

VOLUME EXTRA / EDIZIONE AMPLIATA

Dynamo AI

Illustrata

Dalla frase alla forma

IA E MCP DA ZERO

Crea il tuo server MCP
e collegalo ai tuoi grafi Dynamo



*Per chi usa già Dynamo
e vuole capire l'intelligenza artificiale,
un passo alla volta.*

PAULO GIAVONI

Settembre 2026

Il punto da cui partiamo

Sai già che cos'è un grafo Dynamo. Hai collegato nodi, modificato parametri e osservato un risultato. Questo libro parte da lì: la nuova materia è il collegamento fra **una conversazione con l'IA e un'azione precisa del tuo grafo**.

Non occorre aver creato un server, programmato in Python o configurato un collegamento MCP. Introduciamo queste competenze con esempi piccoli. Il primo strumento calcolerà un volume; il successivo produrrà un file Dynamo; infine scriverai un nuovo tool che crea un cubo con un solo parametro.

Il risultato finale. Scrivi una richiesta come “crea un volume largo 8, profondo 5 e alto 3”. Il tuo server riceve i tre valori, prepara un grafo, lo fa valutare a DynamoCLI e restituisce il percorso del file. Tu apri il grafo e confronti il solido con la richiesta.

REGOLA 0 Un passo, una prova

Continua quando sai riconoscere il risultato del passaggio precedente. Un tool che calcola 120 dimostra una cosa; un grafo che contiene il solido ne dimostra un'altra. Imparerai a verificare entrambe.

Gli schemi sono illustrazioni didattiche originali: non riproducono schermate di un'interfaccia specifica. Blu indica dati e geometria; rosso indica codice e azioni; giallo indica regole e collegamenti.

Che cosa contiene questa edizione

Il percorso principale usa Windows, Python e grafi Dynamo Core. L'host di esempio è un'applicazione locale compatibile con MCP; il capitolo di collegamento mostra una configurazione Codex. Per la geometria serve un'installazione Dynamo funzionante con le librerie necessarie e un DynamoCLI compatibile.

Il kit. Nel pacchetto digitale che accompagna questa edizione trovi la cartella `kit`: contiene tutti gli script citati, i due grafi e le prove. Copiala in una cartella di lavoro prima di iniziare. Non è necessario riscrivere pagine di codice dal PDF: i listati spiegano i file che hai a disposizione.

Per la futura edizione a stampa. Il kit va distribuito insieme al libro, tramite un allegato digitale o un indirizzo di download stabilito dall'editore. Questa edizione di lavoro non dichiara un link pubblico che non esiste.

Versioni del laboratorio. Gli esempi usano l'SDK Python MCP 2.2.0. Il pacchetto è fissato a questa versione per rendere ripetibili le lezioni. I risultati locali documentati riguardano Python 3.11.9 e DynamoCLI 2.19.3.6394; altre combinazioni richiedono la prova di compatibilità descritta nel libro.

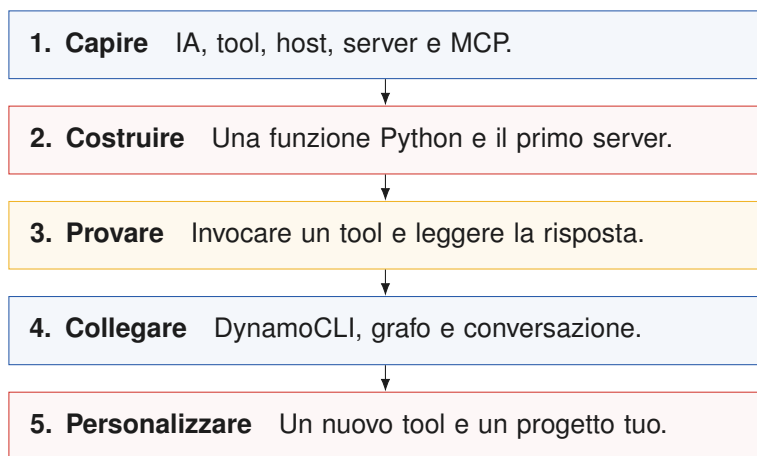
La geometria. Un file `.dyn` contiene la ricetta del grafo. Il nostro risultato non è un file CAD esportato né un elemento Revit. Nel progetto finale apri il grafo in Dynamo per osservare e modificare il solido.

Come copiare il codice. Usa i file del kit per i programmi completi. Le finestre grigie con comandi sono da eseguire in PowerShell; i blocchi JSON e TOML sono da salvare nei file indicati. L'etichetta del paragrafo chiarisce sempre la destinazione.

Indice

Il punto da cui partiamo	ii
La mappa del percorso	v
1 L'AI vista da chi usa Dynamo	1
2 Dalla conversazione a uno strumento	5
3 MCP senza misteri	9
4 Prepara il banco di lavoro	13
5 Il Python che serve per leggere il server	17
6 Il tuo primo server MCP	21
7 Provalo senza aprire una chat	25
8 Prepara il grafo per il dialogo	29
9 Dynamo senza aprire il canvas	32
10 Collega il server al tuo grafo	35
11 La prima conversazione con l'IA	39
12 Scrivi un tool che è tuo	42
13 Quando qualcosa non funziona	45
14 Progetta il tuo prossimo comando	48
Glossario da tenere accanto	53
Ripasso: sai spiegare il percorso?	55
Prove e fonti dell'edizione	56

La mappa del percorso



Tre modi di leggere. Alla prima lettura segui l'ordine. Durante un'esercitazione tieni aperti libro e kit. Quando una prova fallisce, vai al capitolo "Quando qualcosa non funziona" e riparti dall'ultimo risultato verificato.

Il tuo quaderno minimo. Per ogni prova conserva la richiesta, i valori inviati, il messaggio restituito e il file prodotto. Questa piccola abitudine rende comprensibili gli errori e confrontabili le varianti.

LEZIONE 1

L'AI vista da chi usa Dynamo

CAPIRE / DALLA RICHIESTA ALLA PROPOSTA

Sai già cosa succede quando colleghi uno slider a un nodo: il valore entra in una funzione e contribuisce al risultato. Con un assistente AI compare un passaggio diverso. Puoi scrivere *“vorrei un volume più slanciato”* e ricevere una proposta prima di aver indicato numeri precisi. Questa capacità di lavorare con il linguaggio rende interessante il collegamento con Dynamo. Per usarla bene, conviene capire che cosa produce una conversazione e che cosa richiede invece un'esecuzione.

In questo capitolo useremo sempre lo stesso piccolo progetto: un parallelepipedo largo 8, profondo 5 e alto 3 unità. Conosciamo la costruzione geometrica; possiamo quindi concentrare l'attenzione sul nuovo interlocutore. Alla fine saprai distinguere una spiegazione, una proposta di parametri e il risultato di uno strumento realmente eseguito.

Un modello che lavora con il linguaggio

Un modello linguistico è un sistema addestrato a riconoscere e utilizzare relazioni presenti nei dati, comprese quelle tra parole, concetti e istruzioni. Durante una conversazione elabora il contenuto disponibile e genera una risposta, un piccolo frammento di testo alla volta. Questi frammenti si chiamano **token**: possono corrispondere a una parola, a parte di una parola o a un segno.

Non serve conoscere i calcoli interni del modello per usarlo in questo laboratorio. Serve però distinguere la sua risposta dal risultato di un nodo. Se chiedi *“come posso ottenere una torre?”*, l'assistente può proporti proporzioni, descrivere nodi e spiegare una strategia. Una risposta ben scritta può contenere un nome di

nodo inesatto o trascurare una condizione del tuo grafo. La fluidità della spiegazione non certifica la correttezza della costruzione.

Il modello può anche risolvere problemi, confrontare alternative e ragionare sui passaggi. Rimane utile verificare ciò che conta con un riferimento esterno: il grafo aperto, i valori degli input, un calcolo o il risultato di una funzione. Nel nostro esempio il volume atteso è 120 unità cubiche. È un controllo concreto, facile da ripetere.

Prompt: la richiesta che dai all'assistente

Un **prompt** è il contenuto con cui chiedi all'assistente di svolgere un compito. Può essere una frase breve oppure una richiesta accompagnata da vincoli ed esempi. Non è necessario scrivere un testo complicato. È più utile rendere espliciti l'obiettivo e le condizioni che per te sono già ovvie.

Confronta queste due richieste. *“Fammi una bella torre”* lascia aperti forma, dimensioni, posizione e risultato da consegnare. *“Proponi tre combinazioni di larghezza, profondità e altezza per un parallelepipedo slanciato. Mantieni la base di 8 per 5 unità e varia soltanto l'altezza”* definisce un lavoro controllabile. Puoi valutare le proposte senza chiederti quale base abbia immaginato l'assistente.

01 Dichiarare il risultato

Scrivi se vuoi una spiegazione, una proposta oppure un'azione. *“Spiegami il grafo”* e *“crea un nuovo file Dynamo”* richiedono risultati diversi.

02 Indica ciò che deve restare fisso

Specifica i parametri già decisi. Per esempio: base di 8 per 5 unità, origine in $(0, 0, 0)$ e altezza come unica variabile.

03 Chiedi una risposta verificabile

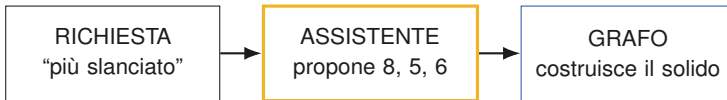
Per una proposta, chiedi i numeri e la motivazione. Per un'esecuzione, chiedi i valori utilizzati e il percorso del file prodotto.

Contesto: ciò che l'assistente può usare adesso

Il **contesto** è l'insieme delle informazioni rese disponibili al modello per la risposta corrente. Può includere la conversazione, istruzioni dell'applicazione, documenti forniti e risultati di strumenti. Non coincide con tutto ciò che esiste nel tuo computer. Avere un grafo aperto sul monitor non significa automaticamente che l'assistente ne conosca nodi e collegamenti.

Se scrivi *“aumenta quello di prima”*, l'assistente deve ricavare dal contesto quale oggetto e quale dimensione intendi. La frase può essere sufficiente in una conversazione breve, ma diventa fragile quando hai discusso più varianti. *“Crea una nuova variante del parallelepipedo 8 per 5 per 3, portando soltanto l'altezza a 6”* conserva il significato anche senza affidarsi a un riferimento vago.

Fornire informazioni nel contesto non equivale, di per sé, ad addestrare nuovamente il modello. Nel laboratorio aggiungiamo dati alla conversazione e rendiamo disponibili strumenti. Non stiamo costruendo un nuovo modello linguistico. Questa distinzione spiega perché un server MCP può essere utile senza richiedere un progetto di addestramento.



La proposta diventa geometria quando i valori raggiungono una costruzione eseguibile.

Schema 1. Linguaggio, parametri e costruzione hanno responsabilità diverse.

