Dia 2. Introducción al Lenguaje Python y Trabajo con Secuencias

¿Por qué empezar por texto?

1) Como biólogos, tenemos un interés particular en trabajar con texto en lugar de números (aunque, por supuesto, también necesitaremos aprender a manipular números). Interés en tipos particulares de texto que llamamos **secuencias**: las secuencias de **ADN**, **RNA** y **proteínas** constituyen los datos con los que más trabajamos en biología.

2) 95% trabajo en bioinformática va a consistir en escribir código que pueda entender la salida de algún otro programa (a esto lo llamamos **parsear**) o producir salida en un formato en el que otro programa pueda operar.

¿Por qué empezar por texto?

String: cadena de caracteres (término técnico de programación para referirnos a secuencias de caracteres).

```
"esto es un string" y "estotambienesunstring"
"atgcg también es un string, es una secuencia
dentro de este gran string"
```

Secuencia: secuencias biológicas de ADN o proteínas

Imprimiendo texto en terminal

String: cadena de caracteres (término técnico de programación para referirnos a secuencias de caracteres).

```
print("Hello world")
```

Imprimiendo texto en terminal

Toda la línea de este script es lo que denominamos una **instrucción** o **sentencia de código**

```
print("Hello world")
```

Imprimiendo texto en terminal

print() es una **función**.

- una **función** va siempre seguida por paréntesis
- el texto entre paréntesis son los **argumentos**
- **pueden tener varios argumentos separados por “,”**
- argumentos indican que tiene que hacer a la función

```
$ print("Hello world")
> Hello world
```

Imprimiendo texto en terminal

Comillas

- En Python, las cadenas siempre están rodeadas por comillas.
- Esto permite a Python distinguir entre las instrucciones y los datos.
- Se pueden utilizar comillas simples o dobles para las cadenas en Python.

```
$ print("Hello world")
$ print('Hello world')

> Hello world
> Hello world
```

Imprimiendo texto en terminal

Comentarios para anotar tu código

- A veces queremos escribir texto en un programa que sea legible para los humanos, en lugar de para que lo ejecute la computadora.
- Llamamos a este tipo de línea un comentario.
- Para incluir un comentario en su código fuente, comience la línea con un símbolo de almohadilla (#)

```
# Esto es un comentario y va a ser ignorado
por el ordenador
print("Los comentarios son Chahis!").
```

Mensajes de Error y Depuración

No te rindas!!!

- Es muy raro que un script funcione a la primera lo normal es que fallemos y haya que revisar el código

- Los lenguajes de programación no son lenguajes naturales, los errores de escritura de código detendrán la ejecución del código y mostrará un mensaje de error

```
print(Hello world)
```

Mensajes de Error y Depuración

- Los **mensajes de error** son notificaciones que indica un programa cuando encuentra un **problema** o una **inconsistencia en el código**.
- La **depuración** es el proceso de **identificar y corregir** errores en el código para que el programa funcione correctamente.

```
#Error por falta de comillas
```

```
$ python error.py
File "error.py", line 1
print(Hello world)
                ^
SyntaxError: invalid syntax
```

Mensajes de Error y Depuración

- Los mensajes de error en Python suelen ser bastante claros y suelen indicar donde se encuentra el error, incluyendo la linea.
- SyntaxError , python no entiende la sintaxis usada.

```
#Error por falta de comillas
```

```
$ python error.py
File "error.py", line 1
print(Hello world)
                  ^
SyntaxError: invalid syntax
```

Mensajes de Error y Depuración

- NameError, en este tipo de error Python especifica qué palabra específica no entiende Python

```
#Error por escritura incorrecta de funciones
$ prin(Hello world)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'prin' is not defined
```

Mensajes de Error y Depuración

Imaginemos que queremos dividir en dos líneas un string

```
#error.py
print(“Hello
world”)
```

#End Of Line (EOL), string literal -> string in quotes

```
$ python error.py
File "error.py", line 1
    print("Hello
                ^
SyntaxError: EOL while scanning string literal
```

"*I started reading a string in quotes, and I got to the end of the line before I came to the closing quotation mark*"

Tratamiento de caracteres especiales

Los **caracteres especiales** en Python son secuencias de escape que se utilizan para representar caracteres que no se pueden escribir directamente en un string

- Para imprimir texto en múltiples líneas, utilizamos el carácter especial `\n`.
- Otros caracteres especial útiles incluyen `\t` para tabulación y `\\` para la barra invertida.

```
$print("Hello\nWorld")
Hello
World
```

Variables

Podemos asignar un nombre a un string y guardarlo usando simplemente el símbolo `=`

```
# almacenamos una secuencia DNA en variable my_dna
my_dna = "ATGCGTA"

# imprimimos la variable
print(my_dna)
ATGCGTA
```

Variables

Las variables son nombres simbólicos que se utilizan para almacenar datos en la memoria de la computadora.

1. Solo se permiten **letras, números** y **guiones bajos** en los nombres de las variables.
2. No se pueden utilizar caracteres especiales como **£, ^** o **%** en los nombres de las variables.
3. No comenzar variable con un número (si números en medio o al final del nombre).
4. No se pueden utilizar palabras reservadas de Python, como "print".
5.
6. Los nombres de variables son **sensibles a mayúsculas y minúsculas**. "my_dna", "MY_DNA", "My_DNA" y "My_Dna" son variables diferentes.

