



# 台大資管營 微課程

Winter 2025

## 人工智慧導論 (Introduction to AI)

講師：林杰

國立臺灣大學 資訊管理學系

These slides are primarily adapted from those created by Dan Klein and Pieter Abbeel (CS188: Introduction to AI, UC Berkeley).

Additional adaptations were made from slides by:

- Hsin-Min Lu (IM5056: Statistical Learning and Deep Learning, NTU),
- Shang-Tse Chen and Vivian Chen (CSIE3005: Foundations of Artificial Intelligence, NTU).

本簡報僅限課程中使用。

# 課程簡報 PDF 與練習檔案

- Github Link: <https://shorturl.at/CzOpG>
- 可以在打開後，點擊右上角的下載按鈕



2025-IM-Camp / intro\_AI\_slide.pdf

jaylin0418 Add files via upload

5.44 MB

台大資管營 微課程

Winter 2025

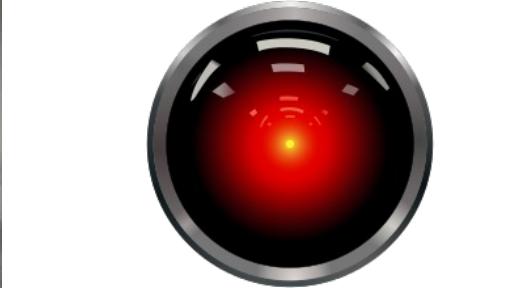
人工智慧導論 (Introduction to AI)

講師：林杰

國立臺灣大學 資訊管理學系

A screenshot of a GitHub repository page for '2025-IM-Camp / intro\_AI\_slide.pdf'. The file was uploaded by 'jaylin0418' 15 hours ago. The file size is 5.44 MB. In the top right corner of the file card, there is a download icon (a downward arrow inside a circle) which is highlighted with a thick red arrow pointing towards it from the left side of the image.

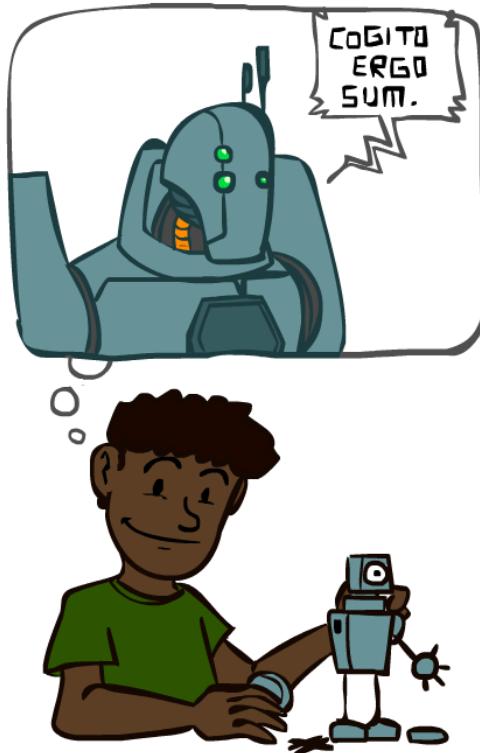
# AI in Science Fiction



# A (Short) History of AI

---

- 1940-1950: Early days
  - 1943: McCulloch & Pitts: Boolean circuit model of brain
  - 1950: Turing's "Computing Machinery and Intelligence"
- 1950—70: Excitement: Look, Ma, no hands!
  - 1950s: Early AI programs, including Samuel's checkers program, Newell & Simon's Logic Theorist, Gelernter's Geometry Engine
  - 1956: Dartmouth meeting: "Artificial Intelligence" adopted
  - 1965: Robinson's complete algorithm for logical reasoning
- 1970—90: Knowledge-based approaches
  - 1969—79: Early development of knowledge-based systems
  - 1980—88: Expert systems industry booms
  - 1988—93: Expert systems industry busts: "AI Winter"
- 1990—: Statistical approaches
  - Resurgence of probability, focus on uncertainty
  - General increase in technical depth
  - Agents and learning systems... "AI Spring"?
- 2000—: Where are we now?

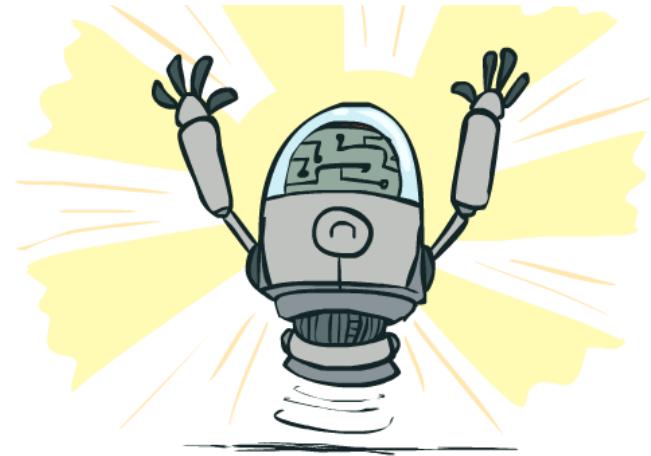


# What Can AI Do?

---

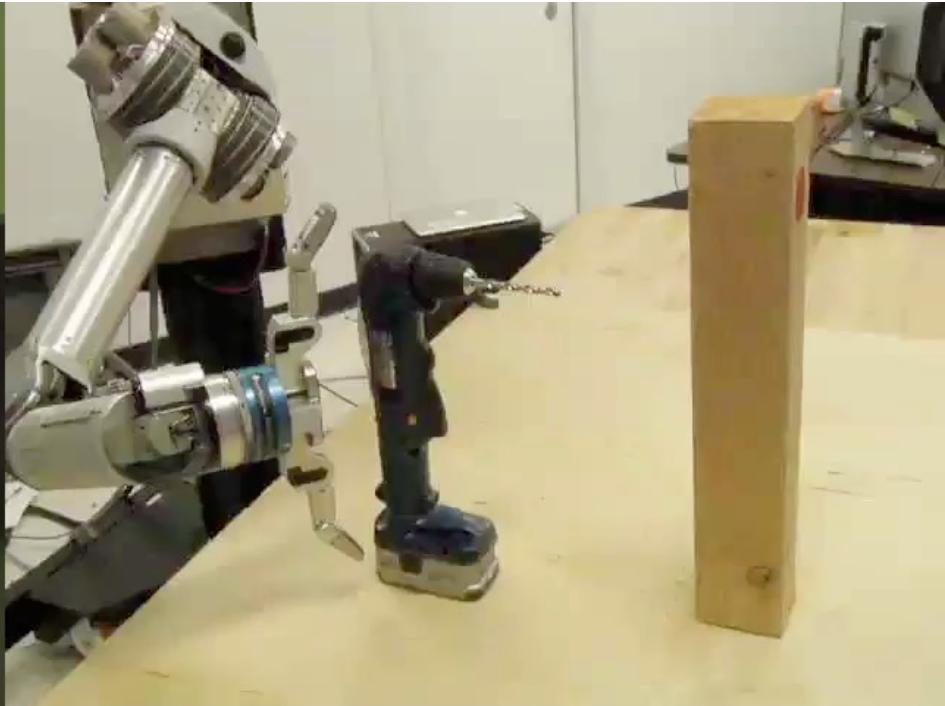
Quiz: Which of the following can be done at present?

- Play a decent game of Bingo?
- Win against any human at chess?
- Win against the best humans at Go?
- Play a decent game of table tennis?
- Grab a particular cup and put it on a shelf?
- Unload any dishwasher in any home?
- Drive safely along the highway?
- Drive safely in Taipei?
- Buy a week's worth of groceries on the web?
- Perform a surgical operation?
- Discover and prove a new mathematical theorem?
- Translate spoken Chinese into spoken English in real time?

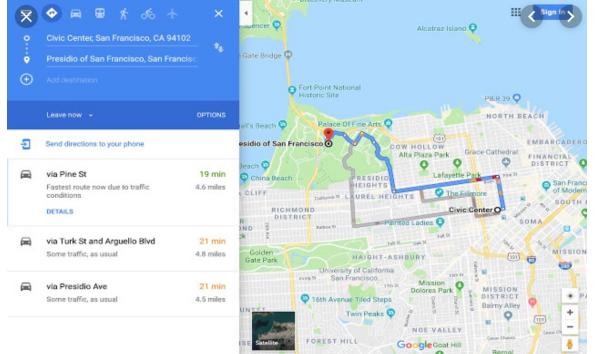


# Robots

---

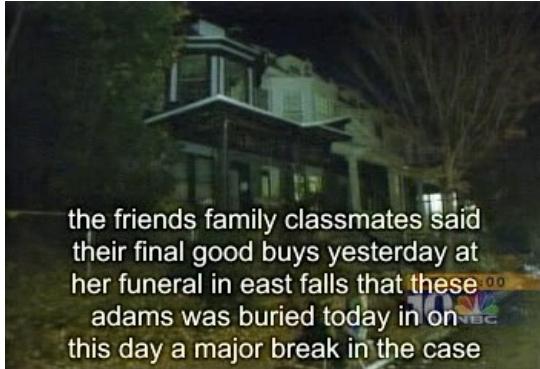


# Tools for Predictions & Decisions



# Natural Language

---



自動圖片描述  
Auto-description

**"Il est impossible aux journalistes de rentrer dans les régions tibétaines"**

Bruno Philip, correspondant du "Monde" en Chine, estime que les journalistes de l'AFP qui ont été expulsés de la province tibétaine de Qinghai "n'étaient pas dans l'illégalité".

**Les faits** Le dalaï-lama dénonce l'"enfer" imposé au Tibet depuis sa fuite, en 1959

**Video** Anniversaire de la rébellion anti-chinoise au Tibet

**"It is impossible for journalists to enter Tibetan areas"**

Philip Bruno, correspondent for "World" in China, said that journalists of the AFP who have been deported from the Tibetan province of Qinghai "were not illegal."

**Facts** The Dalai Lama denounces the "hell" imposed since he fled Tibet in 1959

**Video** Anniversary of the Tibetan rebellion: China on guard

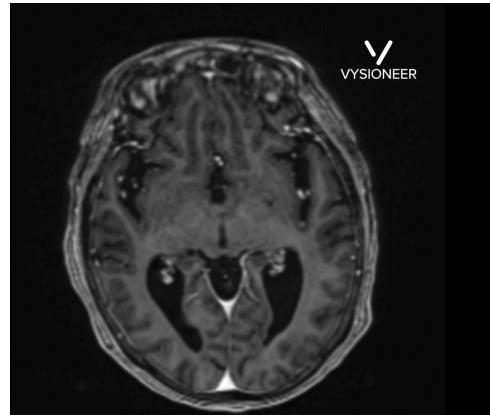
機器翻譯  
Machine Translation

# Recent Applications of AI

---

一些新興的應用：

- 自動駕駛汽車
  - 運用機器學習算法進行路徑規劃和即時決策
- 醫療診斷與監控
  - AI協助醫生分析X光、MRI等醫學影像，提升診斷準確率
- 金融服務
  - AI分析大量數據，預測金融風險



# So, what is AI?

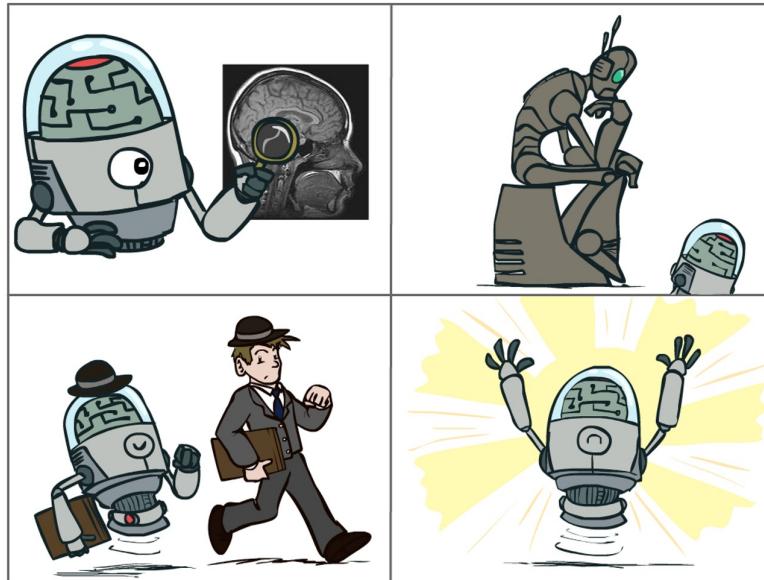
The science of making machines that:

