In [10]:
```
# Get the API key from here: https://ai.google.dev/tutorials/setup
# Create a new secret called "GEMINI_API_KEY" via Add-ons -> Secrets in t
from kaggle_secrets import UserSecretsClient
from IPython.display import display
from IPython.display import Markdown

import pathlib
import textwrap

user_secrets = UserSecretsClient()
apiKey = user_secrets.get_secret("GEMINI_API_KEY")

def to_markdown(text):
  text = text.replace('•', '  *')
  return Markdown(textwrap.indent(text, '> ', predicate=lambda _: True))
```

In [11]:
```
import google.generativeai as genai

genai.configure(api_key = apiKey)
```

Generate text from text-only inputs using `gemini-pro`

The `generate_content` method can handle a wide variety of use cases, including multi-turn chat and multimodal input, depending on what the underlying model supports. The available models only support text and images as input, and text as output.

In the simplest case, you can pass a prompt string to the `GenerativeModel.generate_content` method:

In [12]:
```
model = genai.GenerativeModel('gemini-pro')
response = model.generate_content("What are the available products of Uni
to_markdown(response.text)
```

Out[12]:

1. **SMS:** Unifonic provides a comprehensive range of SMS services, including:
    - **Transactional SMS:** Send time-sensitive messages such as OTPs, appointment reminders, and delivery notifications.
    - **Bulk SMS:** Reach a large audience with promotional messages, marketing campaigns, and alerts.
    - **International SMS:** Send SMS messages to over 200 countries and territories worldwide.
    - **2-Way SMS:** Enable interactive communication with customers through SMS conversations.
2. **Voice:** Unifonic's voice solutions include:
    - **Voice Calls:** Make and receive voice calls using a variety of protocols, including SIP, PSTN, and WebRTC.
    - **IVR:** Create interactive voice response systems to handle incoming calls and provide self-service options.
    - **Outbound Dialer:** Automate outbound calling campaigns for lead generation, appointment setting, and surveys.
3. **Chat:** Unifonic offers a range of chat solutions, including:
    - **WhatsApp Business API:** Integrate WhatsApp with your business to send and receive messages, share media, and provide customer support.
    - **Facebook Messenger API:** Connect with customers on Facebook Messenger and provide real-time support.
    - **Web Chat:** Embed a chat widget on your website or mobile app to enable real-time conversations with customers.
    - **Live Chat:** Offer live chat support to customers on your website or mobile app.
4. **Email:** Unifonic's email services include:
    - **Transactional Email:** Send triggered emails such as order confirmations, invoices, and shipping notifications.
    - **Marketing Email:** Create and send marketing campaigns, newsletters, and promotional emails.
    - **Email Verification:** Validate email addresses to ensure deliverability and reduce bounce rates.
5. **Push Notifications:** Unifonic provides push notification solutions for mobile apps, including:
    - **iOS Push Notifications:** Send push notifications to iOS devices.
    - **Android Push Notifications:** Send push notifications to Android devices.
    - **Cross-Platform Push Notifications:** Send push notifications to both iOS and Android devices.

6. **Authentication:** Unifonic offers a range of authentication solutions, including:
    - **One-Time Password (OTP):** Send OTPs via SMS or email for secure user authentication.
    - **Two-Factor Authentication (2FA):** Add an extra layer of security to user accounts with 2FA.
    - **Passwordless Authentication:** Enable passwordless authentication using OTPs or biometric factors.
7. **Analytics:** Unifonic provides analytics tools to help businesses track and measure the performance of their communication campaigns, including:
    - **SMS Analytics:** Track the delivery, open, and click rates of SMS messages.
    - **Voice Analytics:** Monitor call volumes, average call duration, and abandoned calls.
    - **Chat Analytics:** Analyze chat conversations, response times, and customer satisfaction ratings.
    - **Email Analytics:** Track email delivery rates, open rates, click-through rates, and unsubscribes.
    - **Push Notification Analytics:** Monitor the delivery, open, and click rates of push notifications.