Variables

Las variables toman nombres arbitrarios, hay que otorgar nombres con sentido.

```
# almacenamos una secuencia DNA en variable banana
banana = "ATGCGTA"

# imprimimos la variable
print(banana)
ATGCGTA
```

Herramientas para manipular strings

- **concatenación** (a+b)
- **len()** : calcular longitud string
- **upper() y lower()** : cambiar minúscula y mayúsculas
- **replace()** : reemplazar un caracter por otro
- **slicing**: extraer una parte de un string con índices
- **count()**: contar nº veces aparece un elemento en un string
- **find():** encontrar un elemento en un string

Herramientas para manipular strings

concatenación (a+b)

Podemos unir 2 strings usando símbolo +

```
#concatenar dos strings
my_dna = "AATT" + "GGCC"
print(my_dna)
AATTGGCC
```

```
#podemos concatenar un string y una variable
upstream = "AAA"
my_dna = upstream + "ATGC"
print(my_dna)
AAAATGC
```

Herramientas para manipular strings

concatenación (a+b)

Podemos unir 2 strings usando símbolo +

```
#podemos concatenar múltiples strings
upstream = "AAA"
downstream = "GGG"
my_dna = upstream + "ATGC" + downstream
print(my_dna)
AAAATGCGGG
```

Herramientas para manipular strings

concatenación (a+b)

No podemos concatenar strings con ***integers*** (*números enteros*). Para ello tenemos que convertir los números en *strings* usando la función ***str()***

```
dna = "AAA"
length = 3
result = "mi dna: "+ dna ";tamaño:" + length
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

Herramientas para manipular strings

concatenación (a+b)

No podemos concatenar strings con ***integers*** (*números enteros*). Para ello tenemos que convertir los números en *strings* usando la función ***str()***

```
dna = "AAA"
length = 3
result = "mi dna: "+ dna ";tamaño:" + str(length)
print(result)
mi dna: AAA;tamaño:3
```

Herramientas para manipular strings

concatenación (a+b)

No podemos concatenar strings con ***integers*** (*números enteros*). Para ello tenemos que convertir los números en *strings* usando la función ***str()***

También podemos convertir *integers* en *strings* añadiendo comillas

```
dna = "AAA"
length = "3"
result = "mi dna: "+ dna ";tamaño:" +length
print(result)
mi dna: AAA;tamaño:3
```

Herramientas para manipular strings

len(string)

La función ***len()*** toma un solo parámetro (*string*) y nos devuelve su longitud.
Nos da un valor de ***retorno*** *(ojo es* ***integer no string****)* que podemos almacenar en una variable

calculate_length.py

```
# store the DNA sequence in a variable
my_dna = "ATGCGAGT"
# calculate the length of the sequence and store it in a variable
dna_length = len(my_dna)
# print a message telling us the DNA sequence length
print("The length of the DNA sequence is " + str(dna_length))
The length of the DNA sequence is 8
```

Herramientas para manipular strings

upper() and lower()

upper y lower son **métodos**, es lo mismo que una función pero solo funcionan para un tipo de datos. ***Solo pueden usarse con strings***

```
my_dna = "ATGC"
# print my_dna in lower case
print(my_dna.lower())
atgc

# print my_dna in upper case
print(my_dna.upper())
ATGC
```

Herramientas para manipular strings

replace()

Otro método exclusivo de *strings*, reemplaza un elemento de un string por otro.

replace(*string a reemplazar*, *valor de reemplazo***)**

```
protein = "vlspadktnv"
# replace valine with tyrosine
print(protein.replace("v", "y"))
# we can replace more than one character
print(protein.replace("vls", "ymt"))
# the original variable is not affected
print(protein)
```

```
ylspadktny
ymtpadktnv
vlspadktnv
```

Herramientas para manipular strings

slicing[x:x_{+1}]

¿Qué ocurre si tenemos un string muy largo y queremos solo una porción?. Podemos coger una porción de un string usando [].

string*[posicion inicio*, posicion final**]*

**Python es 0-based!*
*** El substring incluye el inicio pero no la posición final*

```
protein = "vlspadktnv"
# print positions 1 to 4
print(protein[0:4])
vlsp

print(protein[1]
l
```

Herramientas para manipular strings

indexing[x] : devuelve el carácter que se localiza en la posición x del string

```
dna = "atgcgctagctgct"

print(dna)
"atgcgctagctgct"

#index
# python es un lenguaje 0 based, empieza por el 0 y no por el 1
print(dna[0])
'a'
#si es negativo empieza por el último valor
print(dna[-1])
't'
print(dna[-2])
'c'
```

Herramientas para manipular strings

slicing[x:b] : devuelve un substring desde la posición x hasta la posición y

```
dna = "atgcgctagctgct"

#si omitimos el primer valor por default usa 0
print(dna[:3])
'atg'

# omitir el segundo valor imprime, toda la cadena desde el primer índice
hasta el final
print(dna[3:])
'cgctagctgct'

# ::-1 cambia el sentido del string ahora empieza por el final
print(dna[::-1])
'tcgtcgatcgcgta'

# obtener substring desde el final
print(dna[-3:0])
'gct'
```

Herramientas para manipular strings

count(x): metodo de strings que cuenta el número de ocurrencias del elemento x en un string

```
protein = "vlspadktnv"

# count amino acid residues
valine_count = protein.count('v')
leucines_count = protein.count('l')
tryptophan_count = protein.count('w')

# now print the counts
print("valines: " + str(valine_count))
print("leucines: " + str(leucines_count))
print("tryptophans: " + str(tryptophan_count))

#output
valines: 2
leucines: 1
tryptophans: 0
```

Herramientas para manipular strings

find(x): metodo de strings que **busca** el elemento **x** en un string y te **devuelve la posición (index)** de dicho elemento

```
protein = "vlspadktnv"

# find amino acid residues
print(str(protein.find('p')))
print(str(protein.find('kt')))

#output
3
6

# buscamos un elemento que no está presente
print(str(protein.find('w')))

#output
-1
```