Bioinformática

Introducción

Rodrigo Santamaría

Introducción

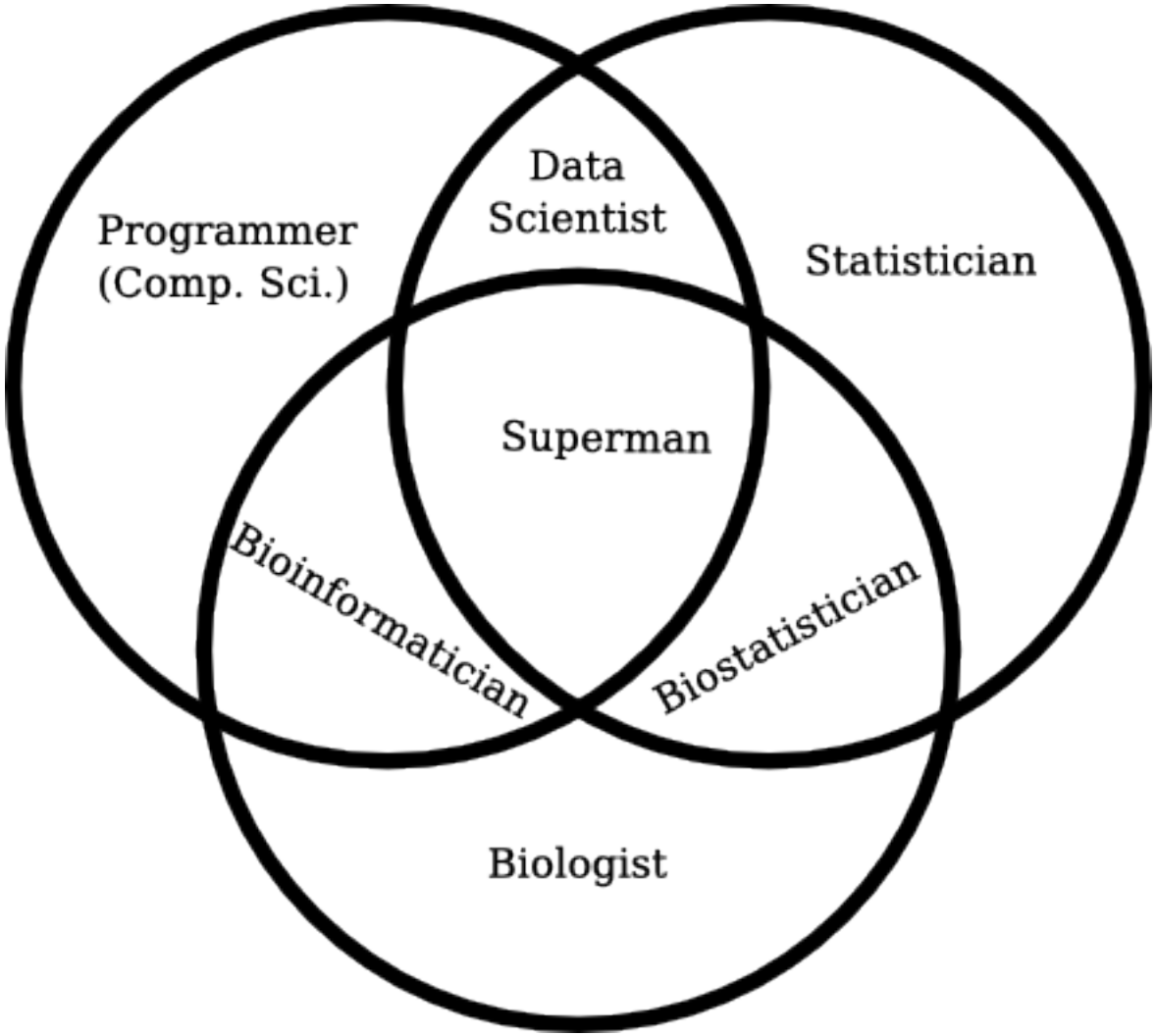
- **Bioinformática: ¿qué es?**
  - **Actores**
- Biología básica
  - Genomas
  - Genes

Bioinformática: ¿qué es?

Actores

Bioinformática

- Aplicación de métodos informáticos para analizar datos biológicos
  - Término difuso
    - Uso de métodos
    - Implementación de métodos
    - Diseño de métodos
  - Área interdisciplinar:
    - Biología
    - Ciencia de la computación
    - Estadística



http://blog.fejes.ca/?p=2418

Introducción

- Bioinformática: ¿qué es?
  - Actores
- **Biología básica**
  - **Genomas**
  - **Genes**

