Excepciones en Python 3

1. ¿Qué son las excepciones?

Python utiliza un tipo de objeto especial denominado excepción para gestionar los errores que surgen durante la ejecución de un programa.

Cada vez que ocurre un error que hace que Python no sepa qué hacer a continuación, crea un objeto de excepción. Si se escribe código que maneja la excepción, el programa continuará ejecutándose. Por el contrario, si no se maneja la excepción, el programa se detendrá y mostrará un pequeño resumen de la excepción que se ha producido.

Las excepciones se pueden manejan a través de las sentencias `try` y `except`.

```
print(var)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-2ea387ab95ff> in <module>
----> 1 print(var)

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

SEARCH STACK OVERFLOW

2. Errores de sintaxis vs Excepciones

A pesar de que en muchas ocasiones estos dos términos se utilizan de manera indistinta cuando hablamos de Python, debemos tener cuidado porque son cosas diferentes.

Los **errores de sintaxis** se producen cuando escribimos una sentencia de código en Python que no es sintácticamente válida. El intérprete de Python indica estos error con el término `SyntaxError` y nos señala con el carácter `^` donde se encuentra el error.

```
print("Hola mundo)
```

```
  File "<ipython-input-2-62a1fbc8656d>", line 1
    print("Hola mundo)
                      ^
SyntaxError: EOL while scanning string literal
```

SEARCH STACK OVERFLOW

Este tipo de errores no se pueden controlar y no se corresponden con excepciones dentro del lenguaje

Por otro lado, las **excepciones** se corresponden con errores que se producen en sentencias de código en Python que son sintácticamente correctas. Esto tipo de errores no son fatales para el programa y pueden ser gestionados o ignorados.

```
print(var)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-3-2ea387ab95ff> in <module>
----> 1 print(var)

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

SEARCH STACK OVERFLOW

```
50/0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-4-ff5d2f231fa4> in <module>
----> 1 50/0

ZeroDivisionError: division by zero
```

SEARCH STACK OVERFLOW

```
'2' + 2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-5-d2b23a1db757> in <module>
----> 1 '2' + 2

TypeError: can only concatenate str (not "int") to str
```

SEARCH STACK OVERFLOW

3. Gestión de excepciones: Sentencias `try` y `except`

3. Gestión de excepciones: Sentencias `try` y `except`

Las sentencias `try` y `except` en Python pueden utilizarse para capturar y manejar excepciones. Python ejecuta el código que sigue a la sentencia `try` como una parte "normal" del programa. El código que sigue a la sentencia `except` se ejecutará si se produce cualquier excepción en la cláusula `try` precedente.

La sintaxis que se utiliza para definir este tipo de comportamiento es la siguiente:

```
try:
    <sentencia(s)>
except <excepción>:
    <sentencias(s) si excepción>
```

```
print(var)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-6-2ea387ab95ff> in <module>
----> 1 print(var)

NameError: name 'var' is not defined

SEARCH STACK OVERFLOW
```

```
try:
    print(var)
except NameError:
    var = "Hola mundo"
```

```
Hola mundo
```

```
print(var)
```

```
Hola mundo
```

Debemos tener en cuenta que el las sentencias de código que se encuentren en el cuerpo de la sentencia `try` y a continuación de la sentencia que emite la excepción, no se ejecutarán

```
print("Sentencias de codigo antes del try")

try:
    print("Codigo antes de la excepcion")
```

```
    10 + "3"
    print("Codigo despues de la excepcion")
except TypeError:
    print("No se puede sumar un numero entero y un string")

print("Sentencias de codigo despues del except")
```

```
Sentencias de codigo antes del try
Codigo antes de la excepcion
No se puede sumar un numero entero y un string
Sentencias de codigo despues del except
```

También podemos utilizar la sentencia `except` sin indicarle el nombre de ninguna excepción, en estos casos capturará todas las excepciones que se produzcan en el código que se encuentra en el cuerpo de la sentencia `try`

```
try:
    10 + "3"
except:
    print("No se puede sumar un entero y un string")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-12-be7886893732> in <module>
----> 1 10 + "3"
      2 try:
      3     10 + "3"
      4 except:
      5     print("No se puede sumar un entero y un string")

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

SEARCH STACK OVERFLOW

Adicionalmente a la sintaxis anterior, podemos capturar varias excepciones de manera simultánea utilizando varias cláusulas `except`

```
try:
    50/0
except NameError:
    print("Gestionando excepcion NameError")
except TypeError:
    print("Gestionando execpcion Type Error")
except ZeroDivisionError:
    print("No puedes dividir un numero por 0")
```

```
No puedes dividir un numero por 0
```

Por último, podemos asignar el objeto excepción capturado a una variable y utilizarlos para mostrar más información al respecto.

