# List Comprehension

List comprehension হলো Python-এর এক compact, expressive উপায় একটি নতুন list বানানোর জন্য — এক বা একাধিক loop এবং optional condition ব্যবহার করে। AI/ML কোডে খুবই প্রচলিত কারণ এটি কোড সংক্ষিপ্ত, দ্রুত (সাধারণত) এবং পাইথনিক করে। নিচে ধারণা থেকে শুরু করে কাজের উদাহরণ, কর্মপ্রক্রিয়া, পারফরম্যান্স ও সতর্কতা সব দেয়া আছে।

### 1) সহজরীতি (syntax)

[ expression for item in iterable ]

উদাহরণ:

```
squares = [x*x for x in range(6)]   # [0, 1, 4, 9, 16, 25]
```

expression — যেকোন ভ্যালু/অভিব্যক্তি যেটা প্রতিটি item থেকে তৈরি করবে। iterable হতে পারে list, range, generator, file ইত্যাদি।

### 2) শর্ত (filter) যুক্ত করলে

[ expression for item in iterable if condition ]

উদাহরণ:

```
even_squares = [x*x for x in range(10) if x % 2 == 0]  # [0,4,16,36,64]
```

if হল ফিল্টার — শুধুমাত্র যেসব item satisfy করে সেগুলোই expression-এ যাবে।

### 3) একাধিক লুপ (nested)

[ expr for a in A for b in B ]

এটা ঠিক একই ক্রমে চলে যেন দুটি nested for:

```
pairs = [(a,b) for a in [1,2] for b in [10,20]]  # [(1,10),(1,20),(2,10),(2,20)]
```

Equivalent nested loops:

```
pairs = []
for a in [1,2]:
    for b in [10,20]:
        pairs.append((a,b))
```

**4) nested comprehension (matrix flattening / 2D → 1D)**

```
matrix = [[1,2,3],[4,5,6]]
flat = [x for row in matrix for x in row]  # [1,2,3,4,5,6]
```

সাবধান: nested comprehension পড়তে একটু কষ্ট হতে পারে—complex হলে loop ব্যবহার করা ভালো।

**5) multiple conditions**

```
res = [x for x in range(30) if x%2==0 if x%3!=0]
```

# দুইটা if আলাদা ভাবে লেখা হয়েছে; একই ফলাফল যেটা হলে 'and' ব্যবহার করেও করা যায়

**6) expression অংশে জটিল লজিক**

```
labels = ["even" if x%2==0 else "odd" for x in range(6)]
```

# ['even','odd','even','odd','even','odd']

এখানে ternary expression ব্যবহার করা হয়েছে: A if cond else B — মনে রাখবে if (filter) এর সাথে if-else (ternary) মিলিয়ে না ফেলবে: filter গতিতে if শেষে আসে, ternary expression if-else expression অংশে থাকে।

**7) generator expression (lazy alternative)**

List comprehension পুরো list দ্রুতই তৈরী করে (memory), কিন্তু যদি lazy evaluation চাইলে generator expression ব্যবহার করো:

```
gen = (x*x for x in range(10))  # round brackets --> generator
next(gen)  # 0
```

Generator ভালো যখন ডেটা বড় (streaming preprocessing) — কিন্তু যদি তুমি indexed/random access চাও তাহলে list দরকার।

**8) list comprehension vs map/filter vs for-loop**

- listcomp সাধারণত পড়তে সহজ এবং Pythonic।
- map/filter functional style; কখনো কখনো ছোট তাৎক্ষণিক কাজ দ্রুত হতে পারে, কিন্তু readibility কমতে পারে।
- for-loop সবচেয়ে explicit এবং debug-friendly — জটিল লজিক আছে হলে for-loop ব্যবহার করো।

উদাহরণ (map vs listcomp):

```
res = [f(x) for x in items if cond(x)]

# map+filter (less readable often)
res = list(map(f, filter(cond, items)))
```

**9) Scoping / variable binding (important subtlety)**

List comprehension ভিতরে ব্যবহৃত loop-ভেরিয়েবল বাইরের scope-এ overwrite করে না (Python 3-এ)। কিন্তু closure তৈরি হলে সাবধান:

```
funcs = [lambda x, i=i: i*x for i in range(5)]
```

```
# এখানে i=i জরুরি; না হলে সব lambda শেষের i পাবে
[f(2) for f in funcs]  # [0,2,4,6,8]
```

ব্যাখ্যা: closure যখন বাইন্ড করা হয়, প্রতিটি lambda-কে আলাদা default arg দিলেই লক্ষ্যভিত্তিক মান ধরে রাখা যায়।

**10) Performance ও memory**

- ছোট-মাঝারি list তৈরিতে list comprehension দ্রুত (C-implemented loops) এবং readable।
- বড় ডেটাসেটে (millions of rows) পুরো list তৈরির আগেই memory ফিলাপ হতে পারে — তখন generator বা streaming/chunked processing (pandas chunks, iterators) ব্যবহার করো।
- profiling: timeit ব্যবহার করে তুলনা করা যায় — কিন্তু readability ও memory cost মাথায় রাখো।

উদাহরণ (benchmark idea):

```
# ক্ষুদ্র উদাহরণ: squares generation
import timeit
timeit.timeit("[x*x for x in range(1000)]", number=1000)
```

**11) Dict / Set Comprehension**

List comprehension-এর মতোই:

```
s = {x*x for x in range(10)}
d = {x: x*x for x in range(5)}  # {0:0,1:1,2:4,3:9,4:16}
```

## 12) Practical AI/ML উদাহরণসমূহ

**(ক) feature extraction — list → features**

```
texts = ["This is good", "bad", "average"]
lengths = [len(t.split()) for t in texts]  # word count per text
```

**(খ) labels → one-hot (small classes)**

```
classes = ["cat","dog","fish"]
one_hot = [{c:1 if c==cls else 0 for c in classes} for cls in ["dog","cat"]]
# [{'cat':0,'dog':1,'fish':0}, {'cat':1,'dog':0,'fish':0}]
```

**(গ) flatten predictions from batches**

```
batches = [[0,1,1],[1,0],[0,0,1]]
flat_preds = [p for batch in batches for p in batch]
```

**(ঘ) preprocess large CSV rows lazily (but map each row fields quickly)**

```
def process_file(path):
  with open(path) as f:
    for line in f:
      tokens = line.strip().split()
      features = [len(tok) for tok in tokens]  # short feature vector
      yield features
```

(এখানে generator ব্যবহার করেছি; প্রতিটি features তৈরি করতে list comprehension ব্যবহার করা হয়)

**(ঙ) convert numpy array rows to tensors (but prefer vectorized ops)**

```
import numpy as np
arr = np.random.rand(1000, 10)
```

# Not recommended — slower than vectorized numpy:

```
row_sums = [row.sum() for row in arr]
```

# Better:

```
row_sums_np = arr.sum(axis=1)
```

**মন্তব্য:** Numpy/Pandas ব্যবহার করলে vectorized ops ব্যবহার করা শ্রেয় — list comprehension স্রেফ Python সাইডে loop করে যা ধীর হতে পারে।

**13) Readability & Best Practices**

- সরল থাকো: এক-লাইনে সবকিছু করার চেষ্টা করো না। যদি expression ৩-লাইনের বেশি জটিল হবে — for-loop নিতে ভাবো।
- PEP8: লাইন 79 চেয়ে ছোট রাখার চেষ্টা করো। অনেক nested comprehension থাকলে ভেঙে লিখো বা function বানাও।
- নামকরণ: short কিন্তু অর্থবোধক ভেরিয়েবল নাম ব্যবহার করো (x ছোট loop-এ ok, কিন্তু feature_vector বড় হলে ভালো)
- কমেন্ট/টেস্ট: ব্যাখ্যা যুক্ত করো যখন logic nontrivial।

**14) সারাংশ (Quick checklist)**

- Use list comprehensions for readable, simple mapping/filtering tasks.
- Use generator expressions for memory efficiency (lazy).
- Use numpy/pandas vectorized ops for numeric heavy-lifting in ML.
- Avoid very complex nested comprehensions — prefer loops or helper functions.
- Watch closures and variable binding — use default args trick when needed.