

## 2.3.-Captura de excepciones

### 2.3. Captura de excepciones

Las excepciones y los errores son dos conceptos fundamentales en programación, pero tienen significados distintos.

#### 1. Diferencia entre errores y excepciones

La principal diferencia entre errores y excepciones radica en su naturaleza y cómo afectan el flujo de ejecución del programa. Los errores indican problemas en el código que **deben ser corregidos**, mientras que las excepciones son **situaciones inesperadas** que pueden ser manejadas para asegurar que el programa siga funcionando sin interrupciones.

##### *1.1. Errores*

Los errores, también conocidos como bugs, son problemas en el código que impiden que el programa funcione correctamente. Pueden ser de diferentes tipos, como errores de sintaxis, errores lógicos (cuando el programa produce resultados incorrectos debido a un error en el algoritmo), o errores de tiempo de ejecución (que ocurren mientras el programa se está ejecutando). Los errores pueden ser causados por una variedad de razones, incluyendo lógica incorrecta, mal uso de funciones o librerías, o incluso problemas con la configuración del entorno de desarrollo.

##### *1.2. Excepciones:*

Las excepciones son eventos anómalos o inusuales que ocurren durante la ejecución de un programa y que afectan el flujo normal del mismo. A diferencia de los errores, las excepciones no siempre indican un fallo en el código. Por ejemplo, si un programa intenta abrir un archivo que no existe, esto generará una excepción `FileNotFoundException`. Sin embargo, esto no es un error en el código en sí, sino una situación inesperada que el programa debe saber cómo manejar. Las excepciones en Python están diseñadas para gestionar este tipo de situaciones y permitir que el programa continúe su ejecución en lugar de detenerse abruptamente.

### 2. Uso de try y except

Hemos visto varios casos de código en donde usábamos las funciones `input` e `int` para leer y analizar un número entero introducido por el usuario. También vimos lo poco seguro que podía llegar a resultar hacer algo así:

```
>>> velocidad = input(prompt)
¿Cual.... es la velocidad de vuelo de una golondrina sin carga?
¿Te refieres a una golondrina africana o a una europea?
>>> int(velocidad)
ValueError: invalid literal for int() with base 10:
>>>
```

Cuando estamos trabajando con el intérprete de Python, tras este error/excepción simplemente nos aparece de nuevo el prompt, así que pensamos “¡epa, me he equivocado！”, y continuamos con la siguiente sentencia.

Sin embargo, si se escribe ese código en un script de Python y se produce el error/excepción, el script se detendrá inmediatamente, y mostrará un “traceback”. No ejecutará la siguiente sentencia.

He aquí un programa de ejemplo para convertir una temperatura desde grados Fahrenheit a grados Celsius:

```
ent = input('Introduzca la Temperatura Fahrenheit:')
fahr = float(ent)
cel = (fahr - 32.0) * 5.0 / 9.0
print(cel)

# Código: https://es.py4e.com/code3/fahren.py
```

Si ejecutamos este código y le damos una entrada no válida, simplemente fallará con un mensaje de error bastante antipático:

```
python fahren.py
Introduzca la Temperatura Fahrenheit:72
22.2222222222
python fahren.py
Introduzca la Temperatura Fahrenheit:fred
Traceback (most recent call last):
  File "fahren.py", line 2, in <module>
    fahr = float(ent)
ValueError: invalid literal for float(): fred
```

Existen estructuras de ejecución condicional dentro de Python para manejar este tipo de errores/excepciones esperados e inesperados, llamadas `try / except`. La idea de `try` y `except` es que, si se sabe que cierta secuencia de instrucciones puede generar un problema, sea posible añadir ciertas sentencias para que sean ejecutadas en caso de error. Estas sentencias extras (el bloque `except`) serán ignoradas si no se produce ningún error.

Puedes pensar en la característica `try` y `except` de Python como una “póliza de seguros” en una secuencia de sentencias.

