

Clase 5: Manejo de Excepciones y Debugging

Módulo 2: Programación Avanzada en Python.

Apoyado por:

CORFO

Clase de hoy

01

Excepciones

Definir y lanzar excepciones,

02

Manejo de excepciones

Impresión de resultados.

03

Debugging

Gestión de errores plasmados en el código.



4A

Excepciones y Manejo de Excepciones

Bloque A

Qué veremos en Bloque A

- Definición de Excepciones.
- Manejar Excepciones.

Excepciones

Excepciones

- En programación reconocemos dos tipos de errores típicamente: “errores de sintaxis” y “excepciones”.
- Los “errores de sintaxis” son equivocaciones del programador que quedan plasmados en el código.
- Por ejemplo, un “error de sintaxis” es obviar la indentación, o los dos puntos, al formar una función.
- Sin embargo, incluso si el código está sintácticamente correcto, en ocasiones los programadores cometen errores que son detectados como resultados erróneos o fallos en la ejecución.

Excepciones

Excepciones

- A estos últimos tipos de errores les llamaremos “excepciones”.
- La idea central del manejo de excepciones se basa en tratar de capturar gran parte de estas y manejarlas adecuadamente.
- Por manejar adecuadamente nos referimos a que, si una excepción ocurre, el sistema debe seguir su funcionamiento.
- La cantidad potencial de excepciones crece con la complejidad del código y con las líneas de código.
- Por lo tanto, capturar todas las excepciones es un asunto imposible de alcanzar.

Excepciones

Excepciones en Python

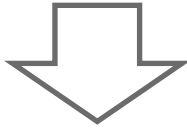
- Las excepciones en Python son “tipos”, es decir, son objetos.
- Veamos algunos tipos de excepciones:
 - **ZeroDivisionError**: emerge cuando el segundo argumento de una división u operación módulo es cero.
 - **TypeError**: emerge cuando una operación o función es aplicada a un objeto de tipo no apropiado.
- La palabra “emerge” es importante: las excepciones “aparecen” cuando la situación que refleja el error es encontrada por la ejecución del sistema.
- Lo importante será que cuando la excepción aparezca, nuestro código sea capaz de capturarla y reaccionar de manera adecuada.

Excepciones

Excepciones en Python

- Ejemplo:

```
print("Hola Mundo"[len("Hola Mundo")])
```



Excepción "IndexError"

```
Traceback (most recent call last):  
  File "directory/file_test.py", line 1, in <module>  
    print("Hola Mundo"[len("Hola Mundo")])  
IndexError: string index out of range
```


Excepciones

Capturando Excepciones

- En el ejemplo anterior, la excepción no está debidamente capturada.
- Cuando la excepción “IndexError” emerge, el programa se detiene completamente.
- Para evitar esto, debemos capturar la excepción y decidir qué hacer con esta.
- Cuando sepamos que hay alguna operación que eventualmente puede terminar en un resultado erróneo, usaremos los bloques:

try:

código que pueda levantar una excepción

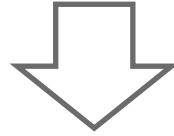
except NombreExcepción:

código que se ejecuta si la excepción emerge

Excepciones

Capturando Excepciones

```
try:  
    print("Hola Mundo"[len("Hola Mundo")])  
except IndexError:  
    print(";;El índice está fuera del string!!")
```



;;El índice está fuera del string!!

Process finished with exit code 0

El código ha terminado exitosamente.

