L4: Describe-and-Generate game 🖍️

Load your HF API key and relevant Python libraries

In [ ]:

```
import os
import io
from IPython.display import Image, display, HTML
from PIL import Image
import base64

from dotenv import load_dotenv, find_dotenv
_ = load_dotenv(find_dotenv()) # read local .env file
hf_api_key = os.environ['HF_API_KEY']
```

In [ ]:

```
# Helper function
import requests, json

#Here we are going to call multiple endpoints!
def get_completion(inputs, parameters=None, ENDPOINT_URL=""):
    headers = {
      "Authorization": f"Bearer {hf_api_key}",
      "Content-Type": "application/json"
    }
    data = { "inputs": inputs }
    if parameters is not None:
        data.update({"parameters": parameters})
    response = requests.request("POST",
                                ENDPOINT_URL,
                                headers=headers,
                                data=json.dumps(data))
    return json.loads(response.content.decode("utf-8"))
```

In [ ]:

```
#text-to-image
TTI_ENDPOINT = os.environ['HF_API_TTI_BASE']
#image-to-text
ITT_ENDPOINT = os.environ['HF_API_ITT_BASE']
```

Building your game with `gr.Blocks()`

In [ ]:

```
#Bringing the functions from lessons 3 and 4!
def image_to_base64_str(pil_image):
    byte_arr = io.BytesIO()
    pil_image.save(byte_arr, format='PNG')
    byte_arr = byte_arr.getvalue()
    return str(base64.b64encode(byte_arr).decode('utf-8'))

def base64_to_pil(img_base64):
    base64_decoded = base64.b64decode(img_base64)
    byte_stream = io.BytesIO(base64_decoded)
    pil_image = Image.open(byte_stream)
    return pil_image

def captioner(image):
    base64_image = image_to_base64_str(image)
    result = get_completion(base64_image, None, ITT_ENDPOINT)
    return result[0]['generated_text']

def generate(prompt):
    output = get_completion(prompt, None, TTI_ENDPOINT)
    result_image = base64_to_pil(output)
    return result_image
```

First attempt, just captioning

In [ ]:

```
import gradio as gr
with gr.Blocks() as demo:
    gr.Markdown("# Describe-and-Generate game 🖍️")
    image_upload = gr.Image(label="Your first image",type="pil")
    btn_caption = gr.Button("Generate caption")
    caption = gr.Textbox(label="Generated caption")

    btn_caption.click(fn=captioner, inputs=[image_upload], outputs=[caption])

gr.close_all()
demo.launch(share=True, server_port=int(os.environ['PORT1']))
```

Let's add generation

In [ ]:

```
with gr.Blocks() as demo:
    gr.Markdown("# Describe-and-Generate game 🖍️ ")
    image_upload = gr.Image(label="Your first image",type="pil")
    btn_caption = gr.Button("Generate caption")
    caption = gr.Textbox(label="Generated caption")
    btn_image = gr.Button("Generate image")
    image_output = gr.Image(label="Generated Image")
    btn_caption.click(fn=captioner, inputs=[image_upload], outputs=[caption])
    btn_image.click(fn=generate, inputs=[caption], outputs=[image_output])

gr.close_all()
demo.launch(share=True, server_port=int(os.environ['PORT2']))
```

Doing it all at once!

In [ ]:

```
def caption_and_generate(image):
    caption = captioner(image)
    image = generate(caption)
    return [caption, image]

with gr.Blocks() as demo:
    gr.Markdown("# Describe-and-Generate game 🖍️ ")
    image_upload = gr.Image(label="Your first image",type="pil")
    btn_all = gr.Button("Caption and generate")
    caption = gr.Textbox(label="Generated caption")
    image_output = gr.Image(label="Generated Image")

    btn_all.click(fn=caption_and_generate, inputs=[image_upload], outputs=[captior

gr.close_all()
demo.launch(share=True, server_port=int(os.environ['PORT3']))
```

In [ ]:

```
gr.close_all()
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: