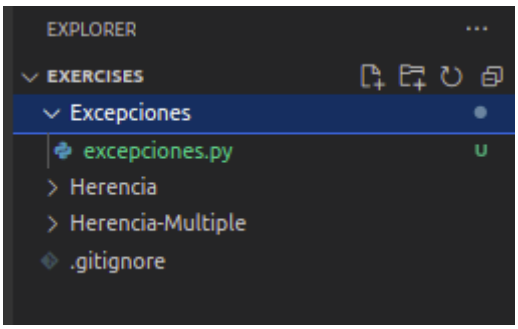
EXERCISES QUE TRABAJAREMOS EN EL CUE

- EXERCISE 1: GESTIÓN DE EXCEPCIONES
- EXERCISE 2: CAPTURA DE ERRORES Y MANEJOS DE MÚLTIPLES EXCEPCIONES
- EXERCISE 3: LANZAR EXCEPCIONES (RAISE)

EXERCISE 1: GESTIÓN DE EXCEPCIONES

Realizaremos un programa que sume 4 números enteros, y que muestre por pantalla el resultado.

Comenzaremos ingresando a *Visual Studio Code* (VSC), y procedemos a crear en el mismo una carpeta llamada Excepciones:



Creamos un archivo llamado **excepciones.py**, donde vamos a escribir el código para el cálculo de la suma:

Excepciones/exepciones.py

```
contador = 0
calculo_suma = 0

print("Sumatoria de 4 numero enteros")
while contador < 4:
    valor = int(input(f'Ingrese el numero entero {contador+1}: '))
    calculo_suma += valor
    contador +=1

print("El resultado de la suma es: ", calculo_suma)
```

Procedemos a ejecutar en el terminal el programa de cálculo ingresando solo números enteros:

Terminal

```
$ python excepciones.py
Sumatoria de 4 numero enteros
Ingrese el numero entero 1: 5
Ingrese el numero entero 2: 5
Ingrese el numero entero 3: 5
Ingrese el numero entero 4: 5
El resultado de la suma es:  20
```

Al ingresar un valor distinto a entero, tenemos el siguiente error en tiempo de ejecución:

Terminal

```
$ python excepciones.py
Sumatoria de 4 numero enteros
Ingrese el numero entero 1: 1
Ingrese el numero entero 2: "
Traceback (most recent call last):
  File "excepciones.py", line 6, in <module>
    valor = int(input(f'Ingrese el numero entero {contador+1}: '))
ValueError: invalid literal for int() with base 10: '"'
```

Obtenemos un tipo de error de ejecución ***ValueError*** ya que insertamos caracteres.

Para mejorar esto realizamos el manejo de excepciones ***try/except.***

Excepciones/exepciones.py

```
contador = 0
calculo_suma = 0

print("Sumatoria de 4 numero enteros")
while contador < 4:
    try:
        valor = int(input(f'Ingrese el numero entero {contador+1}:
'))
        calculo_suma += valor
        contador +=1
    except:
        print("Valor Errado: ingrese un numero entero correcto")

print("El resultado de la suma es: ", calculo_suma)
```

Salida:

Terminal

```
$ python excepciones.py
Sumatoria de 4 numero enteros
```

```
Ingrese el numero entero 1: 1
Ingrese el numero entero 2: 1
Ingrese el numero entero 3: s
Valor Errado: ingrese un numero entero correcto
Ingrese el numero entero 3: 3.5
Valor Errado: ingrese un numero entero correcto
Ingrese el numero entero 3: d
Valor Errado: ingrese un numero entero correcto
Ingrese el numero entero 3: f
Valor Errado: ingrese un numero entero correcto
Ingrese el numero entero 3: 2
Ingrese el numero entero 4: 5

El resultado de la suma es:  9
```

La declaración ***try … except*** tiene una cláusula ***else*** opcional, que, cuando está presente, debe seguir todas las cláusulas ***except***. Es útil para el código que debe ejecutarse si la cláusula ***try*** no lanza una excepción. Por ejemplo:

Excepciones/exepciones.py

```
contador = 0
calculo_suma = 0

print("Sumatoria de 4 numero enteros")
while contador < 4:
    try:
        valor = int(input(f'Ingrese el numero entero {contador+1}:
'))
        calculo_suma += valor
        contador +=1
    except:
        print("Valor Errado: ingrese un numero entero correcto")
    else:
        print("No se capturaron excepciones, todo bien")

print("\nEl resultado de la suma es: ", calculo_suma)
```

Salida:

Terminal

```
$ python excepciones.py
Sumatoria de 4 numero enteros
Ingrese el numero entero 1: 3
No se capturaron excepciones, todo bien
Ingrese el numero entero 2: 3
No se capturaron excepciones, todo bien
Ingrese el numero entero 3: 3
No se capturaron excepciones, todo bien
Ingrese el numero entero 4: 3
```

```
No se capturaron excepciones, todo bien

El resultado de la suma es:  12
```

La cláusula ***finally*** está presente, el bloque ***finally*** se ejecutará al final antes de que todo el bloque ***try*** se complete. La cláusula ***finally*** se ejecuta independientemente de que la cláusula ***try*** produzca o no una excepción.

