What is AT & GC Percentage?

DNA is made of **four nucleotides**:

- **A** = Adenine
- **T** = Thymine
- **G** = Guanine
- **C** = Cytosine

AT Percentage

Percentage of **A + T** in a DNA sequence.

AT%=A+TTotal bases×100AT\% = \frac{A + T}{\text{Total bases}} \times 100AT%=Total basesA+T×100

GC Percentage

Percentage of **G + C** in a DNA sequence.

GC%=G+CTotal bases×100GC\% = \frac{G + C}{\text{Total bases}} \times 100GC%=Total basesG+C×100

Why GC% is important

- High GC → DNA is more **stable**
- Used in:
  - Genome comparison
  - Primer design
  - Species identification
  - Sequence quality analysis

What is a FASTA File?

A **FASTA file** stores biological sequences.

Format:

```
>Sequence_ID
ATGCGTACGTTAGC
```

FASTA with multiple sequences:

```
>seq1
ATGCGTAC
>seq2
GGCCATTA
>seq3
ATATATGC
```

- Line starting with > → **Header**
- Next lines → **Sequence**

How to Read a FASTA File

Install Biopython (one time)

```
pip install biopython
```

Reading FASTA file

```
from Bio import SeqIO

for record in SeqIO.parse("input.fasta", "fasta"):
    print("ID:", record.id)
    print("Sequence:", record.seq)
```

Calculate GC & AT Percentage

GC & AT calculation logic

```
def calculate_gc_at(seq):
    seq = seq.upper()
    length = len(seq)

    g = seq.count("G")
    c = seq.count("C")
    a = seq.count("A")
    t = seq.count("T")

    gc_percent = (g + c) / length * 100
    at_percent = (a + t) / length * 100

    return gc_percent, at_percent
```

Program: Read FASTA & Calculate GC/AT

```
from Bio import SeqIO

def calculate_gc_at(seq):
    seq = seq.upper()
    length = len(seq)

    gc = seq.count("G") + seq.count("C")
    at = seq.count("A") + seq.count("T")

    gc_percent = (gc / length) * 100
    at_percent = (at / length) * 100

    return gc_percent, at_percent

# Read FASTA file
for record in SeqIO.parse("input.fasta", "fasta"):
    gc, at = calculate_gc_at(record.seq)
    print(f"Sequence ID: {record.id}")
```

```
    print(f"GC%: {gc:.2f}")
    print(f"AT%: {at:.2f}")
    print("-" * 30)
```

Write a New FASTA File

Example: write sequences **with GC% in header**

```
from Bio import SeqIO
from Bio.SeqRecord import SeqRecord

output_records = []

for record in SeqIO.parse("input.fasta", "fasta"):
    gc, at = calculate_gc_at(record.seq)

    new_header = f"{record.id}_GC_{gc:.2f}"
    new_record = SeqRecord(
        record.seq,
        id=new_header,
        description=""
    )
    output_records.append(new_record)

# Write new FASTA file
SeqIO.write(output_records, "output.fasta", "fasta")
```

Example Output FASTA

```
>seq1_GC_50.00
ATGCGTAC

>seq2_GC_75.00
GGCCATTA

>seq3_GC_25.00
ATATATGC
```

Problem Statement (Simple Words)

- Input: A FASTA file containing **multiple sequences**
- Task:
  - Check the **length** of each sequence
  - If **length > 20**, keep it
  - If **length ≤ 20**, ignore it
- Output: A **new FASTA file** containing **only long sequences**

Why This Is Useful?

- Short sequences may be:
  - Incomplete
  - Low quality
  - Not biologically meaningful
- Length filtering is common in:
  - Genome analysis

- Protein studies
- FASTA cleaning

Example Input FASTA (`input.fasta`)

```
>seq1
ATGCGTAC

>seq2
ATGCGTACGTTAGCTAGCTAG

>seq3
ATGCGTACGTTAGCTAGCTAGGCTA
```

Lengths:

- seq1 → 8
- seq2 → 21
- seq3 → 25

Python Program

Step 1: Import required modules

```
from Bio import SeqIO
from Bio.SeqRecord import SeqRecord
```

Step 2: Read FASTA & filter sequences

```
filtered_records = []

for record in SeqIO.parse("input.fasta", "fasta"):
    seq_length = len(record.seq)

    if seq_length > 20:
        filtered_records.append(record)
```

Step 3: Write filtered sequences to a new FASTA file

```
SeqIO.write(filtered_records, "output.fasta", "fasta")
```

This creates a new file **output.fasta**

Complete Program

```
from Bio import SeqIO
filtered_records = []

for record in SeqIO.parse("input.fasta", "fasta"):
    if len(record.seq) > 20:
        filtered_records.append(record)

SeqIO.write(filtered_records, "output.fasta", "fasta")
print("FASTA file written with sequences longer than 20 letters.")
```

Output FASTA (`output.fasta`)

```
>seq2
ATGCGTACGTTAGCTAGCTAG

>seq3
ATGCGTACGTTAGCTAGCTAGGCTA
```

What is a GenBank File?

A **GenBank file (`.gb` / `.gbk`)** is a **rich sequence format** that stores:

- DNA / RNA / Protein sequence
- Sequence **ID**
- **Description**
- **Annotations** (organism, molecule type, topology, etc.)
- **Features** (genes, CDS, source)

Unlike FASTA, **GenBank contains metadata + biological information**.

Main Components of a GenBank File

Component	Meaning
ID	Unique sequence identifier
Description	Short explanation of the sequence
Sequence	Actual nucleotide or protein sequence
Annotations	Extra biological info
molecule_type	DNA / RNA / protein
topology	linear or circular
Features	gene, CDS, source, etc.

