سنقوم برحلة تطبيقية لاستكشاف أحد فروع الذكاء الاصطناعي الكلاسيكية والمؤثرة :**النظم الخبيرة (Expert Systems).** تهدف هذه الورشة إلى سد الفجوة بين النظرية والتطبيق، حيث سنبتعد عن النماذج التي تعمل كـ "صندوق أسود (Black Box)"، وسنركز على بناء نظام قائم على المعرفة يتميز بالشفافية والقدرة على تفسير قراراته المنطقية.

منهجية التطبيق (خارطة الطريق في مرحلتين)

لتحقيق أهدافنا، ستكون رحلتنا مقسمة إلى **مرحلتين رئيسيتين** متكاملتين:

- **المرحلة الأولى: التحليل المفاهيمي والتصميم المعماري**
  - في هذه المرحلة التأسيسية، سنركز على بناء الإطار النظري والتصميم الهندسي للنظام. سنبدأ بالإجابة على الأسئلة الجوهرية "لماذا" و "ماذا:"
1. **تحليل المشكلة الجوهرية :**سنقوم بتحليل أزمة استخلاص المعرفة البشرية والحفاظ عليها، وهي المشكلة الأساسية التي جاءت النظم الخبيرة لحلها.
2. **تحديد المكونات المعمارية :**سنقوم بتفكيك النظام الخبير إلى وحداته الوظيفية (قاعدة المعرفة، الذاكرة العاملة، محرك الاستدلال، إلخ)، مع تعريف دقيق لدور كل مكون.
3. **تصميم هياكل البيانات :**سنقوم بترجمة مفاهيم "الحقيقة (Fact)" و "القاعدة (Rule)" إلى مخططات برمجية باستخدام مبادئ البرمجة كائنية التوجه(OOP)
4. **صياغة خوارزمية الاستدلال :**سنضع المخطط المنطقي التفصيلي لعمل محرك الاستدلال الأمامي (Forward Chaining)، وتحديد خطواته وشروط توقفه.

- **المرحلة الثانية: التنفيذ البرمجي والتكامل الوظيفي**
  - هنا، سنقوم بتحويل المخططات والتصاميم من المرحلة الأولى إلى كود بايثون فعال. سنركز على "البناء" العملي:
1. **تنفيذ هياكل المعرفة :**كتابة الكود البرمجي الخاص بأصناف Factو Ruleبناءً على التصميم الذي تم وضعه.
2. **برمجة محرك الاستدلال :**تحويل الخوارزمية المخطط لها إلى دالة (run) وظيفية وفعالة داخل هيكل النظام.

في هذه المرحلة التأسيسية الحاسمة، سنقوم ببناء فهم عميق ومتكامل للنظم الخبيرة. سنبدأ من "الماذا" ظهرت هذه التقنية، مرورًا بـ "ما هي" مكوناتها المعمارية الدقيقة، وانتهاءً بـ "كيف" نترجم هذا الفهم النظري إلى تصميم برمجي منهجي ومدروس. هذه المرحلة تضع الأساس النظري والتصميمي لكل ما سيأتي لاحقًا.

القسم الأول: التحليل المفاهيمي وبناء الإطار النظري

قبل أن يكون النظام الخبير حلاً تقنيًا، كان حلاً لمشكلة تنظيمية واقتصادية جوهرية :**هشاشة وفناء الخبرة البشرية** .تخيل خبيرًا قضى عقودًا في مجاله (مثل تشخيص الأعطال الميكانيكية المعقدة)، يمتلك معرفة لا تقدر بثمن. هذه المعرفة ليست مجرد بيانات أو معلومات مدونة في كتيبات، بل هي مزيج من:

- **القواعد الصريحة" :(Explicit Knowledge)** إذا سمعت صوت طقطقة عند دوران المحرك، افحص مستوى الزيت".
- **الحدس والقواعد الذهنية" :(Heuristics)** في 90% من الحالات المشابهة، يكون السبب هو فلتر الهواء المتسخ، لذا ابدأ به لتوفير الوقت".
- **المعرفة الضمنية :(Tacit Knowledge)** القدرة على تمييز الفروق الدقيقة في أصوات المحرك التي لا يمكن وصفها بالكلمات.

