Aim

- Understand the data types supported by tensors
- Type conversion
- Importance of device and grad while creating a tensor

Tensor Data Types

Tensors support wide range of data types

Category	Data Type	Aliases/Notes
Floating Point	32-bit floating point	`torch.float32` , `torch.float`
	64-bit floating point	`torch.float64` , `torch.double`
	16-bit floating point	`torch.float16` , `torch.half`
	16-bit floating point (bfloat16)	`torch.bfloat16`
	8-bit floating point (e4m3)	`torch.float8_e4m3fn` (limited support)
	8-bit floating point (e5m2)	`torch.float8_e5m2` (limited support)
Complex	32-bit complex	`torch.complex32` , `torch.chalf`
	64-bit complex	`torch.complex64` , `torch.cfloat`
	128-bit complex	`torch.complex128` , `torch.cdouble`
Unsigned Integer	8-bit integer	`torch.uint8`
	16-bit integer	`torch.uint16` (limited support)
	32-bit integer	`torch.uint32` (limited support)
	64-bit integer	`torch.uint64` (limited support)
Signed Integer	8-bit integer	`torch.int8`
	16-bit integer	`torch.int16` , `torch.short`
	32-bit integer	`torch.int32` , `torch.int`
	64-bit integer	`torch.int64` , `torch.long`
Boolean	Boolean	`torch.bool`

Quantized	Quantized 8-bit integer (unsigned)	`torch.quint8`
	Quantized 8-bit integer (signed)	`torch.qint8`
	Quantized 32-bit integer (signed)	`torch.qint32`
	Quantized 4-bit integer (unsigned)	`torch.quint4x2`

torch.device

A tensor can be created on a particular device (CPU, GPU). The `device` argument to the tensor constructor specifies where the tensor needs to be created.

Device options: CPU, CUDA, MPS

Notes:

- CUDA is a parallel computing platform and application programming interface model for Nvidia GPUs.
- MPS (Metal Performance Shaders) is Apple's framework for Apple Silicon GPUs.
- Two tensors need to be on the same device to be operated on.

In [1]:
```
import torch as th
print(f"Is CUDA/GPU available? {th.cuda.is_available()}")
print(f"Is MPS enabled? {th.backends.mps.is_available()}")
print("MPS built:", th.backends.mps.is_built())
```

```
Is CUDA/GPU available? False
Is MPS enabled? True
MPS built: True
```

Creating tensors of specific type

Note: requires_grad flag specifies if gradient needs to be tracked for a tensor. Generally used for backpropagation

In [2]:
```
float_32_vector_on_mps = th.tensor([1.0, 2.0, 3.0],
                                   dtype=th.float32, # Specify the dat
                                   device='mps', # Specify the device
                                   requires_grad=False) # Specify if g

float_16_vector_on_mps = th.tensor([1.0, 2.0, 3.0],
                                   dtype=th.float16, # Specify the dat
                                   device='mps', # Specify the device
                                   requires_grad=False) # Specify if g

second_float_32_vector_on_mps = th.tensor([4.0, 5.0, 6.0],
                                          dtype=th.float32,
                                          device='mps',
                                          requires_grad=False)



float_32_vector_on_cpu = th.tensor([1.0, 2.0, 3.0],
                                   dtype=th.float32,
                                   device='cpu',
                                   requires_grad=False)

print(f'Data Type of float_32_vector_on_mps: {float_32_vector_on_mps.d
print(f'Data Type of float_32_vector_on_cpu: {float_32_vector_on_cpu.d
print(f'Device of float_32_vector_on_mps: {float_32_vector_on_mps.devi
print(f'Device of float_32_vector_on_cpu: {float_32_vector_on_cpu.devi
```

```
Data Type of float_32_vector_on_mps: torch.float32
Data Type of float_32_vector_on_cpu: torch.float32
Device of float_32_vector_on_mps: mps:0
Device of float_32_vector_on_cpu: cpu
```

In [3]:
```
float_32_vector_on_mps + second_float_32_vector_on_mps
```

Out[3]:
```
tensor([5., 7., 9.], device='mps:0')
```

Note: Tensors on different devices cannot be operated on together.

In [6]:
```
# Tensor on MPS cannot be added to a tensor on CPU
try:
    float_32_vector_on_mps + float_32_vector_on_cpu
except RuntimeError as e:
    print(e)
```

```
Expected all tensors to be on the same device, but found at least two
devices, mps:0 and cpu!
```

Tensor type conversion

The `.to` method can be used to convert the type of a tensor. It can also be used to move the tensor from one device to another.

Tensors also have `.float()`, `.bool()` methods for type conversion.

In [5]:
```
f16 = float_32_vector_on_mps.to(dtype=th.float16) # Convert to float16
print(f'Data Type of float_32_vector_on_mps and f16: {f16.dtype} and {

f32_on_mps = float_32_vector_on_cpu.to(device='mps') # Move to MPS
print(f'Device of float_32_vector_on_mps and f32_on_mps: {f32_on_mps.d

# After moving float_32_vector_on_cpu to MPS, it can be added to float
print(float_32_vector_on_mps + f32_on_mps)

# Type converting using .bool()
print(f'Boolean value of float_32_vector_on_mps: {float_32_vector_on_m
```

```
Data Type of float_32_vector_on_mps and f16: torch.float16 and torch.f
loat32
Device of float_32_vector_on_mps and f32_on_mps: mps:0 and mps:0
tensor([2., 4., 6.], device='mps:0')
Boolean value of float_32_vector_on_mps: tensor([True, True, True], de
vice='mps:0')
```