Por ejemplo:

Excepciones/exepciones.py

```
contador = 0
calculo_suma = 0

print("Sumatoria de 4 numero enteros")
while contador < 4:
    try:
        valor = int(input(f'Ingrese el numero entero {contador+1}:
'))
        calculo_suma += valor
        contador +=1
    except:
        print("Valor Errado: ingrese un numero entero correcto")
    else:
        print("No se capturaron excepciones, todo bien")
    finally:
        print("Finally se ejecuta al final de la ejecución con o sin
excepciones")

print("\nEl resultado de la suma es: ", calculo_suma)
```

Salida:

Terminal

```

$ python excepciones.py
Sumatoria de 4 numero enteros
Ingrese el numero entero 1: 1
No se capturaron excepciones, todo bien
Finally se ejecuta al final de la ejecución con o sin excepciones
Ingrese el numero entero 2: 2
No se capturaron excepciones, todo bien
Finally se ejecuta al final de la ejecución con o sin excepciones
Ingrese el numero entero 3: 3
No se capturaron excepciones, todo bien
Finally se ejecuta al final de la ejecución con o sin excepciones
Ingrese el numero entero 4: r
Valor Errado: ingrese un numero entero correcto
Finally se ejecuta al final de la ejecución con o sin excepciones

```

```
Ingrese el numero entero 4: ^CValor Errado: ingrese un numero
entero correcto
Finally se ejecuta al final de la ejecución con o sin excepciones
Ingrese el numero entero 4: 4
No se capturaron excepciones, todo bien
Finally se ejecuta al final de la ejecución con o sin excepciones

El resultado de la suma es:  10
```

EXERCISE 2: CAPTURA DE ERRORES Y MANEJOS DE MÚLTIPLES EXCEPCIONES:

Vamos a manejar múltiples excepciones, y para ello crearemos una función que calcule una división de dos números enteros. Creamos el archivo **gestion-excepsiones.py.**

Excepciones/gestion-excepciones.py

```
def division_entera(x,y):
    dividendo = int(x)
    divisor = int(y)
    return dividendo/divisor

resultado = division_entera(1, 0)
print(resultado)
```

Obtenemos la siguiente salida:

Terminal

```
$ python test.py
Traceback (most recent call last):
  File "test.py", line 24, in <module>
    resultado = division_entera(1, 0)
  File "test.py", line 4, in division_entera
    return dividendo/divisor
ZeroDivisionError: division by zero
```

Procedemos a capturar el error de ejecución:

Excepciones/gestion-excepciones.py

```

def division_entera(x,y):
    try:
        dividendo = int(x)
        divisor = int(y)
        return dividendo/divisor
    except Exception as error:
```

```
        print('Se ha generado un erro no previsto',
type(error).__name__)



resultado = division_entera(1, 0)
print(resultado)
```

Obtenemos la siguiente salida:

Terminal

```
$ python test.py
Se ha generado un erro no previsto ZeroDivisionError
None
```

Agregamos la captura del error en la Excepción:

Excepciones/gestion-excepciones.py

```
def division_entera(x,y):
    try:
        dividendo = int(x)
        divisor = int(y)
        return dividendo/divisor
    except ZeroDivisionError:
        print("El divisor no puede ser cero")
    except Exception as error:
        print('Se ha generado un erro no previsto',
type(error).__name__)

resultado = division_entera(1, 0)
print(resultado)
```

Obtenemos la siguiente salida:

Terminal

```
$ python test.py
El divisor no puede ser cero
None
```

Ahora al pasar un carácter a la función dividir:

Excepciones/gestion-excepciones

```
def division_entera(x,y):
    try:
        dividendo = int(x)
        divisor = int(y)
        return dividendo/divisor
    except ZeroDivisionError:
```

```
 7         print("El divisor no puede ser cero")
 8     except Exception as error:
 9         print('Se ha generado un erro no previsto',
10 type(error).__name__)
11
12 resultado = division_entera(1, 'q')
13 print(resultado)
```

Obtenemos la siguiente salida:

Terminal

```
1 $ python test.py
2 Se ha generado un erro no previsto ValueError
3 None
```

Agregamos la excepción de **ValueError**:

Excepciones/gestion-excepciones

```
 1 def division_entera(x,y):
 2     try:
 3         dividendo = int(x)
 4         divisor = int(y)
 5         return dividendo/divisor
 6     except ZeroDivisionError:
 7         print("El divisor no puede ser cero")
 8     except ValueError:
 9         print("Deben ser numeros enteros")
10     except Exception as error:
11         print('Se ha generado un erro no previsto',
12 type(error).__name__)
13
14
15 resultado = division_entera('q', 0)
16 print(resultado)
```

Obtenemos la siguiente salida:

Terminal

```
1 $ python test.py
2 Deben ser numeros enteros
3 None
```

EXERCISE 3: LANZAR EXCEPCIONES (RAISE)

Excepciones/gestion-excepciones

```
def division_entera(x,y):
    if y == 0:
        raise ZeroDivisionError('No se puede dividir por cero')
    dividendo = int(x)
    divisor = int(y)
    return dividendo/divisor

resultado = division_entera(1, 0)
print(resultado)
```

Obtenemos la siguiente salida:

Terminal

```
$ python gestion-excepciones.py
Traceback (most recent call last):
  File "gestion-excepciones.py", line 10, in <module>
    resultado = division_entera(1, 0)
  File "gestion-excepciones.py", line 3, in division_entera
    raise ZeroDivisionError('No se puede dividir por cero')
ZeroDivisionError: No se puede dividir por cero
```