المشكلة تكمن في أن هذه الخبرة مرتبطة بوجود هذا الخبير. تقاعده، مرضه، أو حتى انشغاله يعني أن هذه "المكتبة الذهنية" تصبح غير متاحة، مما قد يؤدي إلى خسائر فادحة. السؤال الذي طرحه رواد الذكاء الاصطناعي هو" :**كيف يمكننا استخلاص (Elicit)، هيكلة(Structure) ، حفظ(Preserve) ، وتوزيع (Distribute) هذه الخبرة البشرية النادرة والقيمة كأداة آلية موثوقة ومتاحة دائمًا؟"**

التعريف الدقيق :النظام الخبير (Expert System) هو نظام حاسوبي برمجي مصمم لمحاكاة قدرات اتخاذ القرار لدى خبير بشري في مجال معرفي محدد وضيق(Specific Domain) ، وذلك عبر تمثيل المعرفة البشرية بشكل صريح وتطبيق آليات استدلال منطقية عليها للوصول إلى استنتاجات.

الفرق الجوهري الذي يحدد هويته :يكمن السر في المبدأ المعماري الذي يميزه عن أي برنامج تقليدي آخر :**الفصل التام بين المعرفة (Knowledge) والمنطق.(Logic)**

1. **البرمجة التقليدية :(Algorithm-Centric)** المنطق (خوارزمية المعالجة) والمعرفة (البيانات المضمنة في الشروط) متشابكان بعمق داخل الكود. لتغيير قاعدة عمل، مثل إضافة ضريبة جديدة، يتطلب الأمر تعديل الكود البرمجي وإعادة ترجمته.
2. **النظم الخبيرة :(Knowledge-Centric)** تقوم على فصل جذري بين:
   - **قاعدة المعرفة :(Knowledge Base)** تحتوي على "ماذا يعرفه الخبير" (الحقائق والقواعد).
   - **محرك الاستدلال :(Inference Engine)** يحتوي على "كيف يفكر الخبير" (آلية تطبيق القواعد).
   هذا الفصل هو ميزة استراتيجية، حيث يسمح لـ"خبراء المجال" بتحديث وتوسيع قاعدة المعرفة دون الحاجة إلى أي معرفة برمجية، ودون المساس بمنطق عمل النظام.

لفهم كيفية عمل النظام الخبير، يجب تفكيكه إلى مكوناته الوظيفية الخمسة:

1. **قاعدة المعرفة (Knowledge Base): العقل الدائم**
   - **الوظيفة:** المستودع الدائم للمعرفة المتخصصة، يمثل "ذاكرة الخبير طويلة الأمد" المكتسبة عبر السنين.
   - **المحتوى:** لا تحتوي على بيانات خام، بل على نماذج معرفية. أشهرها **القواعد الإنتاجية (Production Rules)** على شكل IF (مقدمات/شروط) THEN (نتائج/استنتاجات).
   - **مثال دقيق**

IF (Temperature > 102°F) AND (Symptom = 'Cough') AND (Cough_Type = 'Dry')
THEN (Hypothesis = 'Viral Infection', Confidence = 0.7).