Required Libraries

Install Biopython (if not installed):

```
pip install biopython
```

Creating a GenBank File Step by Step

We use:

- `Seq`
- `SeqRecord`
- `SeqIO`

Step 1: Import modules

```
from Bio.Seq import Seq
```

```
from Bio.SeqRecord import SeqRecord
from Bio import SeqIO
```

Step 2: Create a Sequence

```
sequence = Seq("ATGCGTACGTTAGCTAGCTAGGCTA")
```

Step 3: Create a SeqRecord (ID + Description)

```
record = SeqRecord(
    sequence,
    id="SEQ001",
    name="ExampleSequence",
    description="This is a sample DNA sequence for GenBank file"
)
```

Step 4: Add Annotations

Required GenBank annotations:

```
record.annotations["molecule_type"] = "DNA"
record.annotations["topology"] = "linear"
record.annotations["organism"] = "Homo sapiens"
record.annotations["data_file_division"] = "BCT"
record.annotations["date"] = "26-JAN-2026"
```

Important:
Without `molecule_type`, GenBank writing will fail

Step 5: Write GenBank File

```
SeqIO.write(record, "output.gb", "genbank")
```

GenBank file created successfully!

Complete Working Program

```
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord
from Bio import SeqIO

# Create sequence
sequence = Seq("ATGCGTACGTTAGCTAGCTAGGCTA")

# Create record
record = SeqRecord(
    sequence,
    id="SEQ001",
    name="ExampleSequence",
    description="This is a sample DNA sequence for GenBank file"
)

# Add annotations
record.annotations["molecule_type"] = "DNA"
record.annotations["topology"] = "linear"
```

```
record.annotations["organism"] = "Homo sapiens"
record.annotations["data_file_division"] = "BCT"
record.annotations["date"] = "26-JAN-2026"

# Write GenBank file
SeqIO.write(record, "output.gb", "genbank")

print("GenBank file created successfully.")
```

Example GenBank Output

```
LOCUS       SEQ001        25 bp    DNA     linear   26-JAN-2026
DEFINITION  This is a sample DNA sequence for GenBank file.
ACCESSION   SEQ001
VERSION     SEQ001
SOURCE      Homo sapiens
  ORGANISM  Homo sapiens
FEATURES             Location/Qualifiers
     source          1..25
ORIGIN
        1 atgcgtacgt tagctagcta ggcta
//
```

Meaning of Key Fields

- **molecule_type** → DNA / RNA / protein
- **topology** →
  - `linear` → chromosomes, contigs
  - `circular` → plasmids, mtDNA
- **annotations** → metadata used by NCBI & tools

What are Sequence Features in GenBank?

Features describe *biological regions* inside a sequence.

Common features:

- `source` → whole sequence
- `gene` → gene region
- `CDS` → coding sequence (protein-coding part)

Each feature has:

- **Location** → start..end position
- **Qualifiers** → gene name, product, etc.

Feature Location (Very Important)

Biopython uses **0-based indexing** internally:

Biological position	Biopython
1	0

Biological position	Biopython
10	9

So:

```
1..15 (GenBank)  →  FeatureLocation(0, 15)
```

Goal

We will create a **GenBank file** with:

- Sequence
- ID & description
- Annotations (molecule_type, topology)
- Features:
  - source (1..30)
  - gene (1..15)
  - CDS (1..15)

Code Create GenBank with Features

```
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord
from Bio.SeqFeature import SeqFeature, FeatureLocation
from Bio import SeqIO

# 1. Create DNA sequence
sequence = Seq("ATGAAACCCGGGTTTAAACCCGGGTTTAAA")

# 2. Create SeqRecord
record = SeqRecord(
    sequence,
    id="GB001",
    name="FeatureExample",
    description="GenBank file with sequence features"
)

# 3. Add required annotations
record.annotations["molecule_type"] = "DNA"
record.annotations["topology"] = "linear"
record.annotations["organism"] = "Escherichia coli"
record.annotations["data_file_division"] = "BCT"
record.annotations["date"] = "26-JAN-2026"

# 4. Create FEATURES

# Source feature (whole sequence)
source_feature = SeqFeature(
    FeatureLocation(0, len(sequence)),
    type="source",
    qualifiers={"organism": ["Escherichia coli"]}
)

# Gene feature (1..15)
gene_feature = SeqFeature(
    FeatureLocation(0, 15),
```

```
    type="gene",
    qualifiers={"gene": ["abcA"]}
)

# CDS feature (1..15)
cds_feature = SeqFeature(
    FeatureLocation(0, 15),
    type="CDS",
    qualifiers={
        "gene": ["abcA"],
        "product": ["sample protein"],
        "codon_start": ["1"]
    }
)

# 5. Add features to record
record.features.append(source_feature)
record.features.append(gene_feature)
record.features.append(cds_feature)

# 6. Write GenBank file
SeqIO.write(record, "feature_example.gb", "genbank")

print("GenBank file with features created successfully.")
```

Generated GenBank File

```
LOCUS       GB001           30 bp    DNA     linear   26-JAN-2026
DEFINITION  GenBank file with sequence features.
ACCESSION   GB001
SOURCE      Escherichia coli
  ORGANISM  Escherichia coli
FEATURES             Location/Qualifiers
     source          1..30
                     /organism="Escherichia coli"
     gene            1..15
                     /gene="abcA"
     CDS             1..15
                     /gene="abcA"
                     /product="sample protein"
ORIGIN
        1 atgaaacccg ggtttaaacc cgggtttaaa
//
```

Feature Types Explained

Feature	Meaning
source	Entire sequence
gene	Gene region
CDS	Protein coding region