BOLETÍN DE EJERCICIOS

CALIDADE E PROBAS EN ROBÓTICA
Curso 2024/25

Ejercicio 1

--

UTILIZANDO UNA METODOLOGÍA TDD implementa las funciones necesarias para convertir números arábigos (números naturales) a su correspondiente representación como números romanos y para convertir números romanos a su correspondiente representación como números arábigos:

```
$ python3
>>> from ejercicio1 import r2a, a2r
>>> print(a2r(104))
CIV
>>> print(r2a("IX"))
9
```

Comentarios

-

El objetivo es evaluar tu capacidad para UTILIZAR TDD EN EL DESARROLLO DE UN CÓDIGO SENCILLO. Utiliza la librería unittest de python para construir las pruebas unitarias necesarias para el desarrollo. Al terminar, entrega tanto el código del ejercicio como el código de las pruebas unitarias.

Recuerda que los símbolos empleados en los números romanos son I (1), V (5), X (10), L (50), C (100), D (500) y M (1000). No hace falta pasar del número 3000.

Hay un par de propiedades que puedes usar para convertir entre números arábigos y romanos, si quieres: La primera es aprovechar que la representación de la primera decena (I, II, III, IV... IX, X) se repite con las decenas (X, XX, XXX, XL ... XC), centenas (C, CC ... CM) y miles (M, MM ...) simplemente cambiando los símbolos. La segunda, que representar un número grande equivale a representar sus miles, centenas, decenas y unidades independientemente (1980 -> M CM LXXX).

Para la conversión romano a arábigo, puedes evaluar consecutivamente cada símbolo de izquierda a derecha, donde los valores de los símbolos se van SUMANDO (MMXII = 100+100+10+1+1+1), con una excepción: cuando el símbolo en la posición N es inferior al símbolo en la posición N+1, el valor del símbolo en la posición N se RESTA en vez de SUMARSE (XIX = 10 -1 +10).

Ejercicio 2

--

UTILIZANDO UNA METODOLOGÍA TDD implementa las clases WALLET y CURRENCY que permitan hacer operaciones monetarias más básicas (sumar, restar, multiplicar por un número, convertir entre unidades) sobre subclases de la segunda:

```
from ejercicio2 import currency, wallet

class euro(currency):
    def __init__(self, n):
        self.simbolo, self.n = "EUR", n

class yen(currency):
    def __init__(self, n):
        self.simbolo, self.n = "YEN", n

money_price = {
    "EUR": {
        "EUR": 1,
        "YEN": 0.007
    },
    "YEN": {
        "EUR": 142.9,
        "YEN": 1
        }
    }

n = euro(3)
n = n.suma(3)
n = n.multiplica(2)
n = n.resta(2)
print(n.mostrar())
> 10 EUR

a = wallet()
a.suma(n)
a.suma(yen(1000))
print(a.mostrar("EUR", money_price))
> 17 EUR
print(a.mostrar("YEN", money_price))
> 2429 YEN
```

Comentarios

-

El objetivo es evaluar tu capacidad para UTILIZAR TDD EN EL DESARROLLO DE UN CÓDIGO SENCILLO. Utiliza la librería unittest de python para construir las pruebas unitarias necesarias para el desarrollo. Al terminar, entrega tanto el código del ejercicio como el código de las pruebas unitarias.

La clase CURRENCY debería implementar los métodos públicos suma(int), resta(int), multiplica(int) y mostrar(). La clase WALLET, suma(currency), resta(currency) y mostrar(str, dict).

Te recomiendo que la clase WALLET simplemente mantenga las cantidades indicadas en su moneda original, y que sea al llamar al método mostrar() cuando las sume todas. Por simplicidad, permite cantidades negativas (si

WALLET contiene x valor en moneda X y se le resta z valor en monedas Z, no es posible saber si el total es positivo o negativo hasta conocer el precio del dinero X y Z).

Aunque en este ejercicio se implementan dos clases (WALLET, CURRENCY), no hacen falta pruebas de integración ni perder el tiempo con stubs, mocks y familia; implementa sólo pruebas unitarias.

Ejercicio 3

--

UTILIZANDO UNA METODOLOGÍA TDD implementa una clase que valide el formato JSON:

```
from ejercicio3 import jsonlib

g = """
{"preprocess": {
    "distribution-masks": [
        "../../sub50.nucleos-y-corredores.asc"
    ],
    "init-values-map": "",
    "convert-area": 0.10,
    "output": "buffer-preprocess.asc",
    "debug-maps": "yes"
}}
"""

j = ""
for gg in g.split("\n"):
    j += gg.strip()
print(jsonlib.validar(j))
> True
```

Comentarios

-

El objetivo es evaluar tu capacidad para UTILIZAR TDD EN EL DESARROLLO DE UN CÓDIGO MEDIANO. Utiliza la librería unittest de python para construir las pruebas unitarias necesarias para el desarrollo. Al terminar, entrega tanto el código del ejercicio como el código de las pruebas unitarias.

