Table of Contents:

Maps API

Unset

```
"""API for google_maps"""

import dataclasses
from typing import Union, Dict


@dataclasses.dataclass
class DirectionsSummary:
 """Directions summary.

 Attributes:
  mapUrl: Map url.
  routes: Routes.
  travelMode: Travel mode.
 """

 mapUrl: str | None = None
 routes: Union[list["Route"], None] = None
 travelMode: str | None = None


@dataclasses.dataclass
class Place:
 """Place.

 Attributes:
  address: Address.
  description: Description.
  map_url: Map url.
  name: Name.
  opening_hours: Opening hours.
  rating: Rating.
  review_count: Review count.
  url: Url.
  user_rating_count: User rating count.
 """

 address: str | None = None
 description: str | None = None
 map_url: str | None = None
 name: str | None = None
 opening_hours: list[str] | None = None
 rating: str | None = None
```

```
 review_count: int | None = None
 url: str | None = None
 user_rating_count: int | None = None


@dataclasses.dataclass
class Route:
 """Route.

 Attributes:
  distance: Distance.
  duration: Duration.
  endAddress: End address.
  mode: Mode.
  startAddress: Start address.
  steps: Steps.
  summary: Summary.
  url: Url.
 """

 distance: str | None = None
 duration: str | None = None
 endAddress: str | None = None
 mode: str | None = None
 startAddress: str | None = None
 steps: list[str] | None = None
 summary: str | None = None
 url: str | None = None


@dataclasses.dataclass
class SummaryPlaces:
 """Summary places.

 Attributes:
  map_url: Map url.
  places: Places.
  query: Query.
 """

 map_url: str | None = None
 places: Union[list["Place"], None] = None
 query: str | None = None
```

```
def find_directions(
  destination: str,
  origin: str | None = None,
  travel_mode: str | None = None,
  waypoints: list[str] | None = None,
  avoid: list[str] | None = None,
  origin_location_bias: str | None = None,
  destination_location_bias: str | None = None,
) -> DirectionsSummary | str:
 """find directions between two places.

 Args:
  destination: The destination place.
  origin: The origin place. It can be the empty if the origin is not found in the user
query. It also can be empty if the origin is user current location.
  travel_mode: The travel mode to use. Supported values are driving, walking,
bicycling, transit, bus, rail, subway, train, and tram. Transit means public
transportation.
  waypoints: A list of locations to pass through or stop over at, e.g. ["Googleplex,
Mountain View, CA", "Computer History Museum, Mountain View, CA"]
  avoid: A list of features to avoid for the routes. Supports tolls, highways, ferries,
and indoor, e.g. ["tolls", "highways", "ferries"].
  origin_location_bias: If set, maps will return the origin that is more relevant to
the location specified.
  destination_location_bias: If set, maps will return the destination that is more
relevant to the location specified.
 """

 ...


def navigate(
  destination: str,
  travel_mode: str | None = None,
  waypoints: list[str] | None = None,
  avoid: list[str] | None = None,
  origin_location_bias: str | None = None,
  destination_location_bias: str | None = None,
) -> DirectionsSummary | str:
 """Navigate user from current location to a destination.

 Args:
  destination: The destination place.
```

```
  travel_mode: The travel mode to use. Supported values are driving, walking,
bicycling, transit, bus, rail, subway, train, and tram. Transit means public
transportation.
  waypoints: A list of locations to pass through or stop over at, e.g. ["Googleplex,
Mountain View, CA", "Computer History Museum, Mountain View, CA"]
  avoid: A list of features to avoid for the routes. Supports tolls, highways, ferries,
and indoor, e.g. ["tolls", "highways", "ferries"].
  origin_location_bias: If set, maps will return the origin that is more relevant to
the location specified.
  destination_location_bias: If set, maps will return the destination that is more
relevant to the location specified.
 """

 ...


def query_places(
  query: list[str],
  location_bias: str | None = None,
) -> SummaryPlaces | str:
 """API used to locate places or geographical entities, including business, stores,
restaurants, parks, attractions, cities or countries and show them on a map. The API
outputs information about one or more places like the place's name, description,
address, website, and rating. The API can display places or geographical entities on a
map. The API can take multiple queries. The API CANNOT handle generic google search
queries or image search queries.

 Args:
  query: Query describing the desired characteristics of business, stores,
restaurants, parks, attractions or geographical entities to locate.
  location_bias: If set, maps will return results that are more relevant to the
location specified.
 """

 ...


# These constants should be used as argument as-is and not meant to be used as part of a
another string by concatenation or other means.
MY_HOME='MY_HOME'
MY_WORK='MY_WORK'
MY_LOCATION='MY_LOCATION'
```