Se puede reescribir nuestro conversor de temperaturas de esta forma:

```
ent = input('Introduzca la Temperatura Fahrenheit:')
try:
    fahr = float(ent)
    cel = (fahr - 32.0) * 5.0 / 9.0
    print(cel)
except:
    print('Por favor, introduzca un número')

# Código: https://es.py4e.com/code3/fahren2.py
```

Python comienza ejecutando la secuencia de sentencias del bloque `try`. Si todo va bien, se saltará todo el bloque `except` y terminará. Si ocurre una excepción dentro del bloque `try`, Python saltará fuera de ese bloque y ejecutará la secuencia de sentencias del bloque `except`.

```
python fahren2.py  
Introduzca la Temperatura Fahrenheit:72  
22.222222222  
python fahren2.py  
Introduzca la Temperatura Fahrenheit:fred  
Por favor, introduzca un número
```

Gestionar una excepción con una sentencia `try` recibe el nombre de *capturar una excepción*. En este ejemplo, la cláusula `except` muestra un mensaje de error. En general, capturar una excepción te da la oportunidad de corregir el problema, volverlo a intentar o, al menos, terminar el programa con elegancia.

### 3. Capturar excepciones concretas

Es posible escribir programas que capturen y manejen determinadas excepciones. Durante el siguiente ejemplo, se le pide al usuario que ingrese un numero hasta que se haya ingresado un número entero válido, aunque el usuario podrá interrumpir el programa (puede variar las formas entre sistemas operativos); En linux/windows se utiliza Control-C y esta interrupción generará la excepción[KeyboardInterrupt](#)

```
>>>x = None  
...while x == None:  
...    try:  
...        x = int(input("Please enter a number: "))  
...    except ValueError:  
...        print("Oops! That was no valid number. Try again...")
```

Ten en cuenta que se a la cláusula `except` le puedes indicar que gestione varias excepciones, añadiendo el nombre de la excepción a continuación de la otra. Además, se pueden añadir varias cláusulas `except` cada diferenciar los bloques que gestionan la excepción en función de la excepción que se ha producido.

### 4. Lanzar excepciones

La declaración [raise](#) permite al programador forzar que ocurra una excepción específica.

Por ejemplo:

```
>>> raise NameError('HiThere')  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
NameError: HiThere
```

El único argumento para `raise` indica la excepción que se va a generar. Debe ser una instancia de excepción o una clase de excepción (una clase que se deriva de [Exception](#)).

Si se pasa una clase de excepción, se instanciará implícitamente llamando a su constructor sin argumentos:

```
raise ValueError # shorthand for 'raise ValueError()'
```

Si quieres saber si se generó una excepción, pero no tienes la intención de manejarla, la siguiente forma de usar la declaración `raise`, te permitirá volver a generarla:

```
>>> try:  
...     raise NameError('HiThere')  
... except NameError:  
...     print('An exception flew by!')  
...     raise  
...  
An exception flew by!  
Traceback (most recent call last):  
  File "<stdin>", line 2, in <module>  
NameError: HiThere
```

# \* Práctica 2.3: Capturas de excepciones

Salvo que se indique lo contrario, el objetivo de los ejercicios es evitar que una excepción llegue al programa principal y se aborte. Es decir, debes controlar cualquier evento que se pueda dar en tu programa. Ante cualquier duda, pregúntala.

## P2.3 - Ejercicios.

Controla las excepciones en los programas, para que los datos introducidos sean correctos.

### *Ejercicio 2.3.1*

Escribir un programa que pregunte al usuario su edad y muestre por pantalla todos los años que ha cumplido (desde 1 hasta su edad).

### *Ejercicio 2.3.2*

Escribir un programa que pida al usuario un número entero positivo y muestre por pantalla todos los números impares desde 1 hasta ese número separados por comas.

### *Ejercicio 2.3.3*

Escribir un programa que pida al usuario un número entero positivo y muestre por pantalla la cuenta atrás desde ese número hasta cero separados por comas. Deberá solicitar el número hasta introducir uno correcto.

### *Ejercicio 2.3.4*

Escribir un programa que pida al usuario un número entero, si la entrada no es correcta, mostrará el mensaje "La entrada no es correcta" y lanzará la excepción capturada.

### *Ejercicio 2.3.5*

Escribir que solicite una contraseña, y si no coincide con la que se tiene, lance la excepción `NameError` con el mensaje, "Incorrect Password!!"