Manejo de Errores

Unidad 5

Apunte de cátedra

Pensamiento computacional (90)
Cátedra: Camejo

.UBAXXI

Manejo de Errores

Cuando trabajamos con datos es una realidad ineluctable la cantidad de errores que con seguridad se producirán en su uso y manipulación. La situación es tan común que el proceso de depuración de errores es parte natural de la producción de software.

Para tener una medida de la magnitud de esto, pensemos que en Analítica de Datos (una área de aplicación de software de moda); donde normalmente se emplea información con algún nivel de digitalización y seguramente alguna clase, aunque sea básica, de estructuración; los recursos que se invierten en mejorar la calidad de la información (parte importante de esto es la eliminación de errores) son enormes. Este proceso puede insumir hasta la tercera parte del tiempo en un Proyecto de esta clase.

Se conoce también que mientras más tarde se identifica algún error, más costos genera el proceso de corrección. *Es mejor darse cuenta de que una ventana no cabrá en una pared en los planos de una casa nueva que en la obra, con la pared levantada. Y es preferible esto último a descubrir la realidad cuando la pared ya está pintada!*

Cuando de Sistemas y Programas hablamos:

- El ingreso de datos es uno de los puntos de mayor frecuencia de errores
- El ingreso de datos es el lugar más económico donde se pueden detectar y salvar errores

Para intentar disminuir la tasa de errores en el ingreso podemos:

- Simplificar el formato de los datos o adaptarlos a sistemas de lectura automática (códigos de barra, qr, etc.)
- Validar los ingresos

Llamamos Validación de Ingreso o Lectura al proceso por el que se verifica, y eventualmente rechaza, un ingreso de datos buscando detectar posibles errores.

¿Hay posibilidades de que nuestro usuario se equivoque al ingresar los datos? SIEMPRE. Por múltiples factores. Como regla general recomendamos que, siempre que escribamos un programa, pensemos que del otro lado, frente al teclado, habrá alguien proclive a cometer algún error.

¿Hay posibilidades de identificar todos los errores que se cometan en el ingreso?
Algunos si, otros no. De los que se pueda identificar, no todos tendrán una ecuación costo-beneficio que justifique la validación. Entonces ¿Cuándo Validar?

- ✔ Cuando se pueda
- ✔ Cuando convenga

Si en un ejercicio se pide el ingreso de un número cualquiera no podremos validar, al menos con nuestras herramientas; pero si se pide el ingreso de un número negativo, un múltiplo de tres, la edad de una

persona, la de un elefante, una fecha, es decir, sólo subconjuntos dentro del universo de posibilidades, entonces puede ser interesante validar.

Si se solicita el ingreso de la edad de una persona viva seguramente podremos detectar que 300 no es una edad válida, al menos en el siglo XXI, pero no tenemos la posibilidad de saber si la edad (dentro de rango) ingresada de 31 es la edad correcta de esa persona (al menos, no sin cruzar datos). Otro ejemplo es en el ingreso de nombres. La verdad es que resulta difícil saber si un nombre está bien escrito o no, y menos si corresponde a la persona en cuestión.

Entonces, se recomienda validar el ingreso siempre que se pueda y la ecuación costo-beneficio sea positiva. Aunque no podamos garantizar la ausencia total de errores, podemos bajar la cantidad que ingresarán y proliferarán en nuestro programa.

Para validar un ingreso bastará con escribir una o varias condiciones que nos permitan testear esos datos, o identificar la posible presencia de errores.

Y si detectamos un ingreso erróneo ¿Qué debemos hacer?

1. ¿Rechazamos el ingreso y paramos la ejecución del programa?
2. ¿Continúa la ejecución como se pueda ignorando el ingreso?
3. ¿Deja de tratarse el caso que contiene un error?
4. ¿Se pide la colaboración del usuario y se solicita un reingreso?

Claro que solicitaremos el reingreso! ¿Cuántas veces?

La idea es ser tolerantes con el usuario. Es decir que, cuando solicitemos un ingreso, lo validaremos y si detectamos un error permaneceremos dándole nuevas oportunidades al usuario sin avanzar en el programa.

Mostraremos un ejemplo de cómo validar números pares:

```
# Valida el ingreso de un número par
par=int(input('Ingresá un número par: '))
while par<0 or par%2!=0: #también te serviría la condición par<0 or par%2
  par=int(input('Ingresá un número par: '))
print('Tu número es',par)
```

Una ejecución ejemplo podría ser:

```
Ingresá un número par: 11
Ingresá un número par: 8
Tu número es 8
>>>
```

Atención!
Retenemos al usuario en el proceso de ingreso cuando se equivoca, por lo tanto la condición para continuar en la calesita (iterar dentro del bucle de captura) es que se cumpla justo lo que no debe pasar! No debemos escribir la condición de ingreso válido, sino lo contrario. En el ejemplo se testeó que el número ingresado no fuera par.

Con esta simple técnica podemos tener bajo control una gran cantidad de situaciones anómalas y trabajar con datos de buena calidad. Sin embargo hay ingresos de datos que pueden provocar errores fatales y no es nada elegante, y menos profesional, que un programa en producción (en uso real) salte por los aires arrojándonos fuera por un error grave. Muchos de esos errores se pueden evitar con una correcta programación, pero hay otros provocados por el usuario que nos obligan a desarrollar funciones de validación muy complejas o trabajosas. Y no es la idea…
A estas alturas ya te habrá pasado que cuando solicitas el ingreso de un número entero, por ejemplo, y lo **casteas** en la misma línea. Como en el ejemplo:

```
par=int(input('Ingresá un número par: '))
```

Si el usuario se equivoca y sin querer introduce un dato no compatible con un número entero Python enloquece y te acompaña a la salida.

Una forma de manejar elegantemente esta situación sería aceptar el ingreso como string, validar que el ingreso sea compatible con un número y recién castear.

```
# Solicita el ingreso de un número
num=input('Ingresá un número: ')
while not num.isnumeric():
  num=input('Ingresá un número: ')
print('Tu número es',int(num))
```

Miremos una probable ejecución:

```
Ingresá un número: cxc
Ingresá un número: 3.66
Ingresá un número: 22
Tu número es 22
>>>
```

Hay otros métodos de string que te pueden ayudar, tales como isdecimal() o isdigit(). Sin embargo no se comportan siempre adecuadamente. Por ejemplo, tienen problemas reconociendo números negativos.
En ese caso podríamos desarrollar una función recursiva que reconozca el patrón compatible con un número entero en la cadena; un mini **lexi**, para los amigos. Suena sencillo, y lo es, si tienes práctica con **programación recursiva** y sabes diseñar **gramáticas regulares** o **autómatas finitos**. Acá te dejo un ejemplo:

```
def entero(t):
  ''' esta función determina si el texto es compatible con un entero'''
  if len(t)==0:
    return False
```

```
  elif t[0] in '+-':
    return e1(t[1:])
  elif t[0] in '0123456789':
    return e2(t[1:])
  else:
    return False

def e1(t):
  if len(t)==0:
    return False
  elif t[0] in '0123456789':
    return e2(t[1:])
  else:
    return False

def e2(t):
  if len(t)==0:
    return True
  elif t[0] in '0123456789':
    return e2(t[1:])
  else:
    return False

num=input('Ingresá un número: ')
while not entero(num):
  num=input('Ingresá un número: ')
print('Tu número es',int(num))
```

Por suerte en Python (que siempre cuida la salud de los programadores) temenos otro recurso más sencillo. Podemos hacer un manejo elegante de las excepciones o errores. Algo así como capturarlas dentro de nuestro programa para evitar que salte todo por los aires.

Disponemos de la sentencia estructurada **try/except** que funciona básicamente intentando realizar una acción (con **try**, intento con recaudos) y si tiene éxito, el programa prosigue relajadamente, sin embargo, si en el intento emerge un error, la sentencia lo captura y lo trata de una manera conveniente en la clausula **except**.
La sintáxis es la siguiente:
try:

-
-
-

except [$códigoDeError_1,\ldots,códigoDeError_k$]**:**

-
-
-

Mostraremos cómo resolver el problema del ingreso de un entero utilizando esta facilidad:

```
sigue=True
while sigue:
  try:
    num=int(input('Ingresá un número entero: '))
    sigue=False
  except:
    print('Debe ser un número entero')
print('Tu número es',num)
```

Una ejecución posible sería:

```
Ingresá un número entero: hola
Debe ser un número entero
Ingresá un número entero: 1.33
Debe ser un número entero
Ingresá un número entero: -77
Tu número es -77
>>>
```

Varias cosas para tener en cuenta:

- ✔ El manejo de excepciones está dentro de un bucle. Recordemos que le daremos al usuario tantos intentos como necesite, de modo que en este caso, si rechazamos un ingreso, debemos permitirle otro. Todos los que necesite! Por eso solo salimos del bucle cuando se puede ejecutar completamente el bloque vinculado a la clausula **try**.
- ✔ Los códigos de error que se registran en la lista del except son optativos. Si no ponemos ninguno capturará todos los errores que puedan producirse. Si decidimos actuar solo con algunos debemos consignarlos. Si son más de uno, van separados por coma.

Atención!
Los errores deben escribirse exáctamente como los identifica Python. En el ejemplo, el Código de error que pretendemos capturer es: ValueError. Debe anotarse tal cual.

```
sigue=True
while sigue:
  try:
    num=int(input('Ingresá un número entero: '))
    sigue=False
  except ValueError:
    print('Debe ser un número entero')
print('Tu número es',num)
```

En este caso la excepción se activará únicamente si el error que se produce es de ese tipo.