Del ADN a la Proteína

Elisa Breeze y Néstor Ortega

Bioinformática

6 Octubre 2025

Ejercicio 1: Replicación del ADN

Objetivo: comprender el mecanismo semiconservativo y las enzimas implicadas.

Instrucciones:
1. Considera la siguiente secuencia de ADN: 5’ – ATG CCG TTA GCT – 3’ / 3’ – TAC GGC AAT CGA – 5’.
2. Realiza una ronda de replicación:

- Identifica las nuevas hebras que se formarán.

  Molécula original:
  5’ – ATGCCGTTAGCT – 3’ (vieja)
  3’ – TACGGCAATCGA – 5’ (vieja)

  Después de la replicación (una ronda):

  Dúplex 1:
  5’ – ATGCCGTTAGCT – 3’ (vieja)
  3’ – TACGGCAATCGA – 5’ (nueva)

  Dúplex 2:
  5’ – ATGCCGTTAGCT – 3’ (nueva)
  3’ – TACGGCAATCGA – 5’ (vieja)

- Indica cuál sería la función de las enzimas helicasa, primasa, ADN polimerasa y ligasa en este proceso.

  Helicasa: Separa las hebras, es decir, desenrolla la doble hélice
  Primasa: Sintetiza pequeños fragmentos de ARN que sirven como cebadores permitiendo que inicien la síntesis de nuevas hebras complementarias. Es decir, da un punto de inicio a la ADN polimerasa.
  ADN polimerasa: Añade nucleótidos complementarios a la hebra molde.
  Ligasa: Une los fragmentos de ADN sintetizados

3. Reflexiona: ¿qué ocurriría si la ADN polimerasa cometiera un error en una base y no se corrigiera?

Si la ADN polimeras cometiera un error en una base y no se corrigiera, se convertiría en una mutación, porque la hebra nueva tendría una base errónea, y en la replicación siguiente se copiará el error, causando una mutación, pudiendo afectar toda la descendencia si no se corrige, volviéndose permanente.

Extensión con Biopython:
Escribe un pequeño script que, dada una cadena de ADN, genere automáticamente su hebra complementaria, y comprueba si tu resultado coincide con lo que obtuviste manualmente.

```
from Bio.Seq import Seq
hebra1 = Seq("ATGCCGTTAGCT")
hebra2 = hebra1.complement()

nueva_hebra1 = hebra1.complement()
print("\nDúplex 1:")
print("5' -", hebra1,       "- 3'    (vieja)")
print("3' -", nueva_hebra1,  "- 5'    (nueva)")

nueva_hebra2 = hebra2.complement()     # devuelve la secuencia de
la hebra superior
print("\nDúplex 2:")
print("5' -", nueva_hebra2, "- 3'    (nueva)")
print("3' -", hebra2,  "- 5'    (vieja)")
```

```
Dúplex 1:
5' - ATGCCGTTAGCT - 3'   (vieja)
3' - TACGGCAATCGA - 5'   (nueva)

Dúplex 2:
5' - ATGCCGTTAGCT - 3'   (nueva)
3' - TACGGCAATCGA - 5'   (vieja)

Process finished with exit code 0
```

Ejercicio 2: Transcripción del ADN a ARN

Objetivo: traducir correctamente la información de la cadena molde.

Instrucciones:
1. Usa la siguiente secuencia de ADN: 5’ – ATG CCT GAA TGC – 3’ / 3’ – TAC GGA CTT ACG – 5’.
2. Identifica cuál es la cadena molde.

La hebra molde va de 3’ a 5’, por lo que sería: 3’ – TAC GGA CTT ACG –5’

3. Obtén el transcrito de ARN correspondiente (recuerda que el ARN se sintetiza en dirección 5’ → 3’ y utiliza uracilo en lugar de timina).

