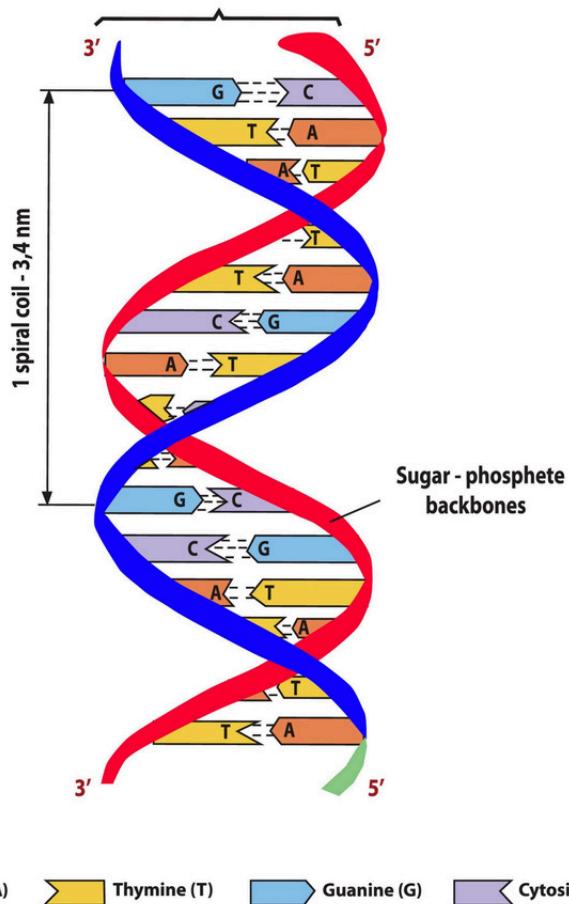




## پروژه: خط لوله جامع تحلیل داده‌های ژنتیکی BioForge

**هدف پروژه:** این پروژه یک چالش برنامه‌نویسی برای ساخت یک خط لوله پردازش داده (Data Pipeline) است. هدف، تبدیل یک ورودی متنی خام به یک گزارش نهایی ساختاریافته از طریق مجموعه‌ای از توابع است که هر کدام وظیفه مشخصی در تبدیل و تحلیل داده‌ها دارند. ما از کانسپت‌های زیستی به عنوان یک سناریوی جذاب برای این چالش استفاده می‌کنیم.

### DNA double helix



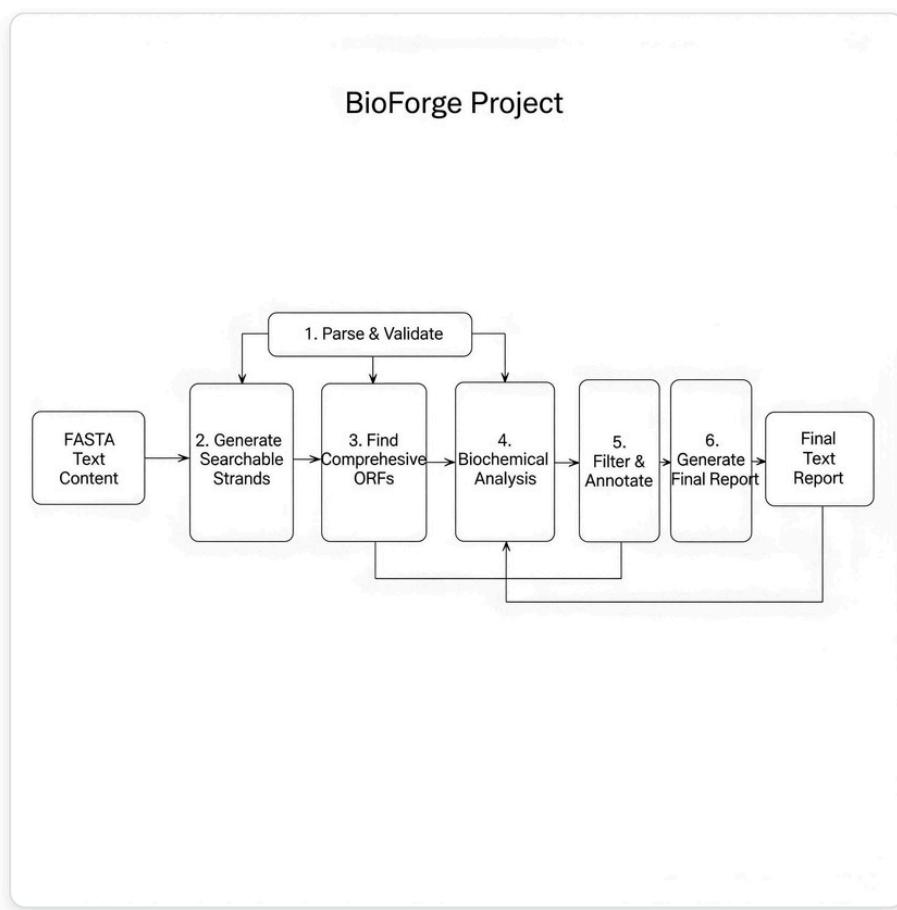
قبل از شروع، باید بینیم این برنامه در دنیای واقعی چه کاربردی دارد:

- **خط لوله بیوانفورماتیک (Bioinformatics Pipeline):** این دقیقاً همان کاری است که برنامه ما انجام می‌دهد. دانشمندان حجم عظیمی از داده‌های ژنتیکی خام (رشته‌های طولانی از A, C, G, T) را از دستگاه‌های توالی‌بایی دریافت می‌کنند. یک خط لوله نرم‌افزاری این داده‌ها را مرحله به مرحله پردازش می‌کند تا اطلاعات مفید، مانند محل ژن‌ها را پیدا کند.
- **وزن مولکولی (MW):** این عدد نشان‌دهنده "جرم" یا "سنگینی" یک پروتئین است. دانشمندان در آزمایشگاه از این عدد برای جداسازی شناسایی پروتئین‌ها (مثلاً با تکنیکی به نام الکتروفورز ژل) استفاده می‌کنند. برنامه ما این ویژگی مهم را برای هر پروتئین کاندید محاسبه می‌کند.

در واقع، برنامه شما یک ابزار نرم‌افزاری قدرتمند می‌سازد که به دانشمندان کمک می‌کند از داده‌های خام ژنتیکی به درک بهتری از پروتئین‌های یک موجود زنده برسند.

## شرح وظایف مراحل پروژه BioForge

پروژه به شش مراحل (تابع) مستقل تقسیم شده است. خروجی هر مراحل، ورودی مراحل بعدی است. در ادامه، وظایف هر مراحل به تفصیل شرح داده شده است.



## ماژول ۱: تابع پردازش و اعتبارسنجی ورودی (Parser & Validator)

### مفهوم علمی

داده‌های خامی که از دستگاه‌های توالی‌بایی DNA به دست می‌آیند، اغلب حاوی خطاهای دستگاهی (کاراکترهای نامفهوم مانند 'N') و فرمتبندی‌های اضافی هستند. این ماژول فرآیند "پاکسازی" داده را شبیه‌سازی می‌کند که اولین قدم در هر تحلیل بیوانفورماتیک واقعی است.

### هدف برنامه‌نویسی و مراحل پیاده‌سازی

هدف این تابع این است که یک رشته ورودی FASTA را دریافت کرده و به یک دیکشنری استاندارد و پاکسازی شده تبدیل کند. این تابع باید قابلیت مدیریت خط را نیز داشته باشد تا ورودی‌های نامعتبر را تشخیص دهد و از ادامه فرآیند با داده‌های معیوب جلوگیری کند.

- اعتبارسنجی نوع و محتوا: ابتدا بررسی شود که ورودی یک رشته است و خالی نیست. همچنین، باید با کاراکتر < شروع شود که نشانگر فرمت FASTA است. در صورت عدم رعایت این موارد، خطای مناسبی (ValueError) یا (TypeError) صادر شود.
- استخراج شناسه (ID): خط اول (هدر) که با < شروع می‌شود، جداسازی شده و قسمت پس از < به عنوان شناسه (ID) استخراج گردد.
- پاکسازی توالی: تمام خطوط بعدی به هم متصل شوند. سپس، تمام کاراکترهای فاصله‌دار و همچنین هر کاراکتری که جزء چهار باز اصلی DNA (A, T, C, G) نیست، حذف شود. توالی نهایی به حروف بزرگ تبدیل گردد.
- ساختار خروجی: نتیجه نهایی به صورت یک دیکشنری با دو کلید 'sequence' و 'id' برگردانده شود.

### مثال ورودی و خروجی

ورودی: یک رشته چندخطی (string)

```
fasta_content = """>test_len_4 | a protein of length 5
AGCT NNN ATGAAA GCATTTGGGTAG GTC"""
```

خروجی: یک دیکشنری (dict)

```
{'id': 'test_len_4 | a protein of length 5', 'sequence': 'AGCTATGAAAGCATTGGGTAGGTC'}
```

نحوه تولید خروجی:

- شناسه test\_len\_4 | a protein of length 5 از خط هدر استخراج می‌شود.
- خطوط توالی به هم متصل و کاراکترهای نامعتبر (NNN و فاصله‌ها) حذف می‌شوند.

## ماژول ۲: تابع تولید رشته‌های مکمل (Complementary String Generator)

### مفهوم علمی

مولکول DNA از دو رشته تشکیل شده است که به صورت مکمل یکدیگر عمل می‌کنند. این مکمل بودن بر اساس قانون جفت‌شدن بازها است: آدنین (A) همیشه با تیمین (T) جفت می‌شود و گوانین (G) همیشه با سیتوزین (C) جفت می‌شود. در سلول، هر دو رشته DNA می‌توانند به عنوان الگو برای ساخت پروتئین‌ها (البته پس از رونویسی به RNA) مورد استفاده قرار گیرند. بنابراین، برای تحلیل جامع پتانسیل پروتئینی یک قطعه DNA، باید هر دو رشته (رشته اصلی و رشته مکمل معکوس) را بررسی کنیم.

### هدف برنامه‌نویسی و مراحل پیاده‌سازی

این تابع یک رشته DNA را گرفته و علاوه بر نگهداری رشته اصلی (forward)، رشته مکمل معکوس (reverse complement) آن را نیز محاسبه و برمی‌گرداند.

1. ذخیره رشته اصلی (Forward): رشته 'sequence' دریافتی از ماژول ۱، به عنوان رشته 'forward' ذخیره می‌شود.
2. ساخت رشته مکمل: برای ساخت رشته مکمل، باید هر باز را به مکمل آن تبدیل کنیم (A به T، T به C، G به G، C به A).

3. معکوس کردن رشته مکمل: رشته مکمل تولید شده در مرحله قبل، باید معکوس (reversed) شود.

4. ساختار خروجی: یک دیکشنری حاوی دو کلید 'reverse\_complement' و 'forward' برگردانده شود.

### مثال ورودی و خروجی

ورودی: رشته 'sequence' از خروجی ماژول ۱

"AGCTATGAAAGCATTGGGTAGGTC"

خروجی: یک دیکشنری (dict)

{'forward': 'AGCTATGAAAGCATTGGGTAGGTC',

'reverse\_complement': 'GACCTACCCAAATGCTTCATAGCT'}

#### نحوه تولید خروجی:

1. رشته ورودی به عنوان مقدار کلید 'forward' ذخیره می‌شود.
2. رشته مکمل ایجاد می‌شود: "TCGATACTTCGTAAACCATCCAG".
3. رشته مکمل معکوس شده و به عنوان مقدار کلید 'reverse\_complement' ذخیره می‌گردد.

## ماژول ۳: موتور شناسایی و استخراج پیام‌های مخفی (ORF Finder & Translator)

### مفهوم علمی

این ماژول هسته بیوانفورماتیک پروژه است و فرآیند مرکزی "بیان ژن" را شبیه‌سازی می‌کند که در آن اطلاعات از DNA به پروتئین تبدیل می‌شود. این فرآیند در دو گام اصلی انجام می‌گیرد:

1. رونویسی (Transcription): در سلول، یک رشته DNA به یک مولکول RNA پیام‌رسان (mRNA) تبدیل می‌شود. تفاوت اصلی در این است که در RNA، باز تیمین (T) با اوراسیل (U) جایگزین می‌شود. رشته‌های RNA سپس به قطعات سه‌تایی به نام \*\*کدون (Codon) خوانده می‌شوند.

2. ترجمه (Translation): ریبوزوم‌ها (ماشین‌های پروتئین‌سازی سلول) رشته mRNA را کدون به کدون می‌خوانند. هر کدون به یک آمینو اسید خاص (واحد سازنده پروتئین) ترجمه می‌شود. این فرآیند از یک \*\*کدون شروع (Start Codon) که همیشه AUG است، آغاز شده و تا رسیدن به یکی از \*\*کدون‌های پایان (Stop Codons) (UAA, UAG, UGA) ادامه می‌یابد. رشته‌ای از آمینو اسیدها که بین یک کدون شروع و یک کدون پایان قرار می‌گیرد، یک \*\*چارچوب خوانش باز (Open Reading Frame - ORF) نامیده می‌شود و کاندیدای یک پروتئین است.

از آنجایی که هیچ سیگنال واضحی برای شروع خواندن در DNA وجود ندارد، ریبوزوم‌ها می‌توانند از سه موقعیت مختلف (سه "چارچوب خوانش" یا "Reading Frame") شروع به خواندن کنند. هر رشته DNA (و مکمل معکوس آن) در واقع دارای \*سه چارچوب خوانش\*\* مختلف است. این تابع تمام این ۶ چارچوب (۳ در رشته forward و ۳ در رشته reverse complement) را برای یافتن ORF‌ها بررسی می‌کند.

### هدف برنامه‌نویسی و مراحل پیاده‌سازی

این تابع باید تمام چارچوب‌های خوانش باز را در هر دو رشته (reverse complement و forward) شناسایی کرده و توالی‌های پروتئینی متناظر با آنها را به همراه اطلاعات تکمیلی (رشته‌ای که پیدا شده و موقعیت شروع) برگرداند.

1. دورسازی رشته‌ها و چارچوب‌ها: برای هر دو رشته 'reverse\_complement' و 'forward' :
- 0 سه چارچوب خوانش مختلف ایجاد کنید (با شروع از اندیس‌های 0, 1 و 2).

◦ هر چارچوب خوانش را از RNA به DNA تبدیل کنید (یعنی تمام 'T' ها را با 'U' جایگزین کنید).

## 2. جستجوی کدون شروع و پایان:

◦ در هر چارچوب RNA، از ابتدا شروع به جستجو برای کدون 'AUG' (کدون شروع) کنید.

◦ هنگامی که یک 'AUG' پیدا شد، شروع به جمع‌آوری آمینو اسیدها کنید (با استفاده از CODON\_MAP) تا زمانی که به یکی از کدون‌های پایان ('UAA', 'UAG', 'UGA') برسید.

◦ نکته مهم: \*اگر رشته قبل از رسیدن به یک کدون پایان تمام شود، یا اگر کدون‌ها کامل (۳ حرفی) نباشند، آن ORF باید بسته شود.

◦ \*\*پروتئین‌های چندگانه:\*\* ممکن است در یک چارچوب خوانش، پس از یک کدون پایان، مجددًا یک کدون شروع 'AUG' دیگر پیدا شود. در این صورت، یک پروتئین جدید شروع می‌شود و باید به عنوان یک ORF جداگانه شناسایی و ذخیره شود.

## 3. ذخیره‌سازی اطلاعات ORF: برای هر ORF پیدا شده، یک دیکشنری شامل:

'sequence' : توالی آمینو اسیدها.

.('reverse\_complement' : نام رشته‌ای که ORF در آن پیدا شده (مثلاً 'forward' یا 'strand')

: موقعیت (اندیس) شروع ORF در رشته DNA اصلی (قبل از تبدیل به RNA و در نظر گرفتن چارچوب). به لیست orf\_candidates اضافه کنید.

4. مدیریت خطای در صورت بروز IndexError: در مثلاً دسترسی به اندیسی که وجود ندارد، هشدار مناسبی صادر شود و فرآیند ادامه یابد.

## مثال ورودی و خروجی

ورودی: دیکشنری خروجی از ماثول ۲ و جدول راهنمای ترجمه

```
strands = {'forward': 'AGCTATGAAAGCATTGGTAGGTC',
           'reverse_complement': 'GACCTACCCAAATGCTTCATAGCT'}
codon_map = # (جدول کدون‌ها در انتهای سند آمده است)
```

خروجی: یک لیست از دیکشنری‌ها

```
[{'sequence': 'MKAFG', 'strand': 'forward', 'start_pos': 4}]
```

## نحوه تولید خروجی:

1. رشته 'forward' دریافت و تمام T ها با U جایگزین می‌شوند.

2. در چارچوب اول (شروع از اندیس ۰)، سیگنال شروع AUG در موقعیت ۴ پیدا می‌شود.

3. رمزگشایی پیام آغاز می‌شود: AUG -> M , AAA -> K , GCA -> A , UUU -> F , GGG -> G

4. کلمه بعدی UAG است که یک سیگنال پایان می‌باشد. فرآیند متوقف می‌شود.

5. پیام نهایی "MKAFG" به همراه اطلاعات موقعیت آن در یک دیکشنری ذخیره شده و به لیست خروجی اضافه می‌شود.

## ماژول ۴: تابع محاسبه‌گر وزن (Weight Calculator)

### مفهوم علمی

هر پروتئین به عنوان یک مولکول فیزیکی، جرم مشخصی دارد. این جرم که "وزن مولکولی" نامیده می‌شود، یکی از پایه‌ایترین ویژگی‌های بیوشیمیایی است و به دانشمندان کمک می‌کند تا پروتئین‌ها را در محیط آزمایشگاه شناسایی و دسته‌بندی کنند.

### هدف برنامه‌نویسی و مراحل پیاده‌سازی

این تابع برای هر آیتم در یک لیست ورودی، یک محاسبه عددی را بر اساس یک جدول جستجو (lookup table) انجام داده و با افزودن نتیجه به داده اصلی، آن را غنی‌سازی (enrich) می‌کند.

۱. یک لیست از دیکشنری‌ها (خروجی ماژول<sup>۳</sup>) را به عنوان ورودی دریافت می‌کند.

۲. برای هر دیکشنری در لیست، مراحل زیر را برای محاسبه وزن مولکولی (MW) انجام می‌دهد:

۳. مرحله اول - جمع وزن‌ها:

○ رشته sequence را از دیکشنری می‌خواند.

○ برای هر کاراکتر در این رشته، مقدار عددی متناظر آن را از "جدول وزن آمینو اسیدها" پیدا کرده و همه را با هم جمع می‌زند.

۴. مرحله دوم - کسر وزن آب:

○ وقتی دو آمینو اسید به هم متصل می‌شوند (پیوند پپتیدی)، یک مولکول آب ( $H_2O$ ) آزاد می‌شود. بنابراین، وزن کل یک پروتئین برابر با مجموع وزن تمام آمینو اسیدهاییش منهای وزن مولکول‌های آبی است که در این فرآیند از دست رفته‌اند.

○ تعداد این مولکول‌های آب همیشه برابر با (تعداد آمینو اسیدها - 1) است. شما باید این مقدار را از مجموع وزن‌ها کم کنید. وزن هر مولکول آب برابر با **18.015** دالتون است.

۵. نتیجه نهایی محاسبه را با کلید mw به دیکشنری اضافه می‌کند.

### مثال ورودی و خروجی

ورودی: لیست خروجی از ماژول<sup>۳</sup>

```
[{'sequence': 'MKAFG', 'strand': 'forward', 'start_pos': 4}]
```

خروجی: همان لیست ورودی با افزودن کلید جدید

```
[{'sequence': 'MKAFG', 'strand': 'forward', 'start_pos': 4, 'mw': 552.69}]
```

## نحوه تولید خروجی:

1. مجموع وزن‌های "M, K, A, F, G" از جدول ۲:  
$$.624.75 = 75.07 + 165.19 + 89.09 + 146.19 + 149.21$$
2. تعداد آمینو اسیدها ۵ است، پس ۴ مولکول آب کسر می‌شود:  $4 * .72.06 = 18.015$
3. نتیجه نهایی:  $552.69 = 624.75 - 72.06$ . این مقدار با کلید 'mw' به دیکشنری اضافه می‌شود.

## ماژول ۵: تابع فیلتر و حاشیه‌نویسی داده (Filter & Annotator)

### مفهوم علمی

در یک ژنوم واقعی، تعداد زیادی ORF کوتاه و بی‌معنی وجود دارد. دانشمندان معمولاً به دنبال ژن‌هایی هستند که پروتئین‌های به اندازه کافی بزرگ و کاربردی تولید می‌کنند. این ماژول این فرآیند فیلتر کردن بیوانفورماتیکی را شبیه‌سازی می‌کند تا فقط کاندیداهای امیدوارکننده باقی بمانند.

### هدف برنامه‌نویسی و مراحل پیاده‌سازی

این تابع داده‌ها را بر اساس مجموعه‌ای از قوانین شرطی دقیق پالایش کرده (filtering) و به رکوردهای معترض، یک شناسه منحصر به فرد و قابل پیش‌بینی اضافه می‌کند (annotation).

1. لیست دیکشنری‌های تحلیل شده را دریافت می‌کند.
  2. یک لیست خالی جدید برای ذخیره نتایج معترض ایجاد می‌کند.
  3. یک متغیر شمارنده (counter) را با مقدار صفر مقداردهی اولیه می‌کند.
  4. در یک حلقه، برای هر دیکشنری در لیست ورودی، دو شرط \*\*دقیق\*\* زیر را بررسی می‌کند:
    - شرط طول: آیا طول توالی (مقدار کلید 'length') بزرگتر یا مساوی ۴ است؟
    - شرط وزن: آیا وزن مولکولی (مقدار کلید 'mw') در بازه ۵۰۰ تا ۲۰۰۰۰۰۰ دالتون قرار دارد؟
  5. اگر یک دیکشنری هر دو شرط بالا را برآورده کند، به عنوان یک کاندیدای معترض شناخته می‌شود:
    - شمارنده یک افزایش می‌یابد.
- ساخت شناسه: یک شناسه منحصر به فرد و قابل پیش‌بینی با فرمت XXX\_BFG\_XXX ساخته می‌شود که XXX شماره ترتیب کاندیدای معترض است (مثلاً BFG\_001, BFG\_002, ...).
- این شناسه با کلید 'protein\_id' به دیکشنری اضافه می‌شود و سپس دیکشنری تکمیل شده به لیست نتایج معترض اضافه می‌گردد.

### مثال ورودی و خروجی

ورودی: لیست خروجی از ماژول ۴

```
[{'sequence': 'MKAFG', 'length': 5, 'mw': 552.69, ...}]
```

خروجی: لیستی که آیتم معتبر را در خود دارد

```
[{'protein_id': 'BFG_001', 'sequence': 'MKAFG', 'length': 5, 'mw': 552.69, ...}]
```

نحوه تولید خروجی:

1. دیکشنری با توالی "MKAFG" بررسی می‌شود.
2. طول آن (5) بزرگتر یا مساوی 4 است. \*\*شرط اول پاس شد\*\*.
3. وزن آن (552.69) در بازه [200000, 500] قرار دارد. \*\*شرط دوم پاس شد\*\*.
4. چون هر دو شرط برقرار است، آیتم \*\*پذیرفته می‌شود\*\*.
5. شمارنده به 1 افزایش یافته و شناسه "BFG\_001" ساخته و به دیکشنری اضافه می‌شود.

## ماژول ۴: تابع تولید گزارش (Report Generator)

### مفهوم علمی

گام نهایی در هر تحلیل علمی، ارائه نتایج به صورت شفاف و سازمان یافته است. این ماژول، داده‌های پردازش شده را به یک فرمت استاندارد و قابل فهم برای انسان تبدیل می‌کند تا به راحتی قابل تفسیر و اشتراک‌گذاری باشد.

### هدف برنامه‌نویسی و مراحل پیاده‌سازی

این تابع داده‌های نهایی و ساختار یافته را به یک خروجی متنی فرمت شده و خوانا برای انسان تبدیل می‌کند.

1. لیست نهایی دیکشنری‌ها را دریافت می‌کند.
2. بررسی می‌کند که آیا لیست خالی است یا نه. اگر خالی بود، یک پیام مناسب بازمی‌گرداند.
3. اگر لیست خالی نبود، با استفاده از تابع `sorted()`، لیست را بر اساس مقدار کلید 'mw' به صورت نزولی مرتب می‌کند.
4. یک رشته هدر برای جدول گزارش ایجاد می‌کند.
5. در یک حلقه، روی لیست مرتب شده پیمایش کرده و برای هر دیکشنری، مقادیر آن را با استفاده از `f-string` ها به صورت ستون‌های مرتب در یک رشته جدید فرمت‌بندی می‌کند.
6. رشته نهایی را که حاوی کل گزارش است، بازمی‌گرداند.

