

Manejo de excepciones

Sintaxis

try:

acción a realizar si NO hay errores dentro del try

except:

acción a realizar si SI hay errores dentro del try

else:

acción a realizar SI NO hay ninguno except (se ejecuta después del finally, si es que hay)

finally:

acción a realizar luego que se ejecute el try o el except, es decir, se ejecuta independientemente de si hubo errores o no

Creación de errores

Cuando conozco los posibles errores que pueda tener mi programa, puede crear mis propios errores y/o ejecutar un except unicamente para un error en específico, aunque este except se puede ejecutar una o varias veces (cada una de ellas con un error diferente), dejando por último a un except general en caso de que hubiera un error que no haya contemplado.

except ZeroDivisionError:

acción a realizar cuando en el try se quiera dividir algo por cero

except TypeError:

acción a realizar cuando en el try se quiera mezclar (y no se pueda) distintos tipos de datos

except (ZeroDivisionError, TypeError):

accion a realizar

except SyntaxError:

acción a realizar cuando en el try haya un error de sintaxis

```
except NameError;
```

acción a realizar cuando en el try se haya usado una variable sin definirla previamente

Raise

Siguiendo con la logica de los except para crear errores, vos también podes crear tus propios mensajes de error para cuando se ejecute un except especial y también forzar al programa a ejecutar errores,

Ejemplo:

```
try:
```

```
    resultado = a + b
```

```
    if resultado < 0:
```

```
        raise ValueError("no se puede resul negativo")
```

```
except ValueError as err
```

```
    print(err)
```

```
try:
```

```
    acción a realizar
```

```
    if condición:
```

```
        raise ZeroDivisionError("Se intento dividir por cero")
```

```
except ZeroDivisionError:
```

```
    acción a realizar
```

```
except:
```

```
    acción a realizar
```

Entonces lo que esta pasando en este ejemplo es que si se cumple la condición en el try se va a ejecutar el error ZeroDivisionError independientemente de si era un error o no, y si no pusiste ningún except se te frenará el programa, pero para eso generará un excepto personalizado para que no pase.

Para devolver el mensaje que le pase a ese error en el raise, puedo hacer lo siguiente:

```
except ZeroDivisionError as variable_error:
```

```
    print(variable_error)
```

Manipulación de archivos

Abrir archivos

Existen dos formas:

```
- archivo = open(path_archivo, modo)
```

archivo.close() => Ya que si lo abro de esta manera hay que cerrarlo manualmente.

Existe un método para preguntarle al programa si el archivo está cerrado o no.

```
- archivo.closed()
```

```
- with open(path_archivo, modo) as variable_para_llamar_archivo:
```

```
    accion a realizar
```

De esta última manera, el archivo se cerrará automáticamente.

path_archivo => es un objeto de tipo str que indica la ruta en la que se encuentra el archivo.

modo => es un objeto de tipo str que indica la forma en la que Python accederá al archivo en cuestión.

- r: abre un archivo solo para la lectura
- r+: abre un archivo solo leer y escribir
- a: abre un archivo para escribir (si ya existe texto dentro del escribo, simplemente se agregará debajo) y si este no existe creará uno con el contenido que le pasemos.
- w: abre un archivo y sobrescribe el contenido que ya exista dentro del mismo, pero si no existe el archivo simplemente creará uno con ese contenido que le pasemos.

Una vez que instanciamos esto, por debajo para podremos acceder a los métodos que queramos (dentro del que hayamos definido como modo).

```
-----
```

```
archivo = open("/examen/examen.txt", "r")
```

```
print(archivo.read())
```

```
print(archivo.readlines())
```

```
archivo.close()
```

```
archivo = open("/examen/examen.txt", "a")
```

```
archivo.write("texto")
```

```
archivo.close()
```

- `archivo.read()`
- `archivo.readline()`
- `archivo.readlines()`
- `archivo.readable()`
- `archivo.write()`
- `archivo.writelines()`
- `archivo.writable()`