Think like  
people

(像人一樣思考)

Act like people

(像人一樣動作)



Think rationally  
(理性思考)

Act rationally  
(理性動作)

# Rational (理性的) Decisions

---

當這堂課講到「理性」這個詞時，定義如下：

- 理性：「最大化」達成「預先定義的」目標
- 只關注所做出的決策
  - (不涉及背後的思考過程)
- 目標以「結果的效用 (**utility of outcomes**)」來表達
- 理性意味著最大化你的期望效用
  - **maximizing your expected utility**

# The Earliest AI: Expert System

---

最早的專家系統：

- 專家系統是一種模擬人類專家決策過程的人工智慧系統
- 最簡單的專家系統中，使用 if-else 條件判斷來做出決策

範例：診斷感冒或流感

如果（有高燒）且（持續時間 > 3 天）：

結論 = 流感

否則：

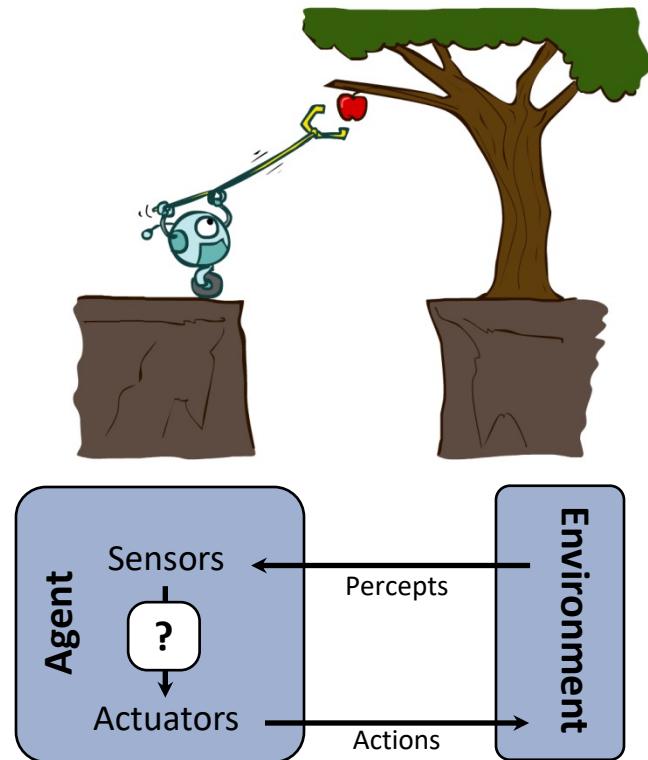
模擬醫生的決策過程！

結論 = 普通感冒

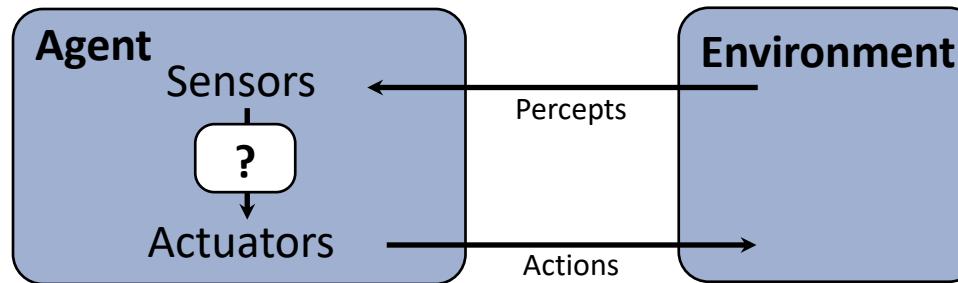
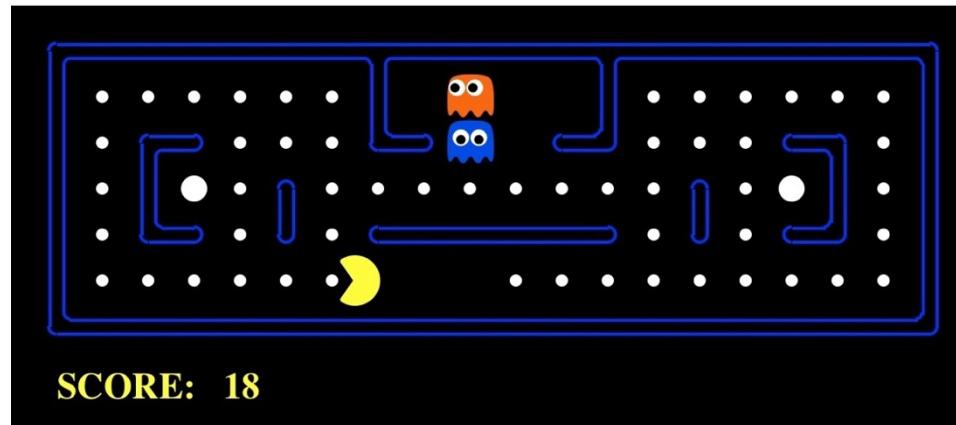


# Designing Rational Agents

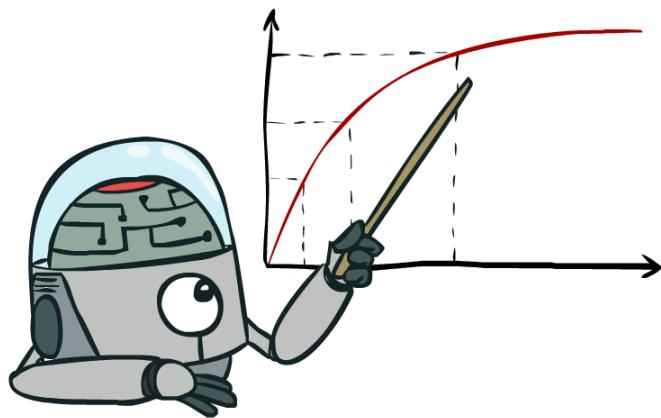
- An **agent** is an entity that *perceives* and *acts*.
- A **rational agent** selects actions that maximize its (expected) **utility**.
- Characteristics of the **percepts**, **environment**, and **action space** dictate techniques for selecting rational actions
- 一個 **agent** 是能夠感知 (*perceives*) 並執行行動 (*acts*) 的個體
- 理性 **agent** 會選擇能最大化期望效用 (*expected utility*) 的行動
- 感知、環境和行動空間 (**percepts**, **environment**, and **action space**) 的特性決定了選擇理性行動的方法



# Pac-Man as an Agent



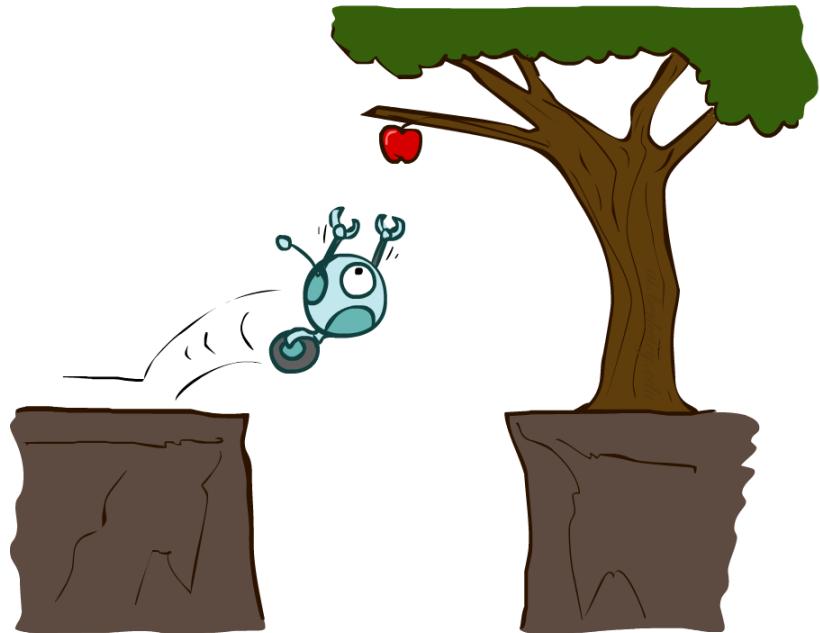
# Our goal: Design the Pacman Agent



# Reflex Agents

---

- Reflex agents (反射型 agents):
  - 根據當前的感知 (current percept) 以及可能的記憶選擇行動
  - 會對所處的世界 (world) 進行建模
  - 不考慮目前行動的未來後果
  - Consider how the world IS
- *Can a reflex agent be rational?*
- 反射型 agent 可以是理性的嗎?

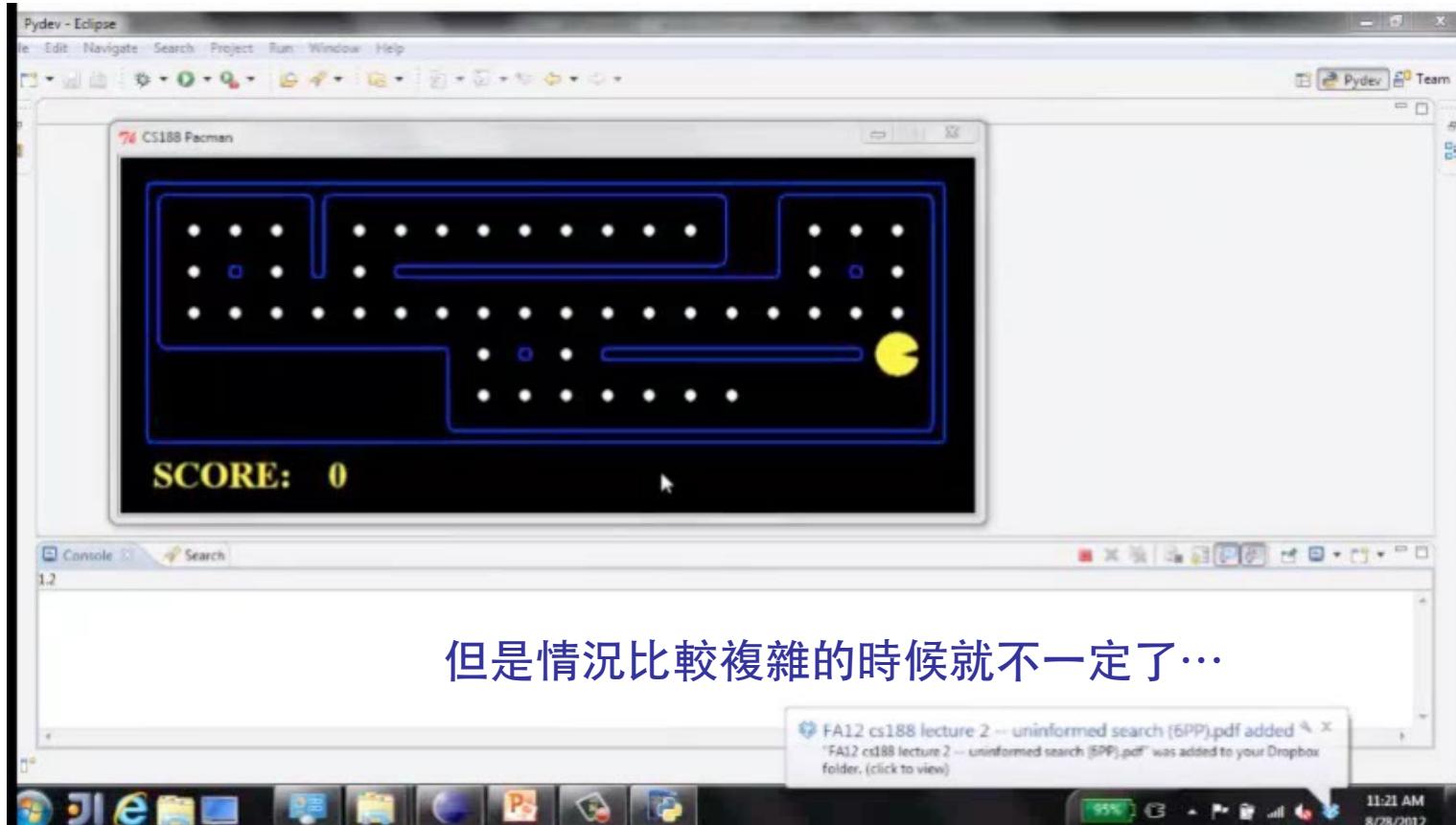


# Video of Demo Reflex Optimal



Reflex Agent 有時確實能有最佳解  
(Move in the direction of the nearest  
uneaten dot)

