Anuncios

1. Recuerden contestar la ECA.
2. Encuesta de medio semestre
3. Actividad formativa. No olviden el *feedback*.

Excepciones

Semana 07 - Jueves 3 de octubre de 2019

Excepción

Situación anómala o inesperada que ocurre durante un proceso de cómputo

```
profesores = ["Fernando","Antonio","Cristian","Vicente"]
print(profesores[4])
print("es el profesor de la cuarta sección")
```

```
IndexError                                Traceback (most recent call last)
<ipython-input-13-9cc76db01104> in <module>
      1 profesores = ["Fernando","Antonio","Cristian","Vicente"]
----> 2 print(profesores[4])
      3 print("es el profesor de la cuarta sección")

IndexError: list index out of range
```

Manejo de excepciones

- try
- except
- else
- finally

Manejando excepciones

```
try:
    profesores = ["Fernando","Antonio","Cristian","Vicente"]
    print(profesores[4])
    print("es el profesor de la cuarta sección")
except IndexError as err:
    print("Ocurrió una excepción...")
print("Pero la vida sigue")
```

```
Ocurrió una excepción...
Pero la vida sigue
```

try - except - else - finally

```
try:
    # Probamos si es posible realizar la operación
    resultado = dividir(10,0)
    print("Esta línea no se ejecuta si se produce una excepción antes.")

except (ZeroDivisionError, TypeError) as error:
    # Este bloque opera para los tipos de excepciones definidos
    print("Revise los datos de entrada. ¡No son int o bien el denominador es 0!")

except ValueError as error:
    # Este bloque sólo maneja excepciones del tipo ValueError
    print("Los valores ingresados son negativos")

else:
    # Como no hubo excepciones puede retornar normalmente el resultado
    # En este caso, si se coloca un return después de la operación y
    # esta es correcta, entonces nunca llegará a este punto.
    print("¡Todo OK!, no hay errores con los datos")

finally:
    print("Recuerde SIEMPRE usar excepciones para manejar los errores\n")
```

Lanzar excepciones

- `raise`

raise

```
def parsear_coordenada(coordenada_como_string):
    # Se espera un string en el formato: ‘400, 300’

    if not isinstance(coordenada_como_string, str):
        # Aquí se genera la excepción y se incluye información para el usuario
        raise TypeError("¡Input debe ser un string!")

    if “,” not in coordenada_como_string:
        # Aquí se genera la excepción y se incluye información para el usuario
        raise ValueError("¡Input debe contener una coma!")

    x, y = coordenada_como_string.split(",")
    return (int(x), int(y))
```

```
parsear_coordenada("300, 400")
parsear_coordenada("600, 800")
parsear_coordenada("300 - 400")
```

raise

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-2-2afca3653183> in <module>
      1 parsear_coordenada("300, 400")
      2 parsear_coordenada("600, 800")
----> 3 parsear_coordenada("300 - 400")