Orientación a Objetos

Capturando más excepciones

- No estamos limitados a capturar una sola excepción:

```
try:
    print("Hola Mundo"[len("Hola Mundo")])
except IndexError:
    print("¡¡El índice está fuera del string!!")
except TypeError:
    print("¡¡El índice debe ser un entero!!")
```

```
try:
    print("Hola Mundo"["a"])
except IndexError:
    print("¡¡El índice está fuera del string!!")
except TypeError:
    print("¡¡El índice debe ser un entero!!")
```

Orientación a Objetos

Capturando más excepciones

- También podemos presentar una respuesta predeterminada en cada caso.

```
try:
    print("Hola Mundo"[len("Hola Mundo")])
except IndexError:
    print("¡¡El índice está fuera del string!!")
except TypeError:
    print("¡¡El índice debe ser un entero!!")
finally:
    print("IndexError y TypeError chequeadas.")
```

```
try:
    print("Hola Mundo"["a"])
except IndexError:
    print("¡¡El índice está fuera del string!!")
except TypeError:
    print("¡¡El índice debe ser un entero!!")
finally:
    print("IndexError y TypeError chequeadas.")
```



Trabajo grupal – Bloque A

Paralelo 2

G1	G2	G3	G4
Nicolas Mardones	Víctor Meza Herrera	Daniela Méndez Gándara	Claudia Blanco
Manuel Denis	Estefania Manriquez	Álvaro Pérez	Ariel Inostroza
Bryan Castillo	Patricio Vera	Pedro Nahum	Héctor Aguayo
GERALDY SUAREZ	Oscar Torres	Javier Gajardo	Ruben Sanhueza Ramirez
Scarlett Espinoza	Braulio Quiroz	Angela Proboste Neira	Félix González
Ulises Campodónico	Yerko Gallardo	Nicolás Guzmán	Ariel Mora
Carol Leiva	Rodrigo Araya		
G5	G6	G7	G8
Fabian Díaz	Camila Oyarzún	Mayerlyn Rodriguez	Daniela Porto
Natalia Rivera	Stefanya Pulgar	Sebastian Vega	Cristian Chavez Jara
Juan Salinas	Carlos Emilio Azócar Riquelme	Efrain Duarte Campos	Juan Rodrigo Vega
Rodrigo Pastén Cortés	Nicolas Rojas	Bianel Bianchini	Rodolfo Cantillana
Flavio Jara R.	luis.paillan.cnc@gmail.com	Bastián Gamboa Labbé	Abraham Ruiz
Daniel García	Cristóbal Gajardo	Pablo Uribe	Rodrigo Álvarez

Trabajo grupal – Ejercicio #1

Entendiendo las excepciones

- Investigue posibles excepciones aplicables a las listas y diccionarios.
- Escriba un código que obligue la emergencia de una excepción y captúrela con los bloques try/except. Imprima por pantalla el nombre de la excepción y el mensaje.
- **Tip:** para identificar las excepciones que aplican a determinados tipos de datos usted siempre debe recurrir a la documentación oficial.

Trabajo grupal – Ejercicio #2

Mejorando el código con manejo de excepciones

- Utilice el código creado en la clase anterior (el problema de la distancia geodésica) y mejórelo con manejo de excepciones.
- Investigue los valores posibles de las latitudes y longitudes.
- Incorpore excepciones de tipo `ValueError` a ser lanzadas cuando se intente crear un objeto con valores de latitude y/o longitude erróneas.
- Pruebe su código forzando la aparición de estas excepciones y capturándolas para mostrar el resultado erróneo.

A photograph of three women in a professional setting, overlaid with a magenta tint. The woman on the left has dark hair and is looking down. The woman in the center has short blonde hair and is smiling while looking at a laptop. The woman on the right has dark hair and is also smiling. They are sitting at a desk with a laptop and some papers. A large blue rectangle with the word 'Break!' in white text is positioned in the lower-left foreground.

Break!



4B

Representación de Clases

Bloque B

Qué veremos en Bloque B

- Obtener detalles de excepciones.
- Lanzar excepciones.
- Realizar debugging con Pycharm y similares.