## مثال ورودی و خروجی

ورودی: لیست خروجی از مازول ۵

```
[{'protein_id': 'BFG_001', 'length': 5, 'mw': 552.69, 'strand': 'forward'}]
```

خروجی: یک رشته (string) چندخطی

===== BioForge Analysis Report ======

ID	Length	MW (kDa)	Strand
----	--------	----------	--------

BFG_001	5	0.55	forward
---------	---	------	---------

### نحوه تولید خروجی:

1. لیست ورودی (که یک عضو دارد) دریافت و مرتب می‌شود.
2. یک رشته هدر ساخته می‌شود.
3. مقادیر دیکشنری اول ("BFG\_001", "forward", 5, 552.69 / 1000) با فواصل معین در یک رشته جدید قرار داده شده و به رشته گزارش اضافه می‌شوند.

## یک مثال واقعی از کاربرد پروژه: کشف انسولین

تصور کنید یک دانشمند در حال بررسی ژنوم یک موجود جدید است و شک دارد که این موجود بتواند پروتئینی شبیه به انسولین (هورمون تنظیم کننده قند خون) تولید کند. او قطعه‌ای از DNA این موجود را توالی‌یابی کرده و به عنوان ورودی به برنامه BioForge شما می‌دهد.

خط لوله شما تمام مراحل را طی می‌کند و در نهایت گزارشی تولید می‌کند که در آن یک پروتئین کاندید با شناسه BFG\_042 پیدا شده است. گزارش نشان می‌دهد که وزن مولکولی این پروتئین حدود 5.8 کیلودالتون (kDa) است.

دانشمند با دیدن این عدد هیجان‌زده می‌شود، زیرا وزن مولکولی انسولین انسان نیز دقیقاً در همین حدود است! این یک سرنخ بسیار قوی است که پروتئین کشف شده توسط برنامه شما، به احتمال زیاد نسخه دیگری از انسولین است. این نتیجه، دانشمند را راهنمایی می‌کند تا آزمایش‌های بیوشیمیایی بعدی خود را دقیقاً روی این پروتئین متمرکز کند.

این قدرت واقعی بیوانفورماتیک است: استفاده از الگوریتم‌های هوشمند برای تحلیل داده‌های عظیم، پیدا کردن الگوهای معنادار، و هدایت تحقیقات علمی در دنیای واقعی.

## جداول داده مرجع (Lookup Tables)

**جدول ۱: جدول ترجمه کلمات سه‌حرفی (کدون) به حروف تکی (آمینو اسید) - (CODON\_MAP)**

حروف تکی	کلمات سه‌حرفی	حروف تکی	کلمات سه‌حرفی
S	UCU, UCC, UCA, UCG	F	UUU, UUC
P	CCU, CCC, CCA, CCG	L	UUA, UUG
T	ACU, ACC, ACA, ACG	L	CUU, CUC, CUA, CUG
A	GCU, GCC, GCA, GCG	I	AUU, AUC, AUA
V	GUU, GUC, GUA, GUG	M (START)	AUG
H	CAU, CAC	Y	UAU, UAC
Q	CAA, CAG	STOP	UAA, UAG, UGA
N	AAU, AAC	C	UGU, UGC
K	AAA, AAG	W	UGG
S	AGU, AGC	R	CGU, CGC, CGA, CGG
R	AGA, AGG	D	GAU, GAC
G	GGU, GGC, GGA, GGG	E	GAA, GAG

## جدول ۲: وزن مولکولی متوسط آمینو اسیدها (دالتون)

گروه	آمینو اسید	کد	وزن (Da)	آمینو اسید	کد	وزن (Da)
غیرقطبی	Alanine	A	89.09	Leucine	L	131.17
	Glycine	G	75.07	Methionine	M	149.21
	Isoleucine	I	131.17	Proline	P	115.13
	Valine	V	117.15			
آروماتیک	Phenylalanine	F	165.19	Tryptophan	W	204.23
	Tyrosine	Y	181.19			
	Asparagine	N	132.12	Serine	S	105.09
قطبی	Cysteine	C	121.16	Threonine	T	119.12
	Glutamine	Q	146.15			
	Aspartic acid	D	133.10	Glutamic acid	E	147.13
بازی	Arginine	R	174.20	Lysine	K	146.19
	Histidine	H	155.16			

# PROTEIN STRUCTURE

