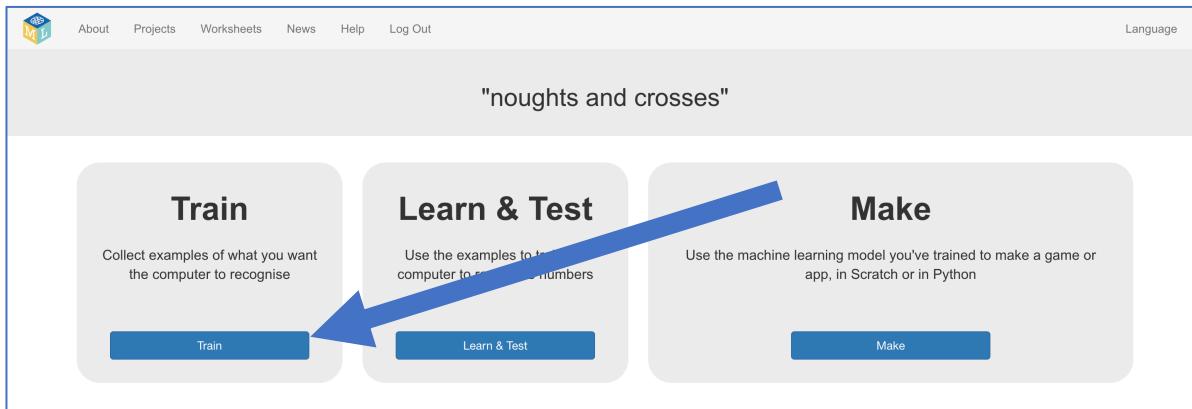
|  |  |   |
|--|--|---|
| Value 1 *<br>TopLeft<br>Type of value *<br>multiple-choice<br>Choices:<br>EMPTY ⚡<br>PLAYER ⚡<br>OPPONENT ⚡    | Value 2 *<br>TopMiddle<br>Type of value *<br>multiple-choice<br>Choices:<br>EMPTY ⚡<br>PLAYER ⚡<br>OPPONENT ⚡    | Value 3 *<br>TopRight<br>Type of value *<br>multiple-choice<br>Choices:<br>EMPTY ⚡<br>PLAYER ⚡<br>OPPONENT ⚡    |
| Value 4 *<br>MiddleLeft<br>Type of value *<br>multiple-choice<br>Choices:<br>EMPTY ⚡<br>PLAYER ⚡<br>OPPONENT ⚡ | Value 5 *<br>MiddleMiddle<br>Type of value *<br>multiple-choice<br>Choices:<br>EMPTY ⚡<br>PLAYER ⚡<br>OPPONENT ⚡ | Value 6 *<br>MiddleRight<br>Type of value *<br>multiple-choice<br>Choices:<br>EMPTY ⚡<br>PLAYER ⚡<br>OPPONENT ⚡ |
| Value 7 *<br>BottomLeft<br>Type of value *<br>multiple-choice<br>Choices:<br>EMPTY ⚡<br>PLAYER ⚡<br>OPPONENT ⚡ | Value 8 *<br>BottomMiddle<br>Type of value *<br>multiple-choice<br>Choices:<br>EMPTY ⚡<br>PLAYER ⚡<br>OPPONENT ⚡ | Value 9 *<br>BottomRight<br>Type of value *<br>multiple-choice<br>Choices:<br>EMPTY ⚡<br>PLAYER ⚡<br>OPPONENT ⚡ |