In [13]:
```
# View the response candidates
response.candidates
```

Out[13]:
```
[content {
  parts {
    text: "1. **SMS:** Unifonic provides a comprehensive range of SMS se
rvices, including:\n    - **Transactional SMS:** Send time-sensitive mes
sages such as OTPs, appointment reminders, and delivery notifications.\n
- **Bulk SMS:** Reach a large audience with promotional messages, market
ing campaigns, and alerts.\n    - **International SMS:** Send SMS messag
es to over 200 countries and territories worldwide.\n    - **2-Way SMS:*
* Enable interactive communication with customers through SMS conversati
ons.\n\n2. **Voice:** Unifonic\'s voice solutions include:\n    - **Voic
e Calls:** Make and receive voice calls using a variety of protocols, in
cluding SIP, PSTN, and WebRTC.\n    - **IVR:** Create interactive voice
response systems to handle incoming calls and provide self-service optio
ns.\n    - **Outbound Dialer:** Automate outbound calling campaigns for
lead generation, appointment setting, and surveys.\n\n3. **Chat:** Unifo
nic offers a range of chat solutions, including:\n    - **WhatsApp Busin
ess API:** Integrate WhatsApp with your business to send and receive mes
sages, share media, and provide customer support.\n    - **Facebook Mess
enger API:** Connect with customers on Facebook Messenger and provide re
al-time support.\n    - **Web Chat:** Embed a chat widget on your websit
e or mobile app to enable real-time conversations with customers.\n    -
**Live Chat:** Offer live chat support to customers on your website or m
obile app.\n\n4. **Email:** Unifonic\'s email services include:\n    - *
*Transactional Email:** Send triggered emails such as order confirmation
s, invoices, and shipping notifications.\n    - **Marketing Email:** Cre
ate and send marketing campaigns, newsletters, and promotional emails.\n
- **Email Verification:** Validate email addresses to ensure deliverabil
ity and reduce bounce rates.\n\n5. **Push Notifications:** Unifonic prov
ides push notification solutions for mobile apps, including:\n    - **iO
S Push Notifications:** Send push notifications to iOS devices.\n    - *
*Android Push Notifications:** Send push notifications to Android device
s.\n    - **Cross-Platform Push Notifications:** Send push notifications
to both iOS and Android devices.\n\n6. **Authentication:** Unifonic offe
rs a range of authentication solutions, including:\n    - **One-Time Pas
sword (OTP):** Send OTPs via SMS or email for secure user authenticatio
n.\n    - **Two-Factor Authentication (2FA):** Add an extra layer of sec
urity to user accounts with 2FA.\n    - **Passwordless Authentication:**
Enable passwordless authentication using OTPs or biometric factors.\n\n
7. **Analytics:** Unifonic provides analytics tools to help businesses t
rack and measure the performance of their communication campaigns, inclu
ding:\n    - **SMS Analytics:** Track the delivery, open, and click rate
s of SMS messages.\n    - **Voice Analytics:** Monitor call volumes, ave
rage call duration, and abandoned calls.\n    - **Chat Analytics:** Anal
yze chat conversations, response times, and customer satisfaction rating
s.\n    - **Email Analytics:** Track email delivery rates, open rates, c
lick-through rates, and unsubscribes.\n    - **Push Notification Analyti
cs:** Monitor the delivery, open, and click rates of push notification
s."
  }
  role: "model"
}
finish_reason: STOP
index: 0
safety_ratings {
  category: HARM_CATEGORY_SEXUALLY_EXPLICIT
  probability: NEGLIGIBLE
}
safety_ratings {
  category: HARM_CATEGORY_HATE_SPEECH
  probability: NEGLIGIBLE
```

```
}
safety_ratings {
  category: HARM_CATEGORY_HARASSMENT
  probability: NEGLIGIBLE
}
safety_ratings {
  category: HARM_CATEGORY_DANGEROUS_CONTENT
  probability: NEGLIGIBLE
}
]
```

In [14]:
```
# If API failed to return a result, use prompt_feedback to see if it was
response.prompt_feedback
```

Out[14]:
```
safety_ratings {
  category: HARM_CATEGORY_SEXUALLY_EXPLICIT
  probability: NEGLIGIBLE
}
safety_ratings {
  category: HARM_CATEGORY_HATE_SPEECH
  probability: NEGLIGIBLE
}
safety_ratings {
  category: HARM_CATEGORY_HARASSMENT
  probability: NEGLIGIBLE
}
safety_ratings {
  category: HARM_CATEGORY_DANGEROUS_CONTENT
  probability: NEGLIGIBLE
}
```