Excepciones

Detalles de excepciones

- Un aspecto interesante de las excepciones es que son modeladas como objetos.
- Por lo tanto, esperamos que estos objetos contengan algo de información para ayudar en el debugging o corrección de defectos.
- Para obtener los detalles debemos definir un alias para la excepción que va a capturar los detalles del objeto:

```
except Exception as error
```

Excepciones

Detalles de excepciones (ejemplo)

```
try:
    print("Hola Mundo"["a"])
except IndexError as error:
    print("¡¡El índice está fuera del string!!")
except TypeError as error:
    print(error) # string indices must be integers
finally:
    print("IndexError y TypeError chequeadas.")
```

Excepciones

Detalles de excepciones

- Podemos también imprimir el nombre de la excepción:

```
try:
    print("Hola Mundo"["a"])
except IndexError as error:
    print("¡¡El índice está fuera del string!!")
except TypeError as error:
    print("Una excepción ha ocurrido:", type(error).__name__, " -> ", error)
    # Una excepción ha ocurrido: TypeError -> string indices must be integers
finally:
    print("IndexError y TypeError chequeadas.")
```

Excepciones

Lanzando excepciones

- En ocasiones no basta con esperar pasivamente una excepción.
- Por ejemplo, cuando instanciamos un objeto, podríamos lanzar una excepción si es que alguna de las condiciones para la inicialización del objeto no se cumple.
- Para esto usaremos decisiones (if) y en caso de que se cumpla una condición no válida, usaremos el siguiente código para lanzar una excepción (ejemplo con ValueError):

```
if valor >= 100:  
    raise ValueError("Mensaje de error")
```

Debugging

¿Qué es debugging?

- El debugging es el nombre con el que se conoce a la actividad del desarrollo que se encarga de remover defectos.
- Los defectos son imperfecciones que existen en un producto de software y sus elementos relacionados.
- A diferencia de las pruebas de software, el debugging se centra en la remoción de los defectos, más que en la identificación de estos.
- Para realizar debugging los entornos de desarrollo nos ofrecen herramientas que permiten identificar los valores de las variables en un momento dado, suspender la ejecución de un programa para verificar el estado de este, entre otras.

Debugging

Debugging en un IDE

- Lo primero que debemos hacer en un IDE para inicializar el debugging es marcar las líneas donde deseamos que se realice una suspensión temporal de la ejecución del código para poder analizar el estado del programa.
- El estado del programa se entenderá como los valores que tienen, en un momento determinado, las diversas variables en juego.
- Para activar estas suspensiones necesitamos:
 - Indicar “**breakpoints**” (una acción denominada “toggle breakpoint”), es decir, los puntos en los que deseamos que se realice la suspensión temporal.
 - Iniciar la ejecución del programa en modo “debugging”.

Debugging

Toggle breakpoints

Breakpoint

```
1  class Video:
2      def __init__(self, name, framerate, frames_count):
3          self.name = name
4          self.framerate = framerate
5          self.frames_count = frames_count
6
7      2 usages
8      def time_length(self):
9          return self.frames_count/self.framerate
10
11      1 usage
12      def time_length_minutes(self):
13          return self.time_length()/60.0
14
15  videos = [Video("DH BikePark", 60, 50000), Video("Subaru Offroad", 60, 120000)]
16
17  for video in videos:
18      print(video.name, video.time_length_minutes(), video.time_length())
```

