Seminario de Lenguajes - Python

Cursada 2024

🫀 Clase 6. Introducción al manejo de excepciones en Python

📈 Probemos el siguiente código:

In [ ]:
```
archivo = open("pp.txt")
```

❌¡Esto no puede pasar nunca en nuestros programas!

- ¿Por qué?
- ¿En qué casos puede pasar este error?

🫀 ¿Qué es un excepción?

> Una **excepción** es un acontecimiento, que ocurre **durante la ejecución** de un programa, que **interrumpe** el **flujo normal** de las instrucciones del programa.

¿Qué situaciones pueden producir excepciones?

- ✅ Abrir un archivo que no existe o donde no tenemos permisos adecuados.
- ✅ Acceder al contenido de un archivo que no respeta el formato.
- ✅ Invocar a un método o función que no fue definida.
- ✅ Referirse a una variable que no fue definida.
- ✅ Mezclar tipos de datos sin convertirlos previamente.
- Etc.

🚧 Excepciones "sin manejar"

```
1 archivo = open("pp.txt")

---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-6-1a34c3f0ef34> in <module>
----> 1 archivo = open("pp.txt")

FileNotFoundError: [Errno 2] No such file or directory: 'pp.txt'
```

Se intentó abrir un archivo que no existe.

```
    350
    351 """
    352 try:
--> 353     obj, end = self.scan_once(s, idx)
    354 except StopIteration as err:
    355     raise JSONDecodeError("Expecting value", s, err.value) from None

JSONDecodeError: Expecting ',' delimiter: line 4 column 9 (char 31)
```

Se intenó cargar un formato json mal formado

🧯 ¿Qué debemos investigar para trabajar con excepciones?

Primero: ¿el lenguaje de programación tiene soporte para el manejo de excepciones?

- Si no presenta ningún mecanísmo para esto, podríamos simularlo con otros recursos. Ejemplo: Pascal o C.
- Si provee mecanismos para el manejo de excepciones: ¿cuáles? Ejemplo: **Python**, Javascript, Java, Ruby, etc.

📌 Si el lenguaje provee manejo de excepciones...

- ¿Qué acción se toma después de levantada y manejada una excepción? ¿Se continúa con la ejecución de la unidad que lo provocó o se termina?
- ¿En qué casos es más legible excepciones y cuándo control de flujo?
- ¿Cómo se alcanza una excepción?
- ¿Cómo se definen los manejadores de excepciones?
- ¿Qué sucede cuando no se encuentra un manejador para una excepción levantada?
- ¿El lenguaje tiene excepciones predefinidas?
- ¿Podemos levantar en forma explícita una excepción?
- ¿Podemos crear nuestras propias excepciones?

🧯 Excepciones en Python

```
try:
    sentencias
except nombreExcepción:
    sentencias
except nombreExcepción:
    sentencias
except:
```

- +Info

Veamos un ejemplo

In [ ]:
```
file_name = 'pp.txt'
try:
    file_data = open(file_name)
except FileNotFoundError:
    print(f'El archivo {file_name} no se encuentra')
```

In [ ]:
```

```

📝 Analicemos el siguiente código:

In [ ]:
```
import json
from pathlib import   Path
file_route = Path('files').resolve()
file_name = 'libros_exception.json'
try:
```

```
    archivo = open(file_route/file_name)
    data_books = json.load(archivo)
except FileNotFoundError:
    print(f'El archivo {file_name} no se encuentra')
```

📌 ¿Por qué da error si hay un bloque try...except?

Podríamos manejar las posibles excepciones:

In [ ]:
```
import json
from pathlib import   Path
file_route = Path('files').resolve()
file_name = 'libros_exception.json'
try:
    archivo = open(file_route/file_name)
    data_books = json.loads(archivo)
except (FileNotFoundError, JSONDecodeError):
    print(f'Al abrir el archivo {file_name} hubo un error')
```

Tenemos dos problemas:

- JSONDecodeError es una excepción del módulo **json**, por lo tanto debemos indicar el nombre del **módulo.exception**

`json.JSONDecodeError`

- El mensaje de error es geńerico y no indica claramente qué error sucedió

📌 Agregamos los manejadores

In [ ]:
```
import json
from pathlib import   Path
file_route = Path('files').resolve()
file_name = 'libros_exception.json'
try:
    archivo = open(file_route/file_name)
    data_books = json.load(archivo)
except FileNotFoundError:
    print(f'El archivo {file_name} no se encuentra')
except json.JSONDecodeError:
    print(f'Al abrir el archivo {file_name} hubo un error con el formato')
```

🚧 ¿Cómo buscamos el manejador?

In [ ]:
```
file_books_csv = 'libross.csv'
import csv
with open(file_route/file_books_csv) as csvfile:
    csv_reader = csv.reader(csvfile, delimiter=';')
```

El error que se produce muestra las excepciones **predefinidas** de Python base.

¿Encontramos excepciones sólo en errores de archivos?

In [ ]:
```
file_books_csv = 'books.csv'
import csv
with open(file_route/file_books_csv) as csvfile:
    csv_reader = csv.reader(csvfile, delimiter=';')
```

```
    for row in csv_reader:
        if row[3] > '2024':
            print(row)
    header, data = next(csv_reader), list(csv_reader)
```

¿Qué sucedió?

- Recorrimos una vez el iterador **csv_reader**.
- Cuando queremos asignar los valores a las variables no tira la excepción predefinida **StopIteration**.
- Excepción que debe ser controlada con programación.

📌 ¿Qué podemos decir del siguiente ejemplo? ¿Cómo es más legible?

Se define una función para abrir el archivo.

In [ ]:
```
import json
from pathlib import   Path
def read_data(data_file):

    file_route = Path('files').resolve()
    file_name = data_file
    try:
        archivo = open(file_route/file_name)
        data_books = json.load(archivo)
        return data_books
    except FileNotFoundError:
        print(f'El archivo {file_name} no se encuentra')
    except json.JSONDecodeError:
        print(f'Al abrir el archivo {file_name} hubo un error con el formato')
```

- Leemos los datos llamando a la función

In [ ]:
```
file_name = 'libros.json'
```

In [ ]:
```
data_books = read_data(file_name)
```

- Y ahora accedemos leyendo los datos

In [ ]:
```
try:
    print(f"autores argentinos {data_books['argentinos']}")
except KeyError:
    print("Con try: No hay autores argentinos")

if 'argentinos' in data_books:
    print(f"autores argentinos {data_books['argentinos']}")
else:
    print("Con if : No hay autores argentinos")
```

📌 ¿Cuál les parece más legible?

Hay diferentes formas de verificar la existencia de un dato, por ejemplo una key:

- El manejo de excepciones hizo que al intentar acceder al diccionario, con una clave inexistente, el programa **no se rompa**.

- La estructura condicional verifica si existe y también evita que se **rompa**.
- En ambos **informamos cuál es el problema**.

Los posibles errores generan las diferentes excepciones que están definidas en:

- **predefinidas** en el código base de Python.
- o en las librerías que **importamos**, como es el caso de json sobre el formato.

🧯 Flujo de las excepciones

- Veamos ahora la misma función pero sin capturar la excepción **json.JSONDecodeError** dentro de la función:

In [ ]:
```
def read_data(data_file):

    file_route = Path('files').resolve()
    file_name = data_file
    try:
        archivo = open(file_route/file_name)
        data_books = json.load(archivo)
        return data_books
    except FileNotFoundError:
        print(f'El archivo {file_name} no se encuentra')
```

In [ ]:
```
file_name = 'libros_exception.json'
try:
    data_books = read_data(file_name)
except:
    print('Errores varios')
```

📌¿Qué sucedió?

- La excepción json.JSONDecodeError no se levantó dentro de la función **read_data**.
- Python **Busca estáticamene** si el bloque está encerrado en otro bloque try except.
- Al no encontrar un manejador para esa excepción en la función ...
  - **Busca dinámicamente** a quién llamó a la función.
  - Al encontrar otro bloque que contempla excepciones **generales**, la captura y termina el programa sin error.

📌 ¿Cómo es la forma de **propagación** que utiliza Python?

- Primero busca **estáticamente**.
- Si no se encuentra, busca **dinámicamente** a quién llamó a la función.
- Si encuentra un manejador lo captura.
- Si no encuentra un manejador... entonces termina el programa ... con error..

📌 Ya respondimos algunas de las preguntas iniciales:

- **¿Qué acción se toma después de levantada y manejada una excepción? ¿Se continúa con la ejecución de la unidad que lo provocó o se termina?**
- **¿En qué casos es mas legible excepciones y cuándo flow-control?**

- **¿Cómo se alcanza una excepción?**
- **¿Cómo especificar los manejadores de excepciones que se deben ejecutar cuando se alcanzan las mismas?**
- **¿Qué sucede cuando no se encuentra un manejador para una excepción levantada?**
- **¿El lenguaje tiene excepciones predefinidas?**
- ¿Podemos levantar en forma explícita una excepción?
- ¿Podemos crear nuestras propias excepciones?

🧯 Y en Streamlit?

Streamlit permite capturar excepciones e informar con mensajes específicos los errores, veamos un ejemplo:

```
file_areas = 'area_prootegida.csv'
try:
    with open(files_directory/file_areas) as file:
        csv_reader = csv.reader(file)
        try:
            header = next(csv_reader)
            data = list(csv_reader)
        except csv.Error as e:
            st.error(f"Error al leer el archivo CSV: {e}")
except FileNotFoundError:
    data = []
    st.error(f"Archivo {file_areas} no encontrado")
```

- ✅ El flujo de try...except es igual y puede utilizarse el método **st.error** para mostrar el mensaje.
- ✅ En la mayoría de los casos no es necesario **reinicar** la aplicación.
- ✅ Los mensajes se muestran en la página web.

📌 Uso de excepciones

Vimos hasta ahora ejemplos de:

- ✅ Archivo no encontrado.
- ✅ Error en el formato del archivo leído.
- ✅ Error en iteradores.
- ✅ Opcionalmente key inexistente en un diccionario.

Otro ejemplo de posibles errores inesperados es cuando trabajamos con datos que no generamos nosotros sino que leemos de un sitio externo o apis, los errores que pueden suceder:

- ✅ La lectura no siempre es posible.
- ✅ El formato puede ser modificando sin saberlo.
- ✅ Los datos no respetan los tipos de datos.
- Etc.

🚨 Desafío

> Analizar el código del la aplicación que procesa el archivo **areas_protegidas.csv** que dejamos como material la semana pasada y veamos en qué casos se puede utilizar el

> manejo de excepciones que realiza.

🧰 La sentencia completa

```
try:
    sentencias
except excepcion1, excepcion2:
    sentencias
except:
    sentencias
else:
    sentencias
finally:
    sentencias
```

📌 Veamos este ejemplo sencillo

In [ ]:
```
file_books_csv = 'libross.csv'
import csv


try:
    with open(file_route/file_books_csv) as csvfile:
        csv_reader = csv.reader(csvfile, delimiter=';')
except FileNotFoundError:
    print("El archivo no se encuentra")
else:
    print("Este mensaje se imprime porque  NO se levantó la excepción")
finally:
    print("Este mensaje se imprime SIEMPRE")
```

Entonces, ¿para qué usamos else y finally?

🧰 else y finally

> Se utiliza la cláusula **else** para incluir el código que se debería ejecutar si **no se levanta ninguna excepción** en el bloque try..except.

> Se utiliza la cláusula **finally** para incluir el código que se ejecuta **siempre**, independientemente si se levanta o no alguna excepción en el bloque try..except

📌 Observemos el bloque finally en este otro ejemplo:

In [ ]:
```
file_route = Path('files').resolve()
file_name = 'libros_exception.json'
try:
    print("Entrando al primer try ...")
    try:
        with open(file_route/file_name) as file:
            data_books = json.load(file)
        print('No hubo error sigo con mi programa')
    except FileNotFoundError:
        print(f'El archivo {file_name} no se encuentra')

    finally:
        print("Saliendo del primer try ... ")
```

```
except json.JSONDecodeError:
        print(f"TRY EXTERNO: error de formato de json en  {file_name}")
print('Sigo con mi programa....')
#print(data_books)
```

- La sentencia dentro de **finally** se ejecuta a pesar de la excepción.

📌 ¿Y el else?

In [ ]:
```
file_route = Path('files').resolve()
file_name = 'libros.json'
try:
    with open(file_route/file_name) as file:
        data_books = json.load(file)

except FileNotFoundError:
    print(f'El archivo {file_name} no se encuentra')
except json.JSONDecodeError:
    print(f'Al abrir el archivo {file_name} hubo un error con el formato')
else:
    print('No hubo error dentro del try sigo con mi programa')
finally:
    print("Siempre me ejecuto")
```

🧰 Podemos levantar explícitamente excepciones

In [ ]:
```
file_route = Path('files').resolve()
file_name = 'libros.json'
try:
    print ('Entramos al bloque try')
    with open(file_route/file_name) as file:
        data_books = json.load(file)
    countries = ['Colombia', 'Brasil', 'Chile', 'Argentina', ]
    for country in countries:
        print(country)
        if country not in data_books.keys():
            raise KeyError
        else:
            print(data_books[country])
    print('Continuamos con el proceso..')
except KeyError:
    data_books[country] = 'NUEVO'
data_books[country]
```

- La sentenica **raise** levantó una excepción explícitamente definida: **KeyError**.
- El bloque la captura con la sentencia except.

También es posible utilizar raise para:

In [ ]:
```
import sys
file_name = 'libros_exception.json'

def read_data(data_file):
    file_route = Path('files').resolve()
    try:
        with open(file_route/data_file) as file:
            data_books = json.load(file)
```

```
    except:
        print('Se levantó un excepción')
        #print(f'detalle: {sys.exc_info()}')
        raise


try:
    read_data(file_name)
except:
    print('No se puede leer el archivo')

try:
    read_data('hola.json')
except:
    print('No se puede abrir el archivo')
```

¿Qué excepción se levantó?

Raise

- Permite generar un traceback de la excepción que se generó.
- Vuelve a lanzar la última excepción que estaba activa en el ámbito actual.

  > las excepciones de los try que llamaron a la función

- Documentación.
- **sys.exc_info()** almacena el estado del hilo de ejecución, el contexto de la excepción.

In [ ]:
```
file_route = Path('files').resolve()
file_name = 'libros.json'
try:
    with open(file_route/file_name) as file:
        data_books = json.load(file)
    if 'Brasil' not in data_books.keys():
        raise
except KeyError:
    print("Manejando KeyError")

read_data(file_name)
```

- Si no hay ninguna excepción activa en el alcance actual, se lanza **RuntimeError** que indica que se trata de un error.

🎒 Algunas de las excepciones predefinidas (Built-in)

- **ImportError**: error con importación de módulos.
- **ModuleNotFoundError**: error por módulo no encontrado.
- **IndexError**: error por índice fuera de rango.
- **KeyError**: error por clave inexistente.
- **NameError**: error por nombre no encontrado.
- **SyntaxError**: error por problemas sintácticos
- **ZeroDivisionError**: error por división por cero.
- **IOError**: error en entrada salida.

Listado completo

📌 En resumen:

```
try:
    sentencias
except excepcion1, excepcion2:
    sentencias
except excepcion3 as variable:
    sentencias
except:
    sentencias
else:
    sentencias
finally:
    sentencias
```

🚨 Repasemos los desafíos de Streamlit que quedaron en el material

Seguimos la próxima ...