2. **الذاكرة العاملة (Working Memory / Fact Base): سبورة المنطق المؤقتة**
   - **الوظيفة:** المكون الديناميكي الذي يمثل "وعي" النظام الحالي بالمشكلة، أو "ذاكرة الخبير قصيرة الأمد" التي تركز على الحالة قيد التحليل.
   - **المحتوى:** تخزن الحقائق المتعلقة بالجلسة الحالية فقط، وهي نوعان: **الحقائق الأولية** (المعطاة من المستخدم) و**الحقائق المستنتجة** (التي يضيفها النظام أثناء عمله). تبدأ فارغة مع كل جلسة جديدة.
3. **محرك الاستدلال (Inference Engine): العقل المفكر**
   - **الوظيفة:** "وحدة المعالجة المنطقية" للنظام. هو المكون المسؤول عن تنسيق التفاعل بين قاعدة المعرفة والذاكرة العاملة لتوليد استنتاجات جديدة.
   - **الآلية) دورة التعرف والتصرف:(Recognize-Act Cycle -**
     1. **المطابقة (Match):** البحث في قاعدة المعرفة عن كل القواعد التي تتطابق شروطها مع الحقائق الموجودة حاليًا في الذاكرة العاملة.
     2. **حل التعارض (Conflict Resolution):** إذا كانت هناك أكثر من قاعدة قابلة للتفعيل، يجب على المحرك اختيار واحدة فقط لتنفيذها باستخدام استراتيجية محددة (مثل: الأولوية، التحديد، الحداثة).
     3. **التنفيذ (Act/Fire):** تطبيق القاعدة المختارة، مما يؤدي عادةً إلى تعديل الذاكرة العاملة (إضافة حقيقة جديدة).
   - **الاستراتيجيات الرئيسية:**
     1. **الاستدلال الأمامي (Forward Chaining):** يبدأ من الحقائق المعروفة ويعمل نحو استنتاج. يُعرف بأنه "مدفوع بالبيانات (Data-Driven)" ومناسب لمشكلات التشخيص والمراقبة والتصميم. ("لدي هذه الأعراض، ما هو المرض؟").
     2. **الاستدلال الخلفي (Backward Chaining):** يبدأ من هدف أو فرضية محتملة ويحاول إثباتها بالبحث عن أدلة تدعمها. يُعرف بأنه "مدفوع بالهدف (Goal-Driven)" ومناسب لمشكلات التحقق والتخطيط. ("هل هذا المريض مصاب بالإنفلونزا؟ دعنا نبحث عن الأدلة").
4. **آلية الشرح (Explanation Facility): بناء الثقة والشفافية**
   - **الوظيفة:** الميزة التي تمنح النظم الخبيرة شفافيتها وتميزها عن نماذج "الصندوق الأسود". هي قدرة النظام على تبرير استنتاجاته والإجابة على أسئلة:
     1. **"كيف؟" (How?):** "كيف توصلت إلى هذا الاستنتاج؟" (يعرض سلسلة القواعد التي تم تفعيلها).
     2. **"لماذا؟" (Why?):** "لماذا تسألني هذا السؤال الآن؟" (يشرح أن السؤال هو للتحقق من شرط في قاعدة يحاول تفعيلها).
5. **واجهة المستخدم (User Interface): الجسر التواصلي**
   - **الوظيفة:** تسهيل الحوار بين المستخدم غير المتخصص والنظام، من خلال جمع المعلومات بطريقة منظمة وعرض النتائج والتوصيات بلغة بشرية واضحة ومفهومة.

القسم الثاني: من المفهوم إلى التصميم البرمجي

في هذا القسم، سنقوم بترجمة المكونات المعمارية المجردة التي ناقشناها (قاعدة المعرفة، الذاكرة العاملة، محرك الاستدلال، آلية الشرح، واجهة المستخدم) إلى "مخططات هندسية" برمجية دقيقة. سنستخدم مبادئ البرمجة كائنية التوجه (OOP) لأنها مثالية لتجسيد هذه المفاهيم بشكل منظم ومستقل.

أولا) تصميم وحدات المعرفة (تجسيد قاعدة المعرفة)

قاعدة المعرفة تتكون من حقائق وقواعد. لذا، يجب أن نصمم أولاً "قوالب" لهذه الوحدات الأساسية.

