Esta clase va a ser

- grabada

Clase 10. PYTHON

Excepciones

Temario

09

Funciones II

- ✓ Argumentos y parámetros

10

Excepciones

- ✓ Errores
- ✓ Excepciones

11

Programación Orientada a Objetos I

- ✓ ¿Qué es la POO?
- ✓ Relación entre clases

CODERHOUSE

Objetivos de la clase

- **Identificar** las diferencias entre errores y excepciones.
- **Utilizar** excepciones existentes.
- **Crear** excepciones propias.

Repositorio Github

Te dejamos el acceso al Repositorio de Github donde encontrarás todo el material complementario y scripts de la clase.

✓ Repositorio Python

CODERHOUSE

Errores y excepciones

Aprendiendo del error

En la programación, se aprende a base de equivocarse, por lo tanto, debemos tolerar nuestros propios fallos para poder avanzar. Programar no es fácil, podemos equivocarnos desde la concepción de la idea o también al escribir código, para lo primero poco podemos hacer, ya que es algo que mejorará con la práctica.

¡A programar se aprende programando y cometer errores es la prueba de que estás avanzando!

Errores y excepciones

Hasta ahora los mensajes de error no habían sido más que mencionados, pero si probaste los ejemplos probablemente hayas visto algunos. Hay (al menos) dos tipos diferentes de errores: errores de sintaxis y excepciones.

Cuando el programa falla ocasiona su detención, por lo cual tenemos que ser capaces de detectar por qué y cómo prevenir estas fallas.

Fuente: EntrenamientoPython

Errores

Errores

Si algo tienen la mayoría de lenguajes de programación es que son exigentes con las instrucciones. Cuando en nuestro programa ocurre algún fallo Python nos lanzará un aviso indicando que ocurrió y detendrá la ejecución.

Los errores sintácticos están ligados a la sintaxis, que es la escritura de las instrucciones.

Fuente: EntrenamientoPython

Errores sintácticos

Algunos de los errores sintácticos más comunes:

- ✓ Cerrar paréntesis
- ✓ Cerrar comillas o utilizar distintas comillas al abrir y cerrar
- ✓ Tabular mal el código
- ✓ No poner dos puntos al definir una función

Fuente: EntrenamientoPython

Errores sintácticos

```
>>> print(“Hola”
File "<ipython-input-1-8bc9f5174855>", line 1
    print("Hola"
                ^
SyntaxError: unexpected EOF while parsing
```

Vamos a provocar algunos errores

EOF significa **end of file**, el error nos indica que esperaba que se cierre con un paréntesis.

Fuente: EntrenamientoPython

Errores de nombre

Se producen cuando el sistema interpreta que debe ejecutar alguna función, pero no está definida. Devuelve NameError:

```
>>> pint(“Hola”)
<ipython-input-2-155163d628c2> in <module>()
----> 1 pint("Hola")

NameError: name 'pint' is not defined
```

Pint no es la función print para imprimir por pantalla.

Fuente: EntrenamientoPython

Errores semánticos

La mayoría de errores sintácticos y de nombre los identifican los editores de código antes de la ejecución, pero existen otros tipos que pasan más desapercibidos.

Estos errores son muy difíciles de identificar porque van ligados al sentido del funcionamiento y dependen de la situación. Algunas veces pueden ocurrir y otras no.

Fuente: EntrenamientoPython

Errores semánticos

La mejor forma de prevenirlos es programando mucho y aprendiendo de tus propios fallos, la experiencia es la clave. Veamos a continuación algunos ejemplos.

Fuente: EntrenamientoPython

Errores semánticos

```
>>> lista = []
>>> lista.pop()
<ipython-input-6-9e6f3717293a> in
<module>()
----> 1 l.pop()

IndexError: pop from empty list
```

Si intentamos sacar un elemento de una lista vacía, algo que no tiene mucho sentido, el programa dará fallo de tipo IndexError.
Esta situación ocurre sólo durante la ejecución del programa, por lo que los editores no lo detectarán:

Fuente: EntrenamientoPython

Errores semánticos

```
>>> lista = []
>>> if len(lista) > 0:
lista.pop()
```

Para prevenir el error deberíamos comprobar que una lista tenga como mínimo un elemento antes de intentar sacarlo, algo factible utilizando la función len():

