```
%pip install -U -q "google-generativeai>=0.7.2"
```

```
import google.generativeai as genai
from google.colab import userdata
GOOGLE_API_KEY=userdata.get('GOOGLE_API_KEY')
genai.configure(api_key=GOOGLE_API_KEY)
```

```
model=genai.GenerativeModel('models/gemini-2.0-flash')
response=model.generate_content('please give me python code to sort a list.')

print(response.text)
```

Here are several ways to sort a list in Python, along with explanations of their dif

1. `list.sort()` (In-place sorting)

* Modifies the original list directly.
* Returns `None`.
* Most efficient for sorting a list in place.

```python
my_list = [3, 1, 4, 1, 5, 9, 2, 6]
my_list.sort()  # Sorts in ascending order
print(my_list)  # Output: [1, 1, 2, 3, 4, 5, 6, 9]

my_list.sort(reverse=True)  # Sorts in descending order
print(my_list)  # Output: [9, 6, 5, 4, 3, 2, 1, 1]

# Sorting a list of strings alphabetically
string_list = ["apple", "banana", "cherry", "date"]
string_list.sort()
print(string_list) # Output: ['apple', 'banana', 'cherry', 'date']
```

2. `sorted()` (Returns a new sorted list)

* Creates a *new* sorted list, leaving the original list unchanged.
* Returns the new sorted list.
* Useful when you want to preserve the original list.

```python
my_list = [3, 1, 4, 1, 5, 9, 2, 6]
new_list = sorted(my_list)  # Sorts in ascending order
print(my_list)  # Output: [3, 1, 4, 1, 5, 9, 2, 6] (original list unchanged)
print(new_list)  # Output: [1, 1, 2, 3, 4, 5, 6, 9]

new_list_desc = sorted(my_list, reverse=True)  # Sorts in descending order
print(new_list_desc)  # Output: [9, 6, 5, 4, 3, 2, 1, 1]

#Sort a list of tuples, based on the first element in the tuple.
list_of_tuples = [(1, 'a'), (3, 'c'), (2, 'b')]
sorted_list = sorted(list_of_tuples)  # sorts by the first element of the tuple
print(sorted_list) # Output: [(1, 'a'), (2, 'b'), (3, 'c')]
```

3. Custom Sorting with `key`

Both `list.sort()` and `sorted()` accept a `key` argument. The `key` argument sho

```python
# Sorting a list of strings by length
words = ["apple", "banana", "kiwi", "orange"]
words.sort(key=len)  # Sorts by length of the string (in-place)
print(words)  # Output: ['kiwi', 'apple', 'banana', 'orange']

# Sorting a list of dictionaries by a specific key
people = [
    {"name": "Alice", "age": 30},
```

```
    {"name": "Bob", "age": 25},
    {"name": "Charlie", "age": 35}
]

people.sort(key=lambda person: person["age"])  # Sorts by age (in-place)
print(people)
# Output:
# [{'name': 'Bob', 'age': 25}, {'name': 'Alice', 'age': 30}, {'name': 'Charlie',

# Using operator.itemgetter (more efficient than lambda for simple key access)
import operator
people.sort(key=operator.itemgetter("age"))
print(people)
```

4. `reverse()` (Reversing a list)

* Reverses the order of elements in a list.
* `list.reverse()` modifies the original list *in place*.
* `reversed(list)` returns an iterator to loop through the reversed list.

```python
my_list = [1, 2, 3, 4, 5]
my_list.reverse()
print(my_list)  # Output: [5, 4, 3, 2, 1]

my_list = [1, 2, 3, 4, 5]
for item in reversed(my_list):
    print(item)  # Prints 5, 4, 3, 2, 1
```

```
**Choosing the Right Method**

*   **`list.sort()`:** Use when you want to sort the list directly and don't need to
*   **`sorted()`:** Use when you want to create a *new* sorted list and preserve the
*   **`reverse()`:** Used for reversing the order of elements, which is a different

**Important Considerations**

*   **Mutability:**  `list.sort()` is a *mutable* operation (modifies the original l
*   **Efficiency:** For in-place sorting, `list.sort()` is generally faster than cre
*   **Readability:** Use `sorted()` if keeping the original list makes your code mor
*   **Key Function:**  The `key` parameter allows you to sort based on complex crite

Example of sorting a list of objects:

```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __repr__(self): # For easier printing
        return f"Person(name='{self.name}', age={self.age})"


people = [
    Person("Alice", 30),
```
```

```
    Person("Bob", 25),
    Person("Charlie", 35)
]

# Sort by age using a lambda function
people.sort(key=lambda person: person.age)
print(people)

# Sort by name using a lambda function
people.sort(key=lambda person: person.name)
print(people)
```

```
from google import genai
from google.genai import types
client = genai.Client(api_key=GOOGLE_API_KEY)
```

```
MODEL_ID = "gemini-2.0-flash" # @param ["gem:
from IPython.display import display, Markdow
response=client.models.generate_content(
    model=MODEL_ID,
    contents="what's the largest planent in (
)
Markdown(response.text)
```

MODEL_ID: gemini-2.0-flash

The largest planet in our solar system is **Jupiter**.