Risposta probabile ed esecuzione ripetibile

L'assistente genera una risposta in base al modello, al contesto e alle impostazioni di generazione. La stessa domanda può produrre formulazioni o proposte diverse. Questo è utile quando cerchi alternative; richiede attenzione quando desideri riprodurre esattamente un risultato.

Un grafo Dynamo fissato, eseguito con gli stessi input e nelle stesse condizioni, offre invece una procedura definita. Per il nostro parallelepipedo, dare 8, 5 e 3 ai tre controlli deve produrre le stesse dimensioni. In grafi reali contano anche versione, pacchetti, file esterni e altri dati: annotare l'ambiente aiuta a rendere la prova ripetibile.

REGOLA A Conserva il passaggio dai desideri ai numeri

L'assistente può aiutarti a scegliere una variante. Prima di eseguirla, rendi riconoscibili i parametri. Dopo l'esecuzione, confrontali con quelli del grafo prodotto.

FERMATI E RISPONDI L'assistente scrive: "Ho creato il volume 8 per 5 per 3". Nella conversazione non compare alcuna chiamata a uno strumento e non ricevi un file. Che cosa puoi concludere?

Risposta guidata. Hai una dichiarazione testuale, ma non ancora una prova dell'esecuzione. Chiedi il file e il risultato dello strumento, oppure costruisci il grafo seguendo la spiegazione. Considera concluso il lavoro quando puoi controllare il risultato richiesto.

RISULTATO ATTESO Sai formulare una richiesta precisa e riconoscere se la risposta contiene una proposta, una spiegazione o l'evidenza di un'azione eseguita.

LEZIONE 2

Dalla conversazione a uno strumento

USARE / UNA FRASE DIVENTA UNA CHIAMATA

Finora l'assistente ha lavorato con il linguaggio. Per creare un file o eseguire un grafo deve poter chiedere un'azione a un programma. Questa azione prende il nome di **tool**, cioè strumento. Nel laboratorio il tool principale si chiama `crea_geometria`: riceve tre dimensioni e avvia una procedura che prepara ed esegue un grafo Dynamo.

Per chi usa Dynamo, un paragone utile è una funzione con ingressi nominati. Conosci già il vantaggio di sapere che un ingresso accetta un punto e un altro un numero. Anche un tool dichiara quali dati richiede. Il paragone riguarda l'interfaccia; la chiamata dell'assistente non è un filo del canvas e non modifica da sola le connessioni del grafo.

Il nome dice che cosa fai, gli argomenti con che cosa

Gli **argomenti** sono i valori forniti a una chiamata. Se lo strumento accetta `larghezza`, `profondita` e `altezza`, una richiesta completa deve permettere di assegnare un numero a ogni nome. Il **tipo** descrive la categoria ammessa: qui numeri; in altri strumenti potrebbero essere testi, valori vero/falso o elenchi.

Una chiamata al nostro tool può contenere questi dati:

```
{"larghezza": 8, "profondita": 5, "altezza": 3}
```

Questo formato si chiama **JSON**. Le parole tra virgolette sono le chiavi; dopo i due punti trovi i valori. Le parentesi graffe raccolgono i dati in un oggetto. Il nome del tool viene trasmesso insieme alla chiamata, ma non è mostrato in questo piccolo esempio. Per ora basta leggere la corrispondenza tra ogni dimensione e il suo numero.

Un valore numerico è diverso dal testo "otto". Inoltre, il fatto che un dato sia un numero non lo rende automaticamente adatto alla geometria. Il server deve controllare anche le regole del progetto: nel kit le dimensioni consentite vanno da 0,1 a 100. Nel JSON un numero decimale usa il punto, per esempio 0.5.

Come l'assistente sceglie l'azione

L'applicazione rende disponibili al modello le descrizioni degli strumenti. Queste informazioni comprendono normalmente un nome, una spiegazione e lo schema degli argomenti. Quando la tua richiesta corrisponde a un'azione disponibile, l'assistente può proporre la chiamata appropriata. L'applicazione gestisce l'esecuzione e gli eventuali controlli previsti dalle sue impostazioni.

La descrizione del tool funziona come un'etichetta operativa. *"Crea un grafo con un parallelepipedo usando tre dimensioni positive"* indica uno scopo preciso. Una descrizione generica come *"fa cose con Dynamo"* rende difficile capire quando usarlo. Scrivere bene la descrizione è una parte della costruzione del tuo MCP, anche quando il codice della geometria è già pronto.

Il modello non può invocare qualsiasi funzione immaginabile. Può lavorare con gli strumenti che l'applicazione gli rende disponibili. Se il tuo server offre soltanto parallelepipedi, chiedere una sfera richiede un altro strumento o una modifica esplicita del progetto. La conversazione non aggiunge automaticamente questa capacità.



Schema 2. Una richiesta in lingua naturale viene tradotta in argomenti espliciti, poi eseguita da una funzione.

Dal risultato tecnico alla risposta leggibile

Dopo l'esecuzione, il tool restituisce un **risultato**. Può essere un numero, una struttura di dati, un percorso oppure un messaggio di errore. L'assistente lo usa per formulare la risposta. Sono due passaggi distinti: prima la funzione produce dati, poi la conversazione li presenta al lettore.

Nel nostro caso cercherai il percorso del nuovo `.dyn`, le dimensioni applicate e la descrizione della geometria ottenuta. Il server può anche restituire il volume teorico, calcolato moltiplicando gli input. Quel numero aiuta il controllo, ma la sua presenza, da sola, non dimostra che il grafo sia stato eseguito. Per questo apriremo il file e verificheremo il solido.

Se uno strumento restituisce un errore, la risposta corretta deve rendere chiaro che cosa è fallito. Un file preparato prima di un errore della CLI non è ancora una geometria verificata. Imparerai a distinguere la generazione della copia del grafo dalla sua esecuzione, così saprai quale passaggio ripetere.

REGOLA B Controlla tre elementi

Confronta la richiesta iniziale, gli argomenti inviati e il risultato restituito. Il collegamento funziona quando raccontano la stessa operazione e il file risultante supera la verifica nel Dynamo.

Una richiesta completa evita congetture

“Rendilo più grande” può significare aumentare tutte le dimensioni, mantenere le proporzioni o cambiare soltanto l'altezza. Prima di eseguire, occorre risolvere questa ambiguità. Puoi specificare di-

rettamente i nuovi valori oppure chiedere una proposta e decidere dopo. Sono entrambi modi utili di lavorare.

01 Formula una prima esecuzione

“Usa crea_geometria per generare un parallelepipedo largo 8, profondo 5 e alto 3 unità. Restituisci il percorso del file Dynamo e i valori usati.”

02 Definisci una variazione controllata

“Crea una variante: larghezza 8, profondità 5, altezza 6. Conserva la precedente e indica il nuovo file.” Nel kit ogni richiesta produce una copia distinta; nei tuoi strumenti dovrai decidere esplicitamente come gestire i salvataggi.

03 Verifica il significato dei numeri

Apri entrambe le varianti. La base deve restare uguale, mentre l'altezza raddoppia. I volumi teorici sono rispettivamente 120 e 240 unità cubiche. Confronta anche gli slider: la prospettiva può rendere poco evidente una differenza di misura.

FERMATI E RISPONDI Parti da 8 per 5 per 3 e chiedi: “Raddoppia il volume mantenendo identica la base”. Quali argomenti deve ricevere lo strumento?

Risposta guidata. La base misura $8 \times 5 = 40$ unità quadrate. Il nuovo volume deve essere 240, quindi l'altezza diventa $240/40 = 6$. La chiamata usa larghezza 8, profondità 5 e altezza 6. Raddoppiare tutte e tre le dimensioni produrrebbe invece un volume otto volte maggiore.

RISULTATO ATTESO Sai leggere gli argomenti di una chiamata, riconoscere una richiesta ambigua e verificare che l'esecuzione corrisponda alla tua intenzione.

LEZIONE 3

MCP senza misteri

COLLEGARE / UN LINGUAGGIO COMUNE PER GLI STRUMENTI

Ora sappiamo che un assistente può chiedere a una funzione di lavorare. Rimane una domanda pratica: come scopre il nome della funzione, quali argomenti accetta e come riceve il risultato? **MCP**, abbreviazione di *Model Context Protocol*, definisce un modo comune con cui applicazioni e server scambiano queste informazioni.

Un **protocollo** è un insieme di regole per comunicare. Stabilisce come presentare una richiesta e interpretare una risposta. Nel laboratorio MCP organizza la comunicazione con il server; la costruzione del solido continua a dipendere dal codice e dal grafo che abbiamo preparato.

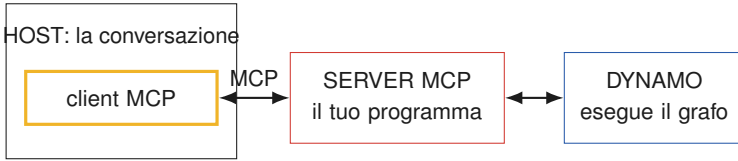
Host, client e server: tre ruoli

L'**host** è l'applicazione nella quale usi l'assistente. Gestisce l'esperienza della conversazione e l'accesso alle connessioni configurate. È il luogo da cui parti quando scrivi la richiesta di creare una geometria.

Il **client MCP** è il componente dell'host che comunica con un server MCP. Si occupa della conversazione tecnica: stabilire la connessione, conoscere le capacità disponibili e scambiare messaggi. Di solito non lo apri come un programma separato; lavora all'interno dell'applicazione.

Il **server MCP** è il programma che espone quelle capacità. Il nostro sarà un processo Python locale che presenta strumenti come `crea_geometria`. Riceve una chiamata, controlla gli argomenti e usa il codice del kit per avviare DynamoCLI. Infine restituisce i dati dell'esecuzione.

La parola *server* può far pensare a una macchina remota. Qui indica prima di tutto un ruolo: un programma risponde alle richieste di un client. Può funzionare sullo stesso computer del lettore, senza richiedere una macchina dedicata.



Schema 3. Il client è nell'host. Il server usa Dynamo attraverso il collegamento previsto dal nostro codice.

Il paragone con nodi e porte

Quando usi un nodo, il nome e le porte ti aiutano a capire quali dati fornire. Un server presenta una descrizione simile dei propri strumenti: operazione, parametri e tipi. Questa descrizione permette al client e all'assistente di lavorare senza conoscere ogni dettaglio del codice interno.

Il paragone ha un limite preciso. MCP non trasforma automaticamente i nodi Dynamo in strumenti disponibili nella chat. Sei tu a scegliere quali operazioni esporre e a scrivere o configurare il collegamento che le realizza. Un solo tool può eseguire un intero grafo; diversi tool possono usare grafi distinti. La struttura del protocollo e la struttura del canvas restano due livelli diversi del progetto.