El transcrito de ARN correspondiente sería: 5’- AUG CCU GAA UGC-3’

4. Explica cuál sería la región promotora y cuál la región codificante.

Región promotora se suele encontrar en el extremo 5’ de la cadena codificante, sirve como sitio de unión para la ARN polimerasa y para los factores de transcripción. En este caso no

se ve ninguna región promotora. La región codificante por otro lado, es la parte del ADN que se transcribe a ARNm y luego se puede traducir en proteína.

La **región promotora:** En este caso no hay región promotora

La **región codificante** sería: 5’ – ATG CCT GAA TGC – 3

Extensión con Biopython:
Crea un script que lea un archivo FASTA con ADN y produzca la secuencia de ARNm. Experimenta cambiando la orientación de la hebra y observa qué ocurre.

Se observa que las dos secuencias de ARNm resultantes son completamente distintas, lo cual nos muestra que la direccionalidad 5’→3’ y la hebra molde correcta son esenciales en la transcripción genética, porque cambiar la orientación de la hebra implica transcribir una cadena distinta, que no produce el mismo ARNm ni codifica la misma proteína.

```python
from Bio import SeqIO

for secuencia in SeqIO.parse("gene.fna", "fasta"):
    dna = secuencia.seq

    print("Secuencia de ADN original (" + secuencia.id + "): ")
    print(dna)

    # Transcripción a ARNm
    arnm = dna.transcribe()
    print("\n Secuencia de ARNm:")
    print(arnm)

    # Cambio de la orientación de la hebra => transcripción de la hebra complementaria inversa
    arnm_complementaria = dna.reverse_complement().transcribe()
    print("\n Cambio Orientación: \n Secuencia de ARNm (hebra complementaria inversa):")
    print(arnm_complementaria)
    print ("\n")
```

Parte del resultado que obtuvimos:

```
Secuencia de ADN original (NC_000017.11:c7687490-7668421):
CTCAAAAGTCTAGAGCCACCGTCCAGGGAGCAGGTAGCTGCTGGGCTCCGGGGACACTTTGCGTTCGGGCTGGGAGCGTGCTTTCCACGACGGTGACACGCTTCCCTGGATTGGGT

 Secuencia de ARNm:
CUCAAAAGUCUAGAGCCACCGUCCAGGGAGCAGGUAGCUGCUGGGCUCCGGGGACACUUUGCGUUCGGGCUGGGAGCGUGCUUUCCACGACGGUGACACGCUUCCCUGGAUUGGGU

 Cambio Orientación:
 Secuencia de ARNm (hebra complementaria inversa):
UGGCAGCAAAGUUUUAUUGUAAAAUAAGAGAUCGAUAUAAAAAUGGGAUAUAAAAAGGGAGAAGGAGGGGAAGGGUGGGGUGAAAAUGCAGAUGUGCUUGCAGAAUGUAAAAGAUG


Secuencia de ADN original (NC_060941.1:c7591594-7572544):
CTCAAAAGTCTAGAGCCACCGTCCAGGGAGCAGGTAGCTGCTGGGCTCCGGGGACACTTTGCGTTCGGGCTGGGAGCGTGCTTTCCACGACGGTGACACGCTTCCCTGGATTGGGT

 Secuencia de ARNm:
CUCAAAAGUCUAGAGCCACCGUCCAGGGAGCAGGUAGCUGCUGGGCUCCGGGGACACUUUGCGUUCGGGCUGGGAGCGUGCUUUCCACGACGGUGACACGCUUCCCUGGAUUGGGU

 Cambio Orientación:
 Secuencia de ARNm (hebra complementaria inversa):
UGGCAGCAAAGUUUUAUUGUAAAAUAAGAGAUCGAUAUAAAAAUGGGAUAUAAAAAGGGAGAAGGAGGGGAAGGGUGGGGUGAAAAUGCAGAUGUGCUUGCAGAAUGUAAAAGAUG
```

Ejercicio 3. Traducción del ARNm a proteína

Objetivo: aplicar el código genético y reflexionar sobre mutaciones.

Instrucciones:
1. Usa el siguiente transcrito de ARN: 5’ – AUG UAU GCU UAA – 3’.
2. Identifica el codón de inicio y el codón de paro.