Search API

```
"""API for google_search"""

import dataclasses
from typing import Union, Dict


@dataclasses.dataclass
class SearchResult:
 """Search result.

 Attributes:
  snippet: Snippet.
  source_title: Source title.
  url: Url.
 """

 snippet: str | None = None
 source_title: str | None = None
 url: str | None = None


def search(
  query: str,
) -> list[SearchResult]:
 """Search Google.

 Args:
  query: The search query string.
 """

 ...
```

Hotels API

Unset

```
"""API for google_hotels"""

import dataclasses
from typing import Union, Dict
from enum import Enum


@dataclasses.dataclass
class Hotel:
 """Hotel.

 Attributes:
   hotel_name: Hotel name.
   dealness_magnitude_tier: Dealness magnitude tier.
   dealness_tip_message: Dealness tip message.
   description: Description.
   display_price: Display price.
   hotel_class: Hotel class.
   image_anchor: Image anchor.
   image_list: Image list.
   review_count: Review count.
   review_rating: Review rating.
   review_rating_float: Review rating float.
   url: Url.
 """
hotel_name: str;

 dealness_magnitude_tier: int | None = None
 dealness_tip_message: str | None = None
 description: str | None = None
 display_price: str | None = None
 hotel_class: str | None = None
 image_anchor: str | None = None
 image_list: list[str] | None = None
 review_count: int | None = None
 review_rating: int | None = None
```

```
 review_rating_float: float | None = None
 url: str | None = None


class AccommodationType(str, Enum):
BEACH_HOTEL = "beach_hotel";

BED_AND_BREAKFAST = "bed_and_breakfast";

BUNGALOW = "bungalow";

CABIN = "cabin";

COTTAGE = "cottage";

HOSTEL = "hostel";

INN = "inn";

IS_APARTMENT = "is_apartment";

IS_HOLIDAY_VILLAGE = "is_holiday_village";

IS_HOUSE = "is_house";

IS_HOUSEBOAT = "is_houseboat";

IS_SPA_HOTEL = "is_spa_hotel";

IS_VILLA = "is_villa";

MOTEL = "motel";

RESORT = "resort";



class Amenity(str, Enum):
AIR_CONDITIONED = "air_conditioned";
```

```
ALL_INCLUSIVE_AVAILABLE = "all_inclusive_available";

BEACH_ACCESS = "beach_access";

FITNESS_CENTER = "fitness_center";

FREE_BREAKFAST = "free_breakfast";

FREE_PARKING = "free_parking";

FREE_WIFI = "free_wifi";

HAS_BAR = "has_bar";

HAS_PARKING = "has_parking";

HAS_RESTAURANT = "has_restaurant";

HAS_SPA = "has_spa";

INDOOR_POOL = "indoor_pool";

KID_FRIENDLY = "kid_friendly";

OUTDOOR_POOL = "outdoor_pool";

PET_FRIENDLY = "pet_friendly";

POOL = "pool";

ROOM_SERVICE = "room_service";

WHEELCHAIR_ACCESSIBLE = "wheelchair_accessible";



def search_hotels(
   query: str,
   check_in_date: str | None = None,
   check_out_date: str | None = None,
```

```
    max_check_in_date: str | None = None,
    length_of_stay: int | None = None,
    adults: int | None = None,
    children: int | None = None,
    children_age_years: list[int] | None = None,
    cheapest: bool | None = None,
    currency: str | None = None,
    max_price: int | None = None,
    min_price: int | None = None,
    amenities: list[Amenity] | None = None,
    accommodation_type: AccommodationType | None = None,
    hotel_class: list[int] | None = None,
    min_user_rating: float | None = None,
) -> list[Hotel] | str:
 """Search or book hotels.

 Args:
    query: The user query. This can include any additional parameters mentioned
(e.g. location, price, star rating)
    check_in_date: The date to check in in YYYY-MM-DD format.
    check_out_date: The date to check out in YYYY-MM-DD format.
    max_check_in_date: The latest date to check in in YYYY-MM-DD format.
    length_of_stay: Length of stay in days from the arrival_date.
    adults: Occupancy of adults. Supports a maximum occupancy of 6, so adults +
children should be <= 6.
    children: Occupancy of children. Supports a maximum occupancy of 6, so
adults + children should be <= 6.
    children_age_years: Age of years for children.
    cheapest: If true, sorts the results by price from low to high.
    currency: If present, returns the hotel prices in that currency. Otherwise,
the user's localized currency is returned.
    max_price: Maximum price of the hotels.
    min_price: Minimum price of the hotels.
    amenities: List of hotel amenities to filter by. If multiple amenities are
present in the list, they are applied conjunctively, as a logical "and".

    accommodation_type: Type of accommodation.
```

```
   hotel_class: The type of hotel class to filter by. Classes are a star rating
on a 0 to 5 scale. If multiple hotel classes are present in the list, they are
applied conjunctively, acting as a logical "or".
The default integer value of 0 corresponds to an unspecified hotel class, so
hotels with any star rating could be returned.

   min_user_rating: When set, only finds hotels having this rating or above.

"""


...
```