ADD ANOTHER VALUE

CREATE CANCEL

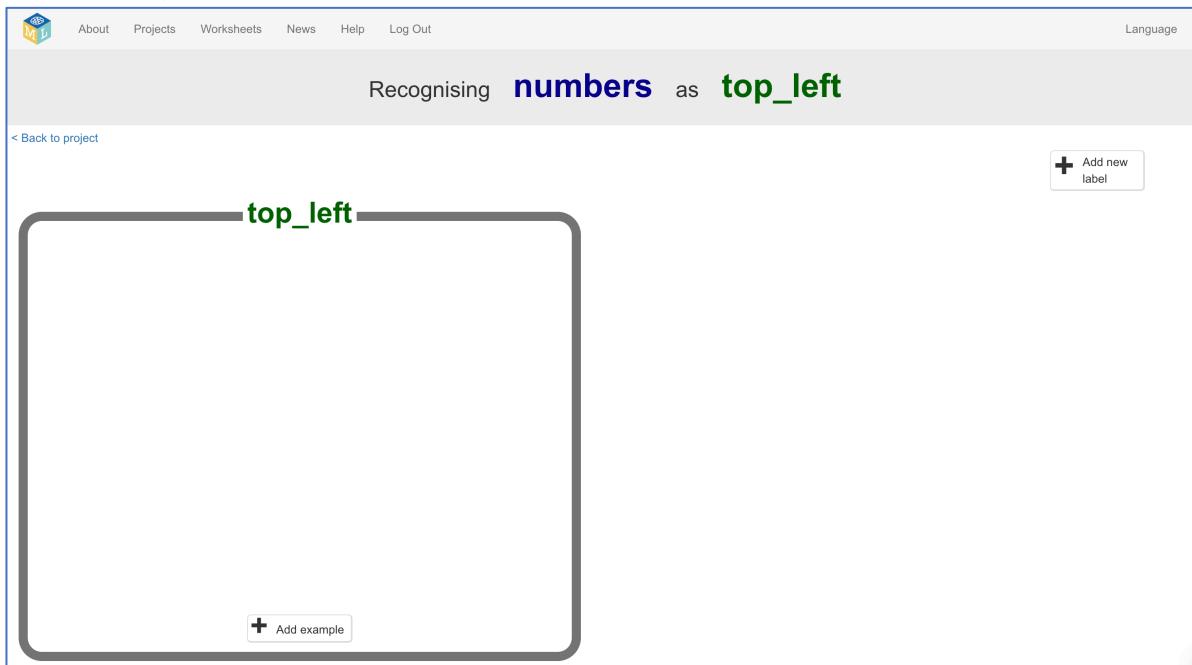
**15.** You should see “noughts and crosses” in your list of projects. Click on it.

## 16. Click the “Train” button



## 17. Click “+ Add new label” and create a label called “top left”

*Examples of making a move in the top-left box (in games that lead to a win) will go in this bucket.*



## 18. Click “+ Add new label” again and create labels for the other eight spaces on the board.

“top middle”, “top right”,  
“middle left”, “middle middle”, “middle right”,  
“bottom left”, “bottom middle”, “bottom right”

*(see the next page for a picture)*

The screenshot shows a project titled "Recognising numbers as top\_left, top\_middle or 7 other classes". It displays six rectangular boxes arranged in a 3x2 grid. Each box has a label above it and a "Add example" button below it. The labels are: top\_left, top\_middle, top\_right (top row); middle\_left, middle\_middle, middle\_right (middle row); and bottom\_left, bottom\_middle, bottom\_right (bottom row). A "Add new label" button is located in the top right corner.

19. Click the “< Back to project” link.

20. Click on “Make”

The screenshot shows a project titled "noughts and crosses". It features three main steps: "Train", "Learn & Test", and "Make". The "Train" and "Learn & Test" steps have "Train" and "Learn & Test" buttons respectively. The "Make" step has a "Make" button. A large blue arrow points from the "Learn & Test" step towards the "Make" step.

21. Click on “Python”

22. Find your project “API key”. You’ll need this code later.

The screenshot shows a project titled "Using machine learning in Python". A message at the top says "You haven't trained a machine learning model yet." Below it, a note says "You can train one now and then come back to start your Python project." A text input field contains the placeholder "If you know how to use it, your API key looks like this: this-is-not-a-real-API-key". A blue arrow points to this input field. At the bottom, a note says "Treat it like a password and make sure that you keep it secret!"