Biología básica: Genomas

estructura del DNA

cadenas, archivos, funciones y listas

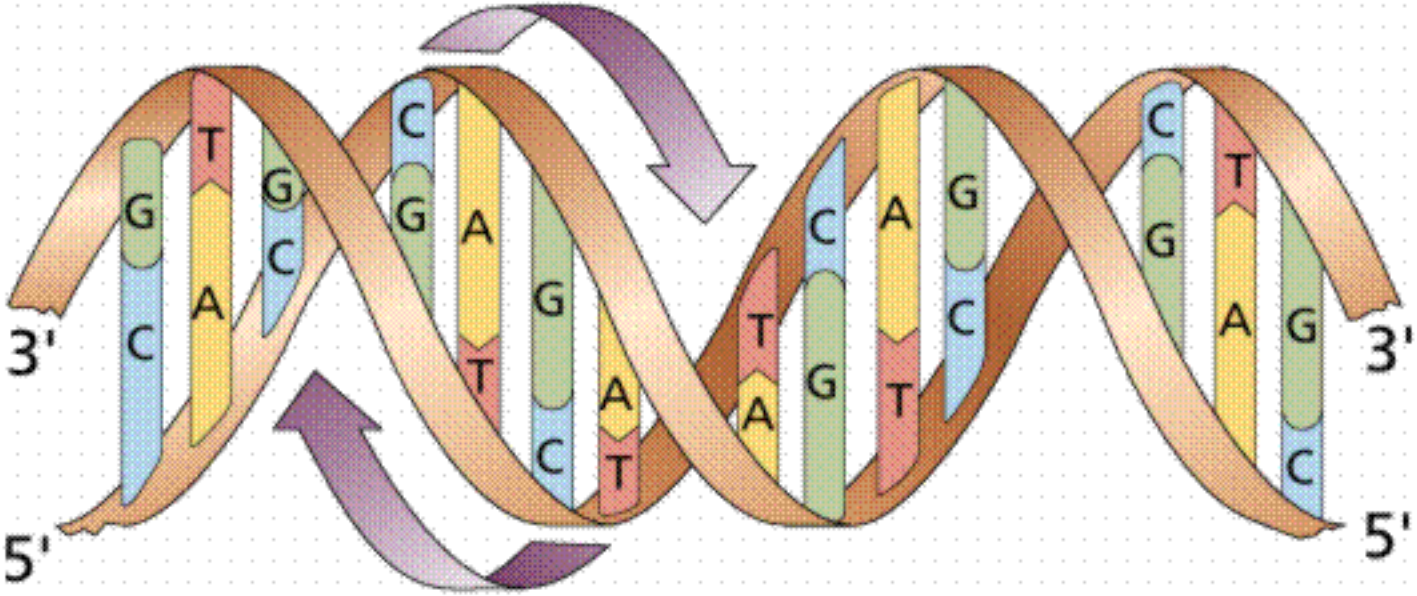
Biología básica

- Para diseñar métodos útiles debemos ser capaces de entender los problemas biológicos
  - Al menos de un modo básico

- La biología básica no es difícil
  - Hay mucha terminología
  - Pero es fundamentalmente descriptiva
    - No hay teoremas o demostraciones complicadas

Genoma

- Material genético de un organismo
  - Codificado como **DNA**
    - Cadena doble de **nucleótidos** (A, T, C, G)
      - 3’ a 5’ (antisentido o upstream)
      - 5’ a 3’ (sentido o downstream)



Sobre los extremos 3’ y 5’ del DNA
https://www.youtube.com/watch?v=p835L4HWH68

Python

- Veremos a continuación algunos aspectos de la sintaxis de Python:
  - Tratamiento de cadenas, archivos y listas
  - Estructuración en funciones y bloques
  - Control de flujo

- Chuleta: http://vis.usal.es/rodrigo/documentos/bioinfo/avanzada/pythonCheatsheet.pdf

Cadenas (str)

```
oric="atcaatgatcaacgtaagcttctaagcatgatcaaggtgctcacacagtttatccacaacctgagtggatg
acatcaagataggtcgttgtatctccttcctctcgtactctcatgaccacggaaagatgatcaagagaggatgattt
cttggccatatcgcaatgaatacttgtgacttgtgcttccaattgacatcttcagcgccatattgcgctggccaagg
tgacggagcgggattacgaaagcatgatcatggctgttgttctgtttatcttgttttgactgagacttgttaggata
gacggtttttcatcactgactagccaaagccttactctgcctgacatcgaccgtaaattgataatgaatttacatgc
ttccgcgacgatttacctcttgatcatcgatccgattgaagatcttcaattgttaattctcttgcctcgactcatag
ccatgatgagctcttgatcatgtttccttaaccctctatttttacggaagaatgatcaagctgctgctcttgatca
tcgtttc"
```

```
oric          #valor
type(oric)    #tipo
len(oric)     #longitud
oric[3]       #acceso
oric[0]       #ojo con el comienzo y fin
oric[len(oric)]

oric[3:7]     #acceso a intervalo
```

`oric` contiene la secuencia de nucleótidos que indica un **origen de replicación** en *Vibrio cholerae*, la bacteria patógena que causa el cólera

Curiosamente, iremos viendo cómo los orígenes de replicación se pueden descubrir mediante un ordenador, sin necesidad del laboratorio!

Archivos

- Como os imaginaréis, no es operativo copiar y pegar toda la secuencia de un genoma* en el programa
  - Recurrimos a la lectura de archivos

```
f=open("/home/rodri/data/genomas/Vibrio_cholerae.txt", "r")
vc=f.readlines()   #lee las líneas del archivo a una lista
vc=vc[0]           #el fichero de genoma sólo tiene una línea
len(vc[0])         #(muy larga, claro)
#vc     #No se os ocurra mostrarla por pantalla, es enooorme!
f.close()      #Cuando salís de casa, cerrad con llave
```

*Podéis encontrar varias secuencias de genomas completos aquí:
http://vis.usal.es/rodrigo/documentos/bioinfo/avanzada/genomas

Archivos FASTA

- Formato común para almacenar secuencias de nucleótidos
  - Cada secuencia corresponde a dos líneas

```
>orf19.2163 orf19.2163 CGDID:CAL0005388
ATGCTGGAAGAAGAAGTTCACGACACGTCATCAGAAGCAAGTGAGGTTTTCACCAACCAG
```

  - Ejemplo:
    - http://vis.usal.es/rodrigo/documentos/bioinfo/avanzada/datos/oric.txt

Funciones

- Sintaxis

```
#Conviene añadir una descripción
def nombreFuncion(argumentos):
    linea1
    …
    linea n
    return valor
```

```
#añade el IVA 'normal'
def aplicarIVA(factura):
    factura *=1.21
    print(factura)
    return factura
```

  - OJO: en Python no tenemos llaves, se usa el sangrado para determinar que el código está dentro de la función
    - Será igual para los bloques `if`, `for`, `while`, etc.

Funciones (II)

- Llamada a funciones

```
factura=aplicarIVA(35.50)
print(factura)

factura=aplicarIVA()  #dará un error ya que 'espera' un argumento
```

- Podemos definir y usar varios argumentos, separados por comas, o no poner ninguno
- Podemos añadir valor por defecto a un argumento

```
def aplicarIVA(factura=10):
```

Bloques de código

- Para organizar vuestro código de la asignatura recomiendo:
  1. Crear un archivo .py por sesión
  2. Delimitar cada ejercicio por bloques con **#%%**
     - Interpretados por Spyder, permite ejecutar todas las líneas de código de un bloque sin ejecutar el archivo entero

```
#%% Este es el bloque 1
blablabla
tralaraliri
...
#%% Este es el bloque 2
...
```

Listas

```
listaVacia=[]
wild_animals=[“perezoso”, “ornitorrinco”, “lemur”]
bonoLoto=[3,0,8,1,4,99]
```

```
wild_animals[2]
bonoLoto[3] + bonoLoto[0]
print(“animal {}, número {}”.format(wild_animals[1],
                                    bonoLoto[1]))
wild_animals[2]=“tigre”

del bonoLoto[4] #elimina el elemento en la posición 4
```

```
len(wild_animals)
wild_animals.append(“elefante”) #adición al final
len(wild_animals)
```

Listas (II)

```
wild_animals.index(“tigre”)
wild_animals.insert(3, “gorila”) #inserta en posición 3
wild_animals

oric.index(“atcg”)  #también funciona con cadenas!
```

```
wild_animals[1:2]   #tomamos el intervalo [1:2)
bonoLoto[2:5]
wild_animals[:3] #si no ponemos nada, hasta inicio o fin
oric[500:]       #funciona con cadenas!
```

```
#recorremos la lista elevando al cuadrado cada elemento x
for x in bonoLoto:  #aprenderemos más sobre for pronto!
    print(x*x)

for base in oric:   #también funciona con cadenas!
    print(base)
```

Control de flujo

- Similar a C/Java
  - Mismas comparaciones (`>`, `<`, `<=`, `>=`, `==`, `!=`)
  - Operadores booleanos: `and`, `or`, `not`
  - Sin paréntesis → con `:` al final
  - Sin llaves → con sangrado
  - `else...if` se contrae a `elif`

```
if 8 < 9:
  print(“¡Ocho es menor que nueve!”)
elif flipando==True:
  print(“OMG: ocho es MAYOR que nueve!”)
else:
 print(“Falsa alarma, estábamos flipando”)
```

1 punto

Ejercicio 1

- Implementar una función `complementaria`
  - Entrada:
    - `seq`: secuencia de nucleótidos
  - Retorno: cadena complementaria a `seq`
- Ejemplo: ACGTTCGT → ACGAACGT
- Prueba:
  - `seq`:
    - Tomar como `seq` los 20 nucleótidos que están en el centro del genoma de *V cholerae*
    - http://vis.usal.es/rodrigo/documentos/bioinfo/avanzada/genomas/Vibrio_cholerae.txt

La cadena de ADN es doble, de manera que dos secuencias de nucleótidos se entrelazan en base a enlaces químicos.
Los pares de bases A y T por un lado, y G y C por otro, son los que forman los enlaces, de modo que tenemos una secuencia de ADN como:

```
5′…ACGTTCGT…3′
    ||||||||
3′…TGCAAGCA…5′
```

Es evidente que dada una de las dos secuencias enlazadas, podemos inferir la **cadena complementaria**. Esta estructura en doble hélice es, en última instancia, la responsable de que las células de tu cuerpo se multipliquen!

Introducción

- Bioinformática: ¿qué es?
  - Actores
- Biología básica
  - Genomas
  - Genes

Biología básica: Genes

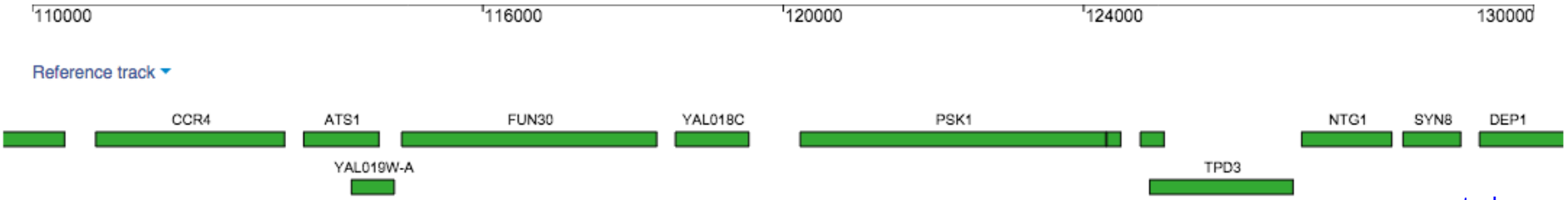
replicación, orígenes y oriC

conjuntos, diccionarios, bucles

temporización y gráficas

Genes y Secuencias no codificantes

- El genoma contiene **genes** y **secuencias no codificantes**



www.genestack.org

Un trozo del cromosoma I de *S cerevisiae* (la levadura del pan), en particular los nucleótidos de la posición 11000 a la 13000.
Debajo, los genes conocidos en la cadena 5' a 3' (sentido) y en la 3' a 5' (anti-sentido).

El espacio en ambas cadenas sin genes es secuencia no codificante.
La secuencia no codificante ha sido relativamente ignorada hasta hace poco (a veces se la nombraba como DNA basura o 'junk DNA'), pero se están descubriendo muchas funciones importantes en ella.

Orígenes de replicación

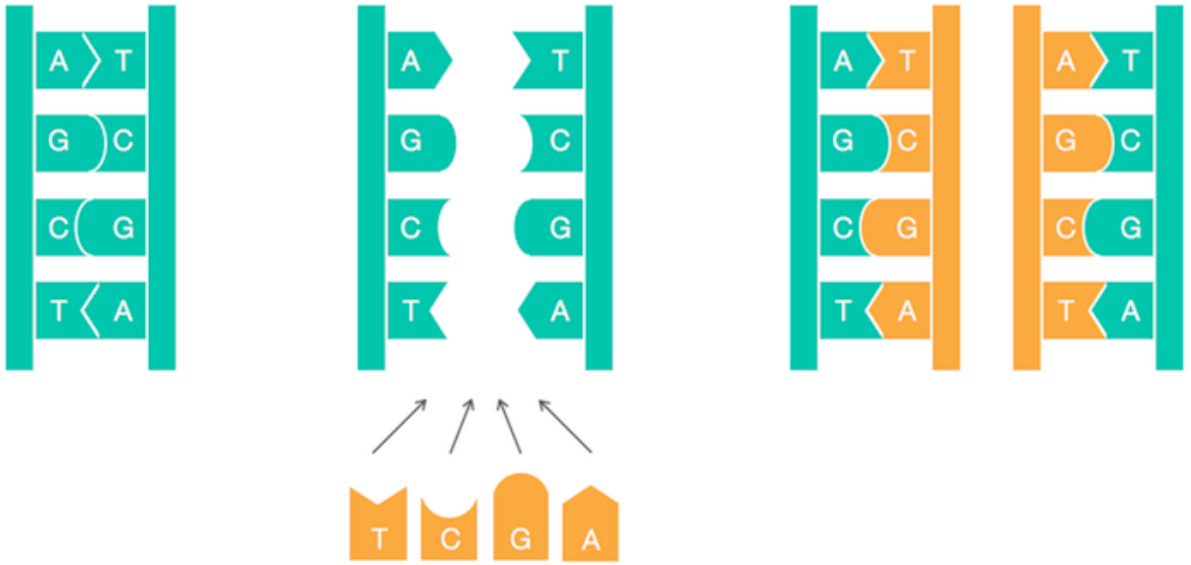
- Un ejemplo de secuencia no codificante clave son los orígenes de replicación (*oriC*)

- Las aproximaciones experimentales para encontrar *oriC* en una determinada especie consumen mucho tiempo, implicaría ir probando a ‘quitar’ trozos del genoma al bicho en cuestión hasta que deje de replicarse (así es como los biólogos prueban casi todo: !mutilando bichos!). Esto hace que, a nivel experimental, *oriC* sólo esté descrito para unas pocas especies.