Flights API

Unset

```
```tool_code
print(extensions.describe(google_flights))
```
```tool_outputs
"""API for google_flights"""

import dataclasses
from typing import Union, Dict
from enum import Enum


@dataclasses.dataclass
class CarrierRow:
 """Carrier row.

 Attributes:
   arrival_airport: Arrival airport.
   carrier: Carrier.
   carrier_image: Carrier image.
   departure_airport: Departure airport.
```
```

```
    duration: Duration.
    non_stop: Non stop.
    price: Price.
    train: Train.
    unpriced_reason: Unpriced reason.
  """

  arrival_airport: str | None = None
  carrier: str | None = None
  carrier_image: str | None = None
  departure_airport: str | None = None
  duration: str | None = None
  non_stop: bool | None = None
  price: str | None = None
  train: bool | None = None
  unpriced_reason: str | None = None


@dataclasses.dataclass
class DestinationResult:
  """Destination result.

  Attributes:
    departure_date: Departure date.
    destination_airports: Destination airports.
    destination_name: Destination name.
    displayed_duration: Displayed duration.
    displayed_price: Displayed price.
    flights_shopping_url: Flights shopping url.
    return_date: Return date.
  """

  departure_date: str | None = None
  destination_airports: list[str] | None = None
  destination_name: str | None = None
  displayed_duration: str | None = None
  displayed_price: str | None = None
  flights_shopping_url: str | None = None
```

```
 return_date: str | None = None


@dataclasses.dataclass
class Flight:
 """Flight.

 Attributes:
   destination: Destination.
   origin: Origin.
   airline: Airline.
   airline_logo: Airline logo.
   arrival_time: Arrival time.
   departure_time: Departure time.
   duration: Duration.
   flight_number: Flight number.
   flight_url: Flight url.
   layover: Layover.
   price: Price.
   stops: Stops.
 """

destination: str;

origin: str;

 airline: list[str] | None = None
 airline_logo: str | None = None
 arrival_time: str | None = None
 departure_time: str | None = None
 duration: str | None = None
 flight_number: list[str] | None = None
 flight_url: str | None = None
layover: Union[(list["Layover"], None)] = None;

 price: str | None = None
 stops: int | None = None
```

```
@dataclasses.dataclass
class Layover:
 """Layover.

 Attributes:
   duration: Duration.
   location: Location.
 """

 duration: str | None = None
 location: str | None = None


@dataclasses.dataclass
class LocationCode:
 """Location code.

 Attributes:
   code: Code.
 """

 code: str | None = None


@dataclasses.dataclass
class MultiDestinationShoppingResult:
 """Multi destination shopping result.

 Attributes:
   destinations: Destinations.
   origin_airports: Origin airports.
   origin_name: Origin name.
 """

destinations: Union[(list["DestinationResult"], None)] = None;

 origin_airports: list[str] | None = None
 origin_name: str | None = None
```

```
@dataclasses.dataclass
class OriginDestinationSpecificDatesResult:
 """Origin destination specific dates result.

 Attributes:
   departure_date: Departure date.
   destination: Represents a location.
   origin: Represents a location.
   return_date: Return date.
   row: Row.
 """

 departure_date: str | None = None
destination: Union[("LocationCode", None)] = None;

origin: Union[("LocationCode", None)] = None;

 return_date: str | None = None
row: Union[(list["CarrierRow"], None)] = None;



@dataclasses.dataclass
class SearchResult:
 """Search result.

 Attributes:
   flights_shopping_url: Flights shopping url.
   multi_destination_shopping_result: Multi destination shopping result.
   one_way: One way.
   origin_destination_specific_dates_result: Result of a universal flights
search for ORIGIN_DESTINATION_SPECIFIC_DATES.
   shopping_result: Shopping result.
 """

 flights_shopping_url: str | None = None
multi_destination_shopping_result: Union[
 ("MultiDestinationShoppingResult", None)
```

```
] = None;

 one_way: bool | None = None
origin_destination_specific_dates_result: Union[
 ("OriginDestinationSpecificDatesResult", None)
] = None;

shopping_result: Union[("ShoppingResult", None)] = None;