Rutas absolutas y relativas

Absoluta: el path o ruta a un archivo, será entonces, el recorrido de directorios o carpetas que debemos recorrer para llegar a nuestro archivo. Esta se escribe separando los nombres de los respectivos directorios separados por "/".

```
"C:\home\Facultad\Fundamentos\Manipulación_de_archivos.md"
```

Relativa: imaginemos que esta es la estructura de archivos de nuestra computadora, donde existen dos carpetas en el home: *Fotos* y *Facultad*. Dentro de la carpeta *Facultad*, podemos encontrar a su vez dos directorios: *Fundamentos* y *Estadística*. Nuestra carpeta de trabajo actual es la de *Fundamentos*, donde tenemos nuestro archivo de interés *Manipulación_de_archivos.md*.

Desde el directorio *Facultad* podemos escribir la ruta relativa a nuestro archivo del siguiente modo:

```
../Fundamentos/Manipulación_de_archivos.md
```

Ahora si quisieramos acceder a las *Fotos*, podemos hacer:

```
cd ../Fotos
```

Modulo os y sus métodos

“os” es un modulo de Python proporciona funciones para interactuar con el sistema operativo.

¿Cómo se usa?

import os: para importar el modulo a nuestro programa

`os.getcwd()`: para conocer el directorio de trabajo en el que estamos

Si quiero pasar mi path para abrir un archivo puedo hacerlo de dos formas:

- `os.getcwd() + "\\archivo.txt"`
- `os.path.join(os.getcwd(), "archivo.txt")`

`os.chdir(path_nuevo)`: para cambiarnos a otro directorio que le pasemos por parametro

`os.mkdir(nombre_carpeta)`: para crear una nueva carpeta

`os.listdir()`: para listar los directorios que existen dentro del nuestro

`os.path.exists(path_buscado)`: pregunta si existe o no ese path

`os.remove(path_buscado)`: elimina el path buscado

Ejercicios

1)

```
def no_empiezan_con_string(string):  
    import os  
  
    archivo = open(os.getcwd() + "\\ejercicio_1.txt", "r")  
  
    lineas = archivo.readlines()  
  
    sum = 0  
  
    for linea in lineas:  
        if not linea.startswith(string):  
            sum += 1  
  
    archivo.close()  
  
    return print(sum)
```

```
no_empiezan_con_string("p")
```

2)

```
def imprimir_lineas(n_lineas):  
    import os  
  
    archivo = open(os.getcwd() + "\\ejercicio_1.txt", "r")  
  
    lineas = archivo.readlines()  
  
    for i in range(0, n_lineas):  
        print(lineas[i])  
  
    archivo.close()  
  
imprimir_lineas(10)
```

3)

```
def imprimir_lineas(n_lineas):  
    import os  
  
    archivo = open(os.getcwd() + "\\ejercicio_1.txt", "r")  
  
    lineas = archivo.readlines()  
  
    cant_lineas = len(lineas) - n_lineas  
  
    print(lineas[cant_lineas:])  
  
    archivo.close()  
  
imprimir_lineas(10)
```

4)

```
def imprimir_cant_lineas():  
    import os
```

```
archivo = open(os.getcwd() + "\\ejercicio_1.txt", "r")

palabras = archivo.read()

cant = palabras.split()

print(cant)


archivo.close()


imprimir_cant_lineas()
```

5)

```
def reemplazar_letra(string):

    import os

    archivo = open(os.getcwd() + "\\ejercicio_1.txt", "r")

    palabras = archivo.read()

    texto_nuevo = palabras.replace(string, string + "\n")

    archivo.close()

    nuevo_archivo = open(os.getcwd() + "\\ejercicio_5.txt", "w")

    nuevo_archivo.write(texto_nuevo)

    nuevo_archivo.close()

reemplazar_letra("a")
```

6)

```
def eliminar_saltos():

    import os

    archivo = open(os.getcwd() + "\\ejercicio_1.txt", "r")
```

```
palabras = archivo.read()

texto_nuevo = palabras.replace("\n", "")

archivo.close()

nuevo_archivo = open(os.getcwd() + "\\ejercicio_6.txt", "w")

nuevo_archivo.write(texto_nuevo)

nuevo_archivo.close()

eliminar_saltos()
```