SDK: gli utensili per costruire il collegamento

Un **SDK**, cioè *Software Development Kit*, è un insieme di strumenti e librerie che aiuta a sviluppare un programma. Il nostro SDK Python gestisce molti dettagli dei messaggi MCP. Possiamo così concentrarci sul nome del tool, sui parametri e sulla funzione da eseguire.

Il protocollo descrive le regole comuni; l'SDK offre codice per applicarle. Dynamo costruisce la geometria; il tuo server coordina l'operazione. Queste responsabilità chiariscono anche dove cer-

care un problema: un argomento rifiutato riguarda la chiamata o la validazione, mentre un nodo geometrico in errore richiede di esaminare il grafo.

Tool, risorse e prompt riutilizzabili

Un server può esporre **tool**, **risorse** e **prompt**. Non deve necessariamente offrire tutte e tre le categorie. Il nostro percorso si concentra sui tool, perché vogliamo eseguire un'azione.

Un **tool** esegue una funzione: per esempio crea una variante del grafo. Una **risorsa** rende consultabile un contenuto, come una descrizione dei parametri o un riepilogo del progetto. Un **prompt MCP** offre una traccia riutilizzabile, eventualmente con campi da compilare, per guidare un'attività. È un uso specifico della parola prompt: qui parliamo di una traccia pubblicata dal server, oltre alle richieste che scrivi liberamente nella chat.

Il modo in cui queste capacità vengono mostrate e utilizzate dipende anche dall'host. Pubblicare una risorsa non significa che il modello la legga in ogni conversazione. Rendere disponibile un tool non significa che venga eseguito a ogni messaggio. Occorre che l'applicazione lo renda accessibile e che sia pertinente al compito.

Locale con stdio, oppure remoto

Nel laboratorio useremo il trasporto `stdio`. La parola riunisce input e output standard: due canali con cui i programmi possono scambiarsi dati. L'host avvia il server come processo locale e comunica attraverso questi canali. Per questa connessione il nostro server non deve aprire una porta di rete.

L'output del processo serve al protocollo. Un messaggio di debug scritto liberamente nello stesso canale può disturbare la comunicazione. Perciò il server usa canali diagnostici adatti, come l'errore standard o un file di log. Vedremo la configurazione concreta prima di avviarlo.

Esistono anche server raggiungibili attraverso la rete. In quel caso

entrano in gioco indirizzo del servizio, connessione e, quando richiesta, autenticazione. Un server MCP locale non implica che ogni parte dell'assistente funzioni offline: la posizione del server degli strumenti e quella del modello sono scelte distinte.

REGOLA C MCP rende disponibile una capacità che hai costruito

La connessione permette di presentare strumenti e scambiare richieste. Il codice del server deve ancora controllare i dati, trovare Dynamo, eseguire il grafo e spiegare il risultato.

Che cosa prepareremo davvero

Configureremo il programma da avviare, il percorso del server e la sua cartella di lavoro. Nel kit indicheremo anche dove si trova DynamoCLI. Queste informazioni permettono ai componenti di incontrarsi; non installano automaticamente dipendenze mancanti, non correggono un grafo difettoso e non concedono accesso al documento Revit.

FERMATI E RISPONDI Il server appare nell'host, ma la creazione geometrica fallisce perché il percorso di DynamoCLI è errato. La connessione MCP è necessariamente guasta?

Risposta guidata. No. La scoperta dello strumento e la ricezione dell'errore mostrano che una parte della comunicazione funziona. Il problema dichiarato riguarda il programma che il tool tenta di avviare. Correggi quel percorso e ripeti una richiesta minima con valori noti.

RISULTATO ATTESO Sai indicare quale programma svolge ogni ruolo e distinguere la connessione MCP dall'esecuzione della geometria. Sei pronto a configurare il tuo primo server con uno scopo preciso.

LEZIONE 4

Prepara il banco di lavoro

DAL TUO DYNAMO A UNA CARTELLA DI ESPERIMENTI

Conosci già nodi, collegamenti e parametri di Dynamo. Adesso aggiungi un piccolo programma capace di offrire una funzione a un'applicazione esterna. Per imparare questo passaggio parti da un risultato che puoi controllare a mente: il volume teorico di un parallelepipedo. La geometria arriverà quando il collegamento sarà comprensibile.

In questi laboratori useremo sempre `C:\DynamoAI\kit`. È una cartella di esempio, scelta per avere percorsi brevi. Contiene i file allegati al libro e un ambiente Python dedicato. Puoi scegliere un'altra posizione: in quel caso devi sostituirla in ogni comando e configurazione. Per la prima prova, mantenere questo percorso riduce il numero di dettagli da controllare.

01 Estrai il kit e rendi visibili le estensioni

Estrai il contenuto della cartella `kit` dell'allegato dentro `C:\DynamoAI\kit`. Aprendola devi vedere direttamente `requirements.txt`, `prima_funzione.py` e `server_primo.py`. Se vedi soltanto un'altra cartella `kit`, sei un livello troppo in alto.

In Esplora file attiva la visualizzazione delle estensioni dei nomi. L'estensione è la parte finale dopo il punto: `.dyn` identifica il grafo che conosci, `.py` un programma Python, `.json` dati strutturati. Un file chiamato `server_primo.py.txt` resta un documento di testo con estensione `.txt`; il suo nome completo rivela l'errore.

Usa un editor per script che conservi testo semplice e spazi. Un elaboratore di testi può aggiungere formattazione e virgolette tipografiche. Apri i file forniti e osservali; modificali dopo la prima prova riuscita.

02 Apri PowerShell nella cartella corretta

PowerShell è la finestra in cui avvii programmi scrivendo comandi. Ogni finestra ha una *cartella corrente*: quando scrivi un nome senza percorso completo, il programma lo cerca a partire da lì. Esegui queste due righe, una alla volta:

```
Set-Location 'C:\DynamoAI\kit'  
Get-Location
```

La seconda riga deve indicare la cartella del kit. Le virgolette delimitano il percorso; non fanno parte del nome. Non copiare anche l'eventuale prefisso `PS` mostrato dal terminale: è il suo invito a scrivere, non una parte del comando.

03 Controlla Python e crea un ambiente dedicato

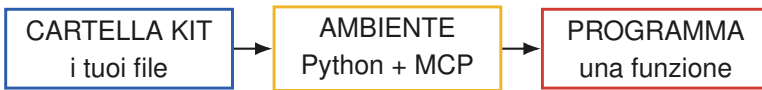
Python è il programma che legge ed esegue i file `.py`. Serve un'installazione di Python per Windows, distinta dal motore Python incorporato in un nodo Dynamo. L'SDK MCP usato qui richiede Python 3.10 o superiore; il riferimento locale di preparazione usa Python 3.11.9. Questa indicazione descrive un ambiente verificato, non garantisce ogni combinazione futura di pacchetti.

Se Python manca, usa la distribuzione ufficiale per Windows e verifica che il comando seguente funzioni prima di continuare.¹ Il comando deve mostrare una versione compatibile:

```
py -3 --version
py -3 -m venv .venv
.\.venv\Scripts\python.exe -m pip install -r requirements.txt
```

La prima riga controlla Python; la seconda crea `.venv`; la terza installa le dipendenze dentro quell'ambiente. Un ambiente virtuale è una cartella con un interprete e i pacchetti destinati a questo progetto. Puoi immaginarlo come un banco con gli attrezzi assegnati al kit. In tutto il libro chiameremo esplicitamente il suo `python.exe`, per sapere quale interprete stiamo usando.

Il file delle dipendenze fissa la versione dell'SDK. La specifica usata è `mcp[cli]==2.2.0`. Il simbolo `==` significa "esattamente questa versione". Le parentesi quadre chiedono anche gli strumenti da terminale del pacchetto. Il `pin` rende ripetibile la scelta dell'SDK; le altre dipendenze hanno comunque le proprie versioni.²



Tre elementi separati: i file del progetto, gli strumenti per eseguirli e il comportamento che scrivi.

04 Prepara l'Inspector

L'Inspector è un'applicazione di prova per server MCP. Il suo avvio usa Node.js e npm: per questo servono anche se il tuo server è scritto in Python. Scegli Node.js 24 LTS dalla pagina ufficiale, seleziona Windows e l'architettura del tuo computer, scarica l'installer `.msi` e segui l'installazione con npm. Poi chiudi e riapri PowerShell. La versione dell'Inspector consultata richiede Node.js almeno 22.19.0.³

¹Download ufficiali Python: <https://www.python.org/downloads/windows/>.

²Release e requisiti: [MCP 2.2.0 su PyPI](#).

³Download: <https://nodejs.org/en/download>. Requisiti dell'Inspector

```
node --version  
npm.cmd --version  
npx.cmd --version
```

Ogni comando deve restituire una versione. Se un nome non è riconosciuto, risolvi l'installazione prima della prova MCP. Scrivere `npm.cmd` e `npx.cmd` seleziona esplicitamente i lanciatori Windows usati in questi esempi.

Account, connessione e costi

La funzione Python e le prove locali non interrogano un modello AI. L'installazione dei pacchetti e il primo avvio dell'Inspector possono scaricare software da Internet. La conversazione con un assistente sarà un passaggio successivo: account, disponibilità delle funzioni e possibili costi dipendono dall'host e dal servizio scelti. Un abbonamento a un'applicazione e l'accesso a un'API sono modalità da verificare separatamente. Nel kit iniziale non devi inserire una chiave API per moltiplicare tre numeri.

RISULTATO ATTESO La cartella del kit è riconoscibile, Python ha creato `.venv`, i pacchetti sono installati e i tre comandi di Node/npm rispondono.

FERMATI E RISPONDI Se hai un nodo Python funzionante in Dynamo, questo dimostra che `py -3` sia disponibile in PowerShell? Perché sono due verifiche diverse?

LEZIONE 5

Il Python che serve per leggere il server

POCHE ISTRUZIONI, OSSERVATE MENTRE LAVORANO

Per seguire il libro devi riuscire a leggere un programma piccolo e a riconoscere dove entrano i dati e dove esce il risultato. Il tuo primo vantaggio è conoscere già Dynamo: una funzione assomiglia a un nodo con ingressi e uscita. Il testo Python scrive quella relazione in righe ordinate.

Apri `prima_funzione.py`. Il file completo del kit è riportato qui. Le righe iniziali eventualmente dedicate a un commento spiegano lo scopo; le istruzioni definiscono il calcolo e mostrano una prova.

```
"""Primo esercizio: una funzione Python, senza MCP e senza
Dynamo."""

def volume(
    larghezza: float,
    profondita: float,
    altezza: float,
) -> float:
    """Moltiplica tre dimensioni espresse nella stessa unità."
    """
    return larghezza * profondita * altezza

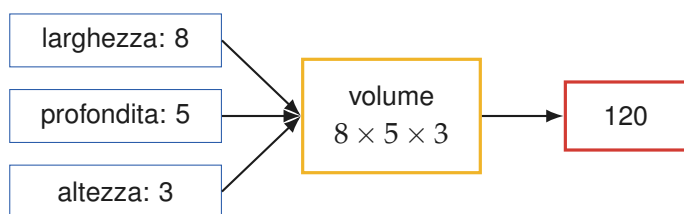
if __name__ == "__main__":
    print(volume(8, 5, 3))
```

Nomi, valori e parametri

Una variabile associa un nome a un valore. Un nome come `altezza` permette di parlare del dato senza ripetere un numero in ogni punto. In una funzione, i nomi degli ingressi si chiamano *parametri*. Nel file, `larghezza`, `profondita` e `altezza` sono i tre parametri di `volume`.

La parola `def` inizia la definizione della funzione. Le parentesi raccolgono gli ingressi; i due punti introducono il corpo. Le righe spostate verso destra appartengono a quel corpo. Quando il programma incontra la definizione, impara quale operazione eseguire; il calcolo avviene quando la funzione viene chiamata con valori concreti.

Nella chiamata `volume(8, 5, 3)`, l'ordine assegna 8 alla larghezza, 5 alla profondità e 3 all'altezza. È come collegare tre slider agli ingressi corretti. Python consente anche una chiamata con nomi espliciti: `volume(larghezza=8, profondita=5, altezza=3)`. In questo caso la scrittura rende visibile l'associazione fra ciascun valore e il suo parametro.



La funzione scritta in Python mantiene la logica che conosci nei nodi: ingressi definiti, una regola, un risultato.

Gli spazi fanno parte della struttura

In Python il rientro delle righe ha significato. Il corpo di una funzione è indentato; in questi file si usano quattro spazi. Non aggiungere o togliere spazi iniziali per allineare il testo a gusto personale. Due righe con lo stesso rientro appartengono allo stesso livello; un rientro ulteriore introduce un livello interno, per esempio il controllo di una condizione.

Il carattere # introduce un commento fino alla fine della riga. Un commento aiuta chi legge; Python non lo esegue come istruzione. Un testo racchiuso fra tre virgolette subito dentro una funzione è invece una *docstring*: descrive la funzione e può essere letto anche dagli strumenti che ne costruiscono la documentazione.

Restituire e stampare sono due operazioni

`return` consegna un valore a chi ha chiamato la funzione. `print` scrive un testo nel terminale. Nel primo esempio lavorano insieme: la funzione restituisce il volume e la chiamata esterna lo stampa per te. Lancia il file dalla cartella del kit:

```
.\.venv\Scripts\python.exe prima_funzione.py
```

RISULTATO ATTESO Compare 120 e torna l'invito di PowerShell. Il programma ha concluso il lavoro.

Modifica soltanto il terzo numero della chiamata, passando da 3 a 6. Salva il file prima di eseguirlo di nuovo. Ora il risultato atteso è 240. Se continua a comparire 120, controlla quale file hai salvato e quale file stai eseguendo. Il terminale legge il file su disco, non una modifica ancora aperta nell'editor.

Numeri con decimali e risultati con nomi

Nel server vedrai parametri annotati con : `float`. Questa annotazione comunica che il parametro rappresenta un numero che può avere decimali. Nel codice Python e nei dati JSON si scrive `2.5`, con il punto. La virgola ha un'altra funzione: separa parametri o elementi. L'SDK usa queste annotazioni per descrivere gli ingressi del tool; aggiungeremo comunque controlli espliciti sui valori ammessi.

Un dizionario Python raccoglie coppie di nomi e valori. Nel nostro risultato, `volume_teorico` identifica il prodotto e `dimensioni` raccoglie i tre ingressi. Il client distingue così i dati senza cercare numeri dentro una frase. Le parentesi graffe delimitano la struttura; i nomi hanno virgolette.

JSON è un formato di scambio dei dati, non un programma Python da eseguire. L'SDK può trasformare un dizionario con valori adatti in un risultato strutturato del protocollo. La somiglianza grafica non significa che ogni espressione Python sia anche JSON valido. Qui usiamo numeri e nomi semplici, quindi la relazione resta facile da osservare.

REGOLA P1 Segui un valore dall'ingresso all'uscita

Per capire un frammento, scegli un dato: per esempio `altezza=3`. Individua dove entra, quale operazione lo usa e con quale nome compare nel risultato. Ripeti con gli altri parametri prima di modificare la struttura.

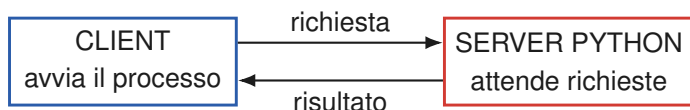
FERMATI E RISPONDI Perché una funzione che esegue soltanto `print(120)` non offre automaticamente lo stesso risultato di una funzione che esegue `return 120`?

LEZIONE 6

Il tuo primo server MCP

LA FUNZIONE DIVENTA UNO STRUMENTO DISPONIBILE

Ora aggiungi a una funzione Python la descrizione necessaria perché un client MCP possa scoprirla e chiamarla. Il primo server espone un solo tool: `volume_teorico`. Riceve tre dimensioni, ne verifica i limiti e restituisce il prodotto insieme agli ingressi. Questo tool calcola un numero; in questo capitolo non crea ancora geometria Dynamo.



Con `stdio`, il client gestisce il processo e i suoi canali. Il terminale non diventa una chat con il server.

REGOLA P2 L'output del server appartiene al protocollo

Nel server usa `return` per i risultati dei tool. Evita `print()` sull'output standard: messaggi estranei possono disturbare lo scambio MCP. Il `print()` del primo esercizio era destinato alla lettura umana nel terminale.

Apri `server_primo.py`. La lista seguente proviene direttamente dal file del kit. Leggila una sezione alla volta, seguendo le spiegazioni successive: non è necessario memorizzare ogni simbolo prima della prova.

Il programma completo

```
"""Primo server MCP: calcola un volume, senza avviare Dynamo."""

import math
from typing import Any

from mcp.server import MCPServer
from mcp.server.mcpserver.exceptions import ToolError

mcp = MCPServer("Il mio primo server MCP")

@mcp.tool()
def volume_teorico(
    larghezza: float, profondita: float, altezza: float
) -> dict[str, Any]:
    """Calcola il volume di tre dimensioni, ciascuna da 0.1 a
    100."""
    dimensioni = {
        "larghezza": larghezza,
        "profondita": profondita,
        "altezza": altezza,
    }
    for nome, valore in dimensioni.items():
        if not math.isfinite(valore) or not 0.1 <= valore <=
100:
            raise ToolError(
                f"{nome} deve essere un numero finito tra 0.1
e 100."
            )
    return {
        "dimensioni": dimensioni,
        "volume_teorico": larghezza * profondita * altezza,
    }

if __name__ == "__main__":
    mcp.run(transport="stdio")
```

Le importazioni preparano gli strumenti

`import math` rende disponibili funzioni matematiche della libreria standard di Python. Qui serve `math.isfinite`, che distingue numeri finiti da valori speciali come infinito. Le righe che iniziano con `from mcp` prendono dall'SDK la classe del server e il tipo di errore usato per comunicare un problema al client.

Il termine *classe* indica una definizione con cui il programma crea un oggetto. Per questo laboratorio basta riconoscere che `MCPServer(...)` crea l'oggetto incaricato di pubblicare e gestire i tool. La variabile `mcp` conserva quell'oggetto. Il testo fra parentesi è il nome del server, utile per identificarlo.

`Any` arriva dal modulo `typing` e compare nell'annotazione di uscita `dict[str, Any]`. La freccia `->` descrive il risultato atteso: un dizionario con nomi testuali e valori che possono avere strutture diverse. Nel nostro caso c'è un dizionario di dimensioni e un numero. Conserva questa annotazione: contribuisce alla descrizione del risultato strutturato nell'SDK usato dal kit.

Il decoratore pubblica la funzione

La riga `@mcp.tool()` è un *decoratore*: associa al server la funzione che compare subito sotto. Senza quella registrazione, una normale funzione Python non diventa automaticamente visibile a un client MCP. Il nome della funzione diventa il nome del tool; i parametri annotati contribuiscono a descriverne gli ingressi; la docstring ne spiega lo scopo.

Questa descrizione costituisce l'interfaccia che un host può presentare a un modello. Una frase come “calcola un volume” è utile per una persona; nomi come `larghezza`, `profondita` e `altezza` rendono precisa la chiamata. Non serve che un assistente indovini l'ordine di tre numeri senza nome.

La funzione decide quali valori accettare

Il server raccoglie le dimensioni in un dizionario e controlla ciascuna coppia nome-valore. Il ciclo `for` ripete la stessa verifica per tutti

e tre gli ingressi. `if` introduce una condizione: quando un valore non è finito oppure esce dall'intervallo ammesso, il programma solleva `ToolError` con un messaggio leggibile.

Accettiamo valori da 0,1 a 100 inclusi. È una regola scelta per il laboratorio, coerente con i controlli del successivo grafo. Non è un limite universale di Dynamo o MCP. Il nome del parametro nel messaggio permette di correggere l'ingresso giusto: se l'altezza vale zero, cambiare la larghezza non risolve il problema.

Se i controlli passano, `return` consegna un dizionario con dimensioni e volume teorico. Il prodotto serve come verifica numerica. Non misura una geometria già costruita e non dimostra da solo che esista un solido: quel controllo entrerà nel percorso con Dynamo.

L'avvio mette il processo in ascolto

La condizione finale `if __name__ == "__main__"` distingue il file avviato direttamente dal file importato da un altro programma. Nell'avvio diretto viene eseguito `mcp.run(transport="stdio")`. Il nome `stdio` indica che client e server scambiano messaggi attraverso i canali di input e output del processo.

Il client avvia Python, Python esegue il server e il processo resta disponibile per le richieste. Non devi scegliere una porta di rete per questo collegamento locale. Se avvii il file manualmente e la finestra resta silenziosa, può essere semplicemente in attesa del client; non hai ancora dimostrato che un tool risponda correttamente. La prova del capitolo successivo lo verificherà.

FERMATI E RISPONDI Quale riga rende visibile il tool? Quale parte stabilisce se l'altezza zero è accettabile? Sono responsabilità diverse.

LEZIONE 7

Provalo senza aprire una chat

DALLA LISTA DEGLI STRUMENTI A UNA CHIAMATA VERIFICATA

Un server può essere provato anche senza un modello AI. L'Inspector fa il lavoro del client: avvia il processo, scopre gli strumenti e invia gli argomenti che scegli. Questo ti permette di distinguere un problema del tool da un'interpretazione imprecisa di una frase in lingua naturale.

01 Avvia il file giusto con il Python giusto

Apri PowerShell. Le prime due righe qui sotto assegnano percorsi assoluti a due nomi temporanei di PowerShell; la terza passa quei percorsi all'Inspector. Il simbolo \$ appartiene a PowerShell. Non devi incollare queste righe nel file Python.

```
$py = 'C:\DynamoAI\kit\.venv\Scripts\python.exe'  
$srv = 'C:\DynamoAI\kit\server_primo.py'  
npx.cmd @modelcontextprotocol/inspector $py $srv
```

Al primo avvio npm può chiedere di scaricare il pacchetto dell'Inspector. Il terminale mostra l'indirizzo da aprire nel browser, eventualmente con un token di sessione: usa quell'indirizzo completo. Il processo deve restare aperto mentre esegui la prova. Questo comando usa direttamente il Python della tua `.venv`; non richiede di aggiungere `uv` al percorso di studio.

L'aspetto dell'Inspector può cambiare fra versioni. Cerca la connessione al server locale avviato e la sezione dedicata agli strumenti disponibili. Quando l'interfaccia richiede di stabilire la connessione, completala usando il comando e gli argomenti già indicati dal lanciatore.

02 Controlla la lista degli strumenti

Richiedi l'elenco dei tool. Nel vocabolario MCP questa operazione è `tools/list`: il client chiede al server quali strumenti espone. Devi trovare `volume_teorico`, i tre ingressi con i nomi previsti e una descrizione del calcolo. Se la lista manca, non iniziare ancora a cambiare numeri: prima bisogna risolvere l'avvio o la connessione.

Se compare uno strumento diverso, controlla il percorso del server. Nel kit esistono più file per tappe diverse. Una connessione funzionante al file sbagliato resta una prova diversa da quella richiesta in questo capitolo.

03 Invia la prima chiamata

Seleziona il tool e inserisci larghezza 8, profondità 5, altezza 3. Se l'interfaccia propone un editor JSON, usa la seguente struttura. I nomi hanno virgolette doppie e i valori numerici non hanno virgolette:

```
{"larghezza": 8, "profondita": 5, "altezza": 3}
```

Invia la richiesta e leggi il risultato strutturato. Una chiamata a un tool è l'operazione `tools/call`; nei programmi Python del kit la esegue il metodo `call_tool`. Il server deve restituire i tre valori dentro `dimensioni` e `volume_teorico` uguale a 120. La visualizzazione può mostrare `120.0`: rappresenta lo stesso valore numerico.

RISULTATO ATTESO Il tool compare nella lista e la chiamata con 8, 5, 3 restituisce volume teorico 120, insieme alle dimensioni usate. Non viene generato un file Dynamo in questa prova.

04 Prova anche un ingresso non valido

Invia una seconda chiamata con larghezza 0, profondità 5 e altezza 3. Questa volta il risultato atteso è un errore che indica la larghezza e l'intervallo da 0,1 a 100. Non correggere il server per ottenere un volume zero: il suo compito è applicare la regola dichiarata. Ripristina 8 e verifica che la richiesta torni a funzionare.

Queste due prove controllano comportamenti diversi. La prima dimostra che il percorso valido restituisce un risultato; la seconda che un ingresso fuori contratto viene riconosciuto. Un errore atteso, con un messaggio utile, è una parte del funzionamento corretto del tool.

La stessa prova attraverso un piccolo client Python

Il kit contiene anche `prova_primo_server.py`. Puoi usarlo come verifica rapida o per proseguire se non hai ancora predisposto il browser dell'Inspector. Dalla cartella del kit esegui:

```
.\.venv\Scripts\python.exe prova_primo_server.py
```

Il programma importa l'oggetto `server` e usa il client in memoria dell'SDK. Elenca il tool, invia la terna valida, verifica il volume e controlla due ingressi non validi. Alla fine deve comparire il messaggio che conferma una chiamata valida e due input rifiutati. Puoi leggere il file per riconoscere i nomi delle operazioni; per adesso non serve studiare la gestione asincrona del client.

Durante i rifiuti previsti possono comparire anche messaggi diagnostici dell'SDK. Confrontali con il caso appena inviato e con il riepilogo finale: la presenza della parola "errore" in questa prova non significa automaticamente che il test sia fallito.

Questa prova verifica funzione e registrazione dei tool senza avviare un processo stdio separato. È utile per restringere la diagnosi: se passa e l'Inspector fallisce, controlla il lanciatore, i percorsi e la connessione. Non dimostra da sola che un host esterno sia configurato correttamente.¹

Come fermarti e come riconoscere un problema

Per terminare l'Inspector, torna al terminale che lo ha avviato e premi `Ctrl+C`. Se avevi avviato manualmente il server stdio, usa lo stesso comando per interrompere la sua attesa. Per eseguire altre istruzioni apri una nuova finestra PowerShell oppure attendi

¹SDK Python: [test con il client in memoria](#).

il ritorno del suo invito: non digitare comandi dentro l'input di un server stdio in esecuzione.

Un errore di importazione di `mcp` suggerisce di controllare ambiente e installazione. Un file non trovato suggerisce di ricontrollare percorso e nome completo. Un messaggio che cita un parametro, dopo una chiamata con zero, può essere proprio la risposta attesa. Prima di modificare il codice, identifica in quale passaggio si trova il problema.

FERMATI E RISPONDI Se il tool compare e rifiuta correttamente larghezza zero, hai un errore di connessione oppure una verifica riuscita? Quale prova manca ancora per arrivare dalla frase alla geometria?

LEZIONE 8

Prepara il grafo per il dialogo

OBIETTIVO

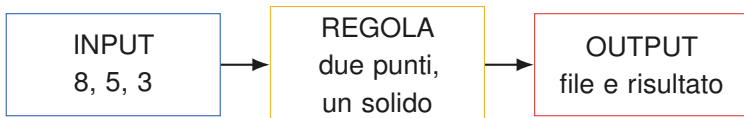
Descrivere un grafo come uno strumento: quali dati accetta, quale regola esegue e quale risultato restituisce.

Hai già provato un server MCP che calcola un volume. Adesso sostituiamo il semplice calcolo con un'azione che conosci: eseguire un grafo Dynamo. Il nuovo passaggio richiede un confine chiaro fra il grafo e chi lo chiama.

Tre domande prima del codice

Prendi il parallelepipedo del kit. Quali sono gli input? Tre dimensioni: larghezza, profondità e altezza. Che cosa fa il grafo? Costruisce due punti e un `Cuboid.ByCorners`. Che cosa vuoi ricevere? Un nuovo file `.dyn`, i valori utilizzati e la descrizione del solido.

Scrivere queste risposte significa definire il **contratto del tool**. La parola “contratto” indica un accordo sui dati: nomi, significato, limiti e risultato. Non è un nuovo formato da imparare. È la stessa chiarezza che cerchi quando prepari un custom node per un collega.



Schema del contratto. Il significato degli input resta uguale nella frase, nel codice e nel grafo.

01 Apri il template del kit

Apri `grafos/parallelepipedo_template.dyn`. Il punto basso è all'origine. I tre slider controllano X, Y e Z del punto alto. La convenzione di questo libro è X = larghezza, Y = profondità, Z = altezza.

02 Fai una prova nota

Imposta 8, 5 e 3. Esegui e osserva il parallelepipedo. Il volume teorico è $8 \times 5 \times 3 = 120$ unità cubiche. Questa verifica precede qualsiasi conversazione con l'IA.

03 Distingui i due risultati

Il Watch del template descrive un oggetto `Cuboid`; non calcola automaticamente il suo volume numerico. Il campo `volume_teorico`, che vedrai nella risposta Python, moltiplica le tre dimensioni. Per misurare il volume geometrico nel grafo puoi aggiungere un nodo dedicato nella tua copia e confrontare i due valori.

REGOLA G1 Mantieni un'unica convenzione di unità

Nel laboratorio i numeri sono unità geometriche del grafo. Scrivere "metri" nella chat non converte da solo le unità di Dynamo o di Revit. Se scegli di interpretare un'unità come un metro, dichiaralo nel progetto e conserva questa convenzione in ogni passaggio.

Il template è una ricetta riutilizzabile

Un **template** è il grafo modello da cui il programma crea copie. Il server conserva la ricetta e cambia solo i valori consentiti. Una richiesta 8, 5, 3 e una richiesta 12, 4, 2 producono due file distinti: puoi aprirli, confrontarli e continuare a modificarli.

Per questa prima integrazione usa il template distribuito. Se ricrei i nodi da zero, Dynamo assegna nuovi identificatori e il manifesto del kit non li riconosce più. Il disegno del grafo può essere identico mentre i suoi identificatori interni sono diversi.

FERMATI E RISPONDI Se l'assistente chiede una sfera, può ottenerla modificando soltanto i tre slider di questo template?

RISPOSTA GUIDATA No. I tre slider cambiano le dimensioni di un parallelepipedo. Una sfera richiede una regola geometrica diversa e un contratto adatto, per esempio centro e raggio. Il tool deve dichiarare ciò che sa costruire.

RISULTATO ATTESO Il template funziona in Dynamo; sai dire che cosa controllano i tre input e distingui la descrizione del solido dal volume teorico.

LEZIONE 9

Dynamo senza aprire il canvas

OBIETTIVO

Verificare DynamoCLI con un grafo semplice prima di collegarlo al server.

La sigla **CLI** significa *Command Line Interface*: un'interfaccia a cui dai istruzioni scritte. DynamoCLI apre ed esegue un grafo senza mostrare il canvas. Il server Python lo userà come motore di calcolo.

Compatibilità prima di automazione

Il laboratorio è stato eseguito con DynamoCLI 2.19.3.6394, presente nel bundle Dynamo for Revit 2024 della macchina di preparazione. Questo dato descrive una prova concreta. Non garantisce che ogni CLI di ogni versione si avvii nello stesso modo: dipendenze e librerie geometriche contano.

Lascia l'eseguibile nella sua cartella di installazione, accanto alle DLL del bundle. Copiare solo `DynamoCLI.exe` nella cartella del kit può interrompere il caricamento delle dipendenze. Il download di Dynamo Sandbox, da solo, non garantisce la disponibilità della geometria: il setup ufficiale specifica i prodotti Autodesk compatibili e i relativi requisiti.

01 Individua il tuo eseguibile

Nel tuo ambiente cerca il file `DynamoCLI.exe` nel bundle Dynamo. Copiane il percorso completo. Il percorso del laboratorio è un esempio da adattare alla tua installazione.

Comandi PowerShell:

```
$cli = 'C:\Program Files\Autodesk\Revit 2024\AddIns\
    DynamoForRevit\DynamoCLI.exe'
Test-Path -LiteralPath $cli
& $cli -h
```

Test-Path deve restituire True. Il carattere & chiede a PowerShell di eseguire il programma indicato dalla variabile. -h richiede l'aiuto; verifica che la tua versione riconosca le opzioni utilizzate.

