Examen 2025-05-23

1 Papers Please

La república de Arstotzka vive tiempos de tensión en sus fronteras. Los oficiales de inmigración inspeccionan cada documento que cruza el puesto de control y determinan si cumple con las normativas estatales. Desde pasaportes hasta visados de entrada, cada detalle cuenta: un sello faltante o una fecha errónea puede significar denegar el acceso a solicitantes desesperados.

El Comité de Seguridad del Estado te ha encargado la tarea de **implementar un sistema automático** que valide los datos de los pasaportes. Deberás **leer la información facilitada**, **verificar los datos** y **aplicar reglas fronterizas**. El objetivo es crear un sistema que determine si un viajero puede pasar (`True`) o no (`False`).

1.a Solicitar pasaporte (1,5 pts)

La primera tarea es solicitar al viajero los datos de su pasaporte. El pasaporte tiene los siguientes campos: `nombre` y `apellido` de tipo textual; `n_pasaporte` y `año_caducidad` de tipo numérico entero; `motivo_visita` de tipo textual (puede ser `turista`, `trabajo`, `asilo` o `residente`); `duracion_visita` de tipo numérico entero.

> Ejercicio
>
> - Define la clase `Pasaporte` con los campos descritos anteriormente.
> - Escribe la función `solicitar_pasaporte()` que interactúe con el viajero (por teclado y pantalla) y devuelva un objeto de tipo `Pasaporte` nuevo.
>   - La interacción deberá seguir el esquema que se muestra **más adelante**.

Ejemplo de ejecución:

```
solicitar_pasaporte()
```

```
Apellido, Nombre: Lestrade, Vince
A. Pasaporte: 44556663
Año caducidad: 1983
Motivo visita: trabajo
Duración visita: 15
```

Solución

```
@dataclasses.dataclass
class Pasaporte:
    nombre: str
    apellido: str
    n_pasaporte: int
    año_caducidad: int
    motivo_visita: str
    duracion_visita: int


def solicitar_pasaporte():
    print("Apellido, Nombre: ", end="")
    apellido_nombre = input()
    nombre = apellido_nombre.split(",")[1].strip()
    apellido = apellido_nombre.split(",")[0].strip()
    print("N. Pasaporte: ", end="")
    pasaporte = int(input())
    print("Año caducidad: ", end="")
    caducidad = int(input())
    print("Motivo visita: ", end="")
    motivo = input()
    print("Duración visita: ", end="")
    duracion = int(input())
    return Pasaporte(
        nombre,
        apellido,
        pasaporte,
        caducidad,
        motivo,
        duracion,
    )
```

1.b Verificar pasaporte (1,5 pts)

La república de Arstotzka tiene un sistema muy estricto de condiciones para la entrada de personas. El incumplimiento de cualquiera de las siguientes condiciones supone la denegación inmediata de la entrada al país.

- Los pasaportes caducados no son válidos.
- En función del `motivo_visita`:

- `turista`: Los turistas tienen prohibido entrar en el país.
- `trabajo`: Las visas de trabajo permiten estar en el país un máximo de 15 días.
- `asilo`: Las visas de asilo permiten estar en el país un máximo de 90 días.
- `residente`: Los residentes pueden permanecer indefinidamente en el país.

Si el año actual es el 1983:

Año caducidad	Motivo visita	Duración visita	Veredicto
1981	residente	0	**DENEGADO**
1987	residente	0	**ACEPTADO**
1983	residente	0	**ACEPTADO**
1987	turista	3	**DENEGADO**
1987	trabajo	20	**DENEGADO**
1987	asilo	20	**ACEPTADO**

Además, a menudo los pasaportes son alterados o falsificados. La identificación de un pasaporte falso supone la denegación inmediata de la entrada al país y la posible detención del portador. Por lo tanto, también se deben verificar los siguientes datos:

- Que `n_pasaporte` es un número de 8 dígitos.
  - Ej. `445566` NO es válido.
- Que el `año_caducidad` no supera por más de 5 años al año actual.
  - Ej. En el año 1983, la máxima fecha aceptable de caducidad es 1988.

> Ejercicio
>
> - Escribe una función `verificar_acceso(pasaporte, año_actual)` que devuelva `True` si se le debe dar paso al país o `False` si no se le debe dar paso.