7)

```
def palabra_mas_larga():

    import os

    archivo = open(os.getcwd() + "\\ejercicio_1.txt", "r")

    texto = archivo.read()

    cada_palabra = texto.split()

    mas_larga = max(cada_palabra, key=len)

    print(mas_larga, len(mas_larga))

palabra_mas_larga()
```

8)

```
def guardar_en_uno():

    import os

    arch1 = open(os.getcwd() + "\\ejercicio_1.txt", "r")

    arch1_texto = arch1.read()

    arch1.close()

    arch2 = open(os.getcwd() + "\\ejercicio_5.txt", "r")
```



```

arch2_texto = arch2.read()

arch2.close()

arch3 = open(os.getcwd() + "\\ejercicio_6.txt", "w")

arch3.writelines([arch1_texto, arch2_texto])

arch3.close()

guardar_en_uno()

```

9)

```

def _frecuencia():

    frecuencia = {}

    with open("ejercicio_1.txt", "r") as texto:

        palabras = texto.read().split()

        for palabra in palabras:

            if palabra in frecuencia:

                frecuencia[palabra] += 1

            else:

                frecuencia[palabra] = 1

        for palabra in frecuencia.keys():

            frecuencia[palabra] = round(frecuencia[palabra] /
len(palabras), 3)

    return print(frecuencia)

_frecuencia()

```

10)

```
import os, glob

def unir_txt():

    lista_txt = glob.glob("*.txt")

    salida = open(os.getcwd() + "\\texto_completo.txt", "a")

    for txt in lista_txt:

        archivo = open(txt, "r")

        salida.write(archivo.read())

        archivo.close()

    salida.close()

unir_txt()
```

Expresiones regulares

Secuencias de escape

- /n => salto de línea
- /t => tab
- ' => comillas simples
- " => comillas dobles
- "'' => comillas triples

Metacaracteres

- \d => dígitos del 0 al 9
- \D => NO dígitos del 0 al 9
- \w => carácter (a-z, A-Z, 0-9, _)
- \W => NO carácter (a-z, A-Z, 0-9, _)
- \s => espacio en blanco
- \S => NO espacio en blanco
- . => cualquier carácter excepto nueva línea
- \ => cancela caracteres especiales
 - \. => para indicar el punto
- ^caracteres => inicio de cadena de caracteres

- caracteres\$ => fin de cadena de caracteres
- \A => inicio de texto
- \Z => fin de texto
- caracteres* => 0 o más (codiciosa)

Ejemplo: Python!* => encuentra un match cuando dentro de mi cadena haya los caracteres “Python” seguido de 0 o más “!”. Si encuentra Python, hay match= Python, si encuentra Python!, hay match= Python!, y así... No hay match si no encuentra “Python”.

- caracteres+ => 1 o más (codiciosa)
- caracteres? => 0 o 1 (perezosa)
- carácter{número} => número exacto de veces que se repite un carácter
- carácter{número,} => número de veces que se repite a partir de ese numero que le pasemos en adelante
- caracter{num1, num2} => rango de numeros de veces que se repite un caracter
- (caracteres) => grupos
- [caracteres] => encuentra los caracteres entre corchetes

Si pongo [98]\d{2} me tomara los primeros dos digitos numericos que arranquen con 9 o con 8. [n1, nu2] funciona como si tuviera un “[”.

Si pongo [a-z] me tomara todos los valores de la a a la z en minuscula.

- [^caracteres] => encuentra caracteres que NO estan dentro de los corchetes
- .+ => toma todo lo que hay dentro de los guiones
- | => condicion, o uno o otro
- \b => limites de palabra
- \B => NO limites de palabra
- ()\1 => para encontrar un grupo (123-) mas de una vez

EJ: En vez de escribir (123-)(123-), escribo (123-)\1 .