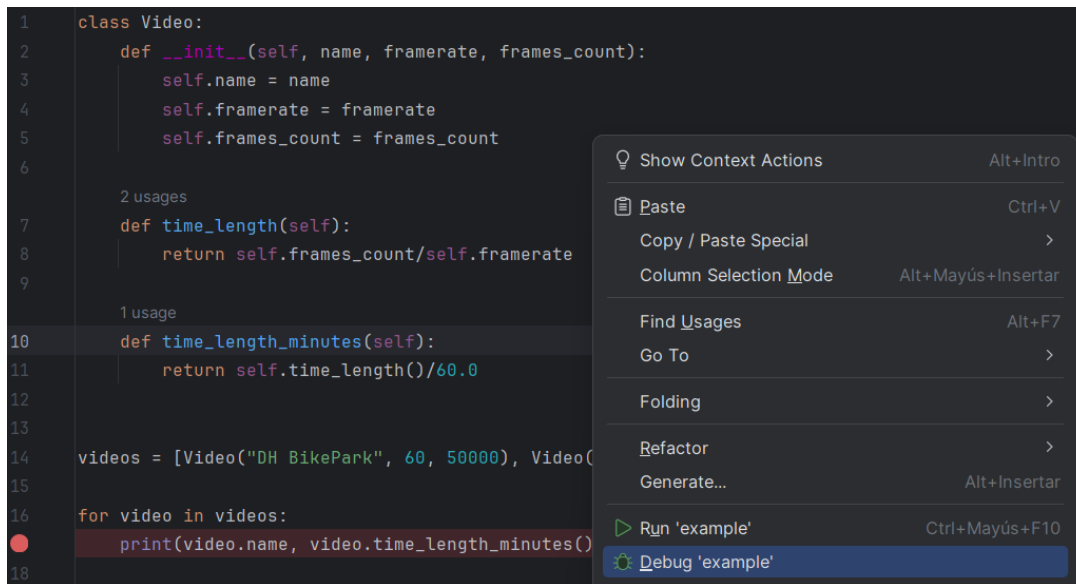


Activar breakpoint con clic en el número de la línea.

Debugging

Iniciar debugging

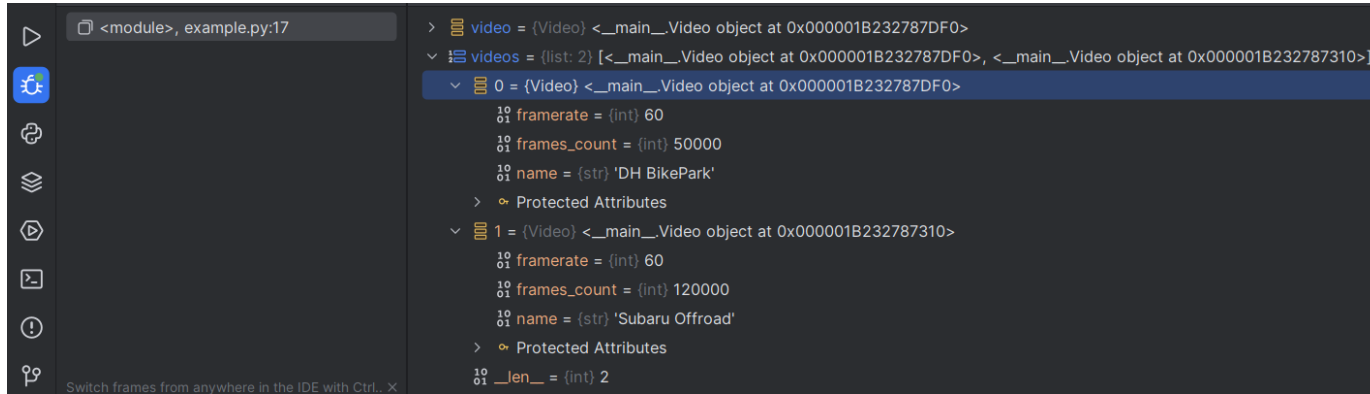
```
1 class Video:
2     def __init__(self, name, framerate, frames_count):
3         self.name = name
4         self.framerate = framerate
5         self.frames_count = frames_count
6
7     2 usages
8     def time_length(self):
9         return self.frames_count/self.framerate
10
11     1 usage
12     def time_length_minutes(self):
13         return self.time_length()/60.0
14
15 videos = [Video("DH BikePark", 60, 50000), Video(...)]
16
17 for video in videos:
18     print(video.name, video.time_length_minutes())
```



Iniciar debugging con
opción “debug”.