Fuente: EntrenamientoPython

Errores semánticos

Cuando leemos un valor con la función input(), éste siempre se obtendrá como una cadena de caracteres. Si intentamos operarlo directamente con otros números tendremos un fallo TypeError que tampoco detectan los editores de código:

```
n = input("Ingresar n:\n")

m = 4

print(f"---> {n}/{m} = {n/m}")
```

Fuente: EntrenamientoPython

CODERHOUSE

Errores semánticos

```
>>> n = float(input("Introduce un número: "))
>>> m = 4
>>> print("{}/{} = {}".format(n,m,n/m))
```

Como ya sabemos este error se puede prevenir transformando la cadena a entero o flotante:

Fuente: EntrenamientoPython

Desafío de errores

Retorna None cuando sucede el fallo

Duración: **10 minutos**

CODERHOUSE

ACTIVIDAD EN CLASE

Desafío de errores

Descripción de la actividad.

En la función de nuestro ejercicio hay un fallo potencial, averigua cuándo sucede y retorna el valor None en ese caso especial, en cualquier otro caso devuelve el resultado normal de la función.

```
>>> def dividir(a, b):
return a/b
```

Pista: Valor indeterminado

Break

¡10 minutos y volvemos!

Excepciones

Excepciones

Incluso si la sentencia o expresión es sintácticamente correcta, puede generar un error cuando se intenta ejecutarla. Los errores detectados durante la ejecución se llaman excepciones, y no son incondicionalmente fatales: pronto aprenderás cómo manejarlos en los programas en Python.

Fuente: EntrenamientoPython

Excepciones

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ZeroDivisionError: integer division or modulo by zero
>>> 4 + spam*3
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'spam' is not defined
```

Sin embargo, la mayoría de las excepciones no son manejadas por los programas, y resultan en mensajes de error como los mostrados aquí:

Fuente: EntrenamientoPython

Excepciones

Como podemos suponer, es difícil prevenir fallos que ni siquiera nos habíamos planteado que podían existir. Por suerte para esas situaciones existen las excepciones.

Las excepciones son bloques de código que nos permiten continuar con la ejecución de un programa pese a que ocurra un error.

Fuente: EntrenamientoPython

Try- Except

CODERHOUSE

Try-Except

```
try:
    n = float(input("Ingresar n:\n") ) #input dan una cadena
    m = 4
    print(f"---> {n}/{m} = {n/m}")
except:
  print("Algo salio!!!! :( ")
```

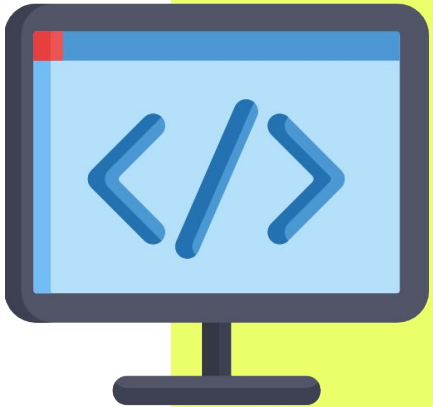
Para prevenir el fallo debemos poner el código propenso a errores en un bloque try y luego encadenar un bloque except para tratar la situación excepcional mostrando que ha ocurrido un fallo:

Fuente: THernandez

Try- Except

La sentencia **try** funciona de la siguiente manera:

- ✓ Se ejecuta el bloque try (el código entre las sentencias try y except).
- ✓ Si no ocurre ninguna excepción, el bloque except se saltea y termina la ejecución de la sentencia try.

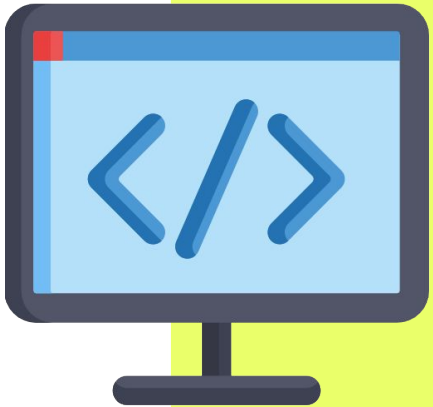


Fuente: THernandez

Try- Except

La sentencia try funciona de la siguiente manera 👇

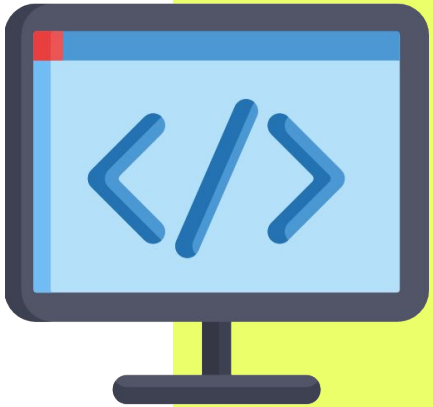
- ✓ Si ocurre una excepción durante la ejecución del bloque try, el resto del bloque se saltea. Luego, si su tipo coincide con la excepción nombrada luego de la palabra reservada except, se ejecuta el bloque except, y la ejecución continúa luego de la sentencia try.



Fuente: THernandez

Try- Except

- ✓ Si ocurre una excepción que no coincide con la excepción nombrada en el except, esta se pasa a declaraciones try de más afuera; si no se encuentra nada que la maneje, es una excepción no manejada, y la ejecución se frena con un mensaje como los mostrados arriba.



Fuente: THernandez

CODERHOUSE

Try-Except

Try - Except **nos permite controlar situaciones excepcionales que generalmente darían error** y en su lugar permite mostrar un mensaje o ejecutar una pieza de código alternativo.

Else

Else

Las declaraciones try - except tienen un bloque else opcional, el cual, cuando está presente, debe seguir a los except. El bloque else es un buen momento para romper la iteración con break si todo funciona correctamente.

El uso de else es mejor que agregar código adicional en el try porque evita capturar accidentalmente una excepción que no fue generada por el código que está protegido por la sentencia try - except.

Else en ejemplo

```
while (True):
   try:
      a = float(input("Introduce un número: "))
      b = float(input("Introduce otro número: "))
      print(a + b)
   except:
      print("Ha ocurrido un error. Tienes que introducir 2
números.")
   else:
      print("La suma se ha realizado correctamente.")
      break  # Importante romper la iteración si todo ha ido
bien.
```

Finally

Finally

La sentencia try tiene otra sentencia opcional que intenta definir acciones de limpieza que deben ser ejecutadas bajo ciertas circunstancias. Una sentencia finaly siempre es ejecutada antes de salir de la sentencia try, ya sea que una excepción haya ocurrido o no.

Finally

Cuando ocurre una excepción en la sentencia try y no fue manejada por una sentencia except (u ocurrió en una sentencia except o else), es relanzada luego de que se ejecuta la sentencia finally.

La sentencia finally es también ejecutada «a la salida» cuando cualquier otra sentencia de la sentencia try es dejada vía break, continue or return.

Finally en ejemplo

```
while (True):
    try:
        a = float(input("Introduce un número: "))
        b = float(input("Introduce otro número: "))
        print(a + b)
    except:
        print("Ha ocurrido un error. Tienes que introducir 2 números.")
    else:
        print("La suma se ha realizado correctamente.")
        break   # Importante romper la iteración si todo ha ido bien.
    finally:
        print("Fin del bucle") # Esto se ejecuta siempre.
```

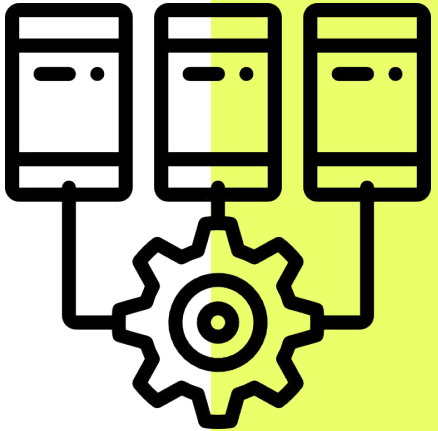
Excepciones múltiples

CODERHOUSE

Excepciones múltiples

En una misma pieza de código pueden ocurrir muchos errores distintos y quizá nos interese actuar de forma diferente en cada caso.

Para esas situaciones algo que podemos hacer es asignar una excepción a una variable.



Excepciones múltiples

De esta forma es posible analizar el tipo de error que sucede gracias a su identificador:

```
>>> try:
        n = input("Introduce un número: ")  # no transformamos a número
        5/n
    except Exception as e:  # guardamos la excepción como una variable e
        print("Ha ocurrido un error =>", type(e).__name__)
```

Excepciones múltiples

```
>>> print( type(e) )
```

Cada error tiene un identificador único que curiosamente se corresponde con su tipo de dato. Aprovechando eso podemos mostrar la clase del error utilizando la sintaxis:

Excepciones múltiples

```
>>> print(type(1))
>>> print(type(3.14))
>>> print(type([]))
>>> print(type(()))
```

Es similar a conseguir el tipo (o clase) de cualquier otra variable o valor literal:

Excepciones múltiples