Métodos y ejemplos

import re => para importar el modulo de expresiones regulares

re.findall(r'patron_a_encontrar', cadena_de_texto) => buscar todos los emparejamientos. Devuelve un array con todos los matches.

re.search(r'patron_a_encontrar', cadena_de_texto) => buscar el primer emparejamiento. Devuelve objeto especificando el match.

re.search(r'patron_a_encontrar', cadena_de_texto).group() =>

`re.match()` => busca el patrón y devuelve la primera aparición y solo al principio de la cadena. Si se encuentra una coincidencia en la primera línea, devuelve el objeto de coincidencia. Pero, si se encuentra una coincidencia en alguna otra línea, devuelve un valor nulo.

`re.sub(r'patron_a_encontrar', nueva_cadena, cadena_de_texto)` => permite reemplazar todos las ocurrencias del patrón por otro patrón en un string.

Flags

Python toma una cadena de caracteres como una sola línea de código, independientemente de si lo hemos escrito separado (salvo que hayamos indicado mediante `"/n"`). Por lo tanto, si deseas que Python interprete tu cadena separado entre líneas debes incluir un "flag" dentro de tu método.

Ejemplo:

```
texto = """Hola mundo.
```

```
Me gusta Python!!!
```

```
Mi primer numero de la suerte es 987-654-321
```

```
Mi segundo numero de la suerte es 876-543-210
```

```
"""
```

```
re.search(r'Mundo.$', texto, flags=re.M)
```

`flags=re.M` => hace que el texto sea multilínea

`flags=re.I` => ignora si el carácter está en minúscula o mayúscula

POO

poo → muchos objetos que se comunican entre sí.

- Un mensaje es un método de un objeto. Los mensajes se usan haciendo `.mensaje()`
- Los objetos tienen estado, cambia estado → comportamiento de los objetos
- Clases nos da una idea de que puede hacer el objeto de esa clase. ej) pepito es un objeto y golondrina la clase

- Interfaz de un objeto, es el conjunto de mensajes que entiende. self = objeto dado Los métodos tienen siempre como primer argumento el "self". __init__ = para construir, es el constructor.

Clases de objetos Para crear una clase se inicia con la palabra class y por consiguiente se escribe el nombre de la clase en mayúscula. Los métodos son los mensajes que puede hacer un objeto. La diferencia entre un método y una función, es que la última está por fuera de la clase. En el método, siempre tienen como primer argumento el self, quien representa un objeto dado

El __init__ es el constructor de pepita por ejemplo. Veámos un ejemplo:

```
class Golondrina:
```

```
    def __init__(self, energia):
```

```
        self.energia = energia
```

```
    def comer_alpiste(self, gramos):
```

```
        self.energia += 4 * gramos
```

```
    def volar_en_circulos(self):
```

```
        self.volar(0) def volar(self, kms):
```

```
            self.energia -= 10 + kms
```

```
pepita = Golondrina(100) # el 100 es el estado inicial
```

__init__ viene de la palabra en inglés initialize que en castellano es inicializar. Es lo que se conoce como el constructor de una clase y nos permite darle valores iniciales a los atributos de sus instancias a la hora de crearlas.

Por su parte, self(que en castellano sería algo así como yo) es un primer parámetro obligatorio que nos permite acceder a los atributos del objeto que estamos instanciando. Si bien ese parámetro no debe llamarse self obligatoriamente, es la convención que se utiliza para nombrarlo y la respetaremos a lo largo de todo el recorrido