<ipython-input-1-e05115d47ad5> in parsear_coordenada(coordenada_como_string)
      9     if ",” not in coordenada_como_string:
     10         # Aquí se genera la excepción y se incluye información para el usuario
---->11         raise ValueError("¡El input debe contener una coma!")
     12
ValueError: ¡El input debe contener una coma!
```

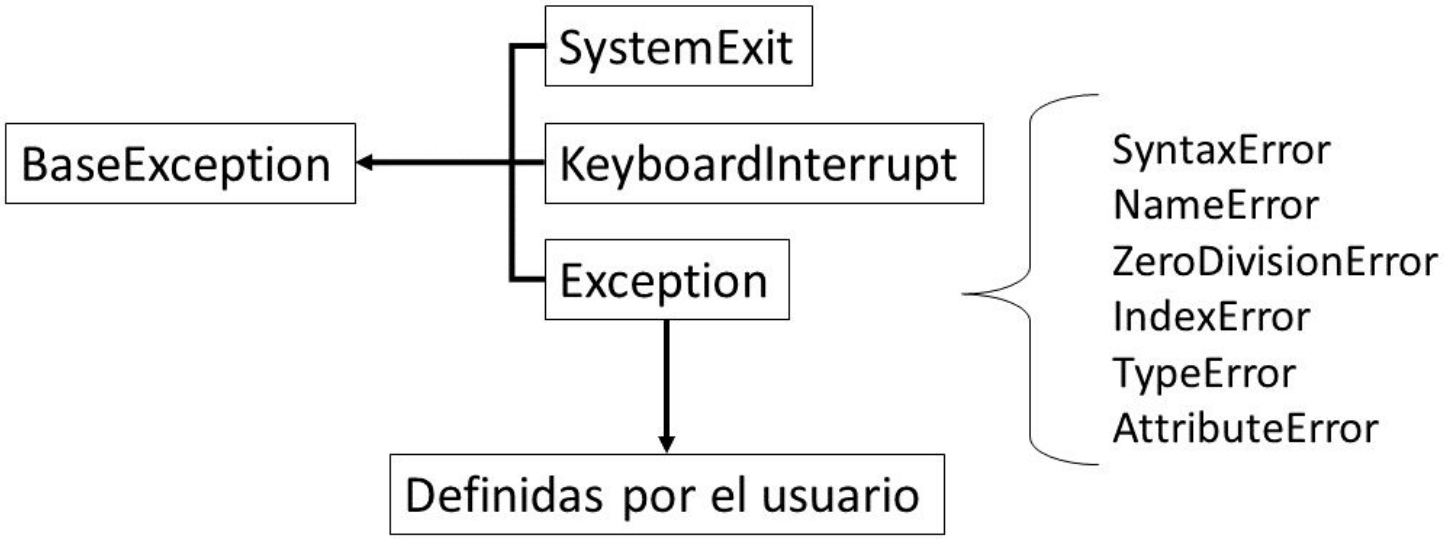
Tipos de excepciones

- SyntaxError
- NameError
- ZeroDivisionError
- IndexError
- TypeError
- AttributeError
- KeyError
- ValueError

Excepciones Personalizadas

- BaseException

Excepciones Personalizadas



Excepciones Personalizadas

```
class ErrorTransaccion(Exception):

    def __init__(self, fondos, gasto):
        super().__init__(f"El dinero en la billetera no alcanza para pagar ${gasto}")
        self.fondos = fondos
        self.gasto = gasto

    def exceso(self):
        return self.gasto - self.fondos
```

Excepciones Personalizadas

```
class ErrorTransaccion(Exception):

    def __init__(self, fondos, gasto):
        super().__init__(f"El dinero en la billetera no alcanza para pagar ${gasto}")
        self.fondos = fondos
        self.gasto = gasto

    def exceso(self):
        return self.gasto - self.fondos

class Billetera:

    def __init__(self, dinero):
        self.fondos = dinero

    def pagar(self, gasto):
        if self.fondos - gasto < 0:
            raise ErrorTransaccion(self.fondos, gasto)
        self.fondos -= gasto
```

Excepciones Personalizadas

```
class Billetera:

    def __init__(self, dinero):
        self.fondos = dinero

    def pagar(self, gasto):
        if self.fondos - gasto < 0:
            raise ErrorTransaccion(self.fondos, gasto)
        self.fondos -= gasto

b = Billetera(1000)

try:
    b.pagar(1500)

except ErrorTransaccion as err:
    print(f"Error: {err}. Hay un exceso de gastos de ${err.exceso()}.")
```

Excepciones Personalizadas

```
class Billetera:

    def __init__(self, dinero):
        self.fondos = dinero

    def pagar(self, gasto):
        if self.fondos - gasto < 0:
            raise ErrorTransaccion(self.fondos, gasto)
        self.fondos -= gasto

b = Billetera(1000)

try:
    b.pagar(1500)

except ErrorTransaccion as err:
    print(f"Error: {err}. Hay un exceso de gastos de ${err.exceso()}.")
```