```
try:
    print(variable)
except NameError as error:
    print("Objeto de tipo:", type(error))
    print("La excepcion consiste en:", error)
```

```
Objeto de tipo: <class 'NameError'>
La excepcion consiste en: name 'variable' is not defined
```

4. Lanzando excepciones personalizadas

Además de las sentencias anteriores que podemos utilizar para controlar excepciones, Python nos proporciona la sentencia `raise` con la que podemos emitir nuestras propias excepciones. Para ello, debemos utilizar la clase por defecto de Python `Exception`

```
help(Exception)
```

```
Help on class Exception in module builtins:

class Exception(BaseException)
 |  Common base class for all non-exit exceptions.
 |
 |  Method resolution order:
 |      Exception
 |      BaseException
 |      object
 |
 |  Built-in subclasses:
 |      ArithmeticError
 |      AssertionError
 |      AttributeError
 |      BufferError
 |      ... and 15 other subclasses
 |
 |  Methods defined here:
 |
 |  __init__(self, /, *args, **kwargs)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  __new__(*args, **kwargs) from builtins.type
```

```
|      Create and return a new object.  See help(type) for accurate signature.
|
|  ----------------------------------------------------------------------
|  Methods inherited from BaseException:
|
|  __delattr__(self, name, /)
|      Implement delattr(self, name).
|
|  __getattribute__(self, name, /)
|      Return getattr(self, name).
|
|  __reduce__(...)
|      Helper for pickle.
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __setattr__(self, name, value, /)
|      Implement setattr(self, name, value).
|
|  __setstate__(...)
|
|  __str__(self, /)
|      Return str(self).
|
|  with_traceback(...)
|      Exception.with_traceback(tb) --
|      set self.__traceback__ to tb and return self.
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from BaseException:
|
```

```
colores_permitidos = ("azul", "verde", "rojo")

color = "amarillo"

if color not in colores_permitidos:
    raise Exception("El color {} no se encuentra en la lista de colores permitidos".format(
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
<ipython-input-20-da78555fe513> in <module>
      4
      5 if color not in colores_permitidos:
----> 6     raise Exception("El color {} no se encuentra en la lista de colores
permitidos".format(color))

Exception: El color amarillo no se encuentra en la lista de colores permitidos
```

SEARCH STACK OVERFLOW

```
colores_permitidos = ("azul", "verde", "rojo")
```

```
color = "verde"

if color not in colores_permitidos:
    raise Exception("El color {} no se encuentra en la lista de colores permitidos".format(
```

5. Excepción `AssertionError`

Como complemento a todas las sentencias anteriores, Python nos proporciona una sentencia adicional que nos permite verificar en un punto determinado de nuestro programa que todo esta funcionando adecuadamente, esta sentencia es `assert`.

```
passwd = input("Introduce una contrasena de mas de 8 digitos:")

assert len(passwd) > 8, "La contrasena es menor a 8 digitos"
```

```
 Introduce una contrasena de mas de 8 digitos:123456789
```

6. Cláusula `else` en excepciones

Curiosamente, Python nos proporciona un mecanismo por el cual utilizando la sentencia `else`, se puede indicar a un programa que ejecute un determinado bloque de código sólo en ausencia de excepciones.

```
try:
    print(variable2)
except NameError:
    print("variable2 no estaba definida, se define con la cadena 'Hola mundo'")
    variable2 = 'Hola mundo'
else:
    print("variable2 ya estaba definida con el valor:", variable2)
```

```
 Hola mundo
 variable2 ya estaba definida con el valor: Hola mundo
```

7. Sentencia `finally`

Python nos proporciona una última sentencia que podemos utilizar para realizar una "limpieza" después de la ejecución de nuestro código al gestionar una excepción. Esta sentencia se denomina `finally` y el código que se localice en su cuerpo, se ejcutará siempre, independientemente de si se produce o no la excepción

Independientemente de si se produce o no la excepción.

```
try:
    print(variable3)
except NameError:
    print("variable3 no estaba definida, se define con la cadena 'Hola mundo'")
    variable3 = 'Hola mundo'
else:
    print("variable3 ya estaba definida con el valor:", variable2)
```

```
 Hola mundo
 variable3 ya estaba definida con el valor: Hola mundo
```

```
try:
    print(variable3)
except NameError:
    print("variable3 no estaba definida, se define con la cadena 'Hola mundo'")
    variable3 = 'Hola mundo'
else:
    print("variable3 ya estaba definida con el valor:", variable2)
finally:
    del variable3
```

```
 variable3 no estaba definida, se define con la cadena 'Hola mundo'
```

Productos de pago de Colab - Cancelar contratos