



Ubaldo impara a leggere la dispensa

Il secondo agente con ADK: strumenti e dati

Nella prima guida abbiamo creato Ubaldo, uno chef burbero che propone ricette a partire dagli ingredienti indicati dall'utente. In questa seconda versione gli daremo due strumenti, chiamati **tool**, per leggere la dispensa da un file e aggiornarla dopo aver cucinato.

Un tool è una funzione che l'agente può richiedere di eseguire. Il modello decide quando utilizzarla e con quali parametri; il programma esegue l'operazione e restituisce il risultato.

Ubaldo potrà quindi consultare gli ingredienti disponibili, proporre una ricetta e registrare le quantità consumate dopo la conferma dell'utente.

Questa guida prosegue dalla precedente: useremo la stessa cartella `adk-svuota-frigo`, lo stesso ambiente Python e un nuovo progetto, `step2_dispensa`. La prima versione resterà disponibile.

1. Apri il terminale e attiva l'ambiente Python

Su Windows, apri il menu Start, cerca **Ubuntu** e avvialo. Su Linux o macOS, apri il terminale che usi normalmente.

Entra nella cartella principale della serie:

```
cd ~/adk-svuota-frigo
```

Attiva l'ambiente virtuale creato nella prima guida:

```
source .venv/bin/activate
```

Questo passaggio va ripetuto ogni volta che apri un nuovo terminale. L'ambiente virtuale rimane sul disco, ma la sua attivazione vale soltanto per il terminale in cui hai eseguito il comando.

Se l'attivazione è riuscita, all'inizio della riga vedrai `(.venv)`. Non devi ricreare l'ambiente né reinstallare ADK.

2. Crea il secondo progetto ADK

Dalla cartella `adk-svuota-frigo`, con l'ambiente virtuale attivo, esegui:

```
adk create step2_dispensa
```

Il comando crea una nuova cartella con i file iniziali dell'agente.

Come nella prima guida, scegli `gemini-3.8-flash`:

```
Choose a model for the root agent:
1. gemini-3.5-flash
2. gemini-3.8-flash
3. Other models (fill later)
Choose model (1, 2, 3): 2
```

Poi scegli **Google AI**, inserendo 1:

```
1. Google AI
2. Vertex AI
3. Login with Google
Choose a backend (1, 2, 3): 1
```

Quando richiesto, inserisci la chiave API creata in Google AI Studio durante la prima guida. Puoi riutilizzare la stessa chiave.

Entra nella nuova cartella e aprila con VSCode:

```
cd step2_dispensa
codium .
```



Lavoreremo su tre file:

- dispensa.json: contiene ingredienti, quantità e unità di misura.
- tools.py: contiene le funzioni per leggere e aggiornare la dispensa.
- agent.py: definisce Ubaldo, le sue istruzioni e gli strumenti disponibili.

3. Crea il file della dispensa

In VSCode, crea un file chiamato `dispensa.json`, nella stessa cartella di `agent.py`.

Inserisci questo contenuto e salva:

```
{
  "uova": {
    "quantita": 4,
    "unita": "pezzi"
  },
  "zucchine": {
    "quantita": 300,
    "unita": "g"
  },
  "ceci cotti": {
    "quantita": 240,
    "unita": "g"
  },
  "rosmarino": {
    "quantita": 2,
    "unita": "rametti"
  }
}
```

Il formato JSON permette di rappresentare dati strutturati in un file di testo. Per ogni ingrediente abbiamo indicato una quantità e la relativa unità di misura.

Le unità sono importanti: consumare 150 di zucchine significa consumarne **150 grammi**, mentre consumare 2 di uova significa consumarne **due pezzi**.

Abbiamo specificato **ceci cotti** perché quelli secchi richiederebbero tempi incompatibili con i 15 minuti concessi a Ubaldo.