- Si diseñamos un buen método computacional para encontrar *oriC*, !los biólogos pueden dedicar su tiempo a otras tareas más interesantes que cortar bichos!

Replicación

- Es uno de los procesos más importantes de la célula
  - Necesario antes de dividirse, para dar una copia genética a la célula hija



Buscando oriC

- En algunas bacterias como *E coli*, oriC se encuentra en zonas 'ricas en AT'
  - Estas caracterizaciones según la frecuencia de aparición de los nucleótidos es frecuente.

El **contenido en GC** (guanina-citosina) es el porcentaje de bases G + C respecto al total en una secuencia de ADN

Los enlaces GC son más fuertes que los AT (tres enlaces de hidrógeno en vez de dos) y por tanto más resistentes, por ejemplo, a la desnaturalización por temperatura.

El contenido en GC varía entre organismos, desde un 20% en el *Plasmodium falciparum* hasta el 70% en algunas bacterias. De hecho, en estas últimas, a veces se utiliza para clasificarlas (bacterias con alto vs bajo GC)

Conjuntos

- Un conjunto es como una lista, pero no permite repeticiones de elementos

```
conjunto=set()              #un conjunto vacío
conjunto.add("bola8")       #añadir un elemento
conjunto.add("bola8")       #añadir un elemento
conjunto

lista=[1,2,4,4]
conjunto.update(lista)      #añadir desde lista
```

Diccionarios

- Son como listas, pero a los valores se accede por una **clave** en vez de por posición (~hash)
- Asignación y acceso

```
dicVacio={}

#animales en riesgo de extinción: el nombre del animal
#es la clave y el número de individuos el valor
wild_animals={“amur leopard”: 37, “black rhino” : 4848,
              “cross-river gorilla” : 250}
wild_animals[“black rhino”]
```

https://worldwildlife.org/species/directory?direction=desc&sort=extinction_status

Diccionarios

- Inserción, modificación y borrado

```
wild_animals[“sumatran elephant”]=2600
wild_animals[“amur leopard”]=23
del wild_animals[“amur leopard”] #borrado por clave
```

- Listas, diccionarios y tipos básicos se pueden **combinar** como queramos

```
#Llévate todo esto cuando salgas de viaje!
inventario={“oro” : 500,
“zurrón” : [“piedra”, “cuerda”, “manzana”],
“morral” : [“daga”, “flauta”, “manta” , “queso”]}

inventario[“bolsillo”]=[“gema”, “pelusa”]
```

bucle for

- Sintaxis:

```
for i in range(10):
  print(i)
```

"Para cada número `i` de la serie de 0 a 9, imprime `i`"

- Recuerda que `for` se puede usar sobre cadenas, listas y diccionarios!

```
for i in coleccion:
  print(i)
```

"Para cada letra/elemento/clave `i` de la cadena/lista/diccionario `coleccion`, imprime `i`"

range

- Sintaxis

```
range([start=0], stop, [step=1])
```

  - Retorna una lista de números desde `start` hasta `stop-1`, con saltos basados en `step`
  - P. ej., para recorrer las *posiciones* de una lista:
    - `range(len(lista));`

bucle while

- Recordad que, como pasaba con `if`:
  - No hay llaves (se usa indentación)
  - La sentencia termina en :
  - `for` y `while` no usan paréntesis

```
recuento = 0
while recuento < 5:
  print(“Hola, soy un bucle while y el recuento es “, recuento)
 recuento+=1
```

1 punto

Ejercicio 2

- Implementar una función `frecuencia`
  - Entrada:
    - `seq`: secuencia de nucleótidos
  - Retorno: diccionario con pares (k,v)=(nucleótido, frecuencia de aparición)
- Ejemplo: `ACGTTCGT` → `{'A':0.125,'C':0.25,'G':0.25,'T':0.375}`
- Prueba:
  - `seq`: Genoma de *Vibrio cholerae*
    - http://vis.usal.es/rodrigo/documentos/bioinfo/avanzada/genomas/Vibrio_cholerae.txt

2 puntos

Ejercicio 3

- Implementar una función `GCcontent`
  - Entrada:
    - `path`: ruta a un fichero en formato FASTA
  - Retorno: porcentaje con el contenido en GC promedio de todas las secuencias en `file`
- Ejemplo:
  - Candida_albicans_SC5314.fasta[1,2] → 0.350689487893

[1]Recordad que tenemos los archivos de genomas aquí: http://vis.usal.es/rodrigo/documentos/bioinfo/avanzada/genomas

[2]NOTA: Podéis observar que en la secuencia hay algunos nucleótidos 'raros' tales como R, Y o N. Nada que ver con A,T,C,G. Para este ejercicio los obviaremos, pero se refieren a puntos donde la secuencia no está clara y se duda entre dos o más nucleótidos (ver http://www.bioinformatics.org/sms/iupac.html)

Ejercicio 3

- Implementar una función `GCcontent`
  - Entrada:
    - `path`: ruta a un fichero en formato FASTA
  - Retorno: porcentaje con el contenido en GC promedio de todas las secuencias en `file`
- Ejemplo:
  - Candida_albicans_SC5314.fasta[1,2] → 0.350689487893

[1]Recordad que tenemos los archivos de genomas aquí: http://vis.usal.es/rodrigo/documentos/bioinfo/avanzada/genomas

[2]NOTA: Podéis observar que en la secuencia hay algunos nucleótidos 'raros' tales como R, Y o N. Nada que ver con A,T,C,G. Para este ejercicio los obviaremos, pero se refieren a puntos donde la secuencia no está clara y se duda entre dos o más nucleótidos (ver http://www.bioinformatics.org/sms/iupac.html)

Ejercicio 3

- Prueba
  - El contenido en GC de algunas bacterias es muy alto. Sin embargo, el parásito de la malaria tiene un contenido en GC muy bajo
    - https://en.wikipedia.org/wiki/GC-content#Application_in_systematics
  - Comprobad que los datos de la Wikipedia respecto a estos porcentajes son correctos
    - Prueba:
      - `ruta`: ruta al archivo fasta de *P falciparum**
    - Otra prueba (*no* para el test):
      - `ruta`: ruta al archivo fasta de *S coelicolor**

*Los tenéis en la misma url que *C albicans*, pero una búsqueda web por 'P falciparum FASTA' retorna como primera ocurrencia la página Ensembl Genomes, encargada de hacer públicas todas las secuencias genómicas de distintos organismos

Cronometrando

```
import time
t0=time.clock()
[...]                       #órdenes a cronometrar
print(time.clock()-t0) #resultado en segundos
```

Los problemas bioinformáticos suelen ser cosa fina. Requieren mucha memoria y capacidad de cálculo. Por eso es muy importante monitorizar nuestro código, para detectar las zonas más lentas y optimizarlas.

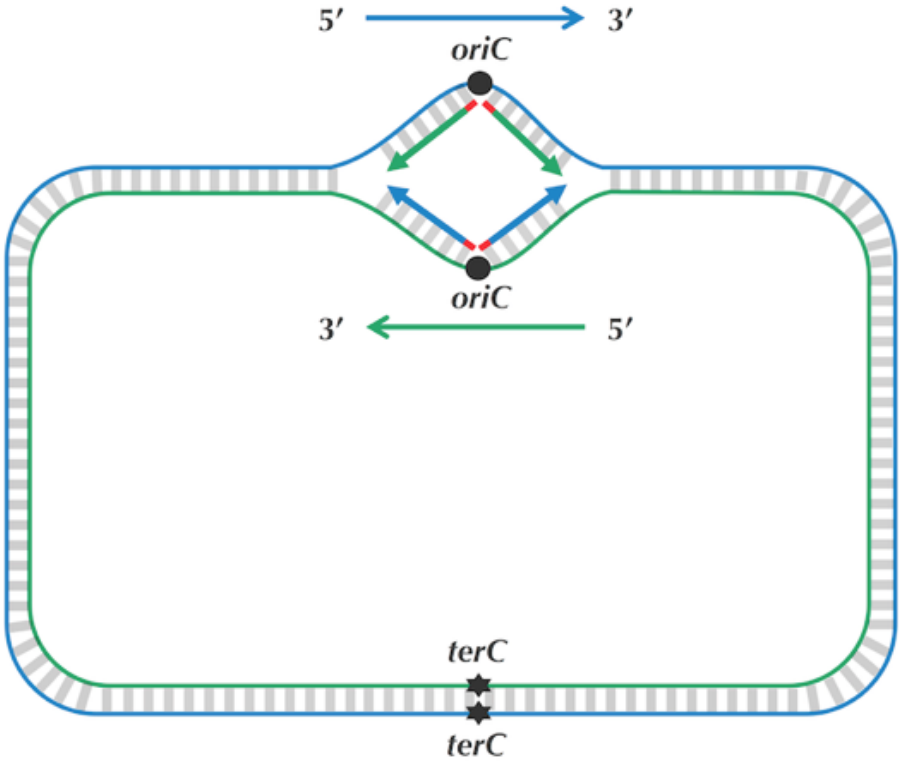
Esto va a ser especialmente importante si tratamos con genomas completos o muchas secuencias, o si utilizamos procedimientos algorítmicos complejos.
¿Cuánto tarda en ejecutar tu código el ejercicio anterior? ¿Hay diferencias respecto a cada organismo?

Buscando oriC (II)

- Todo esto es muy bonito, pero lo cierto es que el conocimiento de que *oriC* es rico en AT (en algunas bacterias) es algo que sabemos *ahora*
  - Para llegar a eso, ¡primero tenemos que ubicar *oriC* en el genoma!
- Vamos a necesitar un cursillo rápido de cómo funciona la replicación

Apertura del genoma

- *DNA polimerasa*: enzima que va 'añadiendo nucleótidos' complementarios a una cadena simple de ADN

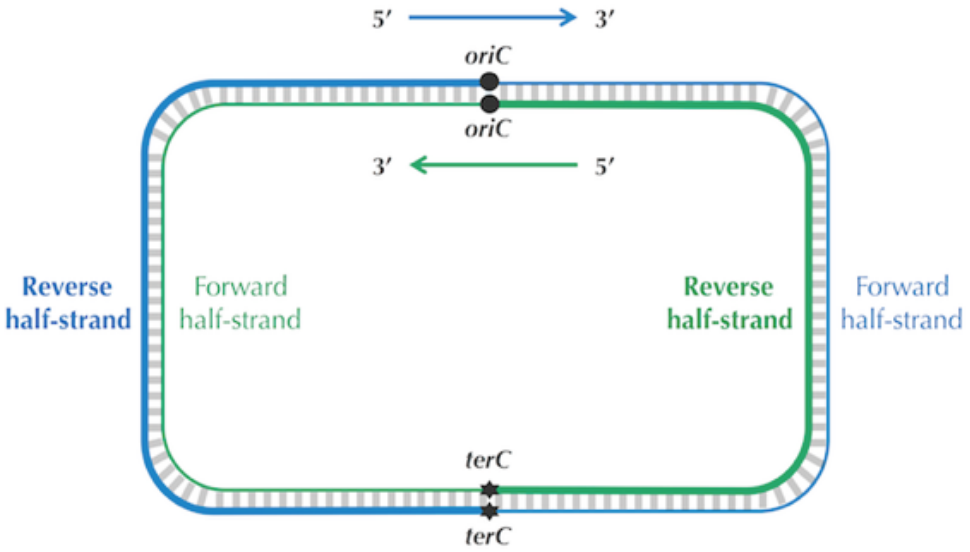


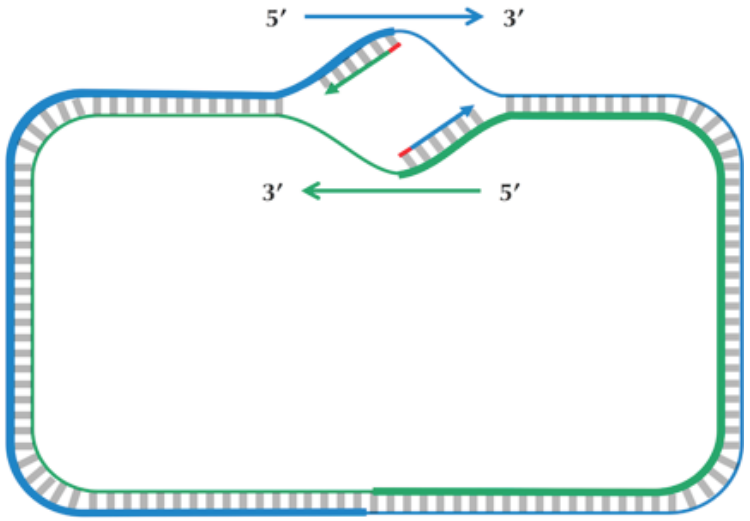
Cuatro DNA polimerasas (rojo) replicando el genoma desde oriC

… no del todo correcto

Asimetría en la replicación

- La DNA polimerasa sólo puede añadir nucleótidos en el sentido 3’ → 5’ (*reverse**)

