Clase 7 Excepciones

Excepciones. Concepto. Cláusulas try, except y finally. Instrucción raise.

The `try` block lets you test a block of code for errors.

The `except` block lets you handle the error.

The `else` block lets you execute code when there is no error.

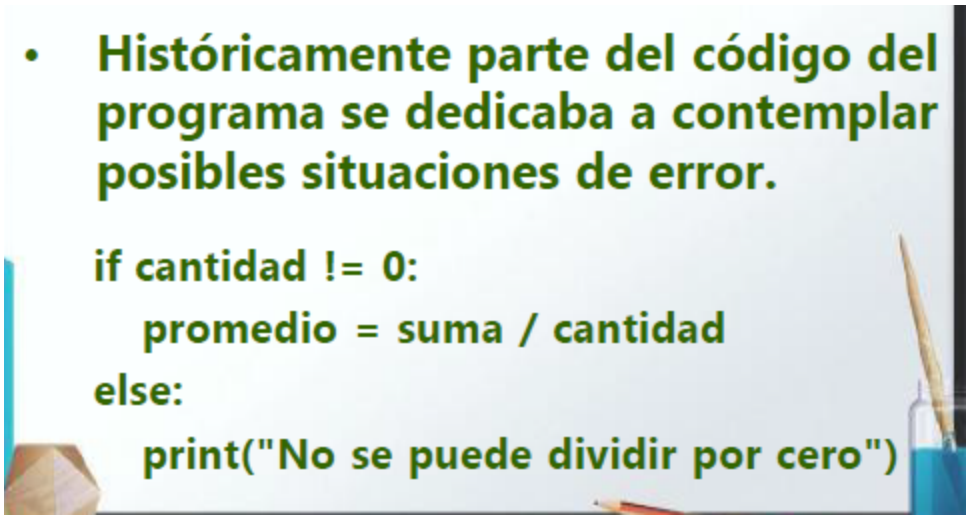
The `finally` block lets you execute code, regardless of the result of the try- and except blocks.

Teoria

Concepto

Excepcion es la indicacion de un problema ocurrido durante la ejecucion de un programa.

Dado que la “regla” es que los programas se ejecuten en forma normal, la excepcion es que ocurra algun problema.



Con el manejo de excepciones es posible independizar la logica del programa del codigo de control de errores , es decir que evita tener que mezclarlos.

De este modo las aplicaciones son robustas y veloces

El control de errores tradicional es **preventivo**: Evitamos que el error ocurra.

El control de errores mediante excepciones es correctivo : Intena solucionar el problema despues de haber ocurrido.

Tipos de excepciones

Cada error genera un tipo de excepción distinto , que puede ser capturado o atrapado por el programador :

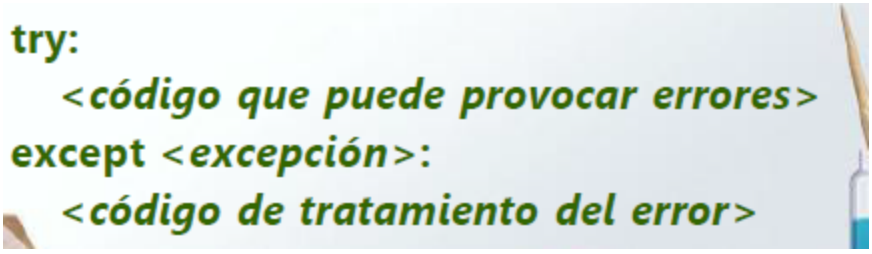
- Division por cero
- Uso de variables no inicializadas
- Subindices fuera de rango
- Tratar de convertir a entero o a float un valor no numérico.

```
print(1/0)

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

```
    File "<pyshell> " , line 1 , in <module>
ZeroDivisionError : division by zero

"En este caso la excepcion es ZeroDivisionError" divisón por
cero
```

Para capturar excepciones se utilizan los bloques **try / except** .

El código ubicado entre try y except se denomina **bloque protegido.**



EJEMPLO 1 - BASE

Impedir que un programa falle debido a una division por cero.

```
try :
    a = int(input("Dividendo ?"))
    b = int(input("Divisor?"))
    cociente = a//b
    resto = a % b
    print()
    print("Cociente : " , cociente)
    print("Resto : " , resto)
except ZeroDivisionError:
    print("No se puede dividir por cero.")
```

Funcionamiento

Primero se ejecuta el bloque protegido , es decir el código ubicado entre try y except que delimita la zona en la que pueden llegar a ocurrir los errores.

Si no ocurren errores el bloque except se saltea y la ejecución continua luego de este bloque except.

Si ocurre un error durante la ejecución del bloque protegido , el resto de este bloque se omite.

Si el tipo de error coincide con la excepción detalla en el except se ejecuta este bloque, que es donde se remediara el problema. Luego el programa continua normalmente.

Si ocurre un error que no coincide con la excepción detallada en el except, esta se traslada a otros bloque try/except exteriores.

Si no se encuentra nada que la maneje, sera considerada como una excepción no manejada y el programa se detendrá con el mensaje de error correspondiente.

Pueden escribirse varios bloque except para manejar distintos tipos de excepciones, los que serán analizados en el orden en que se encuentran.

Cada tipo de excepción debe ser manejado por un único bloque except. No puede haber mas de uno para la misma excepción.

Un except sin tipo de excepcion capturara cualquier error que pudiera producirse.

> No se recomienda hacerlo porque no permite diferenciar los errores.

Se admite escribirlo debajo de otros except, como medida de ultimo recurso.

EJEMPLO 2

Impedir que un programa falle debido a una division por cero o por el ingreso de datos invalidados. El programa finaliza con un mensaje de error.

```
try :
    a = int(input("Dividendo ?"))
    b = int(input("Divisor?"))
```

```
    cociente = a//b
    resto = a % b
    print()
    print("Cociente : " , cociente)
    print("Resto : " , resto)
except ZeroDivisionError:
    print("No se puede dividir por cero.")
except ValueError:
    print("Solo se permiten numeros enteros")
except:
    print("Error desconocido. Intentelo mas tarde")
```

> EL ÚLTIMO EXCEPT → debajo de todo

Observaciones

Errores de sintaxis no pueden ser atrapados mediante excepciones.

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

```
except (ValueError,ZeroDivisionError):
    print("Datos invalidos")
```

En este caso los nombres de las excepciones deben escribirse entre paréntesis.

EJEMPLO 3

OPORTUNIDAD DE CORRECCIÓN

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

```
def leerentero(msj="Ingrese un nro entero: "):
    while True:
        try:
```

```
            n = int(input(msj))
            break
        except  ValueError:
            print("Dato inválido.")
            print("Intente nuevamente")
    return n

#program principal
while True :
    try:
        a = leerentero("Dividiendo? ") #omito el msj y cheque
o que no sea una letra o nro real
        b = leerentero("Divisor? " )
        cociente = a//b
        resto = a %b
        break
    except ZeroDivisionError:
        print("no se puede dividir por cero")
        print("intente de vuelta")
print()
print("cociente: ", cociente)
print("resto: ", resto)
```

EJEMPLO 4

Escribir un programa para imprimir numero enteros a partir del 1, que no pueda ser interrumpido con Ctrl-C.

> CTRL + C frena la ejecución de cualquier código

```
cont = 1
while True:
    try:
        print(cont,end=" ")
```

```
        cont = cont + 1
    except KeyboardInterrupt:
        pass
#se puede detener cerrando consola de Python para detenerlo
#Presionando STOP en el idle
```

Instruction Raise

Se utiliza cuando se desea provocar una excepción

Puede ir seguida del nombre de la excepción a producir

Si no se detalla el tipo de excepción se relanza la ultima excepción producida.

Ejemplo 5

Darle al usuario la posibilidad de abortar la ejecución del programa en caso de producirse un error.

```
def leerentero(msj = "ingrese nro entero"):
    '''Funcion para ingresar un numero entero'''
    while True :
        try:
            n = int(input(msj))
            break
        except ValueError:
            print("dato inválido")
            a = input("desea ingresar otro nro? (S/N) " )
            if a.upper( ) == "N":
                raise
    return n
```

Clausula else:

Clausula opcional al usar try/except

Debe escribirse después de todos los except

Solo sera ejecutada cuando el bloque try precedente haya finalizado en forma normal, es decir sin haberse producido excepciones.

EJEMPLO 6

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