- **تصميم كائن الحقيقة:(Fact Class)**
  - **الغرض :**تمثيل معلومة واحدة، ذرية، وغير قابلة للتجزئة، مثل .(Symptom = 'Cough')هذا هو أصغر مكون في قاعدة المعرفة والذاكرة العاملة.
  - **البيانات :(Attributes)** سيحتوي على خاصية واحدة جوهرية) description :وصف نصي فريد يمثل هوية الحقيقة.(
  - **الوظائف:(Methods)**
    - **المقارنة للمساواة :(__eq__)** وظيفة حيوية تُعرّف متى تكون حقيقتان متطابقتين. هي أساس عملية المطابقة في محرك الاستدلال. يجب أن تتحقق من نوع الكائن الآخر (isinstance) ثم تقارن الوصف.
    - **التجزئة :(__hash__)** ضرورية لجعل كائنات Factقابلة للتخزين في هياكل بيانات سريعة وفعالة مثل المجموعات (set) والقواميس(dict) ، مما يسرع البحث بشكل كبير. يجب أن تعتمد على نفس الخاصية المستخدمة في المقارنة.
    - **التمثيل الرسمي :(__repr__)** وظيفة مساعدة لا غنى عنها للمطور، حيث تعرض الحقيقة بشكل واضح لا لبس فيه أثناء التصحيح والتتبع.
- **تصميم كائن القاعدة:(Rule Class)**
  - **الغرض :**تمثيل علاقة منطقية سببية واحدة(IF-THEN) ، وهي جوهر خبرة الخبير.
  - **البيانات:(Attributes)**
    - :conditionsقائمة من كائنات Factالتي يجب أن تكون جميعها صحيحة لتفعيل القاعدة.
    - :conclusionكائن Factواحد يمثل الحقيقة الجديدة التي سيتم استنتاجها.
  - **الوظائف:(Methods)**
    - **التمثيل الرسمي :(__repr__)** وظيفة مهمة جدًا لعرض القاعدة بشكل مفهوم للبشر .**هذه الوظيفة هي التنفيذ المبدئي لـ "آلية الشرح"**، حيث تمكننا من طباعة سلسلة التفكير المنطقي للنظام.

ثانيا تصميم هيكل النظام الخبير العام(SimpleExpertSystem)

هذا الكائن سيمثل الهيكل العام للنظام وسيكون بمثابة الحاوية التي تجمع كل المكونات وتنسق عملها. سنرى الآن كيف يتم تجسيد كل مكون معماري داخل هذا الهيكل.

- **تصميم المكونات 1 و 2: قاعدة المعرفة والذاكرة العاملة**
  - **الغرض :**توفير حاويات لتخزين المعرفة الدائمة (rules) والحقائق المؤقتة.(facts)
  - **التصميم :**داخل دالة البناء __init__ للفئة :SimpleExpertSystem
    - :[] = self.rules سيتم تعريف قائمة فارغة لتمثل **قاعدة المعرفة .(Knowledge Base)** ستُستخدم هذه القائمة لتخزين جميع كائنات Ruleالتي يعرفها النظام.
    - :set() = self.facts سيتم تعريف مجموعة فارغة لتمثل **الذاكرة العاملة (Working Memory).** اختيار setهو قرار تصميمي لتحسين الأداء (بحث سريع) ومنع تكرار الحقائق تلقائيًا.
  - **الوظائف المساعدة:**
    - :add_rule(rule)وظيفة لإضافة قاعدة جديدة إلى .self.rules
    - :add_fact(fact)وظيفة لإضافة حقيقة جديدة إلى .self.facts
- **تصميم المكون 3: محرك الاستدلال(run Method)**
  - **الغرض :**تصميم الخوارزمية التي ستقوم بعملية التفكير المنطقي (الاستدلال الأمامي).
  - **التصميم :**سيتم تصميم دالة run()لتنفيذ دورة "التعرف والتصرف:"
    - **ابدأ حلقة تكرارية لا نهائية :(Loop)** تمثل استمرار عملية التفكير.
    - **المطابقة :(Match)** داخل الحلقة، سيتم البحث في self.rulesعن كل القواعد التي تتطابق شروطها مع الحقائق الموجودة في .self.facts
    - **التصفية :**سيتم استبعاد القواعد التي استنتاجاتها موجودة بالفعل في self.factsلمنع الحلقات اللانهائية.
    - **شرط التوقف :**إذا لم يتم العثور على أي قواعد قابلة للتفعيل، يتم كسر الحلقة.
    - **حل التعارض :(Conflict Resolution)** للتصميم المبدئي، سيتم اختيار القاعدة الأولى في قائمة القواعد القابلة للتفعيل.
    - **التنفيذ :(Act/Fire)** سيتم إضافة استنتاج القاعدة المختارة إلى .self.facts
- **تصميم المكون 4: آلية الشرح (جزء من run Method)**
  - **الغرض :**تصميم كيفية عرض النظام لتفكيره المنطقي.
  - **التصميم :**سيتم دمج آلية الشرح داخل خطوة "التنفيذ" في دالة run(). بعد اختيار قاعدة لتفعيلها، وقبل إضافة استنتاجها، سيقوم النظام بطباعة تمثيل القاعدة (باستخدام دالة __repr__التي صممناها في Rule). هذا يوضح للمستخدم "لماذا" تم استنتاج الحقيقة الجديدة.
    - مثال للمخرج المخطط له] :تفعيل القاعدة[("Rule(IF Fact("A") THEN Fact("B"))
- **تصميم المكون 5: واجهة المستخدم(get_initial_facts Method)**
  - **الغرض :**تصميم كيفية تجميع النظام للمعلومات الأولية من المستخدم.
  - **التصميم :**سيتم تصميم دالة get_initial_facts()لتكون مسؤولة عن:
    - طباعة رسائل ترحيبية وإرشادية.
    - طرح أسئلة محددة على المستخدم.
    - تحليل إجابات المستخدم وتحويلها إلى كائنات Factمنظمة.
    - إضافة هذه الحقائق الأولية إلى الذاكرة العاملة.(self.facts)

ثالثا) تصميم التطبيق المتخصص(InternetTroubleshooter Class)

هنا نطبق مبدأ "الفصل" بشكل عملي وأنيق باستخدام الوراثة، ونربط كل التصاميم السابقة معًا.

- **الغرض :**بناء نظام خبير متخصص في مجال معين (تشخيص مشاكل الإنترنت).
- **التصميم:**
  - سيكون كائنًا جديدًا (InternetTroubleshooter) **يرث (Inherits)** من SimpleExpertSystem.
  - سيحصل هذا الكائن تلقائيًا على كل الوظائف والتصاميم التي وضعناها (الذاكرة العاملة، قاعدة المعرفة الفارغة، محرك الاستدلال، واجهة المستخدم الأولية).
  - وظيفته الرئيسية هي **توفير المعرفة المتخصصة** من خلال دالة خاصة مثل _build_knowledge_base(). هذه الدالة ستقوم بـ:
    1. إنشاء جميع كائنات Fact وRule المتعلقة بمشاكل الإنترنت.
    2. استخدام وظيفة self.add_rule() الموروثة لملء **قاعدة المعرفة (self.rules)** الفارغة بالمعرفة الفعلية.

بهذه المرحلة المتكاملة، نكون قد انتقلنا من الفهم النظري العميق للمشكلة والحل، إلى وضع مخططات هندسية برمجية واضحة ومحددة ومربوطة بشكل مباشر بالمكونات المعمارية الخمسة. لدينا الآن تصميم كامل جاهز للترجمة إلى كود فعلي.

في هذه المرحلة العملية، سنقوم بتحويل "المخططات الهندسية" التي صممناها في المراحل السابقة إلى كود بايثون فعال وقابل للتشغيل. سنمر على كل جزء من الكود، من أبسط وحدة بناء إلى النظام المتكامل، مع شرح دقيق لوظيفته وأهميته وكيفية ارتباطه بالإطار النظري الأكاديمي. هذه هي مرحلة "إحياء" النظام ورؤية الأفكار المجردة وهي تعمل.

القسم الأول: بناء الأدوات - تمثيل المعرفة ككائنات (Fact و Rule)

هنا نقوم بإنشاء "الذرّات" التي تتكون منها المعرفة في نظامنا. استخدام البرمجة كائنية التوجه (OOP) يسمح لنا بتجسيد هذه المفاهيم المجردة في هياكل برمجية منظمة.

هذا الكائن هو التمثيل البرمجي لأبسط وحدة معرفية لا تقبل التجزئة.

المقطع البرمجي 1.1: تعريف الفئة ودالة البناء

```
class Fact:
    """يمثل حقيقة واحدة، غير ذرية، قابلة للتجزئة في النظام."""
    def __init__(self, description):
        """البناء دالة: تنشئ حقيقة بناءً على وصف نصي."""
        self.description = str(description)
```

- **الشرح:**
  - class Fact:: تُعرّف "صنفًا" أو "فئة" جديدة باسم Fact. الفئة هي المخطط أو القالب الذي سنستخدمه لإنشاء عدد لا نهائي من كائنات الحقائق.
  - def __init__(self, description):: هذه هي **دالة البناء (Constructor)** للفئة __init__. هو اسم خاص في بايثون يتم استدعاؤه تلقائيًا في اللحظة التي يتم فيها إنشاء كائن جديد من هذه الفئة
    - self: هو مرجع يشير إلى الكائن (Instance) نفسه الذي يتم إنشاؤه. بايثون تمرره تلقائيًا.
    - description: هو المُعطى الذي نمرره عند الإنشاء، وهو يمثل جوهر الحقيقة.
  - self.description = str(description): هنا نقوم بتخزين الوصف داخل الكائن كـ "خاصية" (Attribute) اسمها description. استخدام str() هو ممارسة دفاعية (Defensive Programming) لضمان أن الوصف دائمًا من نوع نصي، مما يمنع الأخطاء المستقبلية. **نظريًا، هذه هي اللحظة التي يتم فيها تغليف قطعة من المعرفة المجردة داخل هيكل برمجي ملموس.**

المقطع البرمجي 1.2: الدوال الخاصة للمقارنة والتمثيل

```
def __repr__(self):
    """يوفر تمثيلاً رسمياً وواضحاً للكائن لأغراض التصحيح."""
    return f'Fact("{self.description}")'

def __eq__(self, other):
    """يُعرّف منطق المساواة، وهو حيوي لعملية المطابقة في محرك الاستدلال."""
    return isinstance(other, Fact) and self.description == other.description

def __hash__(self):
    """يسمح باستخدام الحقائق في هياكل بيانات سريعة مثل المجموعات (sets)."""
    return hash(self.description)
```

- **الشرح:**
  - def __repr__(self):: هذه **دالة التمثيل الرسمي** .**(Official Representation)** هدفها ليس للمستخدم النهائي، بل للمطور. توفر تمثيلاً واضحًا لا لبس فيه للكائن، مما يسهل تصحيح الأخطاء .(Debugging) المخرج "(Factالإنترنت_لا_يعمل ("يخبرنا بالضبط ما هو نوع الكائن ومحتواه.
  - def __eq__(self, other):: هذه هي **دالة المقارنة للمساواة(Equality Comparison)** ، وهي من أهم الدوال في نظامنا. يتم استدعاؤها عند استخدام معامل المقارنة .==
    - isinstance(other, Fact):أولاً، نتحقق مما إذا كان الشيء الآخر الذي نقارنه به هو أيضًا كائن من نوع .Factهذا يمنع أخطاء المقارنة مع أنواع بيانات أخرى.
    - and self.description == other.description: إذا كان من نفس النوع، فإننا نعتبرهما متساويين فقط إذا كانت أوصافهما النصية متطابقة.
    - **الأهمية :هذه الدالة هي التي تسمح لمحرك الاستدلال بتنفيذ عملية المطابقة (Match) بشكل صحيح** .بدونها، لن يتمكن المحرك من التحقق مما إذا كانت الحقائق المطلوبة في شرط القاعدة موجودة بالفعل في الذاكرة العاملة.
  - def __hash__(self):: هذه هي **دالة التجزئة** .**(Hashing)** يتم استدعاؤها عندما نحاول وضع الكائن في بنية بيانات تعتمد على التجزئة مثل **المجموعة** .**(set)** توفر طريقة سريعة جدًا للبحث عن عنصر. القاعدة الذهبية في بايثون هي: إذا كان كائنان متساويين(__eq__ -> True) ، فيجب أن يكون لهما نفس قيمة التجزئة. أسهل طريقة لتحقيق ذلك هي تجزئة الخاصية الفريدة التي تحدد المساواة، وهي .self.description

هذا الكائن يمثل علاقة IF-THENالمنطقية التي تشكل جوهر خبرة الخبير.