```
>>> print( type(e).__name__)
>>> print(type(1).__name__)
>>> print(type(3.14).__name__)
>>> print(type([]).__name__)
>>> print(type(()).__name__)
```

Como vemos, siempre nos indica "class" delante. Eso es porque en Python todo son clases, pero hablaremos de este concepto más adelante. Lo importante ahora es que podemos mostrar solo el nombre del tipo de dato (la clase) consultando su propiedad especial name de la siguiente forma:

Excepciones múltiples

Gracias a los identificadores de errores podemos crear múltiples comprobaciones, siempre que dejemos en último lugar la excepción por defecto. Excepción que engloba cualquier tipo de error.

Veamos a continuación un ejemplo

Excepciones múltiples

```
>>> try:
        n = float(input("Introduce un número divisor: "))
        5/n
    except TypeError:
        print("No se puede dividir el número entre una cadena")
    except ValueError:
        print("Debes introducir una cadena que sea un número")
    except ZeroDivisionError:
        print("No se puede dividir por cero, prueba otro número")
    except Exception as e:
        print("Ha ocurrido un error no previsto", type(e).__name__ )
```

Repositorio Github

Te dejamos el acceso al Repositorio de Github donde encontrarás todo el material complementario y scripts de la clase.

✓ Repositorio Python

CODERHOUSE

Desafío de excepciones

Captura la excepción

Duración: **5 minutos**

CODERHOUSE

ACTIVIDAD EN CLASE

Descripción de la actividad.

Tomando la solución del ejercicio anterior, en lugar de devolver None al dividir entre cero, tienes que capturar una excepción que muestre por pantalla EXACTAMENTE el mensaje: “No se puede dividir entre cero”; si falla el código

```
>>> def dividir(a, b):
        if b == 0:
        return None
        return a/b
```

Primera pre-entrega de tu Proyecto final

Debes entregar la primera pre-entrega de tu proyecto final.

CODERHOUSE

Primera pre-entrega

Objetivo

- ✓ Practicar el concepto de funciones. Preparar la parte lógica para el registro de usuarios.

Consigna

- ✓ Crear un programa que permita emular el registro y almacenamiento de usuarios en una base de datos. Crear el programa utilizando el concepto de funciones, diccionarios, bucles y condicionales.

Formato

- ✓ El proyecto debe compartirse utilizando Colab bajo el nombre “**Primera pre-entrega+Apellido**”.

Primera pre-entrega

Se debe entregar

- ✓ Se debe entregar todo el programa.

Sugerencias

- ✓ El formato de registro es: Nombre de usuario y Contraseña.
- ✓ Utilizar una función para almacenar la información y otra función para mostrar la información.
- ✓ Utilizar un diccionario para almacenar dicha información, con el par usuario-contraseña (clave-valor).
- ✓ Utilizar otra función para el login de usuarios, comprobando que la contraseña coincida con el usuario.

Primera pre-entrega

Adicional

- ✓ Utilizando los conceptos de la clase 8, guarde la información en un archivo de texto dentro de su Drive."

Ejemplo

- ✓ Ver video explicativo

Recuerden que en la clase 1 tienen disponible un proyecto modelo de un ex estudiante para tomar como inspiración.

Actividad extra N° 6

Tutorial Excepciones

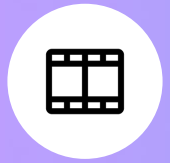
Consigna

✓ Replicar el archivo titulado: Tutorial de Excepciones.

CODERHOUSE

#Codertraining

¡No dejes para mañana lo que puedes practicar hoy! Te invitamos a revisar la **Guía de Ejercicios Complementarios**, donde encontrarás un ejercicio para poner en práctica lo visto en la clase de hoy.



¿Quieres saber más?
Te dejamos material ampliado de la clase

CODERHOUSE

MATERIAL AMPLIADO

Recursos multimedia

Título

- ✓ Artículo: Errores
- ✓ Artículo: Excepciones
- ✓ Artículo: Else
- ✓ Artículo: Finally
- ✓ Artículo: Excepciones integradas
- ✓ Repaso Ejercicios

Disponible en nuestro repositorio.

¿Preguntas?

Resumen
de la clase hoy

- ✓ Errores
- ✓ Excepciones
- ✓ Excepciones Múltiples

Opina y valora esta clase

Muchas gracias.

#DemocratizandoLaEducación