02 Esegui il grafo numerico

Apri PowerShell nella cartella C:\DynamoAI\kit. Esegui questi comandi; New-Item crea la cartella di lavoro se non c'è già.

```
New-Item -ItemType Directory -Path .\uscite -Force
$grafo = (Resolve-Path .\grafos\area Rettangolo.dyn).Path
$ids = [guid]::NewGuid().ToString('N')
$xml = Join-Path $PWD "uscite\area_$ids.xml"
Push-Location (Split-Path -Parent $cli)
& $cli -o $grafo -v $xml --DisableAnalytics
$esito = $LASTEXITCODE
Pop-Location
$esito
Test-Path -LiteralPath $xml
$xml
```

-o indica il grafo da aprire. -v indica dove scrivere le valutazioni dei nodi in XML. I due percorsi hanno ruoli diversi: il primo è un input, il secondo è un risultato.

Le prime righe preparano percorsi assoluti e un nome XML nuovo per ogni prova. Push-Location porta il terminale nella cartella del CLI; Pop-Location lo riporta al kit. Il codice di uscita \$esito deve essere zero e il controllo del nuovo XML deve dare True. Così non scambi il risultato di ieri per una prova riuscita oggi.

03 Leggi una prova concreta

Apri in un editor il file \$xml. Il nodo di risultato del grafo deve riportare $6 \times 4 = 24$. L'XML può contenere anche valori intermedi: cerca l'uscita del nodo finale, non un numero qualsiasi.

SE QUALCOSA NON VA

Il programma non crea l'XML. Conserva il messaggio completo. Ricontra il percorso, bundle e grafo. Il fatto che Dynamo con interfaccia funzioni non prova automaticamente che quel CLI possa avviarsi nello stesso ambiente.

XML: una busta con etichette

Non devi imparare XML per completare il progetto. Ti basta riconoscerlo come un formato che associa valori a etichette. Il server legge la valutazione del nodo che gli abbiamo indicato. La descrizione del Cuboid arriverà da quella valutazione, non da una frase inventata dall'assistente.

Nel percorso del libro non usiamo `-g` per esportare geometria. La documentazione distingue opzioni e comportamenti di Dynamo-CLI e DynamoWPFCLI. Il prodotto di questa esercitazione resta il grafo `.dyn` da aprire.

FERMATI E RISPONDI Perché provare prima l'area di un rettangolo?

RISPOSTA GUIDATA Il risultato 24 è facile da riconoscere e il grafo numerico separa la prova di avvio del CLI dalla prova delle librerie geometriche. Se questo passaggio funziona e il Cuboid fallisce, hai ristretto la ricerca del problema.

Approfondimento ufficiale: [Dynamo Command Line Interface](#).

Approfondimento ufficiale: [Dynamo Primer: setup](#).

LEZIONE 10

Collega il server al tuo grafo

OBIETTIVO

Configurare il ponte completo e ottenere un nuovo file Dynamo attraverso un tool MCP.

Il file `server.py` del kit contiene il ponte completo. Il tuo primo server ti ha insegnato che una normale funzione Python può diventare un tool. Ora la funzione chiama `DynamoCLI`, aspetta la valutazione e restituisce dati leggibili.

Conosci i tre file di controllo

`config.json` dice dove si trova il motore. `manifest.json` elenca i grafi consentiti e i nodi da usare. `server.py` contiene le funzioni. Tenerli distinti aiuta a cambiare il percorso del programma senza riscrivere la logica del server.

01 Crea la configurazione

Nella cartella del kit copia `config.example.json` e chiama la copia `config.json`. Aprila nell'editor e inserisci il tuo percorso, già verificato nella lezione precedente.

Esempio JSON da adattare, nel file config.json:

```
{  
  "dynamo_cli": "C:\\Program Files\\Autodesk\\Revit 2024\\  
    AddIns\\DynamoForRevit\\DynamoCLI.exe"  
}
```

Dentro una stringa JSON ogni barra rovesciata del percorso è scritta due volte. Il percorso reale contiene una sola barra. Non aggiungere commenti o una virgola dopo l'ultimo campo: il formato JSON ha una sintassi precisa.

Il kit riconosce anche la variabile d'ambiente `DYNAMO_CLI_PATH`, che ha precedenza sul file. Per la prima prova usa un solo metodo. Se hai già definito quella variabile in un'esercitazione precedente, controlla che indichi lo stesso eseguibile.

02 Leggi il manifesto senza cambiarlo

Apri il manifesto e cerca i nomi pubblici dei due grafi:

- `area Rettangolo`: il calcolo numerico;
- `parallelepipedo`: il template geometrico.

Ogni voce contiene il file e l'identificatore del nodo che fornisce il risultato.

Un **GUID** è un identificatore: una lunga etichetta che distingue un oggetto dagli altri. Per questo esercizio non serve memorizzarlo. Pensa alla differenza fra il nome visibile di un nodo e la sua identità nel file. Il server usa l'identità per evitare di leggere l'output del nodo sbagliato.

03 Avvia il ponte nell'Inspector

Chiudi la precedente sessione dell'Inspector, poi usa lo stesso comando della prova iniziale cambiando il file in `server.py`:

```
npx.cmd @modelcontextprotocol/inspector "C:\DynamoAI\kit\.venv
  \Scripts\python.exe" "C:\DynamoAI\kit\server.py"
```

Il comando è una singola riga. Nel PDF può andare a capo per ragioni di impaginazione. Se hai salvato il kit altrove, sostituisci entrambi i percorsi.

04 Interroga il catalogo

Invoca `elenca_grafi`, senza argomenti. Devi riconoscere le due voci del manifesto. Poi invoca `esegui_grafo` con questo oggetto:

```
{"nome": "area_rettangolo"}
```

Il campo del risultato deve riportare 24. Hai provato l'intera strada MCP–Python–DynamoCLI usando un caso che conosci già.

05 Crea il parallelepipedo

Seleziona `crea_geometria` e inserisci gli argomenti:

```
{"larghezza": 8, "profondita": 5, "altezza": 3}
```

Attendi la fine della chiamata. Il campo `file_dyn` indica un file nella cartella `uscite`. Apri quel file in Dynamo. È la copia prodotta dalla chiamata, non il template originale.

Leggi la risposta come una scheda tecnica

dimensioni riporta i dati utilizzati. **geometria** descrive ciò che il Watch ha restituito. **volume_teorico** contiene la moltiplicazione. **file_dyn** permette di ritrovare il risultato. Tutti e quattro hanno uno scopo diverso.

Nel test di riferimento, il Cuboid descrive `Length = 5, Width = 8, Height = 3`. Le etichette del costruttore non coincidono nell'ordine con i nomi italiani scelti per X e Y. Confronta i valori e la convenzione dichiarata, senza dedurre una rotazione del solido soltanto dall'ordine delle parole.

REGOLA G2 Una risposta riuscita deve portarti al file

Apri il `.dyn`, controlla gli slider e osserva il solido. Il numero 120, da solo, può essere ottenuto anche dal primo server aritmetico. È l'insieme di dati, file e verifica geometrica a documentare la prova completa.

Che cosa succede dentro `crea_geometria`

1. Controlla che i tre valori siano finiti e compresi fra 0,1 e 100.
2. Legge il template e individua i tre slider attraverso il manifesto.
3. Aggiorna i valori in una nuova copia con un nome distinto.
4. Avvia il CLI usando quel file e aspetta al massimo 120 secondi.
5. Legge l'output del Watch identificato nel manifesto.
6. Restituisce dati e percorso del nuovo grafo.

Questo elenco è una mappa di lettura. Il programma completo, senza parti omesse, è nel file `server.py`. Aprilo e cerca il nome di ciascuna funzione mentre segui il percorso; non occorre comprendere tutti i dettagli della lettura XML per usare il contratto.

FERMATI E RISPONDI Quale parte cambieresti se DynamoCLI fosse installato in un'altra cartella? E se cambiassi il nodo Watch del template?

RISPOSTA GUIDATA Nel primo caso cambia il percorso in `config.json`. Nel secondo cambia l'identificatore del risultato nel manifesto e ripeti le prove. Sono due modifiche diverse: posizione del motore e identità del nodo.

LEZIONE 11

La prima conversazione con l'IA

OBIETTIVO

Collegare un host locale e verificare che una richiesta in italiano diventi la chiamata che ti aspetti.

Hai già dimostrato che il tool funziona nell'Inspector. Il nuovo elemento è l'assistente che interpreta la frase e sceglie gli argomenti. Il programma geometrico resta quello appena verificato.

Scegli un host che possa avviare il processo

Un host locale deve poter eseguire il Python del tuo ambiente virtuale e leggere `server.py`. La configurazione mostrata qui riguarda Codex sul computer del laboratorio. Una chat aperta nel browser non legge automaticamente i programmi installati sul tuo PC. Per altri host usa la loro documentazione MCP e inserisci gli stessi percorsi di avvio.

01 Apri il file di configurazione dell'host

Per Codex la configurazione utente è in `.codex/config.toml` nella cartella del tuo profilo. Conserva una copia del file esistente. Aggiungi la sezione seguente senza cancellare le altre impostazioni. Usa un solo nome di sezione per questo collegamento.

Configurazione TOML, da adattare ai tuoi percorsi:

```
[mcp_servers.dynamo_ai_illustrata]
command = 'C:\DynamoAI\kit\.venv\Scripts\python.exe'
args = ['C:\DynamoAI\kit\server.py']
cwd = 'C:\DynamoAI\kit'
tool_timeout_sec = 150
```

`command` avvia Python; `args` indica lo script; `cwd` imposta la cartella iniziale. Le virgolette singole di TOML conservano le barre Windows senza raddoppiarle. `tool_timeout_sec` lascia all'host più tempo dei 120 secondi consentiti al CLI dal nostro server.

02 Riavvia il collegamento

Salva, riavvia il server o il client secondo l'interfaccia che usi e consulta l'elenco MCP. Nel client da terminale Codex, `/mcp` mostra i server attivi. Cerca i tool del ponte. Se non compaiono, torna ai percorsi: il grafo non è ancora coinvolto in questa fase.

Approfondimento ufficiale: [OpenAI Docs: collegare server MCP e configurare i timeout](#).

03 Invia una richiesta completa

Usa il tool `crea_geometria` del mio server Dynamo. Crea un parallelepipedo con larghezza 8, profondità 5 e altezza 3. Riporta i parametri inviati, il volume teorico e il percorso del file Dynamo prodotto.

Questa richiesta specifica un'azione, tre dati e un risultato verificabile. Se l'host richiede la conferma di una chiamata, leggi il nome del tool e i parametri prima di confermare. La modalità di conferma dipende dall'host e dalle sue impostazioni.

04 Confronta tre livelli

Controlla ciò che hai chiesto, ciò che il tool ha ricevuto e ciò che il grafo contiene. Devono concordare. L'assistente può descrivere correttamente l'intenzione anche dopo una chiamata fallita: il

messaggio di errore e l'esistenza del file hanno quindi un peso preciso nella verifica.