Inicio**: AUG** → Metionina (Met) y señal de inicio.

Paro**: UAA** → codón STOP (no codifica aminoácido).

3. Traduce la secuencia en una cadena de aminoácidos.

AUG → **Met** (Metionina)

UAU → **Tyr** (Tirosina)

GCU → **Ala** (Alanina)

UAA → **STOP** (Codón de terminación, no codifica aminoácido)

Proteína resultante: Met – Tyr – Ala

Abreviatura: MYA

4. Reflexiona: ¿qué pasaría si el codón de inicio mutara de AUG a GUG? ¿Qué ocurriría si el codón de paro desapareciera por mutación?

- **AUG → GUG (inicio mutado):** normalmente no inicia la traducción (AUG es la señal estándar). En algunos sistemas GUG puede iniciar con menor eficiencia; podría no producirse la proteína o producirse muy poco.

- **Desaparece el STOP (UAA muta):** la traducción continúa hasta el siguiente STOP aguas abajo → proteína más larga (extendida), con riesgo de pérdida de función, mal plegamiento o toxicidad.

Extensión con Biopython:
Utiliza el módulo Bio.Seq para traducir automáticamente el ARNm a una secuencia proteica y comprueba si el resultado coincide con el tuyo.

```
from Bio.Seq import Seq
arn = Seq("AUGUAUGCUUAA")
proteina = arn.translate(to_stop=True)
print("Proteína:", proteina)
```

```
Proteína: MYA

Process finished with exit code 0
```

Ejercicio 4. Splicing alternativo

Objetivo: comprender cómo un mismo gen puede generar varias proteínas.

Instrucciones:
1. Considera un gen con 5 exones: Exón 1 – Exón 2 – Exón 3 – Exón 4 – Exón 5.
2. Diseña al menos dos combinaciones de splicing alternativo (por ejemplo, 1-2-4-5 o 1-3-5).

Isoforma A (1–2–3–5)

- Se omite el **Exón 4** (la parte catalítica).
- La proteína resultante no tiene actividad enzimática, pero aún puede unirse a otras proteínas gracias a los exones 2 y 3.
- Podría actuar como proteína reguladora o inhibidora, bloqueando la acción de otras isoformas.

Isoforma B (1–2–4–5)

- Se elimina el **Exón 3** (regulador).
- Mantiene la parte catalítica (Exón 4), por lo que es activa.
- Sin el dominio regulador, podría ser más activa o menos controlada, aumentando la actividad enzimática.

3. Explica qué diferencias esperarías en las proteínas resultantes.

La **isoforma A** (1–2–3–5) pierde el exón 4, por lo que no tiene actividad enzimática, aunque puede unirse a otras proteínas y actuar como reguladora. En cambio, la **isoforma B** (1–2–4–5) conserva el exón 4 (la parte activa), pero pierde el exón 3, que servía para controlar la actividad. Así, la isoforma A es inactiva pero reguladora, mientras que la B es activa pero menos controlada.

4. Reflexiona: ¿por qué este mecanismo aumenta la diversidad proteica sin necesidad de más genes?

El splicing alternativo permite que un solo gen genere múltiples ARNm (y por tanto varias proteínas distintas). No hace falta más genes; basta con reordenar o excluir exones durante el procesamiento del ARN.

