1. Tensors & Shapes

```
x = torch.randn(32, 2)      # shape: [batch_size, dim]
t = torch.rand(32, 1)       # shape: [32, 1]
x_t = (1 - t) * x0 + t * x1
```

2. Indexing & Slicing

```
x[0]         # first element
x[:, 1]      # second column
x[:5]        # first 5 rows
```

3. Functions

```
def interpolate(x0, x1, t):
    return (1 - t) * x0 + t * x1
```

4. Loop

```
for epoch in range(100):
    ...
```

5. List & Dict

```
layers = [nn.Linear(64, 64), nn.ReLU()]
params = {'lr': 1e-3, 'batch_size': 128}
```

6. Broadcasting

```
# t: [32, 1], x: [32, 2]  (1 - t) * x works automatically
```

7. No Gradient / Eval Mode

```
with torch.no_grad():
    sample = model(x)

model.eval()
```

8. Class & Model

```
class MLP(nn.Module):
    def __init__(self):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(3, 64), nn.ReLU(), nn.Linear(64, 3)
        )

    def forward(self, x):
        return self.net(x)
```

9. Training Step

```
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

Pro Tips

```
len(x)  Get batch size
x.view(-1, 1)  Reshape tensor
nn.Sequential(...)  Quick network
t = torch.rand(batch_size, 1)  Per-sample interpolation
x0 + t * (x1 - x0)  Same as (1 - t) * x0 + t * x1
```