Un esperimento che rende visibile il ruolo dell'IA

Invia prima “rendilo più alto”. La frase dipende dal contesto: quale file e quale altezza? Poi invia “crea una nuova variante con larghezza 8, profondità 5 e altezza 6, conservando la precedente”. La seconda richiesta rende esplicita la trasformazione.

Il server crea una nuova copia a ogni chiamata. L'host può ricordare il dialogo, ma il tool riceve comunque i valori della chiamata corrente. Per una variante importante scrivi i tre numeri: sono facili da controllare e da riprodurre in un'altra sessione.

SE QUALCOSA NON VA

La risposta contiene istruzioni ma nessun file. Chiedi all'assistente di dichiarare se ha invocato il tool e quale risultato ha ricevuto. Una spiegazione su come costruire il grafo è un contenuto testuale; la creazione richiesta deve restituire anche il percorso verificabile.

RISULTATO ATTESO L'host invoca `crea_geometria`, riceve una risposta riuscita e indica un file che apri nel tuo Dynamo.

LEZIONE 12

Scrivi un tool che è tuo

OBIETTIVO

Aggiungere `crea_cubo`: un nuovo comando con un solo parametro, costruito usando la logica già verificata.

Un cubo è un caso particolare del parallelepipedo: le tre dimensioni sono uguali. La novità è il modo in cui vuoi esporre questa operazione all'assistente. Aniché chiedere tre numeri uguali, pubblici un tool che accetta `lato`.

Questo esercizio mostra come progettare un piccolo comando personale. La geometria rimane una tua regola esplicita; l'assistente riceve un'interfaccia più semplice.

01 Leggi il contratto prima del programma

Nome: `crea_cubo`. Input: `lato`, numero fra 0,1 e 100. Regola: usa il `lato` come larghezza, profondità e altezza. Output: la stessa scheda del parallelepipedo, con file e parametri. Con `lato 4`, il volume teorico è $4^3 = 64$.

02 Apri il nuovo file completo

Il kit contiene `server_esteso.py`. È un secondo punto di avvio che importa il server esistente e aggiunge il tool. Il listato seguente corrisponde al file completo:

```
"""Aggiunge un cubo al server geometrico già funzionante."""

from typing import Any

from server import crea_geometria, mcp

@mcp.tool()
def crea_cubo(lato: float) -> dict[str, Any]:
    """Crea un cubo Dynamo: il lato deve essere un numero
    finito da 0.1 a 100."""
    return crea_geometria(larghezza=lato, profondita=lato,
                           altezza=lato)

if __name__ == "__main__":
    mcp.run(transport="stdio")
```

Leggi il programma come un piccolo grafo

L'importazione recupera l'oggetto `mcp` e la funzione che sa già creare il parallelepipedo. Il decoratore registra il nuovo tool. La funzione trasferisce lo stesso valore a tre parametri. Il blocco finale avvia il server solo quando questo file viene eseguito come programma principale.

Il codice di `server.py` non viene duplicato. Quando correggi il ponte in quel file, anche il tool del cubo usa la logica aggiornata. È una scelta simile al riutilizzo di un custom node invece di copiare molti nodi in grafi separati.

03 Prova il nuovo punto di avvio

Nel comando dell'Inspector sostituisci `server.py` con `server_esteso.py`. Dopo la connessione, aggiorna l'elenco dei tool e cerca `crea_cubo`. Invocalo con:

```
{"lato": 4}
```

Il risultato deve indicare tre dimensioni pari a 4 e un volume teorico pari a 64. Apri il nuovo file. La prova automatica equivalente è disponibile in `prova_cubo.py`.

04 Aggiorna il collegamento dell'host

Nella stessa sezione TOML cambia soltanto il percorso dello script nell'elenco `args`, da `server.py` a `server_esteso.py`. Salva e riavvia il collegamento. Il nuovo processo deve rileggere il codice: salvare il file Python non aggiorna un processo già in esecuzione.

05 Usa il tuo comando nella conversazione

Scrivi: “usa `crea_cubo` con lato 4 e restituisci il file”. Confronta questa chiamata con quella precedente a tre parametri. Hai deciso come rendere la tua operazione più comprensibile all'assistente e a chi userà il server.

FERMATI E RISPONDI Che cosa accade con lato uguale a zero?

RISPOSTA GUIDATA La validazione del ponte rifiuta la dimensione. Riutilizzare `crea_geometria` conserva anche il controllo dei limiti. Il nuovo tool non deve aggirare una regola geometrica solo perché ha un'interfaccia più corta.

REGOLA G3 Dai al parametro un significato stabile

Un tool chiamato `crea_cubo` deve produrre un cubo. Se vuoi un pilastro con base quadrata e altezza libera, scegli un altro nome e due parametri. Il nome, la descrizione e la funzione devono raccontare la stessa operazione.

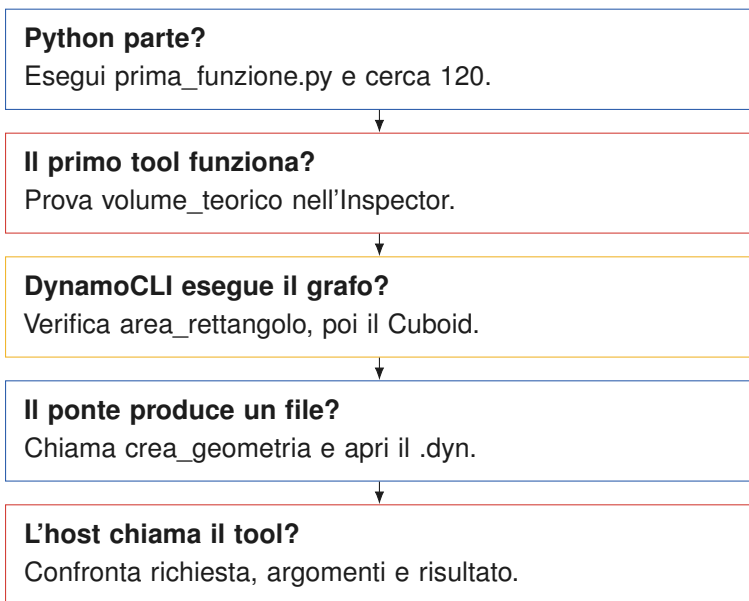
LEZIONE 13

Quando qualcosa non funziona

OBIETTIVO

Individuare il passaggio che ha fallito e fare una prova piccola prima di cambiare altre cose.

La prima integrazione può fallire anche se hai compreso l'idea. Un carattere nel percorso, un processo rimasto aperto o una versione diversa bastano a produrre un errore. La domanda utile è: *qual è l'ultimo passaggio che ho visto funzionare?*



Schema di diagnosi. Fermati al primo passaggio che non riesci a verificare.

Il terminale dice che Python non esiste

Apri una nuova finestra di PowerShell dopo l'installazione. Prova il comando di versione indicato nella lezione di preparazione. Se `python` avvia uno store o un programma diverso, verifica l'installazione e il percorso dell'eseguibile. Nel resto del laboratorio, il percorso esplicito `.venv\Scripts\python.exe` elimina l'ambiguità fra interpreti.

Compare `ModuleNotFoundError: mcp`

Il Python che esegue lo script non trova il pacchetto. Controlla quale interprete hai usato per installarlo e quale hai indicato nell'Inspector o nell'host. Esegui da PowerShell:

```
.\.venv\Scripts\python.exe -m pip show mcp
```

Se il pacchetto manca, ripeti l'installazione di `requirements.txt` con quel Python. Installare MCP in un altro ambiente non lo rende disponibile automaticamente in questo.

Il server resta silenzioso

Un server `stdio` in attesa di messaggi può restare silenzioso. Questo comportamento è diverso da una prova che dovrebbe stampare 120 e terminare. Apri un'altra finestra per i comandi o usa l'Inspector come spiegato. Interrompi il processo con `Ctrl+C` quando vuoi chiuderlo.

Il tool non appare dopo una modifica

Verifica quale file viene avviato: `server_primo.py`, `server.py` e `server_esteso.py` offrono strumenti diversi. Poi riavvia la connessione. Il processo deve caricare il nuovo programma; un file salvato sul disco non sostituisce il codice di un processo già avviato.

Il tool appare ma fallisce

Leggi prima il messaggio restituito. Un valore fuori limite richiede di correggere gli argomenti. Un percorso del CLI errato richiede di correggere la configurazione. Un XML assente richiede la diagnosi dell'esecuzione Dynamo. Aumentare il tempo massimo non corregge un percorso inesistente.

Il grafo esiste ma il risultato non coincide

Apri esattamente il file indicato in `file_dyn`. Controlla i tre slider. Verifica di non avere aperto il template o una variante precedente. Confronta poi l'anteprima e la descrizione del Cuboid. Se hai modificato la struttura del template, ricontrolla gli identificatori nel manifesto.

La chat dichiara successo dopo un errore

Chiedi il risultato della chiamata e il percorso prodotto. Leggi l'errore originale. Una risposta dell'assistente è un'interpretazione dei dati disponibili; può omettere un dettaglio o descrivere il comportamento desiderato. Per il laboratorio consideriamo riuscita l'operazione solo quando la risposta del tool e il file apribile concordano.

La scheda di errore che aiuta davvero

Annota cinque cose: comando o tool invocato; argomenti; errore completo; versioni; ultimo test riuscito. Un messaggio come "non funziona" non distingue un problema Python da uno geometrico. "volume_teorico restituisce 120, ma crea_geometria segnala XML assente" indica già il punto da controllare.

LEZIONE 14

Progetta il tuo prossimo comando

OBIETTIVO

Passare da un esercizio guidato a un'operazione utile del tuo lavoro in Dynamo.

Scegli un grafo che sai eseguire con valori noti. Per la prima estensione preferisci un risultato piccolo: una famiglia di volumi, una misura, una tabella o una variante parametrica. Il fatto che un grafo sia interessante non significa che debba esporre tutti i suoi slider all'assistente.

Un esempio: il pilastro parametrico

Immagina un tool chiamato `crea_pilastro` con due input: `lato_base` e `altezza`. Per una base quadrata, larghezza e profondità sono uguali al lato. Con 2 e 6 il volume teorico è $2 \times 2 \times 6 = 24$.

Hai già gli elementi per scrivere questa estensione sul modello di `crea_cubo`: cambia nome, descrizione e parametri, poi chiama la funzione esistente con i valori corretti. Questa è una consegna da svolgere; il kit non dichiara il tool del pilastro come già installato.

Prova minima. Usa 2 e 6, controlla i tre valori del grafo e confronta 24. **Prova di modifica.** Cambia soltanto l'altezza a 3: il volume teorico deve diventare 12. **Prova di rifiuto.** Usa un'altezza pari a zero: il ponte deve segnalare un errore di input.