@dataclasses.dataclass
class ShoppingResult:
 """Shopping result.

 Attributes:
   departure_date: Departure date.
   destination_airports: Destination airports.
   destination_name: Destination name.
   flights: Flights.
   flights_shopping_url: Flights shopping url.
   origin_airports: Origin airports.
   origin_name: Origin name.
   return_date: Return date.
 """

 departure_date: str | None = None
 destination_airports: list[str] | None = None
 destination_name: str | None = None
flights: Union[(list["Flight"], None)] = None;

 flights_shopping_url: str | None = None
 origin_airports: list[str] | None = None
 origin_name: str | None = None
 return_date: str | None = None


class TripDays(str, Enum):
WEEKEND = "WEEKEND";
```

```
def search(
   origin: str | None = None,
   destination: str | None = None,
   earliest_departure_date: str | None = None,
   earliest_return_date: str | None = None,
   latest_departure_date: str | None = None,
   latest_return_date: str | None = None,
   min_length_of_stay: int | None = None,
   max_length_of_stay: int | None = None,
   include_airlines: list[str] | None = None,
   max_duration_minutes: int | None = None,
   max_price: int | None = None,
   currency: str | None = None,
   max_stops: int | None = None,
   depart_after_hour: int | None = None,
   depart_before_hour: int | None = None,
   arrive_after_hour: int | None = None,
   arrive_before_hour: int | None = None,
   carry_on_bag_count: int | None = None,
   checked_bag_count: int | None = None,
   trip_days: TripDays | None = None,
   seating_classes: list[str] | None = None,
   cheapest: bool | None = None,
   one_way: bool | None = None,
   num_adult_passengers: int | None = None,
   num_child_passengers: int | None = None,
   num_infant_in_lap_passengers: int | None = None,
   num_infant_in_seat_passengers: int | None = None,
) -> SearchResult | str:
 """Search or get booking links for flights.

 Args:
   origin: The location where the trip starts. If the location is not specified
in the user query, it should be left empty.
```

```
   destination: The final destination of the trip. If it is not provided in the user query, it should be left empty.
   earliest_departure_date: Filter for the earliest (or only) departure date in YYYY-MM-DD format.
   earliest_return_date: Filter for the earliest (or only) return date in YYYY-MM-DD format.
   latest_departure_date: Filter for the latest departure date in YYYY-MM-DD format. Set this to be the same as earliest_departure_date unless a departure date range is requested.
   latest_return_date: Filter for the latest return date in YYYY-MM-DD format. Set this to be the same as earliest_return_date unless a return date range is requested.
   min_length_of_stay: Minimum length of stay before returning.
   max_length_of_stay: Maximum length of stay before returning.
   include_airlines: Filter by flights on these airlines only.
   max_duration_minutes: Filter for maximum duration of the flight.
   max_price: Filter for maximum price limit of the flight.
   currency: Price currency.
   max_stops: Filter for maximum number of stops/layovers.
   depart_after_hour: Filter for flights that depart after this hour (1-23).
   depart_before_hour: Filter for flights that depart before this hour (1-23).
   arrive_after_hour: Filter for flights that arrive after this hour (1-23).
   arrive_before_hour: Filter for flights that arrive before this hour (1-23).
   carry_on_bag_count: Filter for number of carry on bags.
   checked_bag_count: Filter for number of checked bags.
   trip_days: Filter for specific grouping of flight trips like Weekends.
   seating_classes: Filter for seating classes of the flight.
   cheapest: If TRUE, the results will be sorted by price (in ascending order).
   one_way: If TRUE, a query will be issued for a one-way trip instead of round-trip (the default).
   num_adult_passengers: The number of adult passengers.
   num_child_passengers: The number of child passengers.
   num_infant_in_lap_passengers: The number of infant passengers (in lap).
   num_infant_in_seat_passengers: The number of infant passengers (in seat).
 """
...
```