Laboratorio di Intelligenza Artificiale

Elementi di Intelligenza Artificiale in Python

Luciano Capitanio

2024-09-25

Table of contents

Prefazione **4**

Introduzione **5**

Touring vs Searle: Un'Analisi del Pensiero Computazionale e della Coscienza 6

Alan Turing e il Test di Turing 7

John Searle e l'Argomento della Stanza Cinese 9

Simulazione vs. Comprensione 10

Prospettive sul Futuro dell'Intelligenza Artificiale 10

Alan Turing: Pioniere dell'Informatica 10

John Searle: Filosofia della Mente e Coscienza 11

Conclusione 12

1 Elementi di Python **13**

1.1 Sintassi e Concetti di Base 13

1.1.1 Assegnazione di Variabili** 13

1.1.2 Operazioni Aritmetiche 14

1.1.3 Controllo del Flusso (Condizioni e Cicli) 14

1.1.4 Gestione delle Eccezioni 17

1.1.5 Stampe a Video 17

1.1.6 Lettura e Scrittura di File 18

1.2 Strutture Dati 19

1.2.1 Liste 19

1.2.2 Dizionari 20

1.2.3 Set o insiemi 20

1.2.4 Set o insiemi 21

1.2.5 Tuple 22

1.3 Funzioni e Moduli 23

1.3.1 Definire e usare le funzioni 23

1.3.2 Creare e importare moduli personalizzati 23

1.3.3 Importare moduli 24

1.4 Gli oggetti 25

1.4.1 Creazione di una classe 26

1.4.2 Creazione di un oggetto 26

1.4.3 Aggiunta di un metodo alla classe 27

1.4.4 Utilizzo di più oggetti 27

1.4.5 Ereditarietà tra classi 28

2 Algoritmi **29**

2.1 Inferenza Logica 29
2.1.1 Proposizioni Logiche 30
2.1.2 Calcolo delle Proposizioni Logiche 31
2.1.3 Basi della Conoscenza 33
2.1.4 Sistemi basati sulla conoscenza 33
2.1.5 Semplice Sistema Esperto in ambito penale 35

Prefazione

Bozza del testo Labopratorio di Intelligenza Artificiale

Introduzione

L'Intelligenza Artificiale (IA) è un campo di studio che si concentra sulla creazione di macchine e software in grado di esibire comportamenti intelligenti. Questa disciplina è è stata definito per la prima volta da John McCarthy nel 1955 come "la scienza e l'ingegneria della produzione di macchine intelligenti" (McCarthy, J., Minsky, M., Rochester, N., & Shannon, C. E. (1955). A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence).

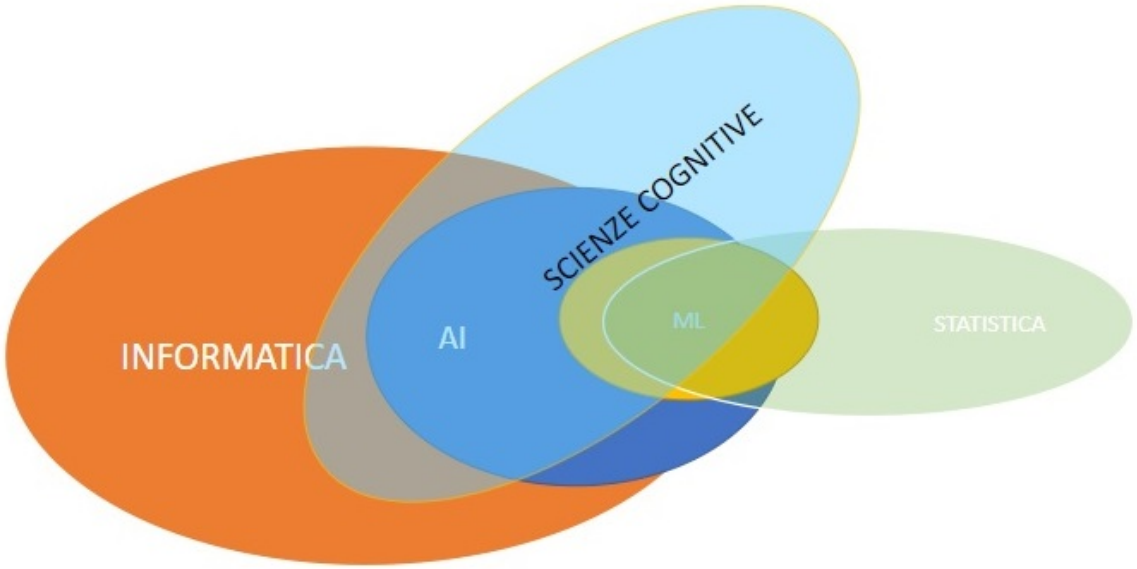

Figure 1: IA è una disciplina di confine

L'AI è contemporaneamente scienza e tecnica, disciplina di frontiera nella quale s'incontrano diversi domini del sapere. Tra questi i più riòlevanti sono : l'Informatica, la Statistica e le Scienze cognitive.

L'IA può essere suddivisa in due categorie principali: l'IA debole e l'IA forte. L'IA debole si riferisce a sistemi che sono progettati per eseguire compiti specifici, come il riconoscimento vocale o la guida autonoma. D'altra parte, l'IA forte si riferisce a sistemi che possiedono una

coscienza e un'intelligenza simili a quelle umane (Searle, J. R. (1980). Minds, brains, and programs).

L'IA sta avendo un impatto significativo su vari settori, tra cui l'assistenza sanitaria, l'istruzione, il trasporto e l'industria. Ad esempio, nell'assistenza sanitaria, l'IA può aiutare i medici a diagnosticare malattie e a sviluppare piani di trattamento personalizzati (Jiang, F., Jiang, Y., Zhi, H., Dong, Y., Li, H., Ma, S., … & Wang, Y. (2017). Artificial intelligence in healthcare: past, present and future).

Nonostante i progressi, l'IA presenta numerose sfide. Queste includono questioni etiche, come la privacy dei dati e l'impiego, e problemi tecnici, come la comprensione del linguaggio naturale e la creazione di algoritmi di apprendimento automatico efficaci (Russell, S., & Norvig, P. (2016). Artificial intelligence: a modern approach).

In conclusione, l'IA è un campo in rapida evoluzione che ha il potenziale per rivoluzionare la società. Tuttavia, è importante affrontare le sfide che presenta per garantire che i suoi benefici siano realizzati in modo etico e sostenibile.

Riferimenti Bibliografici 1. McCarthy, J., Minsky, M., Rochester, N., & Shannon, C. E. (1955). A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence. 2. Searle, J. R. (1980). Minds, brains, and programs. 3. Jiang, F., Jiang, Y., Zhi, H., Dong, Y., Li, H., Ma, S., … & Wang, Y. (2017). Artificial intelligence in healthcare: past, present and future. 4. Russell, S., & Norvig, P. (2016). Artificial intelligence: a modern approach.

Touring vs Searle: Un'Analisi del Pensiero Computazionale e della Coscienza

Alan Turing e John Searle sono due figure chiave nella filosofia della mente e nello studio dell'AI. Sebbene entrambi abbiano contribuito significativamente al campo, le loro teorie e approcci si differenziano notevolmente. Turing, con il suo famoso "Test di Turing", si concentrava sulla capacità delle macchine di simulare il comportamento umano, mentre Searle, con il suo "Argomento della Stanza Cinese", sfidava la nozione che una macchina potesse effettivamente comprendere o essere cosciente. In questo capitolo esploreremo le differenze fondamentali tra i due, analizzando le loro posizioni sul pensiero computazionale e la coscienza.