المقطع البرمجي 1.3: تعريف فئة القاعدة

```
class Rule:


    def __init__(self, conditions, conclusion):
        self.conditions = conditions
        self.conclusion = conclusion

    def __repr__(self):

        return f"Rule(IF {' AND '.join(map(repr, self.conditions))} THEN
{repr(self.conclusion)})"
```

- **الشرح:**
  - def __init__(self, conditions, conclusion):: دالة البناء الخاصة بالقاعدة.
    - conditions: ستكون قائمة من كائنات Fact.
    - conclusion: ستكون كائن Factواحد.
    - تقوم الدالة بتخزين هذه المدخلات كخصائص داخل كائن القاعدة .**نظريًا، هذا هو تغليف علاقة سببية أو استدلالية داخل هيكل برمجي.**
  - def __repr__(self):: دالة التمثيل الرسمي للقاعدة.
    - map(repr, self.conditions): تقوم بتطبيق دالة (reprالتي عرفناها في Factعلى كل حقيقة في قائمة الشروط.
    - ' AND '.join(...): تقوم بدمج النصوص الناتجة مع الفاصل " AND ".

القسم الثاني: بناء الهيكل العامSimpleExpertSystem -

هذا هو الهيكل الذي سيحتوي على المعرفة والمنطق معًا، ويمثل الإطار العام القابل لإعادة الاستخدام.

المقطع البرمجي 2.1: تهيئة الهيكل الأساسي

```
class SimpleExpertSystem:
    def __init__(self):
."""قاعدة المعرفة والذاكرة العاملة: يقوم بتهيئة المكونات الرئيسية        """
        self.rules = []       #
        self.facts = set()    #

    def add_fact(self, fact):
        self.facts.add(fact)

    def add_rule(self, rule):
        self.rules.append(rule)
```

- **الشرح المفصل:**
  - عند إنشاء أي نظام خبير(my_system = SimpleExpertSystem() ، يبدأ هذا الكود بالعمل.
  - self.rules = []: تنشئ **قاعدة المعرفة** كقائمة فارغة، جاهزة لاستقبال كائنات Rule.
  - self.facts = set(): تنشئ **الذاكرة العاملة** كمجموعة فارغة، جاهزة لاستقبال كائنات Fact. اختيارset هو قرار تصميمي مهم لتحسين الأداء) بحث O(1) في المتوسط (ومنع تكرار الحقائق تلقائيًا.
  - add_fact و add_rule: هما واجهتان برمجيتان (APIs) بسيطة ونظيفة لإضافة المعرفة إلى النظام.

المقطع البرمجي 2.2: محرك الاستدلال الأمامي(run Method)

```
def run(self):
    """محرك يمثل الاستدلال الأمامي (Forward Chaining Inference Engine)."""
    print("\n" + "=" * 20 + "\nبدء تشغيل محرك الاستدلال...\n" + "=" * 20)

    # دورة التعرف والتصرف (Recognize-Act Cycle)
    while True:
        # 1. المطابقة (Match): البحث عن كل القواعد التي يمكن تفعيلها
        activated_rules = [rule for rule in self.rules
                           if all(c in self.facts for c in rule.conditions) and
                                 rule.conclusion not in self.facts]

        # 2. شرط التوقف: إذا لم تكن هناك قواعد جديدة، توقف
        if not activated_rules:
            print("... لا توجد قواعد أخرى يمكن تفعيلها. توقف محرك الاستدلال.")
            break

        # 3. حل التعارض (Conflict Resolution): استراتيجية أبسط اختيار (الأولى القاعدة)
        selected_rule = activated_rules[0]

        # 4. التنفيذ (Fire) الشرح مع
        print(f"  [تفعيل القاعدة]: {repr(selected_rule)}")
        self.facts.add(selected_rule.conclusion)
        print(f"  [مستنتجة جديدة حقيقة]: {selected_rule.conclusion}")

    self._display_final_results()  # النتائج عرض دالة استدعاء
```