4. Crea gli strumenti di Ubaldo

Nella stessa cartella, crea il file `tools.py` e inserisci questo codice:

```
import json
import math
from pathlib import Path

DISPENZA_PATH = Path(__file__).parent / "dispensa.json"

def leggi_dispensa() -> dict:
    """Legge gli ingredienti disponibili con quantità e unità di misura.
    Returns:
        Esito dell'operazione e contenuto della dispensa, oppure un errore.
    """
    try:
        dispensa = json.loads(
            DISPENZA_PATH.read_text(encoding="utf-8")
        )
    except (OSError, ValueError):
        return {
            "stato": "errore",
            "messaggio": "Non riesco a leggere il file della dispensa."
        }
    return {
        "stato": "ok",
        "ingredienti": dispensa
    }

def registra_consumo(consumi: dict[str, float]) -> dict:
    """Sottrae dalla dispensa le quantità effettivamente consumate.
```



Da usare soltanto dopo che l'utente ha confermato gli ingredienti e le quantità utilizzati. Non chiamare per una semplice proposta di ricetta.

Args:

consumi: Mappa tra nomi esatti degli ingredienti e quantità consumate. Usa le unità indicate nella dispensa.
Esempio: {"uova": 2, "zucchine": 150}.

Returns:

Esito e dispensa aggiornata, oppure un errore senza modifiche.

```
"""
risultato = leggi_dispensa()
if risultato["stato"] != "ok":
    return risultato
dispensa = risultato["ingredienti"]
if not consumi:
    return {
        "stato": "errore",
        "messaggio": "Non hai indicato ingredienti da aggiornare."
    }
# Prima controlliamo TUTTI gli ingredienti.
for ingrediente, quantita in consumi.items():
    if ingrediente not in dispensa:
        return {
            "stato": "errore",
            "messaggio": f"Ingrediente non presente: {ingrediente}."
        }
    if (
        isinstance(quantita, bool)
        or not isinstance(quantita, (int, float))
        or not math.isfinite(quantita)
        or quantita <= 0
    ):
        return {
            "stato": "errore",
            "messaggio": (
                f"La quantità di {ingrediente} deve essere "
                "un numero positivo e finito."
            )
        }
    scorta = dispensa[ingrediente]
    if scorta["unita"] in ("pezzi", "rametti"):
        if not float(quantita).is_integer():
            return {
                "stato": "errore",
                "messaggio": (
                    f"Per {ingrediente} indica un numero intero "
                    f"di {scorta['unita']}."
                )
            }
    if quantita > scorta["quantita"]:
        return {
            "stato": "errore",
            "messaggio": (
                f"Quantità insufficiente di {ingrediente}: "
                f"disponibili {scorta['quantita']} "
                f"{scorta['unita']}."
            )
        }
}

# Aggiorniamo soltanto dopo aver superato tutti i controlli.
for ingrediente, quantita in consumi.items():
    dispensa[ingrediente]["quantita"] -= quantita
try:
    temporaneo = DISPENSA_PATH.with_suffix(".tmp")
    temporaneo.write_text(
        json.dumps(dispensa, ensure_ascii=False, indent=2),
        encoding="utf-8"
    )
    temporaneo.replace(DISPENSA_PATH)
```



```
except OSError:
    return {
        "stato": "errore",
        "messaggio": "Non sono riuscito a salvare la dispensa."
    }
return {
    "stato": "ok",
    "messaggio": "Consumo registrato.",
    "ingredienti": dispensa
}
```

Come funzionano le due funzioni

leggi_dispensa apre il file JSON e restituisce gli ingredienti. Se non riesce a leggerlo, restituisce un errore. registra_consumo riceve gli ingredienti consumati con le rispettive quantità. Per esempio:

```
{"uova": 2, "zucchine": 150}
```

Prima di modificare il file, verifica che:

- gli ingredienti siano presenti;
- le quantità siano numeri positivi e finiti;
- gli ingredienti misurati in pezzi o rametti abbiano quantità intere;
- le quantità richieste non superino quelle disponibili.

Solo se tutti i controlli passano, sottrae le quantità e salva la dispensa. Scrive prima un file temporaneo, poi lo sostituisce al precedente.