Alan Turing e il Test di Turing

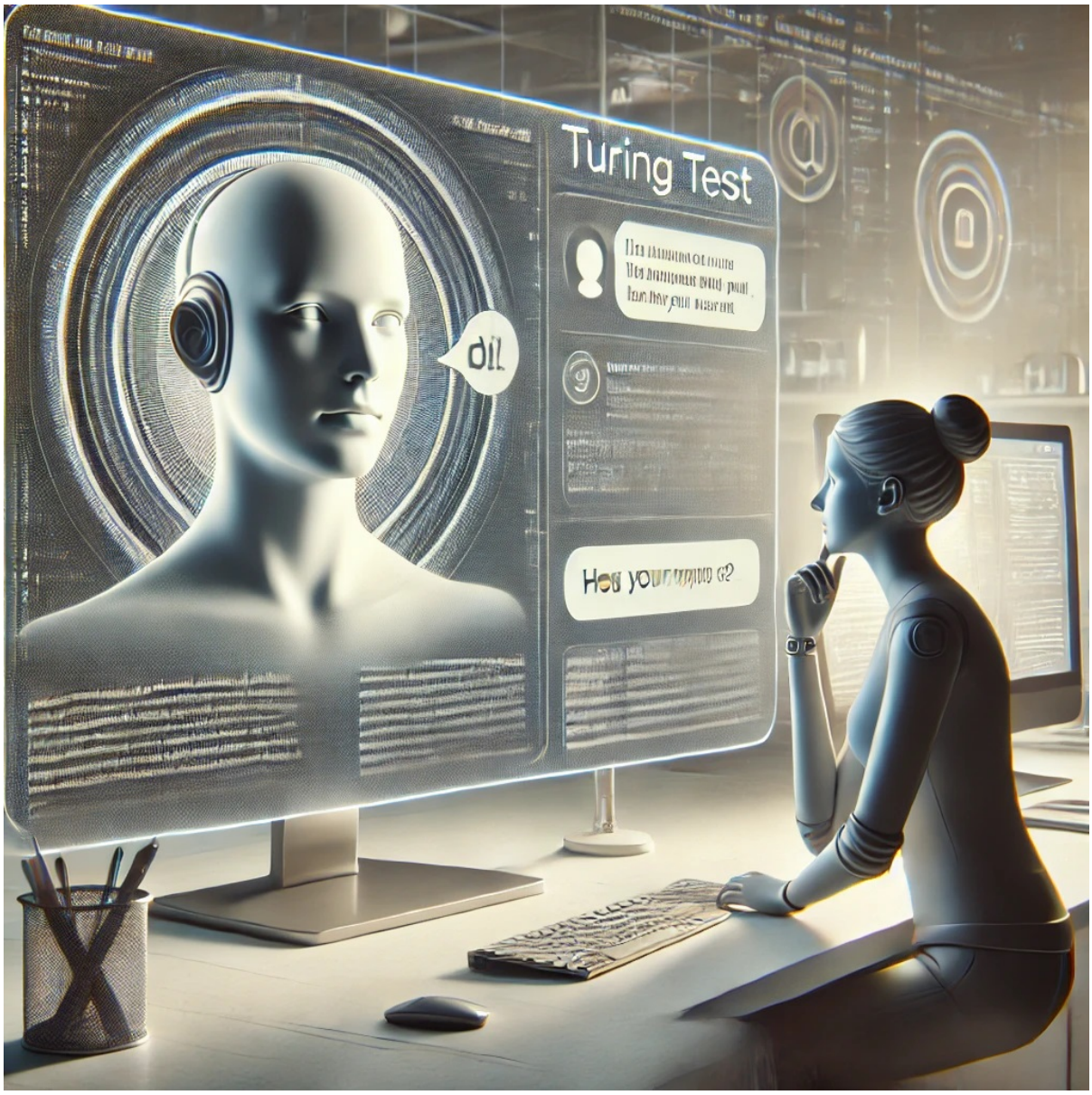

Figure 2: immagine generata da DALL-E del Test di Touring

Alan Turing, matematico britannico e pioniere dell'informatica, è noto per aver proposto quello che oggi è conosciuto come il "Test di Turing" nel suo articolo del 1950 "Computing Machinery and Intelligence". L'idea di base del test è semplice: una macchina può essere considerata "intelligente" se un essere umano, interagendo con essa attraverso una tastiera e uno schermo,

non riesce a distinguere tra le risposte della macchina e quelle di un altro essere umano. Il test non richiede che la macchina pensi o capisca nel senso umano del termine, ma solo che simuli il comportamento umano in modo convincente.

Per Turing, la domanda "Le macchine possono pensare?" è mal posta. Egli suggerisce che dovremmo piuttosto chiedere se le macchine possano esibire comportamenti che noi interpretiamo come pensiero. L'enfasi, quindi, è sulla **simulazione** dell'intelligenza piuttosto che sull'esperienza cosciente.

John Searle e l'Argomento della Stanza Cinese

Figure 3: immagine generata da DALL-E della stanza cinese di Searle

John Searle, filosofo statunitense, ha formulato nel 1980 l'ormai famoso "Argomento della Stanza Cinese" come risposta critica al Test di Turing e alle nozioni di "intelligenza artificiale forte". Searle distingue tra intelligenza artificiale "debole", che descrive sistemi che simulano l'intelligenza, e intelligenza artificiale "forte", che sostiene che una macchina può avere una

mente, una coscienza e una comprensione reale del mondo.

Nel suo esperimento mentale, Searle immagina una persona che non conosce il cinese chiusa in una stanza con una serie di istruzioni per manipolare simboli cinesi (input) e fornire risposte (output) in modo che chi osserva dall'esterno creda che la persona capisca il cinese. Tuttavia, sostiene Searle, la persona non comprende affatto il cinese; sta semplicemente seguendo delle regole per manipolare simboli. L'argomento di Searle dimostra che una macchina potrebbe eseguire compiti simili a quelli di una mente cosciente, ma senza effettivamente **comprendere** il significato di ciò che sta facendo.

Simulazione vs. Comprensione

La differenza principale tra Turing e Searle risiede nel modo in cui concepiscono l'intelligenza e la coscienza. Per Turing, l'intelligenza è essenzialmente una questione di comportamento osservabile: se una macchina può agire in modo che un osservatore la confonda con un essere umano, allora quella macchina è, in un certo senso, "intelligente". Non importa se la macchina capisce realmente o meno ciò che sta facendo; ciò che conta è che sembri farlo.

Searle, d'altra parte, sostiene che il **comportamento esterno** non è sufficiente per definire l'intelligenza o la comprensione. Anche se una macchina può sembrare intelligente, essa manca di una qualità essenziale: la comprensione **interna** o la coscienza. Secondo Searle, seguire delle regole o manipolare simboli non equivale a comprendere il significato di quei simboli. L'intelligenza artificiale "forte", quindi, sarebbe un'illusione: le macchine possono simulare l'intelligenza, ma non possono mai possedere una coscienza o una vera comprensione.

Prospettive sul Futuro dell'Intelligenza Artificiale

Il dibattito tra Turing e Searle ha implicazioni profonde per il futuro dell'intelligenza artificiale e per la filosofia della mente. Le visioni di Turing hanno alimentato lo sviluppo di AI sempre più sofisticate, capaci di eseguire compiti complessi come la traduzione automatica, il riconoscimento delle immagini e la diagnosi medica. Tuttavia, gli argomenti di Searle ci ricordano che anche le macchine più avanzate potrebbero non "capire" veramente cosa stanno facendo, sollevando interrogativi su cosa significhi davvero "pensare" e se la coscienza possa mai emergere da un sistema puramente computazionale.

Alan Turing: Pioniere dell'Informatica

Biografia

Alan Mathison Turing nacque il 23 giugno 1912 a Londra, Regno Unito. Fu un matematico, logico e crittografo britannico, riconosciuto come uno dei fondatori dell'informatica moderna.

Durante la Seconda Guerra Mondiale, lavorò presso Bletchley Park, il centro di crittoanalisi britannico, dove contribuì in modo decisivo alla decrittazione del codice Enigma, utilizzato dalle forze tedesche. Questo lavoro ha accelerato la fine del conflitto, salvando probabilmente milioni di vite.

Turing morì tragicamente l'8 giugno 1954 a Wilmslow, Regno Unito, in circostanze sospette, che furono ufficialmente dichiarate come suicidio. La sua morte sollevò interrogativi sulla discriminazione di cui fu vittima a causa della sua omosessualità, che all'epoca era illegale nel Regno Unito.

Contributi e Onorificenze

Turing è celebre per il Modello di Turing, un concetto teorico che ha gettato le basi per i computer programmabili. Nel suo celebre articolo del 1936, "On Computable Numbers", Turing introdusse la nozione di "Macchina di Turing", una macchina teorica che potesse eseguire calcoli seguendo una serie di istruzioni codificate. Questo concetto ha fondato la teoria dell'informatica.