Ejemplos de formato válido:

```
{}, {"a":0}, {"a":10.5}, {"a":""}, {"a":"txt"},
{"a":{"b":-1}}, {"a":[]},
{"a":[0, 10.5, "0", "", "txt", {"b:":-1}]},
{"clave uno":"valor uno"}
```

Ejemplos de formato no válido (simplificando la especificación JSON para eliminar casos límite):

```
"a":1, {a}, {a:0}, {"a"}, {"a":}, {"a": txt}, {"a":1,}
```

Si tienes dudas, puedes recurrir a https://jsonlint.com/

Ejercicio 4

--

UTILIZANDO UNA METODOLOGÍA TDD implementa un simulador de una plataforma móvil (móvil) que se mueva por un escenario 3D (un mundo de bloques):

```
    from ejercicio4 import simulacion, movil, escenario

    bloques = [[[0 for _ in range(5)]
    for _ in range(5)] for _ in range(4)]
    for i in range(5):
        bloques[0][i] = [1 if random.randint(0, 2) == 0
            else 0 for _ in range(5)]

    simulacion(
        movil((0, 0), 'N', 0, 0, 0,
              escenario((5, 5, 4), bloques)
        ),
        "FORWARD 2, LEFT, SHIFT -1, FORWARD 1"
    )
    > True
```

Móvil

-

El móvil consta de una base con ruedas y un eje central por el que mueve verticalmente un brazo. Este brazo puede extenderse y retraerse a voluntad. Contiene una pinza en su extremo.

Al inicializarse, el móvil recibe sus coordenadas iniciales, orientación y estado del brazo y la pinza. Y el escenario. Durante la simulación irá recibiendo los comandos que modifiquen su posición y su estado. Reconoce los comandos FORWARD N (moverse n posiciones hacia adelante manteniendo la misma orientación; si n es negativo, retrocede), SHIFT N (desplazarse lateralmente a la derecha n posiciones manteniendo la misma orientación; si n es negativo, desplazarse a la izquierda), RIGHT (girar en el mismo lugar 90 grados a la derecha), LEFT (girar en el mismo lugar 90 grados a la izquierda). El brazo puede moverse verticalmente con el comando ELEVATE N (valores positivos y negativos de n); puede extenderse o recogerse con el comando EXTEND N (valores positivos y negativos). Y la pinza puede abrirse y cerrarse con los comandos GRAB y RELEASE.

Escenario

-

El móvil se mueve por un mundo simulado donde aparecen bloques de tamaño unidad, que pueden apilarse unos encima de los otros para formar paredes y obstáculos simples. Un bloque siempre estará en el suelo o posado encima de otro. No son fijos. Cada bloque puede ser desplazado dentro de los límites del mundo simulado. Pero para moverlo, no puede tener otros bloques encima.

Al inicializarse, se especifican las dimensiones del mundo y una representación de los bloques en sus posiciones originales. El suelo por el que se desplazará el móvil es simplemente el límite inferior de este escenario (no hace falta nada más para simularlo).

Simulación

-

Al iniciarse, se lee el conjunto de instrucciones de una cadena de texto que indican las acciones del móvil sobre el mundo simulado. La simulación devuelve True o False según pueda ejecutarse o no.

El móvil puede desplazarse por este mundo sorteando los bloques. También puede interactuar con estos bloques, agarrándolos con la pinza y moviéndolos a una posición distinta. Pero sólo puede desplazar bloques individuales: si un bloque tiene otro descansando encima, no puede acceder a él.

Por simplificar, supondremos que los bloques no pueden ser empujados. Si el móvil tropieza con uno, se detiene y no puede seguir avanzando. Tampoco puede girar el brazo cuando su trayectoria interfiere con la posición de un bloque.

Cuando el móvil suelta un bloque en el aire (debido a la simplificación anterior ésta es la única forma en que un bloque puede cambiar de posición), éste cae verticalmente hasta que lo detenga el suelo u otro bloque.

Comentarios

-

El objetivo es evaluar tu capacidad para UTILIZAR TDD EN EL DESARROLLO DE UN CÓDIGO MEDIANO. Utiliza la librería unittest de python para construir las pruebas unitarias necesarias para el desarrollo. Al terminar, entrega tanto el código del ejercicio como el código de las pruebas unitarias.

La simulación debería realizarse en pasos individuales y evaluar el escenario después de cada uno. Es decir, el comando FORWARD 10 debería dividirse en 10 pasos; antes de cada uno de ellos debería preguntarse al escenario si la nueva posición está libre (comprobar las colisiones de la plataforma, eje, brazo y pinza); debería ejecutarse el movimiento; y actualizar el escenario (por si el móvil abrió la pinza y dejó caer un bloque; con las simplificaciones actuales éste es el único cambio posible en el escenario, pero con otro objeto diferente podríamos incluir desplazamientos de bloques o daños al suelo... que deberían evaluarse después de cada paso individual). Probablemente el móvil implemente un método por cada comando reconocido, mientras que escenario tenga métodos como detectar_colision(coordenadas_destino) y actualizar().

Aunque en este ejercicio se implementan tres elementos (MOVIL, ESCENARIO, SIMULACION), no hacen falta pruebas de integración ni perder el tiempo con stubs, mocks y familia; entrega sólo pruebas unitarias.

Ejercicio 5

--

UTILIZANDO UNA METODOLOGÍA TDD REALIZA LAS PRUEBAS DE INTEGRACIÓN para validar las relaciones entre los módulos principales del siguiente proyecto, ya que suponemos que funcionan bien de manera individual.

Este proyecto en cuestión es un reemplazo web de Powerpoint (¡algo trivial!), donde cada módulo tiene una estructura interna compleja que oculta al resto de módulos, y que expone sólo aquellas funciones pensadas para ser usadas de forma pública (la API). Los módulos más interesantes son el módulo de edición de texto, que se encarga de introducir y manipular contenido en las páginas del documento a través de bloques de texto, y el módulo de idioma, para comprobar la ortografía de los mismos, añadir excepciones (diccionario personalizado), gestionar diccionarios de idiomas, y, tal vez, generar resúmenes automáticamente de toda la presentación.

Si bien el código del módulo editor va a ser llamado por el usuario desde el interfaz gráfico, pulsando en botones, abriendo menús y seleccionando opciones (no nos interesan los detalles), las funciones pueden también ser usadas desde python. Vamos a aprovechar esto para probar la integración entre la edición y la revisión de idioma (construyendo a mano el documento).

```
    from ejercicio5edicion import modulo_editor, \
        objeto_pagina, objeto_bloque, objeto_parrafo, \
        objeto_imagen, objeto_conector, \
        debug_show_document_text_data, test_data, test_config
    from ejercicio5idioma import modulo_idioma

    doc = modulo_editor("documento de ejemplo", test_data, test_config)

    aspell = modulo_idioma(doc.idiomas())

    for i in doc.pages:
        print("page '{}'".format(i.title),
            doc.revisar_pagina_texto(i, aspell))
    > page 'primera página' (30, 0)
    > page 'segunda página' (9, 0)
    > page 'tercera página' (0, 0)
```

Pasos para completar el ejercicio:

1. Identifica las funciones del editor que llaman al módulo de idioma
2. Para cada una,
    2.1. ¿Qué tipo de dato de entrada necesita?
    2.2. ¿Qué comportamientos podemos forzar con los datos de entrada?
    2.3. ¿Podemos forzar errores a través de los datos de entrada?
3. Con esta información, construye y ejecuta los distintos casos de prueba a través de los datos de entrada para lograr comportamientos específicos; para ello debes seleccionar el doble de testeo correcto (dummy, stub, spy, mock y fake)

Ejemplo:

R1 - Encuentramos que en el módulo de edición existe la función revisar_pagina_texto() que conecta con el módulo de idioma (se llama a management_idiom_object.check_block() en cada bloque de la página). Revisando el código de los módulos vemos que:

1. Si el argumento page está vacío, se genera un error
2. Si la página no tiene bloques de texto, no llama al módulo de idioma
3. Esperamos que si se llama al módulo de idioma, devuelva dos números; si devuelve una excepción estamos vendidos

Considero que el primer error no tiene que ver con la integración sino con las pruebas unitarias. Si llamas a una función para usar un objeto y no se lo proporcionas... pues es un error de la propia función.

El siguiente error sí lo podemos convertir en un caso de prueba: vamos a ejecutar el código proporcionando una página sin bloques de texto y el resultado debería ser (0, 0); ¿qué argumento usamos para management_idiom_object? Pues como lo necesitamos, pero no lo vamos a invocar usaremos un DUMMY:

```
    class modulo_idioma_dummy():
        pass

    doc = modulo_editor("ejemplo", [
            objeto_pagina("página con logo", images=[
                objeto_imagen("logo1.svg", (0, 0, -1, -1))
            ])
        ])
    print(doc.revisar_pagina_texto(doc.pages[0], modulo_idioma_dummy()))
    > (0, 0)    # pero hay que convertirlo a un TestCase de unittest!!
```

El siguiente error también lo podemos convertir en un caso de prueba. Obtenemos una excepción en dos casos: cuando el párrafo de texto está en un idioma que no ha sido inicializado en modulo_editor(), y cuando al inicializar modulo_editor() se simula la descarga de un diccionario desde la web y falla (la probabilidad es del 5%); lo primero es un problema de integración, pero lo segundo es cosa exclusiva del módulo de idioma (se valida con pruebas unitarias). Entonces, usando un módulo de idioma sólo con el idioma "es_ES" (español de España) revisamos una página donde un bloque de texto tiene español de venezuela ("es_VE"). ¿Qué doble de testeo usamos para simular esta situación? Un STUB, que siempre devuelva una excepción:

```
    class modulo_idioma_stub_exception():
        def check_block(self, block):
            raise Exception('uf!')

    doc = modulo_editor("ejemplo", [
            objeto_pagina("página con logo", [
                objeto_bloque(
                    [objeto_parrafo("contenido del bloque de texto A",
                        lang="es_VE")], (10, 10, 10, 10)
                )
            ])
        ])
```

```
    print(doc.revisar_pagina_texto(doc.pages[0],
        modulo_idioma_stub_exception()))
    > excepción    # pero hay que convertirlo a un TestCase de unittest!!
```

También deberíamos probar el caso en que funciona correctamente (directamente, sin doble de testeo).

¿Qué más cosas probar? Asegúrate de que check_block() llama a check_paragraph(), por ejemplo (SPY). De que si el documento tiene 4 idiomas, el módulo se idioma va a cargar los 4 idiomas[1] (MOCK), que si un párrafo tiene un idioma X, cada palabra del párrafo se valida contra ese mismo idioma (MOCK) ... ¡lo que veas posible!; tienes permiso para crear objetos de testeo completamente nuevos como hice antes con el dummy, pero también puedes sustituir funciones al módulo de idioma o usar el módulo original. Por ejemplo:

```
    class modulo_idioma_spy_on_check_block(modulo_idioma):
        def check_block(self, b):
            return (-1, -1)

    print(doc.revisar_bloque_texto(doc.pages[0].blocks[0],
        modulo_idioma_spy_on_check_block()))
    > (-1, -1)    # pero hay que convertirlo a un TestCase de unittest!!
```

Comentarios
--
El objetivo es evaluar tu capacidad para UTILIZAR TDD EN EL DESARROLLO DE UN CÓDIGO MEDIANO. Utiliza la librería unittest de python para construir las pruebas unitarias necesarias. TE RECOMIENDO QUE NO USES unittest.mock (objetos Mock, MagicMock y patch), sino que crees tus propios objetos que funcionen como dobles de testeo. Al terminar, entrega el código de las pruebas de integración.

Debería haber documentación en el campus virtual sobre los dobles de testeo. Si aún no está, recuerda:

- DUMMY: un objeto que necesitamos porque se requiere pasar un argumento más a la función, pero que no interesa
- STUB: un objeto de reemplazo que devuelve siempre los mismos valores estáticos
- SPY: un objeto de reemplazo que señala que se ha llamado a una función concreta del objeto
- MOCK: similar al spy, nos asegura que se ha llamado a la(s) funcion(es) concreta(s) del objeto mientras registra los parámetros usados
- FAKE: un objeto que realiza un cálculo simplificado para ofrecer valores de salida en lugar del cálculo original, mucho más complicado

[1] Lo del número de idiomas del documento... El sistema está pensado para que el módulo de idioma se inicialice con los idiomas del documento. Para ello el módulo de edición tiene la función idiomas(), que lista todos los que hay. Entonces, para validar esto, construyes un MOCK que sustituya la función load_dictionary_rules() y recuerde cuántos idiomas se intentan cargar, y lo comparas con los idiomas del documento:

```
    doc = modulo_editor("ejemplo", test_data, test_config)
    print(doc.idiomas())
```

Para las pruebas de integración no te queda más remedio que conocer la estructura interna de los datos que utiliza el módulo de edición (también puedes echar un vistazo al objeto test_data):

documento:
- config: un diccionario, donde lo más interesante es "default_lang":"en_US"
- title: cadena de texto (título del documento)
- pages: un array de objeto_pagina() para guardar el contenido de las páginas

objeto_pagina()
- title: cadena de texto (título de la página)
- blocks: un array de objeto_bloque() con los bloques de texto en la página
- images: un array de objeto_imagen() con las imágenes en la página
- arrows: un array de objeto_enlace() con las flechas/líneas en la página

objeto_bloque()
- text: un array de objeto_parrafo() con los párrafos de texto del bloque
- geometry: la geometría del bloque (x, y, ancho, alto)

  objeto_parrafo()
  - txt: cadena de texto para el párrafo
  - lang: idioma del párrafo; opcional; si no se especifica es "default_lang"

objeto_imagen()
- filename: cadena de texto con fichero de imagen
- geometry: la geometría de la imagen (x, y, ancho, alto)

objeto_enlace()
- geometry: la geometría de la flecha o línea

Ejercicio 6

--

SIGUIENDO LA ESTRATEGIA DE PRUEBAS DE CAJA BLANCA identifica los casos de prueba necesarios para validar el siguiente código:

```
    def ordenamiento_por_insercion(data):
        for i in range(1, len(data)):
            elemento = data[i]
            j = i - 1
            while (j >= 0 and data[j] > elemento):
                data[j+1] = data[j]
                j = j - 1
            data[j+1] = elemento
        return data
```

Pasos para completar el ejercicio:

1. Dibuja el grafo de flujo del código
2. Calcula la complejidad ciclomática de McCabe a partir del grafo
3. Señala cuáles son los caminos independientes de código que hay que probar (el número de caminos coincide con el valor anterior)
4. Identifica los valores de entrada (casos de prueba) necesarios para recorrer cada uno de estos caminos independientes
5. En caso de que en un camino de código haya un bucle, añade nuevos valores de entrada (casos de prueba) para forzar a recorrer el bucle 0, 1 y N veces; si hay bucles anidados, repite lo mismo para cada uno

Comentarios

-

Se puede calcular la complejidad ciclomática a través de tres criterios diferentes (nodos y aristas, condicionales y regiones). Asegúrate de que los valores calculados coinciden. En caso contrario, probablemente estés dibujando un grafo equivocado.

No siempre será posible encontrar datos de entrada para recorrer un camino de código con el mínimo número de pasos. A veces será necesario ejecutar el código de otro(s) camino(s) antes de poder acceder al camino que interesa. Lo importante es llegar a recorrerlo.

Con los bucles sucede algo similar. A veces es simplemente imposible encontrar datos de entrada para que el bucle se ejecute el número de veces deseado, ya que sus condiciones dependen del resto del código.

Recuerda que para probar dos bucles anidados debes forzar el bucle exterior a un valor concreto (por ejemplo, que se ejecute sólo una vez) mientras pruebas el interior para que se ejecute 0, 1 y N veces. Luego, pruebas el bucle exterior 0, 1 y N veces mientras mantienes el interior a un valor concreto. Esto se extiende a múltiples bucles anidados: se prueban de dentro hacia fuera, poniendo todos a un valor fijo excepto con el que se está trabajando.

Ejercicio 7

--

SIGUIENDO LA ESTRATEGIA DE PRUEBAS DE CAJA NEGRA identifica los casos de prueba necesarios para validar un código que hemos subcontratado y al que no tenemos acceso. El objetivo es validar el DNI español:

```
from ejercicio7util import es_dni

print(es_dni("IDESPBAA000589599999999R<<<<<<" +
       "8001014F2501017ESP<<<<<<<<<<<7" +
       "ESPANOLA<ESPANOLA<<CARMEN<<<<<"))
> True
```

Formato del DNI:

```
IDESPBAA000589599999999R<<<<<<
8001014F2501017ESP<<<<<<<<<<<7
ESPANOLA<ESPANOLA<<CARMEN<<<<<
```

Significado:

- *IDESP*, identificador de documento expedido en España
- *BAA000589*, número del soporte ¿oficina?
- *5*, código de control del valor anterior
- *99999999*, número de identificación fiscal
- *R*, código de control del valor anterior (letra del nif)
- *800101*, fecha de nacimiento del titular (formato aa-mm-dd)
- *4*, código de control del valor anterior
- *F*, sexo (valores M/F)
- *250101*, fecha de validez (formato aa-mm-dd)
- *7*, código de control del valor anterior
- *ESP*, ¿país de expedición?
- *7*, código de control de los valores anteriores:
  BAA000589, 5, 99999999, R, 800101, 4, 250101, 7
- *ESPANOLA ESPANOLA, CARMEN*; apellidos y nombre

Pasos para completar el ejercicio:

1. Explicita los requisitos de usuario que hacen válida/inválida una entrada
2. Para cada uno de ellos utiliza las técnicas de CLASES DE EQUIVALENCIAS (CdEq) y de ANÁLISIS DE LOS VALORES LÍMITE (AVL) para identificar valores de entrada (casos de prueba) que deberían producir respuestas válidas y no válidas
3. Comprueba si el código proporcionado por la subcontrata obtiene una respuesta True cuando debe, pero también False cuando corresponda (utiliza la librería unittest de python)

Ejemplo:

```
R1 - Comienza por "IDESP"
    (CdEq) clase válida "IDESP..."; clase no válida "IDFRA..."
R2 - Todas las letras en mayúsculas
    (CdEq) clase válida "IDESPBAA..."; no válida, "IDESPbAA..."
R3 - El número de soporte contiene tres letras y 6 dígitos
    ...
R4 - Las letras del número de soporte...
```

Comentarios
-
El validador sólo necesita las dos primeras líneas de caracteres (60 caracteres); puedes eliminar los apellidos:

```
print(es_dni(
    "IDESPBAA000589599999999R<<<<<<8001014F2501017ESP<<<<<<<<<<<7"))
> True
```

Todas las letras deben estar en mayúsculas.

Lógicamente sólo deberías probar el requisito de mayúsculas en aquellas posiciones que puedan contener letras (en este ejercicio, cuatro). Pero una vez encuentras que el algoritmo acepta minúsculas, puedes dejar esta comprobación y simplemente establecer que "no se respecta el requisito de usuario de sólo mayúsculas".

Para el cálculo de los códigos de control se usan las (26) letras ABCDEFGHIJKLMNOPQRSTUVWXYZ. Para el cálculo de la letra del NIF se eliminan algunas (IOU) que pueden dar lugar a confusión: ABCDEFGHJKLMNPQRSTVWXYZ (23).

SÉ QUE NO ES CORRECTO, pero para aumentar el número de requisitos de entrada considera que el valor del número de soporte está formado por tres letras entre ABCDEFGHIJ y 6 dígitos.

El análisis de valores límite se aplica principalmente sobre valores numéricos (¡ejem! fechas).

El código de ejemplo en ejercicio7util está programado sin hacer demasiadas comprobaciones de seguridad, así que probablemente no siempre genera salidas de error cuando debe. Esos son los casos de prueba que tendrías que enviar a la subcontrata para que corrijan el código.

Te darás cuenta de que las estrategias de clases de equivalencia y análisis de valores límites sirven para identificar entradas válidas y no válidas según los requisitos de usuario, pero no permiten asegurar que los cálculos interiores estén correctos. ¡Tal vez el cálculo de los dígitos de control tenga un error que aparece 1 de cada 100 veces! ¡O 1 de cada millón! Es por eso que la única forma de asegurar que este código es básicamente correcto es la estadística: si con 100 documentos dni los dígitos de control son correctos, y con 1000 se siguen calculando bien, y con 1 millón el código sigue sin fallar... empezamos a pensar que probablemente el algoritmo esté bien hecho.

Ejercicio 8

--

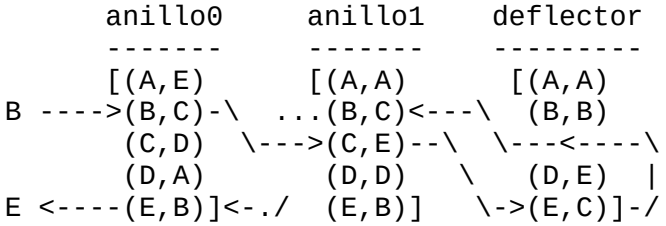
Este ejercicio es un complemento al anterior donde se va a utilizar conocimiento específico del algoritmo implementado (pero todavía sin acceso al código) para generar mejores casos de prueba.

SIGUIENDO LA ESTRATEGIA DE PRUEBAS DE CAJA NEGRA identifica los casos de prueba necesarios para validar un código que hemos subcontratado y al que no tenemos acceso. El objetivo un algoritmo de cifrado. Está basado en la máquina ENIGMA alemana de la segunda guerra mundial, pero muy simplificada. Lo llamaremos RIDDLE.

RIDDLE codifica los mensajes letra a letra. Cada una es convertida a otra a través de CINCO ETAPAS de intercambio. Estos intercambios viene definidos por las estructuras llamadas RING0, RING1 y DEFLECTOR (originalmente las máquinas eran eléctricas y estas estructuras no eran más que cables que conectaban directamente un puerto de entrada a un puerto de salida; en esta simulación informática estas estructuras son una colección de parejas <valor de entrada,valor de salida> elegidas aleatoriamente).

Por simplicidad se muestra el funcionamiento con anillos de tamaño cinco que permiten codificar mensajes formados por cinco letras diferentes (en el mundo real serán mucho mayores):

- primer anillo: [(A,E), (B,C), (C,D), (D,A), (E,B)]
- segundo anillo: [(A,A), (B,C), (C,E), (D,D), (E,B)]
- deflector: [(A,A), (B,B), (C,D), (D,E), (E,C)]
- mensaje de entrada: "BBE"



Puedes ver que cada anillo se usa dos veces, una en cada dirección, mientras que el deflector sólo se usa una vez.

Para eliminar regularidades y hacer más complicado romper el cifrado, RIDDLE tiene otra particularidad (heredada de ENIGMA): cada vez que se codifica una letra, ¡el anillo 0 rota una posición!

- Original: [(A,E), (B,C), (C,D), (D,A), (E,B)]
- Después de la primera letra: [(A,C), (B,D), (C,A), (D,B), (E,E)]
- Después de la segunda letra: [(A,D), (B,A), (C,B), (D,E), (E,A)]

...

Y cuando el anillo 0 ha dado una vuelta completa, el anillo 1 rota una posición:

- Original:
  anillo 0: [(A,E), (B,C), (C,D), (D,A), (E,B)]
  anillo 1: [(A,A), (B,C), (C,E), (D,D), (E,B)]

- Después de codificar cinco letras, el anillo 0 queda en su posición original y el anillo 1 ha rotado una vez:
    anillo 0: [(A,E), (B,C), (C,D), (D,A), (E,B)]
    anillo 1: [(A,C), (B,E), (C,D), (D,B), (E,A)]

Considerando todo esto:

```
from ejercicio8util import riddle

# riddle(anillo0, anillo1, deflector).encode(txt)
print(riddle(
        [('A','E'), ('B','C'), ('C','D'), ('D','A'), ('E','B')],
        [('A','A'), ('B','C'), ('C','E'), ('D','D'), ('E','B')],
        [('A','A'), ('B','B'), ('C','D'), ('D','E'), ('E','C')]
    ).encode("BBE"))
> EAC
```

```
      anillo0     anillo1    deflector
      -------     -------    ---------
      [(A,E)      [(A,A)      [(A,A)
B ---->(B,C)-\  <--(B,C)<-------------\
       (C,D)  \--->(C,E)--\    (C,D)  |
       (D,A)       (D,D)   \   (D,E)  |
E <----(E,B)]<...  (E,B)]   \->(E,C)]-/

    r(anillo0)    anillo1    deflector
      -------     -------    ---------
A <---[(A,C)<--\  [(A,A)      [(A,A)
B ---->(B,D)--\ \  (B,C)       (B,B)
       (C,A)   \ \-(C,E)<------------\
       (D,B)    \->(D,D)------>(D,E)-/
       (E,E)]      (E,B)]      (E,C)]

  r(r(anillo0))   anillo1    deflector
      -------     -------    ---------
      [(A,D)      [(A,A)      [(A,A)
       (B,A)   /---(B,C)<--------<----\
C <----(C,B)<-/ /->(C,E)-\     (C,D)  |
       (D,E)   /   (D,D)  \    (D,E)  |
E ---->(E,C)]-/    (E,B)]  \-->(E,C)]-/
```

Pasos para completar el ejercicio:

1. Explicita los requisitos de usuario que hacen válida/inválida una entrada (características de los anillo y mensaje)
2. Para cada uno de ellos utiliza las técnicas de CLASES DE EQUIVALENCIAS (CdEq) y de ANÁLISIS DE LOS VALORES LÍMITE (AVL) para identificar valores de entrada (casos de prueba) que deberían producir respuestas válidas y no válidas
3. Comprueba si el código proporcionado por la subcontrata obtiene una respuesta correcta cuando debe, pero también incorrecta cuando corresponda (utiliza la librería unittest de python)

Además de lo anterior, ahora tienes suficiente conocimiento interno del algoritmo de cifrado para crear mejores casos de prueba. TRATA DE IDENTIFICAR valores de entrada (casos de prueba) que permitan asegurar...

4. ... que el anillo 0 convierte una letra de entrada en la letra correcta

5. ... que el anillo 1 convierte una letra del anillo 0 en la letra correcta
6. ... que el deflector convierte una letra del anillo 1 en la letra correcta
7. ... que el anillo 1 convierte una letra del deflector en la letra correcta al regresar
8. ... que el anillo 0 convierte una letra del anillo 1 en la letra correcta al regresar
9. ... que el anillo 0 rota correctamente después de cada letra
10. .. que el anillo 1 rota correctamente después de que el anillo 0 haya dado una vuelta completa

En esta ocasión no hace falta respectar los requisitos originales: cambia el tamaño del anillo a lo que prefieras, repite valores, utiliza números negativos para asegurar las rutas de conversión, intenta que el código se rompa si no se ejecutan estas operaciones correctamente...

Ejemplo:

```
    Paso 4
    --
    # conversión correcta; sólo hay una posible combinación
    assert riddle(
        [('A','B'), ('B','x')],
        [('y','y'), ('B','B')],
        [('z','z'), ('B','B')]).encode("A") == 'A'

    # conversión incorrecta; fuera de la combinación anterior
    assert riddle(
        [('A','B'), ('B','x')],
        [('y','y'), ('B','B')],
        [('z','z'), ('B','B')]).encode("B") != 'B'
```

Comentarios
-
Considera los requisitos de usuario evidentes: los anillos (el deflector se considera anillo también) tienen que ser del mismo tamaño, que las letras a codificar dependen del tamaño del anillo, sólo se codifican letras en mayúsculas, los anillos se especificar con parejas de números positivos no repetidos, que cada número tiene que aparecer una única vez a cada lado de cada anillo...

El código de ejemplo en ejercicio8util está programado sin hacer demasiadas comprobaciones de seguridad, así que probablemente no siempre genera salidas de error cuando debe. Esos son los casos de prueba que tendrías que enviar a la subcontrata para que corrijan el código.

Si anteriormente (ejercicio 7) hemos necesitado recurrir a probar un gran volumen de números para asegurar que el algoritmo funciona, en éste (ejercicio 8) utilizamos la descripción de alto nivel del algoritmo para asegurar que los subsistemas que conocemos (anillo 0, anillo 1 y deflector) funcionan como esperábamos. Aunque sigue sin ser una comprobación exhaustiva (que sólo podemos obtener con acceso al código fuente).

Ejercicio 9

--

AMELIA y BASILIO heredan el negocio familiar, una imprenta que sobrevive creando folletos de publicidad de buzón y catálogos fotográficos para el pequeño comercio de su ciudad. Con la ingenuidad de los recién licenciados deciden dar un paso más y extender el negocio hacia el estampado textil (tanto en formato B2B como B2C): bolsas, camisetas, sudaderas, chubasqueros, calcetines, viseras, gorras... y lo que surja.

Ya disponen de un sistema digital para llevar la imprenta. Así que tras años de utilizarlo, tienen una idea de lo más necesario para el nuevo negocio ("IMPRESIONANTE!"). Deciden:

01. El negocio va a tener una página web para promocionarse
02. La página web es actualizada por un administrador del sitio
03. Una persona puede registrarse como cliente en unos pocos pasos fáciles desde la propia web
04. Los clientes tienen acceso 24x7 a secciones específicas de la web orientadas a generar solicitudes de estampado
05. Un cliente podrá solicitar la impresión de logotipos en diferentes tipos de producto (el listado depende de la disponibilidad de producto en el almacén) indicando cantidad, producto, calidad, talla, logotipo, tamaño y tipo de estampado
06. Según el volumen, el cliente se clasifica como PERSONA (menos de 10 prendas), PYME (hasta 1000 prendas) y EMPRESAPLUS (más de 1000 prendas); en cada caso se aplican descuentos al total de 0%, 5% y 10% respectivamente; un responsable del negocio puede cambiar el descuento de un pedido
07. Los clientes tipo EMPRESAPLUS pueden solicitar la participación del diseñador de la casa para optimizar su logotipo
08. Los clientes pagan antes de que comience a ejecutarse su orden de trabajo; se ha registrado con una tarjeta de crédito que ha sido validada haciendo un cargo simbólico
09. Los pedidos superiores a 10 prendas deben ser revisados y aprobados por un responsable en un tiempo máximo de 24 horas
10. El cliente puede cancelar una orden de trabajo antes de que comience a ejecutarse (durante el período de revisión); un responsable del negocio también puede cancelar la orden de un cliente; un administrador del sitio también puede cancelar la orden de un cliente, bajo petición de un responsable; si un cliente solicita varias veces el estampado de un mismo producto (generando varias órdenes de trabajo) se le avisa por si ha cometido un error y se le facilita cancelarlas
11. Los clientes puede actualizar su información; los administradores del sitio tiene acceso a todos los datos de todos los clientes (incluida la información bancaria); pueden actualizar y corregir la información de los clientes, pero necesitan pedir permiso a un responsable; pueden dar de baja y bloquear a clientes, de nuevo bajo supervisión de un responsable
12. Un cliente tiene acceso a todo su historial de pedidos y facturas; esta información puede ser buscada por fecha o producto; el cliente puede solicitar que se le envíe factura en papel en vez de electrónica
13. Los administrativos tienen acceso al listado de clientes y pedidos (pasados y actuales); los administrativos pueden imprimir la factura de cliente para enviársela por correo
14. Toda la información de usuarios, pedidos y facturas se almacena durante al menos cinco años; se hacen dos copias de seguridad todas las semanas
15. Los lunes se generan informes sobre el producto almacenado en el almacén y pedidos pendientes para organizar las compras de material;

estas compras las aprueban los administrativos y se gestionan desde el almacén

¿Quién trabaja en el negocio?

AMELIA y BASILIO son los propietarios con responsabilidades en todos los ámbitos. Ejercen como responsables del negocio para tomar decisiones a alto nivel. Los mejores clientes tratan con ellos directamente.

CÉSAR está contratado por su experiencia en gestión de empresas. Tiene las mismas responsabilidades que los dos propietarios, salvo que no tiene que dar la cara delante de los clientes importantes.

DARÍO gestiona el almacén y organiza las compras de material según le indique el equipo administrativo.

ERNESTO y FELISA son los encargados de mantener y/o extender la plataforma software. Tienen los máximos permisos en ella.

GERARDO, HÉCTOR e ISABEL son el equipo económico y legal (administrativos) del negocio. Acceden a los listados de clientes y pedidos, analizan el riesgo de impago, aprueban la compra de material y le pasan las órdenes a DARÍO.

JOAQUÍN, KEKÉ y LUÍS manejan las máquinas de estampado usando los recursos del almacén. Y, al terminar, empaquetan el resultado para ser recogido por parte del cliente. Pasan esta información al equipo administrativo que son quienes contactan con el cliente.

MANOLO es un becario en prácticas de estampado. Mismas funciones que los anteriores.

NURIA es un cliente que ha solicitado una orden de 2000 impresiones en corbatas y está esperando a que su orden se apruebe.

ÓSCAR es un cliente que ha solicitado estampar un par de polos con su propio logotipo. Le acaban de avisar de que puede pasar a recogerlos cuando quiera.

Pasos para completar el ejercicio:

1. A partir del conjunto inicial de funcionalidades deseables del sistema que han proporcionado, identifica los requisitos funcionales (RF) para la plataforma software y los requisitos no funcionales (RNF). Para estos últimos señala la categoría a la que pertenecen.
2. ¿Hay requisitos que aparenten estar en conflicto con algún otro?
3. Destaca cualquier ambigüedad en los requisitos que necesite ser resuelta. En este caso, identifica a los usuarios o grupo de usuarios que deberían resolverla (aquellos que están más cerca de la funcionalidad).
4. Para el caso de que no se pueda dedicar todo el tiempo deseado al desarrollo de la plataforma software, prioriza los requisitos funcionales para el negocio e identifica la media docena más importante. Justifícalo (un par de frases para cada uno).

Comentarios
-
Esta primera fase de análisis debería permitirnos establecer los requisitos de usuario en un conjunto claro, organizado y no ambiguo de necesidades del cliente. Para este negocio en concreto. Y para la forma concreta de llevar el negocio que tienen en mente.

Clasificación de los RNF (documentación extra en el campus virtual):

- RNF del producto
  - Categoría de usabilidad
  - Eficiencia
  - Dependibilidad
  - Seguridad
- RNF de la organización
  - Entorno
  - Organizacionales
  - Desarrollo
- RNF externos
  - Regulatorios
  - Éticos
  - Legislativos