Extensión con Biopython / bases de datos:
Busca en Ensembl un gen humano conocido con isoformas (por ejemplo, FGFR2) y compara sus diferentes transcritos. Reflexiona sobre cómo estas diferencias podrían afectar a la función de la proteína.

```
import re
from Bio import SeqIO
from Bio.Seq import Seq
from Bio.Align import PairwiseAligner

FASTA = "Homo_sapiens_FGFR2_sequence.fa"

# Cargar y filtrar CDS o cDNA
buckets = {"cds": [], "cdna": []}
for r in SeqIO.parse(FASTA, "fasta"):
   d = (r.description or r.id).lower()
   tipo = "cds" if "cds" in d else ("cdna" if "cdna" in d else
None)
   if not tipo: continue
   iso = re.search(r"\bFGFR2-\d+\b", r.description)
   r.id = (iso.group(0) if iso else r.id.split()[0])
   r.seq = Seq(str(r.seq).upper().replace("U", "T"))
   buckets[tipo].append(r)

# Deduplicar (más larga por isoforma)
def dedupe(recs):
   best = {}
   for r in recs:
       if r.id not in best or len(r.seq) > len(best[r.id].seq):
           best[r.id] = r
   return list(best.values())

cds = dedupe(buckets["cds"]) or dedupe(buckets["cdna"])
if len(cds) < 2:
   raise SystemExit("Necesito ≥2 isoformas (CDS o cDNA).")

# Alineador global
aln = PairwiseAligner()
aln.mode, aln.match_score, aln.mismatch_score = "global", 1, 0
aln.open_gap_score, aln.extend_gap_score = -1, -0.5

# Función para calcular identidad
def pct_id(seq1, seq2):
   a = aln.align(seq1, seq2)[0]
```

```
    A, B, co = str(a.target), str(a.query), a.coordinates
    s1 = s2 = ""
    for k in range(co.shape[1] - 1):
        a0, a1, b0, b1 = map(int, (co[0, k], co[0, k + 1], co[1,
k], co[1, k + 1]))
        if a1 > a0 and b1 > b0:
            s1 += A[a0:a1]; s2 += B[b0:b1]
        elif a1 > a0:
            s1 += A[a0:a1]; s2 += "-" * (a1 - a0)
        else:
            s1 += "-" * (b1 - b0); s2 += B[b0:b1]
    return 100 * sum(x == y for x, y in zip(s1, s2)) / len(s1)


print("Comparaciones por pares (nucleótidos):")
for i in range(len(cds)):
    for j in range(i + 1, len(cds)):
        pid = pct_id(cds[i].seq, cds[j].seq)
        print(f"{cds[i].id} vs {cds[j].id}: {pid:.2f}% | len
{len(cds[i])}/{len(cds[j])}")


prot = {r.id: r.seq.translate(to_stop=True) for r in cds}
print("\nComparaciones por pares (proteínas):")
for i in range(len(cds)):
    for j in range(i + 1, len(cds)):
        pid = pct_id(prot[cds[i].id], prot[cds[j].id])
        print(f"{cds[i].id} vs {cds[j].id}: {pid:.2f}% | aa
{len(prot[cds[i].id])}/{len(prot[cds[j].id])}")
```

```
Comparaciones por pares (nucleótidos):
FGFR2-206 vs FGFR2-203: 99.76% | len 2466/2460
FGFR2-206 vs FGFR2-210: 90.59% | len 2466/2304
FGFR2-203 vs FGFR2-210: 90.34% | len 2460/2304

Comparaciones por pares (proteínas):
FGFR2-206 vs FGFR2-203: 99.76% | aa 821/819
FGFR2-206 vs FGFR2-210: 90.29% | aa 821/768
FGFR2-203 vs FGFR2-210: 90.05% | aa 819/768

Process finished with exit code 0
```

Se compararon las isoformas **FGFR2-206, -203 y -210**. Las comparaciones a nivel de **nucleótidos** muestran que las isoformas **FGFR2-206** y **FGFR2-203** presentan una identidad muy alta (99.76 %), lo que indica que solo difieren en pequeños fragmentos de la secuencia —probablemente por la inclusión o exclusión de un exón corto—, manteniendo así casi la misma información genética. En cambio, la isoforma **FGFR2-210** muestra una

identidad menor (alrededor del 90 %) y una longitud más corta, lo que sugiere la pérdida de uno o varios exones que podrían codificar regiones funcionales importantes.
A nivel de **proteína**, las diferencias siguen el mismo patrón: FGFR2-206 y FGFR2-203 son prácticamente idénticas (99.76 % de identidad, 821 y 819 aminoácidos), mientras que FGFR2-210 es más corta (768 aminoácidos) y conserva solo un 90 % de similitud. Estas variaciones estructurales indican que, aunque las tres isoformas derivan del mismo gen, **FGFR2-210 probablemente tiene una función distinta**, debida a la alteración o pérdida de dominios proteicos implicados en la unión a ligandos o en la señalización celular.

