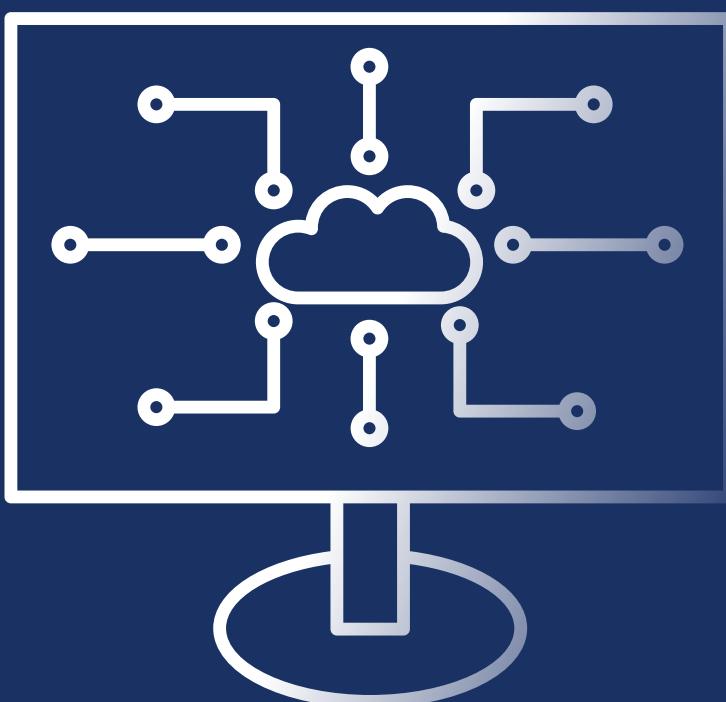


Programación I



Lic. Julia Monasterio
marmonasterio@uade.com.ar



Unidades

-  **Unidad 1:** Repaso de matrices, estructura, operaciones y uso en funciones
-  **Unidad 2:** Implementación y gestión en proyectos en Git
-  **Unidad 3:** Listas avanzadas, cadena de caracteres y expresiones regulares.
-  **Unidad 4:** Manipulación avanzada de diccionarios, tuplas y conjuntos.
-  **Unidad 5:** Excepciones y pruebas unitarias
-  **Unidad 6:** Archivos
-  **Unidad 7:** Recursividad

Clase N°7

TEMAS

- Excepciones

Excepciones

Que es una excepción?

- Una excepción es la indicación de un problema ha ocurrido durante la ejecución de un programa
- Dado que la **regla** es que los programas se ejecuten en forma normal, **la excepción** es que ocurra algún problema

Concepto

- Históricamente parte del código se dedicaba a contemplar situaciones de error

```
suma=19
cantidad=0
if cantidad!=0:
    promedio=suma/cantidad
else:
    print("No se puede dividir por cero")
```

Concepto

- Con el manejo de excepciones es posible independizar la lógica del programa del código de control de errores, es decir que evita tener que mezclarlos. Dicho de otra forma por un lado vamos a tener lo que el programa tiene que hacer y por otro lado el tratamiento de errores.
- De este modo las aplicaciones son más robustas y veloces

Concepto

- El control de errores tradicional (el que hemos realizado hasta ahora) es **preventivo. Evita que el error ocurra**
- El control de errores mediante excepciones es **correctivo**. Es decir intenta “solucionar” el problema después de haber ocurrido.

Tipos de Excepciones

- Cada error genera un tipo de **excepción distinto**, que puede ser **capturado o atrapado** por el programador.
 - División por cero
 - Uso de variables no inicializadas
 - Subíndices fuera de rango
 - Tratar de convertir a entero o a float un valor no numérico

Tipos de Excepciones

```
Traceback (most recent call last):  
  File "c:\Users\julia\OneDrive\Desktop\Organizador\UADE\Segu  
  ica Hecha en clase\ejemplo.py", line 1, in <module>  
    print(1/0)  
    ~~~  
ZeroDivisionError: division by zero
```

En este caso la excepción es **ZeroDivisionError: division by zero**

- El nombre de la excepción es ZeroDivisionError, con este nombre es con el que vamos a controlarlo.
- Luego viene la descripción de la excepción

Tipos de - Operaciones con diferentes tipos de datos

```
numero1= input("ingrese el primer numero")  
  
numero2= input("ingrese el primer numero")  
  
print(numero1/numero2)
```

```
Traceback (most recent call last):  
  File "c:\Users\julia\OneDrive\Desktop\Organizador\UADE\Segundo cuadrimestre Hecha\impresionNumero.py", line 5, in <module>  
    print(numero1/numero2)  
           ^~~~~~  
TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

En este caso la excepción es **TypeError, dado que no se pueden dividir dos string**

Tipos de - Error de indice

```
numeros=[1,2,3]

for i in range(0,7):
    print(numeros[i])
```

```
Traceback (most recent call last):
  File "c:\Users\julia\OneDrive\Desktop\Organizador\UADE\Segunda Hecha\impresionNumero.py", line 4, in <module>
    print(numeros[i])
               ~~~~~^~~
IndexError: list index out of range
```

En este caso la excepción es **IndexError**

Tipos de - Desbordamiento de variables numéricas

Las variables numéricas, tienen un rango posible de valores. Si se intenta almacenar un valor mayor, se desborda su capacidad.

```
valor=2.0
```

```
for i in range(100):
    print(i,valor)
    valor=valor**2
```

```
4 65536.0
5 4294967296.0
6 1.8446744073709552e+19
7 3.402823669209385e+38
8 1.157920892373162e+77
9 1.3407807929942597e+154
```

```
Traceback (most recent call last):
```

```
  File "c:\Users\julia\OneDrive\Desktop\Organizador\UADE\Segundo cuarto\Impresion de Numeros\impresionNumero.py", line 5, in <module>
    valor=valor**2
           ^~~~
```

```
OverflowError: (34, 'Result too large')
```

Control de errores

- Para capturar excepciones se utilizan los bloques **try/except**.
- Try significa intentar, es decir le vamos a decir que “intente” ejecutar el bloque de código.
- Luego colocaremos uno o varios except para manejar los diferentes errores
- El código ubicado entre try y except se denomina **bloque protegido**

Control de errores

```
try:  
    """Codigo que puede provocar errores"""  
except ZeroDivisionError:  
    """Codigo de tratamiento del error"""
```

- NUNCA PUEDE EXISTIR UN EXCEPT SIN UN TRY

Práctica en clase

Impedir que un programa falla debido a una división por cero



Resolución

```
try:  
    numero1= int(input("Ingrese un numero\n"))  
    numero2= int(input("Ingrese un numero\n"))  
  
    division = numero1/numero2  
  
    print("La division es igual a: ",division)  
  
except ZeroDivisionError:  
    print("No se puede dividir por cero")
```

Funcionamiento

1. Primero se ejecuta el **bloque protegido**, es decir el código ubicado entre el try y el except que delimita la zona en la que pueden llegar a ocurrir los errores
2. Si no ocurren errores el bloque except se saltea y la ejecución continua luego de este bloque except.

Funcionamiento

3. Si ocurre un error durante la ejecución del bloque protegido, el resto de este bloque se omite.
4. Si el tipo de error coincide con al excepción detallada en el except se ejecuta este bloque, que es donde se remediará el problema. Luego el programa continua normalmente.

Funcionamiento

- Si ocurre un error que no coincide con la excepción detallada en el `except`, esta se traslada a otros bloques **try/except** exteriores
- Si no se encuentra nada que la maneje será considerada como una excepción no manejada y el programa se detendrá con el mensaje de error correspondiente

Ejemplo

```
try:  
    try:  
        print("Intentando dividir por cero...")  
        x = 1 / 0  
    except ValueError:  
        print("Esto no se ejecuta porque no es un ValueError")  
  
    print("Este código no se ejecutará porque ocurrió una excepción no manejada en el bloque interno.")  
except ZeroDivisionError:  
    print("Excepción ZeroDivisionError manejada en el bloque exterior.")
```

Práctica en clase

Ahora vamos a agregar una excepción que impida que el programa falle debido a una división por datos inválidos.



Resolución

Primero vamos a investigar cuál es la excepción

```
Traceback (most recent call last):
  File "c:\Users\julia\OneDrive\Desktop\Organizador\UADE\Segundo cuatrimestre Hecha\divisionPorCero.py", line 2, in <module>
    numero1= int(input("Ingrese un numero\n"))
               ^
ValueError: invalid literal for int() with base 10: 'adfd'
```

Vemos que en este caso la excepción es **ValueError**

Resolución