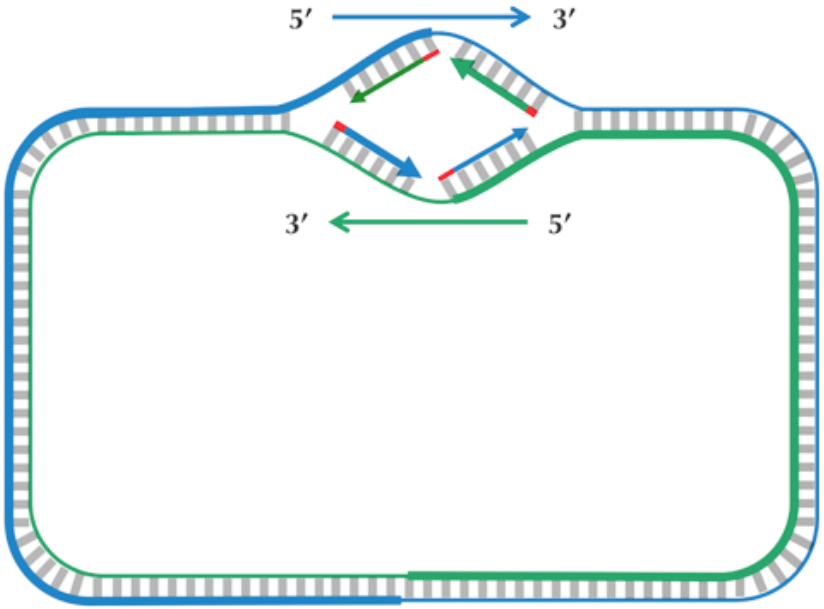


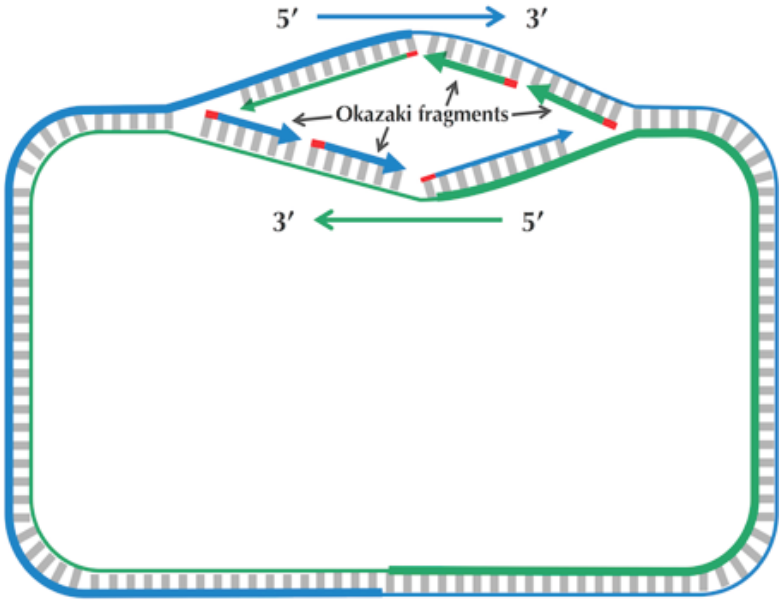


*El sentido de la transcripción es del extremo 5’ al 3’, a veces denominado *positivo*, *downstream, forward o ‘sentido’*, siendo el sentido 3’ a 5’ el *negativo*, *upstream, reverse* o *antisentido.*
5’ y 3’ se refieren a la posición del carbono dentro del anillo riboso, al quinto carbono se adhiere el grupo fostafo, mientras que al tercero se adjunta el grupo hidroxilo.

Asimetría en la replicación (II)

- El sentido 5'→3' (*forward*) se replica 'hacia atrás' cuando la cadena se ha abierto un poco
  - Trocito a trocito (fragmentos de Okazaki)

