Python Binary File and Socket Programming Cheat Sheet

📁 File Operations

Opening Binary Files

```
with open("filename.bin", "rb") as f:  # 'rb' = read binary mode
    # file operations here
```

File Positioning

```
f.seek(offset)          # Move file pointer to specific byte position
f.seek(0)               # Move to beginning of file
f.seek(100)             # Move to byte 100
```

Reading Binary Data

```
data = f.read(size)     # Read 'size' bytes from current position
data = f.read()         # Read entire file from current position
```

Checking Data Length

```
if len(data) < expected_size:
    # Handle insufficient data
```

🔧 Struct Module (Binary Data Parsing)

Common Format Codes

Code	Type	Size (bytes)	Description
`s`	string	varies	Fixed-length string
`i`	int	4	Signed integer
`I`	int	4	Unsigned integer
`f`	float	4	Single precision float
`d`	float	8	Double precision float

Format String Examples

```
'20s'     # 20-byte string
'i'       # 4-byte signed integer
'20si'    # 20-byte string + 4-byte signed integer
'3i'      # Three 4-byte signed integers
```

Key Functions

```
# Calculate size of format
size = struct.calcsize('20si')    # Returns total bytes needed

# Unpack binary data into Python objects
data = struct.unpack('20si', binary_data)
# Returns tuple: (string_as_bytes, integer)
```

Unpacking Example

```
binary_data = f.read(24)  # Read 24 bytes (20 for string + 4 for int)
result = struct.unpack('20si', binary_data)
# result[0] = bytes object (the string)
# result[1] = integer
```

🧵 String Processing

Decoding Bytes to String

```
byte_string = b'hello\x00\x00\x00'
text = byte_string.decode('utf-8')      # Convert bytes to string
```

Removing Null Characters

```
clean_text = text.strip('\x00')         # Remove null padding
# or
clean_text = text.rstrip('\x00')        # Remove trailing nulls only
```

Combined Operation

```
domain = unpacked_data[0].decode('utf-8').strip('\x00')
```

🌐 Socket Module Functions

Getting Hostname

```
import socket
hostname = socket.gethostname()         # Get local machine name
```

Domain to IP Resolution

```
ip_address = socket.gethostbyname('google.com')
# Returns: '172.217.12.142' (example)

# Handle errors:
try:
    ip = socket.gethostbyname(domain)
except socket.gaierror:
    print("Domain not found")
```

Port to Service Name

```
service = socket.getservbyport(80)      # Returns 'http'
service = socket.getservbyport(22)      # Returns 'ssh'
service = socket.getservbyport(443)     # Returns 'https'

# Handle errors:
try:
    service = socket.getservbyport(port)
except OSError:
    print("Unknown service")
```

🧮 Common Calculations

File Offset Calculation

```
# To find the nth record (1-indexed):
offset = (line_number - 1) * record_size

# Example: To get record 5 with 24-byte records:
offset = (5 - 1) * 24 = 96  # Start at byte 96
```

📝 Error Handling Patterns

File Operations

```
try:
    with open(filename, 'rb') as f:
        # file operations
except FileNotFoundError:
    print(f"File {filename} not found")
except Exception as e:
    print(f"Error reading file: {e}")
```

Network Operations

```
try:
    result = socket.gethostbyname(domain)
except socket.gaierror:
    print("Domain resolution failed")
except Exception as e:
    print(f"Network error: {e}")
```

🔍 Debugging Tips

Print Data Types and Values

```
print(f"Type: {type(data)}, Value: {data}")
print(f"Length: {len(data)}")
print(f"Raw bytes: {data!r}")  # Shows \x00 characters
```

Check Unpacked Data

```python
unpacked = struct.unpack('20si', data)
print(f"Unpacked tuple: {unpacked}")
print(f"String part: {unpacked[0]!r}")
print(f"Integer part: {unpacked[1]}")
```

📋 Step-by-Step Workflow

1. **Calculate offset** for desired record
2. **Seek** to that position in file
3. **Read** the exact number of bytes for one record
4. **Check** if enough data was read
5. **Unpack** binary data using struct
6. **Decode and clean** the string data
7. **Use socket functions** for network operations

⚠️ Common Pitfalls

- **Off-by-one errors**: Remember files are 0-indexed, but line numbers are 1-indexed
- **Insufficient data**: Always check if `f.read()` returned enough bytes
- **Null padding**: Binary strings often have `\x00` padding that needs removal
- **Error handling**: Network operations can fail, always use try-except blocks