Oltre ai suoi contributi nel campo della logica e della computazione, il suo Test di Turing (1950) è un punto di riferimento nello studio dell'intelligenza artificiale. Il test pone la questione se una macchina possa essere considerata intelligente se non è possibile distinguerla da un essere umano durante una conversazione.

Riconoscimenti Postumi

Nonostante la sua vita sia stata segnata da ingiustizie legali, Turing ha ricevuto numerosi riconoscimenti postumi. Nel 2009, il governo britannico, sotto il primo ministro Gordon Brown, emise scuse ufficiali per il trattamento che Turing aveva subito. Nel 2013, la Regina Elisabetta II concesse a Turing la grazia reale postuma. In suo onore, la Medaglia Turing, istituita nel 1966 dall'Association for Computing Machinery (ACM), è uno dei premi più prestigiosi nel campo dell'informatica.

John Searle: Filosofia della Mente e Coscienza

Biografia

John Rogers Searle nacque il 31 luglio 1932 a Denver, Colorado, Stati Uniti. Filosofo contemporaneo di grande rilievo, Searle si è concentrato principalmente su questioni relative alla filosofia del linguaggio, alla coscienza e alla filosofia della mente. Ha conseguito il dottorato presso l'Università di Oxford, e ha insegnato per la maggior parte della sua carriera accademica all'Università della California, Berkeley.

Searle ha scritto numerosi testi fondamentali, e il suo pensiero si è sviluppato all'interno della tradizione della filosofia analitica. Oltre al suo celebre "Argomento della Stanza Cinese", ha

sviluppato una teoria della coscienza basata sul realismo biologico, sostenendo che la coscienza sia un fenomeno biologico emergente dai processi del cervello.

Contributi Filosofici e Opere Chiave

Searle è conosciuto soprattutto per il suo contributo alla filosofia del linguaggio e alla filosofia della mente. Nei primi anni della sua carriera, ha lavorato sulla teoria degli atti linguistici, che esamina come il linguaggio sia utilizzato per eseguire azioni, ad esempio promettere, ordinare o chiedere.

Tuttavia, la sua opera più conosciuta è senza dubbio il suo "Argomento della Stanza Cinese", pubblicato per la prima volta nel 1980. In questo esperimento mentale, Searle sostiene che, sebbene una macchina possa eseguire compiti che simulano la comprensione del linguaggio, essa non possiede una comprensione reale dei significati o delle intenzioni dietro le parole. Questo argomento critica l'idea dell'intelligenza artificiale forte, che sostiene che le macchine possano pensare o essere coscienti come gli esseri umani.

Onorificenze e Premi

Searle ha ricevuto numerosi riconoscimenti durante la sua carriera accademica. Nel 2004, ha ricevuto il National Humanities Medal dal presidente degli Stati Uniti George W. Bush per il suo contributo alla filosofia. Nel 2006 è stato premiato con il Premio Mind and Brain per i suoi studi innovativi sulla mente e la coscienza.

Oltre a questi premi, Searle è membro di diverse accademie internazionali, inclusa l'American Academy of Arts and Sciences, e ha ricevuto numerose lauree honoris causa da università di tutto il mondo.

Conclusione

In sintesi, mentre Turing e Searle condividono un interesse comune per la natura dell'intelligenza e della mente, le loro posizioni sono radicalmente diverse. Turing vede l'intelligenza come qualcosa che può essere simulato attraverso il comportamento, mentre Searle insiste che senza comprensione interna, ciò che viene prodotto è una mera simulazione priva di coscienza. Questo dibattito continua a influenzare il campo dell'intelligenza artificiale, sfidando filosofi, scienziati e ingegneri a esplorare cosa significhi davvero essere intelligenti e consapevoli. Si noti che il test di Touring è anche all'origine dell'onnipresente "CAPTCHA" (Completely Automated Public Turing test to tell Computers and Humans Apart) che è una forma di test di Turing inverso utilizzato per distinguere gli esseri umani dai bot nei contesti online.

1 Elementi di Python

Python è un linguaggio di programmazione versatile e potente, conosciuto per la sua sintassi chiara e leggibile. Creato da Guido van Rossum e rilasciato per la prima volta nel 1991, Python è diventato uno dei linguaggi più popolari al mondo grazie alla sua facilità d'uso e alla vasta comunità di sviluppatori. Uno degli aspetti più apprezzati di Python è la sua capacità di permettere ai programmatori di esprimere concetti in meno linee di codice rispetto a molti altri linguaggi, rendendolo ideale sia per i principianti che per gli esperti.

Python è oggi il linguaggio di riferimento nell'intelligenza artificiale e nell'apprendimento automatico grazie a librerie potenti come `TensorFlow`, `Keras`, `PyTorch` e `Scikit-learn`, che facilitano lo sviluppo di modelli di machine learning e deep learning.

In questo capitolo, esploreremo i fondamenti di Python, inclusa la sua sintassi di base, le strutture dati essenziali e le funzionalità principali.

1.1 Sintassi e Concetti di Base

La sintassi di Python è progettata per essere intuitiva e di facile lettura. Un programma Python è costituito da linee di codice che vengono eseguite sequenzialmente. Ogni linea di codice esprime una singola operazione o istruzione. Esploriamo alcuni concetti di base attraverso esempi pratici nel contesto della giurisprudenza:

1.1.1 Assegnazione di Variabili**

In Python, l'assegnazione di un valore a una variabile è semplice e diretta:

** Assegnazione di variabili **

```
fascicolo = "2024/12345"
nomeGiudice = "Giudice Rossi"
casoChiuso = False
```

In questo esempio, abbiamo assegnato una stringa (`"2024/12345"`) a `fascicolo`, una stringa (`"Giudice Rossi"`) a `nomeGiudice`, e un valore booleano (`False`) a `casoChiuso`, indicando che il caso non è ancora chiuso.

1.1.2 Operazioni Aritmetiche

Python supporta tutte le principali operazioni aritmetiche; possiamo applicarle anche a contesti legali, come il calcolo delle pene o delle ammende.

Calcolo di una sanzione basata sui giorni di ritardo

```
giorniRitardo = 10
sanzioneGiornaliera = 50

sanzioneTotale = giorniRitardo * sanzioneGiornaliera  # Sanzione totale: 500€
print(f"La sanzione totale è di {sanzioneTotale} euro.")
```

Output:

```
La sanzione totale è di 500 euro.
```

1.1.3 Controllo del Flusso (Condizioni e Cicli)

1.1.3.1 Lo Statement `if`

Lo statement `if` in Python è una struttura di controllo che consente di eseguire blocchi di codice condizionatamente, in base al risultato di una condizione booleana (vero o falso). La sintassi di base è la seguente:

```
if condizione:
    # codice da eseguire se la condizione è vera
```

Componenti dello statement `if`

1. **Condizione**: È un'espressione che restituisce un valore booleano. Se la condizione è vera (`True`), il blocco di codice indentato successivo verrà eseguito.
2. **Blocco di Codice**: Il codice all'interno del blocco deve essere indentato. L'indentazione è fondamentale in Python e indica quali istruzioni appartengono al blocco `if`.

Esempio di Base

```
x = 10

if x > 5:
    print("x è maggiore di 5")
```

Output:

```
x è maggiore di 5
```

In questo esempio, poiché `x` è effettivamente maggiore di 5, verrà stampato "x è maggiore di 5".

Varianti dello Statement `if`

1. **`if-else`**: Consente di eseguire un blocco di codice alternativo se la condizione è falsa.

   ```
   if x > 5:
       print("x è maggiore di 5")
   else:
       print("x non è maggiore di 5")
   ```

   Output: `x è maggiore di 5`

2. **`if-elif-else`**: Permette di controllare più condizioni in sequenza.

   ```
   if x > 10:
       print("x è maggiore di 10")
   elif x > 5:
       print("x è maggiore di 5 ma minore o uguale a 10")
   else:
       print("x è minore o uguale a 5")
   ```

   Output: `x è maggiore di 5 ma minore o uguale a 10`

Operatori di Confronto

Puoi usare vari operatori di confronto all'interno delle condizioni, tra cui:

- `==`: uguale
- `!=`: diverso
- `>`: maggiore
- `<`: minore
- `>=`: maggiore o uguale
- `<=`: minore o uguale

Esempi di Condizioni Complesse

Puoi anche combinare condizioni usando gli operatori logici:

- `and`: restituisce `True` se entrambe le condizioni sono vere.
- `or`: restituisce `True` se almeno una delle condizioni è vera.
- `not`: inverte il valore della condizione.

```python
if x > 5 and x < 15:
    print("x è compreso tra 5 e 15")
```

Output:

```
x è compreso tra 5 e 15
```

In sintesi, lo statement `if` è una delle basi della logica di programmazione in Python, permettendo di prendere decisioni e controllare il flusso del programma. Python utilizza l'indentazione per definire blocchi di codice, rendendo il codice pulito e leggibile.

1.1.3.2 I cicli

for

```python
# Iterare su una lista di testimoni
witnesses = ["Testimone A", "Testimone B", "Testimone C"]

for witness in witnesses:
    print(f"Interrogare {witness}")
```

Output:

```
Interrogare Testimone A
Interrogare Testimone B
Interrogare Testimone C
```

while

```python
# Ciclo while per contare all'indietro da 5 a 1
n = 5
while n > 0:
    print(n)
    n -= 1
```

Output:

```
5
4
3
2
1
```

1.1.4 Gestione delle Eccezioni

Python permette di gestire gli errori e le eccezioni in modo elegante usando i blocchi `try` e `except`.

```
try:
    # Tentativo di apertura di un file di sentenze
    with open("sentenze.txt", "r") as file:
        content = file.read()
except FileNotFoundError:
    print("Errore: Il file delle sentenze non è stato trovato.")
```

Output:

```
Errore: Il file delle sentenze non è stato trovato.
```

l' istruzione try ammette anche una estensione chiamata finally che viene eseguita alla fine della try per garantire la consistenza delle operazioni svolte nel blocco try. Ad esempio, il precedente blocco try potrebbe avere una finally per chiudere il file alla fine del blocco:

```
try:
    f = open('data.txt', 'r')
    contenuto = f.read()
except FileNotFoundError:
    print("Errore: Il file delle sentenze non è stato trovato.")
finally:
    f.close()  # Il blocco finally garantisce che il file venga chiuso
```

Output:

```
Errore: Il file delle sentenze non è stato trovato.
```

1.1.5 Stampe a Video

La funzione `print` in Python è utilizzata per stampare messaggi a video, rendendo facile il debug e la visualizzazione dei risultati.

```python
# Stampa di informazioni su un caso legale
fascicolo = "2024/12345"
nomeGiudice = "Mario Rossi"
casoChiuso = False

print(f"Numero del caso: {fascicolo}")
print(f"Nome del giudice: {nomeGiudice}")
print(f"Il caso è chiuso? {'Sì' if casoChiuso else 'No'}")
```

Output:

```
Numero del caso: 2024/12345
Nome del giudice: Mario Rossi
Il caso è chiuso? No
```

1.1.6 Lettura e Scrittura di File

Python fornisce metodi semplici per leggere e scrivere file, essenziali per la gestione dei dati legali.

Scrittura di un file testuale

```python
# Scrittura di una sentenza in un file
sentenza = "Il caso è chiuso. La sentenza è di 5 anni di reclusione."
with open("sentenza.txt", "w") as file:
    file.write(sentenza)
```

Lettura di un file

```python
# Lettura di una sentenza da un file
try:
    with open("sentenza.txt", "r") as file:
        content = file.read()
        print("Contenuto del file:")
        print(content)
except FileNotFoundError:
    print("Errore: Il file delle sentenze non è stato trovato.")
```

Output:

```
Contenuto del file:
Il caso è chiuso. La sentenza è di 5 anni di reclusione.
```

In questo paragrafo, abbiamo esplorato alcuni dei concetti di base della sintassi di Python, inclusi l'assegnazione di variabili, le operazioni aritmetiche, le condizioni, i cicli, le funzioni, la gestione delle eccezioni, le stampe a video e la lettura e scrittura di file, applicandoli a contesti giuridici. Questi concetti costituiscono la base della programmazione in Python e ti preparano per affrontare problemi più complessi nei capitoli successivi.

1.2 Strutture Dati

Le strutture dati sono elementi fondamentali per qualsiasi linguaggio di programmazione, e Python offre una serie di strutture dati integrate che rendono la gestione dei dati semplice ed efficiente. Esploriamo alcune delle strutture dati principali e le loro funzionalità attraverso esempi pratici nel contesto della giurisprudenza.

1.2.1 Liste

Le liste sono collezioni ordinate di elementi che possono essere modificati. Possono contenere elementi di qualsiasi tipo, inclusi numeri, stringhe e altre liste.

```python
# Creare una lista di leggi applicabili
leggi = ["Art. 1 - Reati contro la persona", "Art. 2 - Reati contro il patrimonio"]

# Aggiungere una nuova legge
leggi.append("Art. 3 - Reati contro l'ambiente")

# Rimuovere una legge
leggi.remove("Art. 2 - Reati contro il patrimonio")

# Accedere a una legge
print(leggi[1])  # Stampa: "Art. 3 - Reati contro l'ambiente"
print("..............................")
# Iterare su una lista di leggi
for legge in leggi:
    print(legge)
```

Output:

```
Art. 3 - Reati contro l'ambiente
.............................
Art. 1 - Reati contro la persona
Art. 3 - Reati contro l'ambiente
```

1.2.2 Dizionari

I dizionari sono collezioni di coppie chiave-valore che permettono un accesso rapido ai dati. Le chiavi devono essere uniche e immutabili.

```python
# Creare un dizionario per un caso legale
fascicolo = {"numero": "2024/12345", "giudice": "Giudice Rossi", "stato": "aperto"}

# Aggiungere o aggiornare un elemento
fascicolo["stato"] = "chiuso"

# Rimuovere un elemento
del fascicolo["giudice"]

# Accedere a un valore
print(fascicolo["numero"])  # Stampa: "2024/12345"

# Iterare su un dizionario
for chiave, valore in fascicolo.items():
    print(f"{chiave}: {valore}")
```

Output:

```
2024/12345
numero: 2024/12345
stato: chiuso
```

1.2.3 Set o insiemi

I set sono collezioni di elementi unici, utili per operazioni matematiche insiemistiche come l'unione e l'intersezione.

```
# Creare un set di articoli violati
articoliViolati = {"Art. 1", "Art. 3"}

# Aggiungere un articolo violato
articoliViolati.add("Art. 5")

# Rimuovere un articolo violato
articoliViolati.remove("Art. 1")

# Operazioni sui set
articoliAggiuntivi = {"Art. 2", "Art. 4", "Art. 5"}
risultatoUnione = articoliViolati.union(articoliAggiuntivi)  # Unione
risultatoIntersezione = articoliViolati.intersection(articoliAggiuntivi)  # Intersezione

print(risultatoUnione)  # Stampa: {'Art. 2', 'Art. 3', 'Art. 4', 'Art. 5'}
print(risultatoIntersezione)  # Stampa: {'Art. 5'}
```

Output:

```
{'Art. 4', 'Art. 3', 'Art. 5', 'Art. 2'}
{'Art. 5'}
```

1.2.4 Set o insiemi

I set sono collezioni di elementi unici, utili per operazioni matematiche insiemistiche come l'unione e l'intersezione.

```
# Creare un set di articoli violati
articoliViolati = {"Art. 1", "Art. 3"}

# Aggiungere un articolo violato
articoliViolati.add("Art. 5")

# Rimuovere un articolo violato
articoliViolati.remove("Art. 1")

# Operazioni sui set
articoliAggiuntivi = {"Art. 2", "Art. 4", "Art. 5"}
risultatoUnione = articoliViolati.union(articoliAggiuntivi)  # Unione
risultatoIntersezione = articoliViolati.intersection(articoliAggiuntivi)  # Intersezione
```

```
print(risultatoUnione)  # Stampa: {'Art. 2', 'Art. 3', 'Art. 4', 'Art. 5'}
print(risultatoIntersezione)  # Stampa: {'Art. 5'}
```

Output:

```
{'Art. 4', 'Art. 3', 'Art. 5', 'Art. 2'}
{'Art. 5'}
```

1.2.5 Tuple

Le tuple sono simili alle liste ma sono immutabili. Una volta create, non possono essere modificate. L'immutabilità delle tuple offre diversi vantaggi, tra cui la sicurezza dei dati e l'ottimizzazione delle prestazioni. Quando un dato non deve essere modificato, l'utilizzo delle tuple assicura che il dato rimanga costante durante l'esecuzione del programma, riducendo il rischio di errori accidentali. Inoltre, le tuple possono essere utilizzate come chiavi nei dizionari, poiché sono immutabili.

```
# Creare una tupla per le coordinate di un luogo del crimine
coordinateScenaCrimine = (45.4642, 9.1900)

# Accedere a un elemento
print(coordinateScenaCrimine[0])  # Stampa: 45.4642
```

Output:

```
45.4642
```

```
# Le tuple non possono essere modificate
coordinateScenaCrimine[0] = 45.5000  # Questo genererà un errore
```

Output:

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-6-b4ce3edf64a6> in <cell line: 2>()
      1 # Le tuple non possono essere modificate
----> 2 coordinateScenaCrimine[0] = 45.5000  # Questo genererà un errore

TypeError: 'tuple' object does not support item assignment
```

1.3 Funzioni e Moduli

1.3.1 Definire e usare le funzioni

Le funzioni permettono di organizzare il codice in blocchi riutilizzabili, mentre i moduli permettono di suddividere il codice in file separati che possono essere importati e utilizzati in altri programmi. Sia le funzioni che i moduli consentono di organizzare il codice sviluppato in blocchi omogeni facilitando il riuso e lo scambio di codice sorgente. Ad esempio, se abbiamo scritto un blocco di codice che riolve un problema applicativo come il calcolo degli anni di detenzione per un determinato reato possiamo inserire questo blocco di istruzioni python in una funzione dal nome calcoloDurataPena. Mentre, se abbiamo definito un certo numero di funzioni utili nell'ambito penalistico possiamo definire un modulo denominato calcolipenale.py e importarlo ogniqualvolta ne abbiamo bisogno

```
# Funzione per calcolare la durata di una pena basata sulla gravità del crimine
def calcoloDurataPena(crime_severity):
    if crime_severity == "grave":
        return 10
    elif crime_severity == "moderato":
        return 5
    else:
        return 1

anni = calcoloDurataPena("grave")
print(f"La durata della pena è di {anni} anni.")
```

Output:

```
La durata della pena è di 10 anni.
```

1.3.2 Creare e importare moduli personalizzati

È possibile creare moduli personalizzati per organizzare meglio il codice. Ad esempio, se si desidera separare le funzioni di calcolo in ambito penalistico in un modulo separato:

Creare un modulo (calcolipenale.py)

```
# calcolipenale.py
def calcoloDurataPena(crime_severity):
    if crime_severity == "grave":
        return 10
```

```
    elif crime_severity == "moderato":
        return 5
    else:
        return 1

def calcolaSanzione(sanzioneBase, giorniRitardo, mora=50):
    return sanzioneBase + giorniRitardo * mora
```

Importare e usare il modulo python

```
# script_principale.py
import calcolipenale

anni = calcolipenale.calcoloDurataPena("grave")
sanzione = calcolipenale.calcolaSanzione(550,10,10)

print(f"La durata della pena è di 10 anni.")
print(f"La multa totale è di 650 euro.")
```

Output:

```
La durata della pena è di {anni} anni.
La multa totale è di {sanzione} euro.
```

1.3.3 Importare moduli

La possibilità di creare e importare moduli di codice è una caratteristica di molti linguaggi di programmazione che consente:

1. **Riutilizzo del Codice**: L'importazione dei moduli consente di riutilizzare il codice esistente, evitando la duplicazione e rendendo più facile la manutenzione del software.
2. **Organizzazione**: I moduli permettono di organizzare il codice in file separati, facilitando la comprensione e la navigazione nel progetto.
3. **Astrazione**: Utilizzando moduli, si possono nascondere dettagli complessi dietro interfacce più semplici, migliorando la leggibilità e la chiarezza del codice.
4. **Ecosistema Ricco**: Python ha un vasto ecosistema di moduli e librerie che coprono un'ampia gamma di funzionalità, dal trattamento dei dati al web development, permettendo agli sviluppatori di concentrarsi su ciò che è importante per il loro progetto.

In particolare, in Python in ambito IA i moduli consentono di:

1. **Librerie Specializzate**: l'importazione di moduli come NumPy, Pandas, TensorFlow e PyTorch è essenziale per operazioni matematiche complesse, manipolazione dei dati e costruzione di modelli di machine learning.
2. **Facilità di Sperimentazione**: Le librerie consentono di sperimentare rapidamente con algoritmi di IA senza dover scrivere tutto da zero, accelerando il processo di sviluppo e ricerca.
3. **Supporto per il Calcolo Distribuito**: Alcuni moduli permettono di sfruttare il calcolo distribuito, fondamentale per addestrare modelli di IA su grandi quantità di dati.
4. **Standardizzazione**: Utilizzando librerie consolidate, i ricercatori e gli sviluppatori possono garantire che le loro implementazioni siano compatibili con gli standard del settore, facilitando la condivisione e la riproducibilità dei risultati.

In sintesi, l'importazione di moduli in Python è cruciale per lo sviluppo efficace e efficiente, specialmente nell'ambito dell'intelligenza artificiale, dove la complessità e la varietà degli strumenti richiesti sono elevate.

```
# esempio di importazione di una libreria standard
import math

# Calcolare la distanza tra due punti (coordinate di due luoghi del crimine)
def calculate_distance(coord1, coord2):
    return math.sqrt((coord2[0] - coord1[0])**2 + (coord2[1] - coord1[1])**2)

distance = calculate_distance((45.4642, 9.1900), (45.5000, 9.2100))
print(f"La distanza tra i due luoghi del crimine è di {distance} unità.")
```

Output:

```
La distanza tra i due luoghi del crimine è di 0.04100780413531289 unità.
```

1.4 Gli oggetti

In Python, gli oggetti sono entità fondamentali che rappresentano sia i dati che le funzioni. Ogni cosa in Python è un oggetto, inclusi numeri, stringhe, liste, funzioni e persino le stesse classi. Gli oggetti hanno proprietà, chiamate attributi, e comportamenti, definiti dai metodi. Gli oggetti possono essere utilizzati per modellare situazioni reali, come il trattamento di fascicoli giuridici, dove un oggetto può rappresentare un singolo caso legale, contenente informazioni sul giudice, le parti coinvolte e lo stato del procedimento. La programmazione orientata agli

oggetti (OOP) consente di organizzare il codice in modo più modulare e riutilizzabile, facilitando la gestione di progetti complessi e favorendo la collaborazione tra sviluppatori. Nei prossimi esempi, vedremo come creare e gestire oggetti in Python, e come questi possano essere applicati a contesti giuridici concreti.

1.4.1 Creazione di una classe

Una classe è un modello per creare oggetti. Può essere paragonata a una struttura o un modello da cui vengono creati singoli oggetti.

```
class Fascicolo:
    def __init__(self, numero, giudice, chiuso):
        self.numero = numero          # Attributo che contiene il numero del fascicolo
        self.giudice = giudice        # Attributo che contiene il nome del giudice
        self.chiuso = chiuso          # Attributo che indica se il fascicolo è chiuso (True
```

In questo esempio, la classe `Fascicolo` ha tre attributi: `numero`, `giudice` e `chiuso`. Questi vengono definiti quando viene creato un nuovo oggetto della classe.

1.4.2 Creazione di un oggetto

Per creare un oggetto da una classe, si usa la classe come una funzione. Gli oggetti creati da una classe sono chiamati istanze della classe.

```
# Creiamo un nuovo fascicolo
fascicolo_1 = Fascicolo("2024/12345", "Giudice Rossi", False)

# Accesso agli attributi dell'oggetto
print(f"Numero del fascicolo: {fascicolo_1.numero}")
print(f"Giudice: {fascicolo_1.giudice}")
print(f"Il fascicolo è chiuso? {fascicolo_1.chiuso}")
```

Output:

```
Numero del fascicolo: 2024/12345
Giudice: Giudice Rossi
Il fascicolo è chiuso? False
```

Qui abbiamo creato un oggetto `fascicolo_1` con i valori specificati. Abbiamo poi stampato i valori dei suoi attributi.

1.4.3 Aggiunta di un metodo alla classe

I metodi sono funzioni definite all'interno di una classe e possono operare sugli attributi dell'oggetto.

