CURSOS INTERSEMESTRALES

PROTECO

Manejo de excepciones

Python Intermedio
junio 2019

Tipos de Errores

1.-Error de Sintaxis:
Se produce al escribir de forma incorrecta una parte del programa. Detectados por el parser (analizador sintáctico) en el mayoría de los casos.

2.-Error Lógico
Se produce cuando los resultados del programa no son los esperados-

3.- Excepciones (error en tiempo de ejecución)
Se produce cuando no se puede ejecutar una instrucción por un error determinado.

PROTECO

Excepciones

Incluso si la declaración 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.

Sin embargo, la mayoría de las excepciones no son manejadas por los programas, y resultan en mensajes de error como se muestra a continuación:

PROTECO

Excepciones

```
>>> 5*(6/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>> 1+numero*3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'numero' is not defined
>>> 2+'7'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

La última línea de los mensajes de error indica qué sucedió. Las excepciones vienen de distintos tipos, y el tipo se imprime como parte del mensaje. El resto de la línea provee un detalle basado en el tipo de la excepción y qué la causó.

Analizando los errores

Ejemplo 1: Cuando se divide entre 0

x = 5*(6/0)

6/0 no es posible efectuarse, por lo que si integramos eso en nuestro programa nos enviará un error del tipo: 'ZeroDivisionError: division by zero'

Ejemplo 2 : Utilizar variables no definidas

y = 1 + numero*3

'numero' no ha sido definida, por lo que no es posible efectuar esa operación indicada, por lo tanto genera un error de tipo: 'NameError: name 'numero' is not defined'

PROTECO

Analizando los errores

Ejemplo 3: Multiplicar variables de tipo cadena con variables de tipo entero

z = 2+ '7'

'7' es una cadena que contiene al número 7, sin embargo por haber sido definida así, no es posible que se opere con el número 2. Por esta razón genera una excepción de tipo: 'TypeError: unsupported operand type(s) for +: 'int' and 'str''

PROTECO

Más tipos de errores

Puedes consultar la documentación de los tipos de errores existentes:

https://docs.python.org/2/library/exceptions.html

PROTECO

Manejar excepciones

Si surge una excepción, causará que se detenga la ejecución de nuestro código. Y se buscará el código (de haberlo), para manejar la excepción.

¿Para qué sirve manejarlas?

*Evitar un fin abrupto del programa.

*Notificar eventos

*Garantizar que se realizaran ciertas acciones antes de que termine el programa

PROTECO

Capturar excepciones

Los recursos de gestión y captura de excepciones son:

- **try**
- **except**
- **else**
- **finally**

try-except ejemplo

```
try:
    <Terminar la universidad>
except  reprobé materias:
    <cursarlas en extra>
except  no me gustó la carrera:
    <vender Bonice>
```

PROTECO

try-except sintaxis (sólo un except)

```
<flujo principal>
...
...
try:
    <bloque de código en riesgo>
except :
    <bloque inscrito a except>
<flujo principal>
```

try-except sintaxis (varios except)

```
<flujo principal>
...
try:
   <código susceptible de errores>
except <ErrorTipo1>:
   <bloque inscrito a ErrorTipo1>
except< ErrorTipo2>:
   <bloque inscrito a ErrorTipo2>
except:
   <bloque inscrito a except general>
<flujo principal>
```

PROTECO

try-except

*Primero, se ejecuta el bloque try (el código entre las declaración try y except).

*Si no ocurre ninguna excepción, el bloque except se salta y termina la ejecución de la declaración try.

*Si ocurre una excepción durante la ejecución del bloque try, el resto del bloque se salta. 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 declaración try.

*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.

else y finally

Además de except, Python cuenta con otros dos recursos que completan la gestión de excepciones.La expresión **'else'**. Del mismo modo que un if, la expresión else se ejecuta en el caso de que **no se genere una excepción.**

En algunas situaciones, es necesario realizar algunas operaciones independientemente de que haya ocurrido una excepción o no. Para estas situaciones se utiliza la expresión **finally, la cual se ejecuta haya existido una excepción o no.**

PROTECO

try-except-else ejemplo

```
try:
    <Terminar la universidad>
except  reprobé materias:
    <cursarlas en extra>
except  no me gustó la carrera:
    <vender Bonice>
else:
    <Tener un título>
```

try-except-else sintaxis (varios except)