**23.** Visit <https://github.com/dalelane/Noughts-and-Crosses>

**24.** Click the “Clone or download” button, then click “Download ZIP”



**25.** Unpack the ZIP and open the Python code in your preferred editor.

**26.** Paste the API key from Step 22 into the KEY variable

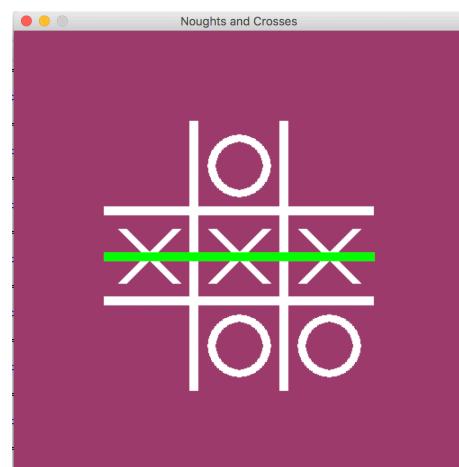
```
3 # this module is used to make HTTP requests to your machine learning model
4 import requests
5 # this module is used to choose a random colour for the user interface and
6 # make random choices about moves the computer should make
7 import random
8
9
10
11 # API KEY - the unique private code for your Machine Learning project
12 global KEY
13 KEY = "put-your-project-API-key-here"
14
15
```

**27.** Make sure you have **requests** and **pygame** installed.  
Ask your teacher if you don't know how to do this.

**28.** Run the Python program.

Play a game of noughts and crosses  
against the computer.

Your machine learning project will be  
choosing where to make its moves at  
random.



## 29. Look at the output from the Python program after the game ends. Next, you will edit the Python program so that the moves made by whoever wins the game are used to train your machine learning model.

```
HUMAN won the game!
Maybe the computer could learn from HUMAN's experience?

At the start of move 1 the board looked like this:
['EMPTY', 'EMPTY', 'EMPTY', 'EMPTY', 'EMPTY', 'EMPTY', 'EMPTY', 'EMPTY', 'EMPTY']
And HUMAN decided to put their mark in bottom_left

At the start of move 2 the board looked like this:
['COMPUTER', 'EMPTY', 'EMPTY', 'EMPTY', 'EMPTY', 'HUMAN', 'EMPTY', 'EMPTY', 'EMPTY']
And HUMAN decided to put their mark in middle_middle

At the start of move 3 the board looked like this:
['COMPUTER', 'EMPTY', 'EMPTY', 'HUMAN', 'EMPTY', 'HUMAN', 'EMPTY', 'EMPTY', 'COMPUTER']
And HUMAN decided to put their mark in top_right
```

## 30. Find the `learn_from_this` function *It is printing out the moves made by whoever won the game.*

```
181
182
183 # Someone won the game.
184 # A machine learning model could learn from this...
185 #
186 # winner      : who won - either HUMAN or COMPUTER
187 # boardhistory : the contents of the game board at each stage in the game
188 # winnerdecisions : each of the decisions that the winner made
189 def learn_from_this(winner, boardhistory, winnerdecisions):
190     print("%s won the game!" % (winner))
191     print("Maybe the computer could learn from %s's experience?" % (winner))
192     for idx in range(len(winnerdecisions)):
193         print("\nAt the start of move %d the board looked like this:" % (idx + 1))
194         print(boardhistory[idx])
195         print("And %s decided to put their mark in %s" % (winner, winnerdecisions[idx]))
196
197
198
```

## 31. Add a line in the `for` loop so that each move is added to your project training data

```
187 # boardhistory      : the contents of the game board at each stage in the game
188 # winnerdecisions : each of the decisions that the winner made
189 def learn_from_this(winner, boardhistory, winnerdecisions):
190     print("%s won the game!" % (winner))
191     print("Maybe the computer could learn from %s's experience?" % (winner))
192     for idx in range(len(winnerdecisions)):
193         print("\nAt the start of move %d the board looked like this:" % (idx + 1))
194         print(boardhistory[idx])
195         print("And %s decided to put their mark in %s" % (winner, winnerdecisions[idx]))
196         add_to_train(boardhistory[idx], winner, winnerdecisions[idx]) ←
197
198
```

## 32. Run your Python program again and play another game.

### 33. Go back to the training page

Leave the Python window to go back to the training tool window.  
Click the “**< Back to project**” link and then click “**Train**”

The screenshot shows a web-based training interface for a project titled "Recognising numbers as top\_left, top\_middle or 7 other classes". The interface features six input fields arranged in two rows of three. Each field has a "Add example" button below it. The first row contains fields for "top\_left", "top\_middle", and "top\_right". The second row contains fields for "middle\_left", "middle\_middle", and "middle\_right". Each field displays a 3x3 grid of labels representing board states. The "top\_right" field shows the following grid:

|              |          |          |
|--------------|----------|----------|
| TopLeft      | EMPTY    | EMPTY    |
| TopMiddle    | EMPTY    | EMPTY    |
| TopRight     | EMPTY    | EMPTY    |
| MiddleLeft   | EMPTY    | EMPTY    |
| MiddleMiddle | EMPTY    | OPPONENT |
| MiddleRight  | OPPONENT | PLAYER   |
| BottomLeft   | PLAYER   | EMPTY    |
| BottomMiddle | EMPTY    | OPPONENT |
| BottomRight  | OPPONENT | EMPTY    |

A "Language" dropdown menu is visible in the top right corner. A circled number "1" is located near the "middle\_right" field.

### 34. Look at your training so far

Each item is a move made by the winning player.

The details in each item describe the state of the board at the time the winning player made that move.

This is collecting training data, but you still need to use it to train a machine learning model.

### 35. Edit the learn\_from\_this function again to add a new line

The new line will use the training data collected so far to create a new machine learning model.

Make sure you get the indenting right so you only train a model after adding all of the moves.

```
187 # boardhistory : the contents of the game board at each stage in the game
188 # winnerdecisions : each of the decisions that the winner made
189 def learn_from_this(winner, boardhistory, winnerdecisions):
190     print("%s won the game!" % (winner))
191     print("Maybe the computer could learn from %s's experience?" % (winner))
192     for idx in range(len(winnerdecisions)):
193         print("\nAt the start of move %d the board looked like this:" % (idx + 1))
194         print(boardhistory[idx])
195         print("And %s decided to put their mark in %s" % (winner, winnerdecisions[idx]))
196         add_to_train(boardhistory[idx], winner, winnerdecisions[idx])
197 train_new_model()
198
199
```

## 36. Play several more games.

Your changes to the code mean that after every game finishes, the moves made by the winner are added to the training data as examples of how to win, and a new model is trained using all examples collected so far.

The longer you play, the more the computer can learn from you, and the better it should get.

## 37. Go back to training page and look at all of your training examples You will need to refresh the page to see them.

## 38. Go to the Learn & Test page

Click the “**< Back to project**” link and then click “**Learn & Test**”

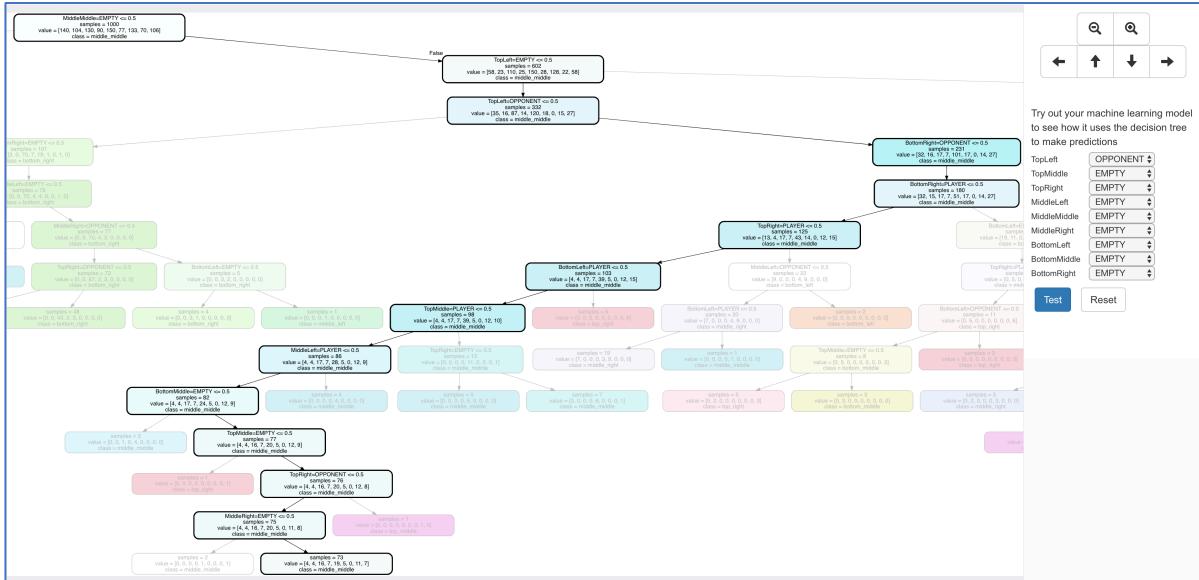
Check when your code last trained a machine learning model

The screenshot shows the 'Learn & Test' page. At the top, there are two sections: '6 examples of bottom\_middle' and '12 examples of bottom\_right'. Below this is a form for testing with dropdown menus for various positions (TopLeft, TopMiddle, etc.) and buttons for 'Test' and 'Describe your model'. A large blue arrow points from the top right towards the bottom of the page. At the bottom, there's a section titled 'Info from training computer' with a timestamp ('Model started training at: Sunday, September 1, 2019 11:11 AM') and a button to 'Delete this model'.

## 39. Click “Describe your model” to see how your model works

The screenshot shows the 'Describe your model' page. At the top, there's a navigation bar with links for 'About', 'Projects', 'Worksheets', 'News', 'Help', 'Log Out', and 'Language'. Below this is a title 'Understand your machine learning model'. The page contains explanatory text about decision trees and a large decision tree diagram. The tree has multiple levels of nodes, each with conditions like 'TopLeft == EMPTY' or 'TopLeft == BottomMiddle'. To the right of the tree, there's a legend for symbols: magnifying glass, double arrows, and arrows pointing up, down, left, and right. A text box on the right says 'Try out your machine learning model to see how it uses the decision tree to make predictions'. At the bottom, there's a form for testing with dropdown menus for positions and buttons for 'Test' and 'Reset'.

**40.** Describe a state of the board on the right and click “Test”  
*The visualisation describes how your model decides where to move.*



**41.** Keep playing until you start to see the computer get better  
*How does that change the machine learning model you create?*

## What have you done?

You've trained a computer to play noughts and crosses.

You didn't have to describe the rules to the computer.

You didn't tell it that it should try to get three noughts in a row.

(The rules are in the Python code so it can be displayed but aren't used in the machine learning model).

You showed it how you play, by collecting examples of decisions that you made when you win. When it makes decisions that leads to it winning, this is added to its training data, so it can be even more confident in that approach in future.

This is called “reinforcement learning” because when it does something good you are “reinforcing” this by rewarding it.

## Tips

### **Don't be kind!**

You might be tempted to go easy on the computer when you're playing against it, particularly when it's just starting to learn and is playing very badly.

For example, you might have two crosses-in-a-row next to a blank space and could win. But instead, you might feel sorry for it doing badly and put a cross somewhere else instead to give it a chance.

Don't.

It is learning from the way that you play. If you don't complete a three-in-a-row when you can, you will be teaching it that it should do that.

If you want it to get better quickly, **play as well as you can**.

### **Mix things up with your examples**

Try to come up with lots of different types of examples.

For example, start from a different position on the board on every turn.

## Did you know?

People have been learning about machine learning by training a computer to play noughts and crosses for decades!

One famous example was **Donald Michie** – a British artificial intelligence researcher. During World War II, Michie worked at Bletchley Park as a code breaker.

In 1960, he developed “**MENACE**” – the Machine Educable Noughts And Crosses Engine. This was one of the first programs able to learn how to play noughts and crosses perfectly.

As he didn’t have a computer he could use, Michie built MENACE using 304 matchboxes and coloured glass beads.

Each matchbox represented a possible state of the board – like the examples that you’ve been collecting in your training data.

He put beads in the matchboxes to show how often a choice led to a win – the number of beads in the matchbox was like the number of times an example shows up in one of the buckets you created for your training data.