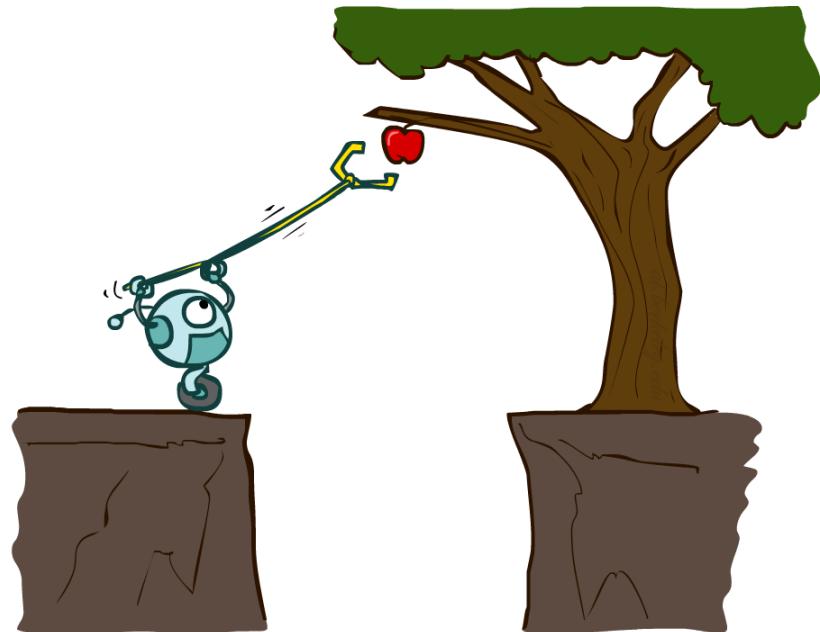
# Video of Demo Reflex Odd



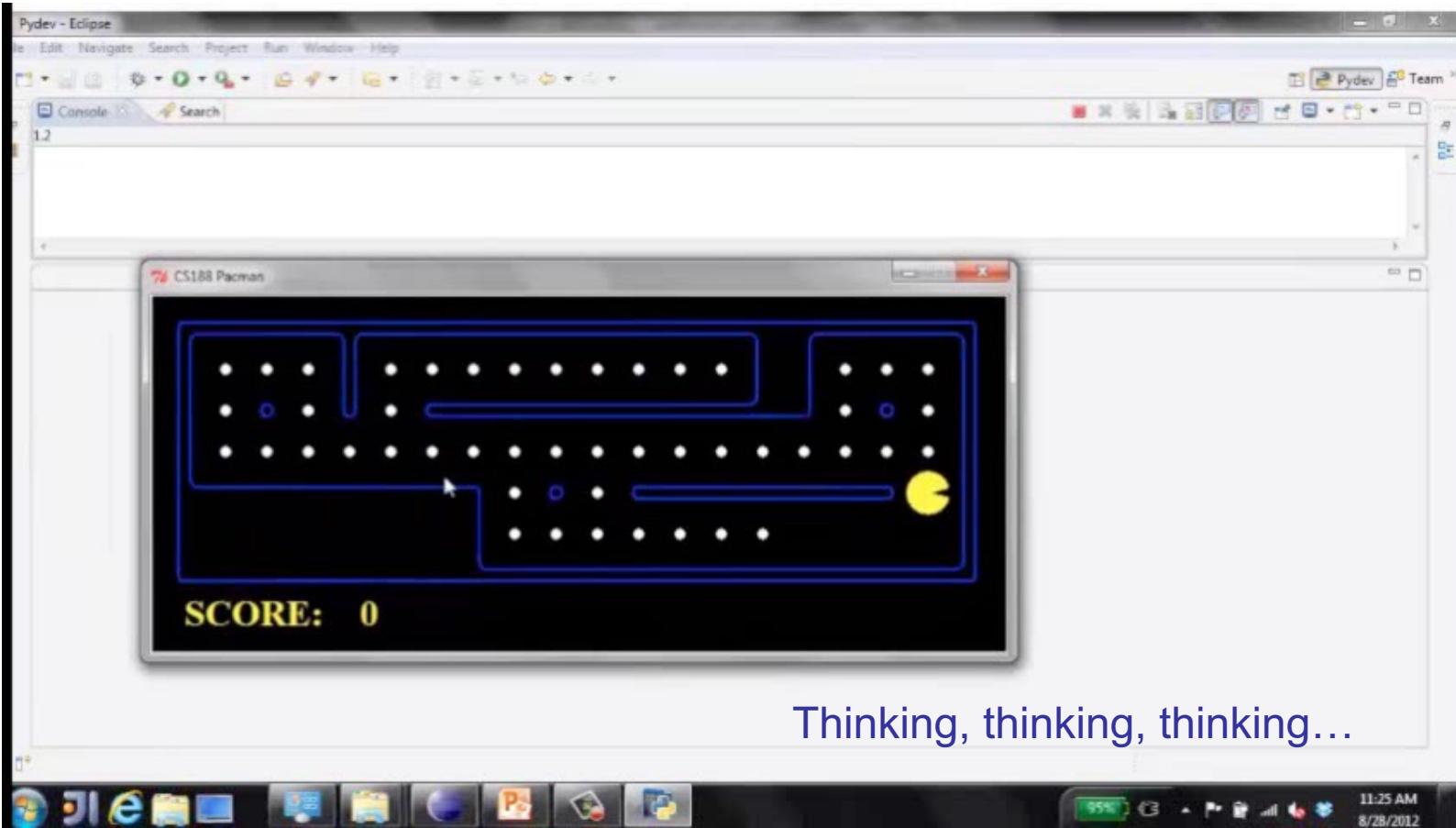
# Planning Agents

---

- Planning agents (計畫型 agents) :
  - Ask “what if”
  - 根據行動 (act) 的「假設性後果」做決策
  - 必須具備一個模型 (model) 描述行動對世界  
的影響
  - 必須設定一個目標 (goal) , 例如: 是否所有  
的點點都被吃完?
  - 精神 : Consider how the world WOULD BE



# Video of Demo Planning



# Search Problems

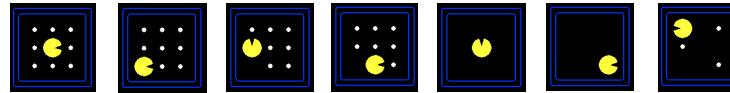
---



# Search Problems

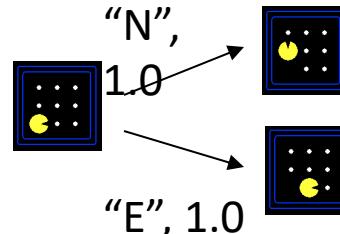
- 搜尋問題 (search problem) 有以下的元素：

- A state space  
( 狀態空間 )



- A successor function  
(with actions, costs)

( 繼承函數，  
包含動作與成本 )

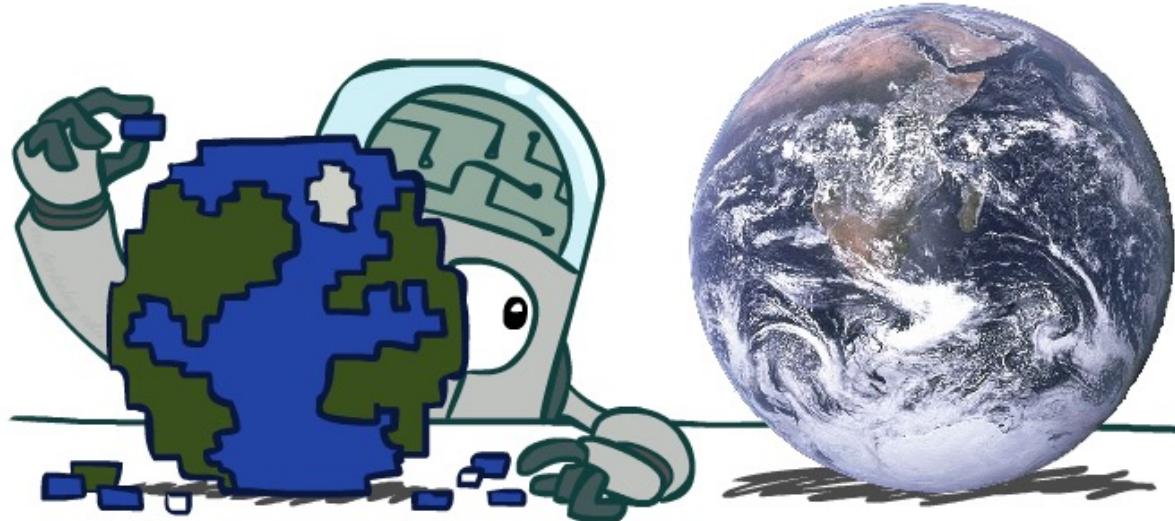


- A start state and a goal test  
( 起始狀態與目標測試 )

- 解答 (solution)：一系列計畫好的行動，將起始狀態轉化為目標狀態

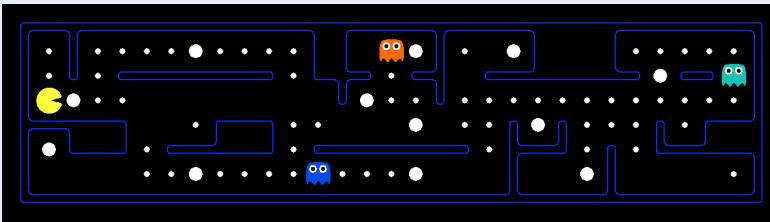
# Search Problems Are Models

---



# What's in a State Space?

World state 包含所有環境中的細節

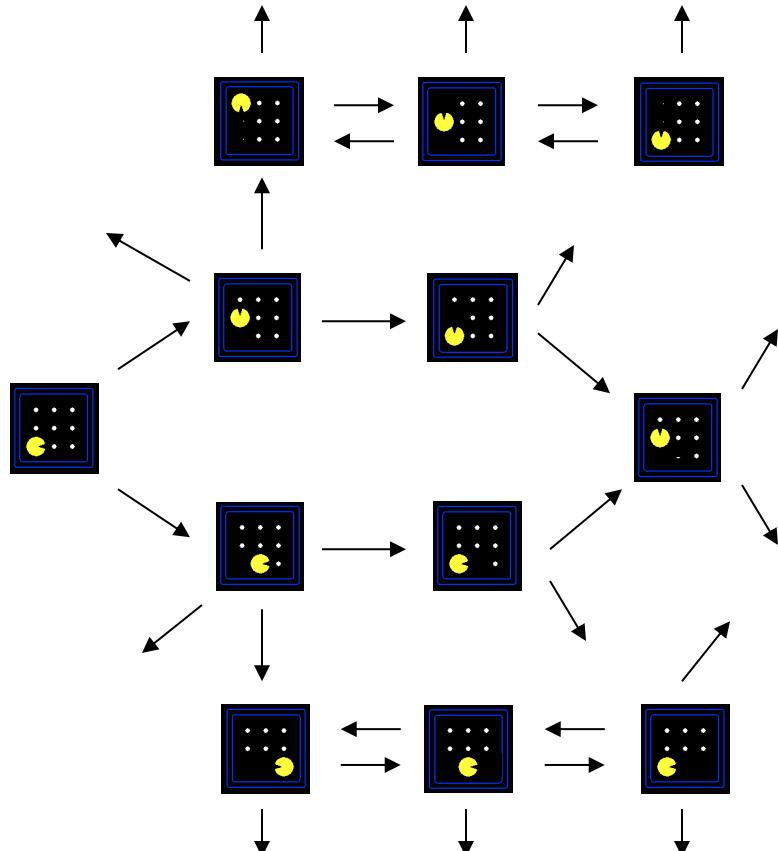


Search state 只保留 planning 所需的細節 (抽象化)

