Exercise 2.2

The code for the data loader with `max_length=2` and `stride=2`:

```
dataloader = create_dataloader(
    raw_text, batch_size=4, max_length=2, stride=2
)
```

It produces batches of the following format:

```
tensor([[  40,  367],
        [2885, 1464],
        [1807, 3619],
        [ 402,  271]])
```

The code of the second data loader with `max_length=8` and `stride=2`:

```
dataloader = create_dataloader(
    raw_text, batch_size=4, max_length=8, stride=2
)
```

An example batch looks like

```
tensor([[   40,   367,  2885,  1464,  1807,  3619,   402,   271],
        [ 2885,  1464,  1807,  3619,   402,   271, 10899,  2138],
        [ 1807,  3619,   402,   271, 10899,  2138,   257,  7026],
        [  402,   271, 10899,  2138,   257,  7026, 15632,   438]])
```

Chapter 3

Exercise 3.1

The correct weight assignment is

```
sa_v1.W_query = torch.nn.Parameter(sa_v2.W_query.weight.T)
sa_v1.W_key = torch.nn.Parameter(sa_v2.W_key.weight.T)
sa_v1.W_value = torch.nn.Parameter(sa_v2.W_value.weight.T)
```

Exercise 3.2

To achieve an output dimension of 2, similar to what we had in single-head attention, we need to change the projection dimension `d_out` to 1.

```
d_out = 1
mha = MultiHeadAttentionWrapper(d_in, d_out, block_size, 0.0, num_heads=2)
```

Exercise 3.3

The initialization for the smallest GPT-2 model is

```
block_size = 1024
d_in, d_out = 768, 768
num_heads = 12
mha = MultiHeadAttention(d_in, d_out, block_size, 0.0, num_heads)
```

Chapter 4

Exercise 4.1

We can calculate the number of parameters in the feed forward and attention modules as follows:

```
block = TransformerBlock(GPT_CONFIG_124M)

total_params = sum(p.numel() for p in block.ff.parameters())
print(f"Total number of parameters in feed forward module: {total_params:,}")

total_params = sum(p.numel() for p in block.att.parameters())
print(f"Total number of parameters in attention module: {total_params:,}")
```

As we can see, the feed forward module contains approximately twice as many parameters as the attention module:

```
Total number of parameters in feed forward module: 4,722,432
Total number of parameters in attention module: 2,360,064
```

Exercise 4.2

To instantiate the other GPT model sizes, we can modify the configuration dictionary as follows (here shown for GPT-2 XL):

```
GPT_CONFIG = GPT_CONFIG_124M.copy()
GPT_CONFIG["emb_dim"] = 1600
GPT_CONFIG["n_layers"] = 48
GPT_CONFIG["n_heads"] = 25
model = GPTModel(GPT_CONFIG)
```

Then, reusing the code from section 4.6 to calculate the number of parameters and RAM requirements, we find

```
gpt2-xl:
Total number of parameters: 1,637,792,000
Number of trainable parameters considering weight tying: 1,557,380,800
Total size of the model: 6247.68 MB
```

Exercise 4.3

There are three distinct places in chapter 4 where we used dropout layers: the embedding layer, shortcut layer, and multi-head attention module. We can control the dropout rates for each of the layers by coding them separately in the config file and then modifying the code implementation accordingly.

The modified configuration is as follows:

```
GPT_CONFIG_124M = {
    "vocab_size": 50257,
```

```
    "context_length": 1024,
    "emb_dim": 768,
    "n_heads": 12,
    "n_layers": 12,
    "drop_rate_attn": 0.1,
    "drop_rate_shortcut": 0.1,
    "drop_rate_emb": 0.1,
    "qkv_bias": False
}
```

Dropout for multi-head attention

Dropout for shortcut connections

Dropout for embedding layer

The modified `TransformerBlock` and `GPTModel` look like

```
class TransformerBlock(nn.Module):
    def __init__(self, cfg):
        super().__init__()
        self.att = MultiHeadAttention(
            d_in=cfg["emb_dim"],
            d_out=cfg["emb_dim"],
            context_length=cfg["context_length"],
            num_heads=cfg["n_heads"],
            dropout=cfg["drop_rate_attn"],
            qkv_bias=cfg["qkv_bias"])
        self.ff = FeedForward(cfg)
        self.norm1 = LayerNorm(cfg["emb_dim"])
        self.norm2 = LayerNorm(cfg["emb_dim"])
        self.drop_shortcut = nn.Dropout(
            cfg["drop_rate_shortcut"]
        )

    def forward(self, x):
        shortcut = x
        x = self.norm1(x)
        x = self.att(x)
        x = self.drop_shortcut(x)
        x = x + shortcut

        shortcut = x
        x = self.norm2(x)
        x = self.ff(x)
        x = self.drop_shortcut(x)
        x = x + shortcut
        return x


class GPTModel(nn.Module):
    def __init__(self, cfg):
        super().__init__()
        self.tok_emb = nn.Embedding(
            cfg["vocab_size"], cfg["emb_dim"]
        )
```

Dropout for multi-head attention

Dropout for shortcut connections