```
class Fascicolo:
    def __init__(self, numero, giudice, chiuso):
        self.numero = numero
        self.giudice = giudice
        self.chiuso = chiuso

    # Metodo per chiudere il fascicolo
    def chiudi_fascicolo(self):
        self.chiuso = True
        print(f"Il fascicolo {self.numero} è stato chiuso.")

# Creiamo un altro fascicolo
fascicolo_2 = Fascicolo("2024/67890", "Giudice Bianchi", False)

# Chiudiamo il fascicolo
fascicolo_2.chiudi_fascicolo()
```

Output:

```
Il fascicolo 2024/67890 è stato chiuso.
```

In questo esempio, abbiamo aggiunto un metodo chiamato `chiudi_fascicolo` che modifica l'attributo `chiuso` di un fascicolo e stampa un messaggio.

1.4.4 Utilizzo di più oggetti

Possiamo creare più istanze della classe `Fascicolo` per rappresentare diversi fascicoli.

```
fascicolo_3 = Fascicolo("2024/54321", "Giudice Verdi", False)
fascicolo_4 = Fascicolo("2024/98765", "Giudice Neri", True)

print(f"Fascicolo 3 - Giudice: {fascicolo_3.giudice}, Chiuso: {fascicolo_3.chiuso}")
print(f"Fascicolo 4 - Giudice: {fascicolo_4.giudice}, Chiuso: {fascicolo_4.chiuso}")
```

Output:

```
Fascicolo 3 - Giudice: Giudice Verdi, Chiuso: False
Fascicolo 4 - Giudice: Giudice Neri, Chiuso: True
```

1.4.5 Ereditarietà tra classi

L'ereditarietà consente di creare una nuova classe che eredita attributi e metodi da un'altra classe. Vediamo come creare una classe `FascicoloPenale` che eredita dalla classe `Fascicolo`.

```
class FascicoloPenale(Fascicolo):
    def __init__(self, numero, giudice, chiuso, reato):
        super().__init__(numero, giudice, chiuso)  # Chiamata al costruttore della classe ge
        self.reato = reato  # Attributo specifico del fascicolo penale

# Creiamo un fascicolo penale
fascicolo_penale = FascicoloPenale("2024/11111", "Giudice Gialli", False, "Frode")

print(f"Fascicolo Penale - Numero: {fascicolo_penale.numero}, Reato: {fascicolo_penale.reato
```

Output:

```
Fascicolo Penale - Numero: 2024/11111, Reato: Frode
```

In questo caso, `FascicoloPenale` eredita attributi e metodi dalla classe `Fascicolo` e aggiunge un nuovo attributo chiamato `reato`.

Gli oggetti in Python offrono una grande flessibilità nel modellare e organizzare il codice. Con la programmazione orientata agli oggetti, puoi creare classi che rappresentano concetti del mondo reale, semplificando la gestione di dati complessi. Con l'uso di attributi, metodi e ereditarietà, diventa possibile creare programmi modulari e riutilizzabili per affrontare problemi sempre più complessi.

In questo capitolo, abbiamo esplorato le principali strutture dati di Python, tra cui liste, dizionari, set e tuple, e le funzionalità principali come le funzioni e i moduli, applicandole a contesti giuridici. Queste strutture dati e funzionalità sono fondamentali per scrivere programmi efficienti e ben organizzati e saranno essenziali per affrontare i compiti di intelligenza artificiale nei capitoli successivi.

2 Algoritmi

Gli algoritmi sono il cuore pulsante dell'intelligenza artificiale e svolgono un ruolo cruciale nel trasformare i dati grezzi in informazioni utili. In questo capitolo, esploreremo vari tipi di algoritmi utilizzati nell'IA. Approfondiremo l'inferenza logica, probabilistica e bayesiana, nonché gli algoritmi di ricerca, equitativi e predittivi.

2.1 Inferenza Logica

> **definizione Treccani**: inferenza logica sinonimo di «argomentazione logica» utilizzato per designare il processo di deduzione di una formula A, detta conclusione, a partire da una o più formule, dette premesse. Secondo A. De Morgan, una inferenza è la «produzione di una proposizione come conseguenza necessaria di una o più proposizioni».

L'inferenza logica è un processo fondamentale nel campo della logica, della matematica e della filosofia, utilizzato per derivare conclusioni a partire da premesse o informazioni date. Questo processo può essere visto come un mezzo per scoprire nuove verità o per confermare la validità di affermazioni esistenti. L'inferenza logica si suddivide principalmente in due categorie: deduttiva e induttiva.

L'inferenza deduttiva è quella in cui la conclusione deriva necessariamente dalle premesse; se le premesse sono vere, la conclusione non può che essere vera. Un classico esempio di inferenza deduttiva è il sillogismo: "Tutti gli uomini sono mortali; Socrate è un uomo; quindi, Socrate è mortale." In questo caso, la verità delle premesse garantisce la verità della conclusione.

L'inferenza induttiva, invece, opera diversamente: partendo da osservazioni specifiche o da una serie di dati, arriva a conclusioni più generali, che non sono necessariamente certe ma probabili. Ad esempio, se si osserva che il sole è sorto ogni giorno, si potrebbe inferire che il sole sorgerà anche domani. Questa forma di inferenza è molto utilizzata nella scienza, dove gli scienziati formulano ipotesi basate su dati osservati e sperimentali.

Un altro tipo di inferenza logica è l'abduzione, che implica la formazione della migliore spiegazione possibile data un insieme di osservazioni. Questo tipo di inferenza è spesso utilizzato nella diagnosi medica, nella ricerca scientifica e nelle indagini criminali, dove si cerca di spiegare i dati osservati nel modo più coerente possibile.

L'inferenza logica è strettamente legata al concetto di validità e di correttezza degli argomenti. Un'argomentazione è valida se la sua struttura logica è tale che, qualora le premesse siano vere, anche la conclusione deve essere vera. Tuttavia, un'argomentazione può essere valida senza essere corretta; per essere corretta, deve avere anche premesse vere. Ad esempio, l'argomentazione "Tutti gli unicorni sono verdi; io possiedo un unicorno; quindi, il mio unicorno è verde" è valida dal punto di vista logico, ma non è corretta perché le premesse non sono vere.

L'inferenza logica è alla base di molti sistemi di intelligenza artificiale e di calcolo automatico, dove gli algoritmi vengono progettati per inferire nuove informazioni a partire da dati iniziali. Nei sistemi esperti, per esempio, vengono utilizzate regole di inferenza per simulare il processo decisionale umano. In conclusione, l'inferenza logica è uno strumento potente e versatile che permea molte aree del pensiero umano e della tecnologia, consentendo di avanzare nella conoscenza e nella comprensione del mondo che ci circonda. L'inferenza logica è una tecnica fondamentale dell'intelligenza artificiale che utilizza le regole logiche per derivare nuove informazioni da quelle esistenti. Nella giurisprudenza, l'inferenza logica può essere utilizzata per analizzare le leggi e determinare le conseguenze logiche delle azioni legali.

> **I sistemi esperti** - Negli anni '80, l'inferenza logica è stata fondamentale nello sviluppo dei sistemi esperti, strumenti avanzati di intelligenza artificiale progettati per risolvere problemi complessi emulando il ragionamento umano. Due noti prodotti commerciali di quel periodo sono stati MYCIN, un sistema esperto per la diagnosi di infezioni del sangue, e XCON, utilizzato per configurare sistemi di computer VAX di Digital Equipment Corporation. MYCIN e XCON sfruttavano regole di inferenza per elaborare informazioni e fornire raccomandazioni o soluzioni, dimostrando l'efficacia dell'inferenza logica in applicazioni pratiche e commerciali > - "Rule-based Expert Systems : The MYCIN Experiments of the Stanford Heuristic Programming Project", edited by Bruce G. Buchanan, Edward H. Shortliffe (AddisonWesley, 1984) - "RI: an Expert in the Computer Systems Domain"

2.1.1 Proposizioni Logiche

Le proposizioni logiche sono dichiarazioni atomiche che possono essere valutate come vere o false. Le proposizioni possono essere combinate utilizzando operatori logici come AND, OR, NOT, IMPLIES, che permettono di costruire regole complesse rappresentate da formule logiche.

Ecco alcuni esempi di proposizioni logiche:

p: "Il sole è luminoso" (Vero) q: "La Luna è fatta di formaggio" (Falso) r: "Se piove, allora la strada sarà bagnata" (Condizionale)

2.1.2 Calcolo delle Proposizioni Logiche

Le proposizioni logiche possono essere manipolate utilizzando vari operatori logici che eseguono operazioni specifiche:

Congiunzione (AND -): L'operatore AND restituisce vero solo quando entrambe le proposizioni coinvolte sono vere. Ad esempio, se abbiamo due proposizioni p e q, p q è vero solo se entrambe p e q sono vere. La cosidetta tabella di verità riportata qui sotto consente di vedere come funziona l'operatore AD.

Table 2.1: Tavola della verità per la congiunzione

p	q	p q
False	False	False
False	True	False
True	False	False
True	True	True

Disgiunzione (OR -): L'operatore OR restituisce vero se almeno una delle due proposizioni coinvolte è vera. Ad esempio, p q è vero se p è vero oppure se q è vero oppure se entrambi sono veri. La tabella di verità riportata qui sotto consente di vedere come funziona l'operatore OR.

Table 2.2: Tavola della verità per la disgiunzione

p	q	p q
False	False	False
False	True	True
True	False	True
True	True	True

Negazione (NOT - ¬): L'operatore NOT cambia il valore di verità di una proposizione. Ad esempio, ¬p è vero se p è falso e viceversa.

Table 2.3: Tavola della verità per la negazione

p	¬p
False	True
True	False

Implicazione (→): L'implicazione è un'operazione logica che collega due proposizioni e stabilisce una relazione di condizionalità. Si rappresenta con il simbolo "→" e si legge come "se... allora". In un'implicazione del tipo "p → q", la proposizione p è chiamata l'antecedente e la proposizione q è il conseguente. L'implicazione è falsa solo nel caso in cui l'antecedente è vero e il conseguente è falso. In tutti gli altri casi, l'implicazione è considerata vera. Poiché questa operazione è alla base di molti algoritmi di inferenza, è importante capire come funziona. La tabella di verità riportata qui sotto consente di vedere come funziona l'operatore implicazione.

Table 2.4: Tavola della verità per l'implicazione

p	q	p → q
False	False	True
False	True	True
True	False	False
True	True	True

Esempio di Implicazione: supponiamo di avere le seguenti proposizioni: - p: Il sole splende. - q: Faccio una passeggiata. - L'implicazione che possiamo formulare è: "Se il sole splende, allora faccio una passeggiata", che si scrive come "p → q". Dalla tabella della verità, possiamo vedere che in tre dei quattro casi l'implicazione "p → q" è vera. L'unico caso in cui l'implicazione è falsa è quando il sole splende (p è vero) ma non faccio una passeggiata (q è falso).

Quindi, in base alla logica dell'implicazione, se il sole splende, sto effettivamente facendo una passeggiata o potrei anche non farla (ad eccezione del caso in cui il sole splenda e io non faccia una passeggiata, in cui l'implicazione è falsa).

Implicazione Bilaterale ( ): L'implicazione bilaterale è un'operazione logica che stabilisce che due proposizioni sono equivalenti, cioè che entrambe le proposizioni hanno lo stesso valore di verità. Si rappresenta con il simbolo " " e si legge come "se e solo se". L'implicazione bilaterale è vera solo quando le proposizioni hanno lo stesso valore di verità, sia entrambe vere che entrambe false.

Table 2.5: Tavola della verità per l'implicazione bilaterale

p	q	p q
False	False	True
False	True	False
True	False	False
True	True	True

L'implicazione bilaterale, anche conosciuta come "se e solo se", è un importante concetto logico che stabilisce che due proposizioni sono logicamente equivalenti, cioè entrambe sono vere o entrambe sono false contemporaneamente.

Esempio di Implicazione Bilaterale: supponiamo di avere le seguenti proposizioni:

- p: Oggi è venerdì.
- q: Domani è sabato.

L'implicazione bilaterale tra p e q può essere scritta come p q, che si legge come "Oggi è venerdì se e solo se domani è sabato".

Dalla tabella di verità, possiamo notare che l'implicazione bilaterale "Oggi è venerdì se e solo se domani è sabato" è vera solo nei casi in cui entrambe le proposizioni sono vere (primo e ultimo caso) o entrambe sono false. Se c'è una discrepanza nelle verità delle proposizioni, l'implicazione bilaterale diventa falsa (secondo e terzo caso).

Quindi, nel nostro esempio, l'affermazione "Oggi è venerdì se e solo se domani è sabato" è vera solo quando entrambe le proposizioni sono vere o entrambe sono false, evidenziando l'equivalenza logica tra le due proposizioni nel contesto dell'implicazione bilaterale.

2.1.3 Basi della Conoscenza

La base della conoscenza in un agente a inferenza logica è costituita da proposizioni logiche, che sono affermazioni dichiarative che possono essere vere o false. Le proposizioni possono essere atomiche o composte e sono spesso rappresentate utilizzando variabili proposizionali. Queste variabili assumono valori di verità (vero o falso) e vengono combinate tramite operatori logici per formare regole logiche complesse. La base della conoscenza in un sistema logico definisce le relazioni tra le proposizioni e fornisce le fondamenta per il ragionamento e l'inferenza. Un agente a inferenza logica usa la Base della Conoscenza per giungere a conclusioni circa il mondo che la circonda; Per fare ciò ha bisogno di regole di implicazione logica ( ): Se , ovvero se implica logicamente , in ogni mondo dove è vera allora è vera. È diversa dall'implicazione perché non è un connettivo logico ma una relazione che dice che se è vera allora è vera e basta!

2.1.4 Sistemi basati sulla conoscenza

I sistemi basati sulla conoscenza sono strumenti informatici progettati per emulare il processo decisionale umano attraverso l'utilizzo di una base di conoscenza strutturata. Questi sistemi raccolgono, organizzano e utilizzano informazioni specifiche di un dominio per risolvere problemi complessi che richiedono competenza specialistica. Una componente fondamentale è la **base di conoscenza**, che contiene fatti, regole ed euristiche rappresentative del sapere umano in un determinato campo. Il **motore di inferenza** è l'altro elemento chiave: applica regole

logiche ai dati presenti nella base di conoscenza per dedurre nuove informazioni o prendere decisioni informate.

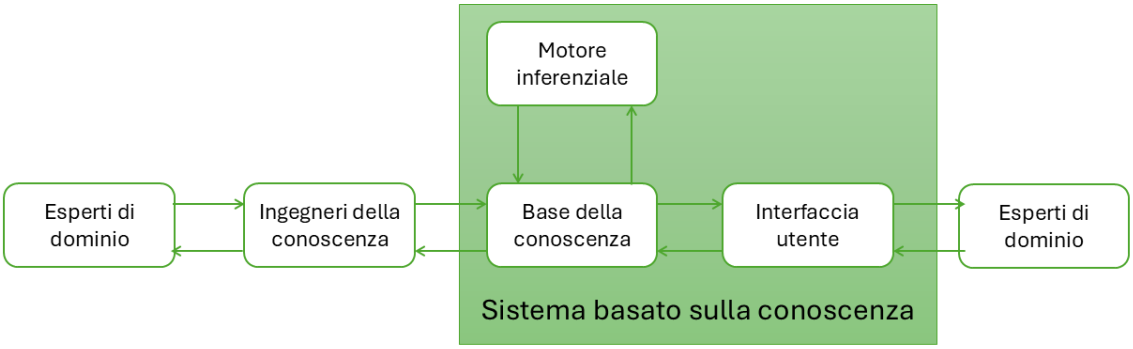

Figure 2.1: Processo di creazione e gestione di un sistema basato sulla conoscenza

Il processo di creazione di un sistema esperto basato sull'inferenza logica inizia con l'acquisizione della conoscenza, dove gli esperti del dominio collaborano per estrarre informazioni e regole rilevanti. Queste conoscenze vengono poi formalizzate nella rappresentazione della conoscenza, utilizzando strutture come regole if-then, ontologie o reti semantiche, che alimentano la base di conoscenza. Il motore di inferenza viene sviluppato per applicare queste regole logiche ai dati forniti, deducendo nuove informazioni o prendendo decisioni informate. La gestione del sistema include l'aggiornamento continuo della base di conoscenza per riflettere nuove scoperte o cambiamenti nel dominio, nonché la verifica e la validazione del sistema per garantirne l'accuratezza e l'affidabilità. Gli utenti interagiscono con il sistema attraverso un'interfaccia che facilita l'inserimento dei dati e la visualizzazione dei risultati, permettendo anche il feedback per miglioramenti futuri.

Questi sistemi trovano applicazione in vari settori, come la medicina, l'ingegneria, la finanza e l'assistenza clienti. Ad esempio, in ambito medico, un sistema basato sulla conoscenza può aiutare nella diagnosi di malattie analizzando sintomi e storie cliniche dei pazienti. L'efficacia di tali sistemi dipende dalla qualità e dall'aggiornamento costante della base di conoscenza, nonché dalla capacità del motore di inferenza di elaborare correttamente le informazioni.

Un vantaggio significativo dei sistemi basati sulla conoscenza è la possibilità di conservare e diffondere l'esperienza di esperti, rendendola accessibile a un pubblico più ampio e contribuendo alla standardizzazione delle pratiche. Tuttavia, la creazione e la manutenzione di una base di conoscenza richiedono notevoli risorse e competenze. Con l'avanzamento dell'intelligenza artificiale e dell'apprendimento automatico, questi sistemi continuano a evolversi, integrando nuove tecniche per migliorare l'efficienza, l'accuratezza e la capacità di apprendimento autonomo nelle loro applicazioni.

2.1.5 Semplice Sistema Esperto in ambito penale

In questo paragrafo, useremo la libreria **SymPy** in Python per creare un semplice sistema esperto basato sull'inferenza logica nell'ambito del diritto penale. Questo sistema aiuterà a determinare se determinati comportamenti costituiscono un reato, in base ai fatti noti e alle norme applicabili. Si noti che la libreria **SymPy** è stata sviluppata per consentire il calcolo simbolico in Python. In questo caso useremo le funzionalità di calcolo simbolico per la rappresentazione della conoscenza, usando le funzionalità di calcolo logico, e per l'inferenza logica.

2.1.5.1 Introduzione

Il diritto penale si basa su norme che definiscono quali comportamenti sono considerati reati e quali elementi devono essere presenti affinché un'azione sia punibile. Un sistema esperto in questo contesto può aiutare a:

- Valutare se un'azione specifica costituisce un reato.
- Identificare gli elementi costitutivi del reato.
- Fornire una base logica per decisioni legali.

Utilizzeremo SymPy per modellare proposizioni logiche, regole legali e per effettuare inferenze.

2.1.5.2 Installazione di SymPy

Assicurati di avere SymPy installato:

```
pip install sympy
```

Se stai utilizzando questo notebook in un ambiente in cui SymPy non è installato, esegui la seguente cella:

```
!pip install sympy
```

2.1.5.3 Concetti di Base nel Diritto Penale

Prima di iniziare, definiamo alcuni concetti chiave:

- **Fatti**: Eventi o azioni specifiche accadute.
- **Reati**: Comportamenti definiti come illeciti dalla legge penale.
- **Elementi Costitutivi del Reato**: Condizioni che devono essere soddisfatte perché un comportamento sia considerato un reato (ad esempio, azione, intenzione, nesso causale).
- **Regole Legali**: Norme che stabiliscono le condizioni in cui un comportamento è punibile.

2.1.5.4 Modellazione con SymPy

Passo 1: Importare i Moduli Necessari

Importiamo i moduli necessari da SymPy per lavorare con la logica proposizionale.

```
from sympy import symbols
from sympy.logic.boolalg import And, Or, Not, Implies, Equivalent
from sympy.logic.inference import satisfiable
```

Passo 2: Definire le Proposizioni Logiche

Definiamo le variabili che rappresentano i fatti e gli elementi costitutivi del reato.

```
# Fatti
Azione, Intenzione, NessoCausale = symbols('Azione Intenzione NessoCausale')

# Reato
Omicidio = symbols('Omicidio')
```

Passo 3: Definire le Regole che discendono dal Codice Penale

Ad esempio, secondo il codice penale, l'omicidio richiede:

- **Azione**: Causare la morte di una persona.
- **Intenzione**: Volontà di causare la morte (dolo).
- **Nesso Causale**: La morte è conseguenza dell'azione.

Definiamo la regola:

```
# Regola: Se c'è Azione, Intenzione e Nesso Causale, allora si configura l'Omicidio
regola_omicidio = Implies(And(Azione, Intenzione, NessoCausale), Omicidio)
```

Passo 4: Definire i Fatti Noti

Supponiamo di avere i seguenti fatti:

- Una persona ha compiuto un'azione che ha causato la morte di un'altra.
- Aveva l'intenzione di causare la morte.
- Esiste un nesso causale tra l'azione e la morte.

```
# Fatti noti
fatto1 = Azione  # L'azione di causare la morte
fatto2 = Intenzione  # Intenzione di causare la morte
fatto3 = NessoCausale  # La morte è conseguenza dell'azione
```

Passo 5: Creare la Base di Conoscenza

Combiniamo fatti e regole:

```
# Base di conoscenza
base_conoscenza = And(fatto1, fatto2, fatto3, regola_omicidio)
```

Passo 6: Inferenza Logica

Verifichiamo se, sulla base dei fatti e delle regole, possiamo concludere che si tratta di omicidio.

```
# Verifichiamo se Omicidio è deducibile
ipotesi = And(base_conoscenza, Not(Omicidio))
risultato = satisfiable(ipotesi)

if not risultato:
    print("Si configura il reato di omicidio.")
else:
    print("Non possiamo concludere che si tratti di omicidio.")
```

```
Si configura il reato di omicidio.
```

Output atteso:

```
Si configura il reato di omicidio.
```

2.1.5.5 Espansione del Sistema

Caso con Mancanza di Intenzione

Supponiamo che l'intenzione non sia presente (ad esempio, si tratta di omicidio colposo).

```
# Fatti noti senza Intenzione
fatto1 = Azione
fatto2 = Not(Intenzione)  # Mancanza di intenzione
fatto3 = NessoCausale

# Base di conoscenza aggiornata
base_conoscenza = And(fatto1, fatto2, fatto3, regola_omicidio)
```

Inferenza per Omicidio

```
# Inferenza
ipotesi = And(base_conoscenza, Not(Omicidio))
risultato = satisfiable(ipotesi)

if not risultato:
    print("Si configura il reato di omicidio.")
else:
    print("Non possiamo concludere che si tratti di omicidio.")
```

```
Non possiamo concludere che si tratti di omicidio.
```

Output atteso:

```
Non possiamo concludere che si tratti di omicidio.
```

2.1.5.5.1 Aggiunta di Altre Regole

Aggiungiamo la regola per l'**omicidio colposo**:

```
# Definizione del reato di Omicidio Colposo
OmicidioColposo = symbols('OmicidioColposo')

# Regola per Omicidio Colposo: Azione e Nesso Causale senza Intenzione
regola_omicidio_colposo = Implies(And(Azione, Not(Intenzione), NessoCausale), OmicidioColpos

# Aggiorniamo la base di conoscenza
base_conoscenza = And(fatto1, fatto2, fatto3, regola_omicidio, regola_omicidio_colposo)
```

Inferenza per Omicidio Colposo

```
# Verifichiamo se si configura l'Omicidio Colposo
ipotesi = And(base_conoscenza, Not(OmicidioColposo))
risultato = satisfiable(ipotesi)

if not risultato:
    print("Si configura il reato di omicidio colposo.")
else:
    print("Non possiamo concludere che si tratti di omicidio colposo.")
```

```
Si configura il reato di omicidio colposo.
```

Output atteso:

```
Si configura il reato di omicidio colposo.
```

2.1.5.6 Conclusione

Abbiamo visto come utilizzare SymPy per modellare un semplice sistema esperto nel campo del diritto penale. Questo esempio illustra come le regole legali e i fatti possono essere formalizzati utilizzando la logica proposizionale, permettendo al sistema di effettuare inferenze logiche.

Ricorda che questo è un modello semplificato e che il diritto penale è complesso e richiede una comprensione approfondita per essere modellato accuratamente. Questo sistema può essere un punto di partenza per sviluppi più avanzati e per esplorare l'intersezione tra intelligenza artificiale e diritto.