Generate text from image and text prompts using `gemini-pro-vision`

In [15]:
```
!curl -o image.jpg https://t0.gstatic.com/licensed-image?q=tbn:ANd9GcQ_Ke
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Cu
rrent
                                 Dload  Upload   Total   Spent    Left  Sp
eed
100  405k  100  405k    0     0  95640      0 --:--:-- --:--:-- --:--:--
0k      0 --:--:-- --:--:-- --:--:-- 9644k
```

In [16]:
```
import PIL.Image

img = PIL.Image.open('image.jpg')
img
```

Out[16]:

In [19]:
```
model = genai.GenerativeModel('gemini-pro-vision')
```

In [20]:
```
response = model.generate_content(img)

to_markdown(response.text)
```

Out[20]:

Meal prepping can be a great way to save time and money, and it can also help you to eat healthier. By planning your meals ahead of time and cooking them in advance, you can make sure that you have healthy and nutritious meals ready to eat when you're hungry.

There are many different ways to meal prep, and the best approach for you will depend on your individual needs and preferences. Some people prefer to cook all of their meals for the week on Sunday, while others prefer to cook just a few meals at a time. You can also choose to meal prep for breakfast, lunch, or dinner, or all three meals.

No matter how you choose to meal prep, there are a few tips that can help you to make the most of it. First, make sure that you have the right tools and equipment. This includes a good set of knives, a cutting board, a measuring cup, and a measuring spoon. You may also want to invest in some airtight containers for storing your food.

Second, plan your meals ahead of time. This will help you to make sure that you have all of the ingredients you need, and it will also help you to stay on track with your meal prepping goals.

Third, cook your food in advance. This will save you time and energy during the week, and it will also help to ensure that your food is fresh and healthy.

Finally, be creative with your meal prepping. There are many different ways to prepare healthy and delicious meals, so don't be afraid to experiment. With a little planning and effort, you can easily create meal prepped meals that you'll love.

Here are some additional tips for meal prepping:

- Choose recipes that are simple and easy to make.
- Use fresh, healthy ingredients.
- Cook your food in bulk so that you have leftovers for later.
- Store your food in airtight containers in the refrigerator or freezer.
- Label your containers so that you know what's inside.
- Reheat your food safely before eating.

Meal prepping can be a great way to save time, money, and eat healthier. By following these tips, you can easily create meal prepped meals that you'll love.

In [21]:
```
response = model.generate_content(["Write a short, engaging blog post bas
to_markdown(response.text)
```

Out[21]:

> Meal prepping is a great way to save time and money, and it can also help you to eat healthier. This meal is a great example of a healthy and delicious meal that can be easily prepped ahead of time.
>
> This meal features brown rice, roasted vegetables, and chicken teriyaki. The brown rice is a whole grain that is high in fiber and nutrients. The roasted vegetables are a great way to get your daily dose of vitamins and minerals. And the chicken teriyaki is a lean protein source that is also packed with flavor.
>
> This meal is easy to assemble, and it can be stored in the refrigerator for up to 4 days. When you're ready to eat, simply reheat the meal in the microwave or oven.
>
> If you're looking for a healthy and delicious meal that can be easily prepped ahead of time, this meal is a great option. Give it a try today!
>
> **Tips for Meal Prepping:**
>
> - Choose healthy and nutritious foods that will keep well in the refrigerator.
> - Cook your food in advance and portion it out into individual containers.
> - Label your containers with the date and contents.
> - Store your containers in the refrigerator or freezer.
> - When you're ready to eat, reheat your food in the microwave or oven.

Chat conversations

Gemini enables you to have freeform conversations across multiple turns. The `ChatSession` class simplifies the process by managing the state of the conversation, so unlike with `generate_content`, you do not have to store the conversation history as a list.

In [23]:
```
model = genai.GenerativeModel('gemini-pro')
chat = model.start_chat(history=[])
chat
```

Out[23]:
```
<google.generativeai.generative_models.ChatSession at 0x7a5c33ada7d0>
```

In [24]:
```
response = chat.send_message("In one sentence, explain how a computer wor
to_markdown(response.text)
```

Out[24]:

> A computer is like a magic machine that can store, understand, and show information by following your instructions.