Ejercicio 5. Introducción a las proteínas

Objetivo: relacionar secuencia, estructura y función.
Instrucciones:
1. Considera la siguiente secuencia de aminoácidos: Met – Ile – Ser – Gly – Val – Lys – His.
2. Identifica el extremo N y el extremo C de la cadena.

Extremo N: Primer aminoácido de la cadena.
Primer aminoácido: Met

Extremo C: El último ácido de la cadena.
Último aminoácido => His

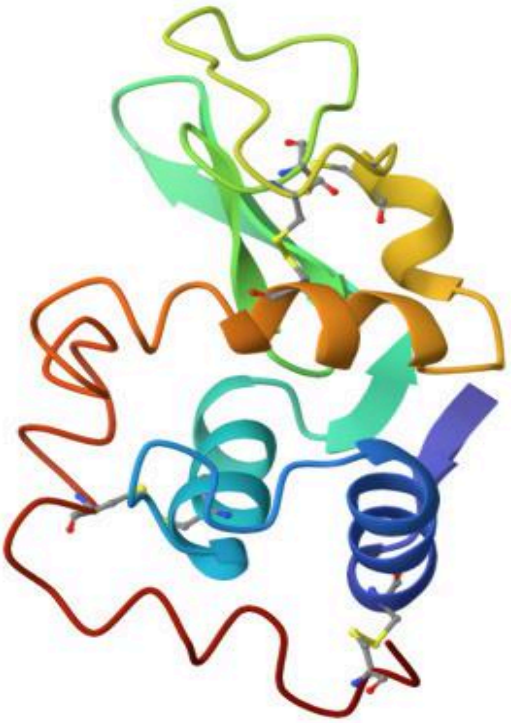
3. Reflexiona: ¿cómo influye el orden de los aminoácidos en la estructura final de la proteína? ¿Qué ocurriría si hubiera una mutación que cambiara un aminoácido hidrofóbico por uno hidrofílico en una región interna de la proteína?

El orden es muy importante, ya que determina cómo se pliega y determina la función de la proteína. Cada aminoácido tiene propiedades químicas específicas y estas determinan el modo en el que se pliega la cadena. Esto a su vez define la estructura final y la función. Si hay una mutación que cambia un aminoácido hidrofóbico (repelente al agua) por uno hidrofílico (atraído por el agua) en una región interna de la proteína, podría desestabilizarse el núcleo y llevar a que no se pliegue correctamente, pudiendo afectar su función, e incluso llevar a que se agregue a otras proteínas o degrade.

Extensión con Bioinformática:
Busca en el Protein Data Bank (PDB) la estructura de una proteína conocida y observa cómo los aminoácidos se organizan en hélices alfa y láminas beta. Reflexiona sobre cómo una mutación puntual podría afectar al plegamiento:

La lisozima es una proteína que daña las paredes celulares de las bacterias, actuando así como una barrera natural contra infecciones.
En la imagen se puede observar su estructura, donde las hélices alfa son las partes enrolladas en forma de muelle, y las láminas beta son las partes más planas y extendidas. Las partes que parecen ganchos, son puentes disulfuro, que se encargan de mantener la forma estable.

Si se cambia un aminoácido, puede tener un efecto importante, pudiendo perderse la estabilidad e incluso llegar a deformarse si se cambia un aminoácido que forma un puente disulfuro. Por otro lado, si se cambian otros aminoácidos podría llegar a deshacerse, romperse o incluso afectar las interacciones con otras partes o con el entorno. En conclusión, una pequeña mutación podría afectar gravemente cómo se pliega, pudiendo cambiar su forma, haciéndola menos estable y que no funcione correctamente.



Ejercicio 6. Actividad integradora: del ADN a la proteína

Objetivo: recorrer el dogma central completo.

Instrucciones:
1. Escoge una secuencia de ADN (codificada en un fichero fasta de algún banco de datos o página especializada públicos).
2. Paso 1: Replica la secuencia indicando las dos nuevas hebras.
3. Paso 2: Transcribe la cadena molde en ARNm.
4. Paso 3: Traduce el ARNm en una cadena de aminoácidos.

```
with open("gene.fna", "r") as f:
   lineas = f.readlines() # leer todas las líneas

adn = ''.join([l.strip().upper() for l in lineas if not l.startswith('>')]) #
descartamos encabezados FASTA (empiezan con >)

print(f"Secuencia original:")
```

```
print(adn)
print("\n")

# Replicación (generación de hebras complementarias)
pares = {'A': 'T', 'T': 'A', 'C': 'G', 'G': 'C'} # Diccionario con bases
complementarias
complementario = ''.join([pares[b] for b in adn])          # Hebra complementaria
molde = complementario[::-1]                               # Hebra molde
(Invirtiendo hebra complementaria)

print("Paso 1: Replicación")
print(f"Hebra complementaria (5'->3'): {complementario}")
print(f"Hebra molde (3'->5'):          {molde}")
print("\n")

# Transcripción (ADN a ARNm)
arnm = molde.replace('T', 'U') # Convertir a Secuencia ARNm
print("Paso 2: Transcripción")
print(f"ARNm: {arnm}")
print("\n")

# Traducción (ARNm a proteína)
codones = {
   'UUU':'F','UUC':'F','UUA':'L','UUG':'L',
   'UCU':'S','UCC':'S','UCA':'S','UCG':'S',
   'UAU':'Y','UAC':'Y','UAA':'STOP','UAG':'STOP',
   'UGU':'C','UGC':'C','UGA':'STOP','UGG':'W',

   'CUU':'L','CUC':'L','CUA':'L','CUG':'L',
   'CCU':'P','CCC':'P','CCA':'P','CCG':'P',
   'CAU':'H','CAC':'H','CAA':'Q','CAG':'Q',
   'CGU':'R','CGC':'R','CGA':'R','CGG':'R',

   'AUU':'I','AUC':'I','AUA':'I','AUG':'M',
   'ACU':'T','ACC':'T','ACA':'T','ACG':'T',
   'AAU':'N','AAC':'N','AAA':'K','AAG':'K',
   'AGU':'S','AGC':'S','AGA':'R','AGG':'R',

   'GUU':'V','GUC':'V','GUA':'V','GUG':'V',
   'GCU':'A','GCC':'A','GCA':'A','GCG':'A',
   'GAU':'D','GAC':'D','GAA':'E','GAG':'E',
   'GGU':'G','GGC':'G','GGA':'G','GGG':'G'
}

arnm = arnm.upper()
proteina = ""

for i in range(0, len(arnm), 3): # Recorrer ARNm de 3 en 3 (codones)
   codon = arnm[i:i+3] # Extraer el codón
   if len(codon) < 3: # Si queda menos de 1 codón entero, se para
       break
   aa = codones.get(codon, '') # Obtenemos aminoácido y lo añadimos a la proteína
   if aa == 'STOP': # si se trata de un codón de parada, se para
       break
   proteina += aa

print("Paso 3: Traducción")
```

```
print(f"Proteína traducida: {proteina}")
print("\n=================================================\n")
```

Parte del Resultado:

```
Secuencia original:
CTCAAAAGTCTAGAGCCACCGTCCAGGGAGCAGGTAGCTGCTGGGCTCCGGGGACACTTTGCGTTCGGGCTGGGAGCGTGCTTTCCA

Paso 1: Replicación
Hebra complementaria (5'->3'): GAGTTTTCAGATCTCGGTGGCAGGTCCCTCGTCCATCGACGACCCGAGGCCCCTGT
Hebra molde (3'->5'):          TGGCAGCAAAGTTTTATTGTAAAATAAGAGATCGATATAAAAATGGGATATAAAAAG

Paso 2: Transcripción
ARNm: UGGCAGCAAAGUUUUAUUGUAAAAUAAGAGAUCGAUAUAAAAAUGGGAUAUAAAAAGGGAGAAGGAGGGGAAGGGUGGGGU

Paso 3: Traducción
Proteína traducida: WQQSFIVK
```

5. Reflexiona: ¿qué punto del proceso es más vulnerable a errores que afecten a la función de la proteína?

El punto del proceso más vulnerable es la traducción, dado que es cuando se determina la secuencia de aminoácidos que forma la proteína, y si hay un error en este paso, se podría alterar su estructura, y por ende su función biológica también. Además de ello, la replicación también es muy importante porque cualquier error ocurrido en este paso se puede transmitir a las células hijas. Por otro lado, los errores durante la transcripción son menos impactantes, ya que los ARNm defectuosos no se heredan.

6. Programa un pipeline que realice los tres procesos (replicación, transcripción, y traducción.

Extensión con Biopython:
El pipeline que realice los tres pasos: generar la hebra complementaria (replicación), obtener el transcrito (transcripción) y traducir a proteína (traducción) debe informar en todo momento de lo que está ocurriendo en los procesos.

```python
from Bio.Seq import Seq
from Bio import SeqIO

for secuencia in SeqIO.parse("gene.fna", "fasta"):
    dna = secuencia.seq
    print(f"Secuencia original ({secuencia.id}):")
    print(dna)
    print("\n")

    # Replicación (generación de hebras complementarias)
    complementario = dna.complement()
    molde = dna.reverse_complement()
    print("Paso 1: Replicación")
    print(f"Hebra complementaria (5'->3'): {complementario}")
    print(f"Hebra molde (3'->5'):          {molde}")
    print("\n")

    # Transcripción (ADN a ARNm)
    arnm = molde.transcribe()
    print("Paso 2: Transcripción")
    print(f"ARNm: {arnm}")
    print("\n")

    # Traducción (ARNm a proteína)
    proteina = arnm.translate(to_stop=True) # Solo hasta el codón Stop
    print("Paso 3: Traducción")
    print(f"Proteína traducida: {proteina}")
```

Parte del resultado:

```
Secuencia original (NC_000017.11:c7687490-7668421):
CTCAAAAGTCTAGAGCCACCGTCCAGGGAGCAGGTAGCTGCTGGGCTCCGGGGACACTTTGCGTTCGGGCTGGGAGCGTGCTTTCCACGACGGTGACACGCTTCCCTGGATTGGGTAAGCTCCTGACTGAACTTGATGAGTCCTCTCTGAC


Paso 1: Replicación
Hebra complementaria (5'->3'): GAGTTTTCAGATCTCGGTGGCAGGTCCCTCGTCCATCGACGACCCGAGGCCCCTGTGAAACGCAAGCCCGACCCTCGCACGAAAGGTGCTGCCACTGTGCGAAGGGACCTAACCCATTCC
Hebra molde (3'->5'):          TGGCAGCAAAGTTTTATTGTAAAATAAGAGATCGATATAAAAATGGGATATAAAAAGGGAGAAGGAGGGGAAGGGTGGGGTGAAAATGCAGATGTGCTTGCAGAATGTAAAAGATGTTGAC


Paso 2: Transcripción
ARNm: UGGCAGCAAAGUUUUAUUGUAAAAUAAGAGAUCGAUAUAAAAAUGGGAUAUAAAAAGGGAGAAGGAGGGGAAGGGUGGGGUGAAAAUGCAGAUGUGCUUGCAGAAUGUAAAAGAUGUUGACCCUUCCAGCUGGACGUGGUGGCUC


Paso 3: Traducción
Proteína traducida: WQQSFIVK
```