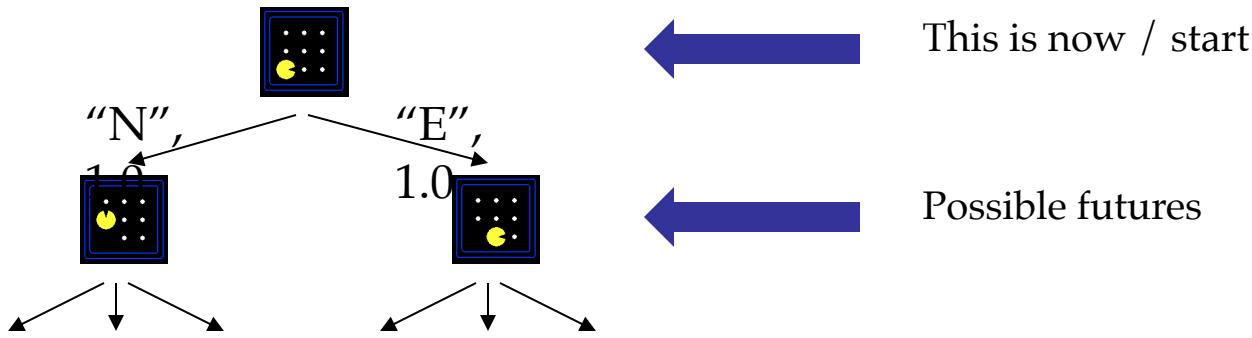
- Problem: Pathing (路徑規劃問題)
  - States:  $(x,y)$  座標
  - Actions: NSEW
  - Successor: 只更新座標資訊
  - Goal test: is  $(x,y) = \text{destination?}$
- Problem: Eat-All-Dots
  - States:  $\{(x,y), \text{dot booleans}\}$
  - Actions: NSEW
  - Successor: 更新座標資訊以及 dot boolean
  - Goal test: 所有的點是否已被吃完?

# State Space Graphs

- 狀態空間圖 (State space graph) : 一種用來表示搜尋問題的數學模型
  - Nodes (節點) are abstracted world configurations
  - Arcs (邊) represent successors (動作的結果)
  - Goal test 是一組 goal nodes 的集合
- 在狀態空間圖中，每個狀態只會出現一次！
- 我們通常無法在記憶體中構建完整的狀態空間圖（因為它太大了…）
- 但這是一個有用的概念



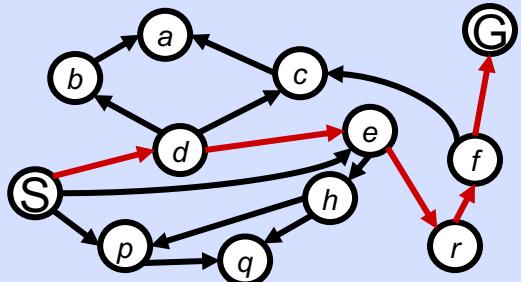
# Search Trees



- 搜尋樹:
  - A “what if” tree of plans and their outcomes
  - 起始狀態 (start state) 是根節點 (root node)
  - 子節點 (children) 對應到後繼節點 (successors)
  - Nodes show states, but correspond to PLANS that achieve those states
  - **對於大多數問題來說，我們實際上無法構建完整的樹**

# State Space Graphs vs. Search Trees

State Space Graph

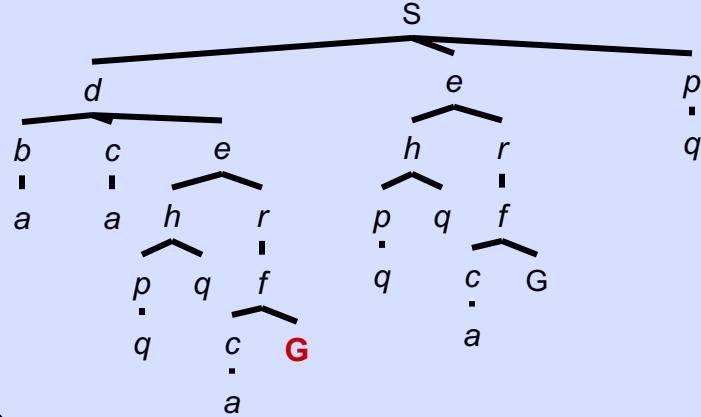


S: Start  
G: Goal

每個節點  
(NODE) 在  
搜尋樹中代表  
狀態空間圖中  
的一整條路徑  
(PATH)

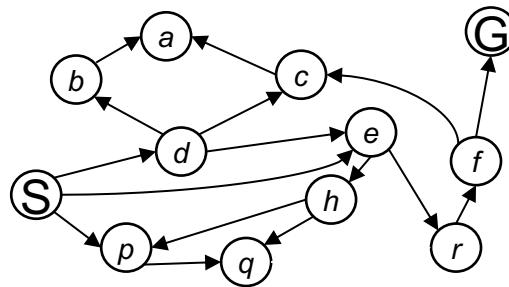
*We construct  
both on  
demand – and  
we construct  
as little as  
possible.*

Search Tree



# Example: Tree Search

---



# Depth-First Search

---

(深度優先搜尋)

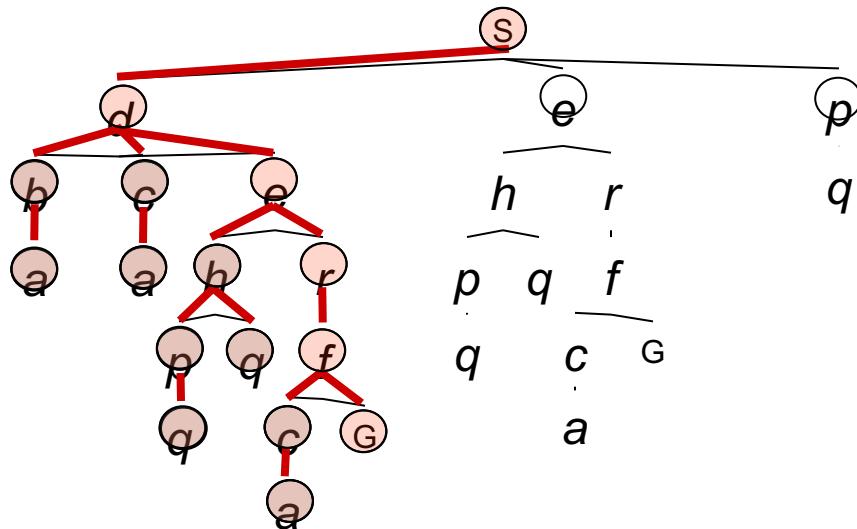
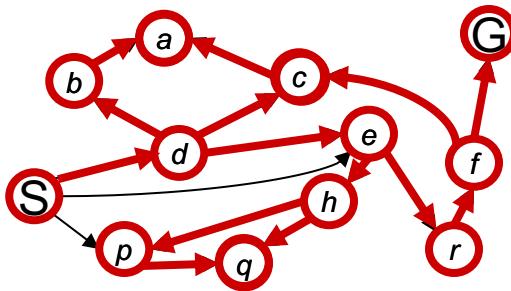


# Depth-First Search

策略：優先展開最深的節點

*Implementation:*  
LIFO stack

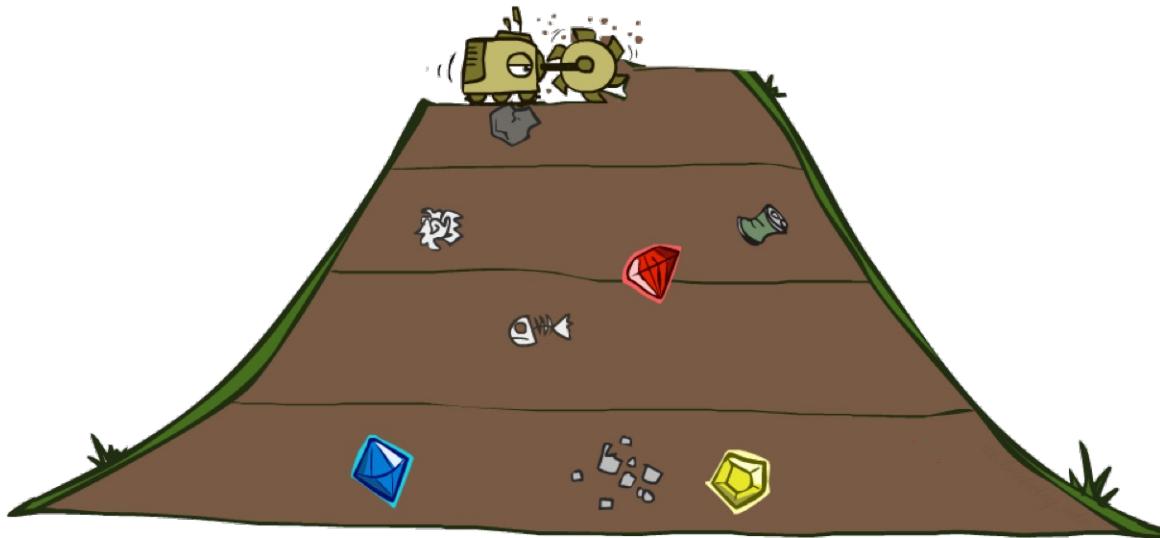
做 DFS 的 Agent 是一種 Planning Agent



# Breadth-First Search

---

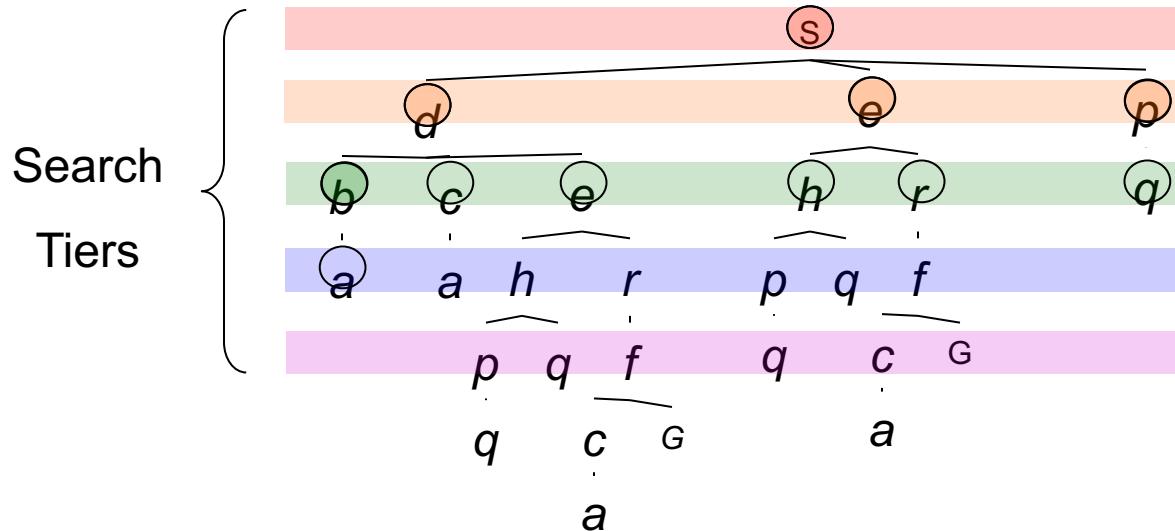
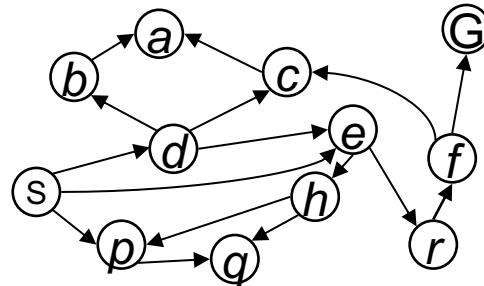
(廣度優先搜尋)



# Breadth-First Search

策略：優先展開  
最淺的節點

*Implementation:*  
*FIFO queue*



# Other Searching Methods

---

其他進階的搜尋演算法：

- Uniform-Cost Search
- Greedy Search
- A\* Search
- You need to understand “Heuristics” if you want to know the last two methods.