Una soluzione da leggere dopo il tentativo

Apri `server_esteso.py`. Aggiungi questa funzione prima del blocco finale che avvia il server. Il file importa già l'oggetto MCP e la funzione geometrica. Il listato è un'estensione: da solo non è un programma completo.

```
from typing import Any

@mcp.tool()
def crea_pilastro(
    lato_base: float,
    altezza: float,
) -> dict[str, Any]:
    """Crea un pilastro con base quadrata."""
    return crea_geometria(
        larghezza=lato_base,
        profondita=lato_base,
        altezza=altezza,
    )
```

L'annotazione `dict[str, Any]` mantiene lo schema esplicito della risposta. Se `Any` è già importato nel file, conserva una sola importazione all'inizio. Dopo la modifica salva, riavvia e aggiorna la discovery nell'Inspector.

Se cambi davvero il tipo di geometria

Una sfera non è un parallelepipedo con numeri diversi. Occorre un template geometrico nuovo, una mappa dei parametri adatta e una lettura del risultato che riconosca la nuova forma. Il controllo finale del ponte attuale cerca un `Cuboid`: riutilizzarlo senza modifiche per una sfera produrrebbe una diagnosi sbagliata.

Scrivi quindi tre prove prima di modificare il server: un caso valido con risultato atteso, una variazione di un input e un valore non ammesso. Poi costruisci il grafo, registra gli identificatori corretti e implementa la funzione. L'Inspector ti permette di verificare l'operazione prima che intervenga la scelta dell'assistente.

Se il tuo grafo usa Revit

I nodi Revit richiedono il contesto del documento e dell'applicazione. Un processo Python esterno che esegue DynamoCLI non eredita quel contesto. Il kit conserva un esempio facoltativo di lettura: Dynamo avviato in Revit esporta un riepilogo JSON; il tool legge quel riepilogo e la data di generazione.

Questo esempio è una fotografia dei dati all'ultima esecuzione manuale. Per un'integrazione che modifichi un modello Revit servono un'architettura interna all'applicazione e il rispetto del suo modello di esecuzione. Affronta quella progettazione dopo aver reso stabile il percorso Core di questo libro.

La domanda finale prima di pubblicare un tool

Una persona che non ha scritto il grafo riesce a capire cosa fa il comando, quali valori accetta e dove trova il risultato? Se la risposta richiede una lunga conversazione ogni volta, rendi più precisi il nome, la descrizione e i campi restituiti.

La tua scheda di progetto

Nome del tool

Operazione che voglio rendere disponibile

Input, significato e unità

Limiti ammessi

Risultato che il tool restituisce

Caso minimo con valori e risposta attesa

Come verifico il risultato in Dynamo

Un caso da rifiutare

Conserva questa scheda insieme al grafo, alla configurazione e a una prova riuscita.

Il registro delle prove

Compila questa pagina quando estendi il server. Scrivi il risultato osservato, anche se è diverso da quello atteso. Conserva il percorso del file: ti servirà per ripetere la verifica.

Data e nome del tool

Versioni: Python, MCP, DynamoCLI

01 Il caso valido

Per il cubo: lato 4; tre dimensioni pari a 4; volume teorico 64; un nuovo file apribile. **Input e risultato osservato**

Percorso del file

02 La variazione controllata

Per il cubo: cambia il lato da 4 a 2; volume teorico 8. Il primo file deve restare disponibile. **Input e risultato osservato**

03 Il caso da rifiutare

Per il cubo: lato 0; errore di validazione. Un errore previsto è una prova riuscita del controllo. **Input e messaggio ricevuto**

Che cosa correggo prima della prossima prova

Glossario da tenere accanto

AI / IA. Abbreviazioni inglese e italiana di intelligenza artificiale. Nel libro indicano soprattutto il modello linguistico usato dall'assistente.

Argomento. Un valore passato a una funzione: in `crea_cubo(4)`, il numero 4 è l'argomento.

CLI. Interfaccia a comandi scritti. DynamoCLI valuta il grafo senza aprire il canvas.

Client MCP. Componente che mantiene il dialogo con un server per conto dell'host.

Contesto. Informazioni disponibili al modello durante una richiesta: messaggi, istruzioni e dati forniti dall'applicazione.

Decoratore. Istruzione Python posta sopra una funzione. Nel nostro caso `@mcp.tool()` registra la funzione come tool.

Discovery. Scoperta degli strumenti disponibili: il client richiede l'elenco e le loro descrizioni.

Funzione. Blocco di codice che riceve dati, esegue operazioni e può restituire un risultato.

GUID. Identificatore usato per distinguere un nodo o un oggetto. Non coincide con il nome visibile.

Host. Applicazione che coordina conversazione, modello e collegamenti agli strumenti.

JSON. Formato testuale di dati con oggetti, elenchi, stringhe e numeri. Nel kit configura il motore e descrive i grafi.

LLM. Large Language Model: modello che elabora sequenze di token e genera risposte in base al contesto.

Manifesto. Nel kit, elenco esplicito di grafi e identificatori. Dice al server quali file usare e quale output leggere.

MCP. Model Context Protocol: protocollo con cui client e server scambiano strumenti, dati e risultati.

Pacchetto Python. Libreria installabile. Il pacchetto MCP implementa parti del protocollo che altrimenti dovresti scrivere.

Parametro. Nome di un input previsto da una funzione. `lato` è il parametro; `4` è un possibile argomento.

Percorso assoluto. Indirizzo completo di un file, indipendente dalla cartella corrente del terminale.

Prompt. Richiesta o istruzione in linguaggio naturale. In MCP, un prompt può anche essere un modello riutilizzabile esposto da un server.

SDK. Software Development Kit: insieme di librerie e strumenti che semplificano la creazione di software.

Server MCP. Programma che espone capacità attraverso MCP. Nel laboratorio è un processo Python locale.

Schema. Descrizione della forma dei dati: nomi, tipi e struttura attesa. Aiuta il client a invocare un tool.

stdio. Trasporto che usa ingresso e uscita standard del processo. Il client avvia il server e comunica su questi flussi.

Template. Modello da copiare. Il nostro template Dynamo conserva la logica del solido; i parametri cambiano nelle copie.

Timeout. Tempo massimo di attesa. Superarlo interrompe una prova; non spiega da solo la causa della lentezza.

Token. Unità in cui il modello rappresenta testo o altri dati. Non corrisponde sempre a una parola intera.

TOML. Formato di configurazione usato nell'esempio dell'host. Le sue regole per stringhe e sezioni sono diverse da JSON.

Tool. Operazione invocabile descritta da nome, input e risultato. Qui può calcolare un volume o generare un grafo.

venv. Ambiente Python separato per il progetto. Contiene il suo interprete e i suoi pacchetti.

XML. Formato testuale a etichette. DynamoCLI lo usa nel laboratorio per registrare le valutazioni dei nodi.

Ripasso: sai spiegare il percorso?

- 1. Qual è la differenza fra il modello e il server?** Il modello interpreta il contesto e può proporre una chiamata. Il server esegue la funzione che hai programmato quando riceve quella chiamata attraverso il client.
- 2. Perché il primo tool calcola un numero?** Consente di verificare Python, registrazione del tool e protocollo con un risultato semplice. La creazione di geometria introduce un secondo programma e altre dipendenze.
- 3. Devo installare MCP dentro ogni nodo Dynamo?** Nel percorso del libro il server è esterno. I grafi contengono la logica; Python li prepara e DynamoCLI li esegue. Questa è l'architettura scelta, non l'unica integrazione possibile.
- 4. Perché la chat non deve inventare il percorso?** Il nome del file viene prodotto dal server. Un percorso plausibile scritto dall'assistente non dimostra che il file esista.
- 5. Se cambia il prompt, cambia anche il codice?** Il tool del laboratorio applica nuovi valori a una struttura nota. La richiesta può cambiare i parametri; una nuova regola geometrica richiede una modifica esplicita dell'implementazione.
- 6. Che cosa prova il volume teorico?** Prova il calcolo dei parametri dichiarati. Per verificare la geometria apri il grafo, controlla le connessioni e confronta le dimensioni o una misura geometrica calcolata nel grafo.
- 7. Un nuovo tool compare appena salvo?** Il processo deve ricaricare il file. Riavvia il collegamento e aggiorna l'elenco degli strumenti.
- 8. Quando ho finito il mio primo progetto?** Quando sai ripetere un caso valido, spiegare un caso rifiutato e ritrovare il file prodotto. Conserva anche versioni e configurazione: rendono la prova riproducibile.

Prove e fonti dell'edizione

Che cosa è stato verificato

Nella preparazione di questa edizione sono state eseguite la funzione aritmetica, la discovery del server minimo tramite client SDK in memoria, una chiamata con 8, 5, 3 e le prove di rifiuto dei valori non ammessi. Il tool del cubo è stato eseguito con il DynamoCLI disponibile: ha restituito un Cuboid con tre dimensioni pari a 4 e ha prodotto un nuovo grafo.

Il file `kit/verifiche.json` registra i controlli e le versioni. Il client in memoria verifica il comportamento del server e dell'SDK. Non equivale a una prova dell'interfaccia web dell'Inspector o di ogni host commerciale. Le loro istruzioni di collegamento sono state confrontate con la documentazione ufficiale.

L'illustrazione di un flusso spiega come funziona; un'uscita riportata come esempio indica che cosa confrontare. Il libro distingue questi materiali dalle prove locali realmente eseguite.

Fonti ufficiali

Documentazione consultata il 23 settembre 2026. Versioni e interfacce possono evolvere; conserva il pin del kit quando vuoi ripetere gli esempi di questa edizione.

MCP: architettura e concetti.

modelcontextprotocol.io: [Architecture overview](#). Host, client, server e livelli del protocollo.

SDK Python MCP.

[Repository ufficiale python-sdk](#); [documentazione Get started](#); [pacchetto MCP 2.2.0 su PyPI](#). API utilizzata nei programmi del kit.

MCP Inspector.

[Documentazione ufficiale dell'Inspector](#). Avvio con un comando, trasporto e requisiti Node.

Python.

[Download ufficiali per Windows](#); [documentazione venv](#). Interpreti e ambienti del progetto.

Node.js.

[Download ufficiali Node.js](#). Installatore LTS e npm per avviare l'Inspector.

DynamoCLI.

[DynamoDS: Command Line Interface](#). Esecuzione del grafo e opzioni di output.

Dynamo: setup e geometria.

[Dynamo Primer: setup](#); [Geometric primitives](#). Dipendenze e costruttori geometrici.

Codex: collegamento MCP.

[OpenAI Docs: Model Context Protocol](#); [Configuration reference](#). Campi di avvio, cartella e timeout.

Revit: contesto di esecuzione.

[Autodesk: External Events](#). Approfondimento per future integrazioni nell'applicazione.

DALLA FRASE ALLA FORMA

Una richiesta chiara. Un tool comprensibile.

Un risultato che puoi verificare.