```
try:  
    numero1= int(input("Ingrese un numero\n"))  
    numero2= int(input("Ingrese un numero\n"))  
  
    division = numero1/numero2  
  
    print("La division es igual a: ",division)  
  
except ZeroDivisionError:  
    print("No se puede dividir por cero")  
except ValueError:  
    print("Valor invalido")
```

Observaciones

- Pueden escribirse varios bloques **except** para manejar distintos tipos de excepciones, los que serán analizados en el orden en que se encuentran.
- Es importante destacar que puede haber varios except pero se ejecutara, a lo sumo uno de ellos.
- Cada tipo de excepción debe ser manejado por un único bloque except. No puede haber mas de uno para la misma excepción.

Except sin tipo de excepcion

- Un **except** sin tipo de excepción capturara **cualquier error** que pudiera producirse
- **No se recomienda hacerlo porque no permite diferenciar errores**
- De hacerlo, se debe escribir debajo de otros except, en el ultimo lugar

Except sin tipo de excepcion

```
try:  
    numero1= int(input("Ingrese un numero\n"))  
    numero2= int(input("Ingrese un numero\n"))  
  
    division = numero1/numero2  
  
    print("La division es igual a: ",division)  
  
except ZeroDivisionError:  
    print("No se puede dividir por cero")  
except ValueError:  
    print("Valor invalido")  
except:  
    print("Error desconocido")
```

Excepciones: as e

- Asignar una excepcion a una variable permite acceder al objeto de la excepcion.
- Se utiliza cuando:
 - Necesitas obtener más información sobre el error que ocurrió
 - Queremos mostrar o registrar el mensaje de error de la excepción

Excepciones: as e

- Ejemplo:

```
try:  
    """ Código que puede lanzar una excepción"""  
except Exception as e:  
    """ "e" es una referencia al objeto de la excepción"""
```

Excepciones: as e

- Ejemplo:

```
try:  
    numero1 = int(input("Ingresa el primer número: "))  
    numero2 = int(input("Ingresa el segundo número: "))  
    resultado = numero1 / numero2  
    print(f"El resultado es: {resultado}")  
except Exception as e:  
    |  
    |  
    print(f"Ocurrió un error: {e}")
```

Importante - Errores de sintaxis

- Los errores de sintaxis no pueden ser atrapados mediante excepciones, porque la sintaxis es verificada durante el análisis sintáctico del programa y no durante la ejecución
- Los errores de sintaxis ocurren cuando el código es leído y analizado por Python. Como este proceso ocurre antes de que el código se ejecute, no es posible capturar estos errores con un bloque try/except.
- En contraste, los errores que ocurren durante la ejecución del programa (como dividir por cero o errores de tipo) sí pueden ser manejados con try/except.

Importante - Errores de sintaxis

- Ejemplo:

```
dev saludo():
    print("Hola mundo")
```

```
dev saludo():
    ^
SyntaxError: invalid syntax
```

Importante - Errores de semántica

- Los errores semánticos son aquellos que no causan errores durante la fase de interpretación, pero provocan que el programa no funcione como se espera.
- Estos errores pueden deberse a problemas en la lógica del programa, algoritmos incorrectos, omisión de sentencias necesarias, o interpretación incorrecta de los requisitos del problema.
- Este código no generara ningún error, pero produce un resultado incorrecto.

Importante - Errores de semántica

- Ejemplo:

```
def areaTriangulo(base,altura):  
    area=(base/altura)*2  
    return area  
  
base=5  
altura=4  
  
resultado= areaTriangulo(base,altura)  
  
print(f"El area del triangulo es {resultado}")
```

Mismo tratamiento para mas de un tipo de error

- Si se le va a dar el mismo tratamiento a más de un tipo de error, puede usarse el mismo bloque **except**:

```
try:  
    numero = int(input("Ingresa un número: "))  
    division = 10 / numero  
    print(f"El resultado de la división es: {division}")  
except (ValueError, ZeroDivisionError):  
    print("Ocurrió un error")
```

- Los nombres de las excepciones deben escribirse **entre parentesis**

Práctica en clase

Utilizar manejo de excepciones para continuar normalmente la ejecución de una función luego de producido un error



Resolución - Función