Path(__file__).parent indica la cartella che contiene tools.py: la dispensa viene cercata lì, indipendentemente dalla cartella da cui avvii ADK.

Le descrizioni tra triple virgolette, chiamate **docstring**, spiegano lo scopo delle funzioni e dei loro parametri. ADK le usa per descrivere gli strumenti al modello.

5. Inserisci il codice dell'agente

Apri agent.py, sostituisci il contenuto con questo codice e salva:

```
from google.adk.agents import Agent; from google.genai import types as t
from .tools import leggi_dispensa, registra_consumo

root_agent = Agent(
    name="ubaldo_chef", model="gemini-3.8-flash",
    generate_content_config=t.GenerateContentConfig(thinking_config=dict(thinking_level="low")),
    description=(
        "Ubaldo, uno chef burbero ma dal cuore d'oro, pragmatico "
        "e allergico agli sprechi, che inventa ricette lampo "
        "con quello che si trova in frigo."
    ),
    instruction="""
Ti chiami Ubaldo. Sei un cuoco di vecchia scuola, pragmatico,
schietto e intollerante agli sprechi.

Non fai complimenti inutili, parli in modo diretto e un po' ruvido,
ma alla fine aiuti sempre chi è in difficoltà davanti al frigo vuoto.

REGOLE PER LE RICETTE:
1. Prima di proporre una ricetta, usa leggi_dispensa per conoscere
   ingredienti, quantità e unità di misura disponibili.
2. Usa solo gli ingredienti disponibili nella dispensa.
   Gli unici extra concessi sono acqua, sale, pepe e olio
   extravergine d'oliva.
3. Non superare le quantità disponibili.
4. Tempo massimo di preparazione e cottura: 15 minuti.
5. Pentolame massimo: una sola padella o pentola e un tagliere.
6. Se l'utente propone cose non commestibili, riprendilo
   e non includerle nella ricetta.
7. Se non puoi rispettare i vincoli, spiegalo senza inventare
   ingredienti, quantità o tempi di cottura.
```



GESTIONE DELLA DISPENSA:

- Per conoscere le scorte, usa sempre `leggi_dispensa`: non affidarti soltanto ai messaggi precedenti.
- Proporre o accettare una ricetta NON significa averla cucinata.
- Non modificare la dispensa durante la proposta di una ricetta.
- Quando l'utente dice di aver cucinato, riepiloga gli ingredienti e le quantità da sottrarre e chiedi conferma.
- Solo dopo la conferma esplicita del riepilogo, chiama `registra_consumo` con tutti gli ingredienti consumati insieme.
- Usa i nomi esatti e le unità presenti nella dispensa.
- Non registrare nuovamente un consumo già registrato.
- Dichiarare che l'aggiornamento è riuscito solo se il tool restituisce stato "ok".
- Se un tool restituisce un errore, spiegalo e chiedi il chiarimento necessario. Non fingere che sia riuscito.

FORMATO DELLE RICETTE:

- 🍴 NOME DEL PIATTO: un nome onesto ma rustico.
- ⌚ TEMPO E DIFFICOLTÀ: stima realistica in minuti.
- 🥘 INGREDIENTI: quantità esatte per ogni ingrediente.
- / PIANO D'AZIONE CRONOLOGICO: passaggi numerati e concreti.
- 🗨 IL CONSIGLIO DI UBALDO: una dritta pratica.

Per chiarimenti, conferme e informazioni sulla dispensa, rispondi brevemente senza utilizzare la scheda della ricetta.

```
"""  
    tools=[  
        leggi_dispensa,  
        registra_consumo,  
    ],  
)
```

La novità principale è la lista `tools`, con cui mettiamo le due funzioni a disposizione dell'agente.

Abbiamo anche modificato le istruzioni: Ubaldo deve leggere le scorte prima di proporre una ricetta e chiedere conferma prima di registrare un consumo.

La scheda della ricetta ora include le quantità degli ingredienti. Per domande e conferme, invece, Ubaldo può rispondere normalmente senza ripetere tutta la scheda.

Prima di proseguire, salva tutti e tre i file in VSCode.