In [25]:
```
chat.history
```

Out[25]:
```
[parts {
   text: "In one sentence, explain how a computer works to a young chil
d."
 }
 role: "user",
 parts {
   text: "A computer is like a magic machine that can store, understand,
and show information by following your instructions."
 }
 role: "model"]
```

In [26]:
```
response = chat.send_message("Okay, how about a more detailed explanation
to_markdown(response.text)
```

Out[26]:

> A computer works by following a set of instructions, called a program, which tells it what to do with data. The data is stored in the computer's memory, and the program is executed by the computer's processor, which performs calculations and makes decisions based on the instructions in the program. The results of these calculations are then displayed on the computer's screen or sent to other devices, such as printers or storage devices.
>
> Here's a simplified analogy: Imagine a computer as a chef following a recipe. The recipe (program) contains instructions on how to cook a dish (data). The chef (processor) follows the recipe, gathering ingredients (data) and performing actions (calculations) to create the dish (results). The dish is then served (displayed or sent to other devices).

In [27]:
```
for message in chat.history:
  display(to_markdown(f'**{message.role}**: {message.parts[0].text}'))
```

> **user**: In one sentence, explain how a computer works to a young child.

> **model**: A computer is like a magic machine that can store, understand, and show information by following your instructions.

> **user**: Okay, how about a more detailed explanation to a high schooler?

> **model**: A computer works by following a set of instructions, called a program, which tells it what to do with data. The data is stored in the computer's memory, and the program is executed by the computer's processor, which performs calculations and makes decisions based on the instructions in the program. The results of these calculations are then displayed on the computer's screen or sent to other devices, such as printers or storage devices.
>
> Here's a simplified analogy: Imagine a computer as a chef following a recipe. The recipe (program) contains instructions on how to cook a dish (data). The chef (processor) follows the recipe, gathering ingredients (data) and performing actions (calculations) to create the dish (results). The dish is then served (displayed or sent to other devices).

Use embeddings

Embedding is a technique used to represent information as a list of floating point numbers in an array. With Gemini, you can represent text (words, sentences, and blocks of text) in a vectorized form, making it easier to compare and contrast embeddings. For example, two texts that share a similar subject matter or sentiment should have similar embeddings, which can be identified through mathematical comparison techniques such as cosine similarity.

Use the `embed_content` method to generate embeddings. The method handles embedding for the following tasks (`task_type`):

Task Type	Description
RETRIEVAL_QUERY	Specifies the given text is a query in a search/retrieval setting.
RETRIEVAL_DOCUMENT	Specifies the given text is a document in a search/retrieval setting. Using this task type requires a `title` .
SEMANTIC_SIMILARITY	Specifies the given text will be used for Semantic Textual Similarity (STS).
CLASSIFICATION	Specifies that the embeddings will be used for classification.
CLUSTERING	Specifies that the embeddings will be used for clustering.

The following generates an embedding for a single string for document retrieval:

In [28]:
```
result = genai.embed_content(
    model="models/embedding-001",
    content="What are the available products of Unifonic company?",
    task_type="retrieval_document",
    title="Embedding of single string")

# 1 input > 1 vector output
print(str(result['embedding'])[:50], '... TRIMMED]')
```

```
[-0.0046282657, -0.015581417, -0.015204371, -0.007 ... TRIMMED]
```

Note: The `retrieval_document` task type is the only task that accepts a title.

To handle batches of strings, pass a list of strings in `content`:

In [29]:
```
result = genai.embed_content(
    model="models/embedding-001",
    content=[
      'What are the available products of Unifonic company?',
      'How much wood would a woodchuck chuck?',
      'How does the brain work?'],
    task_type="retrieval_document",
    title="Embedding of list of strings")

# A list of inputs > A list of vectors output
for v in result['embedding']:
  print(str(v)[:50], '... TRIMMED ...')
```

```
[0.004390321, -0.00063095096, -0.013835617, -0.001 ... TRIMMED ...
[-0.004049845, -0.0075574904, -0.0073463684, -0.03 ... TRIMMED ...
[0.025310587, -0.0080734305, -0.029902633, 0.01160 ... TRIMMED ...
```

While the `genai.embed_content` function accepts simple strings or lists of strings, it is actually built around the `glm.Content` type (like `GenerativeModel.generate_content`). `glm.Content` objects are the primary units of conversation in the API.

