Programación 1

Excepciones

Ing. Verónica Galati
Ing. Fernando Basteiro

¿Qué es una excepción?

Una excepción es un problema ocurrido durante la ejecución de un programa.

```
>>> suma = 15
>>> cant = 0
>>> promedio = suma / cant
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
ZeroDivisionError: division by zero
```

UADE

Excepciones conocidas

- IndexError
- NameError
- ValueError
- ZeroDivisionError
- SyntaxError
- IndentationError
- TypeError
- AtributeError

UADE

Otras excepciones

- AssertionError
- IOError
- KeyError
- KeyboardInterrupt
- MemoryError
- ModuleNotFoundError
- RecursionError

Control de errores tradicional

```
if cant != 0:
    print(f"El promedio es: {suma/cant:4.2f}")
else:
    print(f"La cantidad ingresada es 0")
```

Una parte del código está dedicada a controlar posibles situaciones de error.
Actúa de forma **preventiva**: Evita que el error ocurra.

UADE

Captura de excepciones

```
try:
    print(f"El promedio es: {suma/cant:4.2f}")
except ZeroDivisionError:
    print(f"La cantidad ingresada es 0")
```

La lógica del programa es independiente del control de errores.
Actúa de forma **correctiva**, intentando solucionar el problema luego de ocurrido.

UADE

Estructura Try-Except

```
try:
    <Bloque de código que puede generar errores>
    ...
    ...
except <Excepcion a capturar>:
    <Bloque de código para tratar el error generado>
    ...
    ...
```

Bloque protegido

UADE

Funcionamiento

Se ejecuta primero el bloque protegido.
Si se produce algún error durante la ejecución del mismo, se pasa automáticamente al bloque except que corresponda a la excepción generada.
Si no se produce ningún error, el bloque except es ignorado.

El bloque try-except puede tener tantos except como sea necesario, sin repetir ningún tipo de excepción.

UADE

Funcionamiento

El except puede no estar seguido de un tipo de excepción, en cuyo caso capturará cualquier excepción generada, aunque **no es recomendado** dado que no permite identificar el error.

UADE

Ejemplo

```
import random
lista = [random.randint(1,20) for i in range(random.randint(5,10))]
try:
    posicion = int(input("Ingrese posición del elemento a mostrar: "))
    print(f"El valor en la posición {posicion} es: {lista[posicion]}")
except ValueError:
    print("No se ingreso un número")
except IndexError:
    print("La lista no tiene tantos elementos")
```

UADE

Otras opciones del except

Si a más de una excepción se le va a dar el mismo tratamiento, puede colocarse más de un tipo de excepción en la misma cláusula except.

```
try:
    cant = int(input("Ingrese la cantidad: "))
    promedio = suma / cant
except (ValueError, ZeroDivisionError):
    print("El valor ingresado no es correcto")
```

UADE

Otras opciones del except

Es posible capturar el mensaje de error asociado a la excepción generada utilizando la instrucción **as**.

```
>>> int("aaa")
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'aaa'
>>> try:
        num = int("aaa")
    except ValueError as msg:
        print(msg)
invalid literal for int() with base 10: 'aaa'
```

UADE

Cláusula else

Luego de las cláusulas except, puede agregarse una cláusula else, cuyo bloque de código solo se ejecutará si el bloque protegido se ejecuto sin generar ningún error.

```
try:
    porcentaje = (cant / total) * 100
except ZeroDivisionError:
    print("No puede calcularse el porcentaje")
else:
    print(f"El porcentaje es {porcentaje:5.2f}%")
```

UADE

Cláusula finally

Otra cláusula opcional es finally, cuyo bloque de código se ejecutará siempre, sin importar si hubo un problema durante la ejecución del bloque protegido.

```
try:
    porcentaje = (cant / total) * 100
except ZeroDivisionError:
    porcentaje = 0.0
finally:
    print(f"El porcentaje es {porcentaje:5.2f}%")
```

UADE

Instrucción raise

Permite provocar una excepción. **Puede incluir o no el mensaje de error asociado.**

```
>>> raise ValueError("Error de valor generado")
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
ValueError: Error de valor generado
```

UADE

Instrucción raise

La instrucción raise puede utilizarse también para relanzar la última excepción producida.

```
>>> try:
        num = int("abc")
    except ValueError as msg:
        print("No se pudo guardar el numero")
        print(msg)
        raise
No se pudo guardar el numero
invalid literal for int() with base 10: 'abc'
Traceback (most recent call last):
  File "<pyshell>", line 2, in <module>
ValueError: invalid literal for int() with base 10: 'abc'
```

UADE

Instrucción assert

Se utiliza junto con una operación lógica, que en caso de ser falsa, genera una excepción del tipo **AssertionError**.

```
>>> assert True
>>> assert 5 > 3
>>> assert False
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
AssertionError
>>> assert False, "Mensaje"
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
AssertionError: Mensaje
```

UADE

Jerarquía de excepciones

Los tipos de excepciones están organizados jerárquicamente, de modo que distintas excepciones son a su vez de un mismo tipo más genérico.

```
try:
    capacidadTotal = int(input("Stock máximo: "))
    stockActual = int(input("Stock actual: "))
    if stockActual > capacidadTotal:
        raise OverflowError("El stock supera la capacidad máxima")
    porcOcupado = stockActual * 100 / capacidadTotal
    print(f"El alamacén está a un {porcOcupado:.2f}% de su capacidad")
except ValueError:
    print("El valor ingresado es incorrecto")
except ArithmeticError as msg:
    print(msg)
```

UADE

Excepciones personalizadas

Se pueden generar nuevos tipos de excepción para utilizar con la instrucción raise.

```
class EmptyListError(Exception):
    pass

def promedioLista(lista):
    if len(lista) == 0:
        raise EmptyListError("La lista esta vacía")
    else:
        return sum(lista) / len(lista)
```

UADE