```
def leerentero(msj = "Ingrese un entero: "):
    "Funcion para ingresar un numero entero"
    while True:
        try:
            n = int(input(msj))
        except ValueError:
            print("Dato invalido.")
            print("Intente nuevamente.")
        else:
            break
    return n
```

Clausula finally

Es opcional y se escribe luego de un bloque try o de un bloque except.

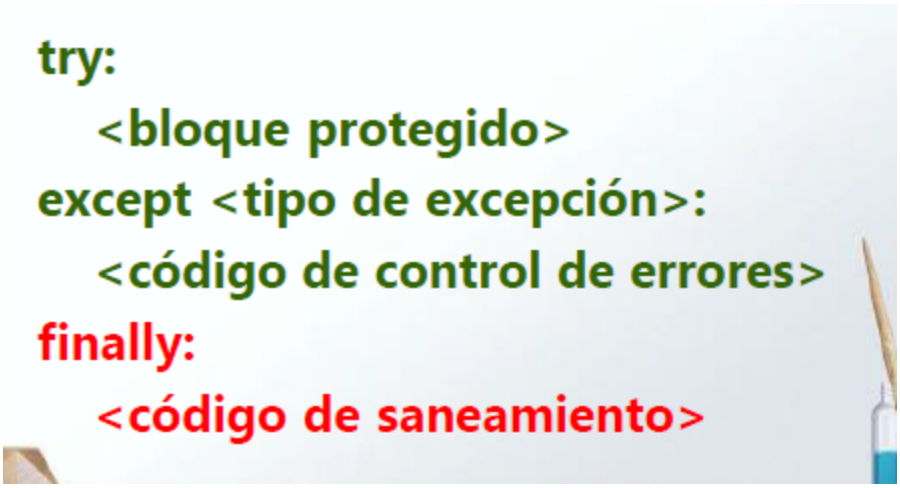
> contener un código que necesite haya error o no. limpieza, evitar que los recursos se agoten

Se utiliza en tareas de limpieza, para liberar recursos previamente asignados y asi evitar que los mismos se agoten, o para continuar en forma prolija.

Los recursos que se protegen con finally son:

- ~~Memoria~~

- ~~Conexiones de red~~
- ~~Sesiones con bases de datos~~
- Archivos temporales
- ~~Opciones de menu interfaz gráfica~~
- ~~Cursores del mouse interfaz gráfica~~



Finally puede utilizarse solo con el bloque try, sin necesidad del except.

Si se uso else: en el bloque try, finally va después de este.

EJEMPLO 7

Imprimir una matriz por pantalla, utilizando manejo de excepciones para darle el formato apropiado.

```
def imprimirmatriz(mat):
    filas = len(mat)
    columnas = len(mat[0])
    elementos = filas*columnas
    f = 0
    c = 0
```

```
    for i in range(elementos):
        try:
            print("%3d" %mat[f][c],end="")
        except IndexError:
            f = f + 1
            c = 0
            print()
            print("%3d" %mat[f][c],end="")
        finally:
            c = c + 1
```

Instruction assert -supere una prueba

La instrucción assert incorpora al programa un control expresado como una condición.

Este control es una afirmación que realiza el programador, y que el programa debe superar para poder continuar.

```
assert <condición>
```

Si la afirmación es cierta (condición verdadera), no ocurre nada.

> Si la condición es falsa, se produce una excepción AssertionError

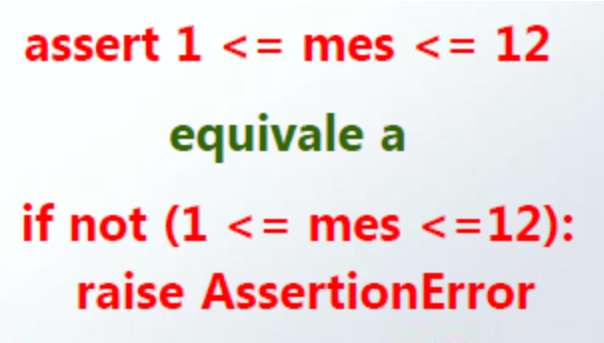
EJEMPLO 8

Verificar el rango de un numero de mes a traves de la instrucción assert.

```
while True:
    try:
        mes = int(input("Ingrese el mes: "))
        assert 1 <= mes <= 12
        break
    except ValueError:
```

```
        print("Solo se permiten numeros")
    except AssertionError:
        print("El mes debe estar entre 1 y 12.")
    print("Intente nuevamente")
print("El mes ingresado es:",mes)
```

Assert puede ser reemplazado por if y raise:



Si se incluyen varios assert en el mismo programa, puede usarse un mensaje para diferenciarlos:

```
palabra = input("Ingrese una palabra: ")
try:
    assert len(palabra)>5, "Palabra muy corta"
    assert palabra.isalpha(), "Solo se permiten letras"
except AssertionError as mensaje:
    print(mensaje)
```