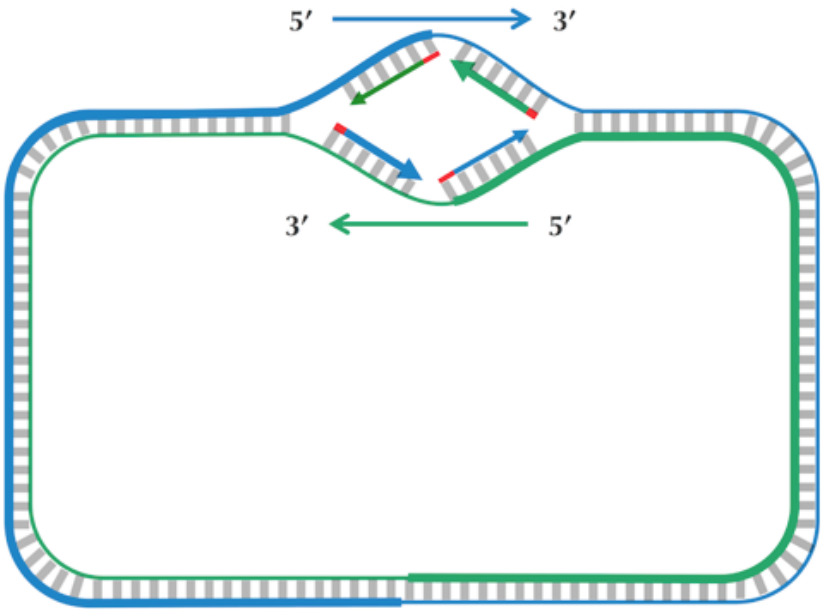


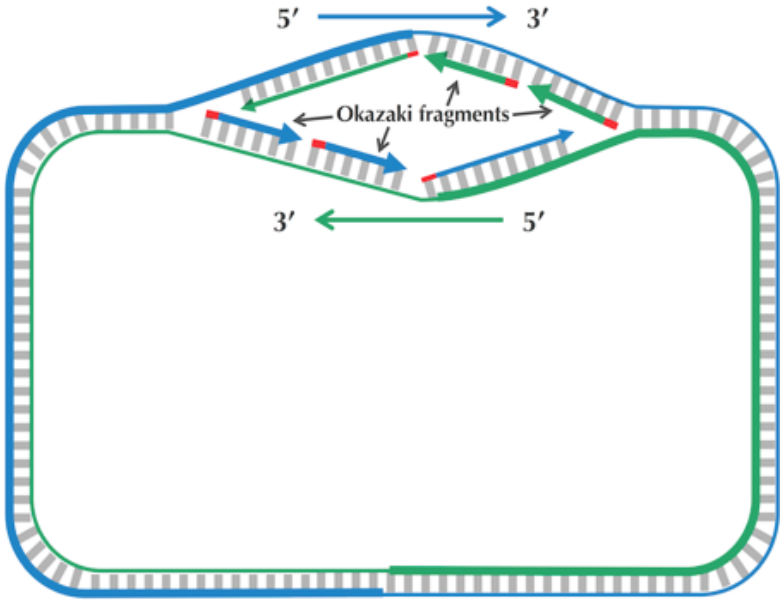


Deaminación

- La cadena 'inversa' tiene que esperar un poco hasta que puede replicarse
  - Es decir, pasa más tiempo como cadena 'simple'
  - Es más fácil que sufra mutaciones
    - En particular, un proceso llamado **deaminación** (mutación de C en T) es 100 veces más probable en cadenas simples que en dobles
- Una mitad de nuestras secuencia se habrá replicado de manera normal y la otra mediante fragmentos de Okazaki
  - Es decir, una mitad de la secuencia tendrá muchas más posibilidades de sufrir deaminación

Buscando oriC con la deaminación





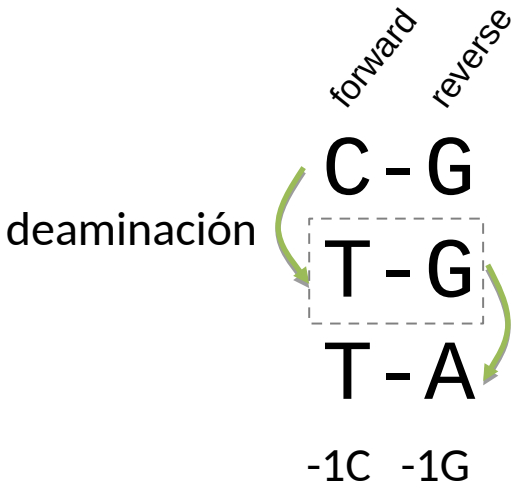
	C	G	A	T
Entire strand	427419	413241	491488	491363
Reverse half-strand	219518	201634	243963	246641
Forward half-strand	207901	211607	247525	244722
Difference	**+11617**	**-9973**	-3562	-1919

de nucleótidos en *Thermotoga petrophila*

Efectivamente, tenemos menos C en la secuencia *forward*, que se mantiene simple durante un tiempo

¿Pero, no deberíamos tener en dicha secuencia más T en lugar de más G?

Arreglando la deaminación



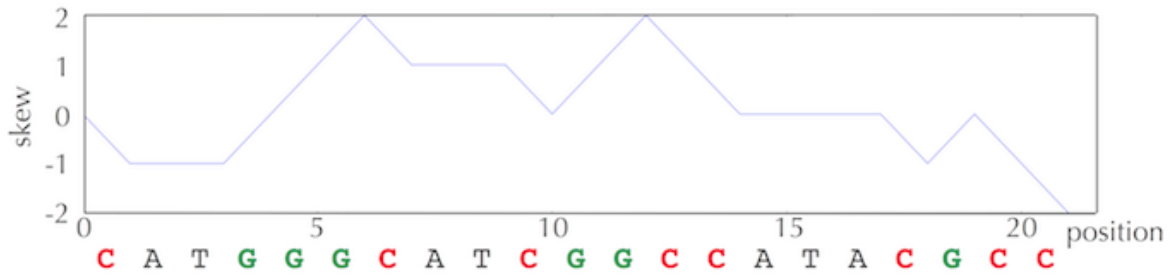
este emparejamiento es incorrecto una 2ª ronda de la DNA polimerasa lo arregla

	C	G	A	T
Entire strand	427419	413241	491488	491363
Reverse half-strand	219518	201634	243963	246641
Forward half-strand	207901	211607	247525	244722
Difference	**+11617**	**-9973**	-3562	-1919

de nucleótidos en *Thermotoga petrophila*

Ejercicio

- Implementar la función `skew`:
  - **entrada**: `seq` (cadena de ADN)
  - **salida**: lista numérica que comienza en 0 y va variando en +1 cuando encontremos una G y en -1 cuando encontremos una C



`CATGGGCATCGGCCATACGCC` → `0 -1 -1 -1 0 1 2 1 1 1 0 1 2 1 0 0 0 0 -1 0 -1 -2`

2 puntos

Ejercicio 4

- Implementar la función `minSkew`:
  - **entrada**: `seq` (cadena de ADN)
  - **salida**: posiciones de seq donde el skew tiene un valor mínimo
- Ejemplo:
  - entrada:
    - `TAAAGACTGCCGAGAGGCCAACACGAGTGCTAGAACGAGGGGCGTAAACGCGGGTCCGAT`
  - salida: `11,24`
- Prueba: *E coli* *

*Recordad que tenemos los archivos de genomas aquí:
http://vis.usal.es/rodrigo/documentos/bioinfo/avanzada/genomas

Dibujando

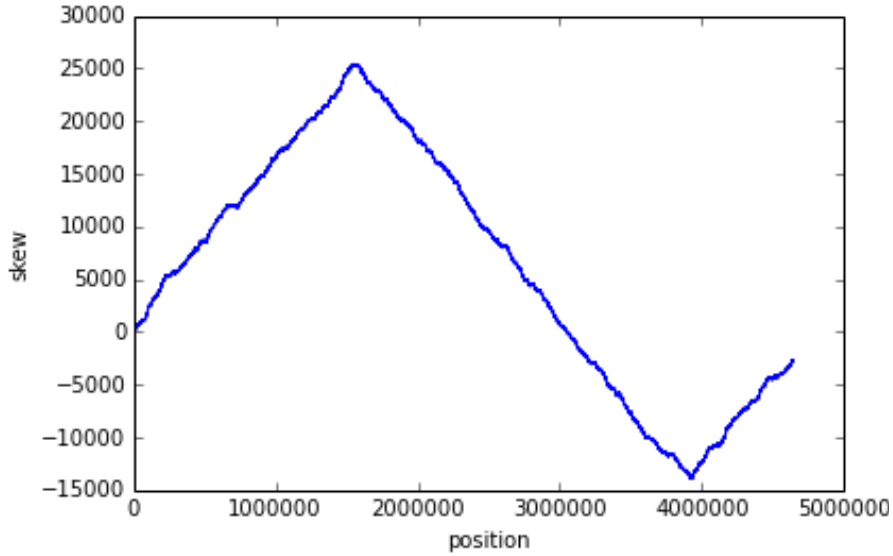
```
import matplotlib.pyplot as plt

plt.plot(lista, “titulo”) #datos a dibujar y título
plt.xlabel(“etiquetaX”)   #etiquetas en los ejes X e Y
plt.ylabel(“etiquetaY”)
plt.show()                #hasta este punto no se dibuja
```