Solución

```python
def verificar_acceso(pasaporte, año_actual) -> bool:
    vigente = pasaporte.año_caducidad > año_actual

    if pasaporte.motivo_visita == "residente":
        motivo_ok = True
    elif pasaporte.motivo_visita == "turista":
        motivo_ok = False
    elif pasaporte.motivo_visita == "trabajo":
        motivo_ok = pasaporte.duracion_visita <= 15
    elif pasaporte.motivo_visita == "asilo":
        motivo_ok = pasaporte.duracion_visita <= 90
    else:
        motivo_ok = False  # No deberíamos llegar aquí

    numero_autentico = 10000000 <= pasaporte.n_pasaporte <= 99999999
```

```
    caducidad_autentica = pasaporte.año_caducidad <= año_actual + 5

    return vigente and motivo_ok and numero_autentico and caducidad_autentica
```

1.c Cargar Censo (2 pts)

Los intentos de falsificación de pasaportes se han vuelto tan frecuentes que las autoridades de Arstotzka han decidido crear un censo estatal de habitantes. Para ello, han centralizado en un fichero de texto (*muy grande*) toda la información de los habitantes Arstotzkos. En las fronteras de la república de Arstotzka, un funcionario trae cada mañana un *pen drive* la versión más reciente del censo en un fichero llamado `censo.txt`.

Las **3 primeras líneas del fichero** son las siguientes:

```
apellido,nombre,n_pasaporte
Lestrade,Vince,44556663
Mykolski,Wladimir,1984
```

Observa cómo la primera línea contiene los **nombres de los campos**, mientras que el resto de líneas contienen los datos de cada **registro censal**.

Ejercicio

- Define la clase `RegistroCensal` con los campos: `nombre`, `apellido` y `n_pasaporte` **con los tipos de datos más adecuados**.
- Escribe la función `cargar_censo()` que lea el fichero `censo.txt` (el fichero siempre se llamará así) y devuelva una lista de objetos de tipo `RegistroCensal`.

Solución

```
@dataclasses.dataclass
class Censo:
    apellido: str
    nombre: str
    n_pasaporte: int


def cargar_censo():
    censo = []
    f = open("censo.txt")
    f.readline()  # cabecera
    linea = f.readline()
    while linea != "":
        datos = linea.strip().split(",")
        censo.append(
            Censo(
                datos[0],
                datos[1],
                int(datos[2]),
            )
```

```
        )
        linea = f.readline()
    f.close()
    return censo
```

1.d Verificar Censo (2 pts)

Las autoridades de Arstotzka te han solicitado que implementes la autenticidad de los pasaportes constrandolos con el censo. Para ello, el sistema debe cotejar todos los datos del pasaporte con el registro censal correspondiente a ese ciudadano. **Si cualquiera de los datos del registro censal no coincide con el pasaporte**, se debe denegar la entrada al país.

> Ejercicio
>
> - Escribe una función `verificar_censo(censo, pasaporte)` que devuelva `True` si el pasaporte es original (coincide con el censo) o `False` si no.

Solución

```
def verificar_censo(censo, pasaporte):
    for i in range(len(censo)):
        if (
            censo[i].apellido == pasaporte.apellido
            and censo[i].nombre == pasaporte.nombre
            and censo[i].n_pasaporte == pasaporte.n_pasaporte
        ):
            return True
    return False
```

2 ADN

Acabas de entrar en el equipo de investigación y desarrollo de un prestigioso laboratorio de biología molecular. La primera tarea que te han encomendado es la de añadir una serie de funciones a un módulo que tu grupo está desarrollando para el procesado de secuencias de ADN.

Las secuencias de ADN son cadenas de nucleótidos que contienen la información genética de un organismo. El ADN está formado por cuatro nucleótidos: Adenina (A), Citosina (C), Guanina (G) y Timina (T). La librería en la que tu grupo está trabajando representa las secuencias de ADN como **listas de strings** donde cada celda contiene uno de los cuatro nucleótidos. A pesar de que le has comentado a tu jefe de grupo que **sería mejor representarlas como cadenas de texto**, ya que eso permitiría utilizar las funciones de Python para cadenas de texto, **él insiste en utilizar listas de strings**.

Ejemplo de secuencia de ADN:

```
secuencia = ["A", "C", "G", "T", "A", "C", "G", "T"]
```

2.a Complemento Inverso (1,5 pts)

La primera función que te han pedido implementar es la de calcular el complemento inverso de una secuencia de ADN.

Cada nucleótido tiene su complementario: `A` <-> `T` y `C` <-> `G`.