While the `glm.Content` object is multimodal, the `embed_content` method only supports text embeddings. This design gives the API the *possibility* to expand to multimodal embeddings.

In [30]:
```
response.candidates[0].content
```

Out[30]:
```
parts {
  text: "A computer works by following a set of instructions, called a p
rogram, which tells it what to do with data. The data is stored in the c
omputer\'s memory, and the program is executed by the computer\'s proces
sor, which performs calculations and makes decisions based on the instru
ctions in the program. The results of these calculations are then displa
yed on the computer\'s screen or sent to other devices, such as printers
or storage devices.\n\nHere\'s a simplified analogy: Imagine a computer
as a chef following a recipe. The recipe (program) contains instructions
on how to cook a dish (data). The chef (processor) follows the recipe, g
athering ingredients (data) and performing actions (calculations) to cre
ate the dish (results). The dish is then served (displayed or sent to ot
her devices)."
}
role: "model"
```

In [31]:
```
result = genai.embed_content(
    model = 'models/embedding-001',
    content = response.candidates[0].content)

# 1 input > 1 vector output
print(str(result['embedding'])[:50], '... TRIMMED ...')
```

```
[-0.002888101, -0.029394535, -0.0005772255, 0.0454 ... TRIMMED ...
```

In [32]:
```
chat.history
```

Out[32]:
```
[parts {
   text: "In one sentence, explain how a computer works to a young chil
d."
 }
 role: "user",
 parts {
   text: "A computer is like a magic machine that can store, understand,
and show information by following your instructions."
 }
 role: "model",
 parts {
   text: "Okay, how about a more detailed explanation to a high schoole
r?"
 }
 role: "user",
 parts {
   text: "A computer works by following a set of instructions, called a
program, which tells it what to do with data. The data is stored in the
computer\'s memory, and the program is executed by the computer\'s proce
ssor, which performs calculations and makes decisions based on the instr
uctions in the program. The results of these calculations are then displ
ayed on the computer\'s screen or sent to other devices, such as printer
s or storage devices.\n\nHere\'s a simplified analogy: Imagine a compute
r as a chef following a recipe. The recipe (program) contains instructio
ns on how to cook a dish (data). The chef (processor) follows the recip
e, gathering ingredients (data) and performing actions (calculations) to
create the dish (results). The dish is then served (displayed or sent to
other devices)."
 }
 role: "model"]
```

In [33]:
```
result = genai.embed_content(
    model = 'models/embedding-001',
    content = chat.history)

# 1 input > 1 vector output
for i,v in enumerate(result['embedding']):
  print(str(v)[:50], '... TRIMMED...')
```

```
[-0.014632266, -0.042202696, -0.015757175, 0.01548 ... TRIMMED...
[-0.028278848, -0.016093584, -0.0016516016, 0.0110 ... TRIMMED...
[-0.010055617, -0.07208932, -0.00011750793, -0.023 ... TRIMMED...
[-0.002888101, -0.029394535, -0.0005772255, 0.0454 ... TRIMMED...
```

Encode messages

The previous sections relied on the SDK to make it easy for you to send prompts to the API. This section offers a fully-typed equivalent to the previous example, so you can better understand the lower-level details regarding how the SDK encodes messages.

Underlying the Python SDK is the `google.ai.generativelanguage` client library:

In [34]:
```
import google.ai.generativelanguage as glm
```

The SDK attempts to convert your message to a `glm.Content` object, which contains a list of `glm.Part` objects that each contain either:

1. a `text` (string)
2. `inline_data` (`glm.Blob`), where a blob contains binary `data` and a `mime_type`.

You can also pass any of these classes as an equivalent dictionary.

Note: The only accepted mime types are some image types, `image/*`.