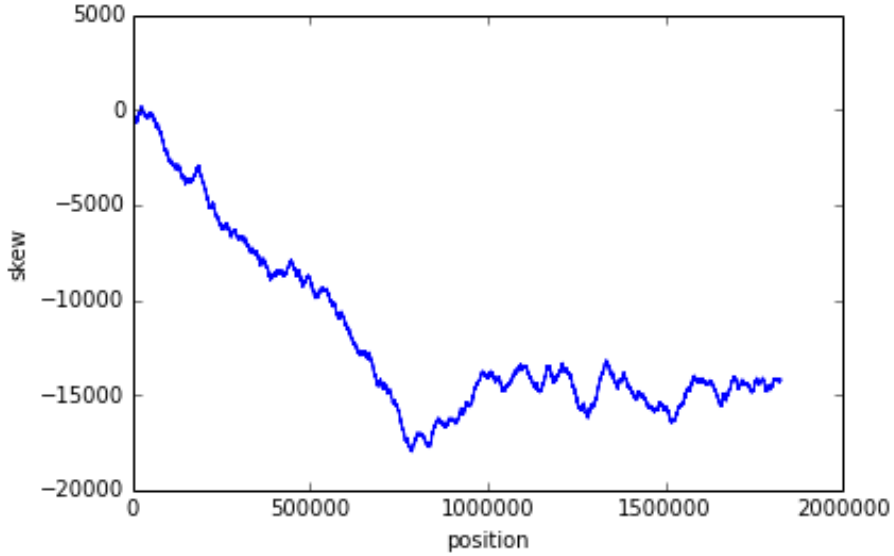
Para que funcionen en Spyder hay que activar el soporte en *Herramientas/Preferencias/Terminal de IPython/Gráficas*
Más información sobre gráficas en http://matplotlib.org/api/pyplot_api.html

Podemos dibujar la función `skew` para distintos organismos y visualizar este fenómeno de la deaminación y la posición de los OriC en los mínimos que aparezcan.

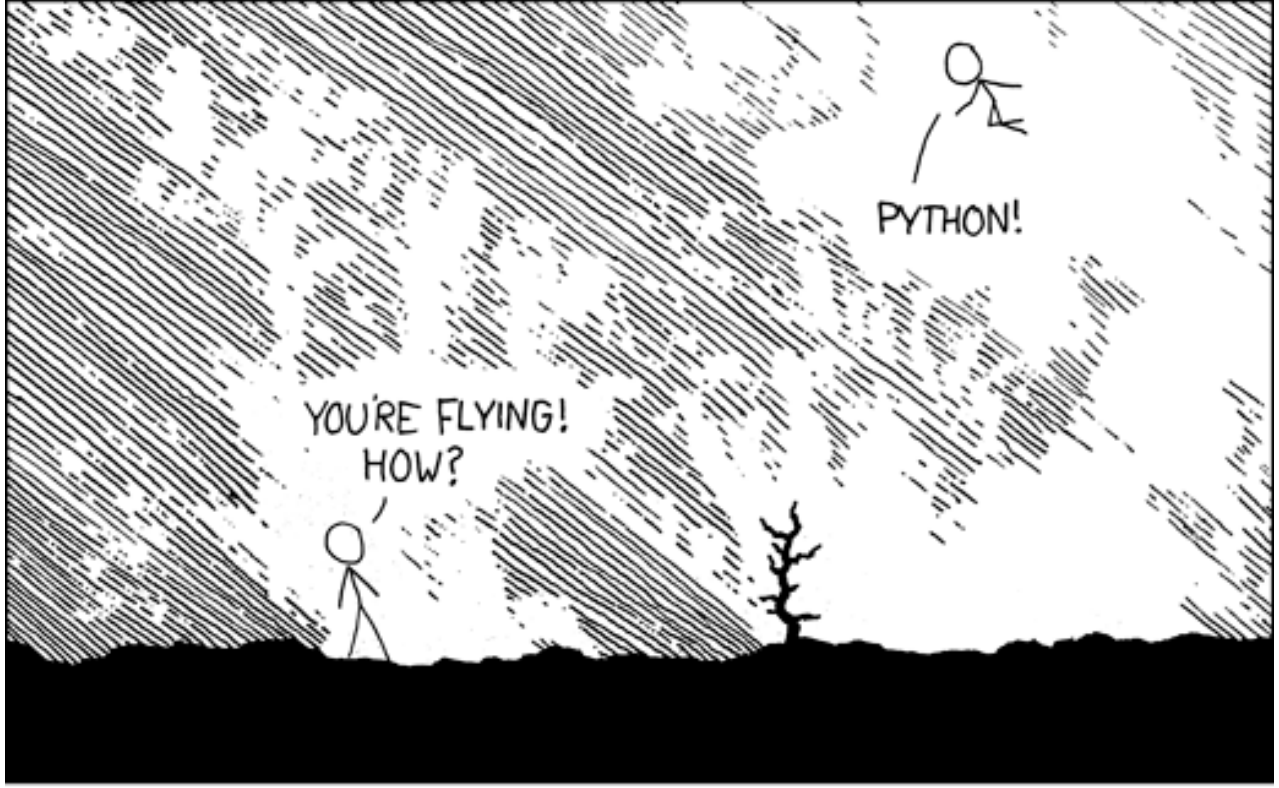
Dibújalos para *V cholerae*, *E coli* y *T petrophila*… ¡a ver qué ocurre!

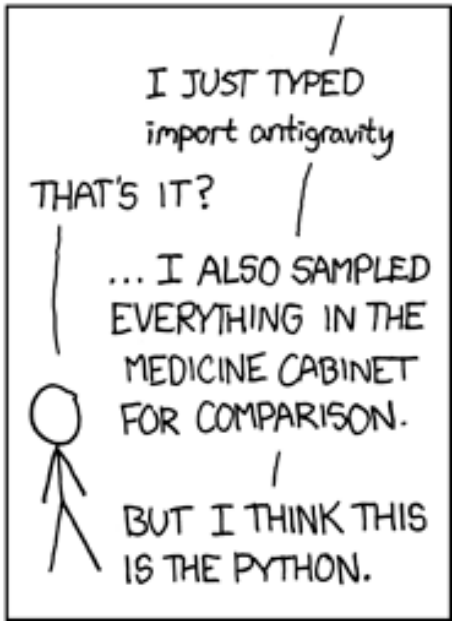


Escherichia coli



Thermotoga petrophila





http://xkcd.com/353/