Por ejemplo, estas dos secuencias son complementarias:

```
["A", "A", "C", "G"]  # Original
["T", "T", "G", "C"]  # Complemento
```

El complemento inverso de una secuencia de ADN es la secuencia que se obtiene al **invertir la cadena** y **sustituir cada nucleótido por su complemento**:

```
["A", "A", "C", "G"]  # Original
["T", "T", "G", "C"]  # Complemento
["C", "G", "T", "T"]  # Complemento inverso
```

Ejercicio

- Escribe una función `complemento_inverso(secuencia)` que reciba una `secuencia` de tipo *lista de strings* y devuelva una nueva lista con el complemento inverso de la `secuencia`.

Ejemplo de uso de la función:

```
complemento_inverso(["A", "A", "C", "G"])  # [ "C", "G", "T", "T"]
complemento_inverso(["A", "T"])            # [ "A", "T"]
complemento_inverso(["G", "T", "T"])       # [ "A", "A", "C"]
```

Solución

```
def complemento(nucleotido):
    if nucleotido == "A":
        return "T"
    elif nucleotido == "T":
        return "A"
    elif nucleotido == "C":
        return "G"
    else:
        return "C"


def complemento_inverso(secuencia):
    resultado = []
    for i in range(len(secuencia) - 1, -1, -1):
        resultado.append(complemento(secuencia[i]))
    return resultado
```

Solución alternativa

```python
def complemento_inverso(secuencia):
    sec_str = "".join(secuencia)

    sec_str = sec_str.replace("A", "X").replace("T", "A").replace("X", "T")
    sec_str = sec_str.replace("C", "X").replace("G", "C").replace("X", "G")

    sec_inv = []
    for i in range(len(sec_str) - 1, -1, -1):
        sec_inv.append(sec_str[i])

    return sec_inv
```

2.b Localiza la secuencia (1,5 pts)

A menudo es interesante buscar subcadenas dentro de una secuencia de ADN. La segunda tarea que te han encomendado es la de implementar una función que busque una subcadena dentro de una secuencia de ADN y devuelva la posición de la primera aparición de la subcadena.

Para comprobar si una cadena están contenida en otra, se comprueba posición por posición si la subcadena coincide con la secuencia a partir de esa posición.

```
Cadena: [A, C, G, T, A, C, G, T]
Subcadena: [G, T]

[A, C, G, T, A, C, G, T]
[G, T]                    ✗

[A, C, G, T, A, C, G, T]
   [G, T]                 ✗

[A, C, G, T, A, C, G, T]
      [G, T]              ✓
```

Ejercicio

- Escribe una función `localiza(secuencia, subcadena)` que devuelva la posición de la primera aparición de `subcadena` en `secuencia`, o `-1` si no se encuentra.
  - Para ello te será útil definir una función `coincide(secuencia, i, subcadena)` que devuelva `True` si la `subcadena` coincide con la secuencia a partir de la posición `i`.
  - Se permite el uso de *early return*.

Ejemplos de `coincide()`:

```python
secuencia = ["A", "C", "G", "T", "A", "C", "G", "T"]
print(coincide(secuencia, 2, ["G", "T"]))       # True
print(coincide(secuencia, 2, ["A", "C"]))       # False
print(coincide(secuencia, 2, ["A", "C"]))       # True
print(coincide(secuencia, 0, ["A", "C", "G"]))  # True
```

Ejemplo de `localiza()`:

```
secuencia = ["A", "C", "G", "T", "A", "C", "G", "T"]
print(localiza(secuencia, ["G", "T"]))       # 2
print(localiza(secuencia, ["A", "C", "G"]))  # 0
print(localiza(secuencia, ["A", "G"]))       # -1
```

Solución

```
def coincide(secuencia, i, subcadena):
    for j in range(len(subcadena)):
        if secuencia[i + j] != subcadena[j]:
            return False
    return True

def localiza(secuencia, subcadena):
    for i in range(len(secuencia) - len(subcadena) + 1):
        if coincide(secuencia, i, subcadena):
            return i
    return -1
```

Solución alternativa

```
def coincide(secuencia, i, subcadena):
    for j in range(len(subcadena)):
        if secuencia[i] != subcadena[j]:
            return False
        i += 1
    return True

def localiza(secuencia, subcadena):
    for i in range(len(secuencia) - len(subcadena) + 1):
        if coincide(secuencia, i, subcadena):
            return i
    return -1
```

Solución alternativa

```
def localiza(secuencia, subcadena):
    for i in range(len(subcadena)):
        for j in range(len(secuencia)):
            if secuencia[j] == subcadena[i]:
                sumatorio = 1
                for k in range(1, len(subcadena)):
                    if subcadena[i + k] == secuencia[j + k]:
                        sumatorio += 1
                if sumatorio == len(subcadena):
                    return j
    return -1
```