Sending Packets
Creating and sending a packet:
```python
>>> packet = IP(dst="4.5.6.7", src="1.2.3.4")/TCP(dport=80, flags="S")
```
Send a packet, or list of packets without custom ether layer:
```python
>>> send(packet)
```

Other send functions:
- `sr()`: sends and receives without a custom ether layer
- `sendp()`: sends with a custom ether layer
- `sr1()`: sends packets without custom ether layer and waits for first answer
- `sr1p()`: sends packets with custom ether layer and waits for first answer

Send function options:
- `filter = <Berkley Packet Filter>`
- `retry = <retry count for unanswered packets>`
- `timeout = <number of seconds to wait before giving up>`
- `iface = <interface to send and receive>`
```python
>>> packets = sr(packet, retry=5, timeout=1.5, iface="eth0", filter="host 1.2.3.4 and port 80")
```

Receiving and Analyzing Packets
Received packets can be stored in a variable when using a send/receive function such as `sr()`, `srp()`, `sr1()` `sr1p()`:
```python
>>> packet = IP(dst="10.10.10.20")/TCP(dport=80)

>>> unans, ans = sr(packet)
```
Received 1086 packets, got 1024 answers, remaining 0 packets

“ans” will store the answered packets:
```python
>>> ans
<Results: TCP:1024 UDP:0 ICMP:0 Other:0>
```

To see a summary of the responses:
```python
>>> ans.summary()
IP / TCP 10.1.1.15:ftp_data > 10.10.10.20:netbios_ssn S ==> IP / TCP 10.10.10.20:netbios_ssn > 10.1.1.15:ftp_data SA / Padding
```

Note: this is the output from port 139 (`netbios_ssn`). Notice how this port was open and responded with a SYN-ACK.

To view a specific answer as a stream in array form:
```python
>>> ans[15]
```

To view the first packet in the stream:
```python
>>> ans[15][0] (this will be packet the Scapy sent)
```

To view the response from the distant end:
```python
>>> ans[15][1]
```

To view the TCP flags in the response packet:
```python
>>> ans[15][1].sprintf("%TCP.flags%")
'RA'
```

Sniffing and pcaps
To sniff using Berkley Packet Filters:
```python
>>> packets = sniff(filter="host 1.1.1.1")
```

Sniffing using counts:
```python
>>> packets = sniff(count=100)
```

Reading packets from a pcap:
```python
>>> packets = rdpcap("filename.pcap")
```

Writing packets to a pcap:
```python
>>> wrpcap("filename.pcap", packets)
```
Scapy Basics

To list supported layers:
>>> ls()

Some key layers are:
arp, ip, ipv6, tcp, udp, icmp

To view layer fields use ls(layer):
>>> ls(IPv6)
>>> ls(TCP)

To list available commands:
>>> lsc()

Some key commands for interacting with packets:
rdpcap, send, sr, sniff, wrpcap

Getting help with commands use help(command):
>>> help(rdpcap)

Basic Packet Crafting / Viewing

Scapy works with layers. Layers are individual functions linked together with the "/" character to construct packets. To build a basic TCP/IP packet with "data" as the payload:
>>> packet = IP(dst="1.2.3.4")/TCP(dport=22)/"data"

Note: Scapy allows the user to craft all the way down to the ether() layer, but will use default values if layers are omitted. To correctly pass traffic layers should be ordered lowest to highest from left to right e.g. (ether -> IP -> TCP).

To get a packet summary:
>>> packet.summary()

To get more packet details:
>>> packet.show()