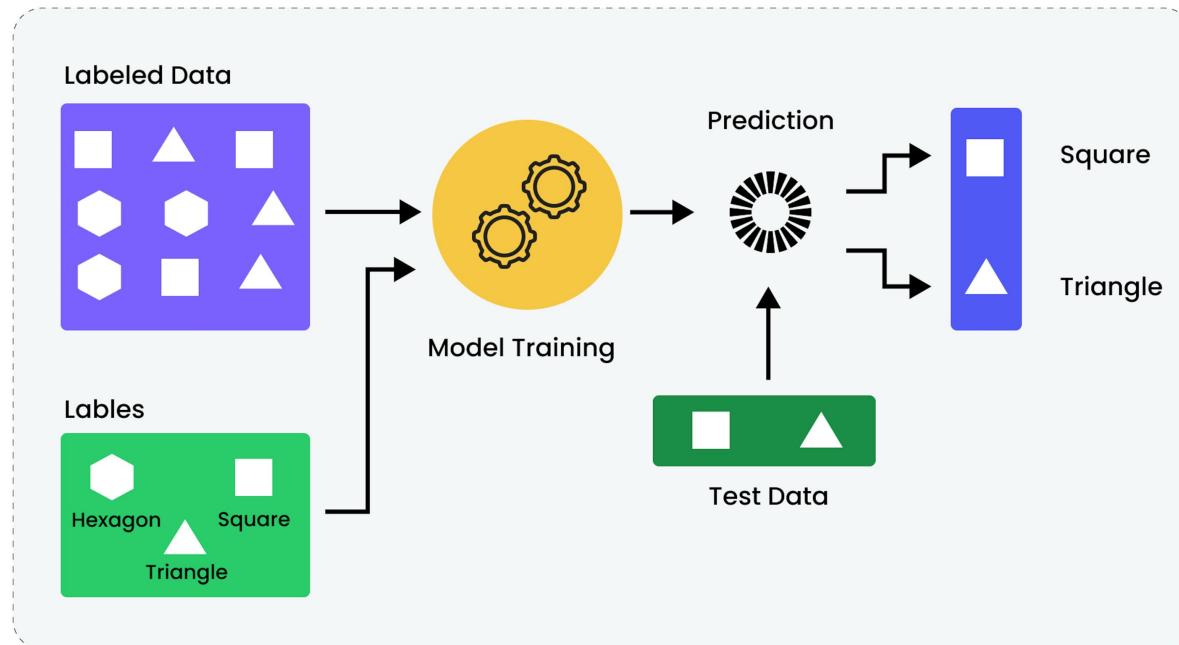
# Supervised Learning

---

- 什麼是監督式學習?
  - 透過已知的「輸入」和對應的「標籤」來訓練模型
  - 目標是讓模型學會從過去的數據中預測或分類新數據
- 1. 訓練資料：包括已標註的輸入數據和對應的真實標籤（如圖片與其標籤）
- 2. 模型學習：模型學會從輸入數據中抽取模式，並預測對應的標籤
- 3. 目標：最小化預測與實際標籤之間的誤差

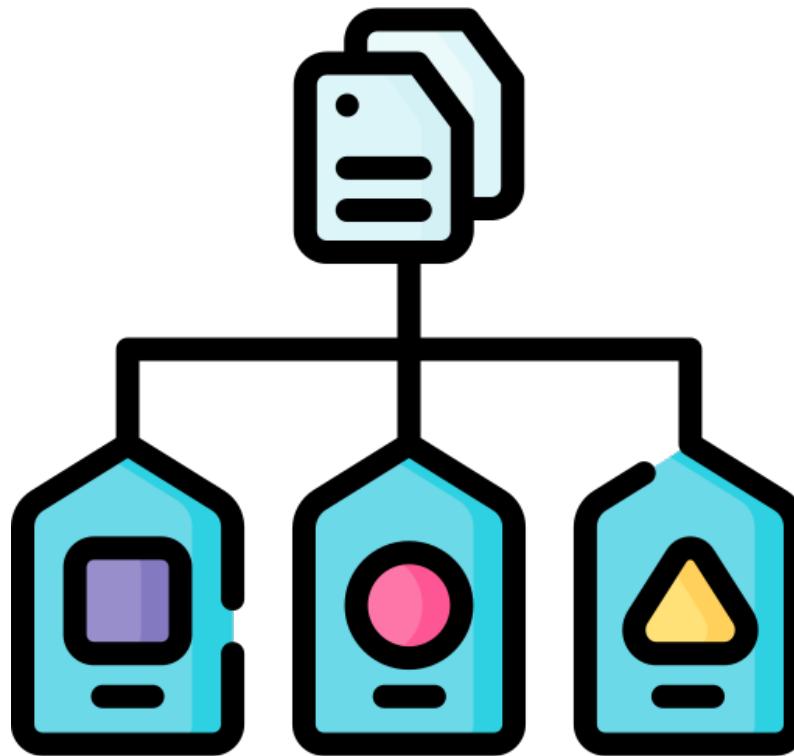
# Supervised Learning

監督式學習: 需要 data, 也需要對應的 label



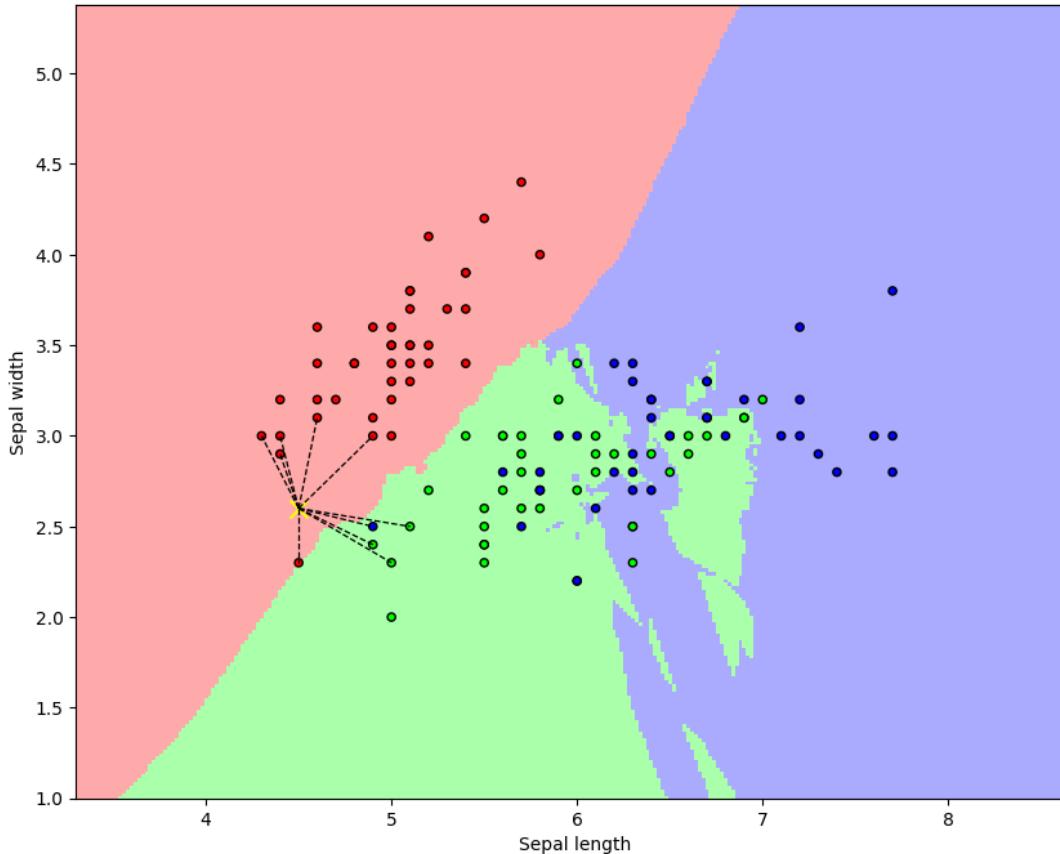
# Classification by kNN

---



# k Nearest Neighbor (kNN)

---



# k Nearest Neighbor (kNN)

---

Training data:  $(x_i, y_i)$

- $y_i$  是標籤
- $x_i$  是長度為  $m$  的特徵向量。可能包括多個特徵

標籤  $y_i$  的兩種情況：

- 1. 分類問題：
  - 如果  $y_i$  是離散值（例如  $y_i \in \{1,2,3\}$ ），表示這是分類任務
  - kNN 將根據鄰近的  $k$  個樣本的標籤進行多數投票，選出一個最可能的類別作為預測結果
- 2. 回歸問題：
  - 如果  $y_i$  是連續值（例如  $y_i \in R$ ），表示這是回歸任務
  - kNN 將計算鄰近  $k$  個樣本標籤的平均值，作為預測結果

# How to Measure Distance?

---

- L2 Norm (Euclidian distance):

$$\|x_i - x_j\| = \sqrt{\sum_{q=1}^m (x_{i,q} - x_{j,q})^2}$$

- L1 Norm:

$$\|x_i - x_j\|_1 = \sum_{q=1}^m |x_{i,q} - x_{j,q}|$$

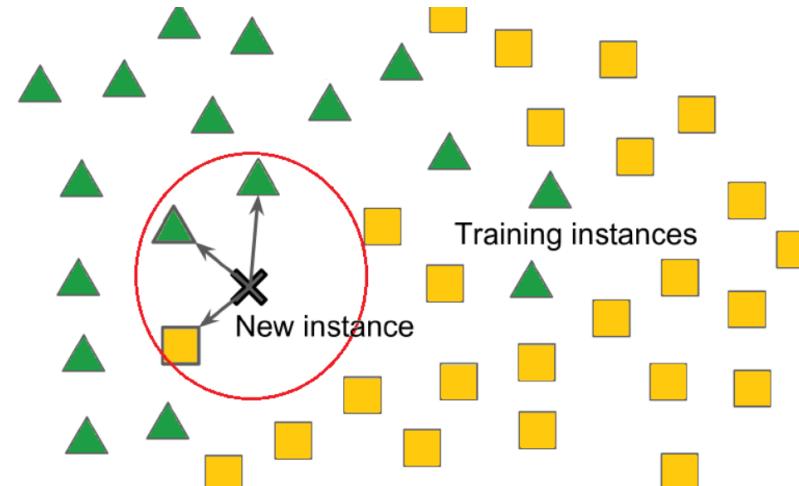
# k Nearest Neighbor (kNN)

Training data:  $(x_i, y_i)$

- $y_i$  是標籤
- $x_i$  是長度為  $m$  的特徵向量。可能包括多個特徵

標籤  $y_i$  的兩種情況：

- 1. 分類問題
- 2. 回歸問題



來源: Ahmed Raafat, "K-Nearest Neighbor (KNN) Explained," *Machine Learning Mastery*, September 9, 2022. <https://machinelearningmastery.com>

# $k$ Nearest Neighbor 分類問題

---

分類問題：

- 如果  $y_i$  是離散值（例如  $y_i \in \{1,2,3\}$ ），表示這是分類任務
- kNN 將根據鄰近的  $k$  個樣本的標籤進行多數投票，選出一個最可能的類別作為預測結果

範例：

- $k = 3$ ，最近的三個樣本標籤為 [1,2,2]
- 多數投票結果：預測類別為 2

# k Nearest Neighbor 回歸問題

---

回歸問題：

- 如果  $y_i$  是連續值（例如  $y_i \in R$ ），表示這是回歸任務
- kNN 將計算鄰近  $k$  個樣本標籤的平均值，作為預測結果

範例：

- $k = 3$ ，最近的三個樣本標籤為  $[3.2, 4.1, 3.8]$
- 平均值： $\frac{3.2+4.1+3.8}{3} = 3.7$
- 預測結果為 3.7

# Measuring Prediction Performance

---

監督式學習訓練流程：

- ① 隨機排列數據，並分成**訓練集**（90%）和**測試集**（10%）
- ② 訓練集可再分為**子訓練集**（80%）和**調參集**（10%）
- ③ 若模型沒有超參數，則直接用訓練集訓練，並用測試集測試
- ④ 若模型有超參數，使用子訓練集來訓練，調整超參數後，固定超參數並重新訓練
- ⑤ 最後，使用測試集測試最終模型的效果

# Discussions on kNN

---

1. k 在這裡是個「超參數」(Hyperparameter)：

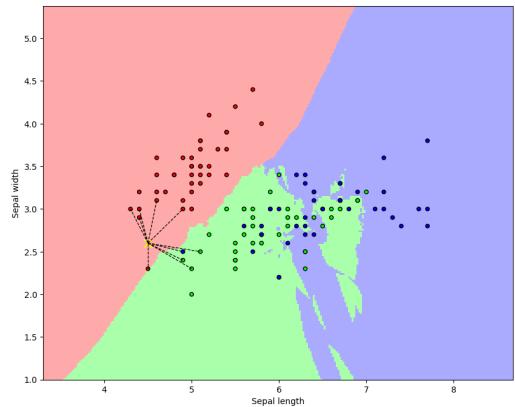
- 使用不同的 k，可能會帶來不同的模型準確率！

2. kNN 是一個「不需要訓練」的演算法

- 簡單易懂

3. 代價：kNN 的計算複雜度很高

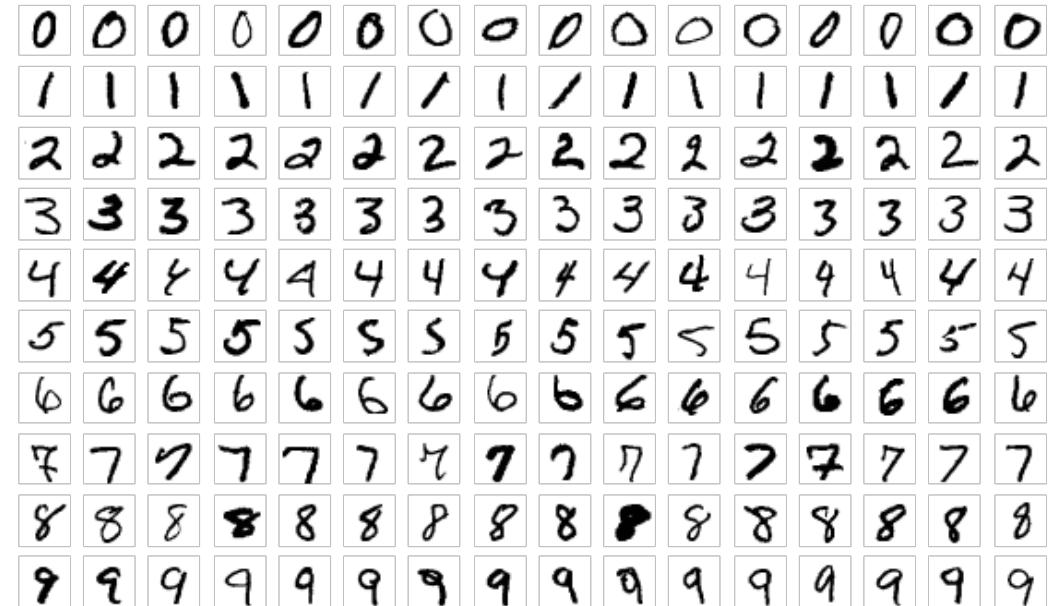
- 找出前 k 個最靠近的鄰居，這件事情是在計算上很複雜的



# kNN 實作手寫數字分類

---

- Google Colab link: <https://shorturl.at/AfaKK>
- 請在打開檔案後，登入自己的 Google 帳號



# kNN Exercise: Solution (1)

---

## 1. 載入手寫數字數據集

我們將使用 `sklearn` 提供的手寫數字數據集，這個數據集包含 1797 張手寫的數字圖片，每張圖片是 8x8 的大小，經過展平後變成 64 維的向量。

```
from sklearn.datasets import load_digits

# 下載手寫數字數據集
digits = load_digits()

# 數字圖片的數據 (1797, 64)，每個數字是一個 8x8 的圖像展平為 64 維
data = digits.data

# 真實標籤 (0-9)，每張圖片對應的真實數字
labels = digits.target
```

這段程式碼會將數據集中的圖片數據 (`data`) 和對應的標籤 (`labels`) 分別存儲在 `data` 和 `labels` 變數中。

# kNN Exercise: Solution (2)

---

## 2. 使用 KNN 進行分類

KNN 是一種常見的分類方法。在這裡，我們會將數據集分成兩部分：訓練集（用來訓練模型）和測試集（用來測試模型的表現）。

```
from sklearn.model_selection import train_test_split  
  
# 將數據集分為訓練集和測試集  
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.3, random_state=42)
```

在這裡，我們使用 `train_test_split` 函數將數據分成 70% 的訓練集和 30% 的測試集。

# kNN Exercise: Solution (3)

---

然後，我們定義 KNN 模型，並使用訓練集來進行訓練。這裡的 `k=3` 表示每次分類時會參考最近的 3 個鄰居。

```
from sklearn.neighbors import KNeighborsClassifier  
  
# 定義 KNN 模型，選擇適當的 k 值  
knn = KNeighborsClassifier(n_neighbors=3)  
  
# 訓練 KNN 模型  
knn.fit(X_train, y_train)
```

KNeighborsClassifier(n\_neighbors=3)

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

這段程式碼建立了 KNN 模型並用訓練集數據來訓練它。

# kNN Exercise: Solution (4)

## 3. 預測結果並視覺化分類結果

一旦模型訓練完成，我們可以用測試集數據來進行預測。這裡的 `predict` 函數會返回模型對測試集每一張圖片的預測結果。

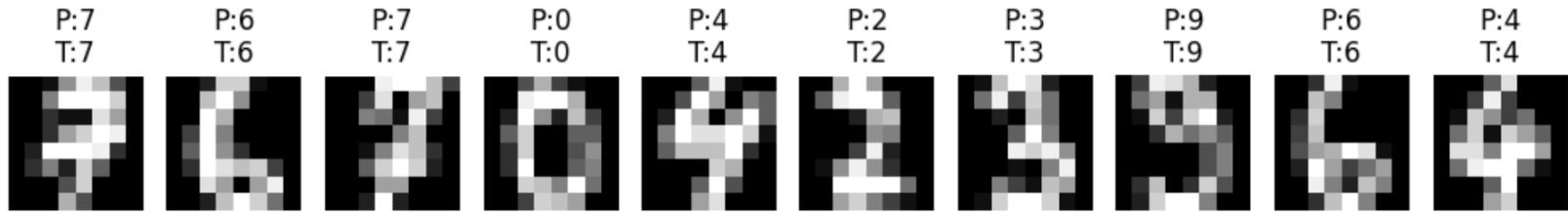
```
# 預測測試集的標籤  
predicted_labels = knn.predict(X_test)
```

接下來，我們會隨機選擇 10 張測試集中的數字圖片，並顯示它們的分類結果。

```
import matplotlib.pyplot as plt  
import numpy as np  
  
# 定義顯示數字圖片的函數  
def plot_digits(data, predicted_labels, true_labels, title):  
    fig, axes = plt.subplots(1, 10, figsize=(10, 3))  
    for i, ax in enumerate(axes):  
        ax.set_axis_off() # 關閉坐標軸  
        ax.imshow(data[i].reshape(8, 8), cmap='gray') # 顯示圖片  
        ax.set_title(f"P:{predicted_labels[i]}\nT:{true_labels[i]}") # 顯示預測結果與真實標籤  
    plt.suptitle(title)  
    plt.tight_layout()  
    plt.show()  
  
# 隨機選擇 10 個數字，顯示其分類結果  
rng = np.random.default_rng()  
indices = rng.choice(len(X_test), size=10, replace=False)  
plot_digits(X_test[indices], predicted_labels[indices], y_test[indices], "KNN Classification")
```

# kNN Exercise: Solution (5)

KNN Classification



在這段程式碼中，我們隨機選擇 10 張圖片並顯示它們的預測標籤（P）和真實標籤（T）。這有助於我們理解模型的表現。

```
# 顯示預測標籤與真實標籤的比較
print("Predicted labels: ", predicted_labels[indices])
print("True labels:      ", y_test[indices])
```

```
Predicted labels: [7 6 7 0 4 2 3 9 6 4]
True labels:      [7 6 7 0 4 2 3 9 6 4]
```

# kNN Exercise: Solution (6)

---

## 4. 計算分類準確率

最後，我們計算模型的準確率，即預測正確的數字佔總數字的比例。

```
from sklearn.metrics import accuracy_score  
  
# 計算 KNN 分類準確率  
accuracy = accuracy_score(y_test, predicted_labels)  
  
# 輸出準確率  
print(f"KNN 分類準確率: {accuracy * 100:.2f}%")
```

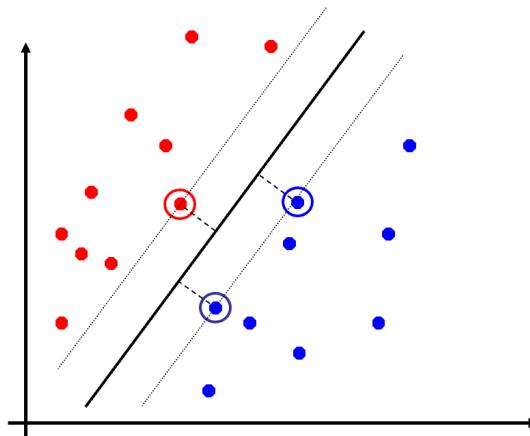
KNN 分類準確率: 98.89%

這段程式碼計算並顯示模型的準確率，告訴我們模型在測試集上的表現。

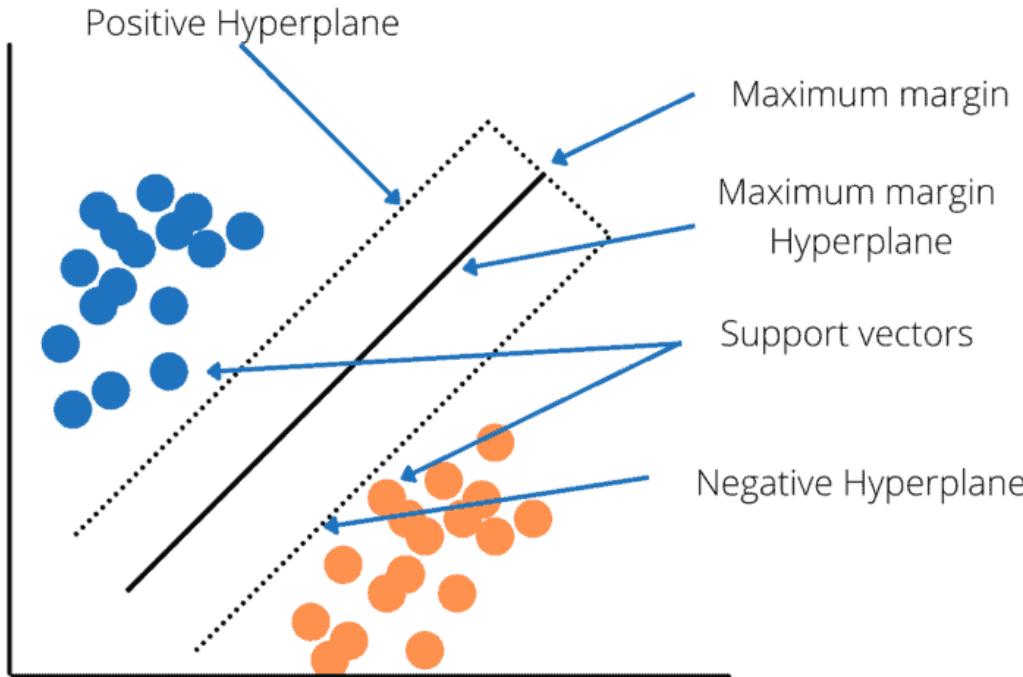
# Another Tool : Support Vector Machines

---

- 最大化邊界 ( Maximizing the margin )
- 只有支持向量 ( Support Vectors ) 對於模型來說是重要的。
- 其他訓練樣本可以忽略
- 支持向量機 ( Support Vector Machines, SVMs ) 找到具有最大邊界的分隔線



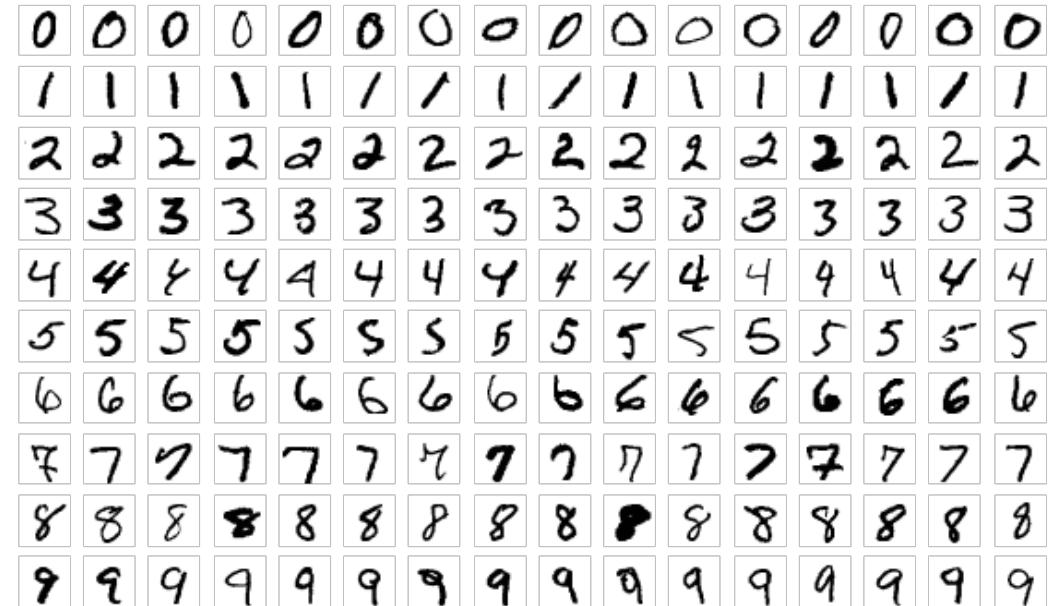
# Example: SVMs



# SVM 實作手寫數字分類

---

- Google Colab link: <https://shorturl.at/GIuIu>
- 請用在打開檔案後，登入自己的 Google 帳號



# SVM Exercise: Solution (1)

---

然後，我們定義 SVM 模型，並使用訓練集來進行訓練。這裡的 `k=3` 表示每次分類時會參考最近的 3 個鄰居。

```
from sklearn.svm import SVC  
  
# 定義 SVM 模型，選擇適當的 kernel 和 C 值  
svm = SVC(kernel='rbf', C=1.0, gamma='scale')  
  
# 訓練 SVM 模型  
svm.fit(X_train, y_train)
```

SVC()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

這段程式碼建立了 SVM 模型並用訓練集數據來訓練它。

# SVM Exercise: Solution (2)

---

## 3. 預測結果並視覺化分類結果

一旦模型訓練完成，我們可以用測試集數據來進行預測。這裡的 `predict` 函數會返回模型對測試集每一張圖片的預測結果。

```
# 預測測試集的標籤  
predicted_labels = svm.predict(X_test)
```

接下來，我們會隨機選擇 10 張測試集中的數字圖片，並顯示它們的分類結果。

- SVM 和 kNN 的分類準確率，哪個比較高？
- 複雜的模型一定表現比較好嗎？
- You can try on other datasets.

# Other Datasets in sklearn...

---

- Iris：用於花卉分類的資料集，這是機器學習中的經典範例，包含 150 筆樣本，每筆樣本有 4 個特徵（例如花瓣長度、寬度等）。（共 150 筆資料，4 個特徵（花萼長度、花萼寬度等），3 個類別。）
- Wine：用於葡萄酒分類的資料集，包含不同種類葡萄酒的化學成分數據。（178 筆葡萄酒樣本，13 個化學屬性，3 個分類標籤。）
- Breast Cancer：用於預測乳腺癌良性或惡性的資料集

```
from sklearn.datasets import load_wine
wine = load_wine()
data, labels = wine.data, wine.target
```

```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
data, labels = cancer.data, cancer.target
```

```
from sklearn.datasets import load_iris
iris = load_iris()
data, labels = iris.data, iris.target
```

# Clustering

---

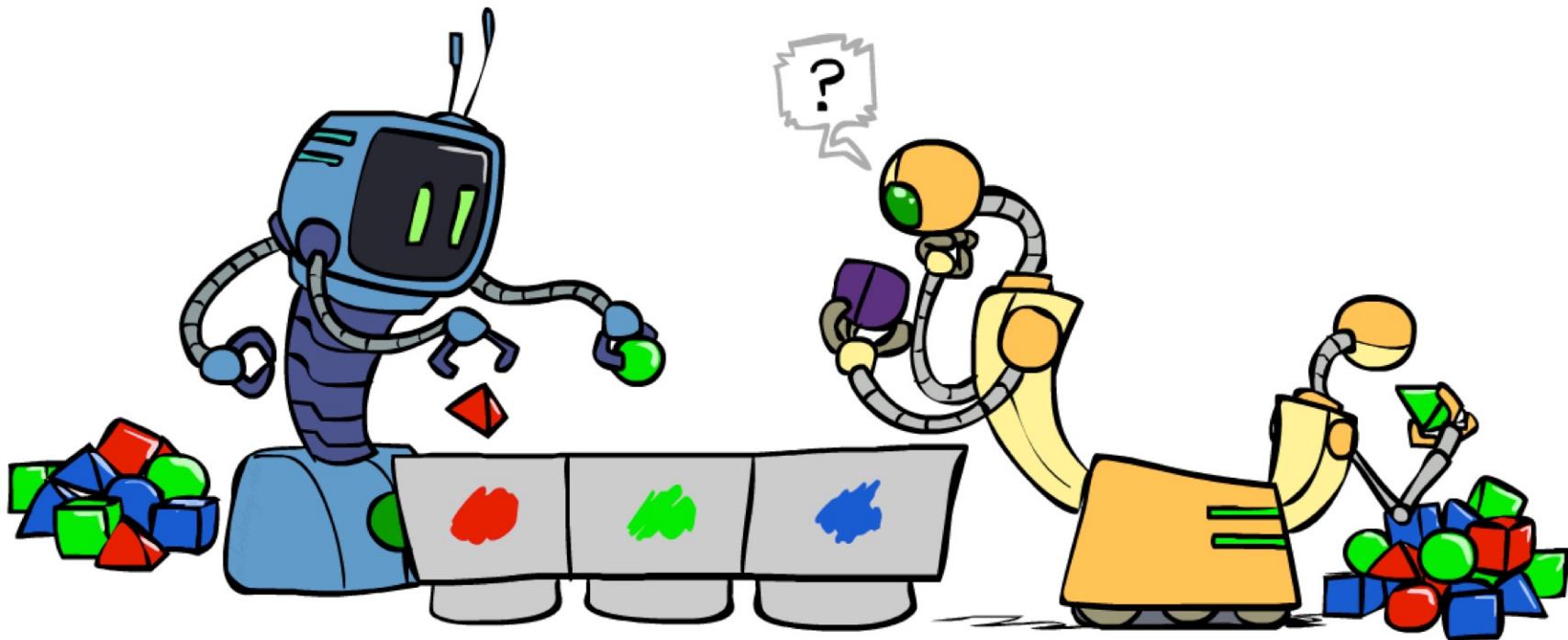
分群系統：

- 非監督式學習
- 在未標記的資料中尋找特徵
  - 對電子郵件或搜尋結果進行分組
  - 依照顧客的特性進行分群
  - 檢測程式的異常行為
- 適用於「不知道具體要尋找什麼」的情況
  - 需要 data, 但不需要 label
- 分群的結果常常很亂



# Clustering

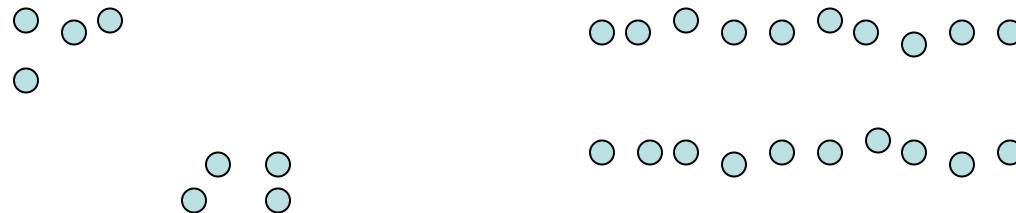
---



# Clustering

---

- 基本概念：將相似的 instance 分在一起
- 以 2D 座標點為例：



- “similar”的意思是什麼？
  - One option: squared Euclidean distance

$$\text{dist}(x, y) = (x - y)^\top (x - y) = \sum_i (x_i - y_i)^2$$

# Review: How to Measure Distance?

---

- L2 Norm (Euclidian distance):

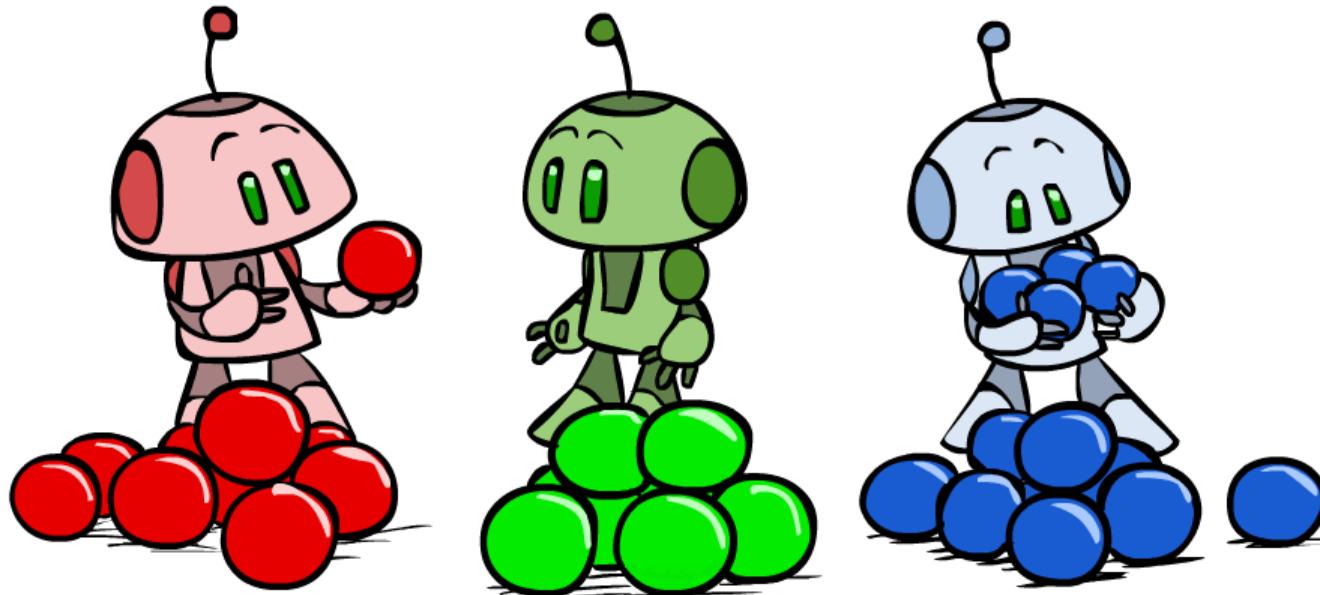
$$\|x_i - x_j\| = \sqrt{\sum_{q=1}^m (x_{i,q} - x_{j,q})^2}$$

- L1 Norm:

$$\|x_i - x_j\|_1 = \sum_{q=1}^m |x_{i,q} - x_{j,q}|$$

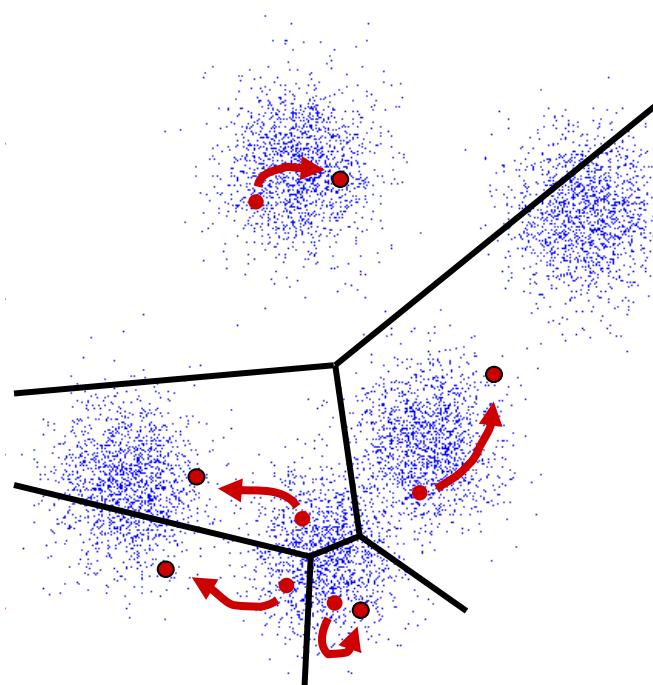
# K-Means

---



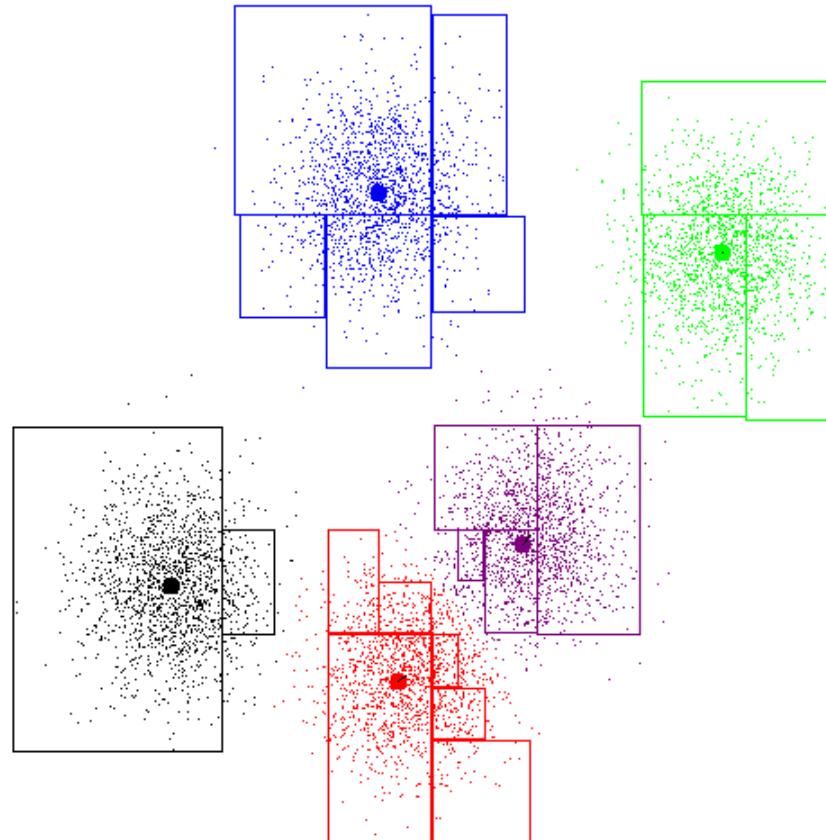
# K-Means

- K-Means 是一個迭代的分群演  
算法：
  - 隨機選擇 K 個點作為群中心
  - 重複以下步驟：
    - 將 data instances 分配到距離最  
近的群中心所屬的群
    - 更新每個群的中心為該群內且偶  
點的平均值
  - 當所有點的分配不再改變時 ->  
停止



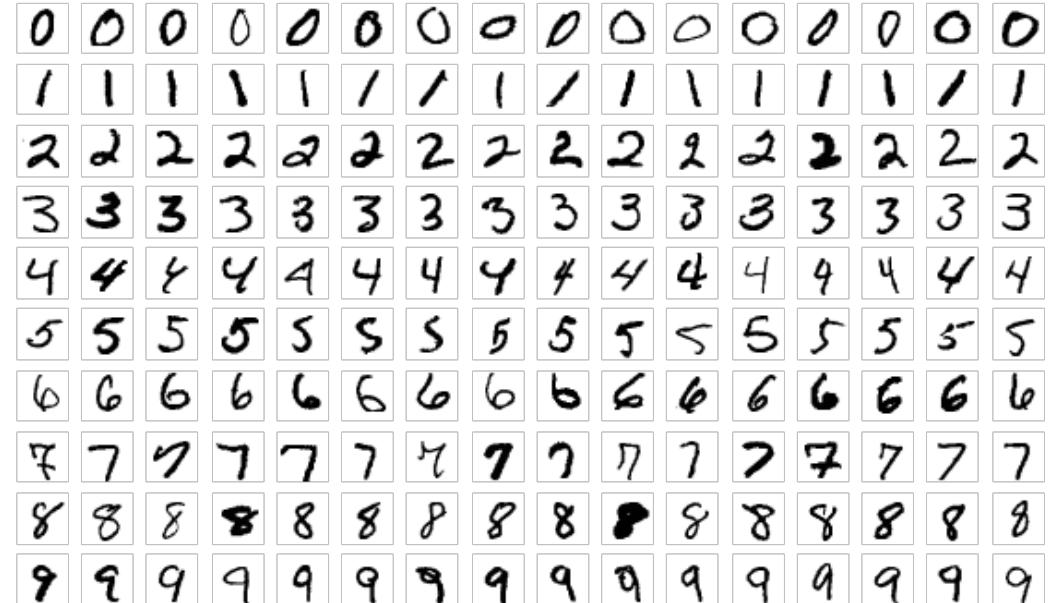
# K-Means Example

---



# K-Means 實作手寫數字分群

- Google Colab link: <https://shorturl.at/Ih9KE>
  - 請在打開檔案後，登入自己的 Google 帳號



# K-Means Exercise: Solution (1)

---

## 1. 載入手寫數字數據集

我們將使用 `sklearn` 提供的手寫數字數據集，這個數據集包含 1797 張手寫的數字圖片，每張圖片是 8x8 的大小，經過展平後變成 64 維的向量。

```
from sklearn.datasets import load_digits

# 下載手寫數字數據集
digits = load_digits()

# 數字圖片的數據
data = digits.data # (1797, 64)，每個數字是一個 8x8 的圖像展平為 64 維
print(data.shape)

# 實際的數字標籤 (0-9)
labels = digits.target
```

(1797, 64)

這段程式碼會將數據集中的圖片數據 (`data`) 和對應的標籤 (`labels`) 分別存儲在 `data` 和 `labels` 變數中。

# K-Means Exercise: Solution (2)

---

## 2. 使用 K-means 分群演算法

K-means 是一種常見的非監督式學習演算法，用來將資料點分成多個群 (cluster)。這裡將數字數據分成 10 群，因為我們有 10 個數字 (0-9)。

```
from sklearn.cluster import KMeans  
  
# 使用 K-means 分群，將資料分為 10 顆群  
kmeans = KMeans(n_clusters=10, random_state=42, n_init=10)  
kmeans.fit(data)  
  
# 得到每個資料點的分群結果  
cluster_labels = kmeans.labels_
```

`KMeans` 會將數據分成 10 顆群，並且將每個資料點對應到一個群標籤。這些標籤會保存在 `cluster_labels` 中。

# K-Means Exercise: Solution (3)

---

## 3. 視覺化分群結果

接下來，我們將視覺化一些隨機選擇的數字圖片，並且展示它們的 K-means 分群結果和真實標籤。

```
import matplotlib.pyplot as plt
import numpy as np

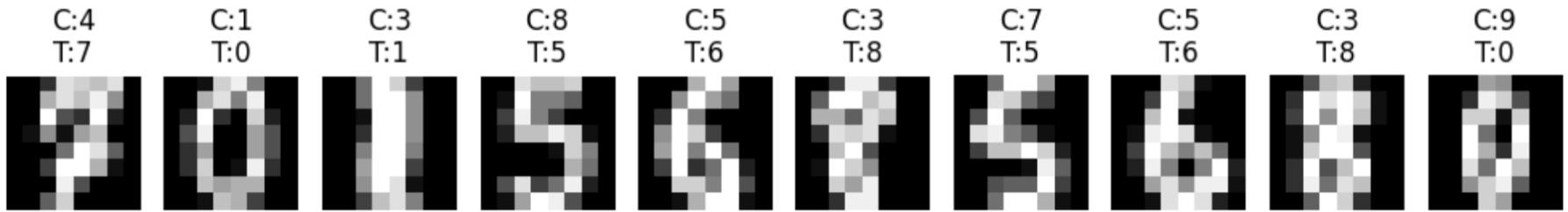
# 用來顯示數字圖片的函數
def plot_digits(data, cluster_labels, true_labels, title):
    fig, axes = plt.subplots(1, 10, figsize=(10, 3))
    for i, ax in enumerate(axes):
        ax.set_axis_off() # 關閉坐標軸
        ax.imshow(data[i].reshape(8, 8), cmap='gray') # 顯示圖片
        ax.set_title(f"C:{cluster_labels[i]}\nT:{true_labels[i]}") # 顯示分群結果和真實標籤
    plt.suptitle(title)
    plt.tight_layout()
    plt.show()

# 隨機選擇 10 個數字，顯示其分群結果
rng = np.random.default_rng()
indices = rng.choice(len(data), size=20, replace=False)
plot_digits(data[indices], cluster_labels[indices], labels[indices], "K-means Clustering")
```

# K-Means Exercise: Solution (4)

---

K-means Clustering



這段程式碼會隨機選擇 10 個數字圖片，並將每張圖片的 K-means 分群結果 (C) 和真實標籤 (T) 顯示在標題中。

# K-Means Exercise: Solution (5)

## 4. 計算分群的準確度

由於 K-means 是非監督式學習，我們無法直接知道分群的結果是否和真實標籤一致（例如：群編號為 5 不代表這個影像所分到的真實數字類別是 5）。因此，我們需要透過比較每個分群的標籤和真實標籤，計算 K-means 分群結果的準確度。

```
from sklearn.metrics import accuracy_score
from scipy.optimize import linear_sum_assignment

# 計算分群準確度
def calculate_accuracy(true_labels, cluster_labels):
    num_classes = len(np.unique(true_labels))
    confusion_matrix = np.zeros((num_classes, num_classes), dtype=int)

    # 建立 confusion matrix
    for true, cluster in zip(true_labels, cluster_labels):
        confusion_matrix[true, cluster] += 1

    # 使用匈牙利算法找到最佳配對
    row_ind, col_ind = linear_sum_assignment(-confusion_matrix) # 最大化配對
    matching = dict(zip(col_ind, row_ind))

    # 重新映射分群標籤
    remapped_labels = np.array([matching[label] for label in cluster_labels])

    # 計算 accuracy
    return accuracy_score(true_labels, remapped_labels)

# 計算 K-means 的準確度
accuracy = calculate_accuracy(labels, cluster_labels)
print(f"K-means 分群準確率: {accuracy * 100:.2f}%)
```

K-means 分群準確率: 79.35%

這段程式碼會計算分群的準確度，並輸出結果。由於 K-means 的分群標籤不一定和真實標籤一致，我們需要使用匈牙利算法來尋找最佳的配對方式。