- **الشرح لكل خطوة داخل run:**
  1. while True:: تبدأ حلقة لا نهائية تمثل **دورة التعرف والتصرف (Recognize-Act Cycle)** المستمرة.
  2. activated_rules = [...]: هذه هي مرحلة **المطابقة (Match)**. السطر يستخدم List Comprehension لإنشاء قائمة بالقواعد القابلة للتفعيل. الشرط المنطقي ... if هو جوهر هذه المرحلة:
     - all(c in self.facts for c in rule.conditions): يتحقق مما إذا كانت **كل** الشروط المطلوبة للقاعدة موجودة حاليًا في الذاكرة العاملة.
     - and rule.conclusion not in self.facts: هذا شرط حاسم **لمنع الدورات اللانهائية**. يضمن أننا لا نعيد استنتاج حقيقة اكتشفناها بالفعل.
  3. if not activated_rules: break: هذا هو **شرط التوقف**. بعد فحص كل القواعد، إذا كانت القائمة فارغة، فهذا يعني أن النظام وصل إلى حالة استقرار معرفي ولا يمكنه استنتاج أي شيء جديد، لذا يخرج من الحلقة.
  4. selected_rule = activated_rules[0]: هذه هي مرحلة **حل التعارض (Conflict Resolution)** في أبسط صورها. نختار ببساطة القاعدة الأولى في القائمة.
  5. **التنفيذ (Fire) والشرح:**
     - self.facts.add(selected_rule.conclusion): هذا هو "فعل" التنفيذ. نقوم بتعديل الذاكرة العاملة بإضافة الحقيقة الجديدة. هذا التعديل هو الذي قد يسمح بتفعيل قواعد جديدة في الدورة التالية.

المقطع البرمجي 2.3: واجهة المستخدم وبناء التطبيق المتخصص

```
# (داخل SimpleExpertSystem)
    def get_initial_facts(self):
        # ... الحوار كود مع المستخدم ...

# (منفصلة فئة)
class InternetTroubleshooter(SimpleExpertSystem):
    def __init__(self):
        super().__init__()
        self._build_knowledge_base()

    def _build_knowledge_base(self):
        # ... الحقائق تعريف f1, f2, ...
        # ... الحقائق وربط القواعد تعريف ...
        self.add_rule(Rule(conditions=[f1, f2], conclusion=c1))
```

- **الشرح:**
  - get_initial_facts: تمثل **واجهة المستخدم** الأولية. وظيفتها هي أن تكون الجسر بين المستخدم والنظام، حيث تترجم إجابات المستخدم إلى كائنات Fact منظمة وتضيفها إلى الذاكرة العاملة.
  - class InternetTroubleshooter(SimpleExpertSystem):: هذا هو المثال العملي لمبدأ **الفصل بين المعرفة والمنطق**. الفئة الجديدة InternetTroubleshooter **ترث (Inherits)** كل منطق المحرك من SimpleExpertSystem.
  - super().__init__(): سطر حاسم في الباني. يقوم باستدعاء باني الفئة الأم لتهيئة الهيكل (self.rules و self.facts).
  - self._build_knowledge_base(): هذه هي **قاعدة المعرفة** الفعلية. وظيفتها الوحيدة هي ملء الهيكل الموروث بالمعرفة المتخصصة (الحقائق والقواعد الخاصة بمشاكل الإنترنت).

المقطع البرمجي 2.4: نقطة الانطلاق (تشغيل النظام)

```
if __name__ == "__main__":
    # 1. إنشاء نسخة من النظام الخبير المتخصص
    troubleshooter = InternetTroubleshooter()

    # 2. جمع الحقائق الأولية من المستخدم
    should_run = troubleshooter.get_initial_facts()

    # 3. تشغيل المحرك إذا لزم الأمر
    if should_run:
        troubleshooter.run()
```

- **الشرح:**
  - if __name__ == "__main__":: يضمن أن هذا الكود يعمل فقط عند تشغيل الملف مباشرة.
  - الخطوات الثلاث تمثل السيناريو الكامل لتشغيل النظام:
    1. **الإنشاء :**بناء النظام وتحميل قاعدة معرفته.
    2. **التفاعل :**التواصل مع المستخدم لملء الذاكرة العاملة الأولية.
    3. **الاستدلال :**تشغيل محرك الاستدلال لحل المشكلة بناءً على الحقائق التي تم جمعها.
  - **نقطة التكامل :**هذه الكتلة هي التي تربط جميع المراحل السابقة معًا بالترتيب المنطقي الصحيح، وتطلق سلسلة التفاعلات الكاملة داخل النظام.