6. Avvia l'interfaccia web

Nel terminale, torna nella cartella principale della serie:

```
cd ~/adk-svuota-frigo
```

Se hai chiuso il terminale e ne hai aperto un altro, riattiva l'ambiente virtuale:

```
source .venv/bin/activate
```

Avvia ADK:

```
adk web
```

Apri nel browser l'indirizzo indicato dal terminale, di solito:

```
http://localhost:8000
```

Seleziona `step2_dispensa` e apri una nuova conversazione.

7. Prova una conversazione completa

Per questa prova, parti dalle quantità iniziali del file: quattro uova, 300 grammi di zucchine, 240 grammi di ceci cotti e due rametti di rosmarino.

Invia i messaggi seguenti **uno alla volta, nella stessa conversazione**. Le parole di Ubaldo possono cambiare: ciò che ci interessa verificare è il suo comportamento.



Prima richiesta: leggere la dispensa

Cosa ho nella dispensa?

Ubaldo deve utilizzare `leggi_dispensa` e riportare ingredienti e quantità presenti nel file.

Seconda richiesta: proporre una ricetta

Proponimi una ricetta con due uova e 150 grammi di zucchine.

Ubaldo deve proporre una ricetta compatibile con le scorte e con i suoi vincoli.

Apri `dispensa.json` in VSCode: le quantità devono essere ancora quelle iniziali. Una proposta non deve modificare la dispensa.

Terza richiesta: cambiare idea

Lascia perdere, stasera ordino una pizza.

Ubaldo non deve registrare alcun consumo. Nel file devono esserci ancora quattro uova e 300 grammi di zucchine.

Quarta richiesta: comunicare il consumo

Alla fine ho cucinato: ho usato due uova e 150 grammi di zucchine.

Ubaldo deve riepilogare gli ingredienti e le quantità da sottrarre, poi chiedere conferma. La dispensa deve rimanere invariata fino alla conferma.

Quinta richiesta: confermare l'aggiornamento

Confermo, aggiorna la dispensa.

Ubaldo deve chiamare `registra_consumo` e comunicare il risultato.

Controlla il file: devono rimanere **due uova e 150 grammi di zucchine**. Ceci e rosmarino devono essere invariati.

Sesta richiesta: controllare le scorte aggiornate

Cosa è rimasto nella dispensa?

Ubaldo deve leggere nuovamente il file e riportare le quantità aggiornate.

Se compare un errore **429** o **503**, come nella prima guida, aspetta qualche secondo e riprova. Se il limite di utilizzo persiste, potrebbe essere necessario attendere più a lungo.

Dopo un errore durante un aggiornamento, chiedi prima di leggere la dispensa: controlla se il consumo è già stato registrato prima di confermarlo nuovamente.

8. Che cosa abbiamo costruito e quali sono i limiti

Ubaldo ora può consultare dati esterni alla conversazione e richiederne la modifica attraverso funzioni Python. Le quantità vengono controllate e aggiornate dal codice.

La richiesta di conferma, invece, è ancora affidata alle istruzioni del modello: **non è un blocco imposto dall'applicazione**. Anche l'indicazione di non registrare due volte lo stesso consumo è un'istruzione, non una garanzia tecnica.

Questo esempio usa un file locale ed è pensato per una persona alla volta. Inoltre, il file deve mantenere la struttura mostrata nella guida: il codice gestisce alcuni errori di lettura e le quantità di consumo, ma non verifica ogni possibile modifica manuale alla struttura della dispensa.

Per ripetere il test dall'inizio, ripristina il contenuto originale di `dispensa.json`, salva e apri una nuova conversazione. **Una nuova conversazione, da sola, non ripristina le scorte**: il file conserva gli aggiornamenti.

Nella prima puntata abbiamo definito il comportamento di Ubaldo. In questa gli abbiamo dato due strumenti per leggere e modificare dati reali. Abbiamo anche iniziato a distinguere ciò che chiediamo al modello attraverso le istruzioni da ciò che controlliamo direttamente nel programma.