```
<flujo principal>
...
try:
  <código suceptible de errores>
except <ErrorTipo1>:
   <bloque inscrito a ErrorTipo1>
except (<ErrorTipo2>, <ErrorTipo2>):
   <<bloque inscrito a ErrorTipo2 y ErrorTipo3>
except:
   <bloque inscrito a except general>
else:
   <bloque inscrito a else>
<flujo principal>
```

try-except-else-finally ejemplo

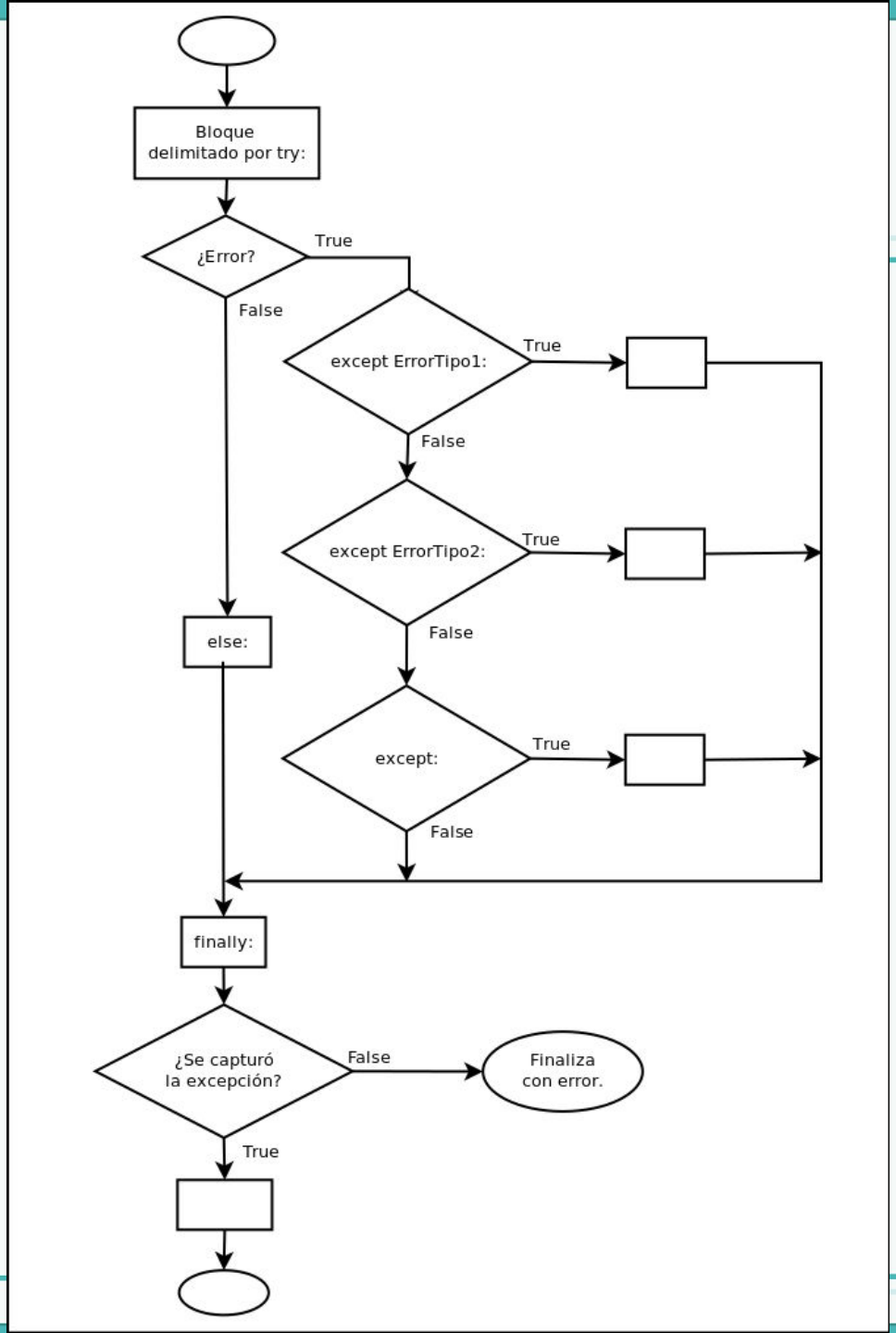
```
try:
   <Terminar la universidad>
except  reprobé materias:
   <cursarlas en extra>
except  no me gustó la carrera:
   <vender Bonice>
else:
   <Tener un título>
finally:
   <Ser feliz>
```

PROTECO

try-except-else-finall y sintaxis (varios except)