```
> Error: El dinero en la billetera no alcanza para pagar $1500. Hay un exceso de
gastos de $500.
```

Veamos código

(Hora de abrir el editor)

En este ejemplo veremos:

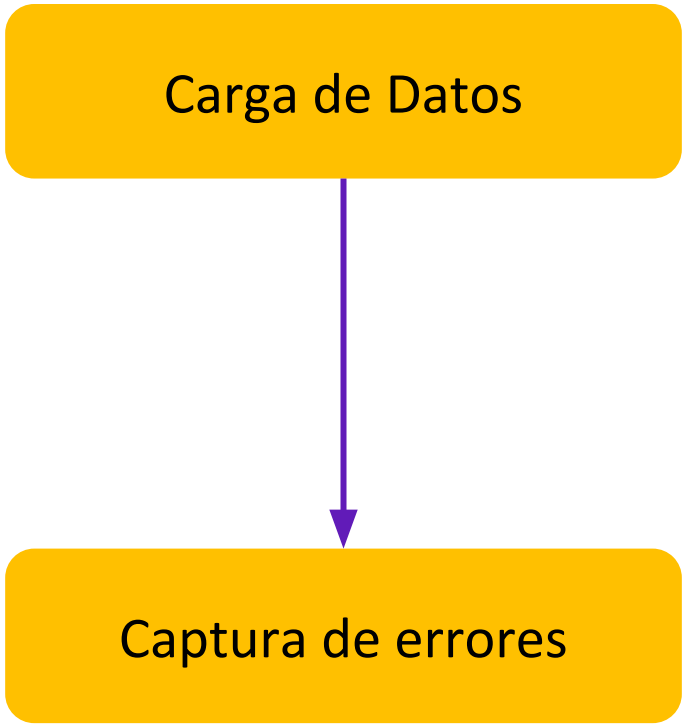
- Un ejemplo de menú que utiliza excepciones para controlar el *input* del usuario

Actividad

1. En el *syllabus*, vayan a la carpeta “Actividades” y descarguen el enunciado de la actividad 6 (AC06) https://github.com/IIC2233/syllabus
2. Trabajen **individualmente** hasta las 16:30.
3. Recuerden hacer *commit* y *push* cada cierto tiempo.

Cierre

Diagrama de flujo de AC




```
cargar_datos("registro_ofizial.json",
"conductores.csv")
DCConductor(registro_oficial,
conductores)
```

```
dcconductor.chequear_celular(conductor)
dcconductor.chequear_rut(conductor)
dcconductor.chequear_nombre(conductor)
```

Excepciones

¿Se podría haber hecho la AC sin el manejo de Excepciones?

Las excepciones nos permiten capturar e identificar posibles errores.

Sin hacer manejo de excepciones, nuestro programa puede estar propenso a fallas.

Main

```
# main.py
# Carga de datos
...
...
if registro_oficial and conductores:
    contador = 0
    for conductor in dcconductor.conductores:
        try:
            dcconductor.chequear_celular(conductor)
            dcconductor.chequear_nombre(conductor)
            dcconductor.chequear_rut(conductor)
            dcconductor.chequear_patente(conductor)
            dcconductor.seleccionados.append(conductor)
...
...
...
```

Main

```
# main.py
...
        except ValueError as err:
            mensaje = err.args[0]
            contador += 1
            print(mensaje)
         except KeyError as err:
            mensaje = err.args[0]
            contador += 1
            print(mensaje)
         except ErrorPatente as err:
            print(err.mensaje)
            contador += 1
print(f"La cuenta de datos erroneos fue: {contador}")
```

De acuerdo al tipo de excepcion capturada podemos tomar una acción apropiado y enviar un mensaje útil al usuario.

DCConductor

```
# dcconductor.py
class DCConductor:
    ...
    def chequear_rut(self, conductor):
        rut = conductor.rut
        if “-” not in rut:
            raise ValueError(f“El rut {rut} no contiene guion.”)
        elif rut[-2] != “-”:
            raise ValueError(f“El rut {rut} no tiene guion en
                                penúltima posición”)
        elif “.” in rut:
            raise ValueError(f“El rut {rut} contiene puntos”)
```

Levantar el tipo de excepcion apropiado nos permite manejarlas de manera diferente, e indicar al usuario cuál fue el error.

Excepción personalizada

```
# excepcion_patente.py
class ErrorPatente(Exception):
    def __init__(self, conductor):
        self.mensaje = f"La patente {conductor.patente} no es la
registrada para {conductor.nombre}."
```

```
# dcconductor.py
class DCConductor:
    ...
    def chequear_patente(self, conductor):
        if conductor.patente != self.registro_oficial[conductor.nombre]:
            raise ErrorPatente(conductor)
```

Utilizar una excepción personalizada nos permite adaptarnos a situaciones que no calzan con las excepciones existentes.

Recordatorios

1. **Terminen la actividad**, les servirá para aprender bien la materia y utilizarla efectivamente cuando tengan la necesidad o la oportunidad
2. Contesten la auto-evaluación hasta **mañana a las 23:59**