Git

Paso 1: Copiar el link del repositorio

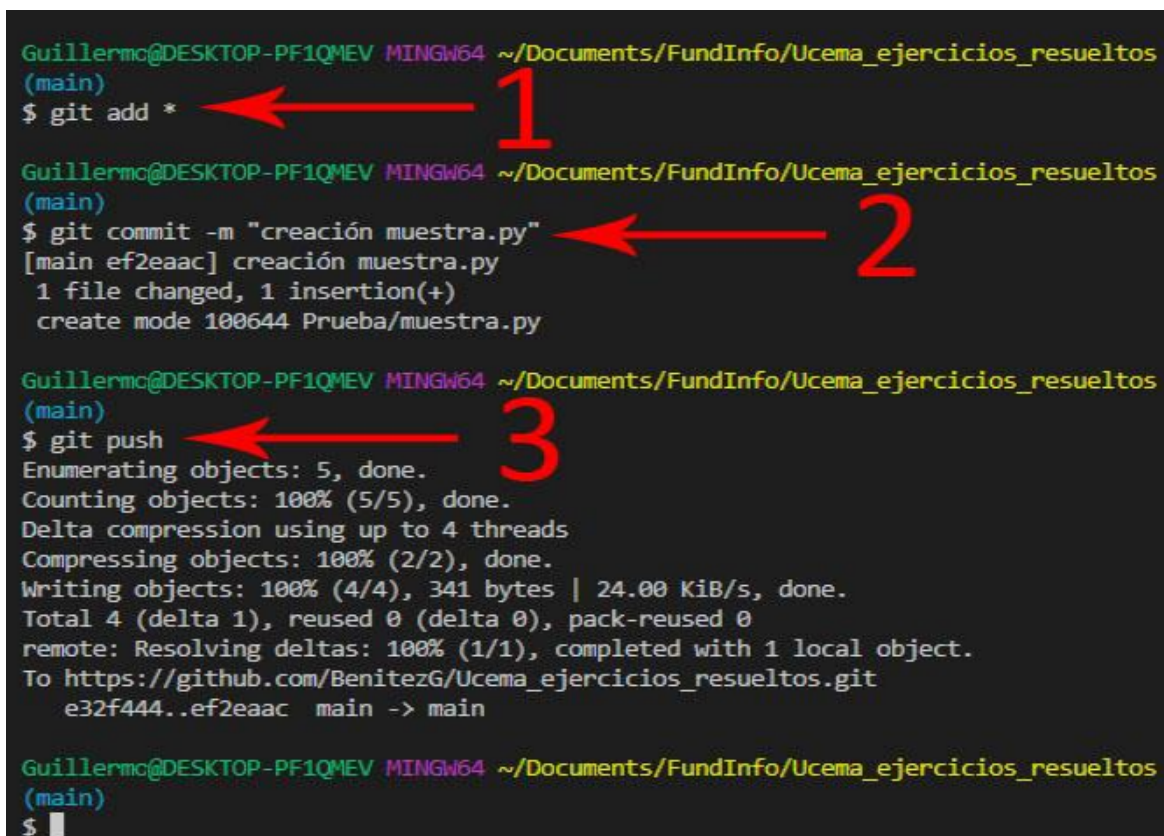
Paso 2: Abrir el Visual Studio Code y en este abrir una carpeta en la cual vamos a clonar nuestro repositorio

Paso 3: Abrir una nueva terminal

Paso 4: Elegir git bash y clonar el repositorio mediante el comando git clone y pegando el link que copiaron en el paso 1 y dándole a Enter. En mi caso sería: git clone <https://github.com/BenitezG/Repositorio.git>

Paso 5: Movernos al repositorio usando cd: cd nombre_repositorio

Paso 6: Por último, una vez en el repositorio (carpeta) a medida que vayamos generando nuevos archivos y carpetas (con las resoluciones de los ejercicios) vamos a tener que ir actualizando el repositorio. Para esto vamos a tener que seguir una serie de tres pasos:



```
Guillermo@DESKTOP-PF1QMEV MINGW64 ~/Documents/FundInfo/Ucema_ejercicios_resueltos
(main)
$ git add * ← 1

Guillermo@DESKTOP-PF1QMEV MINGW64 ~/Documents/FundInfo/Ucema_ejercicios_resueltos
(main)
$ git commit -m "creación muestra.py" ← 2
[main ef2eaac] creación muestra.py
1 file changed, 1 insertion(+)
create mode 100644 Prueba/muestra.py

Guillermo@DESKTOP-PF1QMEV MINGW64 ~/Documents/FundInfo/Ucema_ejercicios_resueltos
(main)
$ git push ← 3
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 341 bytes | 24.00 KiB/s, done.
Total 4 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/BenitezG/Ucema_ejercicios_resueltos.git
e32f444..ef2eaac main -> main

Guillermo@DESKTOP-PF1QMEV MINGW64 ~/Documents/FundInfo/Ucema_ejercicios_resueltos
(main)
$
```