```
<<flujo principal>
...
try:
  <código suceptible de errores>
except <ErrorTipo1>:
   ...
except:
   <bloque inscrito a except general>
else:
   <bloque inscrito a else>
finally:
   <bloque inscrito a finally>
<flujo principal>
```

PROTECO


Bloque
delimitado por try:
¿Error?
True
False
except ErrorTipo1:
True
False
except ErrorTipo2:
True
False
except:
True
False
else:
finally:
¿Se capturó
la excepción?
False
Finaliza
con error.
True

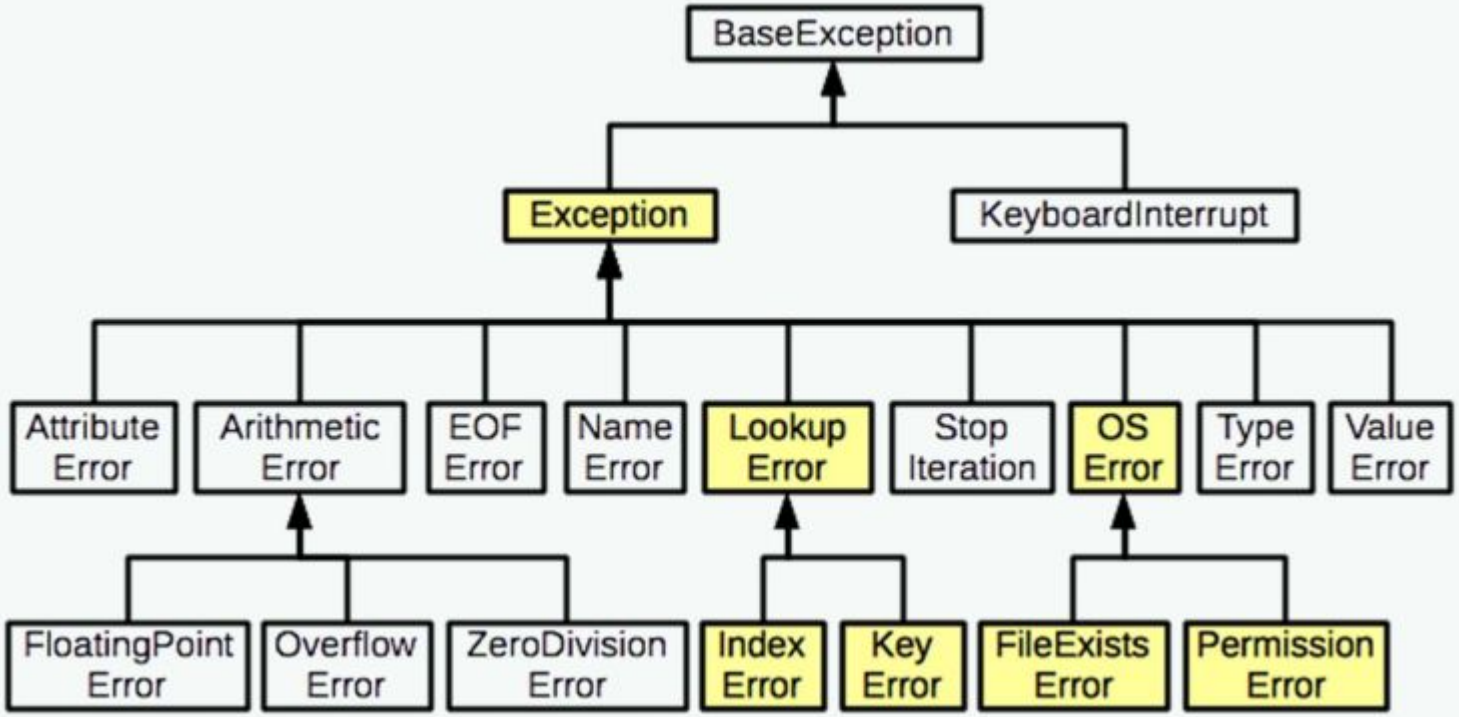
PROTECO

Levantar excepciones

```
try:
    raise tipodeerror
except:
    <bloque inscrito a raise>
```

PROTECO

Jerarquía de excepciones



PROTECO