Errores y Excepciones

Las excepciones en Python son una herramienta muy potente que la gran mayoría de lenguajes de programación modernos tienen. Se trata de una forma de controlar el comportamiento de un programa cuando se produce un error.

Esto es muy importante ya que salvo que tratemos este error, el programa se parará, y esto es algo que en determinadas aplicaciones no es una opción válida.

El lidiar con errores de programación tiene (al menos) dos partes. La primera es cuando te metes en problemas porque tu código, aparentemente correcto, se alimenta con datos incorrectos. Por ejemplo, esperas que se ingrese al código un valor entero, pero tu usuario descuidado ingresa algunas letras al azar.

La segunda parte de lidiar con errores de programación se revela cuando ocurre un comportamiento no deseado del programa debido a errores que se cometieron cuando se estaba escribiendo el código. Este tipo de error se denomina comúnmente "bug" (bicho en inglés).

```
[ ]  value = int(input('Ingresa un número natural: '))
     print('El recíproco de', value, 'es', 1/value)
```

```
Ingresa un número natural: d
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-2-4a82bff98c81> in <cell line: 1>()
----> 1 value = int(input('Ingresa un número natural: '))
      2 print('El recíproco de', value, 'es', 1/value)

ValueError: invalid literal for int() with base 10: 'd'
```

Revisando el código anterior, en dónde se puede presentar el error?

Está el código bien escrito?

```
ValueError                                Traceback (most recent call last)

<ipython-input-2-4a82bff98c81> in <cell line: 1>()
----> 1 value = int(input('Ingresa un número natural: '))
      2 print('El recíproco de', value, 'es', 1/value)

ValueError: invalid literal for int() with base 10: 'd'
```

Cómo se podría llegar a mitigar el error utilizando el operador is ?

Qué es una validación previa de datos?

```
esta no es la forma que Python recomienda.
```

ry-except

En Python existe una regla que dice: "es mejor manejar un error cuando ocurre que tratar de evitarlo".

Sin embargo la practica hasta aquí, ha mostrado que ante un error, el programa se detiene y debo corregir, si es el caso, y volver a ejecutar.

Bueno, para eso existe la rama try-except, que consta de dos espacios, un espacio try donde se pueden realizar diferentes operaciones, incluso aquellas que sabemos van a dar una falla y un segundo espacio except donde es posible capturar ese error y darle manejo.

```
try:
    # Es un lugar donde
    # tu puedes hacer algo
    # sin pedir permiso.
except:
    # Es un espacio dedicado
    # exclusivamente para pedir perdón.
```

Como puedes ver, este enfoque acepta errores (los trata como una parte normal de la vida del programa) en lugar de intensificar los esfuerzos para evitarlos por completo.

El mismo programa anterior pero ahora utilizando lo aprendido, no entrega un error propio de Python a la hora de ingresar un 0 o una letra, sino que permite gestionarlo dentro del **except**

```
[ ]  try:
         value = int(input('Ingresa un número natural: '))
         print('El recíproco de', value, 'es', 1/value)
     except:
         print('No se que hacer con', value)
```

```
Ingresa un número natural: 0
No se que hacer con 0
```

- Cualquier fragmento de código colocado entre try y except se ejecuta de una manera muy especial: cualquier error que ocurra aquí dentro no terminará la ejecución del programa. En cambio, el control saltará inmediatamente a la primera línea situada después de la palabra clave reservada except, y no se ejecutará ninguna otra línea del bloque try;
- El código en el bloque except se activa solo cuando se ha encontrado una excepción dentro del bloque try. No hay forma de llegar por ningún otro medio;
- Cuando el bloque try o except se ejecutan con éxito, el control vuelve al proceso normal de ejecución y cualquier código ubicado más allá en el archivo fuente se ejecuta como si no hubiera pasado nada.

Qué tipo de excepción se ha manejado aquí? recuerda otro tipo?

Más de un except en un try

Como se mencionó en el texto inicial, si el usuario ingresa una letra, un 0 o en general, cualquier tipo de dato que no satisfaga la operación, el programa debería presentar un fallo y, si queremos manejar cada fallo de manera independiente, es necesario modificar un poco el bloque.

```
La divsión por 0 da un error llamado ZeroDivisionError

Un valor incorrecto da un error llamado ValueError
```

La cantidad de except no está limitada - pued especificar tantas o tan pocas como sean necesarias, pero ninguna de las excepciones se puede especificar más de una vez.

```
[ ]  try:
        value = int(input('Ingresa un número natural: '))
        print('El recíproco de', value, 'es', 1/value)
    except ValueError:
        print('No se que hacer con', value)
    except ZeroDivisionError:
        print('La división entre cero no está permitida en nuestro Universo.')
```

Excepción predeterminada

Así como pueden existir errores del tipo ZeroDivisionError o ValueError, existen otros tipos como NameError, TypeError, AttributeError, SyntaxError o RuntimeError, ahora, qué pasa si no crea un except para cada uno de los posibles fallos que existen en Python?

Compruebe que pasa si se omite el ZeroDivisionError en el código anterior y se ingresa un 0 en el input
Aquí es donde entra en juego la excepción predeterminada, si no cae en ninguna excepción específica, el programa no se romperá sino que esta última se disparará.

```
[ ]  try:
        value = int(input('Ingresa un número natural: '))
        print('El recíproco de', value, 'es', 1/value)
    except ValueError:
        print('No se que hacer con', value)
    except ZeroDivisionError:
        print('La división entre cero no está permitida en nuestro Universo.')
    except:
        print('Ha sucedido algo extraño, ¡lo siento!')
```

esta vez no tiene un nombre de excepción específico – podemos decir que es anónimo o (lo que está más cerca de su función real) es el por defecto.

Explicación algunas excepciones en Python

ZeroDivisionError

cualquier operación que provoque una división en la que el divisor es cero o no se puede distinguir de cero.

ValueError

Se genera cuando una función (como int() o float()) recibe un argumento de un tipo adecuado, pero su valor es inaceptable.

TypeError

Aparece cuando intentas aplicar un dato cuyo tipo no se puede aceptar en el contexto actual.

No está permitido usar un valor flotante como índice de una lista

```
short_list = [1]
one_value = short_list[0.5]
```

AttributeError

cuando intentas activar un método que no existe en un elemento con el que se está tratando

```
short_list = [1]
short_list.depend(2)
```

El ejemplo intenta hacer uso de un método que no está incluido en las listas.

SyntaxError

Se genera cuando el control llega a una línea de código que viola la gramática de Python.

Es una mala idea manejar este tipo de excepciones en tus programas. Deberías producir código sin errores de sintaxis, en lugar de enmascarar las fallas que has causado.

Ejercitación descubre el error

Qué tipo de error arroja el siguiente código, o no hay error?

```
temperature = float(input('Ingresa la temperatura actual:'))

if temperature > 0:
    print("Por encima de cero")
elif temperature < 0:
    prin("Por debajo de cero")
else:
    print("Cero")
```

¿Qué sucede si ejecuto el código con un valor como 20 o 0?

Aquí se encuentra otro concepto que se puede abordar y es tan importante como el manejo de errores y el la revisión de código mediante debug o debugging.

Hablar con los estudiantes sobre la estrategia Rubber duck debugging puede ser un buen punto para desde una interacción más natural, abordar un problema o una revisión.

El IDE cuenta con una opción para agregar puntos de depuración o revisar de forma general todo el código. Recuerde que como Python trabaja con un intérprete, solo es revisado el código que en tiempo de ejecución es invocado, si allí no hay errores, el programa corre con normalidad.

Random

Este módulo implementa generadores de números pseudoaleatorios para varias distribuciones, lo que quiere decir es que los números generados no son realmente aleatorios.

Cuenta con un listado de métodos que puede consultar aquí

Para trabajar con el módulo Python es necesario importarlo previamente

El módulo random cuenta con un método seed que permite obtener siempre el mismo número o grupo de números aleatorios a diferencia de no establecerla, lo que provoca que en cada ejecución se generen nuevos números

```
[ ]  import random

     #Seleccionar un elemento aleatorio de una lista
     listado_estudiantes = ["Alice", "Joha", "Damien", "Javi"]
     print(random.choice(listado_estudiantes))



     print(random.random())
     print(random.random())

     #random.seed(4)
     #print(random.random())
     #print(random.random())
```

```
Damien
0.7797455502499977
0.8268716790242738
```