Python Programming

Módulo 2

Capturar excepciones

Capturar excepciones

Python cuenta con soporte de primer nivel para excepciones. Las **excepciones** son un modelo para el manejo de errores.

En otros lenguajes, como en C, se acostumbra a que las funciones retornen un valor específico cuando ocurre algún error. El código que llama a dicha función chequea el valor de retorno y toma las medidas necesarias. En Python, cuando una función falla, en lugar de retornar un valor en particular lanza una excepción (vía la palabra reservada `raise`). El código que invocó a dicha función puede implementar un bloque para capturarla vía `try` y `except`.

Es posible hacer uso de las excepciones que incorpora el lenguaje. Algunas de ellas son: `ValueError, TypeError, RuntimeError, KeyError` (la convención de nombramiento difiere de las funciones y otros objetos; en las excepciones cada término se escribe con su primera letra en mayúscula y no se emplean guiones bajos).

Por ejemplo, `int()` sirve para convertir números de coma flotante y cadenas a su respectiva representación númerica.

```
>>> int("3")
3
```

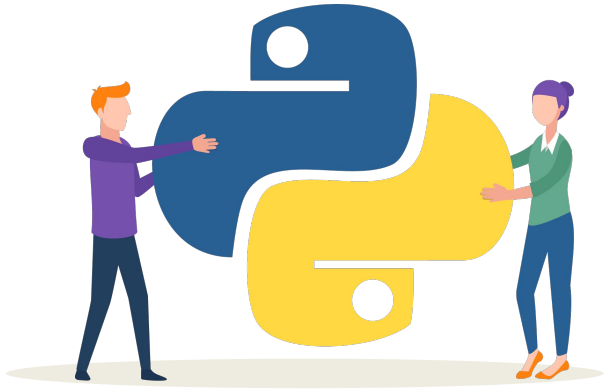
Cuando la cadena no puede ser convertida a entero porque no es un número, lanza la excepción `ValueError`.

```
>>> int("Hola mundo")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'Hola mundo'
```

Utilizando las palabras claves `try` y `except` podemos capturar la excepción, evitando que se propague y generalmente ejecutando alguna otra acción.

```
try:
    int("Hola mundo")
except ValueError:
    print("Eso no es un número.")

out: Eso no es un número.
```

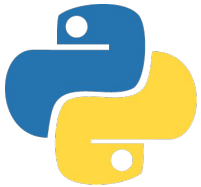


Es importante aclarar que, si una excepción se propaga sin que ningún bloque de código la capture, el programa finaliza inmediatamente sin concluir las líneas de código siguientes si las hubiera.

Ahora bien, `int()` lanzará `TypeError` cuando se le pase como argumento un objeto que no pueda ser representado como un número, por ejemplo, una lista. Simplemente podemos agregar otra cláusula `except` para capturar dicha excepción.

```
try:
    int([1,2,3])
except ValueError:
    print("Eso no es un número.")
except TypeError:
    print("Eso es una colección, no se puede convertir a entero")

out: Eso es una colección, no se puede convertir a entero
```



O bien si queremos capturar ambas excepciones en un mismo bloque de código:

```
try:
    int([1,2,3])
except (ValueError, TypeError):
    print("Tipo de dato incorrecto o no puede ser convertido")
```

```
out: Tipo de dato incorrecto o no puede ser convertido
```

La cláusula `try/except` también acepta `else`, un bloque de código que es ejecutado cuando no se ha capturado ninguna de las excepciones de las contempladas.

```
try:
    int("10")
except Exception:
    print("Ocurrió un error.")
else:
    print("Todo salió bien.")
```

```
out: Todo salió bien
```

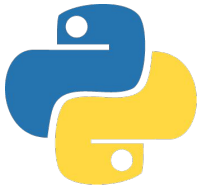
A los ya vistos bloques `try`, `except` y `else` podemos añadir un bloque más, el `finally`. Dicho bloque se ejecuta siempre, haya o no haya habido excepción.

```
try:
    int("10")
except Exception:
    print("Ocurrió un error.")
else:
    print("Todo salió bien.")
finally:
    print("Final del bloque")


out:
Todo salió bien
Final del bloque
```

Otra forma si no sabes qué excepción puede lanzarse, puedes usar la clase genérica `Exception`, que controla cualquier tipo de excepción. Todas las excepciones heredan de Exception.

```
try:
    int("Hola mundo")
except Exception:
    print("Ocurrió un error.")
```



Otras excepciones

La excepción `KeyError` es lanzada cuando se intenta acceder a una clave de un diccionario que no existe:

```
>>> d = {"a": 1}
>>> d["b"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'b'
```

La excepción `IndexError` es lanzada cuando se intenta acceder a un índice de una lista que no existe:

```
>>> numeros = [10,20,30]
>>> numeros[8]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

La excepción `ZeroDivisionError` es lanzada cuando se intenta dividir por cero:

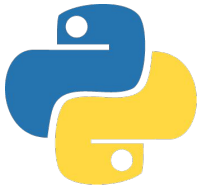
```
>>> 10 / 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

Podemos ver todas las excepciones que maneja Python en el siguiente enlace: [“Excepciones built-in”]

Por último, podemos crear nuevas excepciones del siguiente modo, para luego utilizarla vía `raise` y `except`.

```
class NuevaExcepcion(Exception):
     pass
```

Esta sintaxis se entenderá mejor cuando veamos el Paradigma de Objetos y como se aplica en este lenguaje.



¡Sigamos trabajando!