```
def leerEntero():
    bandera=True

    while bandera:
        try:
            numero=int(input("Ingrese un numero\n"))
            bandera=False
        except ValueError:
            print("Dato invalido")
    return numero
```

Resolución - Programa Principal

```
try:  
    numero1= leerEntero()  
    numero2= leerEntero()  
  
    division= numero1/numero2  
except ZeroDivisionError:  
    print("No se puede dividir por cero")
```

Práctica en clase

Escribir un programa para imprimir números enteros a partir del 1 hasta un millón, que no pueda ser interrumpido con la combinación de Ctrl + C



Resolución

- Primero vamos a investigar cuál es la Excepción que ocurre al cortar la ejecución con Ctrl + C

```
Traceback (most recent call last):  
  File "c:\Users\julia\OneDrive\Desktop\Organizador\UADE\Segundo cu-  
actica Hecha\impresionNumero.py", line 2, in <module>  
    print(i)  
KeyboardInterrupt
```

- Luego capturamos la excepción KeyboardInterrupt

Resolución

```
while True:  
    try:  
        for i in range(1000000):  
            print(i)  
    except KeyboardInterrupt:  
        pass
```

Instrucción raise

- La sentencia **raise** permite al programador forzar a que ocurra una excepción específica. El único argumento a raise indica la excepción a generarse.

```
edad = int(input("Ingresa tu edad: "))

if edad < 0:
    raise ValueError("La edad no puede ser negativa.")

print(f"Tu edad es: {edad}")
```

```
Traceback (most recent call last):
  File "c:\Users\julia\OneDrive\Desktop\Organizador\UADE\Segundo
actica Hecha\impresionNumero.py", line 4, in <module>
    raise ValueError("La edad no puede ser negativa.")
                                         ^
ValueError: La edad no puede ser negativa.
```

Clausula else

- Esta clausula es opcional en un bloque **try/except**, y tiene como objetivo separar la zona que creemos susceptible de generar excepciones de la que esta libre de ellas
- Debe escribirse después de todos los except
- Solo será ejecutada cuando el bloque try precedente haya finalizado de forma normal, es decir sin haberse producido excepciones

Clausula else

- El bloque try contiene el código que puede generar una excepción.
- El bloque except captura y maneja las excepciones que se generen en el bloque try.
- El bloque else, si está presente, se ejecuta solo si no ocurre ninguna excepción dentro del bloque try. Si se lanza una excepción, el bloque else se salta y no se ejecuta.

Ejemplo

```
try:  
    numero = int(input("Ingresa un número: "))  
except ValueError:  
    print("Error: No ingresaste un número válido.")  
else:  
    print(f"Ingresa el número: {numero}")
```

Clausula finally

- La clausula finally es opcional, y se escribe luego de un bloque try o de un bloque except.
- Su propósito es garantizar una porción de código se ejecute **siempre**, sin importar si hubon errores o no

Clausula finally

- Se utiliza en tareas de limpieza, para liberar recursos previamente asignados y así evitar que los mismos se agoten, o para continuar en forma prolja.
- Por ejemplo cerrar conexión a base de datos, o conexión de red.

Clausula finally

- Recursos que se protegen con finally:
 - Memoria
 - Conexiones de red
 - Sesiones con bases de datos
 - Archivos temporales

Clausula finally - Sintaxis

```
try:  
    |     |     """Bloque protegido"""  
  
except """tipo excepcion""":  
    |     |     """codigo control de errores"""  
  
finally:  
    |     |     """Codigo de saneamiento"""
```

Clausula finally

- finally puede utilizarse solo con el bloque try, sin necesidad del except.
- Si se uso else: en el bloque try, finally va despues de este.
- Lo veremos mas en detalle cuando veamos archivos

Clausula finally

- Ejemplo:

```
numero=10
divisor=0

try:
    resultado=numero/divisor
except:
    resultado=None
    print("Se produjo un error durante la operacion")
finally:
    print("Operacion finalizada")
```

Resumen de la clase

- Excepciones. Concepto
- Tipo de excepciones
- Practica de ejercicios

Guía de excepciones

- Realizar la guía de excepciones

Muchas gracias!

Consultas?