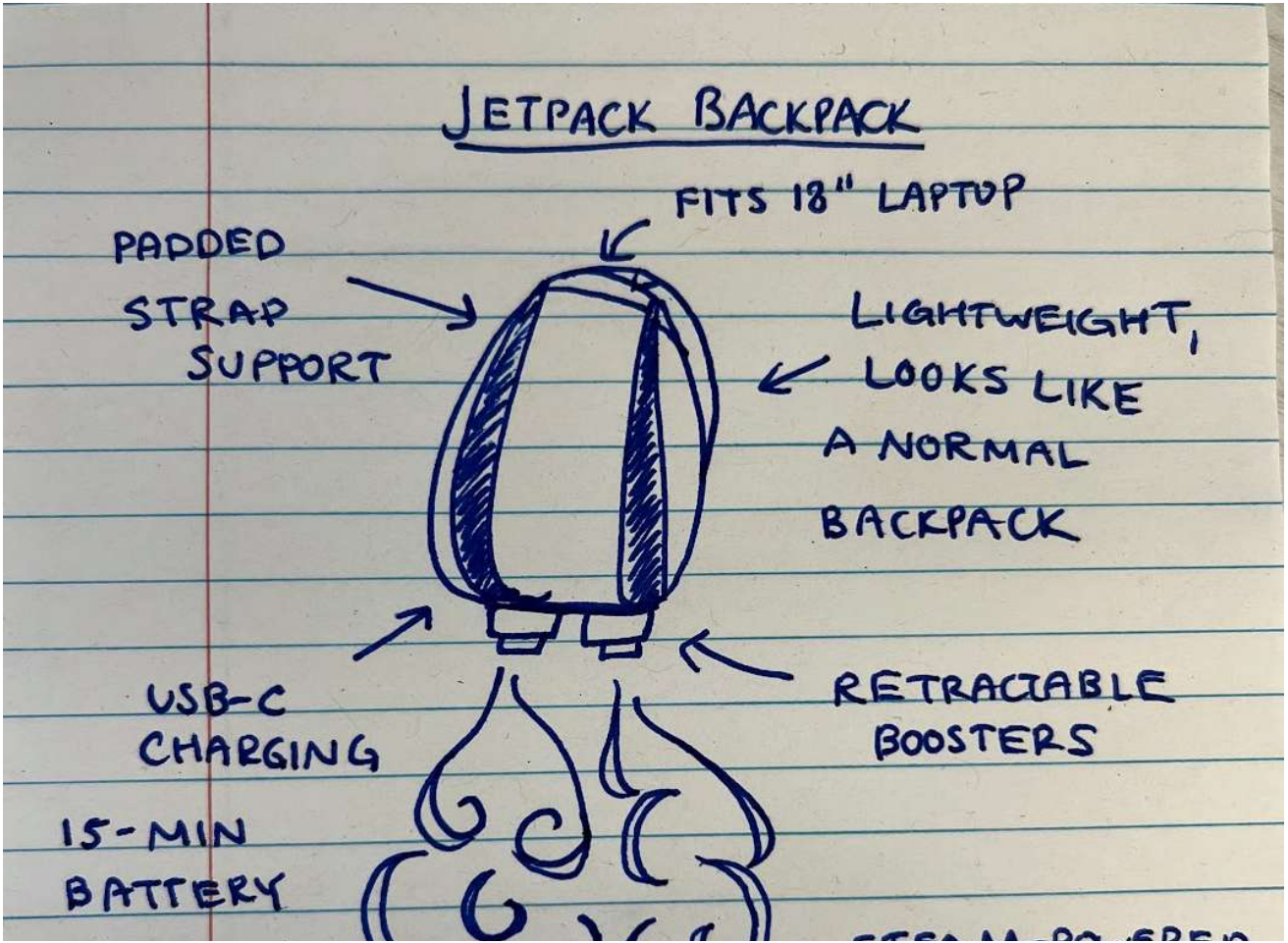
```
response=client.models.count_tokens(
    model=MODEL_ID,
    contents="what is the hightest mountain in india?"
)
print(response)
```

```
total_tokens=10 cached_content_token_count=None
```

```
import requests
import pathlib
from PIL import Image
from io import BytesIO
IMG="https://storage.googleapis.com/generativeai-downloads/data/jetpack.png"
img_path=pathlib.Path('jetpack.png')
response = requests.get(IMG)
img_bytes = response.content
img_path.write_bytes(img_bytes)
```

```
1567837
```

```
from IPython.display import display,Markdown
image=Image.open(img_path)
display(image)
```


JETPACK BACKPACK
FITS 18" LAPTOP
PADDED STRAP SUPPORT
LIGHTWEIGHT, LOOKS LIKE A NORMAL BACKPACK
USB-C CHARGING
RETRACTABLE BOOSTERS
15-MIN BATTERY