Debugging

Suspensión temporal y observar estado



The screenshot shows a Python IDE with a breakpoint set at line 17 of 'example.py'. The variable explorer on the right displays the state of the program at the breakpoint. The 'videos' list contains two Video objects. The first object (index 0) has a framerate of 60, frames_count of 50000, and name 'DH BikePark'. The second object (index 1) has a framerate of 60, frames_count of 120000, and name 'Subaru Offroad'. The '__len__' attribute is also visible, showing a value of 2.

```
> video = {Video} <__main__.Video object at 0x000001B232787DF0>  
v videos = {list: 2} [<__main__.Video object at 0x000001B232787DF0>, <__main__.Video object at 0x000001B232787310>]  
v 0 = {Video} <__main__.Video object at 0x000001B232787DF0>  
  01 framerate = {int} 60  
  01 frames_count = {int} 50000  
  01 name = {str} 'DH BikePark'  
  > Protected Attributes  
v 1 = {Video} <__main__.Video object at 0x000001B232787310>  
  01 framerate = {int} 60  
  01 frames_count = {int} 120000  
  01 name = {str} 'Subaru Offroad'  
  > Protected Attributes  
  01 __len__ = {int} 2
```



El programa se ha detenido en el breakpoint y podemos observar los valores de las variables.



Continuamos la ejecución presionando el botón “resume”:





Trabajo grupal – Bloque B

Paralelo 2

G1	G2	G3	G4
Nicolas Mardones	Víctor Meza Herrera	Daniela Méndez Gándara	Claudia Blanco
Manuel Denis	Estefania Manriquez	Álvaro Pérez	Ariel Inostroza
Bryan Castillo	Patricio Vera	Pedro Nahum	Héctor Aguayo
GERALDY SUAREZ	Oscar Torres	Javier Gajardo	Ruben Sanhueza Ramirez
Scarlett Espinoza	Braulio Quiroz	Angela Proboste Neira	Félix González
Ulises Campodónico	Yerko Gallardo	Nicolás Guzmán	Ariel Mora
Carol Leiva	Rodrigo Araya		
G5	G6	G7	G8
Fabian Díaz	Camila Oyarzún	Mayerlyn Rodriguez	Daniela Porto
Natalia Rivera	Stefanya Pulgar	Sebastian Vega	Cristian Chavez Jara
Juan Salinas	Carlos Emilio Azócar Riquelme	Efrain Duarte Campos	Juan Rodrigo Vega
Rodrigo Pastén Cortés	Nicolas Rojas	Bianel Bianchini	Rodolfo Cantillana
Flavio Jara R.	luis.paillan.cnc@gmail.com	Bastián Gamboa Labbé	Abraham Ruiz
Daniel García	Cristóbal Gajardo	Pablo Uribe	Rodrigo Álvarez

Trabajo grupal – Ejercicio #1

Base de ejercicio

- Incorpore el siguiente código en su IDE y estúdielo.

```
FIN = False

registro_llamado = {"frecuencia": None, "motivo": None, "fecha": None}
listado_llamados = []

while not FIN:
    frecuencia = input("Ingrese frecuencia: ")
    if frecuencia == "FIN":
        FIN = True
    else:
        motivo = input("Ingrese motivo: ")
        fecha = input("Ingrese fecha: ")
        registro_llamado["frecuencia"] = frecuencia
        registro_llamado["motivo"] = motivo
        registro_llamado["fecha"] = fecha

        listado_llamados.append(registro_llamado)

print(listado_llamados)
```

Trabajo grupal – Ejercicio #2

Debugging de código

- Una vez que haya estudiado y ejecutado el código, identifique el principal resultado erróneo.
- Identifique líneas que sean de su interés, actívelas como puntos de suspensión (breakpoint) y ejecute un proceso de debugging.
- Discuta el problema subyacente y corrija el código para que este funcione adecuadamente.

¿Preguntas?

¡Hemos llegado al final de la clase!