So, the fully-typed equivalent to the previous example is:

In [35]:
```
model = genai.GenerativeModel('gemini-pro-vision')
response = model.generate_content(
    glm.Content(
        parts = [
            glm.Part(text="Write a short, engaging blog post based on thi
            glm.Part(
                inline_data=glm.Blob(
                    mime_type='image/jpeg',
                    data=pathlib.Path('image.jpg').read_bytes()
                )
            ),
        ],
    )
)
to_markdown(response.text[:100] + "... [TRIMMED] ...")
```

Out[35]:

> Meal prepping is a great way to save time and money, and it can also help you to eat healthier. By ... [TRIMMED] ...

Multi-turn conversations

While the `genai.ChatSession` class shown earlier can handle many use cases, it does make some assumptions. If your use case doesn't fit into this chat implementation it's good to remember that `genai.ChatSession` is just a wrapper around `GenerativeModel.generate_content`. In addition to single requests, it can handle multi-turn conversations.

The individual messages are `glm.Content` objects or compatible dictionaries, as seen in previous sections. As a dictionary, the message requires `role` and `parts` keys. The `role` in a conversation can either be the `user`, which provides the prompts, or `model`, which provides the responses.

Pass a list of `glm.Content` objects and it will be treated as multi-turn chat:

In [36]:
```
model = genai.GenerativeModel('gemini-pro')

messages = [
    {'role':'user',
     'parts': ["Briefly explain how a computer works to a young child."]}
]
response = model.generate_content(messages)
```

```
to_markdown(response.text)
```

Out[36]:

1. **Have a brain:** The computer has a part called the CPU (Central Processing Unit) which acts like the brain and tells the computer what to do.
2. **Remember things:** The computer uses a part called the memory to remember information, just like you use your memory to remember things.
3. **See and hear:** The computer uses devices like the monitor (like your TV) to show you things, and speakers to play sounds, so you can see and hear what it's doing.
4. **Talk and listen:** You can talk to the computer using the keyboard and mouse, and it listens to you and responds by showing you things on the screen or making sounds.
5. **Store stuff:** The computer can store lots of information, like pictures, music, and videos, on a part called the hard drive, which is like a big box of drawers that never gets full.

To continue the conversation, add the response and another message.

Note: For multi-turn conversations, you need to send the whole conversation history with each request. The API is **stateless**.

In [37]:

```
messages.append({'role':'model',
                 'parts':[response.text]})

messages.append({'role':'user',
                 'parts':["Okay, how about a more detailed explanation to

response = model.generate_content(messages)

to_markdown(response.text)
```

Out[37]:

1. **CPU (Central Processing Unit):** The CPU is the brain of the computer. It controls all the other parts of the computer and tells them what to do. It also performs calculations and makes decisions.
2. **Memory (RAM):** Memory stores the instructions and data that the CPU is currently working on. It's like a short-term memory that the computer uses to quickly access information it needs right away.
3. **Storage (Hard Drive):** Storage is where the computer keeps all of its files and programs. It's like a long-term memory that can store vast amounts of information, even when the computer is turned off.
4. **Input Devices (Keyboard, Mouse):** Input devices allow you to communicate with the computer. The keyboard lets you type in text and commands, while the mouse lets you point and click on things on the screen.
5. **Output Devices (Monitor, Speakers):** Output devices show you the results of what the computer is doing. The monitor displays images and text, while the speakers play sounds.
6. **Network Card:** The network card allows the computer to connect to the internet and other computers. It sends and receives data over the network.
7. **Operating System (OS):** The OS is the software that manages the computer's resources and provides basic services to other software. It's like the traffic controller of the computer, making sure that everything works smoothly.
8. **Software (Programs):** Software is a set of instructions that tells the computer how to perform specific tasks, such as word processing, playing games, or browsing the web.

All of these components work together to make the computer a powerful tool that can be used for a wide variety of tasks.

Generation configuration

The `generation_config` argument allows you to modify the generation parameters. Every prompt you send to the model includes parameter values that control how the model generates responses.

In [38]:
```
response = model.generate_content(
    'Tell me a story about a magic backpack.',
    generation_config=genai.types.GenerationConfig(
        # Only one candidate for now.
        candidate_count=1,
        stop_sequences=['x'],
        max_output_tokens=20,
        temperature=1.0)
)
```

In [39]:
```
to_markdown(response.text)
```

Out[39]:

> In an old, forgotten corner of the world, there lived a child named Lily. She possessed a

In [ ]: